

TikZ & PGF 手册 (3.0.1a) 笔记

2018 年 2 月 27 日

这个笔记是我在学习 TikZ & PGF 手册时整理的，只涉及我感兴趣并且我能读懂的内容。由于我不懂计算机，也没学过编程方面的知识，对 L^AT_EX 的熟悉程度也只是初学水平，所以对手册内容的理解非常业余，笔记中的很多名词都是随意使用的，里面一定有很多错误，请读者谅解。

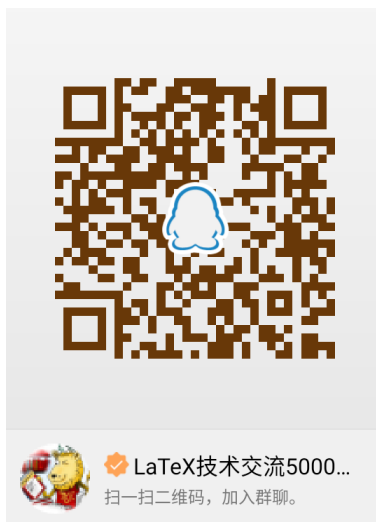
这个笔记适合像我一样的初学者阅读，公布这个笔记是为了让更多的人对 L^AT_EX 和 TikZ (PGF) 产生兴趣。笔记的前一部分内容涉及 TikZ 的绘图环境、命令以及相关的几个程序库；之后的内容是数据可视化；再之后是 key 操作，foreach 语句；再之后是数学引擎；最后一部分内容是关于 PGF 基本层的。

为了编辑出看着顺眼的数学材料，我不得不用 L^AT_EX。处理数学图形时，起初我先用其它画图软件画出图形，然后再插入到 T_EX 文件中，这样插入的图形有时不好看，比如，有时候图形中的文字与正文的文字不协调，有时候图形的尺寸不合适，但放缩图形时又会引起线宽、点的外观、文字形态的变化，不得不回到原来的画图软件中重新编辑图形。

为了能在 T_EX 文件中编辑出看着顺眼的图形，群主王老师推荐我使用 TikZ，于是我就开始学习手册。尽管手册有一千多页，但其中的讲解非常通俗、详细而且循序渐进，很适合初学者学习，像这个手册这样通俗的宏包说明文档真的很难得（手册的作者应该是大神中的大神吧）！

非常感谢王老师！

如果读者对 L^AT_EX 感兴趣，建议扫码加入下面的 QQ 群：



也可以搜索 QQ 群号 91940767 加群。这个群里有很多 L^AT_EX 高手，能够解答关于 L^AT_EX 的各种问题（注意，在向群友们提问前一定要先问一下“如何提问”）。群文件里有很多学习资料，群里还会举办活动来促

进大家学习 \LaTeX . 加入这个群你一定会有所收获, 举我本人的一个例子, 初学者常常会忽略简单却重要的事情, 我也不例外。我使用的编辑器是 TeXworks, 用了很长一段时间也没有发现这个编辑器有菜单选项“格式-智能引号-无”, 这个选项对实现某些 TikZ 命令很关键。例如, TikZ 的选项 `/tikz/`, `/pgf/arrow keys/width` 都用了单引号, 程序库 `quotes` 使用双引号, 输入单引号或双引号前要先选定编辑器的选项“格式-智能引号-无”, 否则输入的引号会导致编译过程中断并报告错误。这个错误困扰了我很久, 在群里高手的提示下才解决这个问题。再如, 很多时候 `foreach` 语句输出的数值是带小数点的, 下面的代码

```
\foreach \x [evaluate=\x as \xeval using int(\x)] in {2^0,2^...,2^8}{\xeval, }
```

将 `foreach` 语句输出的数值变成整数格式, 这个用法也来自群里高手的提示。

如果读者想获得关于 \LaTeX 的资讯、材料也可以扫码关注下面的微信号:



如果读者想浏览关于 \LaTeX 的中文网站, 我推荐 <http://www.latexstudio.net>, 这里面有大量关于 \LaTeX 的资料, 而且对这些资料做了分类整理。在关于 \LaTeX 的中文网站中, 这个网站是非常优秀的。

很多老师有多年教学经验, 希望把自己的教学经验、理念、学科知识总结成书, 我建议这些老师使用 \LaTeX 编辑自己的书。 \LaTeX 是世界上众多专家经过多年共同打造的优秀软件, 她的精美效果一定能更好地展示老师们多年的心血结晶。

目录

11 Design Principles	19
12 Hierarchical Structures:	
Package, Environments, Scopes, and Styles	19
12.1 载入宏包和程序库	19
12.2 绘制图形	19
12.2.1 用环境绘图	19
12.2.2 用命令绘图	20
12.3 {scope} 环境	20
12.3.1 {scope} 环境	20
12.3.2 {scope} 环境的简写形式	21
12.3.3 {scope} 命令	21
12.3.4 在路径之内插入 {scope} 环境	21
12.4 使用图形选项	21
12.4.1 如何处理图形选项	21
12.4.2 对图形使用 style 选项	22
13 设置坐标	22
13.1 Overview	22
13.2 坐标系	22
13.2.1 Canvas, XYZ, and Polar Coordinate Systems	22
13.2.2 质心坐标系	24
13.2.3 node 坐标系	25
13.2.4 tangent 坐标系	26
13.2.5 自定义坐标系	27
13.3 交点坐标	27
13.3.1 水平线与竖直线的交点——perpendicular 坐标系	27
13.3.2 任意路径的交点	27
13.4 相对坐标, 增量坐标	31
13.4.1 指定相对坐标	31
13.4.2 旋转的相对坐标——曲线上一点处的坐标系	31
13.4.3 相对坐标与当前点的局部化	32
13.5 坐标计算	33
13.5.1 一般句法	33
13.5.2 数乘坐标 (向量)	33
13.5.3 比例-角度定点句法	34
13.5.4 距离-角度定点句法	35
13.5.5 正射影-角度定点句法	36

14 设置路径的语句	36
14.1 The Move-To Operation	38
14.2 The Line-To Operation	39
14.2.1 线段	39
14.2.2 横线和竖线	40
14.3 The Curve-To Operation	40
14.4 矩形算子	40
14.5 Rounding Corners	41
14.6 圆、椭圆算子	41
14.7 The Arc Operation	42
14.8 The Grid Operation	42
14.9 The Parabola Operation	43
14.10 The Sine and Cosine Operation	44
14.11 The SVG Operation	45
14.12 The Plot Operation	45
14.13 The To Path Operation	45
14.14 The Foreach Operation	48
14.15 The Let Operation	48
14.16 The Scoping Operation	51
14.17 The Node and Edge Operations	51
14.18 The Graph Operation	51
14.19 The Pic Operation	51
14.20 The PGF-Extra Operation	51
15 Actions on Paths	52
15.1 Overview	52
15.2 Specifying a Color	52
15.3 Drawing a Path	53
15.3.1 Line Width, Line Cap, and Line Join	53
15.3.2 Dash Pattern	54
15.3.3 线条透明度	56
15.3.4 Double Lines and Bordered Lines	56
15.4 Adding Arrow Tips to a Path	56
15.5 Filling a Path	56
15.5.1 Fill Pattern	57
15.5.2 非零规则和奇偶规则	58
15.5.3 填充透明度	58
15.6 Using Arbitrary Pictures to Fill a Path	58
15.7 Shading a Path	60

15.8 Establishing a Bounding Box	61
15.9 Clipping and Fading	63
15.10 Doing Multiple Actions on a Path	64
15.11 Decorating and Morphing a Path	66
15.12 在起点或终点处截去一段路径	66
16 Arrows	66
16.1 Overview	66
16.2 如何添加箭头	66
16.3 设置箭头的外观	68
16.3.1 箭头的“特征尺寸”	68
16.3.2 箭头的比例缩放	70
16.3.3 圆弧箭头	71
16.3.4 倾斜	71
16.3.5 Reversing, Halving, Swapping	71
16.3.6 箭头颜色	72
16.3.7 线型	73
16.3.8 Bending and Flexing	74
16.4 Arrow Tip Specifications	78
16.4.1 句法	78
16.4.2 Specifying Paddings	79
16.4.3 Specifying the Line End	80
16.4.4 Defining Shorthands	80
16.4.5 Scoping of Arrow Keys	81
16.5 Reference: Arrow Tips	82
17 Nodes and Edges	83
17.1 Overview	83
17.2 Nodes and Their Shapes	84
17.2.1 Nodes 命令的句法	84
17.2.2 预定义的形状	86
17.2.3 一般选项	87
17.3 Multi-Part Nodes	90
17.4 node 中的文字	91
17.4.1 文字参数: 颜色、不透明度	91
17.4.2 文字参数: 字体	91
17.4.3 文字参数: 文字换行、对齐方式、文字行宽	91
17.4.4 文字参数: 文字的高度和深度	94
17.5 Positioning Nodes	94

17.5.1 node 的锚位置	94
17.5.2 基本的平移选项	95
17.5.3 高级平移选项	96
17.5.4 排布 node 的高级方法	100
17.6 Fitting Nodes to a Set of Coordinates	100
17.7 变换	100
17.8 在直线段或曲线上显式地摆放 node	101
17.9 在直线段或曲线上隐式地摆放 node	105
17.10 label 和 pin 选项	106
17.10.1 Overview	106
17.10.2 label 选项	106
17.10.3 The Pin Option	108
17.10.4 引用句法	109
17.11 Connecting Nodes: Using Nodes as Coordinates	111
17.12 Connecting Nodes: 用 edge 算子	111
17.12.1 edge 算子的基本句法	111
17.12.2 Nodes on Edges: Quotes Syntax	113
17.13 Referencing Nodes Outside the Current Picture	114
17.13.1 Referencing a Node in a Different Picture	114
17.13.2 引用 Current Page Node——绝对位置	115
17.14 Late Code and Late Options	116
18 Pics: Small Pictures on Paths	117
18.1 Overview	117
18.2 The Pic Syntax	117
18.2.1 指定 pic 图形的名称	118
18.2.2 指定 pic 图形的位置	118
18.2.3 定义 pic 图形	118
18.2.4 pic 的选项的传递	119
18.2.5 指定 pic 图形的遮挡次序	120
18.2.6 设置每个 pic 图形的样式	120
18.2.7 设置 pic 图形中的 node 名称的前缀并引用它	121
18.2.8 引用句法	122
18.3 定义 pic type	122
52 fit 程序库	124
70 topaths 程序库	126
70.1 直线	126
70.2 Move-To	126

70.3 曲线	126
70.4 Loops	130
20 矩阵及其对齐方式	131
20.1 Overview	131
20.2 Matrices are Nodes	131
20.3 元素图形	132
20.3.1 元素图形的对齐方式	132
20.3.2 调整行距和列距	132
20.3.3 设置元素图形样式的选项	134
20.4 矩阵的位置选项	135
20.5 自定义分列符	136
57 matrix 程序库	137
57.1 矩阵中的 node	137
57.2 换行符号与矩阵行的结束符号	139
57.3 定界符	139
22 函数绘图	141
22.1 Overview	141
22.2 plot 路径算子	141
22.3 连点成线	141
22.4 从外部文件中读取数据绘图	142
22.5 用函数表达式绘图	142
22.6 调用 gnuplot 绘制函数图形	144
22.7 给 plot 路径上的样本点加标记	144
22.8 直线、曲线、柱状图、条形图等	146
62 图柄程序库	150
62.1 曲线图柄	151
62.2 Constant 图柄	152
62.3 Comb 图柄	153
62.4 Bar 图柄	154
62.5 Gapped 图柄	156
62.6 Mark 图柄	157
63 用于 plot 绘图的点标记类型	159
23 透明度	160
23.1 Overview	160
23.2 为图形、路径、文字设定透明度	160

23.3 混色模式	161
23.4 颜色淡入、淡出——fading	162
23.4.1 创建 fading 图	162
23.4.2 创建 fading 路径	165
23.4.3 创建 fading 命令组	168
23.5 Transparency Groups	170
24 装饰路径	171
24.1 Overview	171
24.2 用路径算子 decorate 装饰子路径	174
24.3 装饰整个路径	175
24.4 调整装饰路径的外观	176
24.4.1 调整装饰路径与原被装饰路径的相对位置	176
24.4.2 调整装饰路径的始端与终端的形态	177
48 装饰程序库	178
48.1 公共选项	178
48.2 修饰路径的装饰样式	181
48.2.1 由直线段构成的装饰路径	181
48.2.2 由曲线构成的装饰路径	183
48.3 替换路径的装饰样式	186
48.4 标记装饰	190
48.5 自选标记装饰	190
48.5.1 程序库 decorations.markings	190
48.5.2 脚印标记	196
48.5.3 形状装饰	198
48.6 文字装饰	203
48.6.1 装饰样式 text along path	203
48.6.2 装饰样式 text effects along path	210
48.7 分形装饰	219
25 变换	221
25.1 各种坐标系统	221
25.2 xy 坐标系和 xyz 坐标系	223
25.3 坐标变换	224
25.3.1 变换选项	224
25.3.2 注意的问题	226
25.3.3 平面上的轴对称	230
25.4 画布变换	238

50 Externalization Library	238
50.1 Overview	238
50.2 Requirements	238
50.3 A Word About ConTEXt And Plain TEX	239
50.4 输出图形	239
50.4.1 Support for Labels and References In External Files	240
50.4.2 设置输出图形的名称	240
50.4.3 Remaking Figures or Skipping Figures	242
53 定点算术程序库	243
53.1 Overview	243
53.2 在 PGF 和 TikZ 中使用定点算术	243
54 浮点单元程序库	244
54.1 Overview	244
54.2 用法	245
54.3 与定点算术程序库的比较	246
54.4 命令与编程参考	246
54.4.1 浮点数的创建与转换	246
54.4.2 符号舍入操作	249
54.4.3 数学运算命令	250
54.4.4 用于编程的原始数学程序	252
55 Lindenmayer System 分形图程序库	252
55.1 Overview	255
55.1.1 声明一个 L-S	256
55.2 使用 L-S	259
55.2.1 在 PGF 中使用 L-S	259
55.2.2 在 TikZ 中使用 L-S	259
56 数学程序库	261
56.1 Overview	261
56.2 赋值语句	263
56.3 声明变量类型	264
56.4 循环语句	267
56.5 条件语句	268
56.6 声明函数	269
56.7 在命令 <code>\tikzmath</code> 的辖域内执行代码	269
65 shadings 程序库	271

66 shadows 程序库	275
66.1 Overview	275
66.2 一般的阴影选项	275
66.3 预定义的特殊阴影	276
66.3.1 Drop Shadows	276
66.3.2 Copy Shadows	277
66.4 针对圆形的阴影	278
74 数据可视化简介	280
74.1 数据点	280
74.2 可视化管线 (visualization Pipeline)	280
75 数据可视化的基本概念	280
75.1 Overview	280
75.2 数据点与数据格式	281
75.3 轴, 刻度线, 网格	282
75.4 显像器 (visualizer)	283
75.5 样式表和图例	284
75.6 用法	285
75.7 在数据可视化过程中执行用户自定义的代码	290
75.8 创建新对象	290
76 用于数据可视化的数据格式	290
76.1 Overview	290
76.2 简介	290
76.3 内置格式	290
76.4 函数格式	293
76.5 数据处理过程	295
76.6 定义新格式	295
77 坐标轴	295
77.1 Overview	295
77.2 轴的基本设置	295
77.2.1 用法	296
77.2.2 与轴对应的变量	296
77.2.3 变量值的范围	297
77.2.4 轴对数据的变换	298
77.2.5 对数轴	300
77.2.6 设置坐标轴的长度和单位长度	301
77.2.7 坐标轴的标签	303

77.2.8 仿射轴	304
77.3 轴系统	305
77.3.1 用法	305
77.3.2 科学坐标系	306
77.3.3 教科书坐标系	308
77.3.4 底层的笛卡尔坐标系	310
77.4 坐标轴的刻度和网格	311
77.4.1 概略	311
77.4.2 刻度和网格的主要选项	311
77.4.3 计算刻度线和网格线位置的半自动机制	312
77.4.4 计算刻度线和网格线位置的自动机制	313
77.4.5 手工确定刻度线和网格线的位置	315
77.4.6 刻度与网格线的样式：概略	318
77.4.7 刻度与网格线的样式：style 与 node Style	318
77.4.8 网格线的样式	319
77.4.9 刻度线与刻度值标签的样式	321
77.4.10 设置个别刻度的样式	322
77.4.11 其它刻度值标签选项	322
77.4.12 交错叠放刻度值标签	325
77.4.13 自动添加刻度的策略	328
77.4.14 定义新的添加刻度的策略	329
77.5 创建新的轴系统	329
77.5.1 创建一个轴系统	330
77.5.2 坐标轴的可视化	333
77.5.3 可视化网格线	339
77.5.4 刻度线、刻度值标签的可视化	341
77.5.5 坐标轴标签的可视化	344
77.5.6 完整的定义代码	347
77.5.7 Using the New Axis System Key	353
78 Visualizers	353
78.1 Overview	353
78.2 用法	353
78.2.1 使用一个显像器	353
78.2.2 使用多个显像器	354
78.2.3 设置显像器的外观效果	357
78.3 基本的显像器	360
78.3.1 直线段或曲线显像器	360
78.3.2 散点显像器	362

78.4 创建新的显像器	363
79 Style Sheets and Legends	363
79.1 Overview	363
79.2 Style Sheets 的例子	363
79.3 Legends 的例子	365
79.4 Style Sheet 的用法	366
79.4.1 引入一个 Style Sheet	366
79.4.2 创建新的样式表	366
79.4.3 创建新的颜色样式表	371
79.5 预定义的线型样式表	373
79.6 预定义的散点样式表	375
79.7 预定义的颜色样式表	376
79.8 显像器的标签	377
79.8.1 给一组数据点设置标签	377
79.8.2 给一组数据点设置大头针标签	381
79.9 使用图例	383
79.9.1 创建图例, 图例中的条目	384
79.9.2 图例中条目的行列排布	388
79.9.3 确定图例位置的一般方法	391
79.9.4 在绘图区域之外放置图例	393
79.9.5 在绘图区域之内放置图例	395
79.9.6 图例条目的一般样式	396
79.9.7 图例条目中的文字标签	397
79.9.8 条目中文字标签与图示标签的相对位置	398
79.9.9 手工添加条目	399
79.9.10 条目显像器	401
80 极坐标系	406
80.1 Overview	406
80.2 科学极坐标系统	408
80.2.1 角度轴的刻度线	409
80.2.2 角度轴的角度范围	410
80.3 创建新的极坐标系统	411
81 The Data Visualization Backend	412
82 Key Management	412
82.1 简介	412
82.1.1 与其它类似宏包的比较	412

82.1.2 快速引导	413
82.2 The Key Tree	414
82.3 Setting Keys	415
82.3.1 首字符句法检测	416
82.3.2 默认参数值	418
82.3.3 定义一个键，用键为命令的变量赋值	418
82.3.4 Keys That Store Values	420
82.3.5 定义手柄键	420
82.3.6 设置未知键的提示信息	422
82.3.7 用手柄设置键名称的搜索路径	423
82.4 键手柄	423
82.4.1 设置键路径的手柄	423
82.4.2 设置键的默认值的手柄	423
82.4.3 定义键所储存的代码	424
82.4.4 定义样式的手柄	425
82.4.5 Defining Value-, Macro-, If- and Choice-Keys	426
82.4.6 键值的展开，多重键值	427
82.4.7 键值和执行结果的转送	428
82.4.8 测试键的手柄	429
82.4.9 解释键的手柄	429
82.5 提示错误的键	429
82.6 键筛选	430
83 重复操作：foreach 句法	430
83.1 <commands> 的句法	430
83.2 <list> 中的省略号	431
83.3 在 <list> 中使用花括号包裹列举条目	432
83.4 在路径中使用 foreach 语句	432
83.5 多个相互关联的变量	432
83.6 针对变量的选项	435
86 扩展颜色支持	438
88 数学引擎概略	440
89 数学表达式	440
89.1 解析一个表达式	441
89.1.1 命令	441
89.2 长度单位的“显”、“隐”	444
89.3 数学表达式中的算子	445

89.4 数学表达式中的函数	447
89.4.1 基本算术函数	447
89.4.2 舍入函数	451
89.4.3 几个整数运算函数	451
89.4.4 三角函数	452
89.4.5 比较函数与逻辑函数	453
89.4.6 伪随机函数	454
89.4.7 基本的转换函数	455
89.4.8 其它函数	455
90 其它数学命令	457
90.1 基本算术函数	457
90.2 比较与逻辑函数	457
90.3 基本的进位制转换	458
90.4 角度计算	460
91 自定义数学引擎中的函数	460
92 输出数值的格式	463
92.1 基本的命令与选项	463
92.2 输出数值的样式以及标点符号	469
94 基本层 (basic layer) 概略	473
94.1 内核和模块	474
94.2 基本层的宏	474
94.3 以路径为核心的构图方式	475
94.4 坐标变换与画布变换	475
95 层级结构：宏包，环境，子环境，文字	475
95.1 Overview	475
95.1.1 宏包的层级结构	475
95.1.2 图形的层级结构	476
95.2 宏包的层次	476
95.2.1 内核宏包	476
95.2.2 模块	477
95.2.3 程序库宏包	477
95.3 图形的层级	478
95.3.1 主要的环境	478
95.3.2 绘图子环境	480
95.3.3 插入文字和图形	484

95.4 错误信息与警告	485
96 指定坐标	485
96.1 Overview	485
96.2 基本的坐标命令	486
96.3 XY-坐标系中的坐标	486
96.4 三维坐标	486
96.5 用已有坐标构建新的坐标	487
96.5.1 基本的坐标计算	487
96.5.2 直线或曲线上的点	488
96.5.3 矩形或椭圆边界上的点	490
96.5.4 两直线的交点	490
96.5.5 两个圆的交点	491
96.5.6 两个路径的交点	491
96.6 坐标分量	492
96.7 坐标点命令的工作方式	493
97 构建路径	494
97.1 Overview	494
97.2 Move-To 路径操作	495
97.3 Line-To 路径操作	496
97.4 Curve-To 路径操作	496
97.5 Close 路径操作	498
97.6 Arc, Ellipse, Circle 路径操作	498
97.7 Rectangle 路径操作	502
97.8 Grid 路径操作	503
97.9 Parabola 路径操作	504
97.10 Sine 和 Cosine 路径操作	505
97.11 Plot 路径操作	506
97.12 圆角 (Rounded Corners)	506
97.13 跟踪路径或图形的边界盒子	507
98 路径装饰	511
98.1 Overview	511
98.2 装饰自动化 (Decoration Automata)	511
98.2.1 约定路径名称	511
98.2.2 片段 (segment) 与状态 (state)	512
98.3 自定义装饰路径	513
98.4 {pgfdecoration} 环境	527
98.5 Meta-Decorations	532

98.5.1	定义一个 Meta-Decorations	532
98.5.2	预定义的 Meta-decorations	535
98.5.3	{pgfmetadecoration} 环境	535
99	使用路径	535
99.1	Overview	535
99.2	画出路径	536
99.2.1	图形参数: 线宽 Line Width	537
99.2.2	图形参数: 线冠 Caps 与交接 Joins	537
99.2.3	图形参数: 线型 Dashing	538
99.2.4	图形参数: 线条颜色	538
99.2.5	线条透明度	538
99.2.6	双线的内线	539
99.3	给路径加箭头	539
99.4	填充路径	540
99.4.1	图形参数: 判断内部点的规则	540
99.4.2	图形参数: 填充色	541
99.4.3	图形参数: 填充色的不透明度	541
99.5	剪切路径	541
99.6	将路径用作边界盒子	541
100	定义新的箭头	542
100.1	Overview	542
100.2	有关术语	542
100.3	PGF 处理箭头的一般过程	543
100.4	自定义箭头	544
100.5	关于箭头的选项	549
100.5.1	尺寸选项	549
100.5.2	True-False 选项	549
100.5.3	setup code 中不能引用的选项	550
100.5.4	自定义箭头选项	550
101	Nodes and Shapes	553
101.1	Overview	553
101.1.1	创建与索引 node	553
101.1.2	锚 Anchors	553
101.1.3	shape 的“层” Layers	554
101.1.4	Node Parts	554
101.2	创建 node	554
101.2.1	创建简单 node	554

101.2.2 创建 Multi-Part Nodes	556
101.2.3 另一种添加 node 的方法	560
101.3 使用锚位置 Anchors	562
101.3.1 在一个图形中引用锚位置	562
101.3.2 跨图引用 node 的锚位置	564
101.4 特殊 node	564
101.5 定义新的 shape	566
101.5.1 一个 shape 具备的要素	566
101.5.2 Normal Anchors 与 Saved Anchors	567
101.5.3 定义新 shape 的命令	567
102 矩阵	575
102.1 Overview	575
102.2 矩阵元素的对齐方式	575
102.3 矩阵命令	576
102.4 行间距与列间距	578
102.5 调用命令	579
103 坐标变换, 画布变换, 非线性变换	581
103.1 Overview	581
103.2 坐标变换	581
103.2.1 坐标变换矩阵	581
103.2.2 坐标变换命令	581
103.2.3 其它变换	586
103.2.4 保存或使用某个变换矩阵	587
103.2.5 坐标变换中的调整	587
103.3 画布变换	588
103.4 非线性变换	590
103.4.1 导引	590
103.4.2 定义并载入一个非线性变换	591
103.4.3 将非线性变换用于一个点	593
103.4.4 将非线性变换用于一个路径	593
103.4.5 将非线性变换用于文字	595
103.4.6 用线性变换近似非线性变换	596
103.4.7 非线性变换程序库	597
104 图样 Patterns	601
104.1 Overview	601
104.2 声明一个图样	602
104.3 使用图样	604

105 声明、使用外部图形	605
105.1 Overview	605
105.2 声明外部图形	605
105.3 使用外部图形	606
105.4 给图形“带面具”	607
107 创建 Plots	608
107.1 Overview	608
107.2 创建图流	608
107.2.1 图流的基本结构	609
107.2.2 生成图流的命令	611
107.3 图柄	612
107.4 定义新图柄	614
108 图层	616
108.1 Overview	616
108.2 声明图层	616
108.3 在图层上绘图	617
109 颜色渐变	618
109.1 Overview	618
109.2 声明渐变样式	618
109.2.1 横向渐变与纵向渐变	618
109.2.2 辐射渐变	619
109.2.3 函数渐变	620
109.3 使用颜色渐变	624
109.4 关于 type 4 函数的补充	626
110 透明度	631
110.1 指定不透明度	631
110.2 指定混色模式	632
110.3 “褪色”(fading)效果	632
110.4 透明度组	635
111 临时寄存器	636
112 快速命令	638
112.1 快速坐标命令	638
112.2 快速创建路径的命令	638
112.3 快速使用路径的命令	639
112.4 快速文字盒子命令	639

11 Design Principles

在导言区调用 `\usepackage{tikz}`。TikZ 的语法和命令受多个内容的影响。基本命令和路径算子的概念来自 `metafont`，选项机制来自 `pstricks`，样式的概念容易联想到 `svg`，`graphviz` 语句借用自 `graphviz`。

TikZ 的设计主要有以下几个方面

1. 指定坐标的句法。
2. 指定路径的句法。
3. 对路径的操作。
4. 图形参数的 Key-value 句法。
5. 指定 node 的句法。
6. 指定 tree 的句法。
7. 指定 graph 的句法。
8. 图形参数的分组。
9. 坐标变换系统。

12 Hierarchical Structures: Package, Environments, Scopes, and Styles

12.1 载入宏包和程序库

```
\usepackage{tikz}
```

这个宏包没有选项。

```
\usetikzlibrary{<list of libraries>}
```

用在导言区，调用 TikZ 的程序库，例如 `\usetikzlibrary{arrows.meta, intersections}`，其中的花括号可以换成方括号。

12.2 绘制图形

12.2.1 用环境绘图

tikz 的最外层的绘图区域是 `{tikzpicture}` 环境，所有绘图命令（除了命令 `\tikzset`）都要放在环境内。环境内的选项仅在环境内有效。该环境可以用于大多数 `LATEX` 模式、环境、命令内，例如用于页眉、页脚命令内，数学模式内。

```
\begin{tikzpicture}[<options>]
```

```
  <environment contents>
```

```
\end{tikzpicture}
```

环境中的绘图命令会被逐个放入一个盒子中。在处理命令时，每当 PGF 遇到一个坐标，它就会刷新盒子尺寸，从而估计整个图形的尺寸。这个估计并非总是精确。

环境中的非图形文字会被尽可能地抑制，非 PGF 命令不会有任何结果。

`/tikz/baseline=<dimension or coordinate or default>` 默认 0pt

环境选项，指定图形中的水平直线为图形基线，该直线的纵截距是 <dimension>，或者通过坐标点 <coordinate>，默认为直线 $y=0pt$ 。

`/tikz/execute at begin picture=<code>` 无默认

环境参数，设置在环境开始时要执行的代码。如果该选项被设置多次，则其代码会累积叠加。

`/tikz/execute at end picture=<code>` 无默认

环境参数，设置在环境结束时要执行的代码。如果该选项被设置多次，则其代码会累积叠加。



```
\begin{tikzpicture}[execute at end picture=%
{
  \begin{pgfonlayer}{background}
  \path[fill=yellow,rounded corners]
  (current bounding box.south west) rectangle
  (current bounding box.north east);
  \end{pgfonlayer}
}]
\node at (0,0) {X};
\node at (2,1) {Y};
\end{tikzpicture}
```

`/tikz/every picture` (style, 初值 empty)

为每个环境设置统一的样式。

```
\tikzset{every picture/.style=semithick}
```

12.2.2 用命令绘图

`\tikz[<options>]{<path commands>}`

该命令将 <path commands> 放入 {tikzpicture} 环境中。<path commands> 中可以有多个段落，脆弱命令（如抄录文本）。如果其中只有一个绘图命令，无需加花括号，如果有数个绘图命令，需要用花括号将这些命令括起来（与 \foreach 语句类似）。

12.3 {scope} 环境

12.3.1 {scope} 环境

{scope} 环境只能用在 {tikzpicture} 环境中，绘制子图形。

`\begin{scope}[<options>]`

```
<environment contents>
\end{scope}
```

所有 <options>，特别地，clip 路径都只在该环境内有效。该环境可以有以下参数选项。

```
/tikz/every scope (style, 初值 empty)
/tikz/execute at begin scope=<code> 无默认值
/tikz/execute at end scope=<code> 无默认值
```

这些选项的作用与前面 {tikzpicture} 环境对应的选项类似。

12.3.2 {scope} 环境的简写形式

调用 scope 程序库

```
\usetikzlibrary{scopes}
```

这个程序库定义了 {scope} 环境的简写形式。调用这个程序库后，{scope} 环境就可以写成下面的形式

```
{ [<options>]
  <environment contents>
}
```

注意，开花括号后必须跟随方括号（可以空置，但不能省略），如此开启 {scope} 环境，之后对应的闭花括号结束 {scope} 环境。

12.3.3 {scope} 命令

```
\scoped[<options>]<path command>
```

与 \tikz 类似。

12.3.4 在路径之内插入 {scope} 环境



```
\tikz \draw (0,0) -- (1,1)
  {[rounded corners] -- (2,0) -- (3,1)}
  -- (3,0) -- (2,1);
```

可见这样插入的 {scope} 环境是主路径的一部分，其它细节参考 §14.

12.4 使用图形选项

12.4.1 如何处理图形选项

tikz 的环境和命令都接受选项，选项通常称作 key .

```
\tikzset{<options>}
```

这个命令调用 \pgfkeys 命令处理 <options>. <options> 是由 <key>=<value> 组成的列表，之间用逗号分隔。处理一个 <key>=<value> 时，按以下步骤：

1. 如果 `<key>` 是完整的 key，直接按 §82 的描述处理。
2. 否则，检查 `/tikz/<key>` 是否是个 key，如果是，执行之。
3. 否则，检查 `/pgf/<key>` 是否是个 key，如果是，执行之。
4. 否则，检查 `<key>` 是否是个颜色，如果是，执行之。
5. 否则，检查 `<key>` 是否包含一个短线 (dash)，如果是，执行 `arrows=<key>`。
6. 否则，检查 `<key>` 是否是个形状名称，如果是，执行 `shape=<key>`。
7. 其它情况，输出错误信息。

12.4.2 对图形使用 style 选项

13 设置坐标

13.1 Overview

一个坐标 (coordinate) 就是画布 (canvas) 上的一个位置。指定坐标的一般句法是
(`<options><coordinate specification>`)

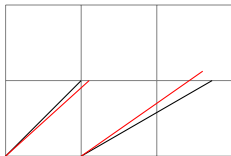
可以用笛卡尔坐标系，极坐标系，球坐标系来指定坐标。有两种方式指定所使用的坐标系：

Explicitly 显示地指定，直接给出坐标系名称，名称后紧跟 `cs:`，这代表 “coordinate system”，再后跟坐标数据：

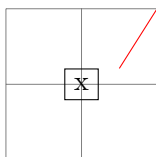
(`<coordinate system> cs:<list of key-value pairs specific to the coordinate system>`)

Implicitly 隐式地，有几个特殊的句法对应不同的坐标系。例如，笛卡尔坐标，`(0,0)`；极坐标，`(30:2)`，`++(30:1 and 2)`。

可以在单个坐标前加选项 `<options>`，但仅限于坐标变换选项，用来变换该坐标。



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \draw (0,0) -- (1,1);
  \draw[red] (0,0) -- ([xshift=3pt] 1,1);
  \draw (1,0) -- +(30:2cm);
  \draw[red] (1,0) -- +([shift=(135:5pt)] 30:2cm);
\end{tikzpicture}
```



```
\begin{tikzpicture}
  \draw [help lines] (-1,-1) grid (1,1);
  \node [draw] (x) {x};
  \draw [red] (1,1) -- ([xshift=5mm] x.north);
\end{tikzpicture}
```

13.2 坐标系

13.2.1 Canvas, XYZ, and Polar Coordinate Systems

canvas

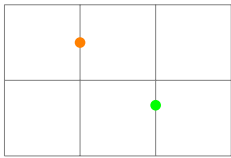
这就是一种笛卡尔坐标系，用 `x=`、`y=` 选项设置坐标数据，数据是带长度单位的，也可以是（展开为长度的）宏。

`/tikz/cs/x=<dimension>` （无默认值，初始值 0pt）

<dimension> 是 x 坐标数据，带有长度单位。

`/tikz/cs/y=<dimension>` （无默认值，初始值 0pt）

类似以上。



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \fill [orange] (canvas cs:x=1cm,y=1.5cm) circle (2pt);
  \fill [green] (canvas cs:x=2cm,y=0.03*\textheight) circle (2pt);
\end{tikzpicture}
```

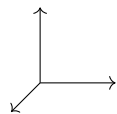
xyz

这就是 3 维笛卡尔坐标系，用 `x=`、`y=`、`z=` 选项设置坐标数据，数据不带长度单位。x 轴、y 轴的单位向量的默认长度为 1cm（可以用 `x=<dimension>`、`y=<dimension>` 修改，<dimension> 带有长度单位，参考坐标变换）。z 轴的单位向量默认为 (-3:85mm,-3:85mm)。

`/tikz/cs/x=<factor>` （无默认值，初始值 0pt）

`/tikz/cs/y=<factor>` （无默认值，初始值 0pt）

`/tikz/cs/z=<factor>` （无默认值，初始值 0pt）



```
\begin{tikzpicture}[->]
  \draw (0,0) -- (xyz cs:x=1);
  \draw (0,0) -- (xyz cs:y=1);
  \draw (0,0) -- (xyz cs:z=1);
\end{tikzpicture}
```

如果给出的隐式坐标 (`<x>`、`<y>`)，其数据都带长度单位，则用 canvas 坐标系；如果都不带长度单位，则用 xyz 坐标系；如果有的数据带长度单位、有的数据不带长度单位，处理规则是：

例如，(2pt,3) 等于 (2pt,0pt)+(0,3)；再如，(3+2pt,4) 等于 (3pt+2pt,0pt)+(0,4)；再如，(3+2pt,4+5pt) 等于 (3pt+2pt,0pt)+(0pt,4pt+5pt) 等于 (5pt,9pt)。

canvas polar

用于指定极坐标点。

`/tikz/cs/angle=<degrees>` （无默认值）

<degrees> 是角度制下的数据，限制在 -360 到 720 之间。

`/tikz/cs/radius=<dimension>` （无默认值）

`/tikz/cs/x radius=<dimension>` (无默认值)

`/tikz/cs/y radius=<dimension>` (无默认值)

当使用 `x=dx`, `y=dy` 选项时, 指定的点在椭圆上, 椭圆横半轴长度 `dx`, 纵半轴长度 `dy`.

隐式的极坐标语句是, 例如, `(30:1pt)`, `(30:1mm and 2pt)`, 角度默认角度制, 长度数据带单位。

`xy polar`

`xyz polar`

这种极坐标系与 `canvas polar` 坐标系类似, 但是解释为 `xy` 或 `xyz` 坐标系。

`/tikz/cs/angle=<degrees>` (无默认值)

`/tikz/cs/radius=<dimension>` (无默认值)

`/tikz/cs/x radius=<dimension>` (无默认值)

`/tikz/cs/y radius=<dimension>` (无默认值)

与前述类似。

隐式的坐标语句是, 例如, `(30:1)`, `(30:1 and 2)`, 角度默认角度制, 长度数据的默认单位是 `cm`, 注意没有 `(30:1mm and 2)` 这种带单位与不带单位混合的情况。

13.2.2 质心坐标系

`barycentric`

这是质心坐标系。给定 n 个向量 v_1, \dots, v_n , n 个数值 a_1, \dots, a_n , 它们决定一个坐标

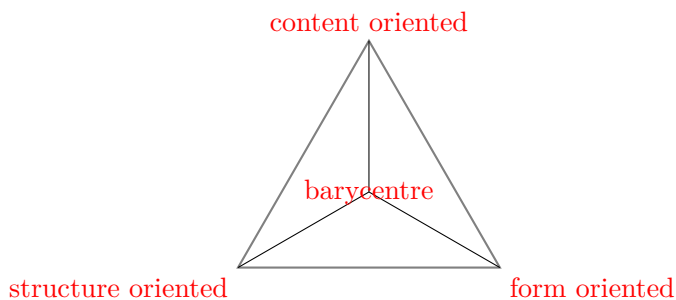
$$\frac{a_1 v_1 + \dots + a_n v_n}{a_1 + \dots + a_n}$$

如果把数值 a_i 解释为对应质点 v_i 的质量, 上式就是这个质点组的质心。

首先用 `coordinate` 语句创建数个 `coordinate` 对象并为之命名, 例如创建 3 个 `coordinate` 对象, 名称分别是 `<name1>`, `<name2>`, `<name3>`, 假设在这 3 个 `coordinate` 位置有质量分别是 `<w1>`, `<w2>`, `<w3>` 的质点, 下面语句计算这 3 个质点的质心坐标

`(barycentric cs: <name1>=<w1>,<name2>=<w2>,<name3>=<w3>)`

目前该坐标系只能用于由 `coordinate` 语句创建的对象名称, 不支持其它位置或名称。



```
\begin{tikzpicture}[every node/.style={red}]
  \coordinate (content) at (90:2cm);
```



```

\coordinate (structure) at (210:2cm);
\coordinate (form) at (-30:2cm);
\node [above] at (content) {content oriented};
\node [below left] at (structure) {structure oriented};
\node [below right] at (form) {form oriented};
\draw [thick,gray] (content.south) -- (structure.north east)
  -- (form.north west) -- (form.north west) -- cycle;
\path
  (barycentric cs:content=1,structure=1 ,form=1)
  edge (content.south)
  edge (structure.north east)
  edge (form.north west) ;
\node at (barycentric cs:content=1,structure=1 ,form=1) {barycentre};
\end{tikzpicture}

```

13.2.3 node 坐标系

node

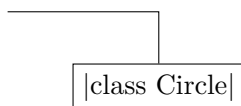
定义一个 node 对象后，就会有与该 node 相关的各种坐标位置。

`/tikz/cs/name=<node name>` (无默认值)

<node name> 是 node 的名称，引用与该 node 相关的坐标位置时要写出 node 的名称。

`/tikz/anchor=<anchor>` (无默认值)

指定 node 的锚位置，在 node 的边界上。



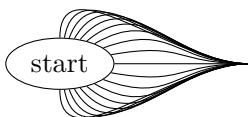
```

\begin{tikzpicture}
  \node (circle) at (2,0) [draw] {class Circle};
  \draw (node cs:name=circle,anchor=north) |- (0,1);
\end{tikzpicture}

```

`/tikz/cs/angle=<degrees>` (无默认值)

指定 node 的角度位置，在 node 的边界上。



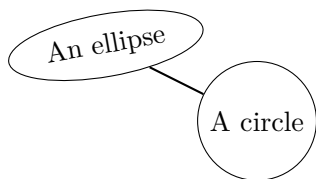
```

\begin{tikzpicture}
  \node (start) [draw,shape=ellipse] {start};
  \foreach \angle in {-90, -80, ..., 90}
  \draw (node cs:name=start,angle=\angle)
    .. controls +(\angle:1cm) and +(-1,0) .. (2.5,0);
\end{tikzpicture}

```

在两个 node 之间画线时，如果不指定 `anchor=`, `angle=` 这些位置，tikz 会自动计算 node 的边界上的

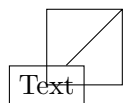
位置并用线连接起来。



```
\begin{tikzpicture}
  \path (0,0) node(a) [ellipse,rotate=10,draw]
        {An ellipse}
        (2,-1) node(b) [circle,draw] {A circle};
  \draw[thick] (node cs:name=a) -- (node cs:name=b);
\end{tikzpicture}
```

如果 tikz 不能确定适当的 node 的边界点，就会使用 node 的中心位置。画连线时，对于“line-to”算子“-”，纵横线算子“|-”和“-|”，“curve-to”算子“..”，会自动计算所需的锚位置 anchor；对于其它算子，例如，parabola 或 plot，使用 node 的中心位置。

如果在 node 与（相对于该 node 的）相对坐标之间画线，并让 tikz 自动计算连线位置，那么相对坐标就相对于 node 的中心位置来计算。



```
\tikz \draw (0,0) node(x) [draw] {Text} rectangle (1,1)
        (node cs:name=x) -- +(1,1);
```

隐式地指定一个 node 坐标的方法很简单，例如，用 node 名称 (a)，用 node 的锚位置 (a.north)，用 node 的角度位置 (a.30)。

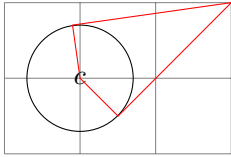
13.2.4 tangent 坐标系

tangent

调用 calc 程序库后才能使用这个坐标系。首先创建一个形状为 circle 且名称为 <name> 的 node，一个点 <point>，过 <point> 可以作两条 <name> 的切线，可以使用这个坐标系找出切点坐标，需要设置以下选项。

- /tikz/cs/node=<node name> (无默认值)
- /tikz/cs/point=<point> (无默认值)
- /tikz/cs/solution=<number> (无默认值)

选项 node= 指定 node 的名称。选项 point= 指定切线经过的点。选项 solution= 用于指定使用哪个切点。



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \coordinate (a) at (3,2);
  \node [circle,draw] (c) at (1,1) [minimum size=40pt] {$c$};
  \draw[red] (a) -- (tangent cs:node=c,point={a},solution=1)
    -- (c.center) -- (tangent cs:node=c,point={a},solution=2)
    -- cycle;
\end{tikzpicture}
```

13.2.5 自定义坐标系

13.3 交点坐标

13.3.1 水平线与竖直线的交点——perpendicular 坐标系

perpendicular

用这个坐标系确定水平线与竖直线的交点，需要设置以下选项。

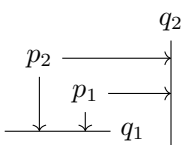
`/tikz/cs/horizontal line through={(<coordinate>)}` (无默认值)

设置过 (<coordinate>) 的水平线。

`/tikz/cs/vertical line through={(<coordinate>)}` (无默认值)

设置过 (<coordinate>) 的竖直线。有了水平线与竖直线就得到它们的交点。

隐式地确定水平线与竖直线的交点，(<p> |- <q>), 或者 (<q> -| <p>), 例如, (2,1 |- 3,4) 与 (3,4 -| 2,1) 表示同一个点，注意其中圆括号的使用 (只在外层使用一对圆括号)。



```
\begin{tikzpicture}
  \path (30:1cm) node(p1) {$p_1$} (75:1cm) node(p2) {$p_2$};
  \draw (-0.2,0) -- (1.2,0) node(xline)[right] {$q_1$};
  \draw (2,-0.2) -- (2,1.2) node(yline)[above] {$q_2$};
  \draw[->] (p1) -- (p1 |- xline);
  \draw[->] (p2) -- (p2 |- xline);
  \draw[->] (p1) -- (p1 -| yline);
  \draw[->] (p2) -- (p2 -| yline);
\end{tikzpicture}
```

这种结构可以连续 (套嵌) 使用，例如

`(0,0 -| 1,1 |- 0,2)`

这个连续使用 -| 符号的形式实际是 -| 算子的套嵌，它所表达的坐标是 (1,2)。

13.3.2 任意路径的交点

首先调用 `intersections` 程序库。由于 \TeX 的精度不高，只能计算“适当复杂度”的路径的交点。例如，不能计算包含很多小线段的 `plots` 或 `decorated` 路径的交点。

要计算两个路径的交点，首先（必须）用下面的选项给路径命名：

`/tikz/name path=<name>` (无默认值)

`/tikz/name path global=<name>` (无默认值)

选项 `name path=` 的命名只在当前环境内有效。选项 `name path global=` 的命名全局有效。这两个选项可以用作路径的选项，也可以作为 `node` 的选项。

给一个路径用 `name path=<name>` 命名后，命名范围不包含添加到路径上的 `node` 对象的边界路径。所以计算路径交点时，添加到路径上的 `node` 对象会被忽略。《手册》中提到，如果用 `name path=<name>` 给路径和路径上的 `node` 对象作相同的命名，那么这个名称会把路径和 `node` 的边界路径作为一个整体来计算路径交点。但经测试，这一点未能实现。不过下面两种交点都能实现：

```
\begin{tikzpicture}
  \draw [name path=a] (0,0) -- (2,3) node [name path=b, ...]{};
  \fill [name intersections={of=a and b,...}] ...;
\end{tikzpicture}
\begin{tikzpicture}
  \draw (0,0) -- (2,3) node [name path=a, ...]{};
  \draw [name path=b] (2,3) -- (3,0);
  \fill [name intersections={of=a and b,...}] ...;
\end{tikzpicture}
```

用下面的 key 设置与“计算交点”相关的内容：

`/tikz/name intersections={<options>}` (无默认值)

其中花括号里的 `<options>` 可以使用以下 key:

`/tikz/intersection/of=<name path 1> and <name path 2>` (无默认值)

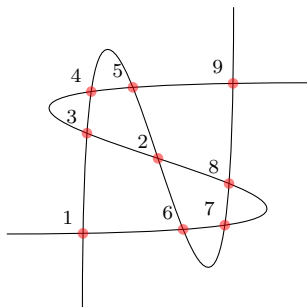
设置两条路径，求二者的交点。

`/tikz/intersection/name=<prefix>` (无默认值，初始值 `intersection`)

此 key 给交点命名，并指定名称的前缀，前缀初始值是 `intersection`，在初始值下交点名称依次是 `intersection-1`，`intersection-2` 等等，名称形式是 `<prefix>-<number>`。当用这种名称引用交点来构建路径时，要给名称加圆括号，如 `(intersection-1)`。

`/tikz/intersection/total=<macro>` (无默认值)

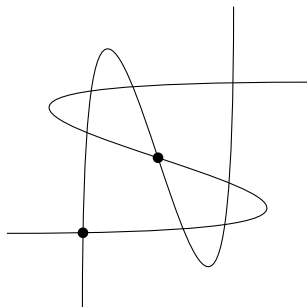
将交点的总个数赋予宏 `<macro>`。



```
\begin{tikzpicture}
  \clip (-2,-2) rectangle (2,2);
  \draw [name path=curve 1] (-2,-1) .. controls (8,-1) and (-8,1) .. (2,1);
  \draw [name path=curve 2] (-1,-2) .. controls (-1,8) and (1,-8) .. (1,2);
  \fill [name intersections={of=curve 1 and curve 2, name=i, total=\t}]
    [red, opacity=0.5, every node/.style={above left, black, opacity=1}]
    \foreach \s in {1,...,\t}{(i-\s) circle (2pt) node {\footnotesize\s}};
\end{tikzpicture}
```

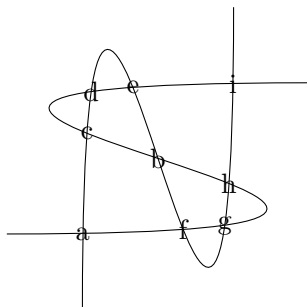
`/tikz/intersection/by=<comma-separated list>` (无默认值)

交点的名称如前可以用 `name=` 定义为 `<prefix>-<number>` 的形式，也可以用 `by=` 来定义。



```
\begin{tikzpicture}
  \clip (-2,-2) rectangle (2,2);
  \draw [name path=curve 1] (-2,-1) .. controls (8,-1) and (-8,1) .. (2,1);
  \draw [name path=curve 2] (-1,-2) .. controls (-1,8) and (1,-8) .. (1,2);
  \fill [name intersections={of=curve 1 and curve 2, by={a,b}}]
    (a) circle (2pt)
    (b) circle (2pt);
\end{tikzpicture}
```

如果 `by=<comma-separated list>` 中的列表项有方括号选项，这些选项会被执行；如果列表项中间可以有省略号“...”，列表会被作为 `\foreach` 陈述来处理。



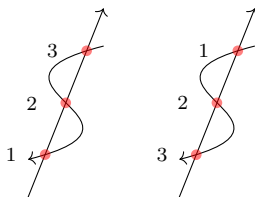
```

\begin{tikzpicture}
  \clip (-2,-2) rectangle (2,2);
  \draw [name path=curve 1] (-2,-1) .. controls (8,-1) and (-8,1) .. (2,1);
  \draw [name path=curve 2] (-1,-2) .. controls (-1,8) and (1,-8) .. (1,2);
  \fill [name intersections={
    of=curve 1 and curve 2,
    by={[label=center:a],[label=center:...],[label=center:i]}}];
\end{tikzpicture}

```

`/tikz/intersection/sort by=<path name>` (无默认值)

在默认下,交点按某个算法规则顺次命名,但这个算法规则并不直观。这个选项按路径 <path name> 的走向将交点顺次命名。



```

\begin{tikzpicture}
  \clip (-0.5,-0.75) rectangle (3.25,2.25);
  \foreach \pathname/\shift in {line/0cm, curve/2cm}{
    \tikzset{xshift=\shift}
    \draw [->, name path=curve] (1,1.5) .. controls (-1,1) and (2,0.5) .. (0,0);
    \draw [->, name path=line] (0,-.5) -- (1,2) ;
    \fill [name intersections={of=line and curve,sort by=\pathname, name=i}]
      [red, opacity=0.5, every node/.style={left=.25cm, black, opacity=1}]
    \foreach \s in {1,2,3}{(i-\s) circle (2pt) node {\footnotesize\s}};
  }
\end{tikzpicture}

```

13.4 相对坐标, 增量坐标

13.4.1 指定相对坐标

在坐标前面加前缀“++”，例如 ++(1,0)，表示将当前坐标平移，平移向量是 (1,0)，并将平移后的坐标设定为当前坐标。



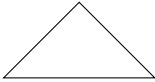
```
\tikz \draw (0,0) -- ++(1,0) -- ++(0,1) -- ++(-1,0) -- cycle;
```

上例中，当前坐标从 (0,0) 逐步变到 (0,1)，图形等价于



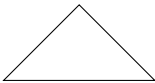
```
\tikz \draw (0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle;
```

在坐标前面加前缀“+”，例如 +(1,0)，表示将当前坐标平移，平移向量是 (1,0)，但不改变当前坐标。



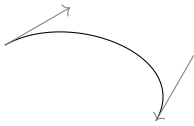
```
\tikz \draw (0,0) -- +(1,0) -- +(0,1) -- +(-1,0) -- cycle;
```

上例中，当前坐标始终是 (0,0)，图形等价于



```
\tikz \draw (0,0) -- (1,0) -- (0,1) -- (-1,0) -- cycle;
```

如果将相对坐标用于控制点，则遵照以下规定：第一个控制点相对于起点；第二个控制点相对于终点；如果终点是相对坐标，则它相对于起点。这样的规定容易确定控制曲线在起点和终点处的切线方向。



```
\begin{tikzpicture}
  \draw (1,0) .. controls +(30:1cm) and +(60:1cm) .. (3,-1);
  \draw[gray,->] (1,0) -- +(30:1cm);
  \draw[gray,<-] (3,-1) -- +(60:1cm);
\end{tikzpicture}
```

13.4.2 旋转的相对坐标——曲线上一点处的坐标系

规定在曲线上某点处的 turn 坐标系：x 轴的单位向量是该点处的单位切向量，y 轴的单位向量是该点处与单位切向量成右手直角系的单位法向量。这种“一点处的”坐标系由选项 turn 设置。这个坐标系类似曲线的自然坐标系，或 Frenet 标架，只是 y 轴单位向量的取法有所不同。

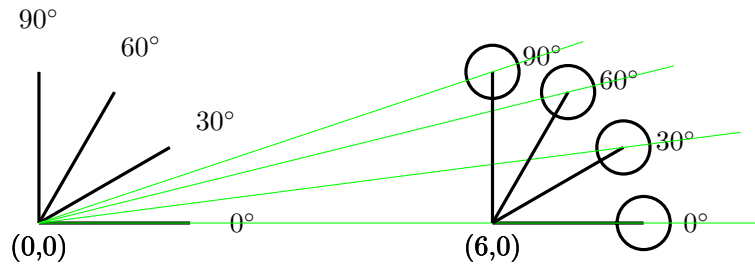
/tikz/turn

当坐标 <coordinate> 带有 turn 选项时，表示该坐标按当前位置的 turn 坐标系计算。



```
\tikz \draw (0,0) arc [start angle=70,delta angle=-50, radius=1.5cm]
  -- ([turn]30:1cm);
```

turn 坐标系最好不要用在 circle 路径后，例如



```
\begin{tikzpicture}
\foreach \ang in {0,30,...,90}
\draw [very thick] (0,0) node [below] {(0,0)}-- (\ang:2)
  ([turn] 0.7,0) node {$\ang^\circ$};
\foreach \ang in {0,30,...,90}
{\draw [very thick] (6,0) node [below] {(6,0)} -- +(\ang:2)
  coordinate (d\ang) circle (10pt)
  ([turn] 0.7,0) node {$\ang^\circ$};}
\draw [green] (0,0) -- ($(0,0)!1.2!(d\ang)$);}
\end{tikzpicture}
```

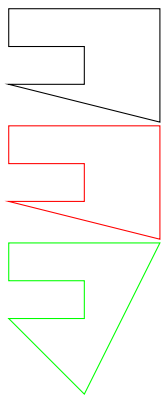
13.4.3 相对坐标与当前点的局部化

通常，当前点不被局部化，例如，在一个路径上可以添加一个 `{scope}` 环境，创建一个 `scope` 路径，这个 `scope` 路径的起始点参照 `{scope}` 环境之前的当前点确定，`scope` 路径内的当前点也是整个（外层）路径的当前点，后续的相对坐标都参照这个当前点。

下面的选项可以使得当前点局部化：

`/tikz/current point is local=<boolean>` （无默认值，初始值 `false`）

该选项的效果是，当局部路径结束后，当前点会返回该局部路径之前的当前点。如下例

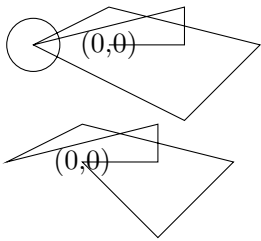


```
\tikz \draw (0,0) -- ++(1,0) -- ++(0,0.5)
  -- ++(-1,0) -- ++(0,0.5) -- ++(2,0)
  -- ++(0,-1.5) -- cycle;

\tikz \draw[red] (3,0) -- ++(1,0) -- ++(0,0.5)
  {-- ++(-1,0) -- ++(0,0.5) -- ++(2,0) }
  -- ++(0,-1.5) -- cycle;

\tikz \draw[green] (6,0) -- ++(1,0) -- ++(0,0.5)
  {[current point is local]-- ++(-1,0) -- ++(0,0.5) -- ++(2,0) }
  -- ++(0,-1.5) -- cycle;
```

当局部路径含有 `circle` 路径时，情况会变得复杂，例如



```
\tikz \draw (0,0) node{(0,0)} -- ++(1,0) -- ++(0,0.5)
  {[current point is local]--(-1,0) circle (10pt)--(0,0.5)--(2,0)}
  -- ++(0,-1.5) -- cycle;

\tikz \draw (0,0) node{(0,0)} -- ++(1,0) -- ++(0,0.5)
  {[current point is local]-- (-1,0) -- (0,0.5) -- (2,0) }
  -- ++(0,-1.5) -- cycle;
```

13.5 坐标计算

首先调用 calc 程序库。

13.5.1 一般句法

([<options>]\$(<coordinate computation>\$).

例如 $(\$ (1,2) - (2,3) \$)$ 计算两个向量的差。注意句式中的两个 \$ 表示 tikz 的坐标计算，不代表 \TeX 的数学模式。

13.5.2 数乘坐标（向量）

<factor>*<coordinate>

注意其中表示乘法的 “*” 符号不能省略，它两侧也不能有空格。

系统会用 `\pgfmathparse` 来解析 <factor>，这意味着 <factor> 可以是比较复杂的算式，例如，

$(\${\atan(\sqrt{5}*(2/3))/100 * \exp(1/5 * \ln(7))}*(2,3)\$)$

其中 $\{\atan(\sqrt{5} * (2/3)) / 100 * \exp(1/5 * \ln(7))\}$ 是 <factor>，整个坐标就是

$$\frac{\arctan(\sqrt{5} \cdot \frac{2}{3})}{100} \cdot \sqrt[5]{7} \cdot \begin{pmatrix} 2 \\ 3 \end{pmatrix}.$$

系统会认为第一个符号组合 “*(” 之前的部分都是 <factor>，所以如果写出 $(\${\sqrt{5}}*(1/3)*(2,3)\$)$ 系统会给出错误信息：

! Package pgf Error: No shape named 1/3 is known.

可以使用花括号明确地给 <factor> 定界，例如 $(\${\sqrt{5}}*(1/3)}*(2,3)\$)$ 就可以接受。在 <factor> 算式内部的 “*” 两侧可以有空格。

语句 $(\$ (1,2) - (2,3) \$)$ 可以接受，但像 $(\$ 2*((1,2) - (2,3)) \$)$ 这种圆括号直接套嵌圆括号的形式会导致错误：

! Package tikz Error: + or - expected.

此时可以将 $(\$ \dots \$)$ 这个句式套嵌使用，如 $(\$ 2*(\$ (1,2) - (2,3) \$) \$)$ 是可以接受的。

如果在 perpendicular 坐标系中使用坐标计算式，可能需要将 \$ 的外层的圆括号换成花括号，例如：

```
\fill ({$(a.south)+(0,-40pt)$} -| {$(a.east)+(40pt,0)$}) circle(2pt);
```

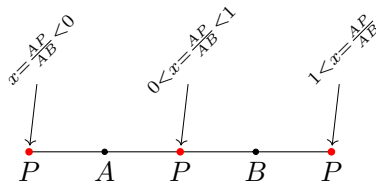
13.5.3 比例-角度定点句法

`<coordinate>!<number>!<second coordinate>`

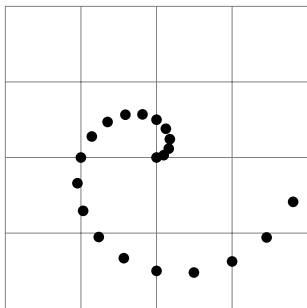
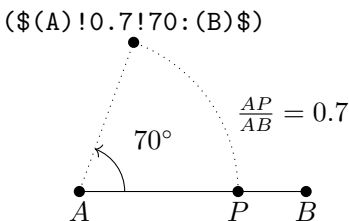
`<coordinate>!<number>!<angle>:<second coordinate>`

其中 `<number>` 是任意实数，`<angle>` 是角度制下的数值。系统会用 `\pgfmathparse` 来解析 `<number>`，所以 `<number>` 可以（如上文的 `<factor>`）是比较复杂的算式。

例如 $(A)!x!(B)$ ，等价于 $((1-x)*A+x*B)$ 。记 $(A)!x!(B)$ 为 P，则 x 与 P 的对应关系如下图所示

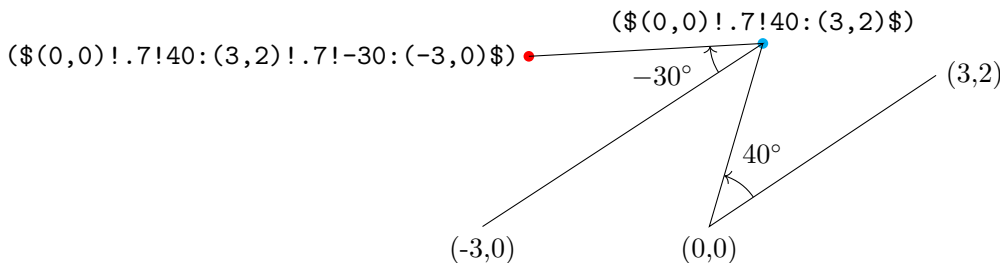


记 $(A)!x!y:(B)$ 为 Q，Q 这样得到：先计算 $(A)!x!(B)$ （记为 P），然后将线段 AP 绕点 A 旋转角度 y，点 P 变成 Q。例如， $(A)!0.7!70:(B)$ 如下图所示



```
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (4,4);
  \foreach \i in {0,0.1,...,2}
    \fill ($(2,2)! \i! \i*180:(3,2)$) circle (2pt);
\end{tikzpicture}
```

这种句式可以套嵌叠加使用，例如，如果设置 `\coordinate (x)at (A)!0.7!70:(B)`；那么 $(A)!0.7!70:(B)!0.5!20:(C)$ 就等价于 $(x)!0.5!20:(C)$ 。



```

\begin{tikzpicture}
\coordinate (o) at (0,0);
\coordinate (b) at (3,2);
\coordinate (a) at ($(o)!.7!40:(b)$);
\coordinate (d) at (-3,0);
\coordinate (c) at ($(a)!.7!-30:(d)$);

\fill [cyan] (a) circle (2pt);
\draw (b) node [right] {(3,2)}
-- (o) node [below] {(0,0)}
-- (a) node [above] {\lstineline&$(0,0)!.7!40:(3,2)$&}
pic [draw, ->, "$40^\circ\text{circ}$", angle radius=7mm,
angle eccentricity=1.7]
{angle = b--o--a};
\fill [red] (c) circle (2pt);
\draw (d) node [below] {(-3,0)}
-- (a)
-- (c)node [left] {\lstineline&$(0,0)!.7!40:(3,2)!.7!-30:(-3,0)$&}
pic [draw, <-, "$-30^\circ\text{circ}$", angle radius=7mm,
angle eccentricity=2]
{angle = c--a--d};
\end{tikzpicture}

```

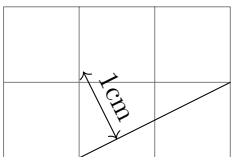
13.5.4 距离-角度定点句法

<coordinate>!<dimension>!<second coordinate>

<coordinate>!<dimension>!<angle>:<second coordinate>

其中 <dimension> 是带单位的长度，可以是负值。例如，如果设置 `\coordinate (C)at ($(A)!-3cm!(B)$)`；，则 (C) 在直线 (A)(B) 上，直线的方向向量是 $(B)-(A)$ ，有向线段 (A)(C) 的长度是 -3cm。

$(A)!-3cm!50:(B)$ ；是在 $(A)!-3cm!(B)$ ；的基础上，经旋转得到的点，与前一种句法类似。



```

\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\coordinate (a) at (1,0);
\coordinate (b) at (3,1);
\draw (a) -- (b);
\coordinate (c) at ($(a)!.25!(b)$);
\coordinate (d) at ($(c)!1cm!90:(b)$);
\draw [<->] (c) -- (d) node [sloped,midway,above] {1cm};
\end{tikzpicture}

```

13.5.5 正射影-角度定点句法

`<coordinate>!<projection coordinate>!<second coordinate>`

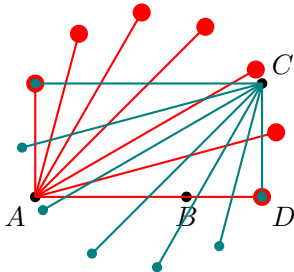
`<coordinate>!<projection coordinate>!<angle>:<second coordinate>`

例如，如果设置

```
\coordinate (D) at ($(A)!(C)!(B)$);
```

```
\coordinate (E) at ($(A)!(C)!50:(B)$);
```

则 (D) 是点 (C) 在直线 (A)(B) 上的垂足。不过，如果这种句法带有角度，其意义并不直观，如下例



```
\begin{tikzpicture}
\coordinate (A) at (0,0);
\coordinate (B) at (2,0);
\coordinate (C) at (3,1.5);
\coordinate (D) at ($(A)!(C)!(B)$);

\fill (A) node [below left] {$A$} circle (2pt)
(B) node [below] {$B$} circle (2pt)
(C) node [above right] {$C$} circle (2pt)
(D) node [below right] {$D$} circle (2pt);

\foreach \ang in {0,15,...,90}
\filldraw [red,line width=0.8pt] (A)
-- ($(A)!(C)!\ang:(B)$) circle (3pt) ;

\foreach \ang in {90,105,...,180}
\filldraw [teal,line width=0.8pt] (C)
-- ($(A)!(C)!\ang:(B)$) circle (1.5pt);
\end{tikzpicture}
```

14 设置路径的语句

设置路径的一般语句是

```
\path <specification>;
```

这个命令只能用在 `{tikzpicture}` 环境里。

一个路径由一个或数个路径算子（操作，operation）构成，例如一个路径可以由 move-to 操作，line-to 操作，curve-to 操作，圆弧算子 arc，圆算子 circle，矩形算子 rectangle，闭合（close）操作 cycle 等构成。一个路径中可以含有附属路径，如用算子 node 创建的路径就是附属部分。一个路径除去它的附属部分，其余部分叫作“主路径”（main path）。

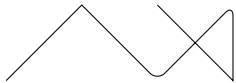
可以在路径的不同位置设置多个方括号（内有选项），选项的作用效果有以下几类：

1. 有的选项只对该选项之后的部分路径有效，例如，rounded corners, sharp corners, 还有变换选项 (transformation options) 是这种类型的。



```
\tikz \draw (0,0) -- (1,1)
[rounded corners] -- (2,0) -- (3,1)
[sharp corners] -- (3,0) -- (2,1);
```

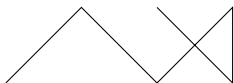
2. 有的选项用在路径上的 {scope} 环境中，其效果只在本 {scope} 环境中有效，选项 rounded corners 属于这种类型。上面的图形也可以如下画出



```
\tikz \draw (0,0) -- (1,1)
{[rounded corners] -- (2,0) -- (3,1)}
-- (3,0) -- (2,1);
```

上面的例子中，{scope} 环境外部的 sharp corners 样式与 {scope} 环境内部的 rounded corners 样式各自独立起作用，互不影响。

3. 有的选项设置，例如，color=, 无论放在“主路径”上的哪个位置，都对整个路径（包括附属路径）有效（不过 node 对象，pic 对象可以在自己的选项里重设颜色）。



```
\tikz \draw (0,0) -- (1,1)
-- (2,0) -- (3,1)
{[color=cyan] -- (3,0) -- (2,1) [line width=4pt]};
```

上面例子中，加在主路径上的 {scope} 环境里的颜色选项 color=cyan, 线宽选项 line width=4pt 都不起作用，因为主路径的默认设置是 draw=black, line width=0.4pt。多数选项属于这种类型。

在默认下，\path 命令只是创建路径，不画出路径，其作用可能仅仅是使得画布尺寸变大。

/tikz/every path (style, 初始值 empty)

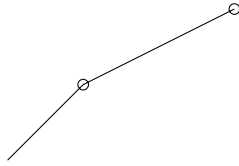
(按设置的有效范围) 为每个路径添加样式。



```
\begin{tikzpicture}
[fill=yellow!80!black, % 只对 fill 路径设置填充色
every path/.style={draw}] % 对所有路径加 draw
\fill (0,0) rectangle +(1,1);
\shade (2,0) rectangle +(1,1);
\end{tikzpicture}
```

/tikz/insert path=<path>

这个选项的参数值是“路径”（不是 node）（这个路径可以只有选项，没有坐标点），用来在路径的适当位置插入 <path>, 插入的 <path> 作为子路径属于整个路径（node 不属于路径）。例如

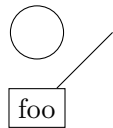


```
\tikz [c/.style={insert path={circle[radius=2pt]}}]
\draw (0,0) -- (1,1) [c] -- (3,2) [c];
```

上面例子中，将选项 `c` 定义为圆形插入路径，`[c]` 展开后就是 `circle[radius=2pt]`，所以 `[c]` 应当用在坐标之后。

`/tikz/append after command=<path>`

这里 `<path>` 是个路径（不是 node）（这个路径可以只有选项，没有坐标点）。如果给路径带上这个选项，在路径完成时，`<path>` 会被执行，被执行的 `<path>` 作为子路径属于整个路径。



```
\tikz \draw node [append after command=
{(foo)--(1,1) (0,1)circle(10pt)},draw]
(foo){foo};
```

如果在一个路径中多次使用该选项，则它们按次序叠加。

`/tikz/prefix after command=<path>`

与 `append after command` 类似，只是这里的 `<path>` 在路径开始创建时，先行执行。

14.1 The Move-To Operation

如果路径至少包含两个点，并且这些点之间有连续的连线，那么连续连线的第一个点就是 `move-to` 操作的对象，也就是说，`move-to` 操作使得“画笔”跳到连线的第一个点然后画线。因此 `move-to` 操作的作用有 2 个：（1）开启路径；（2）开启一个子路径，从而将路径分为数个子路径。一个路径的“子路径”通常指的是被 `moveto` 操作分隔的各个连续部分。例如

```
\draw (0,0)--(1,0) (1,1)--(2,1) (3,0);
```

这个路径中有两段分离的子路径（两个连续线段）和一个孤立点，其中点 `(0,0)`，`(1,1)` 是 `move-to` 操作的对象。`move-to` 操作在点 `(0,0)` 开启路径，在点 `(1,1)` 处开启一个子路径（延续路径），而点 `(3,0)` 是个孤立点，它不是一个子路径，但是属于当前路径，包含在当前路径的边界盒子中。

注意下面的路径

```
\draw (0,0)--(1,0) (1,0)--(2,1);
```

包含两段分离的连续线段，尽管这两个线段有重合点，在视觉上它们是“连在一起的”，但由于第二个线段的起点 `(1,0)` 是 `move-to` 操作的对象，所以这两个线段被 `move-to` 操作分离为两个“独立子路径”。

如果一个路径只包含一个点，例如

```
\path (0,0);
```

那么这个点 `(0,0)` 被看作是一个孤立点。

如果一个路径只包含两个独立点，例如

```
\path (0,0) (1,1);
```

则会导致错误。

有一个名称为 `current subpath start` 的预定义 node，这个 node 对象的形状属性是 `coordinate`（可以把这个 node 当作坐标对待），`current subpath start` 的位置是当前子路径的第一个坐标，所以在构造子路径时可以引用这个位置，例如



```
\begin{tikzpicture}[line width=2mm]
\draw (0,0) rectangle (1,1)
      (2,0)--(3,0)--(3,1) -- (current subpath start) ;
\end{tikzpicture}
```

注意上面例子中，三角形的起点和终点尽管是同一个点，但它不是闭合的，它的起点-终点结合部有个缺口。可以使用关键词 `cycle` 来把三角形做成闭合的，从而补上这个缺口，见下文。

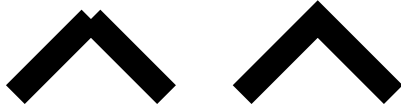
14.2 The Line-To Operation

14.2.1 线段

```
\path . . . --<coordinate or cycle> . . . ;
```

在一个路径中，如果两个坐标（子路径）之间用两个连字符（减号）“--”联系，那么这两个坐标（子路径）之间的联系方式就是“line-to”方式，即用线段联系起来。

线段的起止点由坐标指定，但是线段有一定的线宽，当两个线段有公共点且不在同一直线上时，如果这两个线段在公共点处是“move-to”联系的，在公共点处会有一个“缺口”；如果这两个线段在公共点处是“line-to”联系的，在公共点处没有“缺口”，例如



```
\begin{tikzpicture}[line width=10pt]
\draw (0,0) --(1,1) (1,1) --(2,0);
\draw (3,0) -- (4,1) -- (5,0);
\end{tikzpicture}
```

一个封闭曲线（多边形）的起止点相同，为了不在该点出现缺口，使用 `cycle` 关键词，它使得曲线（多边形）变成闭合的。例如



```
\begin{tikzpicture}[line width=10pt]
\draw (0,0) -- (1,1) -- (1,0) -- (0,0) (2,0) -- (3,1) -- (3,0) -- (2,0);
\draw (5,0) -- (6,1) -- (6,0) -- cycle (7,0) -- (8,1) -- (8,0) -- cycle;
\useasboundingbox (0,1.5); % make bounding box higher
\end{tikzpicture}
```

`cycle` 可以用于“--”，“..”，“sin”，“grid”之后，但不能用于 `graph` 或 `plot` 之后。

14.2.2 横线和竖线

```
\path . . . -|<coordinate or cycle> . . . ;
\path . . . |-<coordinate or cycle> . . . ;
```

将两个位置用横线和竖线连接起来。

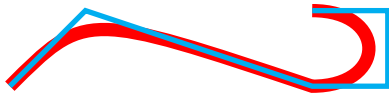
14.3 The Curve-To Operation

curve-to 算子创建 Bézier 曲线。

```
\path . . . ..controls<c>and<d>..<y or cycle> . . . ;
```

句子用起点、第一控制点 (<c>)、第二控制点 (<d>)、终点 (<y> 或 cycle) 来构造 3 次 Bézier 曲线。

如果没有 and<d> , 就默认 <d> 等同于 <c>.



```
\begin{tikzpicture}
\draw[color=red,line width=5pt] (0,0) .. controls (1,1) .. (4,0)
.. controls (5,0) and (5,1) .. (4,1);
\draw[color=cyan,line width=2pt] (0,0) -- (1,1) -- (4,0)
-- (5,0) -- (5,1) -- (4,1);
\end{tikzpicture}
```



```
\begin{tikzpicture}
\draw[line width=10pt] (0,0) -- (2,0) .. controls (1,1) .. cycle;
\end{tikzpicture}
```

当线段与控制曲线有公共点时, 在公共点处也有是否连接的问题, 如果二者是 move-to 连接方式, 连接部分可能有缺口。



```
\begin{tikzpicture}[line width=10pt]
\draw (0,0) -- (1,1) (1,1) .. controls (1,0) and (2,0) .. (2,0);
\draw [yshift=-1.5cm]
(0,0) -- (1,1) .. controls (1,0) and (2,0) .. (2,0);
\end{tikzpicture}
```

14.4 矩形算子

```
\path . . . rectangle<corner or cycle> . . . ;
```

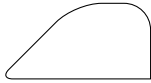
这个算子画由水平线和竖直线构成的矩形, 当前点由 rectangle 前的点变成 rectangle 后的点, 这两个点作为矩形对角线的端点。

14.5 Rounding Corners

`/tikz/rounded corners=<inset>` (默认值 4pt)

当使用这个选项时，所有直角都改为圆角，<inset> 设置圆角弧的半径，默认为 4pt. 注意 <inset> 不受选项 `scale` 的影响。

在一个路径上，可以在表示线条的“-”符号之前使用 `rounded corners` 选项，之后可以用 `sharp corners` 选项关闭圆角选项功能，也可以在适当位置重设圆角半径。



```
\begin{tikzpicture}
\draw (0,0) [rounded corners=10pt] -- (1,1) -- (2,1)
          [sharp corners] -- (2,0)
          [rounded corners=5pt] -- cycle;
\end{tikzpicture}
```

注意，仅当直线角是 90° 时，`rounded corners` 选项生成的圆角才是圆弧。如果将 `rounded corners` 选项用于很小的线段构成的直线角，可能会得到意外结果。

`/tikz/sharp corners`

该选项用于关闭圆角选项功能。

14.6 圆、椭圆算子

`\path . . . circle[<options>] . . . ;`

以当前点为圆心，用 `circle` 算子画圆。也可以在方括号里用 `at` 选项指定圆心坐标。方括号里的 <options> 设置圆的半径。

`/tikz/x radius=<value>`

如果使用这个选项，就指定画椭圆，该选项指定椭圆的横半轴长度。<value> 可以带有长度单位，也可以不带长度单位，如果不带长度单位就使用 `xy-` 坐标系。

`/tikz/y radius=<value>`

类似 `x radiu`.

`/tikz/radius=<value>`

同时设置 `x radiu` 和 `y radius`，可以作为环境选项。

`/tikz/at=<coordinate>`

设置圆心坐标。注意这个选项不能用作环境选项。

<options> 还可以含有其它选项，例如，`rotate` 或 `scale`.

`/tikz/every circle` (style, 无默认值)

用这个 key 设置圆的某些样式，例如，半径，颜色等。这个 key 对 `ellipse` 算子也有效。

如果你觉得选项 `radius` 或 `x radius` 写起来过长，不方便，可以自定义 key 来代替它们：

```
\tikzset{r/.style={radius=#1},rx/.style={x radius=#1},ry/.style={y radius=#1}}
```

这样定义后，就可以使用 `circle [r=1cm]` 或 `circle [rx=1,ry=1.5]` 这样的语句了。

注意，有一个较旧的句法来指定圆的半径，例如 `circle (1pt)`，`circle (0.5)`（数字单位默认为 cm），用圆括号指定半径。

```
\path . . . ellipse[<options>] . . . ;
```

与 `circle` 类似。也有个较旧的句法：`ellipse (<x radius> and <y radius>)` 来指定椭圆的横半轴和纵半轴。

14.7 The Arc Operation

Arc 操作的详细解释见 §97.6.

```
\path . . . arc[<options>] . . . ;
```

以当前点为起点，用 `arc` 算子画圆弧或椭圆弧。在方括号里，用选项 `x radius` 和 `y radius` 设置半径，用选项 `start angle`，`end angle`，`delta angle` 设置圆弧的起止角度或起止角度之差。当然，`start angle` 和 `end angle`，`start angle` 和 `delta angle`，`end angle` 和 `delta angle`，这三种组合都可确定角度范围。

```
/tikz/start angle=<degrees>
```

```
/tikz/end angle=<degrees>
```

```
/tikz/delta angle=<degrees>
```

注意圆弧从起点到终点的方向，如果 `start angle` 大于 `end angle`，则按顺时针方向；如果 `start angle` 小于 `end angle`，则按逆时针方向。

有一个较旧的句法来指定圆弧：`arc(<start angle>:<end angle>:<radius>)`。

14.8 The Grid Operation

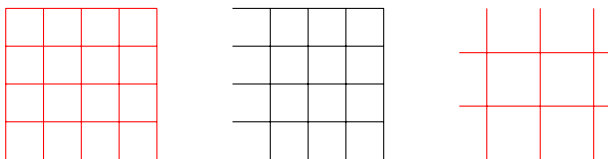
画网格

```
\path . . . grid[<options>]<corner or cycle> . . . ;
```

以 `grid` 之前和之后的点为网格对角线的端点，当前点由前一个端点变成后一个端点。方括号里的 `<options>` 可以有以下选项。

```
/tikz/step=<number or dimension or coordinate> (无默认值，初始值 1cm)
```

同时设置网格的横向和纵向间距。如果设置的数值不带长度单位，例如 `step=2`，表示间距为单位向量长度的 2 倍。单位向量长度用选项 `x=<dimension>`，`y=<dimension>` 来设置（参考坐标变换）。如果设置 `step={{a,b}}`，则横向间距为 `a`，纵向间距为 `b`。



```
\begin{tikzpicture}
\draw [red] (0,0) grid [step=0.5cm] (2,2);
```

```
\draw (3,0) grid [step=0.5] (5,2);
\draw [red] (6,0) grid [step=(45:0.5*sqrt(2))] (8,2);
\end{tikzpicture}
```

`/tikz/xstep=<dimension or number>` (无默认值, 初始值 1cm)

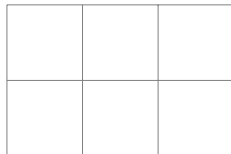
`/tikz/ystep=<dimension or number>` (无默认值, 初始值 1cm)

分别设置横向、纵向间距。

注意, 坐标系的网格总是以原点为一个格点。由于数据计算时有误差, 网格外层本该有的线可能被忽略, 此时需要手工添加。

`/tikz/help lines` (style, 初始值 line width=0.2pt,gray)

这个样式是预定义的, 常用来画网格。



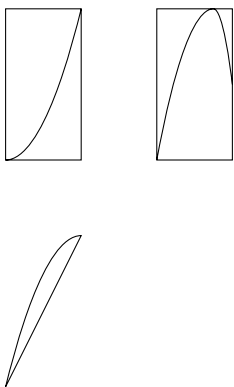
```
\tikz \draw[help lines] (0,0) grid (3,2);
```

14.9 The Parabola Operation

parabola 算子画出来的图形实际是一段或两段抛物线。

`\path . . . parabola [<options>] bend <bend coordinate> <coordinate or cycle> . . . ;`

以当前点为起点, parabola 算子画出一段或两段抛物线。如果 parabola 算子画出一段抛物线, 那么这段抛物线的起点或终点是抛物线的顶点。bend<bend coordinate> 指定抛物线的顶点坐标, 如果不写出 bend <bend coordinate>, 就默认当前点是顶点。如果写出 bend <bend coordinate>, 那么 parabola 算子通常画出两段抛物线, 这两段抛物线有公共的顶点, 但二者未必属于同一个抛物线函数, 这决定于命令中具体的坐标设置。观察下面的例子。

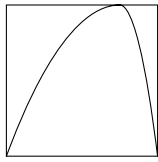


```
\begin{tikzpicture}
\draw (0,0) rectangle (1,2)
(0,0) parabola (1,2);
\draw[xshift=2cm] (0,0) rectangle (1,2)
(0,0) parabola bend (.75,2) (1,1);
\draw[yshift=-3cm] (0,0) --
(1,2) parabola cycle;
\end{tikzpicture}
```

<options> 可以是以下选项。

`/tikz/bend=<coordinate>`

作用与 bend<bend coordinate> 类似, 设置抛物线顶点坐标, <coordinate> 可以是相对坐标。注意, 画出的两段抛物线未必属于同一个抛物线函数。



```
\begin{tikzpicture}
  \draw (0,0) rectangle (2,2)
        (0,0) parabola bend (1.5,2) (2,0);
\end{tikzpicture}
```

`/tikz/bend pos=<fraction>`

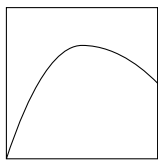
当 `bend=<coordinate>` 是相对坐标时本选项有效。例如，

```
\draw (a) parabola [bend pos=r] bend=+(b) (c);
```

这个语句中 `bend=` 使用相对坐标，`bend pos=r` 是任意实数。所确定的顶点坐标是 $(a)+r*((c)-(a))+b$ ，在 (a) 与顶点之间画抛物线，在 (c) 与顶点之间也画抛物线。

`/tikz/parabola height=<dimension>`

这个选项等价于 `[bend pos=0.5,bend={+(0pt,<dimension>)}]`，也就是说，顶点在起点与终点的中间，顶点相对于“起-止点的中点”的高度是 `<dimension>`，注意 `<dimension>` 要带上长度单位，否则默认长度单位是 `pt`。



```
\begin{tikzpicture}
  \draw (0,0) rectangle (2,2)
        (0,0) parabola [parabola height=1cm] (2,1);
\end{tikzpicture}
```

有几个便捷选项

`/tikz/bend at start` (style, 无默认值)

等效于选项 `bend pos=0,bend={+(0,0)}`。

`/tikz/bend at end` (style, 无默认值)

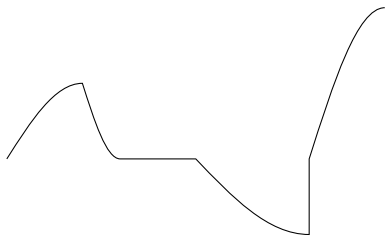
等效于选项 `bend pos=1,bend={+(0,0)}`。

14.10 The Sine and Cosine Operation

```
\path . . . sin<coordinate or cycle> . . . ;
```

以当前点为起点，随后以 `sin` 算子画曲线，`sin` 算子之后的坐标点是曲线终点。起点与终点之间的曲线，是将正弦函数 $\sin(x)$ 在区间 $[0, \frac{\pi}{2}]$ 上的图像作某种变换后得到的图形。

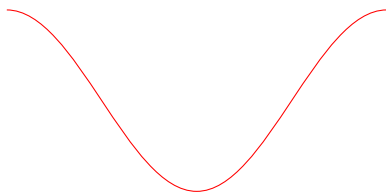
如果起点与终点在同一水平线或竖直线上，则 `sin` 算子画一个线段。如果起点在终点下方，则 `sin` 算子画的曲线形态类似函数 $\sin(x)$ 在区间 $[0, \frac{\pi}{2}]$ 上的图像。如果起点在终点上方，则 `sin` 算子画的曲线形态类似函数 $\sin(x)$ 在区间 $[\pi, \frac{3\pi}{2}]$ 上的图像。



```
\begin{tikzpicture}
\draw (0,0) sin (1,1) sin (1.5,0) sin (2.5,0)
      sin (4,-1) sin (4,0) sin (5,2);
\end{tikzpicture}
```

`\path . . . cos<coordinate or cycle> . . . ;`

cos 算子与 sin 算子类似，起点与终点之间的曲线，是将余弦函数 $\cos(x)$ 在区间 $[0, \frac{\pi}{2}]$ 上的图像作某种变换后得到的图形。



```
\begin{tikzpicture}[xscale=1.57,scale=0.8]
\draw[color=red] (0,1.5) cos (1,0) sin (2,-1.5)
      cos (3,0) sin (4,1.5);
\end{tikzpicture}
```

14.11 The SVG Operation

14.12 The Plot Operation

14.13 The To Path Operation

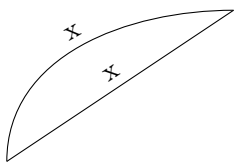
`\path . . . to[<options>] <nodes> <coordinate or cycle> . . . ;`

to 算子在起点 (to 之前的点) 与目标点 (to 之后的点) 之间插入一个路径。例如，

(a) to (b) 画一个线段，等价于 (a) -- (b)

(a) to [out=135,in=45] (b) 画一个曲线

(a) to node {x} (b) 画一个线段并给线段加标签，等价于 (a) -- node {x} (b)

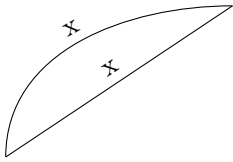


```
\begin{tikzpicture}
\draw (0,0) to node [sloped,above] {x} (3,2);
\draw (0,0) to[out=90,in=180] node [sloped,above] {x} (3,2);
\end{tikzpicture}
```

与 to 算子相关的选项如下。

`/tikz/edge node=<node specification>`

这个选项设置 to 路径的 node 标签。对比前面的例子：



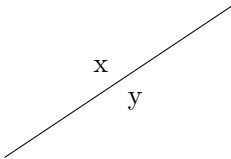
```
\begin{tikzpicture}
\draw (0,0) to [edge node={node [sloped,above] {x}}] (3,2);
\draw (0,0) to [out=90,in=180,
edge node={node [sloped,above] {x}}] (3,2);
\end{tikzpicture}
```

`/tikz/edge label=<text>`

这是 `edge node={node[auto]{<text>}}` 的简洁用法。

`/tikz/edge label'=<text>`

这是 `edge node={node[auto,swap]{<text>}}` 的简洁用法。



```
\tikz \draw (0,0) to [edge label=x, edge label'=y] (3,2);
```

如果调用 `quotes` 程序库，还有其它设置标签的方法。

`/tikz/every to (style, 初始值 empty)`

这个选项设置的样式会作为（针对 `to` 路径的）预先设置，作用于每个 `to` 路径。

`/tikz/to path=<path>`

这个选项设置 `to` 算子绘制的路径的默认样式，即定义 `to` 算子的实际效果。该选项实际执行下面的代码模式：

```
{[every to,<options>] <path>}
```

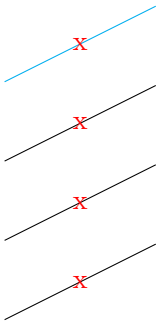
其中的 `every to` 代表由该 key 设置的选项，`<options>` 是针对 `to` 算子的其它选项，而 `<path>` 设置需要用到以下三个宏：

- `\tikztostart`，这个宏将展开为起点坐标的数值，即不带圆括号的数据，例如“1,2”，如果给宏加圆括号 (`\tikztostart`)，则展开为“(1,2)”。
- `\tikztotarget`，这个宏将展开为目标点坐标的数值，不带圆括号。
- `\tikztonodes`，这个宏将展开为 `to` 路径的 `node` 标签，默认标签位置在路径中间。

`<path>` 的默认设置是：`--(\tikztotarget) \tikztonodes`，所以

(a) `to[red] node {x}` (b) 展开为 (a) `-- node[red] {x}` (b)

注意这里的 `red` 颜色选项仅仅作用于标签 `x`，观察下面的例子



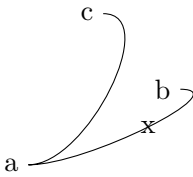
```
\tikz \draw (0,0) to[red] node {x} (2,1)[cyan];

\tikz \draw (0,0) {[red]--node{x} (2,1)};

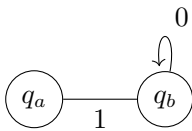
\tikz[to path={[red] -- (\tikztotarget) \tikztonodes}]
  \draw (0,0) to node{x} (2,1);

\tikz [every to/.style={red}] \draw (0,0) to node {x} (2,1);
```

注意句式 `--\tikztonodes` (`\tikztotarget`) 是错误的, 因为 Tikz 不允许 “--” 后面的宏展开为 node 语句。

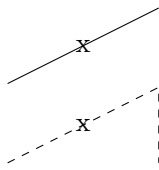


```
\begin{tikzpicture}[to path={
.. controls +(1,0) and +(1,0) .. (\tikztotarget) \tikztonodes}]
\node (a) at (0,0) {a};
\node (b) at (2,1) {b};
\node (c) at (1,2) {c};
\draw (a) to node {x} (b)
      (a) to (c);
\end{tikzpicture}
```



```
\tikzset{
  my loop/.style={to path={
    .. controls +(80:1) and +(100:1)
    ..(\tikztotarget) \tikztonodes}},
  my state/.style={circle,draw}}
\begin{tikzpicture}[shorten >=2pt]
\node [my state] (a) at (210:1) {$q_a$};
\node [my state] (b) at (330:1) {$q_b$};
\draw[->] (a) to node[below] {1} (b)
          to [my loop] node[above right] {0} (b);
\end{tikzpicture}
```

由于 to 路径是由 to path 选项设置的, 这些设置处于一个花括号分组内, every to 设置的某些选项, 例如 draw, 会被花括号外的主路径的选项抑制, 从而无效。可以使用 append after command 选项来修改主路径的设置, 观察下例:



```
\tikz[every to/.style={dashed}]
  \draw (0,0) to node {x} (2,1);

\tikz[every to/.style={append after command={[[draw,dashed]]}}]
  \draw (0,0) to node {x} (2,1)--(2,0);
```

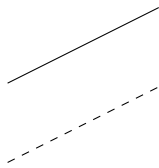
`/tikz/execute at begin to=<code>`

<code> 会在 to 路径之前被执行，可以用来添加其它路径或者做某些计算。

`/tikz/execute at end to=<code>`

<code> 会在 to 路径之后被执行，可以用来添加其它路径。

execute at end to 与 append after command 有所不同，比较下面的例子和上面的例子



```
\tikz[execute at end to={[[draw,dashed]]}]
  \draw (0,0) to (2,1);

\tikz[execute at end to={\path[draw,dashed];}]
  \draw (0,0) to (2,1);
```

在 topaths 程序库中定义了一些专用于 to 算子的选项，见 §70.

14.14 The Foreach Operation

14.15 The Let Operation

先看一个例子



```
\tikz {\draw [red,line width=4pt]
  let
    \n1={sqrt(2)+0.5},
    \n2=1
  in
    (0,0)--(\n1,\n2);}
```

这个例子中，let 引起对宏 \n1, \n2 的赋值，然后在 in 引起的命令中使用这些宏。这就是 let ... in 算子的基本思路。

`\path . . . let <assignment>,<assignment>,<assignment>. . . in . . . ;`

被赋值的宏有 3 种类型：

1. 数值 (number): 用宏 \n<number register> 来存储数值，**注意这里必须用字母“n”，而且这个宏必须带后缀**。例如可以带数字后缀，\n1, \n2；可以带文字字符后缀，此时需要给文字字符加花括号，如 \n{text}, \n{string} 等等。

2. 坐标点 (point): 用宏 `\p<number register>` 来存储坐标, **注意这里必须用字母“p”, 而且这个宏也必须带后缀**, 后缀格式与 `\n<number register>` 类似。
3. 坐标分量: 设置坐标点宏后, 可以用坐标分量宏来引用坐标分量, 例如, 设置 `let \p3=(a,b)` 后, 则宏 `\x3` 的值就是分量 a, 宏 `\y3` 的值就是分量 b; 再如, 设置 `let \p{text}=(a,b)` 后, 宏 `\x{text}` 的值就是分量 a, 宏 `\y{text}` 的值就是分量 b. 也就是说, 坐标分量宏与坐标点宏是通过后缀对应起来的, 注意这里坐标分量宏的名称必须使用字母“x, y”.
对坐标分量宏的赋值由程序自动完成。

以上宏的值存储在专门的寄存器 (register) 中, 这是专属于 Tikz 的寄存器。

`\n<number register>={<formula>}`

`<formula>` 可以是比较复杂的形式, 它会被宏 `\pgfmathparse` 计算, 计算结果存储在以 `<number register>` 为名称的寄存器中。如果 `<formula>` 中出现长度单位 (如 $2*3\text{cm}/2$), 则计算结果是以 pt 为单位的长度。作为宏后缀的 `<number register>` 是宏参数, 它是个 $\text{T}_{\text{E}}\text{X}$ 参数, 其中可以有空格。

`\p<number register>={<formula>}`

这个宏只储存坐标数据, 所存储的坐标数据不带圆括号, 并且所有数据都转为以 pt 为单位的长度储存起来。

设置 `let \p1=(1pt,1pt+2pt)` 后, `\p1` 展开为不带圆括号的数据 “1pt,3pt”, 而 `(\p1)` 展开为坐标形式 (1pt,3pt), 所以在引用坐标宏绘图时应该给宏加上圆括号。

`\x{<point register>}`

这个宏用来引用后缀为 `<point register>` 的坐标宏的第一个分量, 它的值是一个以 pt 为单位的长度。

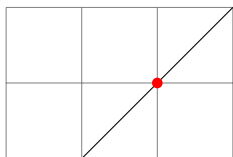
`\y{<point register>}`

这个宏用来引用后缀为 `<point register>` 的坐标宏的第二个分量, 它的值是一个以 pt 为单位的长度。

按照坐标处理规则, $(1\text{pt}+2,3)$ 等于 $(3\text{pt},3\text{cm})$, $(1\text{pt}+2,3+4\text{pt})$ 等于 $(3\text{pt},7\text{pt})$, 所以在使用 `\x{<point register>}` 和 `\y{<point register>}` 做计算时要注意长度单位。

注意 let ... in 语句中的标点符号, 宏赋值以逗号结尾, 但 in 之前没有逗号。

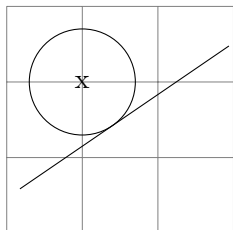
let 引起的赋值只在 let ... in 语句范围内有效, 但是 let ... in 语句内的 coordinate 赋值在整个 `{tikzpicture}` 环境内有效, 如下例



```

\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\path
  let
    \p1 = (1,0),
    \p2 = (3,2),
    \p{center} = ($ (\p1) !.5! (\p2) $)
  in
    coordinate (p1) at (\p1)
    coordinate (p2) at (\p2)
    coordinate (center) at (\p{center});
\draw (p1) -- (p2);
\fill[red] (center) circle [radius=2pt];
\end{tikzpicture}

```

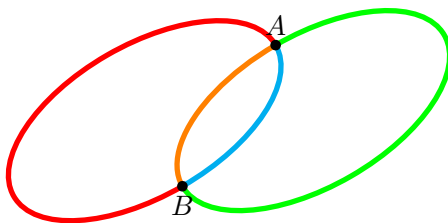


```

\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,3);
\coordinate (a) at (rnd,rnd);
\coordinate (b) at (3-rnd,3-rnd);
\draw (a) -- (b);
\node (c) at (1,2) {x};
\draw let \p1 = ($ (a)!(c)!(b) - (c) $),
          \n1 = {veclen(\x1,\y1)}
        in circle [at=(c), radius=\n1];
\end{tikzpicture}

```

下面是个稍微复杂一些的例子:



```

\begin{tikzpicture}[rotate=30]
\draw [name path=p1] (0,0) ellipse (2 and 1);
\draw [name path=p2] (2,-1) ellipse (2 and 1);
\path [name intersections={of=p1 and p2,by={A,B}}];
\coordinate (A') at ($(A)-(2,-1)$);
\coordinate (B') at ($(B)-(2,-1)$);

```

```

\draw [red,line width=2pt]
  let \p1=(A), \p2=(B)
  in (A) arc [start angle={atan(\y1/\x1)}, end angle={atan(\y2/\x2)+180},
            x radius=2cm, y radius=1cm];
\draw [cyan,line width=2pt]
  let \p1=(A), \p2=(B)
  in (A) arc [start angle={atan(\y1/\x1)}, end angle={-atan(\y2/\x2)},
            x radius=2cm, y radius=1cm];
\draw [green,line width=2pt]
  let \p1=(A'), \p2=(B')
  in (A) arc [start angle={atan2(\y1,\x1)}, end angle={-atan2(\y2,\x2)},
            x radius=2cm, y radius=1cm];
\draw [orange,line width=2pt]
  let \p1=(A'), \p2=(B')
  in (A) arc [start angle={atan2(\y1,\x1)}, end angle={atan2(\y2,\x2)},
            x radius=2cm, y radius=1cm];
\fill (A) circle (2pt) node [above] {$A$}
      (B) circle (2pt) node [below] {$B$};
\end{tikzpicture}

```

14.16 The Scoping Operation

14.17 The Node and Edge Operations

14.18 The Graph Operation

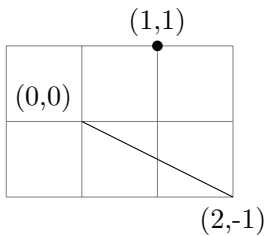
14.19 The Pic Operation

14.20 The PGF-Extra Operation

在构建路径过程中，有时候需要中断路径，执行某些 $\text{T}_{\text{E}}\text{X}$ 代码，然后再继续构建路径，这就用到下面的命令。

`\pgfextra{<code>}`

这个命令只能用在路径构建过程中，<code> 的有效范围也限于路径之内。



```

{\newdimen\mydima
\newdimen\mydimb
\begin{tikzpicture}
  \mydima=1cm
  \mydimb=1cm
\draw [help lines] (-1,-1) grid (2,1);
\draw (0,0) node[above left]{(0,0)}
  \pgfextra{\mydima=2cm \mydimb=-1cm}
    -- (\mydima,\mydimb) node[below]{(2,-1)};
\fill (\mydima,\mydimb) node[above]{(1,1)}
  circle (2pt);
\end{tikzpicture}}

```

`\pgfextra<code> \endpgfextra`

如果 `\pgfextra` 之后的 `<code>` 没有用花括号括起来，就需要用 `\endpgfextra` 结束 `\pgfextra` 的作用。

15 Actions on Paths

15.1 Overview

`\draw` 用在 `{tikzpicture}` 环境内，是 `\path[draw]` 的缩略。

`\fill` 用在 `{tikzpicture}` 环境内，是 `\path[fill]` 的缩略。

`\filldraw` 用在 `{tikzpicture}` 环境内，是 `\path[fill,draw]` 的缩略。

`\pattern` 用在 `{tikzpicture}` 环境内，是 `\path[pattern]` 的缩略。

`\shade` 用在 `{tikzpicture}` 环境内，是 `\path[shade]` 的缩略。

`\shadedraw` 用在 `{tikzpicture}` 环境内，是 `\path[shadedraw]` 的缩略。

`\clip` 用在 `{tikzpicture}` 环境内，是 `\path[clip]` 的缩略。

`\useasboundingbox` 用在 `{tikzpicture}` 环境内，是 `\path[useasboundingbox]` 的缩略。

15.2 Specifying a Color

`/tikz/color=<color name>`

这个选项为环境中的 `fill`, `draw`, `text` 等设置颜色（不包括 shading）。`<color name>` 可以是预定义的颜色名称，也可以是自定义的颜色。支持 `xcolor` 宏包的叹号句法，例如

```
\tikz \fill[color=red!20] (0,0) circle (1ex);
```

“color=” 可以省略，只写出 `<color name>`。

15.3 Drawing a Path

`/tikz/draw`

`/tikz/draw=<color>` (默认为环境的颜色设置)

画出路径线，指定线条的颜色。线条颜色默认为环境的颜色设置，默认环境颜色是黑色。

如果设置 `draw=none`，则取消画线。

本选项通常用作路径选项，也可以用于环境选项来指示环境内线条的颜色（仅仅指示颜色，不画线）。

15.3.1 Line Width, Line Cap, and Line Join

以下选项是关于线条的，仅在启用画线功能（使用 `draw` 选项）时有效。

`/tikz/line width=<dimension>` (无默认值，初始值 0.4pt)

设置线条的线宽，尺寸带单位。

`/tikz/ultra thin` (style, 无默认值)

本选项设置线宽为 0.1pt.

`/tikz/very thin` (style, 无默认值)

本选项设置线宽为 0.2pt.

`/tikz/thin` (style, 无默认值)

本选项设置线宽为 0.4pt.

`/tikz/semithick` (style, 无默认值)

本选项设置线宽为 0.6pt.

`/tikz/thick` (style, 无默认值)

本选项设置线宽为 0.8pt.

`/tikz/very thick` (style, 无默认值)

本选项设置线宽为 1.2pt.

`/tikz/ultra thick` (style, 无默认值)

本选项设置线宽为 1.6pt.

`/tikz/line cap=<type>` (无默认值，初始值 `butt`)

设置路径线条端点的“帽子”样式，可选的样式是 `round`（圆形帽子），`rect`（方形帽子），`butt`（光头，无帽子，但是方头），观察下面的例子：



```
\begin{tikzpicture}
\begin{scope}[line width=10pt]
\draw[line cap=rect] (0,0) -- (2,0);
\draw[line cap=butt] (0,.5) -- (2,.5);
\draw[line cap=round] (0,1) -- (2,1);
\end{scope}
\end{tikzpicture}
```

`/tikz/line join=<type>` (无默认值, 初始值 `miter`)

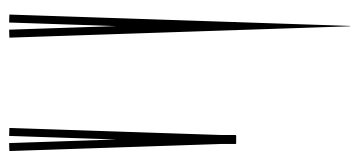
设置路径上的角（不包括路径端点）的外缘的外观样式。可选的样式是 `round`（圆），`bevel`（平），`miter`（尖）。



```
\begin{tikzpicture}[line width=10pt]
\draw[line join=round] (0,0) -- ++(.5,1) -- ++(.5,-1);
\draw[line join=bevel] (1.25,0) -- ++(.5,1) -- ++(.5,-1);
\draw[line join=miter] (2.5,0) -- ++(.5,1) -- ++(.5,-1);
\useasboundingbox (0,1.5);
\end{tikzpicture}
```

`/tikz/miter limit=<factor>` (无默认值, 初始值 `10`)

当对角设置 `line join=miter` 并且角度非常小时，角的外缘会很尖锐，角尖很长。如果角尖突出角顶点的距离大于 `<factor>` 与线宽的乘积，程序会自动设置 `line join=bevel`。



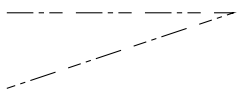
```
\begin{tikzpicture}[line width=3pt]
\draw (0,0) -- +(3,.1) -- +(0,.2);
\draw[miter limit=50] (0,1.5) -- +(3,.1) -- +(0,.2);
\useasboundingbox (14,0);
\end{tikzpicture}
```

15.3.2 Dash Pattern

`/tikz/dash pattern=<dash pattern>`

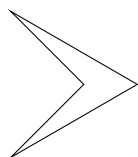
设置虚线样式，句法与 `metafont` 相同。例如，`on 2pt off 3pt on 4pt off 4pt`，其中 `on` 表示移动并画线，`off` 表示移动但不画线；`on` 后的尺寸 `2pt` 表示画线的长度，`off` 后的尺寸 `3pt` 表示移动的距离。这样就定义了一段虚线样式，它会在起止点间重复画出。

注意，这种定义必须以 `on` 开头。



```
\begin{tikzpicture}[dash pattern=on 10pt off 2pt on 2pt
    off 2pt on 5pt off 5pt]
  \draw (0pt,0pt) -- (3cm,0pt)--(0,-1);
\end{tikzpicture}
```

如果定义虚线 `dash pattern=on 0pt off 4cm`，实际上就是画线（并非不画线）但线条没有颜色。



```
\begin{tikzpicture}
\draw [-{Stealth[angle=60:2cm,open,inset=1cm]},
    dash pattern=on 0pt off 4cm ]
  (0,0)--(3,0);
\end{tikzpicture}
```

`/tikz/dash phase=<dash phase>` (无默认值，初始值 0pt)

由于 `dash pattern` 定义的虚线样式是重复画出的，可以看作是周期性的曲线，因而是有相位的，带单位的尺寸 `<dash phase>` 代表相位，相当于平移。

`/tikz/solid` (style, 无默认值)

实线，这是线条的默认值。

`/tikz/dotted` (style, 无默认值)

点线。

`/tikz/densely dotted` (style, 无默认值)

密集点线。

`/tikz/loosely dotted` (style, 无默认值)

稀疏点线。

`/tikz/dashed` (style, 无默认值)

短线构成的虚线。

`/tikz/densely dashed` (style, 无默认值)

密集虚线。

`/tikz/loosely dashed` (style, 无默认值)

稀疏虚线。

`/tikz/dash dot` (style, 无默认值)

短线-点虚线。

`/tikz/densely dash dot` (style, 无默认值)

密集短线-点虚线。

`/tikz/loosely dash dot` (style, 无默认值)

稀疏短线-点虚线。

`/tikz/dash dot dot` (style, 无默认值)

短线-点-点虚线。

`/tikz/densely dash dot dot` (style, 无默认值)

密集短线-点-点虚线。

`/tikz/loosely dash dot dot` (style, 无默认值)

稀疏短线-点-点虚线。

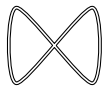
15.3.3 线条透明度

15.3.4 Double Lines and Bordered Lines

`/tikz/double`

`/tikz/double=<core color>` (默认值 white)

指示画线条，并且画双线，设置双线间的颜色。双线效果是这样实现的：例如，先沿着路径画一条黑色粗线，比如线宽是 10pt，然后沿着路径一条白色细线，比如线宽是 6pt，这样就得到双线效果；单条黑色线的线宽是 2pt，双线间距是 6pt，双线间的颜色是白色。



```
\tikz \draw[double] plot[smooth cycle]
coordinates{(0,0) (1,1) (1,0) (0,1)};
```

`/tikz/double distance=<dimension>` (无默认值, 初始值 0.6pt)

本选项会自动开启画双线功能，并设置双线间距，间距以双线的内侧边界为测量界限。

`/tikz/double distance between line centers=<dimension>`

本选项会自动开启画双线功能，并设置双线间距，间距以双线的中心为测量界限。

`/tikz/double equal sign distance` (style, 无值)

15.4 Adding Arrow Tips to a Path

15.5 Filling a Path

“填充路径”指的是将路径围起来的“内部”区域填色。这首先需要路径是封闭的，但如果路径是开的，程序先将其作成闭合的，然后填充。当路径比较复杂时，判断一个点是否属于应该填色的“内部”就不太容易。程序总是用 nonzero rule 或者 even odd rule 来判断填充区域，默认用 nonzero rule. 在不太复杂的情况下判断还算准确。

`/tikz/fill`

`/tikz/fill=<color>` (默认环境的颜色设置)

本选项指示用颜色填充路径。如果 fill=none，则取消填充。

如果 fill 和 draw 都是一个路径的选项（如 \filldraw 命令），则程序会先 fill 再 draw，这样填充色就不会掩盖线宽。



```
\begin{tikzpicture}[fill=yellow!80!black,line width=5pt]
\filldraw (0,0) -- (1,1) -- (2,1);
\filldraw (4,0) circle (.5cm) (4.5,0) circle (.5cm);
\filldraw[even odd rule] (6,0) circle (.5cm) (6.5,0) circle (.5cm);
\filldraw (8,0) -- (9,1) -- (10,0) circle (.5cm);
\end{tikzpicture}
```

上面第 1 个和第 4 个路径都是非封闭的，但程序把它们作成封闭的然后确定在那个区域填充颜色。

15.5.1 Fill Pattern

调用 patterns 程序库（见 §60）。

除了用颜色填充路径外，也可以用图样砖（tiling pattern）来填充。tiling pattern 分两种：不变色的（inherently colored patterns）和可变色的（form-only patterns）。pattern 缺少可变性，它不会因放缩、旋转变换而变化。

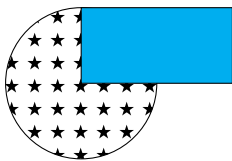
/tikz/pattern

/tikz/pattern=<name> （默认为环境设置的 pattern）

本选项指示用（环境设置的）pattern 样式填充路径。如果给出 pattern 样式名称 <name>，就用该样式填充路径。

如果设置 pattern=none，则取消 pattern 样式填充路径。

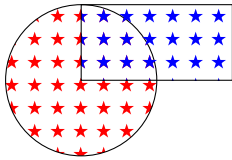
pattern 和 fill 的功能类似，当这两个选项并列时，后面的会抑制前面的。



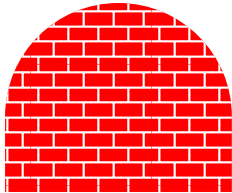
```
\begin{tikzpicture}
\draw[fill=red,pattern=fivepointed stars,]
(0,0) circle (1cm);
\draw[pattern=fivepointed stars,fill=cyan]
(0,0) rectangle (2,1);
\end{tikzpicture}
```

/tikz/pattern color=<color>

本选项用于可变色的 pattern，对不可变色的 pattern 无效。



```
\begin{tikzpicture}
\draw[pattern color=red,pattern=fivepointed stars]
(0,0) circle (1cm);
\draw[pattern color=blue,pattern=fivepointed stars]
(0,0) rectangle (2,1);
\end{tikzpicture}
```



```
\begin{tikzpicture}
\def\mypath{(0,0) -- +(0,1) arc (180:0:1.5cm) -- +(0,-1)}
\fill [red] \mypath;
\pattern[pattern color=white,pattern=bricks] \mypath;
\end{tikzpicture}
```

15.5.2 非零规则和奇偶规则

这两个规则是程序用来判断“一个点”是否属于该填充颜色的区域的，默认非零规则。

`/tikz/nonzero rule`

这是默认填色规则；为了确定点 A 是否属于被填充的区域，考虑从 A 出发的射线 \vec{l} 并从 0 开始计数。如果 \vec{l} 与区域的边界相交且边界线从射线 \vec{l} 的左侧穿到右侧，则计数加 1，否则计数减 1；若计数结果非 0，则点 A 属于该区域，否则不属于该区域。

`/tikz/even odd rule`

与上一规则做法类似，如果 \vec{l} 与区域的边界线相交的次数是奇数，则点 A 属于该区域，否则不属于该区域。

15.5.3 填充透明度

15.6 Using Arbitrary Pictures to Fill a Path

除了可以用颜色，`pattern` 填充路径外，还可以用自定义的路径，从外部图形插入图形来填充路径，这个效果当然可以用 `clip` 选项做到，也可以用下面的选项实现。

`/tikz/path picture=<code>`

当路径的 `fill`, `pattern`, `shade` 等具有填充作用的选项生效后，该选项再生效，开启一个局部分组，执行 `<code>`，插入相应内容来填充路径，但是插入内容受到被填充路径的剪切，然后结束分组，最后画出路径线条（如果有 `draw` 选项的话）。该选项用在路径之内才有效。

`<code>` 可以是 Tikz 的绘图命令，也可以是插入外部图形的命令。如果要插入外部图形，必须把 `graphicx` 宏包的 `\includegraphics` 命令放在 `\node` 命令中。

预定义 `node` 对象：`path picture bounding box`

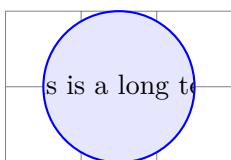
每个路径都有自己的 `bounding box`，在默认下这个盒子是恰好装下该路径的矩形盒子。每当创建一个路径时，程序都会根据该路径的结构自动生成一个名称为 `current path bounding box` 的矩形

node, 这个 node 的尺寸、位置与该路径的 bounding box 相同。作为一个 node 对象, current path bounding box 有自己的 node 坐标系 (参考 §13)。

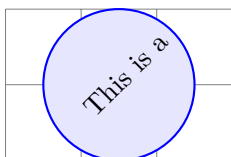
注意计算 current path bounding box 时程序不考虑路径的线宽, 也不把添加到路径上的 node 计算在内, 参考 §101.4。

当使用 path picture 选项时, 程序会自动创建一个名称为 path picture bounding box 的矩形 node, 在没有执行 `<code>` 时, 该 node 的形状、位置、尺寸与当前路径的 current path bounding box 相同; 在执行 `<code>` 时, `<code>` 会改变 current path bounding box 的尺寸、位置, 而 path picture bounding box 用在 `<code>` 中来引用当前的 current path bounding box, 所以实际上 path picture bounding box 与 current path bounding box 异名而同物。

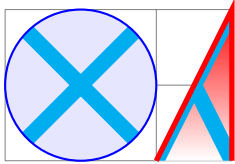
`<code>` 插入的内容处于一个分组内, 这个分组有自己的坐标系, 其坐标原点就是被填充路径的 path picture bounding box 的中心。



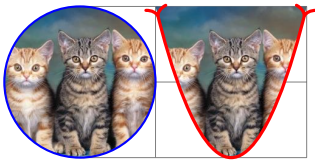
```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\filldraw [fill=blue!10,draw=blue,thick] (1.5,1) circle (1)
[path picture={
\node at (path picture bounding box.center) {
This is a long text.
};}
];
\end{tikzpicture}
```



```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\filldraw [fill=blue!10,draw=blue,thick] (1.5,1) circle (1)
[path picture={
\node [rotate=45,xshift=1cm]
at (path picture bounding box.center) {
This is a long text.
};}
];
\end{tikzpicture}
```



```
\begin{tikzpicture}[cross/.style={path picture={
  \draw[cyan,line width=6pt]
    (path picture bounding box.south east) --
    (path picture bounding box.north west)
    (path picture bounding box.south west) --
    (path picture bounding box.north east);
  }}]
\draw [help lines] (0,0) grid (3,2);
\filldraw [cross,fill=blue!10,draw=blue,thick]
  (1,1) circle (1);
\path [cross,top color=red,draw=red,line width=2pt]
  (2,0) -- (3,2) -- (3,0);
\end{tikzpicture}
```



```
\begin{tikzpicture}[path image/.style={
  path picture={
    \node at (path picture bounding box.center) {
      \includegraphics[height=3cm]{#1}
    };}]
\draw [help lines] (0,0) grid (3,2);
\draw [path image=cats,draw=blue,thick]
  (0,1) circle (1);
\draw [path image=cats,draw=red,very thick,>-<]
  (1,2) parabola[parabola height=-2cm] (3,2);
\end{tikzpicture}
```

15.7 Shading a Path

`/tikz/shade`

这个选项会给路径添加颜色渐变效果。程序会先变化颜色然后画出路径线条（如果有 `draw` 选项的话）。

如果同时使用 `shade` 和 `fill` 选项，则 `fill` 选项无效。

默认的渐变模式是上部灰色、下部白色。

`/tikz/shading=<name>`

这个选项设置颜色渐变的模式，具体参考 `Shadings Library`。

`/tikz/shading angle=<degrees>` (无默认值，初始值 0)

这个选项设置一个方向，使得颜色沿着这个方向渐变。

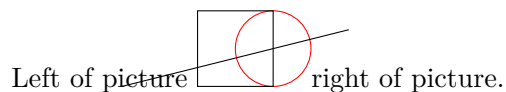
可以使用底层命令自定义颜色渐变模式。参考 §109.3。

15.8 Establishing a Bounding Box

\LaTeX 中有盒子命令 `\makebox[宽度][位置]{对象}`，通过设置宽度、位置能得到各种效果。图形的 bounding box 设置与之类似，只是稍微复杂。

`/tikz/use as bounding box`

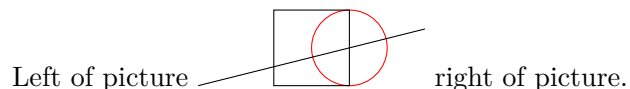
每个路径都有自己的 bounding box，即刚好装下该路径的矩形盒子。整个图形也有自己的 bounding box，在默认下这个盒子是刚好装下该图形的矩形盒子，不过这个盒子的尺寸是可以调整的。每当图形中创建了新路径后，程序会将图形的 bounding box 刷新（因为图形中有了新的元素，可能会改变图形尺寸）。也就是说，图形的 bounding box 是随着路径的创建而逐步刷新的。如果一个路径带有这个选项，那么将该路径用于刷新图形的 bounding box 后就停止刷新，即该路径之后创建的路径都不用于刷新图形的 bounding box。



```
Left of picture
\begin{tikzpicture}
  \draw [red] (3,0.5) circle (0.5cm);
  \draw[use as bounding box] (2,0) rectangle (3,1);
  \draw (1,0) -- (4,.75);
\end{tikzpicture}
right of picture.
```

上面例子中，图形的 bounding box 仅由前两个路径决定。

但是注意，如果这个选项处于 \TeX 分组内，其作用受到分组的限制（因此也会受到 `{scope}` 环境的限制，如果在 `{scope}` 环境内的话），比较下面的例子



```
Left of picture
\begin{tikzpicture}
  \draw [red] (3,0.5) circle (0.5cm);
  {\draw[use as bounding box] (2,0) rectangle (3,1);}
  \draw (1,0) -- (4,.75);
\end{tikzpicture}
right of picture.
```

`\useasboundingbox`

这个命令是 `\path[use as bounding box]` 的缩写，注意其中没有 `draw` 选项。



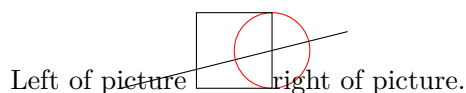
```

Left of picture
\begin{tikzpicture}
  \useasboundingbox (0,0) rectangle (3,1);
  \fill (.75,.25) circle (.5cm);
\end{tikzpicture}
right of picture.

```

`\pgfresetboundingbox`

这是个不带参数的命令，它指示重设图形的 bounding box，即该命令之前创建的路径都不用于构成图形的 bounding box。



```

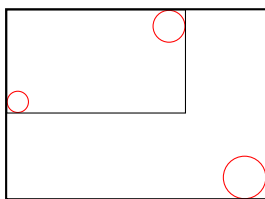
Left of picture
\begin{tikzpicture}
  \draw [red] (3,0.5) circle (0.5cm);
  \pgfresetboundingbox
  \draw[use as bounding box] (2,0) rectangle (3,1);
  \draw (1,0) -- (4,.75);
\end{tikzpicture}
right of picture.

```

上面例子中，命令 `\pgfresetboundingbox` 和选项 `[use as bounding box]` 配合使用，使得图形的 bounding box 仅由第二个路径决定。

预定义 node 对象: `current bounding box`

`current bounding box` 的形状、尺寸、位置与图形当前的 bounding box 一致。作为一个 node 对象，它有自己的 node 坐标系。注意，由于图形当前的 bounding box 随着路径的创建而不断刷新，故 `current bounding box` 也是不断变化的。



```

\begin{tikzpicture}
\draw[red] (0,0) circle (4pt);
\draw[red] (2,1) circle (6pt);
\draw (current bounding box.south west) rectangle
  (current bounding box.north east);
\draw[red] (3,-1) circle (8pt);
\draw[thick] (current bounding box.south west) rectangle
  (current bounding box.north east);
\end{tikzpicture}

```

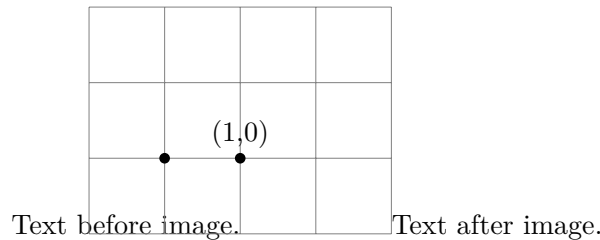
`/tikz/trim left=<dimension or coordinate or default>` (默认 0pt)

这个选项作为环境选项，设置整个图形的 bounding box 的左侧边界，默认为 0pt，即以坐标原点或说以直线 $x=0\text{pt}$ 来标示左侧边界。

如果设置 `trim left=<dimension>`，则以直线 $x=<dimension>$ 来标示左侧边界（这里 `<dimension>` 是带单位的尺寸）。

如果设置 `trim left={(a,b)}`，则以直线 $x=a$ 来标示左侧边界，注意这里要用花括号将坐标括起来。

`trim left=default` 用于重设左侧边界。



```
Text before image.%
\begin{tikzpicture}[trim left={(1,2)}]
  \draw [help lines](-1,-1) grid (3,2);
  \fill (0,0) circle (2pt)
    (1,0) node[above]{(1,0)} circle (2pt);
\end{tikzpicture}%
Text after image.
```

`/tikz/trim right=<dimension or coordinate or default>` (无默认)

这个选项作为环境选项，设置整个图形的 bounding box 的右侧边界。

`trim right=default` 用于重设右侧边界。

`baseline`, `trim left`, `trim right` 这些选项对 `external` 程序库的作用会有影响。

`/pgf/trim lowlevel=true|false` (无默认值, 初始值 `false`)

如果设置 `trim lowlevel=false`，则 `external` 程序库在输出图形时会输出完整的图形。如果设置 `trim lowlevel=true`，则 `external` 程序库在输出图形时只输出 `trim left` 之右，`trim right` 之左的部分。

15.9 Clipping and Fading

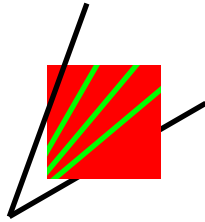
把一个矩形金属长条的两条对边接起来，做成环带，当作一个模子，拿一团橡皮泥压到模子里，掉出模子外的丢掉，留在模子内的取出，就把橡皮泥做成成了一个圆形。带有 `clip` 选项的路径就有这种模子的作用，姑且称之为“剪切路径”。

`/tikz/clip`

如果一个路径带有 `clip` 选项，该路径会对其后的路径进行剪切。如果该路径是自交的，程序会使用非零规则（默认）或奇偶规则来判断路径的内部和外部。

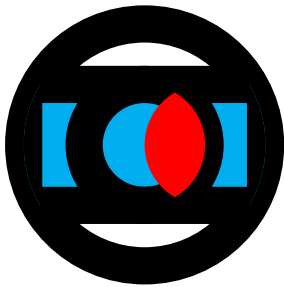
如果一个路径带有 `clip` 选项，则不能再有其它的选项，例如颜色、线宽等，这些选项可以在环境选项中列出。

`clip` 是图形状态参数，只在当前环境（例如 `{scope}` 环境）内有效，其效力持续至当前环境结束。



```
\begin{tikzpicture}[line width=2pt]
\draw (0,0) -- (30:3cm);
\begin{scope}[fill=red,draw=green]
  \fill[clip] (0.5,0.5) rectangle (2,2);
  \draw (0,0) -- (40:3cm);
  \draw (0,0) -- (50:3cm);
  \draw (0,0) -- (60:3cm);
\end{scope}
\draw (0,0) -- (70:3cm);
\end{tikzpicture}
```

如果一个环境内有多个剪切路径，则剪切效果会被累计。



```
\begin{tikzpicture}[line width=14pt,scale=0.8]
\draw [clip] (0,0) circle (2cm);
\draw [fill=cyan] (-2,-1) rectangle (2,1);
\draw[clip] (0,0) circle (1cm);
\fill[red] (1,0) circle (1cm);
\end{tikzpicture}
```

上面例子中线条粗细出现变化，是因为非零规则的作用，并且选项的作用次序依次是 `fill`, `draw`, `clip`.

`\clip`

这个命令是 `\path[clip]` 的缩写。注意用这个命令画路径时不能带有其它选项。

15.10 Doing Multiple Actions on a Path

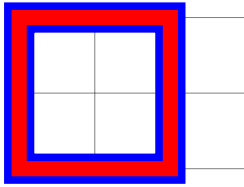
当画一个路径时，可能有多个选项对该路径起作用，这些选项可能来自环境选项设置，也可能来自路径本身的选项设置。这些选项的作用有一定次序，例如一般是 `fill`, `draw`, `clip` 依次作用。

下面的选项可以用于改变选项的作用次序。

`/tikz/preaction=<options>`

这个选项可以用于 `\path`, `\draw` 等路径命令，也可以作为 `node` 对象的选项，但如果用于 `{scope}` 环境的环境选项则无效。

当这个选项用作路径命令的选项时，程序首先分析、构建路径（构建指的是确定结构、坐标等，不等于画出），然后将该路径画两次，第一次使用 `preaction` 指定的 `<options>` 画出，第二次使用路径命令的其它选项画出。

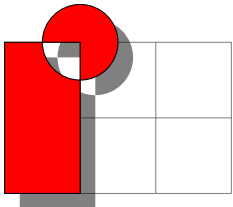


```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw
  [preaction={draw,line width=4mm,blue}]
  [line width=2mm,red]
  (0,0) rectangle (2,2);
\end{tikzpicture}
```

上面例子中，对同一个矩形，先以 4mm 线宽、蓝色画出，然后再以 2mm 线宽、红色画出。

注意，在第一次用 `<options>` 画出路径之前，路径就已被构建了，针对路径的坐标变换选项（如 `rotate`）在 `<options>` 中无效。第二次画路径时所使用的（由通常方式设置的）选项中的坐标变换选项对整个路径（两次画出的路径）有效。

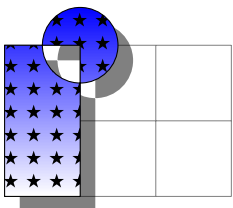
但 `preaction` 指定的 `<options>` 中可以有画布变换选项。



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw
  [preaction={fill=black,opacity=.5,
transform canvas={xshift=2mm,yshift=-2mm}}]
  [fill=red]
  (0,0) rectangle (1,2)
  (1,2) circle (5mm);
\end{tikzpicture}
```

用上面例子中的办法可以定义阴影样式（`shadow style`），更多阴影样式参考 `shadow library`。

可以多次使用 `preaction` 选项，程序会参照每个 `preaction` 选项的 `<options>` 将路径画一次，但是 `<options>` 的作用效果不会被累计。这样就可以自定义多个选项的作用次序。



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw [pattern=fivepointed stars]
  [preaction={fill=black,opacity=.5,
transform canvas={xshift=2mm,yshift=-2mm}}]
  [preaction={top color=blue,bottom color=white}]
  (0,0) rectangle (1,2)
  (1,2) circle (5mm);
\end{tikzpicture}
```

`/tikz/postaction=<options>`

与 `preaction` 类似，只是在用通常设置的路径选项画出路径后，再用 `<options>` 第二次画路径。

15.11 Decorating and Morphing a Path

参考 §24.

15.12 在起点或终点处截去一段路径

```
/tikz/shorten <=<dimension>
```

```
/tikz/shorten >=<dimension>
```

这两个选项通常配合箭头选项一起使用。

在 §99.3 中有命令 `\pgfsetshortenstart` 和 `\pgfsetshortenend`，这两个命令可以在路径的起点或终点处截掉一段路径。在文件 `tikz.code.tex` 中定义了两个 `tikz` 选项：

```
\tikzoption{shorten <}{\pgfsetshortenstart{#1}}
\tikzoption{shorten >}{\pgfsetshortenend{#1}}
```

使用选项 `shorten <=<dimension>` 可以在路径的起点处截掉一段长度为 `<dimension>` 的路径。使用选项 `shorten >=<dimension>` 可以在路径的终点处截掉一段长度为 `<dimension>` 的路径。



```
\begin{tikzpicture}
  \node [draw] (p) {x};
  \node [draw] (q) at(2,0) {x};
  \draw [green](p.north east)--(q.north west);
  \draw [red,shorten <=5mm](p.south east)--(q.south west);
\end{tikzpicture}
```

16 Arrows

16.1 Overview

首先调用 `arrows.meta` 程序库。

TikZ 允许在线条的端点处画一个或数个箭头，并可以设置每个箭头的方向、颜色等外观样式。有多种预定义的箭头，也可以自定义一种箭头，参考 §100.

本节所介绍的箭头特性都属于 TikZ 的 3.0 版本，旧版本的箭头没有这些特性（例如 `scale=2` 对旧版本箭头无效）。为了区别，在 \LaTeX 中，新版箭头的名称的首字母都用大写。

旧的程序库 `arrows` 和 `arrows.spaced` 已经被新程序库 `arrows.meta` 代替，但是如果你愿意，仍可调用旧的程序库来使用旧的箭头。

16.2 如何添加箭头

添加箭头时要注意以下事项：

1. 使用选项 `arrows` 或其简缩形式来为线条添加箭头。
2. 可以为选项 `tips` 赋值，来决定是否画出箭头（默认画出箭头）。
3. 箭头选项不能与 `clip` 选项同时用于同一个路径。

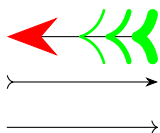
4. 被添加箭头的路径不能是“封闭的”，例如，不能给由 `circle`, `rectangle` 算子生成的路径加箭头；不能给使用 `cycle` 结尾的路径加箭头。

`/tikz/arrows=<start arrow specification>-<end arrow specification>`

这个选项给线条的始端或末端添加箭头，其中短线“-”代表路径。它有简缩形式，可以省略 `arrows=`，只写出等号右侧的部分，用小于号“<”和大于号“>”代表箭头，符号组合“->”，“<-”分别指示在路径的末端、始端加箭头并且规定了箭头方向；“<->”，“>-<”等符号的意思类似；“-Stealth”则规定路径末端采用 Stealth 样式的箭头。

在默认下，符号“>”代表的箭头样式名称是 Computer Modern Rightarrow，而“<”代表的箭头样式是 Computer Modern Rightarrow 的翻转。数学模式中的命令 `\to` 也默认 Computer Modern Rightarrow 样式。

实际上，表示箭头的符号“>”，“<”以及箭头样式名称“Stealth”都是“算子”，跟算子 `circle`, `rectangle` 一样，可以带有选项。

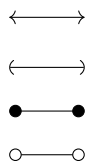


```
\begin{tikzpicture}[scale=2]
\draw[->] (0,0) -- (1,0);
\draw[>-Stealth] (0,0.3) -- (1,0.3);
\draw[Stealth[scale=4,red]]-<[scale=4,green,line width=1pt]
<[scale=4,green,line width=2pt]
<[scale=4,green,line width=4pt]]]
(0,0.6) -- (1,0.6);
\end{tikzpicture}
```

注意上面的例子表明，环境选项 `scale=2` 对箭头算子无效。还要注意，如果箭头算子带有选项，要把箭头算子及其选项用花括号括起来，如上例，这样 TikZ 就不会把属于 Stealth 的（用于界定选项列表的）方括号的结束符号“]”与命令 `\draw` 的相混淆。

箭头算子画出来的其实是个路径，如上例，Stealth 箭头是个四边形（有一个内凹的顶角）并且默认是填充颜色的，默认的 Computer Modern Rightarrow 箭头只画出路径线条，没有填充颜色的属性。

这几个符号：“<”，“>”，“（”，“）”，“*”，“o”都是箭头算子，它们画出的箭头形态如下：



```
\tikz\draw[<->] (0,0)--(1,0);
\tikz\draw[(-)] (0,0)--(1,0);
\tikz\draw[*-*] (0,0)--(1,0);
\tikz\draw[o-o] (0,0)--(1,0);
```

`/pgf/tips`

`/pgf/tips=<value>`

(默认值 `true`，初始值 `on draw`)

这个选项决定在什么情况下给路径添加箭头。其中的 `<value>` 可以是以下取值情况：

- true, 这是默认值, 即当只写出 `tips` 而不用等号 “=” 给它赋值时, 其值就默认为 true.
- proper
- on draw, 这是初始值, 即当不使用该选项时, 程序实际会以 `tips=on draw` 的设置工作。
- on proper draw
- never 或 false

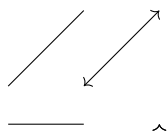
总结起来, 在以下情况下路径中不会出现箭头：

- 没有设置箭头算子, 例如只写出 `arrows=-` 就没有箭头。
- 使用了 `clip` 选项。
- 将 `tips` 选项的值设为 never 或 false.
- 将 `tips` 选项的值设为 on draw 或 on proper draw, 但是没有同时给出 draw 选项。
- 空路径 (不含任何坐标) 不会有箭头。
- 如果一个路径中存在闭合子路径 (例如, 由 circle, rectangle, cycle 所引起的路径是闭合的), 那么主路径上没有箭头。

当路径中的子路径都非闭合时, 才有可能为这个路径加箭头, 并且箭头只可能加在最后一个子路径上。

对于给“最后一个子路径”加箭头还有以下几种情况。

1. 如果该子路径是非退化的, 即至少包含两个不同坐标点, 箭头按正常方式添加。
2. 如果该子路径不包含任何坐标点, 则没有箭头。
3. 如果该子路径只包含一个坐标点, 那么仅当 `tips=true` 或 `tips=on draw` 时, 箭头添加到这个点上, 并且箭头指向上; 但如果 `tips=proper` 或 `tips=on proper draw`, 则没有箭头。



```
\tikz \draw [<->] (0,0)--(1,1) (1,0)--(2,1);
\tikz [<->] \draw (0,0) -- (1,0) (2,0);
```

箭头的初始指向都是指向右侧的, 箭头尖端的最末一个像素的位置是路径的端点。箭头算子不接受 `rotate` 选项。当箭头添加到路径上时, 程序会自动将箭头绕着箭头尖端旋转, 使得箭头的指示方向与子路径 (在端点附近的) 方向一致。程序在确定箭头旋转状态时要参考箭头轮廓线与子路径的公共点 (见 §16.3.8), 如果子路径长度太短造成公共点缺失就导致程序计算失败, 结果可能出人意料。所以被添加箭头的子路径的长度与箭头尺寸要匹配, 最好不要太短。

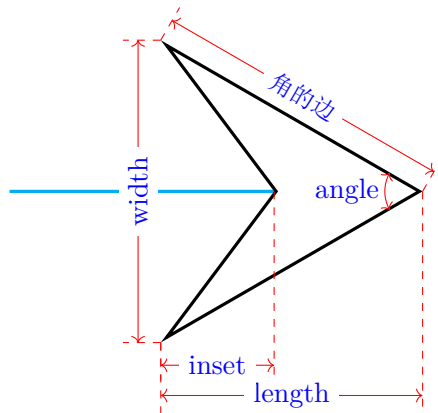
16.3 设置箭头的外观

对于标准的箭头, 例如 `Stealth`, `Latex`, `Bar`, 可以设置其尺寸、宽高之比、颜色等参数, 这与设置文字的字体有些类似。设置箭头外观是通过设置箭头算子的选项来实现的, 例如 `Stealth[length=4pt,width=2pt]`, 在前文已有例子。

16.3.1 箭头的“特征尺寸”

长、宽、高是一个立方体的“特征尺寸”, 这些“特征尺寸”可以确定一个立方体。一个箭头也有各种“特征尺寸”, 并且不同的箭头由于形态不同, 具有不同的“特征尺寸”。下面以比较典型的 `Stealth` 箭头为

例来说明各种“特征尺寸”。



首先注意，箭头尖端的最末一个像素的位置是路径的端点。

`/pgf/arrow keys/length=<dimension> <line width factor> <outer factor>`

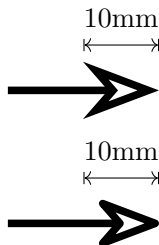
这个选项设置箭头长度，注意箭头长度指的是从箭头最左侧的“像素”到最右侧的“像素”的水平距离。这个选项的值有 3 个元素，这 3 元素之间用空格分隔。<dimension> 是带长度单位的尺寸，它规定箭头长度的基础尺寸，不能省略。<line width factor> 和 <outer factor> 是纯数值，可以省略，由于它们之间用空格分隔，所以此二者的省略次序是从后向前的，如果省略，就默认其值为 0。

如果给出 <line width factor>，那么箭头长度就是 $\text{<dimension>} + \text{<line width factor>} * w$ ，其中 w 是被添加箭头的路径的线宽，这就使得箭头长度与路径线宽具有相关性。例如 `length=0pt 5`，就保持箭头长度与路径线宽之比为 5:1。

<outer factor> 仅在路径是双线时有效。当路径是双线时，路径有 3 种线宽：外侧线宽 w_o ，内部线宽 w_i ，总线宽 w_t ，当然它们之间有关系 $w_t = w_i + 2w_o$ 。当路径是双线且给出 <outer factor> 时，前面的 w 就由等式 $w = \text{<outer factor>} w_o + (1 - \text{<outer factor>}) w_t$ 来确定。所以，如果 <outer factor> 是 0，则 $w = w_t$ 。

例如，Latex 箭头的默认长度设置是 `length=3pt 4.5 0.8`，当路径线宽是 1.2pt 且为双线，双线间距为 2pt 时，该选项决定的箭头长度是： $3\text{pt} + 4.5 * (0.8 * 1.2\text{pt} + (1 - 0.8) * 4.4\text{pt}) = 11.28\text{pt}$ 。

Stealth 箭头是个封闭的路径（四边形），可以对其顶角设置 `line join=<round or miter>` 选项，该选项的值 `round` 与 `miter` 会导致不同的箭头形态，但程序规定二者有相同的长度，观察下图，注意箭头尖端的最末一个像素的位置是路径的端点：



`\tikz{`

```

\draw [line width=1mm, -{Stealth[length=10mm, open]}]
      (0,0) -- (2,0);
\draw [|<->|] (2,.6) -- node[above=1mm] {10mm} ++(-10mm,0);
}

\tikz{
\draw [line width=1mm, -{Stealth[length=10mm, open, round]}]
      (0,0) -- (2,0);
\draw [|<->|] (2,.6) -- node[above=1mm] {10mm} ++(-10mm,0);
}

```

`/pgf/arrow keys/width=<dimension> <line width factor> <outer factor>`

这个选项的设置与前一个选项 `length` 类似。

`/pgf/arrow keys/width'=<dimension> <length factor> <line width factor>`

这个带撇号的选项的值的意义与前面的类似。这个选项的主要作用是将箭头的宽度与长度保持在某个比例上，当箭头长度变化时，箭头宽度随之变化。

注意当给出多个 `length` 值时，最后一个有效：

```
length = 10pt, width' =5pt 2, length=7pt 等价于 length = 7pt, width' =5pt 2
```

`/pgf/arrow keys/inset=<dimension> <line width factor> <outer factor>`

`inset` 尺寸指的是从箭头最左侧向箭头内部直到内凹顶点的长度，这也是路径插入箭头内部的尺寸（所以叫作 `inset`）。这个选项的值的意义与前面的类似。注意有的箭头，例如 Latex 箭头，因其本身的形状所限（它没有内凹的顶点），没有 `inset` 这个尺寸，因此这个选项对它无效。

`/pgf/arrow keys/inset'=<dimension> <length factor> <line width factor>`

这个选项与 `width'` 类似。例如 Stealth 箭头的默认设置是 `inset' =0pt 0.325`，将内凹深度与长度之比保持为 0.325:1。

`/pgf/arrow keys/angle=<angle>:<dimension> <line width factor> <outer factor>`

这个选项设置箭头尖角的角度与角的边长，其中 `<angle>` 设置角度，`<dimension> <line width factor>` `<outer factor>` 设置角的边长。在这个设置下，箭头的长度和宽度可以由“角度”和“角的边长”确定（用三角函数计算），但 `inset` 尺寸需要另外设置。

`/pgf/arrow keys/angle'=<angle>`

这个选项的值是角度。按三角函数关系，箭头的长度、宽度、角度这 3 个量，其中任意两个可以确定第 3 个，而 `inset` 尺寸需要另外设置。

16.3.2 箭头的比例缩放

`/pgf/arrows keys/scale=<factor>` （无默认值，初始值 1）

当前文所述的尺寸选项处理完毕后，TikZ 再处理这个 `scale` 选项，将箭头按比例 `<factor>` 做缩放，即

做位似变换，位似中心是路径端点。

注意 `scale` 选项对箭头的线宽、叠放箭头的间距（`sep` 选项的设置）无效。

`/pgf/arrows keys/scale length=<factor>` （无默认值，初始值 1）

这个选项类似 `scale`，只是只对箭头的 `length`，`inset` 起作用，对箭头的 `width` 无作用。

`/pgf/arrows keys/scale width=<factor>` （无默认值，初始值 1）

这个选项类似 `scale`，只是只对箭头的 `width` 起作用。

16.3.3 圆弧箭头

有的箭头由圆弧组成，且圆弧长度可调。

`/pgf/arrow keys/arc=<degrees>` （无默认值，初始值 180）

设置圆弧的角度。



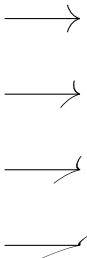
```
\tikz [ultra thick] {
  \draw [arrows = {-Hooks[scale=2}}] (0,2) -- (1,2);
  \draw [arrows = {-Hooks[arc=90,scale=2}}] (0,1) -- (1,1);
  \draw [arrows = {-Hooks[arc=270,scale=2}}] (0,0) -- (1,0);
}
```

16.3.4 倾斜

箭头可以像文字的 `italic` 形态那样倾斜。

`/pgf/arrow keys/slant=<factor>` （无默认值，初始值 0）

箭头的倾斜效果是使用“画布变换（`canvas transformation`）”得到的。当 `<factor>` 比较大时，如下面例子中 `slant=2`，箭头会倾斜得比较严重，但箭头的尖端（的最后一个像素）始终保持在路径的端点处。



```
\tikz {
  \draw [arrows = {->[scale=2}}] (0,3) -- (1,3);
  \draw [arrows = {->[slant=.5,scale=2}}] (0,2) -- (1,2);
  \draw [arrows = {->[slant=1,scale=2}}] (0,1) -- (1,1);
  \draw [arrows = {->[slant=2,scale=2}}] (0,0) -- (1,0);
}
```

16.3.5 Reversing, Halving, Swapping

`/pgf/arrow keys/reverse`

这个选项会使得箭头原地翻转，指向相反的方向。如果使用该选项两次，则会取消翻转。有的箭头翻转后，与路径的交接部分会有细微的变化。



```
\tikz [ultra thick]
  \draw [arrows = {-Stealth[reversed]}] (0,0) -- (1,0);

\tikz [ultra thick]
  \draw [arrows = {-Stealth[reversed, reversed]}] (0,0) -- (1,0);
```

/pgf/arrow keys/harpoon

这个选项只画出左半边的半个箭头，形似鱼叉。

/pgf/arrow keys/swap

这个选项与 harpoon 一起使用，只画出右半边的半个箭头。

/pgf/arrow keys/left

等效于 harpoon.

/pgf/arrow keys/right

等效于 harpoon, swap.



```
\tikz [ultra thick]
  \draw [arrows = {-Stealth[harpoon]}] (0,0) -- (1,0);

\tikz [ultra thick]
  \draw [arrows = {-Stealth[harpoon,swap]}] (0,0) -- (1,0);

\tikz [ultra thick]
  \draw [arrows = {-Stealth[right]}] (0,0) -- (1,0);
```

16.3.6 箭头颜色

/pgf/arrow keys/color=<color or empty> (无默认值，初始值 empty)

通常，箭头的颜色与箭头所在的路径的颜色一致。本选项可以为箭头单独设置颜色。color= 这一部分可以省略，只写出颜色，这与路径颜色的设置是一样的。

当本选项的设置是空（不做设置）时，程序会将箭头路径的颜色用于填充箭头内部，主路径中的填充色选项对箭头无效。



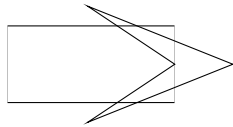
```
\tikz [ultra thick]
  \draw [draw=red, fill=cyan,
        arrows = {-Stealth[length=10pt]}]
        (0,0) -- (1,1) -- (2,0);
```

上面的例子中，主路径原来的末端点变成了箭头的尖端，主路径的末端点向后移动了一段距离——这是添加箭头的常见现象，即箭头会在路径端点处“裁剪”路径（参考 §100），因此当填充主路径时，填充区域不能包括箭头。所以尽量避免给填充路径加箭头。

/pgf/arrow keys/fill=<color or none>

这个选项指定箭头内部的填充色。fill=none 表示不填充颜色（注意这不同于填充白色）。

如果箭头有 inset 尺寸，那么若箭头不填充颜色，可能会产生问题，下面是个极端的例子



```
\tikz \draw [double distance=1cm,
arrows = {-Stealth[length=2cm,width=1.6cm,
inset=1.2cm,fill=none}}]
(0,0) -- (3,0);
```

上面例子中，路径是双线，由于 inset 过深且双线间距过大，造成双线突出箭头外部，比较意外。

/pgf/arrow keys/open

等效于 fill=none.

如果在箭头选项中同时使用 color 和 fill 选项，那么 fill 选项应该放在 color 选项之后，否则 fill 选项会被 color 选项重设。

如果当前主路径有填充颜色，那么这个颜色名称就会存放在 pgffillcolor 中，下面例子中，箭头填充色设置为 pgffillcolor，使得箭头内部颜色与主路径填充色一致。



```
\tikz [ultra thick] \draw [draw=red, fill=red!50,
arrows = {-Stealth[length=15pt, fill=pgffillcolor}}]
(0,0) -- (1,1) -- (2,0);
```

16.3.7 线型

本小节可以与 §15.3.1 的内容对照。

/pgf/arrow keys/line cap=<round or butt>

设置箭头路径端点的“帽子”样式，对于箭头来说只有“圆帽子”（round）和“光头”（butt，无帽子）两个选择。这个选项对箭头的各种尺寸没有影响，箭头最末端的点仍然是路径的端点。



```
\tikz [line width=2mm]
\draw [arrows = {-Bracket[reversed,line cap=butt}}]
(0,0) -- (1,0);

\tikz [line width=2mm]
\draw [arrows = {-Bracket[reversed,line cap=round}}]
(0,0) -- (1,0);
```

箭头是个路径，如果箭头不是闭合的路径，它的端点就有默认的“帽子”（参考 §15.3.1），箭头“帽子”都可以用这个选项来修改。

/pgf/arrow keys/line join=<round or miter>

设置箭头路径上的角（不包括路径端点）的外缘形态，对于箭头来说只有 round（圆形），miter（尖角形）两个选择。这个选项对箭头的各种尺寸没有影响，箭头最末端的点仍然是路径的端点。



```
\tikz [line width=2mm]
  \draw [arrows = {-Computer Modern Rightarrow[line join=miter}}]
    (0,0) -- (1,0);

\tikz [line width=2mm]
  \draw [arrows = {-Computer Modern Rightarrow[line join=round}}]
    (0,0) -- (1,0);
```

`/pgf/arrow keys/round`

这是 `line cap=round, line join=round` 的简缩。

`/pgf/arrow keys/sharp`

这是 `line cap=butt, line join=miter` 的简缩。

`/pgf/arrow keys/line width=<dimension> <line width factor> <outer factor>`

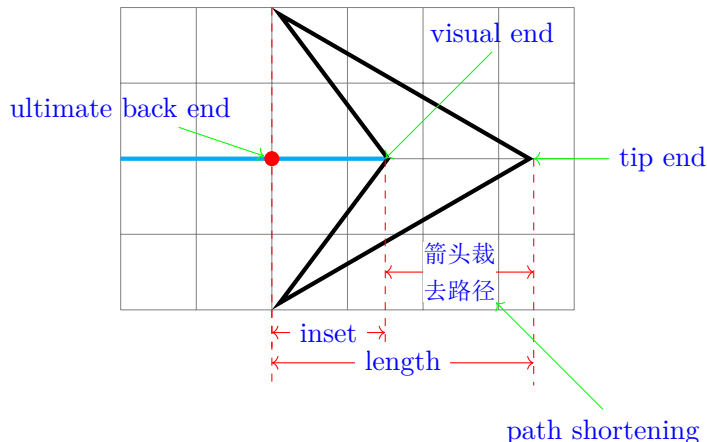
本选项设置箭头路径的线宽。选项值的意义与 `length` 同。如果设置本选项为 `0pt`，那么程序会只填充箭头，当箭头带有 `bend` 选项时，设置箭头线宽为 `0pt` 可能会比较好。

`/pgf/arrow keys/line width'=<dimension> <length factor>`

本选项设置箭头路径的线宽，选项值的意义与 `line width` 类似。

16.3.8 Bending and Flexing

首先规定几个名词，参照下图



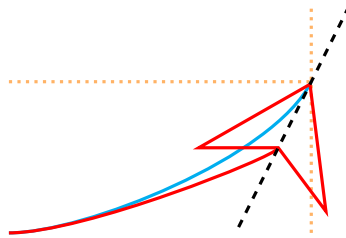
上面图中，首先画出网格，然后画出路径和箭头，箭头使用了 `open` 选项，然后再画出各个虚线和标示。箭头没有遮挡网格，却裁去了一部分路径，被裁去的路径长度是 `length - inset`，这种裁去一段路径的行为称为 `path shortening`，即“裁线”。箭头最前端的那个像素是路径的端点，称之为 `tip end`。路径与箭头轮廓线的公共点（图中是箭头内凹的顶点）称作 `visual end`。

一般情况下，箭头添加到路径上后，箭头的指向应当与箭头所处的路径端点处的切线方向一致。这对于直线段路径的情况比较容易实现，对于曲线路径就比较复杂。程序按照是否载入 `bending` 程序库来分别决定箭头指向。

当不载入 `bending` 程序库时，程序库会默认使用下一个选项决定箭头指向：

`/pgf/arrow keys/quick`

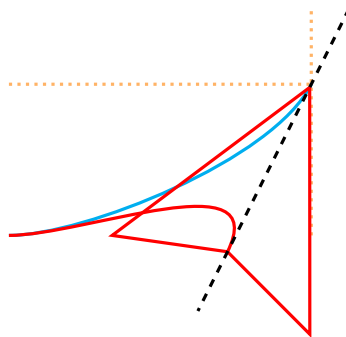
下面以一段控制曲线为例来说明这个选项的作用。为了明显，箭头的尺寸设置得比较大。



```
\begin{tikzpicture}[line width=1.2pt]
\draw [dotted,orange!60] (2,-2) -- (2,1) (-2,0) -- (2,0);
\draw [cyan] (-2,-2) .. controls (-1,-2) and (1.5,-1) .. (2,0);
\draw [red,-{Stealth[length=1.5cm,width=2cm,inset=0.5cm,open,quick]}]
(-2,-2) .. controls (-1,-2) and (1.5,-1) .. (2,0);
\draw [dashed] ($(2,0)!-1!(1.5,-1)$)--($(2,0)!2!(1.5,-1)$);
\end{tikzpicture}
```

上面例子中，红色控制曲线的端点处有个不填充颜色的 `Stealth` 箭头，箭头使用了 `quick` 选项。青色和红色的控制曲线的路径坐标是完全一样的，青色曲线是原本的形态。红色箭头的指向与青色曲线在点 `tip end` 处的切向是一致的（注意 `tip end` 总是控制曲线的端点），也就是控制曲线的终点与第 2 控制点所决定的方向（图中黑色虚线的方向）。但由于箭头的这种设置使得红色曲线偏离了原来的形态，这是 `quick` 选项的效果。

将上面例子中箭头的尺寸修改一下，得到下面比较极端的例子

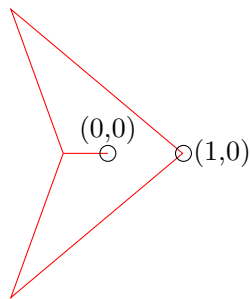


```
\begin{tikzpicture}[line width=1.2pt]
\draw [dotted,orange!60] (2,-2) -- (2,1) (-2,0) -- (2,0);
\draw [cyan] (-2,-2) .. controls (-1,-2) and (1.5,-1) .. (2,0);
\draw [red,-{Stealth[length=3cm,width=3cm,inset=0.5cm,open,quick]}]
(-2,-2) .. controls (-1,-2) and (1.5,-1) .. (2,0);
\end{tikzpicture}
```

```
\draw [dashed] ($(2,0)!-1!(1.5,-1)$)--($(2,0)!3!(1.5,-1)$);
\end{tikzpicture}
```

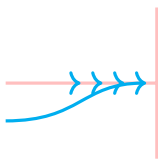
可见，由于箭头尺寸过大，造成很意外的结果。

注意，如果不载入 bending 程序库，程序默认 quick 选项起作用（即使箭头没有 quick 选项，程序也会按 quick 选项的效果画出箭头）。因此，箭头的尺寸应当与路径的长度相称。下面的例子中，线段长度是 1cm，明显小于箭头尺寸，意外地出现了一段线段：



```
\begin{center}
\begin{tikzpicture}
\draw [red,-{Stealth[angle=80:3cm,inset=0.7cm,open,quick]}]
(0,0)--(1,0);
\draw (0,0) circle(3pt) node [above] {(0,0)};
\draw (1,0) circle(3pt) node [right] {(1,0)};
\end{tikzpicture}
\end{center}
```

另外，当串联多个箭头时，quick 选项还导致下面图形中，箭头不在路径上的问题：

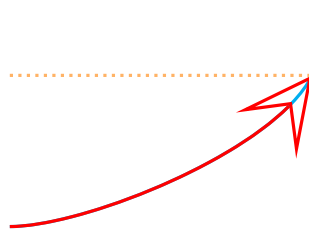


```
\begin{tikzpicture}[line width=1.2pt]
\draw [red!25,] (-1,0) -- (1,0) (1,-1)--(1,1);
\draw [cyan,-{[quick,sep=4pt]>>>>}]
(-1,-.5) .. controls (0,-.5) and (0,0) .. (1,0);
\end{tikzpicture}
```

当载入 bending 程序库后，程序库提供 flex, flex', bend 选项来调节箭头指向，其中的 flex 是默认选项（除非指定其它选项）。

`/pgf/arrow keys/flex=<factor>` （默认值 1）

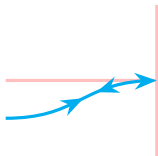
这个选项会使得箭头绕点 tip end 旋转，如果 flex=0，则箭头的指向与路径端点处的切线方向一致。如果 <factor> 是大于 0 的数值，则箭头顺时针旋转。如果 <factor> 是小于 0 的数值，则先把箭头旋转 180 度，即把箭头变成它关于点 tip end 的中心对称图形，再将它逆时针旋转。



```
\begin{tikzpicture}[line width=1.2pt]
\draw [dotted,orange!60] (2,-2) -- (2,1) (-2,0) -- (2,0);
\draw [cyan] (-2,-2) .. controls (-1,-2) and (1.5,-1) .. (2,0);
\draw [red,-{Stealth[length=1cm,width=1cm,inset=0.5cm,open]}]
(-2,-2) .. controls (-1,-2) and (1.5,-1) .. (2,0);\end{tikzpicture}
```

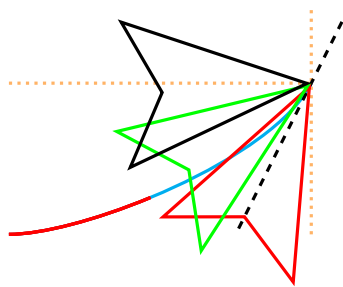
上面的例子是 `flex` 选项的默认效果，在箭头尺寸适当的情况下，箭头绕点 `tip end` 旋转，使得 `visual end` 点恰好位于路径（控制曲线）上。

下面例子是 `flex` 取负值的情况。



```
\begin{tikzpicture}[line width=1.2pt,>=Stealth]
\draw [red!25,] (-1,0) -- (1,0) (1,-1)--(1,1);
\draw [cyan,-{>>[flex=-2,sep=20pt]>}]
(-1,-.5) .. controls (0,-0.5) and (0,0) .. (1,0);
\end{tikzpicture}
```

如果箭头尺寸相对于路径长度过大，就会出现这个问题。



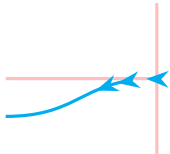
```
\begin{tikzpicture}[line width=1.2pt]
\draw [dotted,orange!60] (2,-2) -- (2,1) (-2,0) -- (2,0);
\draw [cyan] (-2,-2) .. controls (-1,-2) and (1.5,-1) .. (2,0);
\draw [dashed] ($(2,0)!-1!(1.5,-1)$)--($(2,0)!2!(1.5,-1)$);

\foreach \n / \c in {0/red,1/green,3/black}
{\draw [red,-{Stealth[length=2.5cm,width=2cm,inset=0.5cm,open,flex=\n,color=\c]}]
(-2,-2) .. controls (-1,-2) and (1.5,-1) .. (2,0);}
\end{tikzpicture}
```

在上面例子中，由于箭头尺寸过大，造成路径不能连接到 `visual end` 点（箭头内凹的顶点），使得路径

与箭头之间有空白。而且当 `flex=1` 时, `visual end` 点也不位于路径(控制曲线)上。

如果在路径上串联多个箭头, 将选项 `flex` 置于所有箭头符号之前, 则该选项会对每个箭头起作用。



```
\begin{tikzpicture}[line width=1.2pt,>=Stealth]
\draw [red!25,] (-1,0) -- (1,0) (1,-1)--(1,1);
\draw [cyan,-{flex=-2,sep=4pt}>>>]
(-1,-.5) .. controls (0,-0.5) and (0,0) .. (1,0);
\end{tikzpicture}
```

`/pgf/arrow keys/flex' =<factor>` (默认值 1)

这个选项与 `flex` 类似, 只不过当 `<factor>` 等于 1 时, 选项 `flex` 使得 `visual end` 点位于路径上, 而选项 `flex'` 使得箭头最后部的中间点“ultimate back end”位于路径上。

`flex=0` 与 `flex' =0` 等效。

如果一个箭头没有 `inset` 尺寸, 从而没有内凹的 `visual end` 点(例如 `Computer Modern Rightarrow` 箭头), 那么就适合使用 `flex'` 选项来设置箭头的指向。

`/pgf/arrow keys/bend`

这个选项会使箭头轮廓随着路径的弯曲而弯曲, 且程序会自动把箭头的指向与路径的切向符合起来。

在箭头的各个尺寸都适当的情况下, 弯曲箭头比较美观, 但如果箭头尺寸太大可能会走样。



```
\tikz \draw [red,line width=1mm,-{Stealth[bend,round,length=20pt]]
(0,-.5) .. controls (1,-.5) and (0.25,0) .. (1,0);

\tikz \draw [red,line width=1mm,-{Stealth[bend,length=40pt,
inset=20pt,width=15pt]]
(0,-.5) .. controls (1,-.5) and (0.25,0) .. (1,0);
```

16.4 Arrow Tip Specifications

16.4.1 句法

作为针对箭头的选项, 指定箭头样式的句法是

`<start specification>-<end specification>`

即“< 路径起点箭头样式 >-< 路径终点箭头样式 >”, 下面主要叙述路径终点箭头样式的设置。

路径端点的箭头样式设置的句法是

```
[<options for all tips>] <first arrow tip spec> <second arrow tip spec>
<third arrow tip spec> . . .
```

开头的方括号里的选项对后续的各个箭头都有效。方括号后的箭头又可以分为 3 种形式:

1. `<arrow tip kind name>[<options>]`, 例如 `Stealth[]`, 当某个箭头名称后面还有其它箭头符号、箭头名称等时, 要注意保留该箭头名称后的方括号(即使里面没有选项)。诸如“`Stealth >`”, “`Stealth`

Latex”这种句式都是错误的，因为程序会把“Stealth >”看作箭头名称，但没有名称为“Stealth >”的箭头。如果箭头名称后面有方括号，程序把方括号作为箭头名称结束的标志，就不会出现这种无法判断名称的问题。

2. `<shorthand>[<options>]`，这里的 `<shorthand>` 是用 `.tip=` 定义的 key，与上一种形式类似，也要在适当的位置用方括号来隔开箭头。
3. `<single char shorthand>[<options>]`，这里的 `<single char shorthand>` 可以是“<”，“>”，“（”，“）”，“*”，“o”，“x”这几个符号（但不能是括号或短线），它们分别代表一种箭头。与前两种形式不同，这种符号箭头与后续的箭头之间并非必须用方括号来分隔。

程序对以上形式的箭头句式做如下处理，举例来说，假设把 `abc[]` 作为箭头句式写入文档，当程序处理到这里时，首先，程序会检查从当前位置到下一个开方括号“[”之前的内容，即 `abc`，看看这个内容是否是（属于以上形式的）`<arrow tip kind name>` 或 `<shorthand>`；如果是，执行之，否则，检查该内容的第一个符号，即检查 `a`，看看这个符号是否是 `<shorthand>` 或 `<single char shorthand>`；如果是，执行之，然后按同样的规则检查剩余的内容，即检查 `bc`；否则，给出错误信息。

总结起来，可以用下面的例子来说明箭头句式是否正确：

- `->>>`，正确，这是在路径末端加 3 个箭头。
- `->[]>>`，正确，等效于上一个句式。
- `-[]>>>`，正确，等效于上一个句式。
- `->[]>[]>[]`，正确，等效于上一个句式。
- `->Stealth`，正确。
- `-Stealth >`，错误，因为 `Stealth >` 和符号 `S` 都不代表箭头。
- `-Stealth[]>`，正确。
- `<Stealth-`，正确，它是 `-Stealth[]>` 的翻转形式。
- `-Stealth[length=5pt] Stealth[length=6pt]`，正确，箭头名称后的方括号设置该箭头的长度。

在如下的箭头选项句式中

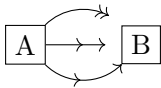
```
[<ops for all>] {<arrow1>[<ops1>] <arrow2>[<ops2>] <arrow3>[<ops3>] . . .}
```

如果某个方括号里有 `flex`, `flex'`, `bend` 这 3 个选项之一，那么所有方括号里的 `quick` 选项都当作 `flex` 来处理。另外注意，如果某个箭头算子后面有方括号，还需要用花括号将所有箭头算子及其方括号选项括起来。

16.4.2 Specifying Paddings

`/pgf/arrow keys/sep=<dimension> <line width factor> <outer factor>` （默认值 `0.88pt .3 1`）

这个选项可以用于所有箭头（放在箭头序列之前的方括号里），也可以只用作某一个箭头的选项，它在箭头的后面插入一个间隔，这样箭头之间或箭头与路径端点之间就不是一个紧贴一个的，而是适当疏松的，会显得美观。注意这个选项造成的箭头之间的间隔不是空白（箭头之间仍然有路径线条相连），但造成的箭头与路径端点之间的间隔却是空白。



```
\tikz {
\node [draw] (A) {A};
\node [draw] (B) [right=of A] {B};
\draw [->>[sep=6pt]] (A) to [bend left=45] (B);
\draw [->[sep=15pt]>] (A) to [bend right=45] (B);
\draw [-{[sep=6pt]>>}] (A) to (B);
}
```

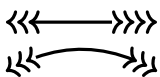
在箭头选项的语句中，可以使用下标线“_”，它是一个特殊的箭头算子，它不画出任何东西，代表的长度为 0，但程序默认它带有一个 sep 选项，所以它的作用就是插入一个 sep 间距。它是一个 <single char shorthand>，所以它与其后的箭头之间不必用方括号来间隔。

→→→ `\tikz \draw [->_____>] (0,0) -- (1,0);`

显然，用“_”插入间距不如用 sep 选项来得快捷。

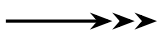
16.4.3 Specifying the Line End

有个 <single char shorthand> 符号“.”，即点号，将它用在箭头语句中时，它与路径端点之间的那部分路径会被隐藏，形成间断路径效果。



```
\tikz [very thick] \draw [<<.<-.>>.>>] (0,0) -- (2,0);
\tikz [very thick] \draw [<<<.<-.>>] (0,0) to [bend left] (2,0);
```

注意，如果将符号“.”放在某个非 <single char shorthand> 箭头符号之后，还需要用花括号将所有箭头算子及其方括号选项连同符号“.”都括起来。



```
\tikz [very thick] \draw [-{Stealth[] . Stealth[] Stealth[]}]
(0,0) -- (2,0);
```

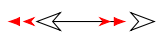
符号“.”可以带有选项，但所带选项都无效。

16.4.4 Defining Shorthands

有时设置路径端点的箭头样式的代码可能比较复杂，而这个样式又要在多个地方使用，如果每次使用这个样式都写出那么多代码就显得过于繁琐。此时，可以把这些样式设置存储在一个 key 中，用这个 key 代替复杂的代码。

<key>/.<tip>=<end specification>

这种设置是针对路径末端的箭头的，当在路径始端使用这个 key 时，箭头以及相关样式会自动翻转。



```
\tikz [foo /.tip = {Stealth[sep] Latex[sep]},
bar /.tip = {Stealth[length=10pt,open]}]
\draw [{foo[red] . bar}-{foo[red] . bar}] (0,0) -- (2,0);
```


上面例子中, `foo[red]` 的效果等效于 `Stealth[sep,red] Latex[sep,red]`, 也就是说当 `foo[red]` 展开为箭头设置时, 颜色 `red` 对箭头算子 `Stealth`, `Latex` 遵从“分配律”。

这种定义的 key 可以套嵌使用, 即某个 key 可以用于其它 key 的定义的中, 例如修改一下上面的例子



```
\tikz [foo /.tip = {Stealth[sep] Latex[sep]}],
      bar /.tip = {foo[length=10pt,open]}]
\draw [bar-bar] (0,0) -- (3,0);
```

符号“<”, “>”设置箭头同时还指定箭头方向, “(”与“)”的作用类似。但是箭头名称设置的箭头的指向, 在默认下, 始端与末端的箭头指向相反。



```
\tikz \draw [>->] (0,0) -- (1,0);

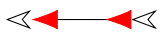
\tikz \draw [Stealth-Stealth] (0,0) -- (1,0);
```

如果调用了 `arrows.meta` 程序库, 那么默认“>”是 `To` (箭头名称, 等效于 `Computer Modern Rightarrow`) 的简缩。如果不载入 `arrow.meta` 程序库, `To` 相当于 `to` (旧程序库中的箭头名称)。

可以对“<”, “>”重新“赋值”, 来指定路径始端和末端的箭头样式, 这用到类似下面的句法

```
<-> /.tip = {<end arrow specification>}
>-> /.tip = {<end arrow specification>}
>-< /.tip = {<end arrow specification>}
<-< /.tip = {<end arrow specification>}
```

注意这是路径末端的 `<end arrow specification>`。



```
\tikz [>-> /.tip = {Latex[length=10pt,red].Stealth[open,scale=2]}]
\draw [>-<] (0,0) -- (2,0);
```

在上面例子中, 用“>”设置了始端和末端的箭头样式, 在画路径时, 路径末端的箭头符号用“<”, 指示各个 (将“>”设置中的) 箭头原地反向; 路径始端的箭头用“>”, 结果 (将“>”设置中的) 箭头整体翻转 (镜像)。注意箭头样式设置中使用了间断符号“.”, 由于 `Latex` 箭头后部没有内凹, 所以间断符号“.”的效果只在路径始端显示出来。

```
/tikz/>=<end arrow specification>
```

这是 `<->/>.tip=<end arrow specification>` 的简缩。

```
/tikz/shorten <=<dimension>
```

```
/tikz/shorten >=<dimension>
```

这两个选项可以在路径的起点或终点处截掉一段路径, 参考 §15.12, 或文件 `tikz.code.tex`。

16.4.5 Scoping of Arrow Keys

```
/tikz/arrows=[<start arrow specification>-<end arrow specification>]
```

这个 `arrows` 选项为当前环境中的箭头设置样式。

下面总结一下各种箭头选项的作用次序：

1. 首先是选项的默认值。
2. `arrows=[<keys>]` 中的选项设置。
3. 由 `/.tip={}` 定义的样式。
4. 箭头序列前面方括号里的选项设置。
5. 单个箭头后面方括号里的选项设置。

后作用的选项对前作用的选项有“优先权”。

16.5 Reference: Arrow Tips

调用程序库 `arrows.meta`，这个程序库定义了很多“标准箭头”，它们具有多种属性，可以比较细致地改变其外观。

唯一不默认自动加载这个程序库的原因是为了与旧版的 TikZ 兼容。当然，`arrows.meta` 可以与旧的 `arrows`，`arrows.spaced` 程序库同时加载使用。

`arrows.meta` 定义的箭头可以分为以下几类：

- 钩形箭头 (barbed arrow tips)，其外形为开路径，没有填充特性，`fill` 选项对其无效。

在默认下，若路径线宽为 0.4pt，则这种箭头的宽度是 (11pt 的 Computer Modern 字体中的) 字母“x”的高度。

钩形箭头	0.4pt	0.8pt	1.6pt
Arc Barb			
Bar			
Bracket			
Hooks			
Parenthesis			
Straight Barb			
Tee Barb			

- 数学箭头 (Mathematical arrow tips)，这种箭头属于钩形箭头，但把它们单独列出，其外形与 TeX 的数学模式种的 `\to` 的外形一样。

数学箭头	0.4pt	0.8pt	1.6pt
Classical TikZ Rightarrow			
Computer Modern Rightarrow			
Implies 用于双线路径			
To			

- 几何箭头 (Geometric arrow tips)，是由几何图形做的箭头，并且其内部是默认填充的，可以用 `open` 选项取消填充。

数学箭头	0.4pt	0.8pt	1.6pt
Circle			
Diamond			
Ellipse			
Kite			
Latex			
Rectangle			
Square			
Stealth			
Triangle			
Turned Square			

- 帽子箭头 (Cap arrow tips), 路径的端点有 3 种基本的帽子: round, rectangular, butt (见 §15.3.1)。除了这 3 种帽子, 还可以使用以下箭头帽子。

帽子箭头	1ex	1em
Butt Cap		
Fast Round		
Fast Triangle		
Round Cap		
Triangle Cap		

open 选项对帽子箭头无效。

- 射线箭头 (Rays), 其名称是 Rays, 其形态是由一点发出的数条线段, 象征放射状射线。默认它有 4 条射线。它可以带有选项 `n=<number>` 来设置射线数目。

`/pgf/arrow keys/n=<number>` (无默认值, 初始值 4)

这个选项设置 Rays 箭头的射线数目, `<number>` 应当是正整数。

射线箭头	0.4pt	0.8pt	1.6pt
Rays			
Rays [n=8]			
Rays [n=5]			

关于以上各种箭头的详细讲解参考 §16.5.1, §16.5.2, §16.5.3, §16.5.4, §16.5.5.

17 Nodes and Edges

17.1 Overview

通常, 一个 node 就是一个矩形或圆形或其它形状, 内部可以有文字、绘图命令、插入的图形等内容。node 是添加到路径上的, 但 node 不属于被添加的路径, 因此, 对该路径的某些操作选项, 例如 `scale`, `rotate`, `draw`, `fill` 等一般情况下对路径上的 node 无效。

17.2 Nodes and Their Shapes

17.2.1 Nodes 命令的句法

```
\path . . . node<foreach statements>[<options>](<name>)at(<coordinate>){<node contents>}
. . . ;
```

语句中各部分的次序 语句中的 `node` 是个算子，`{<node contents>}` 是 `node` 的内容。程序会检测 “`node`” 与 “`{`”，并将二者之间的内容看作是属于 `node` 算子的（处于该 `node` 的内部的）、对 `node` 的设置，设置包括 4 个部分，这 4 个部分都是可选的（可有可无，视情况而定）。如果有 `<foreach statements>`，则这个部分必须紧跟在 `node` 之后。

另外的 3 个部分：“`[<options>]`”，“`(<name>)`”，“`at(<coordinate>)`”，它们的次序是随意的。如果给出两个 “`(<name>)`”（例如 `(n1) (n2)`），则后一个（`(n2)`）有效；如果给出两个 “`at(<coordinate>)`”，则后一个坐标有效。但如果给出多个 “`[<options>]`”，它们的作用会累计。

node 的内容 由于程序会检测 “`{`”，且括号必须成对出现，所以 `{<node contents>}` 不能省略，而 `<node contents>` 可以空置。`<node contents>` 中可以有多种内容，甚至可以使用脆弱命令（例如 `\verb`）或者改变符号类别。但是如果在 “`[<options>]`” 中使用下一个选项，则应去掉 `{<node contents>}`：

```
/tikz/node contents=<node contents>
```

这个 key 用于 “`[<options>]`” 中，设置 `node` 的内容。如果 “`[<options>]`” 中使用了这个 key，那么程序会以符号 “`]`” 为标志，结束对 `node` 的解析，因此程序会认为符号 “`]`” 之后的 “`[<options>]`” 或者 `{<node contents>}` 不属于该 `node` 算子。这个 key 的 `<node contents>` 中不能包含脆弱命令。

node 的位置 `at(<coordinate>)` 指定 `node` 的位置，如果没有这一部分，就默认 `node` 之前的坐标点是 `node` 的位置。一个位置（坐标点）后可以跟随多个 `node` 语句。



```
\tikz \draw (0,0) |- (2,1)
      node[above]{a}
      node[right]{b}
      node[below]{c};
```

另外可以在 “`[<options>]`” 中用下面的 key 设置 `node` 的位置：

```
/tikz/at=<coordinate>
```

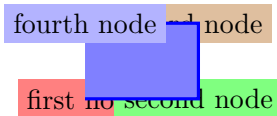
注意在 “`[<options>]`” 中用这个选项时，要把 `<coordinate>` 用花括号括起来，例如 `at={(1,1)}`，否则 `<coordinate>` 中的逗号会引起 TeX 错误。

一般情况下，后画出的 `node` 会遮挡之前画出的 `node`（如果二者有重叠的话）；而 `node` 在画出路径之后再画出，所以如果 `node` 与路径有重叠，则 `node` 会遮挡路径。可以用下面的选项确定 `node` 先于或后于路径画出，从而确定当 `node` 与路径有重叠时谁遮挡谁：

```
/tikz/behind path
```

这个 key 用于 “[<options>]” 中，程序先会画出 node 再画出路径，若 node 与路径有重叠，会得到路径遮挡 node 的效果。这个 key 只能用于当前路径中，不能用作环境选项。

如果路径上的多个 node 都有这个选项，那么它们依照先后次序，后面的遮挡前面的。



```
\tikz \fill [fill=blue!50, draw=blue, very thick]
(0,0)
  node [behind path, fill=red!50] {first node}
-- (1.5,0)
  node [behind path, fill=green!50] {second node}
-- (1.5,1)
  node [behind path, fill=brown!50] {third node}
-- (0,1)
  node [fill=blue!30] {fourth node};
```

/tikz/in front of path

这个 key 的效果与 behind path 的效果相反，即它使得 node 遮挡路径，这是程序默认的行为。

node 的名称 (<name>) 给 node 命名，这一种隐式命名方法。可以在 “[<options>]” 中用下面的选项显示地命名：

/tikz/name=<node name>

用这个 key 设置的是 “high-level” 名称（驱动程序不会去 “识别” 它），所以 <node name> 的构成比较随意，其中可以含有空格、数字、字母、汉字、下划线等等，但不能包含逗号、点句号、冒号等标点，因为逗号用于分隔坐标数据、分隔选项，点句号用于引用 node 坐标，冒号用于指定极坐标点。

/tikz/alias=<another node name>

给 node 另外起一个名字，这个 key 可以多次使用，从而给 node 起多个名字，这些名字都可以用来引用 node 坐标系中的坐标。

node 的选项 node 内部的选项设置 “[<options>]” 只对该 node 有效。

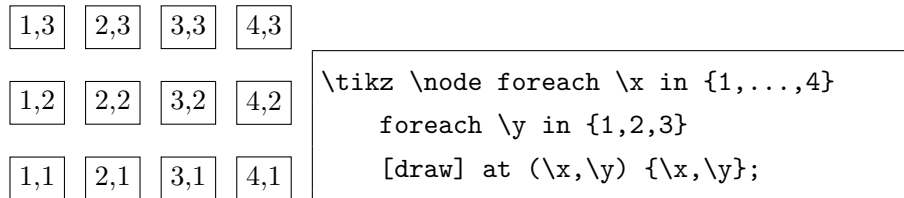
node 的形状 node 的形状指的是它的边界路径的形状，默认为矩形。预定义的形状有 rectangle, circle, coordinate. 调用与 shapes 有关的程序库（例如 shapes.geometric, shapes.symbols, shapes.callouts, shapes.misc, shapes.arrows 等）后，可以使用程序库提供的更多形状。

下面的选项用于选择 node 的形状：

/tikz/shape=<shape name> (无默认值，初始值 rectangle)

这个选项可以用作路径、环境选项、花括号分组内的 \tikzset 的选项，对分组内的所有 node 有效。可以只写出 <shape name>，省略 “shape=”。

node 中的 foreach 语句 注意 node 之后的 foreach 不带反斜线 “\”。node 之后可以使用多重 foreach 语句。foreach 语句的用法参考 §83。



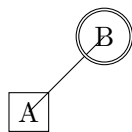
node 的样式 下面的 key 可以用作路径、环境选项、花括号分组内的 `\tikzset` 的选项，设置分组内的 node 的样式，或某一形状的 node 的样式。

`/tikz/every node` (style, 初始值 empty)

在有效范围内设置所有 node 的样式。

`/tikz/every <shape> node` (style, 初始值 empty)

在有效范围内设置所有形状为 <shape> 的 node 的样式。



```
\begin{tikzpicture}
  [every rectangle node/.style={draw},
  every circle node/.style={draw,double}]
  \draw (0,0) node[rectangle] {A}
    -- (1,1) node[circle] {B};
\end{tikzpicture}
```

node 的名称的前缀和后缀 可以给 node 的名称加前缀和后缀

`/tikz/name prefix=<text>` (无默认值, 初始值 empty)

这个选项用作当前分组的选项，给分组内的所有 node 的名称加前缀 <text> . 在该环境内部引用（该环境内的）node 名称时可以不带前缀，但在环境外部引用（该环境内的）node 名称时要加前缀。

`/tikz/name suffix=<text>` (无默认值, 初始值 empty)

这个选项用与 name prefix 类似，它给当前分组内的所有 node 的名称加后缀 <text> .

17.2.2 预定义的形状

如前述，预定义的 node 形状有 rectangle, circle, coordinate, 下面解释一下 coordinate 形状。

`\node` 是 `\path node` 的简缩

`\coordinate` 是 `\path coordinate` 的简缩

`\node[shape=coordinate][<options>](<name>)at(<coordinate>){};`

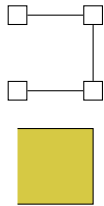
等效于

`\path . . . coordinate[<options>](<name>)at(<coordinate>) . . . ;`

所以命令 `\coordinate` , 算子 `coordinate` , 都创建一个 node.

通常，当用线条连接两个 node 时，线条的起点和终点都位于 node 的边界上，而不是位于 node 的中心，这好像是 node 对线条的“裁切”，称为“裁线规则 (line shortening rules)”。

在几何上，坐标 (coordinate) 对应没有内部尺寸的点 (没有长度、面积、体积)。为了照应这个数学观念，当一个 node 的形状是 coordinate 时，程序会认为其尺寸为 0，TikZ 自然不会对它使用“裁线规则”，它也没有任何内容，即 {<node contents>} 以及选项 node contents 此时对它无效。它的作用是给一个坐标点命名，以便于引用。



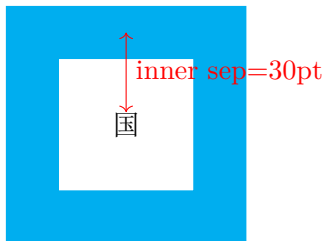
```
\begin{tikzpicture}[every node/.style={draw}]
\path[yshift=1.5cm,shape=rectangle]
(0,0) node(a1){} (1,0) node(a2){}
(1,1) node(a3){} (0,1) node(a4){};
\filldraw[fill=yellow!80!black]
(a1) -- (a2) -- (a3) -- (a4);
\path[shape=coordinate]
(0,0) coordinate(b1) (1,0) coordinate(b2)
(1,1) node (b3) {b3} (0,1) node (b4) {b4};
\filldraw[fill=yellow!80!black]
(b1) -- (b2) -- (b3) -- (b4);
\end{tikzpicture}
```

17.2.3 一般选项

`/pgf/inner sep=<dimension>` (无默认值, 初始值 `.3333em`)

`/tikz/inner sep=<dimension>` (无默认值, 初始值 `.3333em`)

设置 node 的内容与边界路径的坐标位置 (线条的中间, 不是线条的边界) 的间距 (包括水平和竖直两个)。初始值是 `0.3333em`, 所以如果令边界路径线宽是 `0.6666em` 会使得路径线条与内容相邻接。



```
\begin{tikzpicture}
\node[line width=20pt,inner sep=30pt,draw=cyan]
(g) {国};
\draw [<->,red](g.north)++(0,-10pt)
--node[right]{inner sep=30pt}++(0,-30pt);
\end{tikzpicture}
```

`/pgf/inner xsep=<dimension>` (无默认值, 初始值 `.3333em`)

`/tikz/inner xsep=<dimension>` (无默认值, 初始值 `.3333em`)

设置 node 的内容与边界路径的水平方向的间距。

`/pgf/inner ysep=<dimension>` (无默认值, 初始值 `.3333em`)

`/tikz/inner ysep=<dimension>` (无默认值, 初始值 `.3333em`)

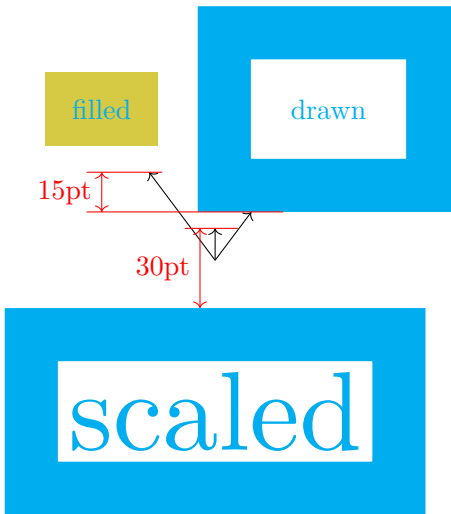
设置 node 的内容与边界路径的竖直方向的间距。

`/pgf/outer sep=<dimension or "auto">`

`/tikz/outer sep=<dimension or “auto” >`

在 node 外部额外添加一个不可见的空间，扩展 node 的边界，这个扩展间距从 node 的轮廓线条的坐标位置（线条中间）开始计算，而不是从轮廓线条的外侧边界开始计算。这个选项不改变 node 原来的轮廓线，但它使 node 的 anchor 位置向外扩展到新边界上。

为了与旧版本相兼容，这个选项的默认值是线宽的一半。因为路径坐标处于路径线条的中间，故路径坐标与路径线条边界的间距恰好是线宽的一半，所以 `outer sep` 的默认值一般应该是合适的。但是如果线宽值较大，node 只有 `fill` 选项而没有 `draw` 选项，就可能造成 node 的 anchor 位置距离文字内容过远，不够美观。而且 `outer sep` 选项设置的扩展边界还会接受 `scale` 选项的作用，但 node 的轮廓线路径的线宽不受 `scale` 选项的影响，这样也有可能造成 node 的 anchor 位置距离文字内容过远。

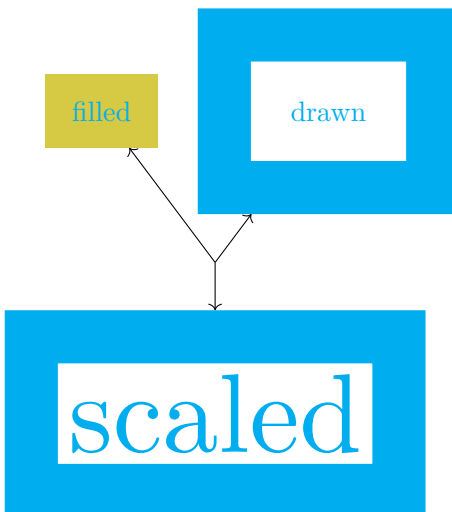


```
\begin{tikzpicture}
\draw[line width=20pt,cyan]
  (-0.5,0) node[fill=yellow!80!black,inner sep=10pt] (f) {filled}
  (2.5,0) node[draw,inner sep=25pt] (d) {drawn}
  (1,-4) node[draw,scale=4] (s) {scaled};
\draw[->] (1,-2) -- (f);
\draw[->] (1,-2) -- (d);
\draw[->] (1,-2) -- (s);
\draw [red] (f.south)+(-2mm,0)---(8mm,0);
\draw [red] (f.south |- d.south)+(-2mm,0)---(24mm,0);
\draw[<->,red] (f.south)--node[left]{15pt}(f.south |- d.south);
\draw[red] (s.north)+(-4mm,0)---(3mm,0);
\draw[<->,red] (s.north)+(-2mm,0)--node[left]{30pt}++(0,-30pt);
\end{tikzpicture}
```

上面例子中，(f) 与 (d) 的 `inner sep` 值相差 15pt，而 (s) 中有 `scale=4` 选项，所以坐标 (s.north) 与 (s) 原来的轮廓线条的距离是 $20\text{pt} \times 0.5 \times 4 - 20\text{pt} \times 0.5 = 30\text{pt}$ 。显然，指向 (f) 的箭头、指向 (s) 的箭头与目

标的间距过大。

解决这个问题的办法是使用 `outer sep=auto` 选项。在上面例子中给环境选项加上 `outer sep=auto` 得到如下图形



可见箭头的位置自动纠正了。

```
/pgf/outer xsep=<dimension>      (无默认值, 初始值 .5\pgflinewidth)
/tikz/outer xsep=<dimension>      (无默认值, 初始值 .5\pgflinewidth)
```

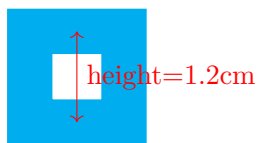
在水平方向上给 node 添加外部边界。注意如果同时写出 `outer sep` 和 `outer xsep` 选项, 则 `outer sep` 选项具有优先地位。

```
/pgf/outer ysep=<dimension>      (无默认值, 初始值 .5\pgflinewidth)
/tikz/outer ysep=<dimension>      (无默认值, 初始值 .5\pgflinewidth)
```

与 `outer xsep` 选项类似, 在竖直方向上给 node 添加外部边界。

```
/pgf/minimum height=<dimension>   (无默认值, 初始值 0pt)
/tikz/minimum height=<dimension>   (无默认值, 初始值 0pt)
```

设置 node 的最小高度, 即 node 可以高于这个高度, 但不能低于这个高度。这个高度的上下端都在轮廓线条中间。



```
\begin{tikzpicture}
\draw (0,0) node[line width=0.6cm,minimum height=1.2cm,
draw=cyan] (a){\rule{1cm}{0cm}};
\draw [<->,red] (a.south)++(0,0.3cm)--
node[right]{height=1.2cm}++(0,1.2cm);
\end{tikzpicture}
```

```
/pgf/minimum width=<dimension>     (无默认值, 初始值 0pt)
/tikz/minimum width=<dimension>     (无默认值, 初始值 0pt)
```

设置 node 的最小宽度。

`/pgf/minimum size=<dimension>` (无默认值)

`/tikz/minimum size=<dimension>` (无默认值)

同时设置 node 的最小高度和宽度。

`/pgf/shape aspect=<aspect ratio>` (无默认值)

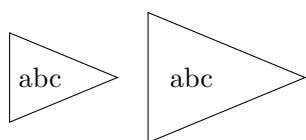
`/tikz/shape aspect=<aspect ratio>` (无默认值)

设置 node 的宽度与高度之比。

`/pgf/shape border uses incircle=<boolean>` (默认 true)

`/tikz/shape border uses incircle=<boolean>` (默认 true)

如果使用 `shape border uses incircle` 或 `shape border uses incircle=true`, 那么程序会对 node 的边界路径的各个尺寸做调整, 使得 node 的边界路径的内切圆将 node 的内容全部含在圆内, 这样内容与边界的距离就会比较匀称。



```
\tikzstyle{every node}=[isosceles triangle, draw]
\begin{tikzpicture}
\node {abc};
\node [shape border uses incircle] at (2,0) {abc};
\end{tikzpicture}
```

`/pgf/shape border rotate=<angle>` (无默认值, 初始值 0)

`/tikz/shape border rotate=<angle>` (无默认值, 初始值 0)

这个选项将 node 的边界旋转 `<angle>` 角度, 但不旋转内容。如果 node 没有带选项 `shape border uses incircle`, 则该选项只能将边界旋转 90° 的整数倍 (转角是与 `<angle>` 最临近的 90° 的整数倍)。如果 node 带有选项 `shape border uses incircle`, 则转角就没有这个限制。

用这个选项旋转边界时, node 坐标系中的罗盘位置 (与方向名称 north, east 等有关的位置, 用角度指出的位置)、与内容的基线相关的位置 (与 base 有关的位置) 都不被旋转, 其它的锚位置 (用上下左右指出的位置) 会随同边界一起旋转。

并非所有的 node 形状都支持选项 `shape border rotate` 的旋转, 例如 `rectangle` 形状就不支持。支持这个选项的形状不区分 `outer xsep` 和 `outer ysep` 这两个选项, 而是将这两个选项值中的较大者作为 `outer sep` 的值。

17.3 Multi-Part Nodes

一个形状为, 例如, `circle` 的 node, 可以将其中间画一横线, 分为上下两部分, 在上下部分别添加文字, 得到一个多部分 node。

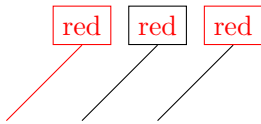
参考 `shapes.multipart` 程序库。

17.4 node 中的文字

17.4.1 文字参数：颜色、不透明度

node 中的文字的颜色由之前的 `color=` 选项来规定，也可以由下一选项来设置

```
/tikz/text=<color>
```



```
\tikzstyle{every node}=[rectangle, draw]
\begin{tikzpicture}
\draw[red] (0,0) -- +(1,1) node[above] {red};
\draw[text=red] (1,0) -- +(1,1) node[above] {red};
\draw (2,0) -- +(1,1) node[above,red] {red};
\end{tikzpicture}
```

文字的不透明度用选项 `text opacity=<value>` 设置，参考 §23.

17.4.2 文字参数：字体

字体包括字族、字形、尺寸等文字属性。

```
/tikz/node font=<font commands>
```

这个选项设置 node 内容中的文字字体。有时会希望文字的尺寸与 node 的形状尺寸相匹配，例如 `minimum size=4em` 规定 node 的最小尺寸，其中用了相对长度单位 `em`，`em` 会随着正文默认的字体尺寸的变化而变化。选项 `node font` 设置的文字尺寸会对以 `em`，`ex` 定义的 node 形状尺寸产生影响，即在当前分组或路径内，用选项 `node font` 设置字体尺寸可以重设 `em`，`ex` 的长度。



```
\tikzstyle{every node}=[rectangle, draw]
\tikz \node [node font={\it \tiny}, minimum height=3em, draw]
{tiny};
\tikz \node [node font={\sf \small}, minimum height=3em, draw]
{small};
```

```
/tikz/font=<font commands>
```

这个选项设置 node 内容中的文字字体，但不会重设当前分组或路径内的 `em`，`ex` 的长度。

17.4.3 文字参数：文字换行、对齐方式、文字行宽

一般情况下，node 的文字会被放入一个水平盒子中，不会自动换行，有时会显得太长。创建多行内容的办法有以下几种：

1. 使用 L^AT_EX 环境或其它宏包提供的环境，例如表格环境 `{tabular}`，矩阵环境。
2. 使用对齐选项 `align`，然后在内容中使用两个反斜线 `\\` 来换行（必须先设置对齐方式）。可以使用 `\\[<dimension>]` 在换行时追加垂直间距，这个间距可以是负的。

3. 使用选项 `text width` 设置文本行的宽度，这样可以自动换行，而且也可以用 `\` 手动换行。注意这个选项会使得 `node` 的宽度不小于所设置的文本行的宽度。

```
/tikz/text width=<dimension>
```

这个选项会将 `node` 的内容放入一个宽度为 `<dimension>` 的盒子中，盒子作用类似 `{minipage}` 环境，其中可以自动换行，也可手动换行，也会使得 `node` 的宽度不小于 `<dimension>`。给出这个选项后，内容会自动使用左对齐方式 (`align=left`)。行末可能出现断词（连字符）。

如果 `<dimension>` 留空，本选项无效，取消自动换行。

```
/tikz/align=<alignment option>
```

这个选项设置多行 `node` 内容的对齐方式。如果使用了 `text width=<dimension>` 且 `<dimension>` 非空，则 `align` 选项会参照 `<alignment option>` 设置 `\leftskip` 以及 `\rightskip` 来实现对齐和换行，这个状况叫作“有断行对齐”。如果没有使用 `text width=<dimension>` 或者 `<dimension>` 为空，则 `align` 选项会使用选项 `node halign header` 确定的机制来实现对齐，这个状况叫作“无断行对齐”，此时可以使用 `\` 来手动换行。

对齐方式 `<alignment option>` 有以下几种：

```
align=left
```

这个对齐方式使用 plain $\text{T}_{\text{E}}\text{X}$ 所定义的左对齐（ragged right）方式， $\text{T}_{\text{E}}\text{X}$ 会尽量平衡文本行以减少文本右侧的不平整程度，因此可能会在行末自动断词并添加连字符号。

```
align=flush left
```

这个对齐方式使用 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 所定义的左对齐方式，不会自动平衡文本行，不会在行末出现断词（连字符），故文本右侧可能很是参差不齐。

```
align=right
```

类似 `align=left`。

```
align=flush right
```

类似 `align=flush left`。

```
align=center
```

类似 `align=left`。

This is a
demonstration text for
alignments.

```
\tikz \node[fill=yellow!80!black,align=center]
{This is a\\ demonstration text for\\
alignments.};
```

This is a demon-
stration text for
showing how line
breaking works.

```
\tikz \node[fill=yellow!80!black,text width=3cm,
align=center]
{This is a demonstration text for
showing how line breaking works.};
```

当 `text width=<dimension>` 设置的宽度很长,但 `node` 的文字内容很短时,TeX 会给出由 `align=center` 创建的盒子所引起的水平方向的劣质警告信息 (horizontal badness warnings), 这是无法避免的, 而默认 TikZ 关闭这些水平方向的劣质警告以及其它某些 (可能有用的) 警告, 可以使用下一选项恢复这些警告:

```
/tikz/badness warnings for centered text=<true or false> (无默认值, 初始值 false)
```

如果这个选项的值设为 `true`, 则会发出针对由 `align=center` 创建的盒子的各种警告, 如果这些盒子是你需要的设计效果, 当然可以置之不理。

```
align=flush center
```

类似 `align=flush left`.

```
align=justify
```

在“无断行对齐”时,即没有使用 `text width=<dimension>` 或者 `<dimension>` 为空,这个选项等效于 `align=left`; 在“有断行对齐”时,即使用了 `text width=<dimension>` 且 `<dimension>` 非空, 这个选项会使文本行分散对齐 (文字在一行中均匀分布)。

```
align=none
```

这个选项取消对齐, 也不能使用 `\\` 手动换行。

```
/tikz/node halign header=<macro storing a header> (无默认值, 初始值 empty)
```

这个选项会使用命令 `\halign` 的机制。命令 `\halign` 是 TeX 的基本命令 (参考《The TeXbook》), 它可以制造表格, 其句法如下

```
\halign{列格式行\cr 表格行1\cr ... 表格行n\cr}
```

例如

<i>China</i>	Apple	\$1.0
<i>France</i>	Banana	\$0.574

```
\halign{\it#\tabskip=1em & \hfil#\hfil & \hfil\$\#\cr
China & Apple & 1.0\cr
France & Banana & 0.574\cr}
```

这个制表例子中, `\cr` 是换行标志。第一行定义列格式, `#` 代表单元格数据, `&` 是分列符号, 所以第一行定义了一个 3 列表格, 第一列 (按默认编辑方式) 左对齐, 第二列用两个 `\hfil` 命令包围单元格数据实现居中对齐, 第三列右对齐。命令 `\tabskip=1em` 在当前列与右侧各列之间插入 `1em` 的间距。

选项 `node halign header` 借用了命令 `\halign` 的机制, 但这个选项只能定义“只有一列的表格”。这个选项将该命令的换行符号 `\cr` 改为双反斜线 `\\`, 在文本中用 `\\` 手动换行。文本中的每个换行都对应表格的“新行”, 每行文本都放入一个水平盒子中。最后一行 (包括只有一行的文本) 不必使用 `\\` 来结束。

对齐方式由 `<macro storing a header>` 规定, 例如

```
\def\myheader{\hfil\hfil##\hfil\cr}
```

```
\tikz [node align header=\myheader] ...
```

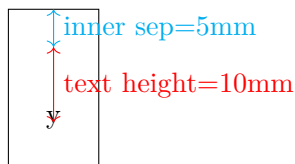
这个例子中，首先定义宏 `\myheader`，这个宏的内容就是命令 `\halign` 的列格式定义，但只能定义一列（不出现 `&` 符号）。然后将 `\myheader` 作为选项 `node align header` 的值。注意不能直接将列格式定义 `\hfil\hfil#\hfil\cr` 作为该选项的值，而必须用这种曲折的方式。

17.4.4 文字参数：文字的高度和深度

选项 `text height` 和 `text depth` 可以调整文字盒子的高度和深度，从而控制 `node` 的尺寸。

```
/tikz/text height=<dimension>
```

如果 `<dimension>` 空置，则使用文字盒子的“自然高度”。



```
\begin{tikzpicture}
\node (y)[draw,text height=10mm,inner sep=5mm]{y};
\draw [<->,red](y.base)--
    node[right]{text height=10mm}++(0,10mm);
\draw [<->,cyan](y.base)++(0,10mm)--
    node[right]{inner sep=5mm}++(0,5mm);
\end{tikzpicture}
```

```
/tikz/text depth=<dimension>
```

如果 `<dimension>` 空置，则使用文字盒子的“自然深度”。

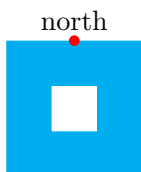
设置文字的高度和深度可以实现某种对齐效果。

17.5 Positioning Nodes

一般情况下，把 `ndoe` 放在某个坐标位置上时，`node` 的中心会处于这个位置上。

17.5.1 node 的锚位置

PGF 可以使用锚机制来调整 `ndoe` 的位置。每个 `node` 形状都有自己的坐标系，坐标原点位于 `node` 形状的中心。`node` 的锚位置就是这个坐标系中的点，这些点（除了 `center`）都位于形状线条的外侧边界上（不是线条的中间）。



```
\tikz {\node [draw=cyan,inner sep=0.6cm,line width=0.6cm] (n){};
\fill [red](n.north)circle(2pt);
\node [above]at(n.north){north};
}
```

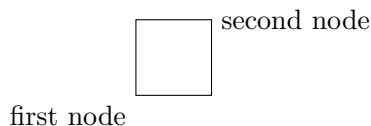
锚位置包括

- 带有方向名词的名称：north, east, north east 等。
- 与内容文字的基线 `base` 有关的名称：base, base west, base east.

- 与 node 形状的竖直方向的中点 mid 有关的名称: mid, mid west, mid east.
各种形状的锚位置参考 §67.

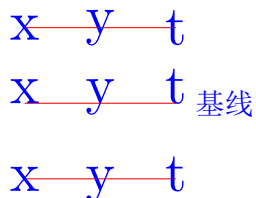
`/tikz/anchor=<anchor name>`

这个选项将 node 的名称为 <anchor name> 的位置放在 (该 node 指向的) 指向点上。注意“锚与船的位置是相对的”，如果将 node 的位置 north 放在某个坐标点上，那么 node 的中心将位于该点的“南方”；如果将 node 的位置 north east 放在某个坐标点上，那么 node 的中心将位于该点的“西南方”。



```
\tikz \draw (0,0) node[anchor=north east] {first node}
      rectangle (1,1) node[anchor=west] {second node};
```

使用关于 base, mid 的锚位置可以实现某种对齐效果:

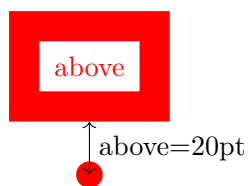


```
\begin{tikzpicture}[scale=2,red,transform shape]
\draw[anchor=center] (0,1) node{x} -- (0.5,1) node{y} -- (1,1) node{t}
      node[anchor=west]{\tiny 基线};
\draw[anchor=base] (0,.5) node{x} -- (0.5,.5) node{y} -- (1,.5) node{t};
\draw[anchor=mid] (0,0) node{x} -- (0.5,0) node{y} -- (1,0) node{t};
```

17.5.2 基本的平移选项

`/tikz/above=<offset>` (默认值 0pt)

如果不指定 <offset>, 那么这个选项的作用等于 `anchor=south`; 如果指定 <offset>, 那么 node 会相对于指向点向上平移, 使得 node 的 south 位置位于指向点之上的 <offset> 距离处。



```
\tikz {\fill [red](0,0) circle (5pt)
      node(上)[above=20pt,draw,line width=4mm,
      inner sep=4mm] {above};
\draw [<->](上.south)--node[right]{above=20pt}
      ++(0,-20pt);
}
```

`/tikz/below=<offset>`

(默认值 0pt)

类似 `above=<offset>`.

`/tikz/left=<offset>` (默认值 `0pt`)

类似 `above=<offset>`.

`/tikz/right=<offset>` (默认值 `0pt`)

类似 `above=<offset>`.

`/tikz/above left`

等效于 `anchor=south east`, 注意 `[above,left]` 这两个选项只有后者有效, 故不等于 `[above left]`.

`/tikz/above right`

`/tikz/below left`

`/tikz/below right`

`/tikz/centered`

17.5.3 高级平移选项

首先调用 `positioning` 程序库, 然后就可以用相对复杂的句式来规定平移距离和平移方式。

`/tikz/above=<specification>` (默认值 `0pt`)

`<specification>` 有两种情况:

- 只有 `<shifting part>`, 即只有指定平移距离的句式, 这里的句式又有以下 3 种形式:
 1. 关于长度 (带有长度单位) 的算式, 例如 `2cm` 或 `3cm/2+4cm`, 如同前面所述的 (相当于不载入 `positioning` 程序库的) `above` 选项的作用, 这会使得 `node` 的 `south` 位置位于指向点之上, 二点距离为算式所指定。
 2. 关于纯数字 (不带单位) 的算式, 例如 `2` 或 `3+sin(60)`, 如果算式的计算结果是 `<number>`, 则 `node` 的 `south` 位置相对于指向点上移, 平移向量为 $(0, \langle \text{number} \rangle)$ 。
 3. 用 `and` 给出两个数据, `<number or dimension 1>` and `<number or dimension 2>`, 例如 `above=.2 and 3mm`, 这里 `.2` 的默认长度单位是 `cm`. 此时程序会构造一个平移向量: $(\langle \text{number or dimension 2} \rangle, \langle \text{number or dimension 1} \rangle)$, 例如 $(3\text{mm}, 0.2\text{cm})$, 注意其中将 `and` 后的数据作为横标, `and` 前的数据作为纵标。横标指示横向平移距离, 纵标指示纵向平移距离。而 `above` 是纵向平移, 所以只有纵标, 即 `and` 前的数据对 `above` 选项有意义。
- 带有 `<of-part>`, 即用 `of` 指定 `node` 的指向点。 `<of-part>` 可以用以下形式:

`of 1,2`

`of 1,2 -| 2,1`

`of {(0,0)!0.5!(2,2)}` 其中必须用花括号把坐标算式括起来

`of {\sqrt{2}},{\exp(0.5)}` 其中必须使用二重套嵌花括号

`of a.north` 其中 `a` 是某个 `node` 的名称

有了指向点, 然后平移 `node`, 使其 `south` 位置位于指向点之上, 二点的距离由 `of` 之前的句式指定。所以


```
\node [above=5mm of somenode.north east]{};
```

等效于

```
\node [above=5mm] at(somenode.north east){};
```

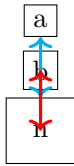
如果不指定平移距离，即等号“=”后没有算式，例如 `above= of somenode.north`，那么平移距离就由选项 `node distance` 规定。

2. `<of-part>` 是 `of <node name>` 这种形式，例如

```
\node (a)[above=5mm of somenode]{};
```

此时 (a) 的锚位置 `south` 位于 (somenode) 的 `north` 位置之上 5mm 处。如果不指定平移距离，即等号“=”后没有算式，那么平移距离就由选项 `node distance` 规定。

如果在选项中给出锚位置和平移距离，例如 `[above=6mm,anchor=center]`，那么后者优先。如果将某个 `node` 的名称用做算子 `at` 的参数，如 `at(<node name>)`，则算子 `at` 决定的指向点位于 `<node name>` 的中心，即其 `center` 位置。



```
\tikz {\node [draw,inner sep=10pt] (n){n};
  \node (a)[above=8mm of n,draw]{a};
  \node (b)[above=8mm,anchor=center,draw] at(n){b};
  \draw [<->,cyan,very thick] (n.north)--++(0,8mm);
  \draw [<->,red,very thick] (n.center)--++(0,8mm);
}
```

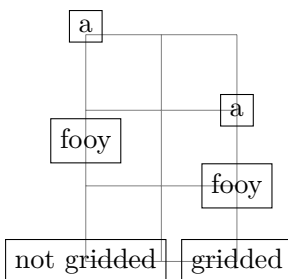
`/tikz/on grid=<boolean>` (无默认值，初始值 `false`)

前面讲解 `above` 选项的平移作用时，平移距离指的都是 `node` 的轮廓线条的外侧边界之间的距离（把坐标看作一种无内部尺寸的 `node`），即新 `node` 的 `south` 位置与旧 `node` 的 `north` 位置（或某个坐标点）之间的距离。

当设置选项 `on grid=true` 后，举例来说，

```
\node (a)[above=5mm of somenode]{};
```

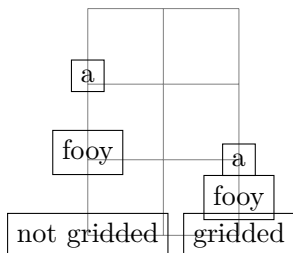
其中 (a) 的中心 `center` 位置位于 (somenode) 的中心 `center` 之上 5mm 处。



```
\begin{tikzpicture}[every node/.style=draw]
\draw[help lines] (0,0) grid (2,3);
% Not gridded
\node (a1) at (0,0) {not gridded};
\node (b1) [above=1cm of a1] {fooy};
\node (c1) [above=1cm of b1] {a};
% gridded
\node (a2) at (2,0) {gridded};
\node (b2) [on grid,above=1cm of a2] {fooy};
\node (c2) [on grid,above=1cm of b2] {a};
\end{tikzpicture}
```

`/tikz/node distance=<shifting part>` (无默认值, 初始值 1cm and 1cm)

如果不指定平移距离, 即等号 “=” 后没有算式, 那么平移距离就由该选项规定。



```
\begin{tikzpicture}[every node/.style=draw,node distance=5mm]
\draw[help lines] (0,0) grid (2,3);
% Not gridded
\node (a1) at (0,0) {not gridded};
\node (b1) [above=of a1] {fooy};
\node (c1) [above=of b1] {a};
% gridded
\begin{scope}[on grid]
\node (a2) at (2,0) {gridded};
\node (b2) [above=of a2] {fooy};
\node (c2) [above=of b2] {a};
\end{scope}
\end{tikzpicture}
```

`/tikz/below=<specification>`

类似 above.

`/tikz/left=<specification>`

类似 above.

`/tikz/right=<specification>`

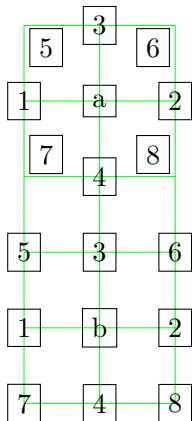
类似 above.

`/tikz/above left=<specification>`

类似 above, 但 <shifting part> 的形式决定的平移方式有所不同:

1. 如果 <shifting part> 的形式是 <number or dimension 1> and <number or dimension 2>, 程序会构造一个平移向量: (<number or dimension 2>, <number or dimension 1>), 注意其中将 and 后的数据作为横标, and 前的数据作为纵标。横标指示横向平移距离, 纵标指示纵向平移距离。也就是说, 纵向平移量为 <number or dimension 1>, 横向平移量为 <number or dimension 2>.
2. 如果 <shifting part> 的形式是 <number or dimension>, 那么平移向量是极坐标 (135:<number or dimension>), 其中的 135° 对应 “左上角” 方向。

下面的例子显示不同的 <shifting part> 的形式造成称的差别:



```

\begin{tikzpicture}[every node/.style={draw,rectangle},on grid]
\draw[help lines,green] (1,-1) grid (3,4);
\begin{scope}[node distance=1]
\node (a) at (2,3) {a};
\node [left=of a] {1}; \node [right=of a] {2};
\node [above=of a] {3}; \node [below=of a] {4};
\node [above left=of a] {5}; \node [above right=of a] {6};
\node [below left=of a] {7}; \node [below right=of a] {8};
\end{scope}
\begin{scope}[node distance=1 and 1]
\node (b) at (2,0) {b};
\node [left=of b] {1}; \node [right=of b] {2};
\node [above=of b] {3}; \node [below=of b] {4};
\node [above left=of b] {5}; \node [above right=of b] {6};
\node [below left=of b] {7}; \node [below right=of b] {8};
\end{scope}
\end{tikzpicture}

```

`/tikz/below left=<specification>`

类似 above left.

`/tikz/above right=<specification>`

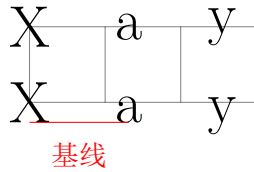
类似 above left.

`/tikz/below right=<specification>`

类似 above left.

`/tikz/base left=<specification>`

左移 node, 使得其 base east 位置位于指向点的左侧, 并指定二点距离。如果 <specification> 中有 of <node name> 这种句式, 则新 node 的文字基线与旧的 <node name> 的文字基线在同一水平上, 新 node 的 base east 位置位于旧的 <node name> 的 base west 的左侧, 并指定二点距离。



```

\begin{tikzpicture}[node distance=1ex]
\draw[help lines] (0,0) grid (3,1);
\huge
\node (X) at (0,1) {X};
\node (a) [right=of X] {a};
\node (y) [right=of a] {y};
\node (X) at (0,0) {X};
\node (a) [base right=of X] {a};
\node (y) [base right=of a] {y};
\draw [red](X.base)--
      node[below]{\normalsize 基线}(a.base);
\end{tikzpicture}

```

`/tikz/base right=<specification>`

类似 base left.

`/tikz/mid left=<specification>`

类似 base left.

`/tikz/mid right=<specification>`

类似 base left.

17.5.4 排布 node 的高级方法

参考 graphdrawing 程序库和 matrix 程序库。

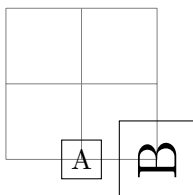
17.6 Fitting Nodes to a Set of Coordinates

参考 fit 程序库, §52.

17.7 变换

路径上的 node 不属于路径, 所以路径选项中的变换, 例如 scale, rotate 等对路径上的 node 无效。

可以给 node 加变换选项来实现 node 的放缩、旋转、平移等变换, node 的旋转和平移都是相对于它的指向点的。



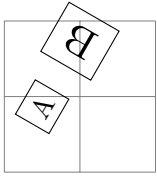
```

\begin{tikzpicture}[every node/.style={draw}]
\draw[help lines](0,0) grid (2,2);
\draw (1,0) node{A}
      (2,0) node[rotate=90,scale=2] {B};
\end{tikzpicture}

```

`/tikz/transform shape nonlinear=<true or false>` (无默认值, 初始值 false)

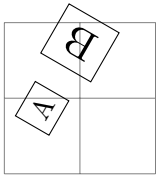
如果 node 带有这个选项, 那么路径选项中的变换选项将对该 node 起作用。



```
\begin{tikzpicture}[every node/.style={draw}]
\draw[help lines](0,0) grid (2,2);
\draw[rotate=60] (1,0) node[transform shape] {A}
(2,0) node[transform shape,rotate=90,scale=1.5] {B};
\end{tikzpicture}
```

上面例子中的第二个 node 本身带有变换选项, 由于它有 transform shape 选项, 它还受到路径选项中的变换选项的作用。

如果 transform shape 选项用作环境选项, 则会对环境内的所有 node 有效, 所以上面的例子还可以如下画出:



```
\begin{tikzpicture}[every node/.style={draw},transform shape]
\draw[help lines](0,0) grid (2,2);
\draw[rotate=60] (1,0) node {A}
(2,0) node[rotate=90,scale=1.5] {B};
\end{tikzpicture}
```

`/tikz/transform shape nonlinear=<true or false>` (无默认值, 初始值 false)

目前, 默认 TikZ 关闭“非线性变换”功能, 这个选项将允许 TikZ 对路径以及 node 作非线性变换。参考 §103.4.

17.8 在直线段或曲线上显式地摆放 node

本节下文列举的选项仅对“--”, “arc”, 控制曲线符号“..”, 纵横线符号“-|”这4种算子生成的路径(子路径)上的 node 有效。这里所说的“显示地(explicitly)”的意思是, node 语句位于这些算子确定的路径(子路径)的终点之后, 例如 $(0,0)\text{--}(1,1)\text{node}\{a\}$ 。如果 node 语句位于这些算子之后, 例如 $(0,0)\text{--node}\{a\}(1,1)$, 则是“隐式地(implicitly)”, 这两种方式的区别在下节解释。

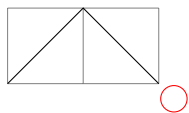
`/tikz/pos=<fraction>` (无默认值)

当 node 带有这个选项后, 程序会把该 node 与其之前的路径算子联系起来, 程序会根据这些路径算子画出的路径(子路径), 以及 <fraction> 的值来决定该 node 的位置(指向点)。但是这些路径算子目前仅限于“--”, “arc”, 控制曲线符号“..”, 纵横线符号“-|”这4种。

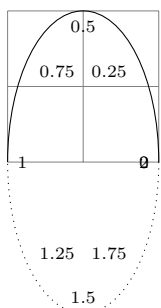
如果路径算子画出的路径(子路径)具有起点和终点, 那么当 <fraction> 是 1 时, 该 node 的位置(指向点)在终点; 当 <fraction> 是 0 时, 该 node 的位置(指向点)在起点; 当 <fraction> 大于 0 小于 1 时, 该 node 的位置(指向点)在起点与终点之间; 当 <fraction> 是其它值时, 该 node 的位置(指向点)在路径(子路径)之外。对不同的路径算子会有不同的情况。

而且一个路径算子后面可以有多个 node。

举例如下：



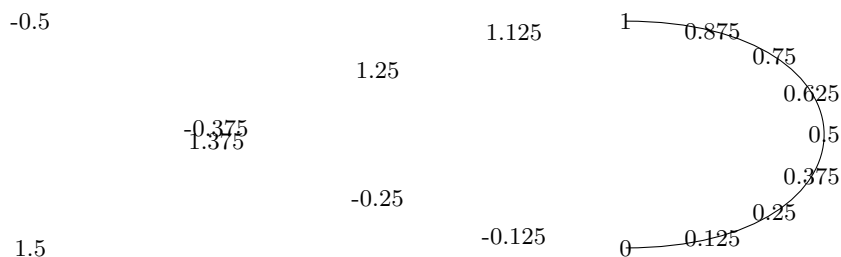
```
\begin{tikzpicture}[every node/.style={draw},transform shape]
\draw[help lines](0,0) grid (2,1);
\draw (0,0)--(1,1)--(2,0)node [circle,draw=red,pos=1.2]{} ;
\end{tikzpicture}
```



```
\tikz {\draw [help lines] (0,0) grid (2,2);
\draw (2,0) arc [x radius=1, y radius=2, start angle=0,
end angle=180]
node foreach \t in {0,0.25,...,2}
[pos=\t,auto,font=\scriptsize] {\t};
\draw [y={(0,-1)},dotted](2,0) arc [x radius=1, y radius=2,
start angle=0,end angle=180]
}
```

上面例子显示，对于 arc 算子而言，当 $\langle \text{fraction} \rangle$ 大于 1 时，node 的指向点会沿着 arc 算子决定的椭圆路径延伸摆放。

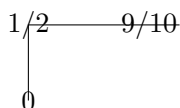
关于控制曲线的例子如下：



```
\tikz \draw (0,0) .. controls +(right:3.5cm) and +(right:3.5cm) .. (0,3)
node foreach \p in {-0.5,-0.25,...,1.5} [pos=\p]{\small \p};
```

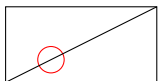
对于控制曲线而言， $\langle \text{fraction} \rangle$ 决定 node 的指向点的数学机制稍微复杂， $\text{pos}=0.5$ 未必决定 node 的指向点在曲线的一半长度的位置，具体可以参考“计算机图形学”或“Bézier”曲线方面的资料。

对于纵横线算子“|-”而言， $\text{pos}=0.5$ 决定的位置在拐角点处：



```
\tikz \draw (0,0) |- (2,1)
node[pos=0]{0}
node[pos=0.5]{1/2}
node[pos=0.9]{9/10};
```

对于其它的路径算子，目前 $\text{pos}=\langle \text{fraction} \rangle$ 选项不能获得期望的位置。例如该选项尚未与“circle”，“sin”算子画出的路径联系起来。对于“rectangle”算子，该选项决定的位置在对角线上：



```
\begin{tikzpicture}[every node/.style={draw},transform shape]
\draw (0,0) rectangle (2,1) node[pos=0.3,draw=red, circle]{};
\draw (0,0)--(2,1); % 对角线
\end{tikzpicture}
```

/tikz/auto=<direction> (默认为 **scope** 的设置)

node 带有这个选项后, 将使它位于路径的 (沿着路径的方向确定的) 某一侧。

auto

如果选项 auto 没有 <direction> 值, 那么程序会参照之前的 auto 的值来确定 node 将位于路径的哪一侧。

auto=left

使得 node 位于路径的左侧。

auto=right

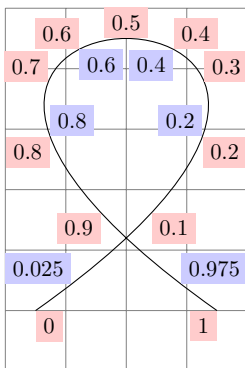
使得 node 位于路径的右侧。

auto=false

取消自动确定 node 位置的机制, 还会取消选项 anchor, above, below 等的作用。

/tikz/swap (无值)

如果 auto 选项决定了 left (right) 一侧, 那么 swap 选项将把 node 的位置转换到 right (left) 一侧。



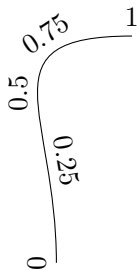
```
\begin{tikzpicture}[auto,scale=0.8,transform shape]
\draw[help lines,use as bounding box]
(0,-1) grid (4,5);
\draw (0.5,0) .. controls (9,6) and (-5,6) .. (3.5,0)
node foreach \pos in
{0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1}
[pos=\pos,swap,fill=red!20] {\pos}
node foreach \pos in {0.025,0.2,0.4,0.6,0.8,0.975}
[pos=\pos,fill=blue!20] {\pos};
\end{tikzpicture}
```

/tikz/' (无值)

这个是单引号, 或者撇号, 等效于 swap 选项。

/tikz/sloped (无值)

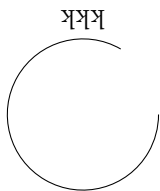
如果这个选项作用于 node, 那么其内容将会被旋转, 使得内容的展开方向沿着路径的切线方向。sloped 选项在旋转 node 的内容时, 不会使得内容“上下颠倒”。



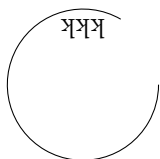
```
\tikz \draw (0,0) .. controls +(up:2cm) and +(left:2cm) .. (1,3)
node foreach \p in {0,0.25,...,1} [sloped,above,pos=\p]{\p};
```

`/tikz/allow upside down=<boolean>` (默认值 `true`, 初始值 `false`)

当 node 带有 `allow upside down` 或 `allow upside down=true` 时, 该 node 的内容会“上下颠倒”。



```
\begin{tikzpicture}[auto]
\draw (0,0) arc (60:360:1)node[pos=0.1,sloped,
allow upside down]{kkk};
\end{tikzpicture}
```



```
\begin{tikzpicture}[auto]
\draw (0,0) arc (60:360:1)node[pos=0.1,sloped,swap,
allow upside down]{kkk};
\end{tikzpicture}
```

`/tikz/midway` (style, 无值)

这个选项等效于 `pos=0.5`.

`/tikz/near start` (style, 无值)

这个选项等效于 `pos=0.25`.

`/tikz/near end` (style, 无值)

这个选项等效于 `pos=0.75`.

`/tikz/very near start` (style, 无值)

这个选项等效于 `pos=0.125`.

`/tikz/very near end` (style, 无值)

这个选项等效于 `pos=0.875`.

`/tikz/at start` (style, 无值)

这个选项等效于 `pos=0`.

`/tikz/at end` (style, 无值)

这个选项等效于 `pos=1`.

at end
very near end
near end
midway
near start
very near start
at start

```
\tikz [scale=0.9]
\draw (0,0) .. controls +(up:2cm) and +(left:3cm) .. (1,5)
  node[at end] {\texttt{at end}}
  node[very near end] {\texttt{very near end}}
  node[near end] {\texttt{near end}}
  node[midway] {\texttt{midway}}
  node[near start] {\texttt{near start}}
  node[very near start] {\texttt{very near start}}
  node[at start] {\texttt{at start}};
```

17.9 在直线段或曲线上隐式地摆放 node

“隐式地 (implicitly)”的意思已在前节解释，即把 node 语句放在路径算子之后。在多数情况下，“显示”与“隐式”的效果是一样的，但程序对二者的处理方式不同，以致有特殊的差别。

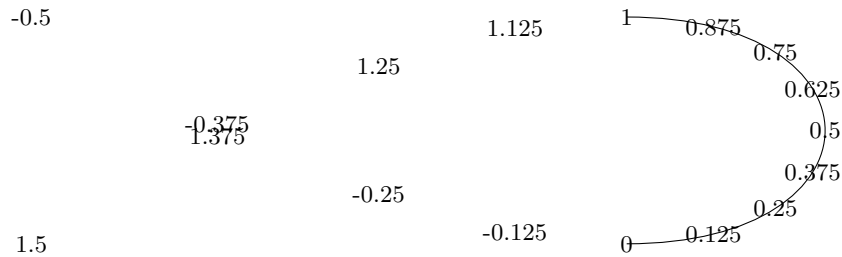
举例而言，如果 node 的内容中有抄录命令，那么显式的 $(0,0)\text{--}(1,1)\text{node}\{\text{\verb\&\small\&}\}$ 可以接受，但隐式的 $(0,0)\text{--}\text{node}\{\text{\verb\&\small\&}\}(1,1)$ 不能接受。

当写出 $(0,0)\text{--}\text{node}\{a\}(1,1)$ 时，node 的内容 a 将位于路径的中间位置。

当写出

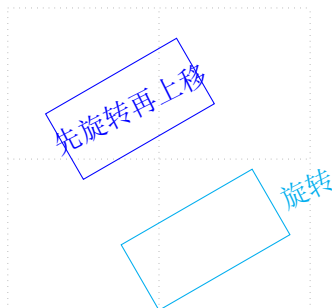
```
(0,0) -- node <node specification> (1,1)
```

时，程序读取 <node specification> 并将其另外存储，读完坐标 (1,1) 后，另存的 <node specification> 才被调出并处理。当 <node specification> 被存储时，其中各个符号的类别就已确定，因此 node 的内容中不能有抄录命令。



```
\tikz\draw (0,0) .. node foreach \p in {-0.5,-0.375,...,1.5} [pos=\p]{\small \p}controls
  +(right:3.5cm) and +(right:3.5cm) .. (0,3);
```

如果把 node 语句放在算子 rectangle 之后，那么 node 将位于矩形的中心。



```
\begin{tikzpicture}[scale=2]
\draw [help lines,dotted] (-1,0) grid (1,2);
\draw [rotate=30] [cyan] (0,0) rectangle node[sloped,right=1cm]{旋转} (1,0.5) ;
\draw [rotate=30,yshift=1cm] [blue]
(0,0) rectangle node[sloped]{旋转, 上移} (1,0.5);
\end{tikzpicture}
```

17.10 label 和 pin 选项

17.10.1 Overview

17.10.2 label 选项

`/tikz/label=[<options>]<angle>:<text>`

若一个 node 带有这个选项，当该 node 完成后，程序会给它加一个 node 作为它的标签。这两个 node 暂时分别称为 main node 和 label node，选项 label 的各个值解释如下：

1. `<angle>` 指定 main node 的坐标系中的点，这个点就是需要加标签的点，即 label node 的指向点。`<angle>` 的值可以是数值（代表角度），锚位置名称（north, east 等），平移位置名称（above, left 等），程序会把 above 转换成角度 90，把 left 转换成角度 180，等等。

注意，对于 label node 来说，`<angle>=north east` 与 `<angle>=above right` 的效果并不相同。`<angle>=north east` 确定的点是 main node 的锚位置 north east；而 `<angle>=above right` 的作用则是平移 label node，即假定从 main node 的中心点出发且角度是 45° 的射线与 main node 的边界线交于点 P ，点 P 就是 label node 的指向点，平移 label node 使其锚位置 south west 与点 P 重合。

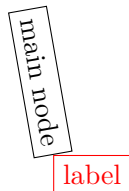
如果不给出 `<angle>`，那么就默认使用下一个选项的值：

`/tikz/label position=<angle>` （无默认值，初始值 above）

这个选项的初始值是 above，也就是说，选项 `label=<text>` 会把标签 `<text>` 加在 main node 的上部。

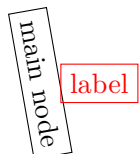
`/tikz/absolute=<>true or false>` （默认值 true）

如果 main node 受到变换，例如旋转变换，程序会默认 main node 的坐标系是固着在 main node 上随之一起旋转的，因此 label 选项中的 `<angle>` 也会被旋转。如果设置 absolute 或 `absolute=true`，程序会认为 main node 的坐标系是不随着 main node 一起旋转的，因而是绝对的（不变方向的），当然 `<angle>` 所指出的点仍然在边界上。这个选项所说的“绝对（absolute）”就是这个意思。



```
\tikz [rotate=-80, every label/.style={draw, red}]
\node [transform shape, rectangle, draw,
label=right:label] {main node};
```

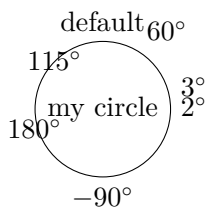
上面例子中，main node 被旋转了 -80 度，它的 right 位置也被旋转了 -80 度，标签使用了 right 位置，故标签跟随这个位置。但此时标签的“above left”位置位于 main node 的 right 位置上。



```
\tikz [rotate=-80, every label/.style={draw, red}, absolute]
  \node [transform shape, rectangle, draw,
        label=right:label] {main node};
```

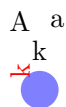
上面例子中，使用了 absolute 选项，main node 的 right 方向不跟随旋转，right 位置仍然在水平向右一侧的边界上，此时标签的“left”位置处于 main node 的 right 位置上。

2. `<angle>` 确定了 label node 的指向点，程序会把相应的 label node 的锚位置放在这个点上，放置规则是：0, 90, 180, 270，这四个角度是“主角度 (major angle)”；如果 `<angle>` 是 0 ± 2 ，则 label node 的 west 位置处于该点上；如果 `<angle>` 是 90 ± 2 ，则 label node 的 south 位置处于该点上；如果 `<angle>` 是 180 ± 2 ，则 label node 的 east 位置处于该点上；如果 `<angle>` 是 270 ± 2 ，则 label node 的 north 位置处于该点上；如果 `<angle>` 是其它角度，则 label node 的其它锚位置处于该点上，例如，若 `<angle>` 是 30，则 label node 的 south west 处于该点上。



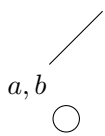
```
\tikz
  \node [circle, draw,
        label=default,
        label=60:$60^\circ\text{\circ}$,
        label=below:$-90^\circ\text{\circ}$,
        label=3:$3^\circ\text{\circ}$,
        label=2:$2^\circ\text{\circ}$,
        label={[below]180:$180^\circ\text{\circ}$},
        label={[centered]135:$115^\circ\text{\circ}$}] {my circle};
```

3. 如果 `<angle>` 是 center，则 label node 会处于 main node 的中心。
4. 如果使用多个 label 选项，这些标签会依次画出，而且 label 选项可以套嵌使用。



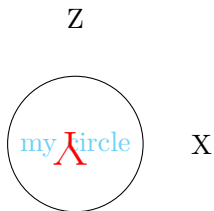
```
\tikz \node [circle, fill=blue!50, minimum size=0.5cm,
            label=90:k,
            label={[text=red, rotate=90,
                  label={[absolute, label distance=1ex,
                        label=right:a]90:A}]
            90:k}] {};
```

label 选项实际生成一个 node，所以那些能用在算子 node 后的方括号里的选项也能够用于 label 选项，这些选项要放在 `<angle>` 之前的方括号里，此时还要把整个 label 选项的值用花括号括起来。



```
\begin{tikzpicture}
\node [circle,draw,label={[name=label node]above left:$a,b$}] {};
\draw (label node) -- +(1,1);
\end{tikzpicture}
```

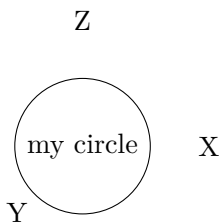
如果 label 选项带有 rotate 选项，则 label node 会围绕它的指向点 ($\langle angle \rangle$ 确定 main node 上的点) 整体 (包括它的内容) 旋转。



```
\tikz[label distance=5mm]
\node [circle,draw,label=right:X,
label={[rotate=180,red]above right:{\LARGE Y}},
label=above:Z] {\color{cyan!50}my circle};
```

/tikz/label distance= $\langle distance \rangle$ (无默认值, 初始值 Opt)

这个选项设置 $\langle angle \rangle$ 确定的指向点与 label node 的锚位置之间的距离，可以是负值。



```
\tikz[label distance=5mm]
\node [circle,draw,label=right:X,
label={[label distance=-2.5cm]above right:Y},
label=above:Z] {my circle};
```

/tikz/every label (style, 默认值 draw=none, fill=none 初始值 empty)

设置每个 label node 的样式，默认值 draw=none, fill=none.

可以给一个 node 加多个 label 选项，产生多个标签。

17.10.3 The Pin Option

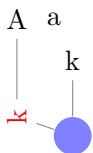
/tikz/pin=[$\langle options \rangle$] $\langle angle \rangle$: $\langle text \rangle$

这个选项与 label 选项类似，用作 node 的选项，给该 node 添加一个作为标签的 node，所加的标签像大头针。 $\langle angle \rangle$ 确定 main node 坐标系中的点，是大头针的针尖指向的位置，针后端是与 $\langle angle \rangle$ 相应的 label node 的锚位置。

可以给一个 node 加多个 pin 选项，产生多个标签。

pin 选项可以与 pin 选项套嵌使用，也可以与 label 选项套嵌使用。

如果 pin 带有 rotate 选项，标签会围绕它的指向点 ($\langle angle \rangle$ 确定 main node 上的点) 整体 (包括它的内容) 旋转，但是“针”并未必能获得正确的方位。



```
\tikz \node [circle,fill=blue!50,minimum size=0.5cm,
  pin=90:k,
  pin={[text=red,rotate=90,
    pin={[absolute,pin distance=5ex,
      label=right:a]90:A}]
    90:k}] {};
```

`/tikz/pin distance=<distance>` (无默认值, 初始值 `3ex`)

设置 pin 标签的锚位置与其指向点的距离。

`/tikz/every pin` (`style`, 初始值 `draw=none,fill=none`)

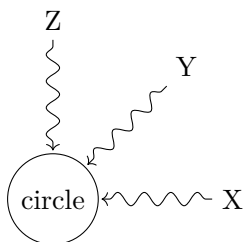
设置每个 pin 标签的样式。

`/tikz/pin position=<angle>` (无默认值, 初始值 `above`)

类似 label position.

`/tikz/every pin edge` (`style`, 初始值 `help lines`)

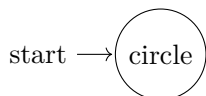
设置每个 pin 标签的“针”，即“边”的样式。



```
\tikz [pin distance=15mm,
  every pin edge/.style={<- ,shorten <=1pt,decorate,
  decoration={snake,pre length=4pt}}]
\node [circle,draw,
  pin=right:X,
  pin=above right:Y,
  pin=above:Z] {circle};
```

`/tikz/pin edge=<options>` (无默认值, 初始值 `empty`)

设置 pin 标签的“针”，即“边”的样式。注意 pin 选项中的颜色、线宽、线型等选项只是针对大头针的“头”的设置，“针”即边的样式默认为 help lines，如果需要其它样式就得用这个选项单独设置。



```
\tikz [every pin edge/.style={},
  initial/.style={pin={[pin distance=5mm,
  pin edge={<- ,shorten <=1pt}]left:start}}]
\node [circle,draw,initial] {circle};
```

17.10.4 引用句法

label 和 pin 选项可以为 node 创建标签，但是句法稍微繁琐。下面介绍相对简捷一些的引用句法。首先调用 quotes 程序库，然后就可以在 label, pin, edge node, pic 等等中使用引用句法。引用句法不使用 `<key>=<value>` 这种赋值句式，其句式为：

"<text>"<options>

程序会检查符号串 (string) 是否以双引号开头 (TikZ 的 key 不以双引号开头), 若是则确认为引用句法。

如果 <options> 中有逗号, 就必须用花括号把整个 <options> 括起来, 否则可以不加花括号 (选项前可以加空格)。在 <options> 中只能使用 above, left, below right 等等平移位置来确定标签相对于 main node 的方位, 不能使用 north, east, south west 等罗盘位置, 也不能使用表示角度的数字, 例如 "\$60^\circ\$" 60 会导致错误:


! Package pgfkeys Error: I do not know the key '/tikz/60' and I am going to ignore it.

程序会把平移位置转换为角度选项, 例如将 above 转换为 label position=90.

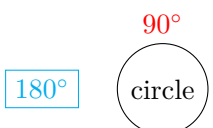
上面的句式会被转换为

label={ [<options>] < 平移位置 > : <text> }

所以, 可用于 label 选项的选项也可以用于引用句式。

 `\tikz \node [draw, "label" {red,draw,thick,below, label distance=3mm}] {E};`

不过 north, east, south west 等罗盘位置, 以及表示角度的数字可以 <direction>:<actual text> 的形式用在 <text> 中。

 `\tikz \node ["90:90°" red, "west:180°" {cyan,draw,label distance=5mm}, circle, draw] {circle};`


`/tikz/quotes mean label` (无值)

这个选项的字面意思是: 引用内容意味着 label, 就是将引用句法转换为 label 选项的句法。当调用 quotes 程序库后, 这个选项是默认使用的。如果不调用这个程序库, 可以给某个 node 加上这个选项, 这个选项会使得程序对该 node 启动引用句法功能并执行之。


`/tikz/every label quotes` (style, 无值)

为 (等效于 label 选项的) 引用句法产生的 node 设置样式。

因为程序把逗号当作分隔两个 key 的标志, 所以如果 <text> 中含有逗号, 就必须用花括号把逗号括起来, 或者用花括号把整个 <text> 括起来。

`yes, we can`
 `\tikz [red]\node [{"yes, we can"}, draw] {foo};`

冒号可以用于极坐标, 如 (30:1), 或标签, 如 <angle>:<text>, 冒号的作用是将方向、方位与其它内容分隔开来。所以如果 <text> 中含有冒号, 就必须用花括号把冒号括起来, 或者用花括号把整个 <text> 括起来。

`yes: we can`
 `\tikz \node [red, "yes{:} we can", draw] {foo};`

在 "<text>" 之后可以加一个单引号 (撇号), 它等效于选项 swap, 撇号与其它选项的衔接句式如下:

```
"foo"' 等效于 "foo" {'}
"foo"' red 等效于 "foo" {'red}
"foo"'{red} 等效于 "foo" {'red}
"foo"{'red} 等效于 "foo" {'red}
"foo"{}{red,'} 等效于 "foo" {red,'}
"foo" {'red} 是非法句式，因为程序会把 "red" 当作一个选项
"foo" {red'} 是非法句式，因为程序会把 "red'" 当作一个选项
```

`/tikz/quotes mean pin` (无值)

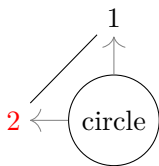
作用类似于 `quotes mean label`，这个选项的字面意思是：引用内容意味着 `pin`，就是将引用句法转换为 `pin` 选项的句法，故可用于 `pin` 选项的选项也可以用于引用句法。

`/tikz/every pin quotes` (`style`, 无值)

为（等效于 `pin` 选项的）引用句法产生的 `node` 设置样式。

`/tikz/node quotes mean=<replacement>` (无默认值)

`<replacement>` 是一组 `key` 设置，其中包含 `#1` 和 `#2` 两个变量。作出这个设置后，引用句法 `"<text>"<options>` 的实际效果将由该设置决定，其中 `<text>` 对应 `#1`，`<options>` 对应 `#2`，换言之，这个设置就是引用句法的定义。程序会使用 `\pgfkeys` 来解析 `<replacement>`。



```
\tikzset{node quotes mean={pin={[#2,
    pin distance=0.5cm,
    pin edge={->[scale=2]}},
    name={#1}]#1}}}
\tikz {
  \node ["1", "2" {red,pin position=left},
    circle, draw] {circle};
  \draw (1) -- (2);
}
```

17.11 Connecting Nodes: Using Nodes as Coordinates

当创建一个 `node` 并为之命名后，就会有与此名称对应的 `node` 坐标系，参考 §13.2.3。

17.12 Connecting Nodes: 用 `edge` 算子

17.12.1 `edge` 算子的基本句法

`edge` 算子的用法类似 `to` 算子和 `node` 算子，`edge` 算子会抑制当前主路径的构建并构建一个新的路径，在画出主路径后，再画出这个新路径。`edge` 算子可以带有诸多选项来设定其所创建的路径样式。

如果主路径上有多个 `to` 算子和 `node` 算子，它们创建的路径会被依次画出。

`edge` 算子的基本句法是：

```
\path . . . edge[<options>] <nodes> (<coordinate>) . . . ;
```

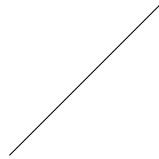
edge 算子的实际效果是在画出主路径后执行下面的语句模式：

```
\path[every edge,<options>] (\tikztostart) <to path>;
```

注意其中的 <to path> 部分是选项 to path 定义的路径样式（参考 §14.13），也就是 edge 算子所画出的路径样式。上面的模式中，(\tikztostart) 是 edge 算子之前的坐标点，这与 to 算子和 node 算子一样，但有一个不同，如果 edge 算子之前是个 node 语句，那么这个 edge 算子就以该 node 为开始坐标。在默认下，选项 to path 定义的路径样式是

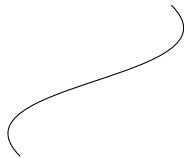
```
--(\tikztotarget) \tikztonodes
```

这个路径样式是个直线段



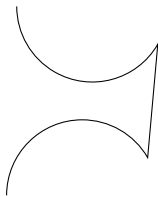
```
\begin{tikzpicture}
  \path (0,0) edge (2,2);
\end{tikzpicture}
```

可以将 to path 定义为控制曲线

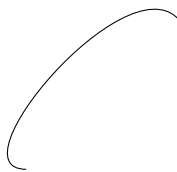


```
\begin{tikzpicture}
  [to path={.. controls +(-1,1) and
    +(1,-1) .. (\tikztotarget) \tikztonodes}]
  \path (0,0) edge (2,2);
\end{tikzpicture}
```

再如



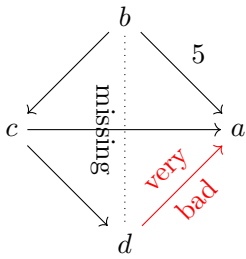
```
\begin{tikzpicture}
  \path (0,0) edge[to path={arc(180:30:1)--(\tikztotarget)
    arc(-30:-180:1) \tikztonodes}]
    (2,2);
\end{tikzpicture}
```



```
\begin{tikzpicture}[control/.style 2 args={to path=
  {.. controls +(#1) and +(#2) ..
    (\tikztotarget) \tikztonodes}}]
  \path (0,0) edge[control={-1,0}{135:1}] (2,2);
\end{tikzpicture}
```

如果一行之中有多个 edge 算子，那么这些 edge 算子的起始点相同，即第一个 edge 算子之前的点，但目标点不同。这一点与 node 算子类似。

edge 算子接受主路径的选项，但也可以在 edge 算子处作局部修改。



```

\begin{tikzpicture}
\node foreach \name/\angle in {a/0,b/90,c/180,d/270}
  (\name) at (\angle:1.5) {$\name$};
\path[->] (b) edge node[above right] {$5$} (a)
  edge (c)
  edge [-,dotted] node[below,sloped] {missing} (d)
  (c) edge (a)
  edge (d)
  (d) edge [red] node[above,sloped] {very}
  node[below,sloped] {bad} (a);
\end{tikzpicture}

```

`/tikz/every edge` (style, 初始值 draw)

用作环境或主路径的选项，设置每个 edge 路径的样式。

17.12.2 Nodes on Edges: Quotes Syntax

给 edge 路径加标签的方法有：

- 在 edge 之后、目标点之前加 node 语句。
- 给 edge 算子加 edge node, edge label 选项。
- 调用 quotes 程序库，使用引用句法选项，参考 §17.10.4。

引用句法选项

`"<text>"<options>`

会被转换为如下选项

`edge node=node [every edge quotes]<options>{<text>}`

其中的 every edge quotes 是个样式设置选项：

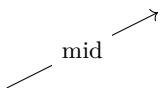
`/tikz/every edge quotes` (style, 初始值 auto)

这个选项的初始值是 auto，它规定引用句法创建的标签位于 edge 路径的左侧。可以用 auto=right 或用带撇号的引用句法来转换标签位置。



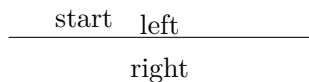
```
\tikz \draw (0,0) edge ["right"{'},red], ->] (2,0);
```

如果给出该选项但选项值不包含 auto 设置，则引用句法创建的标签位于 edge 路径中间。



```
\tikz [every edge quotes/.style={fill=white,font=\small}]
\draw (0,0) edge ["mid", ->] (2,1);
```

可以使用自定义样式：



```
\tikz [tight/.style={inner sep=1pt},
      loose/.style={inner sep=.7em}]
\draw (0,0) edge ["left" tight,
                 "right" loose,
                 "start" near start] (4,0);
```

17.13 Referencing Nodes Outside the Current Picture

17.13.1 Referencing a Node in a Different Picture

通常，一个 `{tikzpicture}` 环境创建一个图片，一个图片内的 node 坐标系中的点只能在本图片内引用。通过适当操作可以实现跨图引用 node 坐标系中的点。

如果要在图形 G_2 中引用图形 G_1 中的点（例如，将 G_2 中的某个点与 G_1 中的某个点用线连起来），那么首先要让程序记住这两个图形中的点在页面中的位置，然后才能跨图引用这些点。当给 G_1 和 G_2 的环境选项都加上选项 `remember picture` 后，程序会记住图形中的点在页面中的位置，这需要后台驱动的支持（如果驱动不支持就不能实现这一点），并且还需要运行 \TeX 两次才能引用这些点。

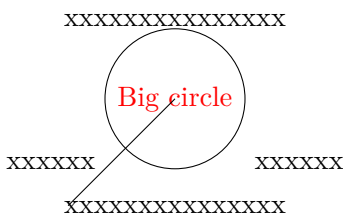
当在图形 G_2 中引用图形 G_1 中的点时，TikZ 总是会试图扩大图形 G_2 的边界盒子（bounding box）以包含图形 G_1 中被引用的点。这样图形 G_2 就可能变得很大以致页面比较难看。这就需要给图形 G_2 的环境选项加 `overlay` 选项，或者给图形 G_2 中包含（图形 G_1 中的）被引用点的子环境或路径加 `overlay` 选项。

如果图形的环境选项中有 `overlay` 选项，那么在计算该图形的边界盒子时，`overlay` 选项会让程序忽略该环境内的所有内容。



```
xxxxxxxxxxxxxxxxxxx \par
xxxxxxx
\begin{tikzpicture}[overlay]
  \node (c) [circle,draw,text=red] {Big circle};
\end{tikzpicture}
xxxxxxx \par
xxxxxxxxxxxxxxxxxxx
```

如果图形中的某个路径有 `overlay` 选项，那么在计算该图形的边界盒子时，`overlay` 选项会让程序忽略该路径的内容。



```
\begin{center}
xxxxxxxxxxxxxxxxxxx \par
```

```

xxxxxx
\begin{tikzpicture}
  \node (c) [circle,draw,text=red] {Big circle};
  \draw [overlay](0,0)--(-135:2);
\end{tikzpicture}
xxxxxx \par
xxxxxxxxxxxxxxxxxxxx
\end{center}

/tikz/remember picture=<boolean>      (无默认值, 初始值 false)
\tikzstyle{every picture}+=[remember picture]

```

这个设置会让 TikZ 记住所有图形在页面中的位置。

```

/tikz/overlay=<boolean>      (默认值 true)

```



```

\tikz[remember picture]
  \node[circle,fill=red!50] (n1) {};

```

```

\tikz[remember picture]
  \path(0,0) (1,0)node[fill=blue!50] (n2) {};

```

```

\begin{tikzpicture}[remember picture,overlay]
  \draw[->,very thick] (n1) -- (n2);
\end{tikzpicture}

```

```

\begin{tikzpicture}[remember picture]
  \node (c) [circle,draw] {Big circle};
  \draw [overlay,->,very thick,red,opacity=.5]
    (c) to[bend left] (n1) (n1) -| (n2);
\end{tikzpicture}

```

还要注意，在跨图引用 node 时，所涉及的各个图形应当在同一个页面内，否则结果会很意外。

17.13.2 引用 Current Page Node——绝对位置

有个名称是 current page 的预定义的特殊 node，它对应当前页面，它的形状是矩形，它的锚位置 south west 是当前页面的左下角，它的锚位置 north east 是当前页面的右上角。给环境加上 remember picture 和 the overlay 选项后，就可以在该环境中引用当前页面中的位置。

下面的代码在当前页面的左下角添加文字：

```

\begin{tikzpicture}[remember picture,overlay]
  \node at (current page.south west)

```

```
[text width=7cm,fill=red!20,rounded corners,above right]
{
This is an absolutely positioned text in the
lower left corner. No shipout-hackery is used.
};
\end{tikzpicture}
```

下面的代码在当前页面的中心添加文字:

```
\begin{tikzpicture}[remember picture,overlay]
\node [rotate=60,scale=10,text opacity=0.2]
at (current page.center) {Example};
\end{tikzpicture}
```

17.14 Late Code and Late Options

```
\path . . . node also[<late options>](<name>) . . . ;
```

假设已经创建了名称为 <name> 的 node 对象, 在之后的编辑过程中发现需要对 <name> 做某些修改, 此时就可以用这个句法。这个句法中, 算子 `node also` 之后是方括号选项, 再后是之前创建的 node 的名称 <name>, 这个次序不能变, 注意没有内容 <text>, 方括号里的选项会作用于 <name>, 例如, `label`, `append after command`, `prefix after`, 但注意 <name> 的多个特性 (形状、颜色等) 都不能用这个句法修改。

world



```
\begin{tikzpicture}
\node [draw,circle] (a) {Hello};
\node also [label=above:world] (a);
\end{tikzpicture}
```

```
\tikzlastnode
```

```
/tikz/late options=<options>
```

这个选项的意思是“迟到的选项”, 即“补充的选项”, 它只能用作路径选项, 不能用作 node 选项. 这个选项也能实现算子 `node also` 的功能:

world



```
\begin{tikzpicture}
\node [draw,circle] (a) {Hello};
\path [late options={name=a, label=above:world}];
\end{tikzpicture}
```

18 Pics: Small Pictures on Paths

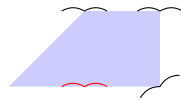
18.1 Overview

node 对象的形状（如矩形、圆形、梯形、云状等等）是预定义的，有时候需要在路径上添加一个自定义形状，而不是预定义的某个形状，此时可以用 `pic` 语句。所要做的就是先用 TikZ 代码定义一个图形，然后在路径上用 `pic` 语句插入这个图形。

凡是可以用 node 语句的地方都可以用 `pic` 语句，适用于 node 算子的绝大多数选项（key）也都适用于 `pic` 算子。node 对象的形状由单词 `shape` 来描述，`pic` 对象的形状由单词 `type` 来描述。`pic` 对象是添加到路径上的图形，它（与 node 对象一样）不属于路径，因此路径选项中的某些 key 对该路径上的 `pic` 图形无效。

与 node 对象不同的是，`pic` 对象不能被引用，但可以引用 `pic` 图形中的 node 对象。

下面是个例子：



```
{\tikzset{ % 下面的代码定义一个海鸟形状的图形，其名称为 seagull
  seagull/.pic={
    \draw (-3mm,0) to [bend left] (0,0) to [bend left] (3mm,0);
  }
}
\tikz \fill [fill=blue!20]
  (1,1) % 下面用 pic 语句插入 seagull
  -- (2,2) pic {seagull}
  -- (3,2) pic {seagull}
  -- (3,1) pic [rotate=30] {seagull}
  -- (2,1) pic [red] {seagull};
}
```

18.2 The Pic Syntax

`\pic`

在 `{tikzpicture}` 环境中，这是 `\path pic` 的简写。

```
\path . . . pic<foreach statements>[<options>](<prefix>)at(<coordinate>){<pic type>} . .
. ;
```

`pic` 语句与 `node` 语句非常类似，程序会把“`pic`”与“`{`”之间的 4 个部分看作是属于该 `pic` 算子的，这 4 个部分都是可有可无的（视情况而定），各部分的次序如同 `node` 语句，`<foreach statements>` 必须放在 `pic` 之后（如果有的话）。

下面解释可用于 `[<options>]` 的选项。

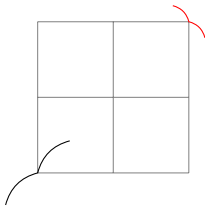
18.2.1 指定 pic 图形的名称

`/tikz/pic type=<pic type>`

这个选项指定预定义的 pic 形状,与选项 `node contents` 类似,使用这个选项后就不必写出 `{<pic type>}` 这一部分。

18.2.2 指定 pic 图形的位置

pic 图形所指向的坐标 (点), 或者是 pic 之前的点, 或者是由 `at(<coordinate>)` 这一部分指定, 或者是用选项 `at=<coordinate>` 指定, pic 图形的指向点与 pic 图形的原点相同。注意 pic 算子的变换选项, 如 `shift` 选项, 会影响 pic 图形的原点与其它点的相对距离, 从而影响 pic 图形的尺寸。



```
{\tikzset{
  seagull/.pic={
    \draw (-3mm,0) to [bend left] (0,0) to [bend left] (3mm,0);
  }
}
\tikz {
  \draw [help lines](0,0) grid (2,2);
  \pic [red,shift={(1,1)},rotate=-45]at(1,1){seagull};
  \pic [at={(0,0)},rotate=45,scale=2]{seagull};
}
}
```

与 `node` 算子一样, 如果 pic 算子带有 `transform shape` 选项, 则 pic 图形接受路径选项的作用。可以在一个点之后使用多个 pic 语句, 它们会依次画出。可以给 pic 算子使用 `sloped`, `pos`, `near end`, `near start` 等选项来调整 pic 图形在路径上的位置。

18.2.3 定义 pic 图形

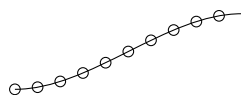
`/tikz/pics/code=<code>`

这个选项用于存储画 pic 图形的代码。画 pic 图形的代码可以由 `<pic type>` 定义, 也可以作为这个选项的值直接用在 pic 的选项中。



```
\tikz \pic [pics/code={\draw (-3mm,0) to[bend left] (0,0)
  to[bend left] (3mm,0);} ] {};
```

`<pic type>` 所定义的是个 key 列表, 各个 key 的前缀是 `/tikz/pics/`, 原则上, key 可以是任何可以被命令 `\pgfkeys` 解析的定义。前面的例子中只定义了一个 key, 即 `seagull`, 当程序处理 `pic{seagull}` 时, 程序会执行 `/tikz/pics/seagull`, 这与选项 `code={\draw(-3mm,0)...}` 等价。



```
\tikz \draw (0,0) .. controls(1,0) and (2,1) .. (3,1)
  foreach \t in {0, 0.1, ..., 1} {
    pic [pos=\t] {code={\draw circle [radius=2pt];}}
  };
```

18.2.4 pic 的选项的传递

如前面的例子所示，pic 选项中的颜色选项对 pic 图形有效，即能传递给定义 pic 图形的代码，但 pic 选项中的 fill, draw, shading, clip 等选项对 pic 图形无效，也就是说这些选项不能传递给定义 pic 图形的代码。修改一下上面的例子：

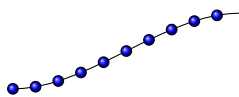


```
\tikz \draw (0,0) .. controls(1,0) and (2,1) .. (3,1)
  foreach \t in {0, 0.1, ..., 1} {
    pic [draw,fill,pos=\t] {
      code={\path circle [radius=2pt];}
    };
```

可见，修改后路径上的圆圈没有画出。为了能够让 pic 选项中的 fill, draw, shading, clip 等选项对 pic 图形有效，需要在定义 pic 图形的代码中的路径语句里添加下面的选项：

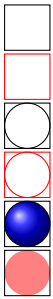
`/tikz/pic actions` (无值)

例如上一个图可以这样画：



```
\tikz \draw (0,0) .. controls(1,0) and (2,1) .. (3,1)
  foreach \t in {0, 0.1, ..., 1} {
    pic [draw,shading=ball,pos=\t] {
      code={\path [pic actions]circle [radius=2pt];}}
  };
```

再如



```

\tikzset{
  my pic/.pic = {
    \path [pic actions] (0,0) circle[radius=3mm];
    \draw (-3mm,-3mm) rectangle (3mm,3mm);
  }
}
\tikz \pic {my pic}; \par
\tikz \pic [red] {my pic}; \par
\tikz \pic [draw] {my pic}; \par
\tikz \pic [draw=red] {my pic}; \par
\tikz \pic [draw, shading=ball] {my pic}; \par
\tikz \pic [fill=red!50] {my pic};

```

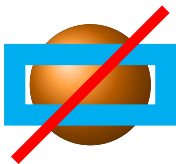
18.2.5 指定 pic 图形的遮挡次序

与 node 类似，pic 算子可以带有 behind path 或 in front of path 选项。此外，还可以使得 pic 图形的某一部分遮挡另一部分，这需要下面的 key:

`/tikz/pics/foreground code=<code>` (无值)

`/tikz/pics/background code=<code>` (无值)

各类绘图代码仍然按次序画出，只不过，foreground code 存储的代码（即其中有 behind path 选项）所绘出的图形将遮挡 background code 存储的代码所绘出的图形。也就是说，这两个选项的行为仅仅限于 pic 图形的内部。



```

\tikzset{
  my pic/.pic = {
    background code={\fill [ball color=orange]
      (0,0) circle[radius=8mm];}
    foreground code={\draw [line width=8pt,cyan]
      (-10mm,-4mm) rectangle (10mm,4mm);}
    \draw [line width=4pt,red](-1,-1)--(1,1);
  }
}
\tikz \pic {my pic};

```

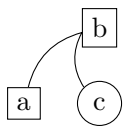
18.2.6 设置每个 pic 图形的样式

`/tikz/every pic` (style, 初始值 empty)

这个选项将用于每个 pic 图形的构建过程的开头。

18.2.7 设置 pic 图形中的 node 名称的前缀并引用它

pic 图形是不能被引用的，故不能像 node 那样为它定义名称，但是 pic 图形中的 node 是可以被引用的。如果 pic 图形中的 node 有名称，可以用这个名称直接引用它。可以为 pic 图形中的 node 的名称加前缀，有 3 种加前缀的方式：第一，在 pic 语句中可以使用 (<prefix>); 第二，可以给 pic 添加 name=<name> 选项来定义 node 的名称前缀；这两种方式实际上都会启用选项 name prefix 来为这个 pic 图形中的 node 定义名称的前缀；第三，可以为 pic 图形中的 node 加 name prefix 选项来直接定义这个 node 的名称前缀，这个方式定义的前缀比前 2 种方式定义的前缀具有优先地位。当 pic 图形中的 node 有了名称前缀后，若要在这个 pic 图形之外引用其中的 node，就需要带有这个前缀。



```
\tikz{
\pic (x) [pics/code={
  \node [draw] (a) at(0,0) {a};
  \node [draw,name prefix=y] (b) at(1,1) {b};}] {};
\pic [name=z,pics/code={\node [circle,draw](c) {c};}] at(1,0) {};
\draw (xa) to [bend left] (yb) to [bend right] (zc);
}
```

显然，如果要多次使用同一个 pic 图形并且要引用其中的 node，给各个 pic 图形分别加前缀名称会便于引用。当为 pic 图形中的 node 的名称加前缀时，可能会遇到这种复杂一些的情况：定义 pic 图形的代码中有 node，且在代码中又引用了这个 node，例如，

```
\tikzset{
  seagull/.pic={
    \coordinate (-left wing) at (-3mm,0);
    \coordinate (-head) at (0,0);
    \coordinate (-right wing) at (3mm,0);
    \draw (-left wing) to [bend left] (0,0)
      (-head) to [bend left] (-right wing);
  }
}
```

如果是用上面的代码定义 pic 图形 seagull 的，那么下面的语句

```
\tikz {
  \pic (Emma) {seagull};
  \pic (Alexandra) at (0,1) {seagull};
  \draw (Emma-left wing) -- (Alexandra-right wing);
}
```

会导致错误：

```
! Package pgf Error: No shape named EmmaEmma-left wing is known.
```

此时就需要用前面提到的第 3 种加前缀的方式来避免错误。

18.2.8 引用句法

载入 quotes 程序库后, 可在 pic 的选项中使用引用句法:

```
"<text>"'<options>
```

这一串字符将被转换为

```
every pic quotes/.try, pic text=<text>, pic text options={<options>}
```

其中的 /.try 参考 §2.4.8, 另外两个选项解释如下:

```
/tikz/pic text=<text>      (无值)
```

这个选项会把 <text> 储存在宏 \tikzpictext 之中, 这个宏由 \let 命令定义, 默认为 \relax.

```
/tikz/pic text options=<options>      (无值)
```

这个选项会把 <options> 储存在宏 \tikzpictextoptions 之中, 这个宏由 \let 命令定义, 默认为空。

```
/tikz/every pic quotes      (style, 初始值 empty)
```

有个用于描绘角度的预定义 pic 图形, 其名称为 angle, 参考 angles 程序库。

18.3 定义 pic type

前面已经有例子展示如何定义 pic type, 即定义 pic 图形。除了用 pic 算子的 code 选项外, 还可以使用“手柄”定义 pic 图形, 主要用到与 style 和 code 有关的手柄, 例如, 与 style 有关的手柄如下:

```
\tikzset{
  pics/seagull/.style ={
    code = { %代码例如
      \draw (...) ... ;
    }
  }
}
```

注意其中 /.style 之前必须使用 pics/<key name> 这种格式 (这是键路径, 参考 §2), <key name> 是所定义的 pic 图形的名称, 在这里名称是 seagull. 还要注意其中用到 code={ } 来约束绘图代码, 代码中可以含有 foreground code, background code 等, 最多可以使用 1 个变量。如果需要在代码中使用 2 个或更多变量, 可以用下面的句式:

```
<key>/.style 2 args={<key list>}
```

这个句式允许使用 2 个变量, 将 code={代码} 用于 <key list>.

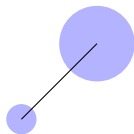
```
<key>/.style args={<argument pattern>}{<key list>}
```

这个句式至多允许使用 9 个变量, 需要声明给变量赋值的样式。

```
<key>/.style n args={<argument count>}{<key list>}
```

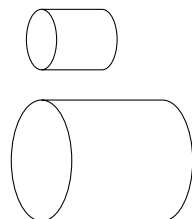
这个句式至多允许使用 9 个变量, 需要声明变量个数。

如果用以上手柄定义 pic 的代码中含有变量, 那么使用该 pic 图形时, 给图形名称赋值, 也就使得变量被赋值, 如下面的例子所示:



```
\tikzset{
  pics/my circle/.style = {
    background code = { \fill circle [radius=#1]; }
  }
}
\tikz [fill=blue!30]
\draw (0,0) pic {my circle=2mm} -- (1,1) pic {my circle=5mm};
```

在下面的例子中，定义 pic 图形的手柄用了 `/.style 2 args`，其中用了 `code={}` 来约束绘图代码，而且在命令 `\draw` 的选项中使用了 `pic actions` 选项，这是为了使得选项 `scale=` 有效：



```
\begin{tikzpicture}
\tikzset{
  pics/cylinder/.style 2 args={
    code={
      \draw [pic actions](0,0) arc (-90:90:1 and #1) coordinate(coord1)--(#2,#1+#1)
        arc (90:-90:1 and #1)--cycle;
      \draw [pic actions](0,0) arc (270:90:1 and #1);}
  }
}
\draw (0,0) pic [scale=0.2]{cylinder={2}{4}};
\draw (0,-2) pic [scale=0.4]{cylinder={2}{4}};
\end{tikzpicture}
```

对比上面的例子，如下代码

```
\tikzset{
  pics/cylinder/.code 2 args={
    \draw [pic actions](0,0) arc (-90:90:1 and #1) coordinate(coord1)--(#2,#1+#1) arc
      (90:-90:1 and #1)--cycle;
    \draw [pic actions](0,0) arc (270:90:1 and #1);
  }
}
```

其中的 `pic actions` 选项无效，所以为了使用这个选项，应该使用 `style` 手柄。

当然本节一开的例子中使用的 `/.pic` 也是一个手柄，

```
\tikzset{
```

`seagull/.pic = { %代码例如`

`\draw (...) ... ;`

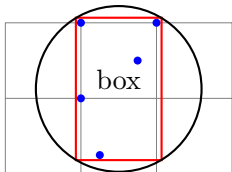
这种定义句式叫作“key handler”，参考 §82。

52 fit 程序库

调用程序库 `\usetikzlibrary{fit}`，用这个程序库提供一种方法，能够把已创建的一个或数个坐标点，或 nodes 放入另一个 node 中（为了方便称这个 node 为 fit node）。

`/tikz/fit=<coordinates or nodes>`

这个选项只能用作 node 命令或算子的选项。<coordinates or nodes> 是由坐标点或 node 名称组成的列表，列表项之间不用逗号分隔，可以用空格分隔。使用本选项后，程序会创建一个 fit node，将 <coordinates or nodes> 中列出的坐标点和 nodes 包含在内。程序首先计算一个尺寸尽量小的盒子，将列出的坐标点以及各 nodes 的锚位置 `east`, `west`, `north`, `south` 包含在盒子内，这个盒子就是被创建的 fit node 的文字盒子。



```
\begin{tikzpicture}[inner sep=0pt,thick,
  dot/.style={fill=blue,circle,minimum size=3pt}]
  \draw[help lines] (0,0) grid (3,2);
  \node[dot] (a) at (1,1) {};
  \node[dot] (b) at (2,2) {};
  \node[dot] (c) at (1,2) {};
  \node[dot] (d) at (1.25,0.25) {};
  \node[dot] (e) at (1.75,1.5) {};
  \node[draw=red, fit=(a) (b) (c) (d) (e)] {box};
  \node[draw,circle,fit=(a) (b) (c) (d) (e)] {};
\end{tikzpicture}
```

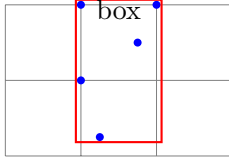
`/tikz/every fit` (style, 初始值 empty)

这是个样式，它针对所有的 fit node。

关于 fit node 注意以下几点：

1. fit node 的文字盒子包含 <coordinates or nodes> 中列出的坐标点，以及列出的 nodes 的锚位置 `east`, `west`, `north`, `south`（这 4 个锚位置是 4 个坐标点）。
2. 可以用选项 `text width` 来调节 fit node 的文字盒子的宽度。
3. fit node 的文字盒子使用对齐方式 `align=center`，这个对齐方式是固定不变的，因此不能使用选项 `align` 来改变 fit node 的文字盒子的对齐方式。
4. 可以给 fit node 使用 `at` 选项来确定它的指向点。
5. fit node 的锚位置 `center` 位于它的指向点上。
6. 程序根据 fit node 的文字盒子的内容来确定 fit node 的宽度和高度。

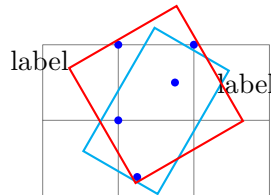
由于 fit node 的文字盒子的对齐方式是固定不变的，因此要想调整其文字盒子里的文字内容的位置就需要变通的方法，如下面的例子所示，另作一个包含文字的 node 添加到图形中：



```
\begin{tikzpicture}[inner sep=0pt,thick,
  dot/.style={fill=blue,circle,minimum size=3pt}]
\draw[help lines] (0,0) grid (3,2);
\node[dot] (a) at (1,1) {};
\node[dot] (b) at (2,2) {};
\node[dot] (c) at (1,2) {};
\node[dot] (d) at (1.25,0.25) {};
\node[dot] (e) at (1.75,1.5) {};
\node[draw=red,fit=(a) (b) (c) (d) (e)] (fit) {};
\node[below] at (fit.north) {box};
\end{tikzpicture}
```

`/tikz/rotate fit=<angle>` (无默认值, 初始值 0)

这个选项将 fit node 旋转 <angle> 角度，它的副作用是会把 /tikz/rotate 的值也设成 <angle>。注意对于 fit node 来说，选项 rotate fit 与 rotate 的作用是不同的。



```
\begin{tikzpicture}[inner sep=0pt,thick, dot/.style={fill=blue,circle,minimum size=3
  pt}]
\draw[help lines] (0,0) grid (3,2);
\node[dot] (a) at (1,1) {};
\node[dot] (b) at (2,2) {};
\node[dot] (c) at (1,2) {};
\node[dot] (d) at (1.25,0.25) {};
\node[dot] (e) at (1.75,1.5) {};
\node[draw=cyan, rotate=-30, fit=(a) (b) (c) (d) (e), label={above right:label}]
  {};
\node[draw=red, rotate fit=30, fit=(a) (b) (c) (d) (e), label={above left:label}]
  {};
\end{tikzpicture}
```

70 topaths 程序库

调用程序库 `\usetikzlibrary{topaths}`, TikZ 会自动调用这个程序库。本程序库定义了数个选项,专门用于 `to` 算子或 `edge` 算子构建的路径,见 §14.13.

70.1 直线

`/tikz/line to`

这个选项使得 `to` 算子或 `edge` 算子创建直线段。



```
\tikz {\draw (0,0) to[line to] (1,0);}
```

70.2 Move-To

`/tikz/move to`

这个选项使得 `to` 算子或 `edge` 算子执行 `move to` 操作。



```
\tikz \draw (0,0) to[line to] (1,0)
to[move to] (2,0) to[line to] (3,0);
```

70.3 曲线

`/tikz/curve to`

这个选项使得 `to` 算子或 `edge` 算子在两点之间构建一段控制曲线。下面的选项可以调节控制曲线的形态。

`/tikz/out=<angle>`

这个选项确定控制曲线在始点的方向角度,如果“出发地”是 `node`,例如 `node` 的名称是 `a`,那么始点就是点 `(a.<angle>)`。



```
\begin{tikzpicture}[out=45,in=135]
\draw (0,0) to (1,0)
(0,0) to (2,0)
(0,0) to (3,0);
\end{tikzpicture}
```

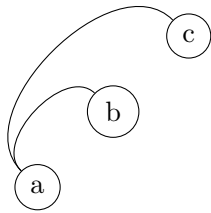
`/tikz/in=<angle>`

这个选项确定控制曲线在终点的方向角度。如果“目标地”是 `node`,例如 `node` 的名称是 `a`,那么终点就是点 `(a.<angle>)`。

`/tikz/relative=<true or false>` (默认值 `true`)

这个选项决定由选项 `in`, `out` 确定的角度是“绝对的”(absolute)还是“相对的”(relative)。假如一个坐标系的 `x` 轴在页面上是水平向右的, `y` 轴在页面上是竖直向上的, 并且这个坐标系不接受变换, 那么这个坐标系就是绝对的, 在这个坐标系内确定的角度就是“绝对的”。

以控制曲线的起点为原点, 以起点到终点的有向直线为 `x` 轴, 而 `y` 轴与 `x` 轴成右手系, 这个坐标系就是相对的, 在这个坐标系内确定的角度就是“相对的”。

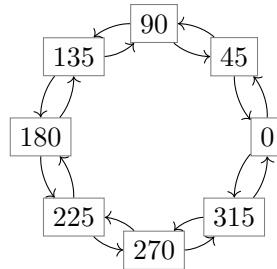


```
\begin{tikzpicture}[out=90,in=90,relative]
  \node [circle,draw] (a) at (0,0) {a};
  \node [circle,draw] (b) at (1,1) {b};
  \node [circle,draw] (c) at (2,2) {c};
  \path (a) edge (b)
        edge (c);
\end{tikzpicture}
```

`/tikz/bend left`

`/tikz/bend left=<angle>` (默认值 last value)

这个选项等效于 `out=<angle>`, `in=180-<angle>`, `relative`. 如果不明确给出 `<angle>`, 那么程序就向前查找最近使用过的选项 `bend left` 或 `bend right` 所确定的角度值, 并使用这个角度值。



```
\begin{tikzpicture}
  \foreach \angle in {0,45,...,315}
  \node[rectangle,draw=black!50] (\angle) at (\angle:1.5) {\angle};
  \foreach \from/\to in {0/45,45/90,90/135,135/180,180/225,225/270,270/315,315/0}
  \path (\from) edge [->,bend right=22,looseness=0.8] (\to)
                edge [<-,bend left=22,looseness=0.8] (\to);
\end{tikzpicture}
```

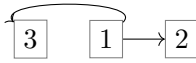
`/tikz/bend right`

`/tikz/bend right=<angle>` (默认值 last value)

类似 `bend left`.

`/tikz/bend angle=<angle>`

这个选项声明一个角度 `<angle>`, 如果选项中还有 `bend left` 或 `bend right`, 就把角度 `<angle>` 作为这两个选项的角度值。注意本选项只是声明一个角度, 不会引入选项 `curve to` 或 `relative`.



```

\begin{tikzpicture}
  \node[rectangle,draw=black!50] (1) {1};
  \node[rectangle,draw=black!50] (2) at(1,0) {2};
  \node[rectangle,draw=black!50] (3) at(-1,0) {3};
  \path (1) edge [->,bend angle=30] (2)
           edge [->,bend angle=30,looseness=0.8] (3);
\end{tikzpicture}

```

上面例子中，选项 `looseness=0.8` 引入 `curve to` 操作。

`/tikz/looseness=<number>` (无默认值，初始值 1)

这个选项确定控制曲线的“松弛程度”。`<number>` 是个实数，它的值越大，控制曲线就越“圆满”。TikZ 会根据某个固定系数 s 以及 `<number>` 来选择控制曲线的支撑点，从而调整控制曲线的形状。假设选项 `out=<ang1>`，`in=<ang2>`，控制曲线的起点是 P ，终点是 Q ，那么第一个支撑点就是

$$P + s \cdot \langle \text{number} \rangle \cdot |PQ| \cdot (\langle \text{ang1} \rangle : 1),$$

第二个支撑点就是

$$Q + s \cdot \langle \text{number} \rangle \cdot |PQ| \cdot (\langle \text{ang2} \rangle : 1).$$

当 `looseness=1` 且 `in`，`out` 的角度值相差 90° 时，得到的控制曲线是 $\frac{1}{4}$ 圆弧。



```

\tikz \draw (0,0) to [out=0,in=-90] (1,1);
\tikz \draw (0,0) to [out=0,in=-90,looseness=0.5] (1,1);

```

上面例子中第一个命令绘制一段 $\frac{1}{4}$ 圆弧，这段圆弧的 4 个控制点是 $(0,0)$ ， $(0.5,0)$ ， $(1,0.5)$ ， $(1,1)$ ，因此固定系数 $s = \frac{\sqrt{2}}{4}$ 。

`/tikz/out looseness=<number>`

假设选项 `out=<ang1>`，控制曲线的起点是 P ，终点是 Q ，那么第一个支撑点就是

$$P + s \cdot \langle \text{number} \rangle \cdot |PQ| \cdot (\langle \text{ang1} \rangle : 1).$$

`/tikz/in looseness=<number>`

假设选项 `in=<ang2>`，控制曲线的起点是 P ，终点是 Q ，那么第一个支撑点就是

$$Q + s \cdot \langle \text{number} \rangle \cdot |PQ| \cdot (\langle \text{ang2} \rangle : 1).$$

`/tikz/min distance=<distance>`

假设选项 `out=<ang1>`，`in=<ang2>`，控制曲线的起点是 P ，终点是 Q ，如果 $s \cdot \langle \text{number} \rangle \cdot |PQ|$ 小于 `<distance>`，就用 `<distance>` 替换 $s \cdot \langle \text{number} \rangle \cdot |PQ|$ ，也就是说，第一个支撑点是

$$P + \langle \text{distance} \rangle \cdot (\langle \text{ang1} \rangle : 1),$$

第二个支撑点是

$$Q + \langle \text{distance} \rangle \cdot (\langle \text{ang2} \rangle : 1).$$

`/tikz/max distance=<distance>`

假设选项 `out=<ang1>`, `in=<ang2>`, 控制曲线的起点是 P , 终点是 Q , 如果 $s \cdot \langle \text{number} \rangle \cdot |PQ|$ 大于 `<distance>`, 就用 `<distance>` 替换 $s \cdot \langle \text{number} \rangle \cdot |PQ|$, 也就是说, 第一个支撑点是

$$P + \langle \text{distance} \rangle \cdot (\langle \text{ang1} \rangle : 1),$$

第二个支撑点是

$$Q + \langle \text{distance} \rangle \cdot (\langle \text{ang2} \rangle : 1).$$

`/tikz/out min distance=<distance>`

假设选项 `out=<ang1>`, 控制曲线的起点是 P , 终点是 Q , 在确定第一个支撑点时, 如果 $s \cdot \langle \text{number} \rangle \cdot |PQ|$ 小于 `<distance>`, 就用 `<distance>` 替换 $s \cdot \langle \text{number} \rangle \cdot |PQ|$, 也就是说, 第一个支撑点是

$$P + \langle \text{distance} \rangle \cdot (\langle \text{ang1} \rangle : 1).$$

`/tikz/out max distance=<distance>`

假设选项 `out=<ang1>`, 控制曲线的起点是 P , 终点是 Q , 在确定第一个支撑点时, 如果 $s \cdot \langle \text{number} \rangle \cdot |PQ|$ 大于 `<distance>`, 就用 `<distance>` 替换 $s \cdot \langle \text{number} \rangle \cdot |PQ|$, 也就是说, 第一个支撑点是

$$P + \langle \text{distance} \rangle \cdot (\langle \text{ang1} \rangle : 1).$$

`/tikz/in min distance=<distance>`

类似 `out min distance`, 只是针对第二支撑点。

`/tikz/in max distance=<distance>`

类似 `out max distance`, 只是针对第二支撑点。

`/tikz/distance=<distance>`

本选项同时设置 `min distance=<distance>` 和 `max distance=<distance>`.

`/tikz/out distance=<distance>`

本选项同时设置 `out min distance=<distance>` 和 `out max distance=<distance>`.

`/tikz/in distance=<distance>`

本选项同时设置 `in min distance=<distance>` 和 `in max distance=<distance>`.

`/tikz/out control=<coordinate>`

本选项直接指定第一个支撑点, 这里 `<coordinate>` 可以使用相对坐标形式, 例如 `+(1,1)`, 这里的相对坐标相对于起点计算。

`/tikz/in control=<coordinate>`

本选项直接指定第二个支撑点, 这里 `<coordinate>` 可以使用相对坐标形式, 例如 `+(1,1)`, 这里的相对坐标相对于终点计算。

`/tikz/controls=<coordinate1> and <coordinate2>`

本选项直接指定两个支撑点。如果 `<coordinate1>` 是相对坐标形式，则它相对于起点计算。如果 `<coordinate2>` 是相对坐标形式，则它相对于终点计算。



```
\tikz \draw (0,0) to [controls=+(90:1) and +(90:1)] (2,0);
```

70.4 Loops

`/tikz/loop`

本选项类似 `curve to`，也构建一段控制曲线，不同的是：（1）本选项确定的控制曲线的起点与终点重合，因此本选项构建的控制曲线像是一个“环”（loop），而且在使用本选项时，代码中的“终点”只需要用一对圆括号代表；（2）每当使用本选项时，都会使用固定选项值 `looseness=8`，`min distance=5mm`，也就是说，当使用本选项时这两个选项值不能改变（对于这两个固定选项值来说，当 `out`，`in` 的角度相差 30° 时，得到的控制曲线较美观。）



```
\begin{tikzpicture}
  \node [circle,draw] {a} edge [in=30,out=60,loop] ();
\end{tikzpicture}
```

`/tikz/loop above` (style)

这是个样式 (style)，这个样式创建的环（控制曲线）总是位于起点（终点）的上方，并且，如果给这个环添加 node 标签，那么该标签会位于环的上方。



```
\begin{tikzpicture}
  \node [circle,draw] {a} edge [loop above] node {x} ();
\end{tikzpicture}
```

`/tikz/loop below` (style)

这是个样式 (style)，这个样式创建的环（控制曲线）总是位于起点（终点）的下方，并且，如果给这个环添加 node 标签，那么该标签会位于环的下方。

`/tikz/loop left` (style)

这是个样式 (style)，类似样式 `loop above`。

`/tikz/loop right` (style)

这是个样式 (style)，类似样式 `loop above`。

`/tikz/every loop` (style, 初始值 `->`, `shorten >=1pt`)

这是个样式 (style)，针对的是每个 `loop`，所设置的选项会用在每个 `loop` 的开头。



```
\begin{tikzpicture}[every loop/.style={}]
  \draw (0,0) to [loop above] () to [loop right] ()
           to [loop below] () to [loop left] ();
\end{tikzpicture}
```

20 矩阵及其对齐方式

20.1 Overview

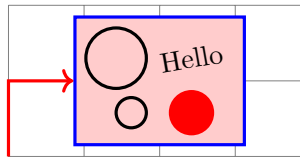
一个 TikZ 的矩阵 (matrix) 类似于 L^AT_EX 的 {tabular} 或 {array} 环境，只不过矩阵的元素 (cell) 是 TikZ 的绘图代码 (或者留空)。矩阵的每一行 (包括最后一行) 都用 \\ 结束，一行内的相邻元素之间用 & 分隔。注意 & 和 \\ 都具有“算子”的特性，它们后面可以带有方括号选项。

20.2 Matrices are Nodes

必须用 node 路径来构造矩阵。当 node 带有 matrix 选项后，这个 node 就用于构造矩阵，也就是说，一个矩阵其实是个 node 对象，该 node 的名称就是矩阵名称。

`/tikz/matrix=<true or false>` (默认值 true)

这个选项用于 node，使得该 node 用来构造矩阵。



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (4,2);
\node [matrix,fill=red!20,draw=blue,very thick] (my matrix) at (2,1)
{
  \draw (0,0) circle (4mm); & \node[rotate=10] {Hello}; \\
  \draw (0.2,0) circle (2mm); & \fill[red] (0,0) circle (3mm); \\
};
\draw [very thick,red,->] (0,0) |- (my matrix.west);
\end{tikzpicture}
```

`/tikz/every matrix` (style, 初始值 empty)

这个选项设置的样式 (在有效范围内) 用于每个矩阵。

`\matrix`

这个命令等效于 `\path node[matrix]`.

矩阵 (作为 node) 也可以添加到别的路径上，也可以引用矩阵的 node 坐标或者引用矩阵内部的 node。针对 node 的大多数 (不是全部的) 操作、选项都可以用于矩阵。针对整个矩阵的旋转和放缩变换无效，针

对矩阵元素的变换有效。对于命令 `\matrix`（或其等效语句）而言，以 `text` 开头的选项（如 `text width`）无效。

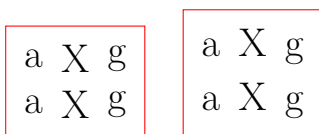
20.3 元素图形

20.3.1 元素图形的对齐方式

矩阵的元素是“图形”，绘制图形当然需要一个坐标系。当创建某个元素图形时，程序会开启一个专属于这个元素图形的“私有”坐标系，定义该元素图形的各条绘图命令就在这个坐标系内画图。

在默认之下：对于一行的元素图形而言，它们的坐标系的原点处于同一水平线上，以此线为界，元素图形在此线之上的部分属于该元素图形的高度，在此线之下的部分属于该元素图形的深度；一行的高度等于该行中诸元素图形的最大高度，是该行的上界；一行的深度等于该行中诸元素图形的最大深度，是该行的下界；对于相邻的两行而言，上行的下界紧邻下行的上界，上下两行的间距为 0，除非用 `row sep` 选项或用其它方式设置两行间距。

在默认之下：对于一列的元素图形而言，它们的坐标系的原点处于同一竖直线上；对于相邻的两列而言，左列的右边界紧邻右列的左边界，左右两列间距为 0，除非用 `column sep` 选项或其它方式设置两列间距。



```
\tikz[font=\Large]\matrix [draw=red]
{
  \node {a}; & \node {X}; & \node {g}; \\
  \node {a}; & \node {X}; & \node {g}; \\
};
\quad
\tikz[font=\Large,anchor=base]\matrix [draw=red]
{
  \node {a}; & \node {X}; & \node {g}; \\
  \node {a}; & \node {X}; & \node {g}; \\
};
```

上面例子中，第二个图形用了 `anchor=base`，这个选项对该图形内的所有 `node` 有效，将各图的 `base` 位置置于各图的原点，使得对齐效果不同于第一个图形。

20.3.2 调整行距和列距

调整行距和列距有 3 种方式：

- 默认下，间距是从边界到边界的（对应选项 `between borders`），可以改为从原点到原点（对应选项 `between origins`）。
- 用 `column sep` 和 `row sep`。

- 给 & 或 \\ 带上长度选项。

between origins

between borders

这两个选项指定间距的计算方式。

/tikz/column sep=<spacing> (默认值 0pt)

这个选项用于指定列间距，注意 <spacing> 是个带单位的尺寸，可以是负值。默认下，间距是从边界到边界的，即从左列的右边界到右列的左边界。如果同时使用了 between origins 选项，那么间距是从左列的原点所在的竖直线到右列的原点所在的竖直线。如果左右两个元素图形有重叠，则右侧的遮挡左侧的。

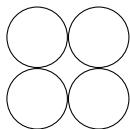


```
\tikz[font=\Huge]
  \matrix [draw=red,column sep=-4mm]
    {
      \node [fill=red]{a}; & \node [fill=cyan]{X}; \\
      \node {a}; & \node {X}; \\
    };
```

/tikz/row sep=<spacing> (默认值 0pt)

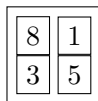
这个选项用于指定行间距，注意 <spacing> 是个带单位的尺寸，可以是负值。默认下，间距是从边界到边界的，即从上行的下边界到下行的上边界。如果上两个元素图形有重叠，则下部的遮挡上部的。

由 & 或 \\ 的长度选项指定的间距会叠加在选项 column sep 或 row sep 指定的间距上。下面例子中，在两个地方指定行距，总行距是 1mm-1mm=0mm:



```
\begin{tikzpicture}
\matrix [row sep=1mm]
{
\draw (0,0) circle (4mm); & \draw (0,0) circle (4mm); \\[-1mm]
\draw (0,0) circle (4mm); & \draw (0,0) circle (4mm); \\
};
\end{tikzpicture}
```

当用 &[<space>] 指定某两列的间距时，只能用在第一行，否则无效。



```
\begin{tikzpicture}
\matrix [draw,nodes=draw,column sep=1mm]
{
\node {8}; & \node{1}; \\
\node {3}; & [20mm,between origins] \node{5}; \\
};
\end{tikzpicture}
```

20.3.3 设置元素图形样式的选项

注意 `\matrix` 的 `draw`, `fill` 选项不能传递给矩阵的元素图形, 但颜色选项则能传递。

`/tikz/every cell={<row>}{<column>}` (style, 无默认值, 初始值 empty)

这个 key 设置的样式会加在每个元素图形的开头。注意这个样式中的 `draw`, `fill` 等选项不会传递给元素图形, 但颜色选项则能传递。

当前行(列)的行号(列号)是个计数器数值, 当前行号计数器是 `\pgfmatrixcurrentrow`, 当前列号计数器是 `\pgfmatrixcurrentcolumn`。

`/tikz/cells=<options>` (无默认值)

等效于 `every cell/.append style=<options>`。

`/tikz/nodes=<options>` (无默认值)

等效于 `every node/.append style=<options>`。如果把这个选项作为 `matrix` 的选项, 则这个选项的设置对每个元素图形有效, 但对矩阵本身无效。这个选项会把 `draw`, `fill` 等选项传递给所有元素图形。

以下作为样式 (style) 的 key 可以在一个矩阵中重复使用, 它们设置的样式会被叠加。注意它们不会把 `draw`, `fill` 等选项传递给元素图形。

`/tikz/column <number>` (style, 无默认值)

其中 `<number>` 是个整数, 指定列号, 这个 key 定义的样式只针对这一列。

`/tikz/every odd column` (style, 无默认值)

这个 key 定义的样式只针对奇数列。

`/tikz/every even column` (style, 无默认值)

这个 key 定义的样式只针对偶数列。

`/tikz/row <number>` (style, 无默认值)

其中 `<number>` 是个整数, 指定行号, 这个 key 定义的样式只针对这一行。

`/tikz/every odd row` (style, 无默认值)

这个 key 定义的样式只针对奇数行。

`/tikz/every even row` (style, 无默认值)

这个 key 定义的样式只针对偶数行。

`/tikz/row <row number> column <column number>` (style, 无默认值)

这个 key 定义的样式只针对指定元素。

```
123 456 789
12  45  78
1   4   7
```

```
\begin{tikzpicture}
  [column 1/.style={anchor=base west},
  column 2/.style={anchor=base east},
  column 3/.style={anchor=base}]
\matrix
{
  \node {123}; & \node{456}; & \node {789}; \\
  \node {12}; & \node{45}; & \node {78}; \\
  \node {1}; & \node{4}; & \node {7}; \\
};
\end{tikzpicture}
```

有的矩阵的元素具有极其类似的代码，例如，元素可能都是数字，这样元素代码的开头和结尾就都是一样的，如果为所有元素设置相同的开头和结尾就比较便利，这要用以下两个 key，它们针对非空元素：

`/tikz/execute at begin cell=<code>` (无默认值)

设置所有元素的代码的开头部分。

`/tikz/execute at end cell=<code>` (无默认值)

设置所有元素的代码的结尾部分。

```
8 1 6
3 5 7
4 9 2
```

```
\begin{tikzpicture}
  [matrix of nodes/.style={
  execute at begin cell=\node\bgroup,
  execute at end cell=\egroup;%
  }]
\matrix [matrix of nodes]
{
  8 & 1 & 6 \\
  3 & 5 & 7 \\
  4 & 9 & 2 \\
};
\end{tikzpicture}
```

`/tikz/execute at empty cell=<code>` (无默认值)

设置空元素的图形。

20.4 矩阵的位置选项

一个 node 有自己的坐标系，其中有各种位置，这里只涉及锚位置 (north, east 等) 和角度位置，不涉及平移位置 (above, left 等)。

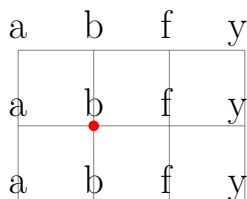
矩阵是 node 对象，它有各种位置，如果矩阵元素是 node，则它们也有各自的位置，用这些位置可以调整矩阵的位置。

`/tikz/matrix anchor=<anchor>` (无默认值)

这个选项将矩阵的 `<anchor>` 位置放在矩阵的指向点处。矩阵的指向点就是选项 `at` 指定的位置，或者其它类似的定位形式确定的位置。

`/tikz/anchor=<anchor or node.anchor>` (无默认值)

这个选项针对矩阵内的各个元素，将元素图形的 `<anchor or node.anchor>` 位置放在该元素图形的指向点上。



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\matrix[matrix anchor=inner node.south,anchor=base,
row sep=3mm,column sep=5mm,font=\Large] at (1,1)
{
\node {a}; & \node {b}; & \node {f}; & \node {y}; \\
\node {a}; & \node(inner node) {b}; & \node {f}; & \node {y}; \\
\node {a}; & \node {b}; & \node {f}; & \node {y}; \\
};
\fill [red](inner node.south) circle (2pt);
\end{tikzpicture}
```

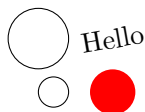
上面例子中，选项 `matrix anchor=inner node.south` 将点 `inner node.south` 与 `(1,1)` 重合，这在整体上决定了矩阵的位置。选项 `anchor=base` 将各个元素图形的 `base` 位置与该图形的原点重合，这决定了元素的对齐方式。

20.5 自定义分列符

在构造矩阵时，TikZ 用 `&` 作为分列符，而 PGF 使用命令 `\pgfmatrixnextcell` 来分隔左右相邻的两个元素。由于在 L^AT_EX 的 `{tabular}` 环境中也是用 `&` 作为分列符的，所以，如果在 `{tabular}` 环境中使用 TikZ 矩阵，将 `&` 作为分列符会导致错误。此时，可以用下面的选项自定义 TikZ 矩阵的分列符：

`/tikz/ampersand replacement=<macro name or empty>` (无值)

如果这个选项值是个宏，那么这个宏就等价于命令 `\pgfmatrixnextcell`。



```
\tikz
\matrix [ampersand replacement=\spc]
{
\draw (0,0) circle (4mm); \spc \node[rotate=10] {Hello}; \\
\draw (0.2,0) circle (2mm); \spc \fill[red] (0,0) circle (3mm); \\
};
```

关于矩阵的其它内容参考 matrix 程序库。

57 matrix 程序库

首先调用矩阵程序库: `\usetikzlibrary{matrix}`, 之后可以使用以下介绍的内容。

57.1 矩阵中的 node

如果一个 TikZ 矩阵的元素都是 node, 那么用下一个矩阵选项会比较便利:

`/tikz/matrix of nodes` (无值)

这个选项会在每个元素代码的开头加 “`\node{`”, 在每个元素代码的结尾加 “`}`”, 所以, 如果元素代码是文字、数字、数学模式、 \LaTeX 表格等内容, 这个选项会把元素代码完善为一个完整的 node 语句。并且, 这个选项还会把元素图形的锚位置 (anchor) 设为 base, 还会将每个元素图形的名称 (name) 设为 `<matrix name>-<row number>-<column number>` 这种形式, 以便于引用。

8 1 6	\begin{tikzpicture}
3 → 7	\matrix (magic) [matrix of nodes]
4 9 2	{
	8 & 1 & 6 \\
	3 & 5 & 7 \\
	4 & 9 & 2 \\
	};
	\draw[thick,red,->] (magic-1-1) - (magic-2-3);
	\end{tikzpicture}

如果要单独设置某个元素图形的样式, 可以使用 `row <row number> column <column number>` 这个样式 key 来设置:

```
8 1 6
3 5 7
4 9 2
```

```
\begin{tikzpicture}
  [row 2 column 2/.style={font=\Huge,red,shift={(0,-1mm)}}]
\matrix [matrix of nodes]
  {
    8 & 1 & 6 \\
    3 & 5 & 7 \\
    4 & 9 & 2 \\
  };
\end{tikzpicture}
```

选项 `matrix of nodes` 只是把 “`\node{`” 和 “`}`” 加在元素代码上，针对单个元素图形的特殊选项还需要个别设置。在使用选项 `matrix of nodes` 的情况下，针对单个元素图形的选项放在方括号里，方括号还必须放在该元素代码的前面，还要在方括号前后加竖线定界符。

```
8 1 6
3 5 7
4 9 2
```

```
\begin{tikzpicture}
  [row 2 column 2/.style={font=\Huge,red,shift={(0,-1mm)}}]
\matrix [matrix of nodes]
  {
    8 & & 1 & 6 \\
    3 & |[draw,fill=cyan]| 5 & 7 \\
    4 & & 9 & 2 \\
  };
\end{tikzpicture}
```

其中需要在方括号前后加竖线定界符，是因为分列符 `&` 的后面可以带有方括号选项，如果不加竖线定界符，程序会把方括号看作是属于分列符 `&` 的。

在使用选项 `matrix of nodes` 的情况下，如果某个元素图形的代码以 `\path`、`\draw`、`\node`、`\fill` 等命令开头，或者以能够展开为这些命令的符号串开头，那么对于该元素而言，选项 `matrix of nodes` 的添加符号 “`\node{`” 和 “`}`” 的作用会被抑制，但其它作用（即设置锚位置和命名作用）仍然有效。这样就可以单独设置该元素图形的外观。

```
8 1 6
3 5 7
4 9 2
```

```
\begin{tikzpicture}
\matrix [matrix of nodes]
  {
    8 & 1 & 6 \\
    3 & 5 & \node[red]{7}; \draw(0,0) circle(10pt); \\
    4 & 9 & 2 \\
  };
\end{tikzpicture}
```

`/tikz/matrix of math nodes` (无值)

这个选项的作用与 `matrix of nodes` 类似，它会在每个元素代码的开头加 “`\node{}`”，在每个元素代码的结尾加 “`}`”。

`/tikz/nodes in empty cells=<true or false>` (默认值 `true`)

这个选项会在空元素（没有代码的元素）的位置处设置一个内容为空的 `node`。

57.2 换行符号与矩阵行的结束符号

符号 `\\` 是 $\text{T}_{\text{E}}\text{X}$ 的换行符号，也是 TikZ 矩阵的分行符号。如果矩阵的某个元素是以文字为内容的 `node`，并且文字内使用 `\\` 换行，就可能会造成歧义。此时，程序应用以下规则：

1. 在矩阵内部，`\\` 是分行符号。
2. 如果在 `\\` 与它前面的分列符 `&` 之间只有一层花括号，并且开括号 “`{`” 紧跟在 `&` 之后，那么 `\\` 是属于这层花括号之内的文本换行符号。

row 1	upper line
lower line	
row 2	hmm

```
\begin{tikzpicture}
\matrix [matrix of nodes,nodes={text width=16mm,draw}]
{
row 1 & upper line \\ lower line \\
row 2 & hmm \\
};
\end{tikzpicture}
```

row 1	upper line lower line
row 2	hmm

```
\begin{tikzpicture}
\matrix [matrix of nodes,nodes={text width=16mm,draw}]
{
row 1 & {upper line \\ lower line} \\
row 2 & hmm \\
};
\end{tikzpicture}
```

注意 `a&b{c\\d}\\` 这种形式是错的，因为 `&` 与 `{` 这两个符号不是紧邻的，它们之间有 `b`。

57.3 定界符

定界符通常具有括号的形式，或具有类似括号的用处，例如矩阵两侧的括号是一种定界符。任何 `node` 对象都可以带有左右定界符，只需要为它添加以下选项。

`/tikz/left delimiter=<delimiter>`

`/tikz/right delimiter=<delimiter>`

`/tikz/above delimiter=<delimiter>`

`/tikz/below delimiter=<delimiter>`

分别设置上、下、左、右的定界符。

```
/tikz/every delimiter      (style, 初始值 empty)
/tikz/every left delimiter (style, 初始值 empty)
/tikz/every right delimiter (style, 初始值 empty)
/tikz/every above delimiter (style, 初始值 empty)
/tikz/every below delimiter (style, 初始值 empty)
```

这些是作为样式 (style) 的 key.

$$\left(\int_0^1 x dx \right)$$

```
\begin{tikzpicture}
\node [fill=red!20,left delimiter=(,right delimiter=\}]
      {\displaystyle\int_0^1 x\,\mathrm{d}x};
\end{tikzpicture}
```

$$\left(\begin{array}{ccc} a_8 & a_1 & a_6 \\ a_3 & a_5 & a_7 \\ a_4 & a_9 & a_2 \end{array} \right)$$

```
\begin{tikzpicture}
  [every left delimiter/.style={red,xshift=1ex},
  every right delimiter/.style={xshift=-1ex}]
\matrix [matrix of math nodes,
  left delimiter=(,
  right delimiter=\}]
{
  a_8 & a_1 & a_6 \\
  a_3 & a_5 & a_7 \\
  a_4 & a_9 & a_2 \\
};
\end{tikzpicture}
```

$$\overbrace{\left(\begin{array}{ccc} a_8 & a_1 & a_6 \\ a_3 & a_5 & a_7 \\ a_4 & a_9 & a_2 \end{array} \right)}$$

```
\begin{tikzpicture}
\matrix [matrix of math nodes,%
  left delimiter=\|,right delimiter=\rmoustache,%
  above delimiter=(,below delimiter=\}]
{
  a_8 & a_1 & a_6 \\
  a_3 & a_5 & a_7 \\
  a_4 & a_9 & a_2 \\
};
\end{tikzpicture}
```

22 函数绘图

如果你想比较简便而精细地绘制科技方面的图形，请参考 `pgfplots` 和本手册的第六部分 `Data Visualization`.

22.1 Overview

对于十分复杂、精细的图形来说，`TikZ` 的绘图能力不如专业的数学软件（例如 `gnuplot`, `mathematica`），但在 `TEX` 中使用 `TikZ` 绘图的优势是它与 `TEX` 的兼容性。例如，从外部插入一个图形并对该图做放缩，图形中的文字字体、公式尺寸就可能与正文不匹配。

用 `TikZ` 绘图的方式主要有 3 种：

1. 使用 `plot` 路径算子。
2. 使用 `datavisualization` 路径命令。
3. 使用 `pgfplots` 宏包。

22.2 `plot` 路径算子

`plot` 算子绘制的路径可以作为主路径的一个子路径。

```
\path . . . --plot<further arguments> . . . ;
```

这个句式中，`plot` 算子绘制的子路径与主路径之间是“line-to”的联系方式。

```
\path . . . plot<further arguments> . . . ;
```

这个句式中，`plot` 算子绘制的子路径与主路径之间是“move-to”的联系方式。

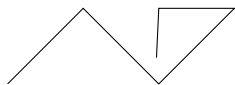
以“line-to”的联系方式为例，`plot` 语句的句法可以是：

1. `--plot[<local options>]coordinates{<coordinate 1 ><coordinate 2> . . . <coordinate n>}`
2. `--plot[<local options>]file{<filename>}`
3. `--plot[<local options>]<coordinate expression>`
4. `--plot[<local options>]function{<gnuplot formula>}`

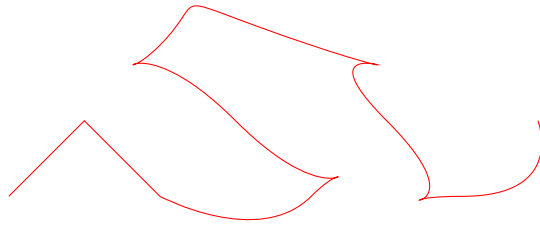
注意以上句式中的 `<local options>` 应当是专门针对 `plot` 算子设计的选项，即后文将要介绍的选项，例如，`smooth`, `variable`, `domain`, `mark` 等，这些选项的作用范围也仅限于其所从属的 `plot` 算子，不影响其它 `plot` 算子。而其它的选项，例如 `draw`, `fill`, `color=red` 等，用在这里是无效的。

22.3 连点成线

可以用 `plot` 算子将多个点用直线段或有一定弯曲状态的曲线连接起来。



```
\tikz \draw plot coordinates
      {(0,0) (1,1) (2,0) (3,1) (2,1) (10:2cm)};
```



```
\begin{tikzpicture}
\draw[color=red,xshift=5cm,smooth,tension=2.5]
(0,0) -- (1,1) -- plot coordinates {(2,0) (4,0) (2,2) (4,2) (5,1) (6,0) (7,1)};
\end{tikzpicture}
```

22.4 从外部文件中读取数据绘图

用语句 `--plot[<local options>]file{<filename>}` 从外部文件中读取数据绘图。

目前限于 TikZ 的读取能力，被读取的外部文件需要参照以下规则编辑：一行可以是空行；如果一行以 # 或 % 开头，则认为该行是空行，所以文件的注释内容可以用这两个符号开头；非空行必须是两个数字，之间用空格分隔；非空行的第二个数字之后可以跟随文字，但除了字母“o”和“u”，其它文字都会被忽略。

对文件的数据的进一步处理，参考 §107.2.

22.5 用函数表达式绘图

用句式 `--plot[<local options>]<coordinate expression>`，其中的 `<coordinate expression>` 是用圆括号括起来的坐标形式；如果圆括号里有 2 个表达式（之间由逗号分隔），第一个表达式的计算结果是横标，第二个表达式的计算结果是纵标；如果圆括号里有 3 个表达式，其意义也是类似的。如果某个表达式中有圆括号，则该表达式要用花括号括起来。

可以设置关于函数的自变量、定义域、样本点等内容。

`/tikz/variable=<macro>` (无默认值，初始值 `x`)

这个选项设置自变量，注意选项的值是个宏，初始值是 `\x`.

`/tikz/samples=<number>` (无默认值，初始值 `25`)

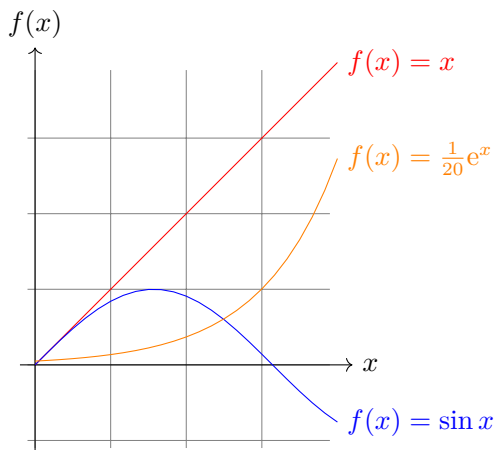
这个选项设置样本点的数目，初始值 `25`.

`/tikz/domain=<start>:<end>` (无默认值，初始值 `-5:5`)

设置定义域，即函数曲线的范围。

`/tikz/samples at=<sample list>`

其中的 `<sample list>` 是个数组，是自变量的取值点，在这些点上计算函数值，以此得到样本点。数组中可以使用省略号“...”来构造等差数列（参考 `foreach` 语句）。注意这个选项会抑制 `samples` 和 `domain` 选项。



```

\begin{tikzpicture}[domain=0:4]
\draw[very thin,color=gray] (-0.1,-1.1) grid (3.9,3.9);
\draw[->] (-0.2,0) -- (4.2,0) node[right] {$x$};
\draw[->] (0,-1.2) -- (0,4.2) node[above] {$f(x)$};
\draw[color=red] plot (\x,\x) node[right] {$f(x) = x$};
\draw[color=blue] plot (\x,{sin(\x r)}) node[right] {$f(x) = \sin x$};
\draw[color=orange] plot (\x,{0.05*exp(\x)})
  node[right] {$f(x) = \frac{1}{20} \mathrm{e}^x$};
\end{tikzpicture}

```

上面例子中，符号 $\backslash x r$ 是将角度 $\backslash x$ 转换为弧度。



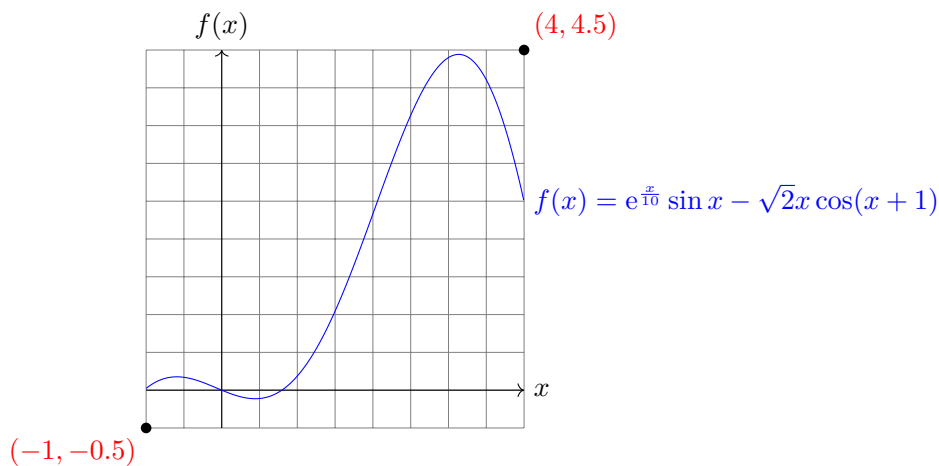
```

\tikz \draw[domain=0:360,smooth,variable=\t]
  plot ({sin(\t)},\t/360,{cos(\t)});

```

上面的例子绘制 3 维曲线。

下面例子中的运算需要调用 `math`, `cal` 程序库来配合。



```

\begin{tikzpicture}[samples=500,domain=-1:4]
\draw [help lines,step=0.5cm]
  (-1,-0.5)node[below left,red]{\(-1,-0.5\)} grid (4,4.5)node[above right,red]{\$(4,4.5)
  \$};
\draw[->] (-1,0) -- (4,0) node[right] {\$x\$};
\draw[->] (0,-0.5) -- (0,4.5) node[above] {\$f(x)\$};
\draw[color=blue] plot (\x,{exp(\x / 10) * (sin(\x r)) - (2 ^ (1/2)) * \x * (cos(\x r +
  1))}) node[right] {\$f(x) = \mathrm{e}^{-\frac{x}{10}} \sin x - \sqrt{2} x \cos (x+1)\$
  };
\fill (-1,-0.5) circle [radius=2pt] (4,4.5) circle [radius=2pt];
\end{tikzpicture}

```

22.6 调用 gnuplot 绘制函数图形

22.7 给 plot 路径上的样本点加标记

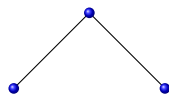
与 node 类似，在整个主路径画出后，标记才会添加到路径的样本点上。

`/tikz/mark=<mark mnemonic>`

这个选项设置点标记的类型。点标记要提前用命令 `\pgfdeclareplotmark` 来定义，该命令的句法是 `\pgfdeclareplotmark{<mark name>}{<绘制点标记的代码>}`

默认下，有 `*`, `+`, `x` 这 3 种标记类型可用，分别代表圆点，加号，叉号。在调用 `plotmarks` 程序库后可以使用更多标记类型，见 §62, §63。

有个特殊的点标记类型，名称为 `ball`，只能用在 TikZ 中，选项 `ball color` 可以设置 `ball` 的颜色。



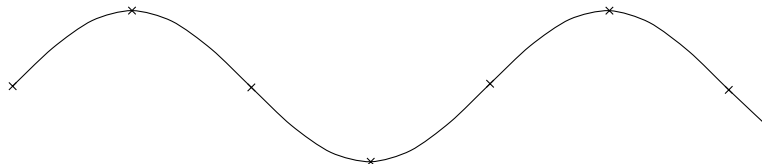
```

\begin{tikzpicture}
\draw plot [mark=ball,ball color=blue]
  coordinates {(0,0)(1,1)(2,0)};
\end{tikzpicture}

```

`/tikz/mark repeat=<r>`

标记第 1 个点，第 `<r>+1` 个点，第 `2*<r>+1` 个点……，故被标记的点的序号是 $(n-1) * <r> + 1$ ， $n = 1, 2, 3, \dots$ ，是以 `<r>` 为公差的等差数列。



```

\tikz \draw plot[mark=x,mark repeat=3,smooth] coordinates
{
  (0.00000, 0.00000) (0.52632, 0.50235) (1.05263, 0.86873) (1.57895, 0.99997)
}

```



```
(2.10526, 0.86054) (2.63158, 0.48819) (3.15789, -0.01630) (3.68421, -0.51638)
(4.21053, -0.87669) (4.73684, -0.99970) (5.26316, -0.85212) (5.78947, -0.47390)
(6.31579, 0.03260) (6.84211, 0.53027) (7.36842, 0.88441) (7.89474, 0.99917)
(8.42105, 0.84348) (8.94737, 0.45948) (9.47368, -0.04889) (10.00000, -0.54402)
};
```

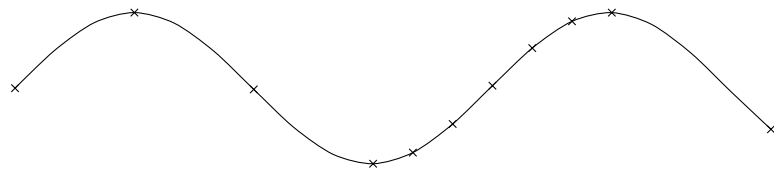
上面的图形中有 20 个样本点，被标记的是第 1, 4, 7, 10, 13, 16, 19 个点，共七个点。

```
/tikz/mark phase=<p>
```

如果要使用这个选项，则它应当与 `mark repeat=<r>` 配合使用。这个选项使得 TikZ 先标记第 `<p>` 个点，然后标记第 `<p>+<r>` 个点，然后标记第 `<p>+2*<r>` 个点……，点的序号是以 `<r>` 为公差的等差数列。

```
/tikz/mark indices={<list>}
```

这里 `<list>` 是个正整数列表，代表样本点的序号，标记序号对应的样本点。`<list>` 中可以使用省略号，程序按照 `\foreach` 语句的规则来解释省略号。



```
\tikz \draw plot[mark=x,mark indices={1,4,...,10,11,12,...,16,20},smooth] coordinates
{
(0.00000, 0.00000) (0.52632, 0.50235) (1.05263, 0.86873) (1.57895, 0.99997)
(2.10526, 0.86054) (2.63158, 0.48819) (3.15789, -0.01630) (3.68421, -0.51638)
(4.21053, -0.87669) (4.73684, -0.99970) (5.26316, -0.85212) (5.78947, -0.47390)
(6.31579, 0.03260) (6.84211, 0.53027) (7.36842, 0.88441) (7.89474, 0.99917)
(8.42105, 0.84348) (8.94737, 0.45948) (9.47368, -0.04889) (10.00000, -0.54402)
};
```

```
/tikz/mark size=<dimension>
```

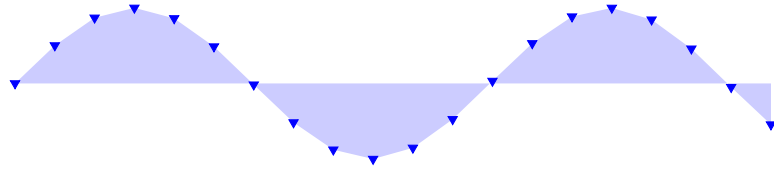
这个选项设置点标记的“半径”，注意 `<dimension>` 带有长度单位。也可以用 `scale=<factor>` 来调节标记尺寸。

```
/tikz/every mark (style, 无值)
```

这个 key 的设置会加在每个点标记的开头。

```
/tikz/mark options={<options>}
```

清空 `every mark` 的设置，将 `<options>` 作为标记的选项设置。



```
\tikz \fill[fill=blue!20] plot[mark=triangle*,mark options={color=blue,rotate=180}]
coordinates
{
(0.00000, 0.00000) (0.52632, 0.50235) (1.05263, 0.86873) (1.57895, 0.99997)
(2.10526, 0.86054) (2.63158, 0.48819) (3.15789, -0.01630) (3.68421, -0.51638)
(4.21053, -0.87669) (4.73684, -0.99970) (5.26316, -0.85212) (5.78947, -0.47390)
(6.31579, 0.03260) (6.84211, 0.53027) (7.36842, 0.88441) (7.89474, 0.99917)
(8.42105, 0.84348) (8.94737, 0.45948) (9.47368, -0.04889) (10.00000, -0.54402)
} |- (0,0);
```

注意上面例子中，符号“|-”把点 (10.00000, -0.54402) 与点 (0,0) 连接起来了。

`/tikz/no marks` (style, 无值)

取消点标记，等价于 `mark=none`。

`/tikz/no markers` (style, 无值)

取消点标记，等价于 `mark=none`。

22.8 直线、曲线、柱状图、条形图等

如果没有特别的设置，`plot` 算子会用直线段来连接样本点。使用下面的选项可以得到相应的图形。

`/tikz/sharp plot` (无值)

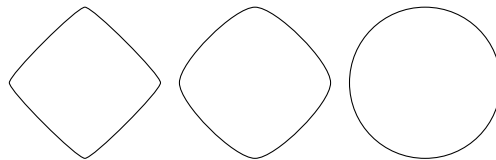
这个选项指示 `plot` 算子用直线段来连接样本点，这是默认的做法。

`/tikz/smooth` (无值)

这个选项指示 `plot` 算子用曲线段来连接样本点，并让曲线在样本点处（转角处）光滑。注意这个选项作用并不够智能，其效果可能不如意。连接点处的转弯角度越小（最好小于 30° ），并且各点的间距越是均匀，则曲线效果越好。

`/tikz/tension=<value>` (默认值 0.55)

这个选项调节曲线的“张力”、“膨胀状态”，即弯曲状态，数值越大，弯曲愈著。若有 4 个样本点均匀分布于一个圆上且张力值 `<value>` 是 1，则绘制的曲线是个圆。`<value>` 的默认值是 0.55。



```
\begin{tikzpicture}[smooth cycle]
```

```
\draw plot[tension=0.2] coordinates{(0,0) (1,1) (2,0) (1,-1)};
\draw[xshift=2.25cm] plot[tension=0.5] coordinates{(0,0) (1,1) (2,0) (1,-1)};
\draw[xshift=4.5cm] plot[tension=1] coordinates{(0,0) (1,1) (2,0) (1,-1)};
\end{tikzpicture}
```

`/tikz/smooth cycle` (无值)

这个选项指示 `plot` 算子用曲线段来连接样本点，让曲线在样本点处（转角处）光滑，且曲线是封闭的。



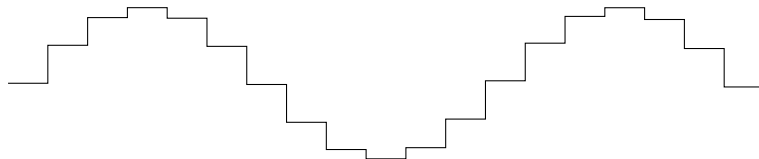
```
\tikz[scale=0.5]
\draw plot[smooth cycle] coordinates{(0,0) (1,0) (2,1) (1,2)}
plot coordinates{(0,0) (1,0) (2,1) (1,2)} -- cycle;
```

`/tikz/only marks` (无值)

这个选项指示 `plot` 算子只标记样本点，不画线。

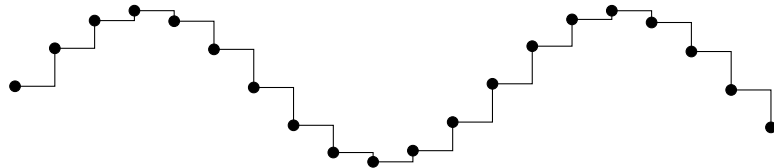
`/tikz/const plot` (无值)

本选项用样本点构造阶梯图。



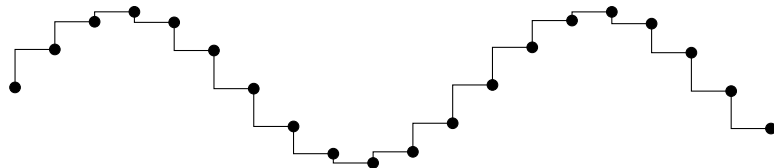
```
\tikz\draw plot[const plot] file{pgfmanual-sine.table};
```

`/tikz/const plot mark left` (无值)



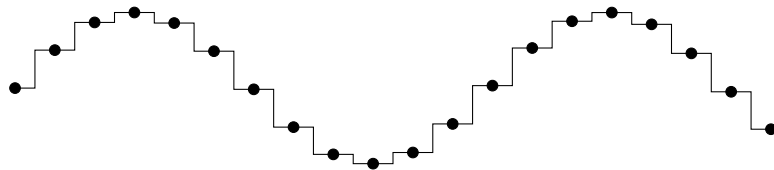
```
\tikz\draw plot[const plot mark left,mark=*] file{pgfmanual-sine.table};
```

`/tikz/const plot mark right` (无值)



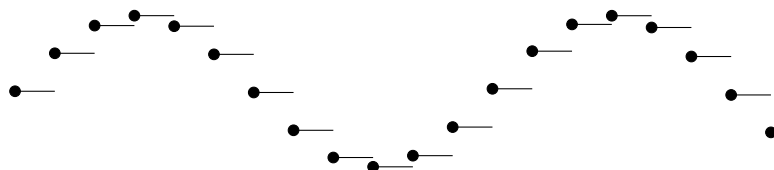
```
\tikz\draw plot[const plot mark right,mark=*] file{pgfmanual-sine.table};
```

`/tikz/const plot mark mid` (无值)



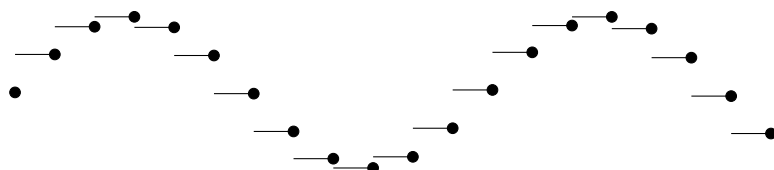
```
\tikz\draw plot[const plot mark mid,mark=*] file{pgfmanual-sine.table};
```

```
/tikz/jump mark left (无值)
```



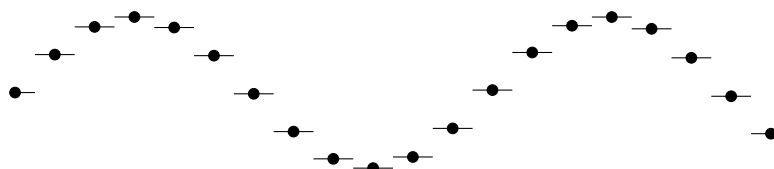
```
\tikz\draw plot[jump mark left, mark=*] file{pgfmanual-sine.table};
```

```
/tikz/jump mark right (无值)
```



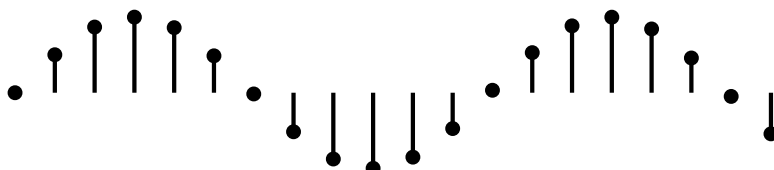
```
\tikz\draw plot[jump mark right, mark=*] file{pgfmanual-sine.table};
```

```
/tikz/jump mark mid (无值)
```



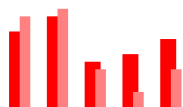
```
\tikz\draw plot[jump mark mid, mark=*] file{pgfmanual-sine.table};
```

```
/tikz/ycomb (无值)
```



```
\tikz\draw[ultra thick] plot[ycomb,thin,mark=*] file{pgfmanual-sine.table};
```

下面的例子是本选项的变通用法，画了一个柱状图：



```
\begin{tikzpicture}[ycomb]
  \draw[color=red,line width=6pt]
    plot coordinates{(0,1) (.5,1.2) (1,.6) (1.5,.7) (2,.9)};
  \draw[color=red!50,line width=4pt,xshift=3pt]
    plot coordinates{(0,1.2) (.5,1.3) (1,.5) (1.5,.2) (2,.5)};
\end{tikzpicture}
```

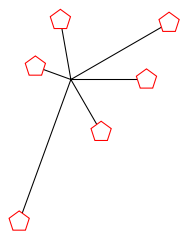
`/tikz/xcomb` (无值)



```
\tikz \draw plot[xcomb,mark=x,mark options={color=red}]
  coordinates{(1,0) (0.8,0.2) (0.6,0.4) (0.2,1)};
```

`/tikz/polar comb` (无值)

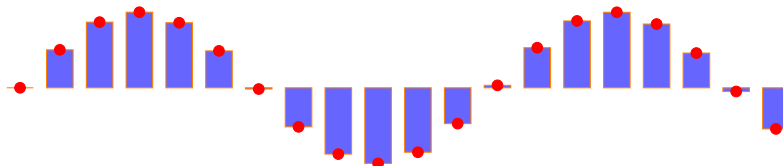
这个选项的作用是，从原点到样本点连线，使图形呈现放射状。



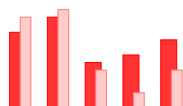
```
\tikz \draw plot[polar comb,
  mark=pentagon*,
  mark options={fill=white,draw=red},
  mark size=4pt]
  coordinates {(0:1cm) (30:1.5cm) (160:.5cm)
    (250:2cm) (-60:.8cm) (100:.8cm)};
```

`/tikz/ybar`

这个选项的作用类似 `ycomb`，本选项用样本点构造矩形，可以画出、填充矩形，外观是柱状图。

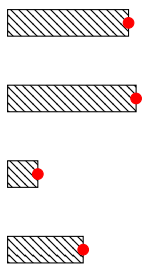


```
\tikz{\draw[draw=orange,fill=blue!60!white] plot[ybar] file{pgfmanual-sine.table};
  \draw plot[mark=*,only marks,mark options={color=red}] file{pgfmanual-sine.table};}
```



```
\begin{tikzpicture}[ybar]
  \draw[color=red,fill=red!80,bar width=6pt]
    plot coordinates{(0,1) (.5,1.2) (1,.6) (1.5,.7) (2,.9)};
  \draw[color=red!50,fill=red!20,bar width=4pt,bar shift=3pt]
    plot coordinates{(0,1.2) (.5,1.3) (1,.5) (1.5,.2) (2,.5)};
\end{tikzpicture}
```

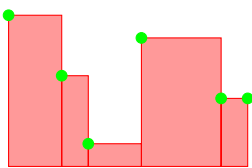
`/tikz/xbar` (无值)



```
\tikz {
  \draw[pattern=north west lines] plot[xbar]
    coordinates{(1,0) (0.4,1) (1.7,2) (1.6,3)};
  \draw[pattern=north west lines] plot[mark=*,only marks,
    mark options={color=red}]
    coordinates{(1,0) (0.4,1) (1.7,2) (1.6,3)};}
```

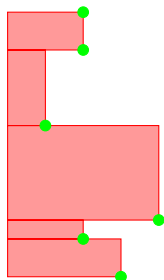
`/tikz/ybar interval` (无值)

这个选项用样本点构造柱状图。



```
\begin{tikzpicture}[ybar interval,x=10pt]
  \draw[color=red,fill=red!40!white] plot
    coordinates{(0,2) (2,1.2) (3,.3) (5,1.7) (8,.9) (9,.9)};
  \draw plot[mark=*,only marks,mark options={color=green}]
    coordinates{(0,2) (2,1.2) (3,.3) (5,1.7) (8,.9) (9,.9)};
\end{tikzpicture}
```

`/tikz/xbar interval` (无值)



```
\begin{tikzpicture}[xbar interval,x=0.5cm,y=0.5cm]
  \draw[color=red,fill=red!40!white] plot
    coordinates {(3,0) (2,1) (4,1.5) (1,4) (2,6) (2,7)};
  \draw plot[mark=*,only marks,mark options={color=green}]
    coordinates {(3,0) (2,1) (4,1.5) (1,4) (2,6) (2,7)};
\end{tikzpicture}
```

对于 `ybar`, `xbar` 类型的柱状图, 可以用选项 `bar width` 和 `bar shift` 调整其外观。对于 `ybar interval`, `xbar interval` 类型的柱状图, 可以用选项 `bar interval width` 和 `bar interval shift` 调整其外观。见 §62.4.

62 图柄程序库

首先调用程序库 `plohandlers`

```
\usepgflibrary{plohandlers} % LATEX and plain TEX and pure pgf
```

本程序库定义了一些图柄, 在 §107.3 中介绍了图柄 (plot handlers)、图流 (plot stream), 先阅读 §107.

TikZ 会自动加载本程序库。

首先注意, 用 `\pgfsetmovetofirstplotpoint` 或 `\pgfsetlinetofirstplotpoint` 可以改变画线图柄对图流的第一个点的处理, 参考 §107.3.

先定义 3 个图流, 以便在后文的例子中引用:

```
\pgfplothandlerrecord{\lizi}
```

```

\pgfplotstreamstart
\pgfplotstreampoint{\pgfpoint{0cm}{0cm}}
\pgfplotstreampoint{\pgfpoint{1cm}{1cm}}
\pgfplotstreampoint{\pgfpoint{2cm}{0.5cm}}
\pgfplotstreampoint{\pgfpoint{3cm}{0cm}}
\pgfplotstreampoint{\pgfpoint{3cm}{-1cm}}
\pgfplotstreampoint{\pgfpoint{2.5cm}{-.5cm}}
\pgfplotstreampoint{\pgfpoint{1cm}{-1cm}}
\pgfplotstreamend

```

```

\pgfplothandlerrecord{\lizitwo}
\pgfplotstreamstart
\pgfplotstreampoint{\pgfpoint{0cm}{0cm}}
\pgfplotstreampoint{\pgfpoint{1cm}{1cm}}
\pgfplotstreampoint{\pgfpoint{2cm}{0.5cm}}
\pgfplotstreamend

```

```

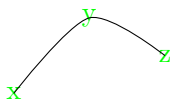
\pgfplothandlerrecord{\lizithree}
\pgfplotstreamstart
\pgfplotstreampoint{\pgfpoint{0cm}{2cm}}
\pgfplotstreampoint{\pgfpoint{1cm}{1cm}}
\pgfplotstreampoint{\pgfpoint{2cm}{0.5cm}}
\pgfplotstreampoint{\pgfpoint{4cm}{0.7cm}}
\pgfplotstreampoint{\pgfpoint{5cm}{0.5cm}}
\pgfplotstreampoint{\pgfpoint{6cm}{1cm}}
\pgfplotstreamend

```

62.1 曲线图柄

`\pgfplothandlercurveto`

这个图柄把命令 `\pgfpathcurveto` 用于图流的点（图流的第一个点除外）。



```

\begin{tikzpicture}
\draw[green] (0,0) node {x} (1,1) node {y} (2,.5) node {z};
\pgfplothandlercurveto
\lizitwo
\pgfusepath{stroke}
\end{tikzpicture}

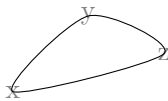
```

`\pgfsetplottension{<value>}`

这个命令设置曲线的“张力”值，默认值是 0.5。

\pgfplotsandlerclosedcurve

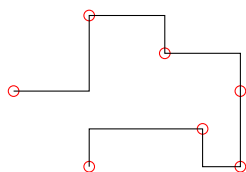
这个图柄类似 `\pgfplotsandlercurveto`，只是这个图柄会把图流作成闭合的，在图流的最后一个点后面使用命令 `\pgfpathclose`。



```
\begin{tikzpicture}
  \draw[gray] (0,0) node {x} (1,1) node {y} (2,.5) node {z};
  \pgfplotsandlerclosedcurve
  \liziwo
  \pgfusepath{stroke}
\end{tikzpicture}
```

62.2 Constant 图柄**\pgfplotsandlerconstantlineto**

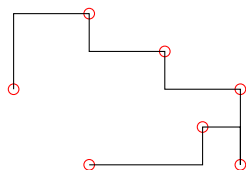
这个图柄的作用类似 `plot[const plot]`。



```
\begin{tikzpicture}
  \pgfplotsandlermark{\color{red}\pgfuseplotmark{o}}
  \lizi
  \pgfplotsandlerconstantlineto
  \lizi
  \pgfusepath{stroke}
\end{tikzpicture}
```

\pgfplotsandlerconstantlinetomarkright

这个图柄的作用类似 `plot[const plot mark right]`。



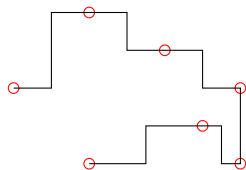
```
\begin{tikzpicture}
  \pgfplotsandlermark{\color{red}\pgfuseplotmark{o}}
  \lizi
  \pgfplotsandlerconstantlinetomarkright
  \lizi
```



```
\pgfusepath{stroke}
\end{tikzpicture}
```

`\pgfplothandlerconstantlinetomarkmid`

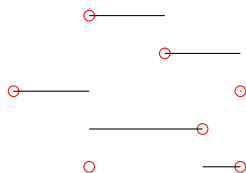
这个图柄的作用类似 `plot[const plot mark mid]`.



```
\begin{tikzpicture}
\pgfplothandlermark{\color{red}\pgfuseplotmark{o}}
\lizi
\pgfplothandlerconstantlinetomarkmid
\lizi
\pgfusepath{stroke}
\end{tikzpicture}
```

`\pgfplothandlerjumpmarkleft`

这个图柄的作用类似 `plot[jump mark left]`.



```
\begin{tikzpicture}
\pgfplothandlermark{\color{red}\pgfuseplotmark{o}}
\lizi
\pgfplothandlerjumpmarkleft
\lizi
\pgfusepath{stroke}
\end{tikzpicture}
```

`\pgfplothandlerjumpmarkright`

这个图柄的作用类似 `plot[jump mark right]`.

`\pgfplothandlerjumpmarkmid`

这个图柄的作用类似 `plot[jump mark mid]`.

62.3 Comb 图柄

`\pgfplothandlerxcomb`

这个图柄的作用类似 `plot[xcomb]`.

`\pgfplotshandlerycomb`

这个图柄的作用类似 `plot[ycomb]`.

`\pgfplotshandlerpolarcomb`

这个图柄的作用类似 `plot[polar comb]`.

`\pgfplotxzerolevelstreamconstant{<dimension>}`

举个例子说明这个命令的作用。假设某个部门在 1 月, 2 月, 3 月的盈利情况如下表:

各月盈利			各月相对 1 月的盈利		
1 月	2 月	3 月	1 月	2 月	3 月
15	5	25	0	-10	10

用下图表示上面第一个表格:

— * 3 月
 — * 2 月
 — * 1 月

```

\begin{tikzpicture}[scale=0.6,x=0.1*1cm]
  \draw [cyan] plot[xcomb,mark=x,mark options={color=red}]
    coordinates {(15,1)(5,2)(25,3)};
  \foreach \Yue/\yue in {(15,1)/1,(5,2)/2,(25,3)/3}
    \node [right] at \Yue {\yue 月};
\end{tikzpicture}

```

这个图形大体上可以比较各月的相对盈利程度。

为了用类似的图形表示上面第二个表格, 可以修改上面图形中的点的坐标, 不过也可以使用命令 `\pgfplotxzerolevelstreamconstant`:

— * 3 月
 — * 2 月
 — * 1 月

```

\begin{tikzpicture}[scale=0.6,x=0.1*1cm]
  \pgfplotxzerolevelstreamconstant{1.5cm}
  \draw [cyan] plot[xcomb,mark=x,mark options={color=red}]
    coordinates {(15,1)(5,2)(25,3)};
  \foreach \Yue/\yue in {(15,1)/1,(5,2)/2,(25,3)/3}
    \node [right] at \Yue {\yue 月};
\end{tikzpicture}

```

上面两个图形展示了命令 `\pgfplotxzerolevelstreamconstant` 的作用, 这个命令只对 `xcomb` 或 `xbar` 这两种图形有效, 对其它路径无效。

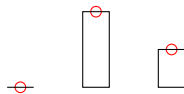
`\pgfplotyzerolevelstreamconstant{<dimension>}`

类似 `\pgfplotxzerolevelstreamconstant`, 本命令只对 `ycomb` 或 `ybar` 这两种图形有效, 对其它路径无效。

62.4 Bar 图柄

\pgfplotshandlerybar

这个图柄的作用类似 `plot[ybar]`.



```
\begin{tikzpicture}
  \pgfplotshandlermark{\color{red}\pgfuseplotmark{o}}
  \lizitwo
  \pgfplotshandlerybar
  \lizitwo
  \pgfusepath{stroke}
\end{tikzpicture}
```

\pgfplotshandlerxbar

这个图柄的作用类似 `plot[xbar]`.

对于 Bar 类型的柱状图，有以下选项可以调节其外观。

`/pgf/bar width={<dimension>}` (无默认值, 初始值 10pt)

或者 `/tikz/bar width={<dimension>}`

这个选项设置柱状图的柱宽。对于 `ybar` 图形，柱宽指的是柱的水平方向的尺寸；对于 `xbar` 图形，柱宽指的是柱的竖直方向的尺寸。<dimension> 可以是带有长度单位的表达式，会被数学解析器解析。

`/pgf/bar shift={<dimension>}` (无默认值, 初始值 0pt)

或者 `/tikz/bar shift={<dimension>}`

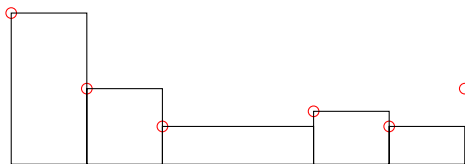
这个选项等效于平移选项 `xshift`，但只对柱状图有效。<dimension> 可以是带有长度单位的表达式，会被数学解析器解析。

\pgfplotbarwidth

这个宏的展开值是选项 `/pgf/bar width` 的值。

\pgfplotshandlerybarinterval

这个图柄的作用类似 `plot[ybar interval]`.



```
\begin{tikzpicture}
  \pgfplotshandlermark{\color{red}\pgfuseplotmark{o}}
  \lizithree
  \pgfplotshandlerybarinterval
\end{tikzpicture}
```

```

\lizithree
\pgfusepath{stroke}
\end{tikzpicture}

```

`\pgfplothandlerxbarinterval`

这个图柄的作用类似 `plot[xbar interval]`。

对于 `bar interval` 类型的柱状图，有以下选项可以调整其外观。

`/pgf/bar interval shift={<factor>}` (无默认值, 初始值 0.5)

或者 `/tikz/bar interval shift={<factor>}`

`/pgf/bar interval width={<scale>}` (无默认值, 初始值 1)

或者 `/tikz/bar interval width={<scale>}`

注意这两个选项的值不是尺寸，而是数字或者运算结果为数字的表达式，`<factor>` 与 `<scale>` 都会被数学解析器解析。

对于 `ybar interval` 类型的柱状图，这两个选项的作用如下：设 (x_i, y_i) 与 (x_{i+1}, y_{i+1}) 是图流中前后相继的两个点，图柄会在这两个点之间构造一个矩形；矩形中心点的横坐标是 $x_i + \text{<factor>} \cdot (x_{i+1} - x_i)$ ；矩形宽度（水平方向的尺寸）是 $\text{<scale>} \cdot (x_{i+1} - x_i)$ ；矩形的“身高”是 $\|y_i\|$ ；图流的最后一个点“落单”。

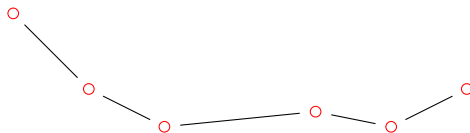
对于 `xbar interval` 类型的柱状图，这两个选项的作用如下：设 (x_i, y_i) 与 (x_{i+1}, y_{i+1}) 是图流中前后相继的两个点，图柄会在这两个点之间构造一个矩形；矩形中心点的纵坐标是 $y_i + \text{<factor>} \cdot (y_{i+1} - y_i)$ ；矩形宽度（垂直方向的尺寸）是 $\text{<scale>} \cdot (y_{i+1} - y_i)$ ；矩形的“水平长度”是 $\|x_i\|$ ；图流的最后一个点“落单”。

62.5 Gapped 图柄

`\pgfplothandlergaplineto`

这个图柄会在图流的点之间画直线段，但每个直线段的起点和终点并非恰好落在图流的点上，而是距离图流的点还有一段距离，从而造成“缺口”效果。这段距离由下面的选项指定：

`/pgf/gap around stream point=<dimension>` (无默认值, 初始值 1.5pt)



```

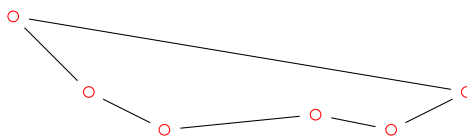
\begin{tikzpicture}
\pgfplothandlermark{\color{red}\pgfuseplotmark{o}}
\lizithree
\pgfplothandlergaplineto
\pgfkeys{/pgf/gap around stream point=6pt}
\lizithree

```

```
\pgfusepath{stroke}
\end{tikzpicture}
```

\pgfplothandlergapcycle

这个图柄会在图流的点之间画直线段，并在图流的终点后使用命令 `\pgfpathclose` 将图流作成闭合的多边形，但多边形的每个边的起点和终点并非恰好落在顶点上，而是距离顶点还有一段距离，从而造成“缺口”效果。这段距离也是由上面的选项 `/pgf/gap around stream point` 指定。



```
\begin{tikzpicture}
  \pgfplothandlermark{\color{red}\pgfuseplotmark{o}}
  \lizithree
  \pgfplothandlergapcycle
  \pgfkeys{/pgf/gap around stream point=6pt}
  \lizithree
  \pgfusepath{stroke}
\end{tikzpicture}
```

62.6 Mark 图柄

\pgfplothandlermark{<mark code>}

<mark code> 规定某种类型的点标记及其外观样式，本图柄用此类标记来标记图流中的点。

有两种编写 <mark code> 的思路：

1. 使用已定义的标记，例如可以使用程序库 `plotmarks` 定义的标记，前文的例子中有如下代码：

```
\pgfplothandlermark{\color{red}\pgfuseplotmark{o}}
```

其中用命令 `\pgfuseplotmark{o}` 调用了标记类型“o”。

2. 用绘图命令自定义一种标记。首先你要假设有一个“标记坐标系”，在这个坐标系中用绘图命令画出标记。当用自定义标记来标记某个点时，自定义标记的“标记坐标系”的原点会放在“被标记”的点上。

①
②
③

```
\begin{tikzpicture}
  \draw[gray] (0,0) node {x} (1,1) node {y} (2,.5) node {z};
  \pgfplothandlermark{
    \pgfpathcircle{\pgfpointorigin}{4pt}
    \pgfusepath{stroke}}
  \lizitwo
  \pgfusepath{stroke}
\end{tikzpicture}
```

\pgfsetplotmarkrepeat{<repeat>}

这个命令的效果类似 `plot[mark repeat=<r>]`.

\pgfsetplotmarkphase{<phase>}

这个命令的效果类似 `plot[mark phase=<p>]`.

\pgfplothandlermarklisted{<mark code>}{<index list>}

这个命令的效果类似 `plot[mark indices=<list>]`. <mark code> 规定或定义一种标记, <index list> 用来指定被标记的点。



```
\begin{tikzpicture}
  \draw[gray] (0,0) node {x} (1,1) node {y} (2,.5) node {z};
  \pgfplothandlermarklisted{
    \pgfpathcircle{\pgfpointorigin}{4pt}
    \pgfusepath{stroke}}
  {1,3}
  \lizenitwo
  \pgfusepath{stroke}
\end{tikzpicture}
```

\pgfuseplotmark{<plot mark name>}

<plot mark name> 是某个已定义的标记, 本命令调用这个类型的标记。

\pgfdeclareplotmark{<plot mark name>}{<code>}

本命令定义一种名称为 <plot mark name> 的标记. <code> 是绘图代码, 是对标记的具体定义. 在编写 <code> 时, 你也要假设有一个“标记坐标系”, 在这个坐标系中用绘图命令画出标记. 当用自定义标记来标记某个点时, 自定义标记的“标记坐标系”的原点会放在“被标记”的点上。



```
\pgfdeclareplotmark{my plot mark}{
  \pgfpathcircle{\pgfpoint{0cm}{1ex}}{1ex}
  \pgfusepathqstroke}
\begin{tikzpicture}
  \draw[gray] (0,0) node {x} (1,1) node {y} (2,.5) node {z};
  \pgfplothandlermark{\pgfuseplotmark{my plot mark}}
  \lizenitwo
  \pgfusepath{stroke}
\end{tikzpicture}
```

\pgfsetplotmarksize{<dimension>}

本命令将 $\text{T}_{\text{E}}\text{X}$ 的尺寸宏 `\pgfplotmarksize` 的值设为 <dimension>, 用以规定标记的“半径”。这个值是个“推荐值”, 在某些情况下这个值会被忽略。

这个命令的作用类似 `plot[mark size=<dimension>]`。

`\pgfplotmarksize`

这是个 $\text{T}_\text{E}_\text{X}$ 的尺寸宏，它的值是标记的“推荐值”。

PGF 预定义 3 种类型的 plot 标记，其“名称”分别是：*，x，+，是 3 个符号，分别对应小圆圈、叉号、加号。



```
\tikz \draw plot[mark=*,mark options={color=red}]
coordinates{(0,0)(0.5,0.25)(1,0)};
```



```
\tikz \draw plot[mark=x,mark options={color=red}]
coordinates{(0,0)(0.5,0.25)(1,0)};
```



```
\tikz \draw plot[mark=+,mark options={color=red}]
coordinates{(0,0)(0.5,0.25)(1,0)};
```

63 用于 plot 绘图的点标记类型

首先调用程序库：`\usepgflibrary{plotmarks}`。

如果不调用这个程序库，在默认下只有 *，+，x 这 3 种类型的点标记可用。plotmarks 提供多种类型的点标记。（参考手册内容。）

带有星号 * 的点标记可以被填充颜色，见 §22.7，第 332 页。

点标记可以被旋转

```
mark options={rotate=90}
every mark/.append style={rotate=90}.
```

`/pgf/mark color={<color>}` （无默认值，初始值 empty）

这个选项设置点标记的颜色，颜色是填充的。如果选项值留空，则无填充色。如果选项值是 none，则取消填充。有的点标记只能填充其内部的一半区域。

`/pgf/text mark={<text>}` （无默认值，初始值 p）

用 <text> 来标记样本点，这里的 <text> 可以是任何 $\text{T}_\text{E}_\text{X}$ 内容，比如文字，图形，数学公式，表格等。

`/pgf/text mark as node={<boolean>}` （无默认值，初始值 false）

这个选项决定由选项 `text mark={<text>}` 给出的点标记 <text> 是否转为 node 对象。

`/pgf/text mark style={<options for mark=text>}`

这个选项设置由选项 `text mark={<text>}` 给出的点标记 <text> 的样式。

如果 `/pgf/text mark as node=false`，那么样式可以是 left, right, top, bottom, base, rotate 等等。

如果 `/pgf/text mark as node=true`，那么样式可以使用 node 算子能接受的诸多选项来设置。

23 透明度

23.1 Overview

通常，TikZ 画出的图形都是不透明的。

pdfTEX 对透明度效果的支持最好。

23.2 为图形、路径、文字设定透明度

选项 `opacity=<value>` 设置“不透明度”，`<value>` 是 0 到 1 之间的数字，表示不透明的程度，若 `<value>` 大于 1，则当作 1 看待；若 `<value>` 小于 0，则当作 0 看待。

`/tikz/draw opacity=<value>` (无默认值)

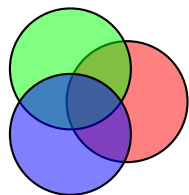
这个选项设置路径线条的“不透明度”，若 `<value>` 是 1，则线条完全不透明；若 `<value>` 是 0，则线条完全透明，即不可见。



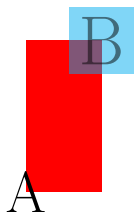
```
\begin{tikzpicture}[line width=1ex]
  \draw (0,0) -- (3,1);
  \filldraw [fill=yellow!80!black,draw opacity=0.5] (1,0) rectangle (2,1);
\end{tikzpicture}
```

`/tikz/fill opacity=<value>` (无默认值)

这个选项设置路径填充区域的“不透明度”，对路径上的文字标签、填充的颜色、颜色渐变、插入的外部图形都有效。



```
\begin{tikzpicture}[thick,fill opacity=0.5]
  \filldraw[fill=red] (0:.5cm) circle (8mm);
  \filldraw[fill=green] (120:.5cm) circle (8mm);
  \filldraw[fill=blue] (-120:.5cm) circle (8mm);
\end{tikzpicture}
```



```
\begin{tikzpicture}
  \fill[red] (0,0) rectangle (1,2);
  \node at (0,0) {\huge A};
  \node[fill opacity=0.5, fill=cyan] at (1,2) {\Huge B};
\end{tikzpicture}
```

`/tikz/opacity=<value>`

同时设置 `draw opacity=<value>` 和 `fill opacity=<value>`.

`/tikz/text opacity=<value>` (无默认值)

这个选项设置文字标签的“不透明度”，对文字的作用优于 `fill opacity` 选项。

`/tikz/fill opacity=<value>` (无默认值)

设置 `draw opacity` 和 `fill opacity` 的值。

`/tikz/transparent` (style, 无值)

完全透明，不可见。

`/tikz/ultra nearly transparent` (style, 无值)

`/tikz/very nearly transparent` (style, 无值)

`/tikz/nearly transparent` (style, 无值)

`/tikz/semitransparent` (style, 无值)

`/tikz/nearly opaque` (style, 无值)

`/tikz/very nearly opaque` (style, 无值)

`/tikz/ultra nearly opaque` (style, 无值)

`/tikz/opaque` (style, 无值)

完全不透明。

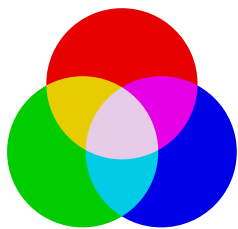
如果两个有某种不透明度的区域重叠，则重叠部分的不透明度会叠加。如果不希望出现叠加，可以在环境选项中使用 `transparency group` 选项，见后文 `Transparency Groups` 一节。

23.3 混色模式

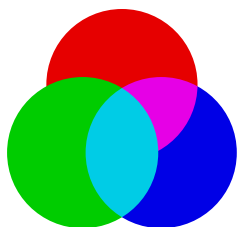
当两个或多个有某种不透明度的区域重叠时，重叠部分的颜色是混合颜色，有多种混色模式来决定重叠部分的颜色。这里用的混色模式是 PDF 参考文件 (PDF Reference, six edition, §7.2.4) 中的模式。

`/tikz/blend mode=<mode>`

这个选项选定混色模式，`<mode>` 是混色模式的名称。混色是 PDF 格式的高级功能，不同的预览器对颜色渲染 (color render) 的显示效果有所差别。为了确保显示效果一致，TikZ 对选项 `blend mode` 的使用位置做了限制：如果这个选项用作环境选项，则这个环境必须是套嵌在某个外层环境内部的子环境，而且外层环境必须带有 `transparency group` 选项；如果这个选项用作某个绘图命令的选项，则这个命令所从属的环境必须带有 `transparency group` 选项。如果这个选项用作环境选项，则对环境内的各个图形有效，即按混色模式画出图形的重叠部分。如果这个选项用作某个绘图命令的选项，则该命令绘出的图形与其它图形出现混色效果。



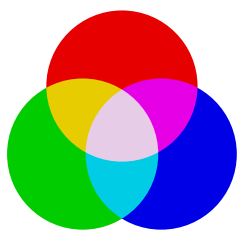
```
\tikz {
\begin{scope}[transparency group]
  \begin{scope}[blend mode=screen]
    \fill[red!90!black] ( 90:.6) circle (1);
    \fill[green!80!black] (210:.6) circle (1);
    \fill[blue!90!black] (330:.6) circle (1);
  \end{scope}
\end{scope}
}
```



```
\tikz {
\begin{scope}[transparency group]
  \fill[red!90!black] ( 90:.6) circle (1);
  \fill[green!80!black] (210:.6) circle (1);
  \fill[blue!90!black,blend mode=screen]
    (330:.6) circle (1);
\end{scope}
}
```

`/tikz/blend group=<mode>`

这个选项只能用作环境选项，它使得当前环境是个“有透明度的环境”（transparency group），环境内的混色模式选定为 <mode>。这个选项是上一选项的简捷形式。



```
\tikz [blend group=screen] {
  \fill[red!90!black] ( 90:.6) circle (1);
  \fill[green!80!black] (210:.6) circle (1);
  \fill[blue!90!black] (330:.6) circle (1);
}
```

各种颜色模式参考手册。

23.4 颜色淡入、淡出——fading

颜色的淡入、淡出即颜色逐渐消退（作为动词或名词），英文单词称之为：fading, soft masks, opacity masks, masks, soft clips.

参考 §110.

23.4.1 创建 fading 图

创建一个具有颜色淡出效果的图形（fading）的基本方法是使用灰度图（fading picture）。灰度图就是只有黑、白、灰三种颜色的图，即黑白图。颜色具有明度（也叫做亮度）属性，颜色的明度使用黑色、白色、

各种灰色来标示，白色的明度最高，黑色的明度最低，灰色的明度居间。

要使用灰度图，先调用 `fadings` 程序库。

把通常的绘图命令放入环境 `{tikzfadingfrompicture}` 中，对路径的不同部位规定不同的明度，就作成一个灰度图。通常灰度图是不可见的，它起到“底片”的作用。用 `name=<name>` 选项给灰度图命名，然后在环境 `{tikzfadingfrompicture}` 之外，在正式的绘图环境中，将灰度图的名称作为某个绘图命令的选项，就把灰度图作为一个不可见的潜在图形引入绘图环境中。绘图命令定义的路径（图形）应当与灰度图有重叠，因为颜色的淡出效果正是针对这个重叠部分的。程序参照绘图命令的颜色设置以及灰度图的明度设置，将颜色和透明度赋予重叠部分中的点，得到需要的 `fading`。

为了方便，下面称绘图命令定义的路径与灰度图的重叠部分为“重合淡出”。

灰度图的“底片”作用是这样的。假设 `fading` 中的一个像素点为 P ，而 `fading picture` 中的像素点 P' 与 P 处于相同的坐标位置。点 P 与点 P' 的关系是：点 P' 的明度与点 P 的不透明度正相关，若点 P' 是白色，则点 P 不透明；若点 P' 是黑色，则点 P 透明。而点 P 的颜色由绘图命令的颜色选项指定，这样点 P 既有颜色也有透明度。这种“黑透白不透”的对应关系有点反直觉，所以 TikZ 定义了一个名称为 `transparent` 的颜色，它实际上等效于黑色。颜色表达式 `transparent!<percentage>` 用小数 `<percentage>` 来表示透明度，数值越大，越是透明。

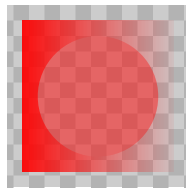
如果没有重合淡出，或者重合部分过小，就可能没有预期的 `fading` 效果，为了避免这个情况，程序提供逻辑值选项 `fit fading`，并设定其初始值为 `true`，其作用是将灰度图做适当的平移和放缩，来匹配绘图命令定义的路径，使得重合部分尽可能地大。

```
\begin{tikzfadingfrompicture}[<options>
  <environment contents>
\end{tikzfadingfrompicture}
```

这个环境定义灰度图，但不可见，环境中的绘图命令是通常的绘图命令。该环境可以带有命名选项：

```
/tikz/name={<name>}
```

这个选项作为环境 `{tikzfadingfrompicture}` 的选项，给该环境命名。命名后，可以在绘图环境中用 `path fading` 选项引用这个名称。



```
% 定义灰度图
\begin{tikzfadingfrompicture}[name=fade right]
  \shade[left color=transparent!0,right color=transparent!100] (0,0) rectangle
    (2,2);
  \fill[transparent!50] (1,1) circle (0.7);
\end{tikzfadingfrompicture}
% 下面在 {tikzpicture} 环境中画出 fading 图
\begin{tikzpicture}
```

```

% 把背景设为 black!20
\fill [black!20] (-1.2,-1.2) rectangle (1.2,1.2);
\pattern [pattern=checkerboard,pattern color=black!30] (-1.2,-1.2) rectangle
(1.2,1.2);
% 使用 fill 命令画出 fading 图
\fill [path fading=fade right,red] (-1,-1) rectangle (1,1);
\end{tikzpicture}

```

上面图形中，定义灰度图时用了 `\shade` 命令，故最终图形中的矩形具有颜色渐变和透明度两个属性。引用灰度图名称 `fade right` 的是最后的命令 `\fill`，它还给出了 `red` 选项，这个命令定义的矩形路径与灰度图的一部分重合，颜色淡出效果只在这一重合部分上显现。



```

% 定义灰度图
\begin{tikzfadingfrompicture}[name=tikz]
  \node [text=transparent!60]
    {\scalebox{5}{\usefont{U}{yfrak}{m}{n}\selectfont TikZ}};
\end{tikzfadingfrompicture}
% 下面在 {tikzpicture} 环境中画出 fading 图
\begin{tikzpicture}
  \fill [black!20] (-2,-1) rectangle (2,1);
  \pattern [pattern=checkerboard,pattern color=black!30] (-2,-1) rectangle (2,1);
  % 使用 shade 命令画出 fading 图
  \shade[path fading=tikz,fit fading=false,left color=blue,right color=black]
    (-2,-1) rectangle (2,1);
\end{tikzpicture}

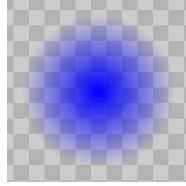
```

`\tikzfading[<options>]`

这个命令的 `<options>` 可以使用以下选项：

1. 用 `name=<name>` 设置名称。
2. 用 `shading` 选项指定一种渐变模式，参考 §65, Shadings Library.
3. 指定渐变模式后，再分别给渐变的始末位置或中间位置赋予 `transparent!<percentage>` 值，来指定各个位置的透明度。

由于在该命令中可以使用 `shading` 选项，故它可以同时引入颜色淡出和颜色渐变两种变化方式，但与环境 `{tikzfadingfrompicture}` 不同，它并不直接定义一个灰度图。当某个路径把 `path fading=<name>` 作为选项后，可以认为该路径就成为一个灰度图，其明度设置由 `<name>` 定义，然后参照这个灰度图画出“它自己”。



```

% 定义灰度图
\tikzfading[name=fade out,
  inner color=transparent!0,
  outer color=transparent!100]
% 下面在 {tikzpicture} 环境中画出 fading 图
\begin{tikzpicture}
  \fill [black!20] (-1.2,-1.2) rectangle (1.2,1.2);
  \path [pattern=checkerboard,pattern color=black!30] (-1.2,-1.2) rectangle
    (1.2,1.2);
  \fill [blue,path fading=fade out] (-1,-1) rectangle (1,1);
\end{tikzpicture}

```

23.4.2 创建 fading 路径

下面介绍使得一个路径具有 fading 效果的相关选项。

`/tikz/path fading=<name>` (默认为环境的设置)

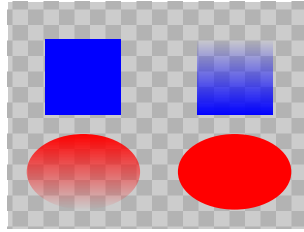
当一个路径带有这个选项后，该路径具有 fading 效果，而环境选项中的 `path fading` 选项不会直接地对路径有效。这里的 `<name>` 可以是环境 `{tikzfadingfrompicture}` 定义的灰度图名称，或命令 `\tikzfading[<options>]` 定义的名称，也可以是 `fadings` 程序库提供的预定义的灰度图名称，具体参考 §51。

`fadings` 程序库提供的预定义的灰度图名称是：

- `west`，左侧透明，右侧不透明。这个选项定义一个中心在 origin，边长大约为 4cm 的矩形灰度图。
- `east`，右侧透明，左侧不透明。这个选项定义一个中心在 origin，边长大约为 4cm 的矩形灰度图。
- `north`，上部透明，下部不透明。这个选项定义一个中心在 origin，边长大约为 4cm 的矩形灰度图。
- `south`，下部透明，上部不透明。这个选项定义一个中心在 origin，边长大约为 4cm 的矩形灰度图。
- `circle with fuzzy edge 10 percent`，这个选项定义一个中心在 origin，半径大约为 1cm 的圆形灰度图。这个选项用作圆、椭圆路径的选项时有效，边界上有半透明的“毛边”，从外向内，毛边透明度从 100% 到 0%，毛边宽度为半径的 10%。
- `circle with fuzzy edge 15 percent`，这个选项定义一个中心在 origin，半径大约为 1cm 的圆形灰度图。这个选项用作圆、椭圆路径的选项时有效，边界上有半透明的“毛边”，从外向内，毛边透明度从 100% 到 0%，毛边宽度为半径的 15%。
- `circle with fuzzy edge 20 percent`，这个选项定义一个中心在 origin，半径大约为 1cm 的圆形灰度图。这个选项用作圆、椭圆路径的选项时有效，边界上有半透明的“毛边”，从外向内，毛边透明度从 100% 到 0%，毛边宽度为半径的 20%。

- fuzzy ring 15 percent, 对圆、椭圆有效。

如果路径上的 path fading 值未被指定, 则其值使用环境选项中 path fading 的值。



```
\begin{tikzpicture}[path fading=south] % 将 path fading 作为环境选项
  \fill [black!20] (0,0) rectangle (4,3);
  \pattern [pattern=checkerboard,pattern color=black!30] (0,0) rectangle (4,3);
  \fill [color=blue] (0.5,1.5) rectangle +(1,1);
  \fill [color=blue,path fading=north] (2.5,1.5) rectangle +(1,1);
  \fill [color=red,path fading] (1,0.75) ellipse (.75 and .5); % 用环境选项的值
  \fill [color=red] (3,0.75) ellipse (.75 and .5);
\end{tikzpicture}
```



```
\begin{tikzpicture}
  \fill [color=red,path fading=circle with fuzzy edge 10 percent]
    (0,0.5) ellipse (0.75 and 0.5);
  \fill [color=red,path fading=circle with fuzzy edge 20 percent]
    (0,-0.5) ellipse (0.75 and 0.5);
\end{tikzpicture}
```

`/tikz/fit fading=<boolean>` (默认值 true, 初始值 true)

若这个选项的值为 true, 则程序会把灰度图做适当的平移和放缩, 并使得灰度图的中心与绘图命令定义的路径的中心重合 (这里说的中心是它们各自的 bounding box 的中心), 使得二者重合部分尽可能地大。若这个选项的值为 false, 则没有对灰度图做适当的平移和放缩, 那么可能没有重合淡出, 或者重合部分过小。



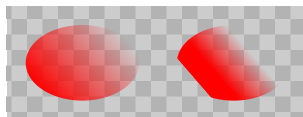
```

\begin{tikzfadingfrompicture}[name=tikz]
\node [text=transparent!50]
  {\fontsize{45}{45}\selectfont Ti\emph{k}Z};
\end{tikzfadingfrompicture}
\begin{tikzpicture}
  \fill [black!20] (-2,-2) rectangle (4,2);
  \pattern [pattern=checkerboard,pattern color=black!30] (-2,-2) rectangle (4,2);
  \draw[path fading=tikz,fit fading=true,red,line width=1cm] (-2,0) -- (2,2)
    (-2,-1.5) -- (2,0.5);
  \shade[path fading=tikz,fit fading=true,left color=blue,right color=black] (2,1)
    rectangle (4,1.6); % 有重合, 文字较扁。
  \shade[path fading=tikz,fit fading=true,left color=blue,right color=black]
    (2,-1.5) rectangle (3,1); % 有重合, 文字较瘦。
  \shade[path fading=tikz,fit fading=false,left color=blue,right color=black]
    (2,-1.6) rectangle (4,-1); % 没有重合, 故没有画出任何点。
\end{tikzpicture}

```

`/tikz/fading transform=<transformation options>` (无默认值)

这个选项值所设定的变换是针对灰度图的, 先将这个选项指出的变换施加于灰度图, 然后再确定“重合淡出”。



```

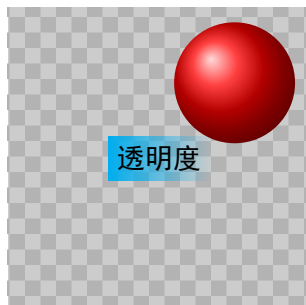
\begin{tikzpicture}[path fading=south]
  \fill [black!20] (0,0) rectangle (4,1.5);
  \path [pattern=checkerboard,pattern color=black!30] (0,0) rectangle (4,1.5);
  \fill [red,path fading,fading transform={rotate=120}] (1,0.75) ellipse (.75 and
    .5);
  \fill [red,path fading,fading transform={rotate=120,yscale=0.4}] (3,0.75) ellipse
    (.75 and .5);
\end{tikzpicture}

```

`/tikz/fading angle=<degree>`

等价于 `fading transform={rotate=<degree>}`.

注意如果 node 算子带有 path fading 选项, 则淡出的是其背景, 而不是其中的文字。



```
\tikzfading[name=fade inside,
  inner color=transparent!100,
  outer color=transparent!40]
\begin{tikzpicture}
\fill [black!20] (0,0) rectangle (4,4);
\path [pattern=checkerboard,pattern color=black!30] (0,0) rectangle (4,4);
\shade [ball color=red] (3,3) circle (0.8);
\shade [ball color=white,path fading=fade inside] (2,2) circle (1.8);
\node [fill=cyan,path fading=east] at(2,2) {\heiti 透明度};
\end{tikzpicture}
```

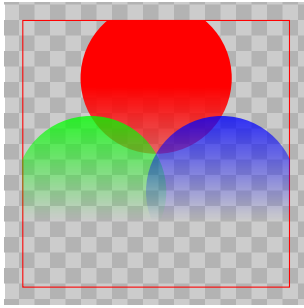
23.4.3 创建 fading 命令组

在环境中，一个路径带有 clip 选项后，该路径作为剪切路径，对它之后（该环境中）的路径具有剪切作用。选项 scope fading 的作用类似于 clip。

```
/tikz/scope fading=<fading>
```

<fading> 可以是环境 {tikzfadingfrompicture} 定义的灰度图名称，或命令 \tikzfading[<options>] 定义的名称，也可以是 fadings 程序库提供的预定义的灰度图名称。当某个路径带有这个选项后，选项 fit fading 和 fading transform 对这个路径仍然有效。默认该路径具有 fit fading=true 的初始设置，故程序会针对该路径（仅针对该路径）来平移或放缩灰度图，然后将这个灰度图应用于该路径以及其后的各个路径。所以该路径的尺寸和位置可以决定具有 fading 效果的区域的范围。

该选项的有效范围受到环境分组的限制。



```
\begin{tikzpicture}
\fill [black!20] (-2,-2) rectangle (2,2);
\pattern [pattern=checkerboard,pattern color=black!30]
(-2,-2) rectangle (2,2);
\draw [red] (-50bp,-50bp) rectangle (50bp,50bp);
\path [scope fading=south,fit fading=false] (0,0);
\fill[red] ( 90:1) circle (1);
\fill[green] (210:1) circle (1);
\fill[blue] (330:1) circle (1);
\end{tikzpicture}
```



```
\begin{tikzpicture}
\tikzfading[name=fade out,inner color=transparent!0,outer color=transparent!100]
\path[use as bounding box] (-3,-1.5) rectangle (3,1.5);
\filldraw [scope fading=fade out,inner color=cyan, outer color=black]
(-3,-1.5) rectangle (3,1.5);
\fill[red] ( 90:1) circle (0.8);
\fill[green] (210:0.7) circle (1);
\fill[blue] (330:2) circle (1.7);
\end{tikzpicture}
```

如果 `scope fading` 用作 `node` 算子的选项, 则 `node` 的背景和文字都具有淡出效果。这与选项 `path fading` 不同。

This is some text that will fade out as we go right and down. It is pretty hard to achieve this effect in other ways.

```
\tikz \node [fill=cyan,scope fading=south,
fading angle=45,text width=3cm]
{
This is some text that will fade out as we go right
and down. It is pretty hard to achieve this effect in
other ways.
};
```

23.5 Transparency Groups

前面介绍混色模式时已经涉及 transparency group 选项，现在看一下这个选项对“叠加透明颜色”的效果的影响。

在下面的图形中，右侧图形中的圆与斜线段都有不透明度，它们重叠部分的不透明度出现叠加，对于一个标志来说这显然不太合适。



```
\begin{tikzpicture}
\node [forbidden sign,line width=2ex,
      draw=red,fill=white]
  at (0,0) {Smoking};
\node [opacity=.5,forbidden sign,line width=2ex,
      draw=red,fill=white]
  at (2.2,0) {Smoking};
\end{tikzpicture}
```

Transparency groups 可以解决这类问题。当环境带有 transparency group 选项后，就成为一个 transparency group.

`/tikz/transparency group=<options>`

这个选项只能用作环境选项。用它就不会出现不透明度叠加的情况。环境内的命令绘图时，会忽略它之前诸命令的不透明度对该命令的影响。不透明度的叠加就是颜色的亮度降低，只要适当提高重叠区域的亮度就可以取消不透明度的叠加，而且还能保留颜色的透明效果。



```
\begin{tikzpicture}
\pattern[pattern=checkerboard,pattern color=black!15](-1,-1) rectangle (3.2,1);
\node at (0,0) [forbidden sign,line width=2ex,draw=red,fill=white] {Smoking};
\begin{scope}[transparency group,opacity=.5]
\node at (2.2,0) [forbidden sign,line width=2ex,draw=red,fill=white] {Smoking};
\end{scope}
\end{tikzpicture}
```

对于带有选项 transparency group 的环境，假设其环境选项中有 opacity=<value1> 选项，如果环境中的命令不带有 opacity 选项，则画出图形的不透明度是 <value1>；如果带有 opacity 选项，例如第 2 个命令带有 opacity=<value2> 选项，则第 2 个命令画出的图形的透明度是 <value1>*<value2>，即两个值的乘积。



```

\begin{tikzpicture}
\pattern[pattern=checkerboard,pattern color=black!15](-1,-1) rectangle (3.2,1);
\node at (0,0) [forbidden sign,line width=2ex,draw=red,fill=white] {Smoking};
\begin{scope}[transparency group,opacity=.5]
\node (s) at (2.2,0) [forbidden sign,line width=2ex,draw=red,fill=white] {Smoking};
\draw [opacity=.5, line width=2ex, red] (1.4,0) -- (3,0);
\draw [opacity=1,line width=2ex, red] (1.4,-.4) -- (3,-.4);
\draw [line width=2ex, red] (1.4,0.4) -- (3,0.4);
\end{scope}
\end{tikzpicture}

```

上面图形中，右侧图中中间的横线的不透明度值是 $0.5*0.5=0.25$ 。

选项 `transparency group=[<options>]` 中的 `<options>` 可以是以下值：

- `knockout`，这个选项导致“透视镜”效果。环境中的某个路径带有 `opacity=<value>` 选项后，该路径就成为一个“透视镜”，透过它可以“看穿”一层障碍——这层障碍当然是之前创建的、被该路径遮挡的那些点。`<value>` 决定透视的清晰度。
注意有的渲染器（renderer）不支持这个功能。
- `isolated=false`，在默认下，`transparency group` 是被隔离的。将 `<options>` 设为 `isolated=false` 则取消隔离，详情参考 PDF Reference, six edition, §7.3.4。

TikZ 会自动计算图形的位置和尺寸，如果绘图时使用了 `overlay` 或 `transform canvas` 选项，可能导致 TikZ 无法顺利计算坐标位置。参考 §110。

24 装饰路径

24.1 Overview

下面用几个例子展示一下什么是装饰路径（decorated paths），为了能顺利编译涉及的各种装饰路径的代码，首先调用各种相关程序库：

- `decorations`
- `decorations.pathmorphing`
- `decorations.pathreplacing`
- `decorations.markings`
- `decorations.footprints`
- `decorations.shapes`
- `decorations.text`

- decorations.fractals

当调用以 decorations 为前缀的程序库后, decorations 程序库会被自动加载。

画一个由直线段和圆弧构成的路径:



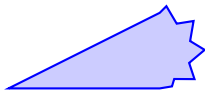
```
\tikz \fill [fill=blue!20,draw=blue,thick]
(0,0) -- (2,1) arc (90:-90:.5) -- cycle;
```

然后给绘图命令添加 decorate, decoration 选项, 将直线段和圆弧换成 zigzag 线型:



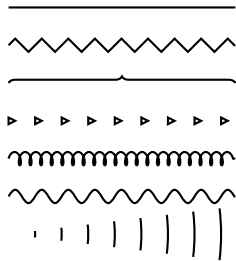
```
\tikz \fill [decorate,decoration={zigzag}]
[fill=blue!20,draw=blue,thick]
(0,0) -- (2,1) arc (90:-90:.5) -- cycle;
```

也可以保留直线段, 仅把圆弧换成 zigzag 线型:



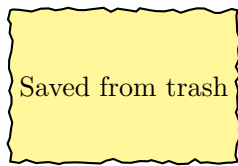
```
\tikz \fill [decoration={zigzag}]
[fill=blue!20,draw=blue,thick]
(0,0) -- (2,1) decorate { arc (90:-90:.5) } -- cycle;
```

装饰路径就是用具有装饰效果的线型或者标记来替换原有的直线或者曲线。有数个程序库 (如前列出) 分别提供多种不同的线型和标记以供使用。



```
\begin{tikzpicture}[thick]
\draw (0,3) -- (3,3);
\draw[decorate,decoration=zigzag] (0,2.5) -- (3,2.5);
\draw[decorate,decoration=brace] (0,2) -- (3,2);
\draw[decorate,decoration=triangles] (0,1.5) -- (3,1.5);
\draw[decorate,decoration={coil,segment length=4pt}]
(0,1) -- (3,1);
\draw[decorate,decoration={coil,aspect=0}]
(0,.5) -- (3,.5);
\draw[decorate,decoration={expanding waves,angle=7}]
(0,0) -- (3,0);
\end{tikzpicture}
```

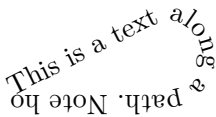
也可以装饰 node 的轮廓线条:



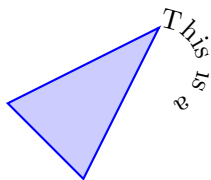
```
\begin{tikzpicture}
\node [fill=yellow!50,draw,thick,
      minimum height=2cm, minimum width=3cm,
      decorate,
      decoration={random steps,segment length=3pt,
                  amplitude=1pt}]
{Saved from trash};
\end{tikzpicture}
```

有 3 种不同类型的路径装饰：

1. path morphing, 即路径变体, 这种装饰不改变原路径的连续性和子路径个数。这种装饰线型由程序库 `decorations.pathmorphing` 提供, 例如 `zigzag`, `bumps`, `coil`, `snake`, `saw` 等线型。
2. path replacing, 路径替换, 将某种特殊的符号或标记沿着路径放置, 路径不再连续, 故不能填充颜色。这种装饰由程序库 `decorations.pathreplacing` 和 `decorations.shapes` 提供, 例如 `brace`, `ticks`, `waves`, `crosses`, `triangles` 等。
3. path removing, 路径清除, 将被装饰的路径清除, 将文字或 `node` 沿着被装饰的路径放置。原路径的选项 (如 `draw`, `color=`, `fill` 等) 对放置的文字或 `node` 没有作用。如果被装饰路径是主路径的一段子路径, 在装饰完这段子路径后继续构建主路径, 构建主路径时会忽略这一段子路径, 也就是说, 一旦某个路径 (或子路径) 被装饰, 那么被装饰部分会被程序 “丢掉”。



```
\tikz \fill [decorate,
            decoration={text along path,
                        text=This is a text along a path.
                        Note how the path is lost.}]
[fill=blue!20,draw=blue,thick,red]
(0,0) -- (2,1) arc (90:-90:.5) -- cycle;
```

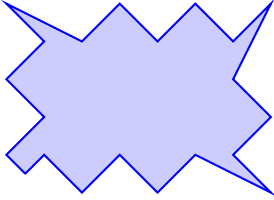


```
\tikz \fill [fill=blue!20,draw=blue,thick]
(0,0) -- (2,1)
decorate [decoration={text along path,
                    text=This is a text along a path.
                    Note how the path is lost.}]
{arc (90:-90:.5)}
-- (1,-1)--cycle;
```

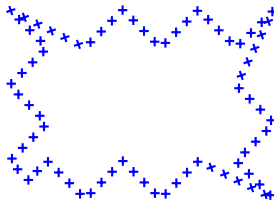
装饰路径操作可以套嵌使用。



```
\tikz \fill [fill=blue!20,draw=blue,thick]
(0,0) rectangle (3,2);
```



```
\tikz \fill [fill=blue!20,draw=blue,thick]
decorate[decoration={zigzag,
segment length=10mm,amplitude=2.5mm}]
{(0,0) rectangle (3,2)};
```



```
\tikz \fill [fill=blue!20,draw=blue,thick]
decorate[decoration={crosses,
segment length=2mm}]
{decorate[decoration={zigzag,
segment length=10mm,amplitude=2.5mm}]
{(0,0) rectangle (3,2)}
};
```

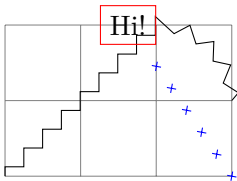
装饰路径的排版比较慢，也不太精确，这是因为 PGF 要做大量计算，而 $\text{T}_\text{E}_\text{X}$ 并不太擅长计算。

24.2 用路径算子 decorate 装饰子路径

用路径算子 decorate 可以构成主路径的子路径，基本句法是：

```
\path . . . decorate[<options>]{<subpath>} . . . ;
```

算子 decorate 引起对 <subpath> 的装饰。<subpath> 中可以含有直线段，曲线，圆弧，椭圆弧，椭圆，圆，矩形，或已经装饰过的路径。可以在 <subpath> 中使用 node，但 node 只是添加到 <subpath> 上的，不属于该 <subpath>，故 node 不被装饰。



```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw decorate [decoration={name=zigzag}]
{(0,0) -- (2,2) node (hi) [left,draw=red] {Hi!}
arc(90:0:1)};
\draw [blue] decorate [decoration={crosses}]
{(3,0) -- (hi)};
\end{tikzpicture}
```

<options> 中使用以下选项：

```
/pgf/decoration=<decoration options>
```

```
/tikz/decoration=<decoration options>
```

这个选项指定装饰类型的名称，也可以用作环境选项。装饰路径的局部选项对全局选项有“优先权”。不同类型的装饰有不同的选项，具体参考 §48。

`/pgf/decoration/name=<name>` (无默认值, 初始值 `none`)

注意这个 key 用在 `decoration` 之下时，用于指定装饰类型的名称，可以省略 `name=` 而只写出名称。如果令 `name=none` 则取消装饰。

前面提到，如果被装饰路径是主路径的一段子路径，在装饰完这段子路径后，继续构建主路径，构建主路径时会忽略这一段子路径。

前面也提到，装饰操作可以套嵌使用，可以用这种方式制作分形图形。



```
\begin{tikzpicture}[decoration=Koch snowflake,
  draw=blue,fill=blue!20,thick]
\filldraw (0,0) -- ++(60:1) -- ++(-60:1) -- cycle ;
\filldraw decorate{ (0,-1) -- ++(60:1) -- ++(-60:1) -- cycle };
\filldraw decorate{ decorate{ (0,-2.5) -- ++(60:1) -- ++(-60:1)
  -- cycle }};
\end{tikzpicture}
```

24.3 装饰整个路径

下面两句等效：

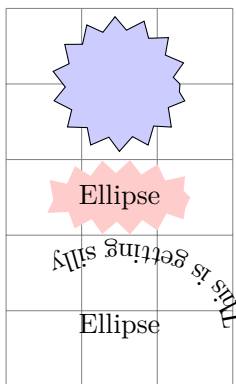
`\path decorate[<options>] {<path>};`

`\path [decorate,<options>] <path>;`

第一句是前一小节所讲，其中 `decorate` 作为绘图算子使用。第二句中 `decorate` 作为选项使用。

`/tikz/decorate=<boolean>` (默认值 `true`)

这个选项作为绘图命令的选项，对该命令画出的路径启用装饰功能。但是使用该选项两次不意味着套嵌装饰操作，因为作为选项的 `decorate` 只是意味着逻辑值 `true`，这与作为算子的 `decorate` 很不一样。

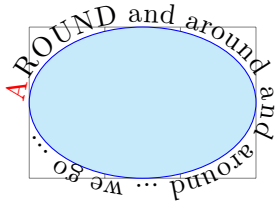


```
\begin{tikzpicture}[decoration=zigzag]
\draw [help lines] (0,0) grid (3,5);
\draw [fill=blue!20,decorate] (1.5,4) circle (0.8cm);
\node at (1.5,2.5) [fill=red!20,ellipse,decorate] {Ellipse};
\node at (1.5,0.8) [inner sep=6mm,fill=red!20,ellipse,
  decorate,decoration=
    {text along path,text={This is getting silly}}]
  {Ellipse};
\end{tikzpicture}
```

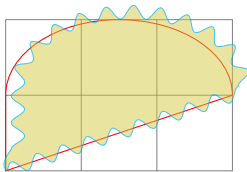
上面例子中，选项 `text along path` 清除了原来的被装饰路径，并使得文字沿着原来的被装饰路径排出。在默认下，文字会排在路径的左侧。上面例子中，最后一个 `node` 的形状是 `ellipse`，其路径方向是逆时

针的。

可以将装饰选项作为选项 `preaction` 或 `postaction` 的值添加到绘图命令选项中，这样在画出主路径之前或之后再画出装饰路径会，从而使被装饰路径仍然能得到显示。



```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\fill [draw=blue,fill=cyan!20,
      postaction={decorate,
                  decoration={raise=2pt,text along path,
                              text={{\color{red}A}ROUND and around and around
                                      ... we go ...}}}]
(0,1) arc (180:-180:1.5cm and 1cm);
\end{tikzpicture}
```



```
\begin{tikzpicture}[decoration=snake]
\draw [help lines] grid (3,2);
\draw [postaction={decorate,fill=yellow!80!black,
                  fill opacity=0.5,draw=cyan,draw opacity=0.7},red]
(0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```

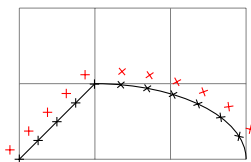
24.4 调整装饰路径的外观

24.4.1 调整装饰路径与原被装饰路径的相对位置

下面的选项只能用于 TikZ 中。

`/pgf/decoration/raise=<dimension>` (无默认值, 初始值 0pt)

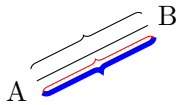
这个选项设置装饰路径偏离原被装饰路径的距离，默认沿着路径方向“向左偏”，它在选项 `transform` (见下文) 作用后再起作用。如果 `<dimension>` 是负值尺寸，则装饰路径沿着路径方向“向右偏”。



```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) arc (90:0:2 and 1);
\draw decorate [decoration=crosses]
{ (0,0) -- (1,1) arc (90:0:2 and 1) };
\draw[red] decorate [decoration={crosses,raise=5pt}]
{ (0,0) -- (1,1) arc (90:0:2 and 1) };
\end{tikzpicture}
```

`/pgf/decoration/mirror=<boolean>` (默认值 true)

将原被装饰路径作为“镜面”，将装饰路径从“镜面”的一侧变到另一侧。它在选项 `transform`（见下文）和 `raise` 作用后再起作用。



```
\begin{tikzpicture}
\node (a) {A};
\node (b) at (2,1) {B};
\draw (a) -- (b);
\draw[decorate,decoration={brace,raise=5pt}] (a) -- (b);
\draw[decorate,decoration={brace,raise=-5pt},red] (a) -- (b);
\draw[decorate,decoration={brace,mirror,raise=5pt},
      blue,line width=2pt]
      (a) -- (b);
\end{tikzpicture}
```

注意上面例子中，`raise=-5pt` 与 `mirror, raise=5pt` 不一样。

`/pgf/decoration/transform=<transformations>`

这里的 `<transformations>` 是通常的 TikZ 变换，如 `shift`，`rotate`，变换是针对装饰路径的，该选项会在前面讲的选项 `raise` 和 `mirror` 之前起作用。

24.4.2 调整装饰路径的始端与终端的形态

装饰路径从原路径的起点延续到终点，装饰路径的形态可能会使得原路径的起点和终点不容易辨认，也不够美观。这时候就需要调整装饰路径的始端与终端的形态，调整的方法不唯一，比较便利的是使用下面的选项，注意它们只能用于 `decorations`，不能用于 `meta-decorations`。

`/pgf/decoration/pre=<decoration>` （无默认值，初始值 `lineto`）

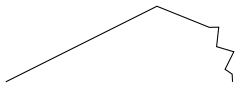
`<decoration>` 可以是 `lineto`，`curveto`，`moveto`，或者其它装饰路径类型。这个选项使得装饰路径的始端点在原路径的起点之后（仍在原路径上），而两点间的联系方式由 `<decoration>` 指定，两点间的联系长度由选项 `pre length` 规定。`<decoration>` 的默认值是 `lineto`，即用直线段联系两点；值 `curveto` 用曲线联系两点；`moveto` 使得两点间没有连线。



```
\begin{tikzpicture}
\tikz [decoration={zigzag,pre=crosses,pre length=1cm}]
\draw [decorate] (0,0) -- (2,1) arc (90:0:1);
\end{tikzpicture}
```

`/pgf/decoration/pre length=<dimension>` （无默认值，初始值 `0pt`）

作用见上一选项。



```
\begin{tikzpicture}
\tikz [decoration={zigzag,pre length=3cm}]
\draw [decorate] (0,0) -- (2,1) arc (90:0:1);
\end{tikzpicture}
```



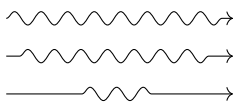
```
\begin{tikzpicture}
\tikz [decoration={zigzag,pre=curveto,pre length=3cm}]
\draw [decorate] (0,0) -- (2,1) arc (90:0:1);
\end{tikzpicture}
```

/pgf/decorations/post=<decoration> (无默认值, 初始值 `lineto`)

作用类似 `pre`, 使得装饰路径的终端点在原路径的终点之前 (仍在原路径上), 而两点间的联系方式由 `<decoration>` 指定, 两点间的联系长度由选项 `post length` 规定。

/pgf/decoration/post length=<dimension> (无默认值, 初始值 `0pt`)

作用见上一选项。



```
\begin{tikzpicture}
[decoration=snake,
line around/.style={decoration={pre length=#1,post length=#1}}]
\draw[->,decorate] (0,0) -- ++(3,0);
\draw[->,decorate,line around=5pt] (0,-5mm) -- ++(3,0);
\draw[->,decorate,line around=1cm] (0,-1cm) -- ++(3,0);
\end{tikzpicture}
```

48 装饰程序库

48.1 公共选项

装饰程序库定义了许多装饰路径的装饰路径。关于装饰路径的用法参考 §24, §98。

装饰程序库定义了许多装饰样式, 有一些选项对很多装饰样式都有效, 是公共选项, 这些选项由 `decoration` 模块直接定义。有的选项只针对某个装饰样式有效, 这种选项由相应的装饰程序库定义。下面介绍公共选项, 有的公共选项的值保存在 `TEX` 寄存器中, 有的公共选项的值保存在宏中。

注意这些选项 (key) 的前缀都是 `/pgf/decoration/`, 因此它们都作选项 `decoration={<options>}` 的值。

/pgf/decoration/amplitude=<dimension> (无默认值, 初始值 `2.5pt`)

这个选项设置装饰路径的“振幅” (amplitude), 例如, 装饰样式 `zigzag` 是沿着被装饰路径放置的“之字形”装饰路径, 之字形装饰路径的典型形式是 \sim , 这是个振动形式, 其振幅是它的高度的一半。

本选项通过重设 `TEX` 寄存器 `\pgfdecorationsegmentamplitude` 的值来发挥作用, 可以直接修改这个寄存器的值来调节装饰路径的振幅。

`/pgf/decoration/meta-amplitude=<dimension>` (无默认值, 初始值 2.5pt)

这个选项针对 meta-decoration 的振幅, 参考 §98.5.

本选项设置 TeX 宏 `\pgfmetadecorationsegmentamplitude` 的值。

`/pgf/decoration/segment length=<dimension>` (无默认值, 初始值 10pt)

有的装饰路由小线段构成 (如 zigzag), 本选项设置这些小线段的长度。

本选项设置 TeX 寄存器 `\pgfdecorationsegmentlength` 的值

`/pgf/decoration/meta-segment length=<dimension>` (无默认值, 初始值 1cm)

这个选项针对 meta-decoration 的小线段的长度。

本选项设置 TeX 宏 `\pgfmetadecorationsegmentlength` 的值。

`/pgf/decoration/angle=<degree>` (无默认值, 初始值 45)

有的装饰样式具有“角度”属性, 例如, 装饰样式 wave, 它由数个小圆弧组成, 小圆弧有自己的角度。

本选项设置 TeX 宏 `\pgfdecorationsegmentangle` 的值。

`/pgf/decoration/aspect=<factor>` (无默认值, 初始值 0.5)

有的装饰样式具有“宽高比例”属性, 例如, 装饰样式 brace 是个大括号, 它有宽高比。

本选项设置 TeX 宏 `\pgfdecorationsegmentaspect` 的值。

`/pgf/decoration/start radius=<dimension>` (无默认值, 初始值 2.5pt)

本选项的值直接保存在它自己这里。

`/pgf/decoration/end radius=<dimension>` (无默认值, 初始值 2.5pt)

本选项的值直接保存在它自己这里。

`/pgf/decoration/radius=<dimension>` (无默认值)

本选项的值直接保存在它自己这里。

`/pgf/decoration/path has corners=<boolean>` (无默认值, 初始值 false)

本选项决定装饰路径是否采用圆角。如果装饰路径本身是有“尖角”的, 那么设置本选项值为 true 可能会改善装饰路径的外观, 但如果装饰路径本身没有尖角, 或者组成装饰路径的线段太短, 那么设置本选项值为 true 可能会出现意外状况。

本选项设置 TeX-if `\ifpgfdecoratepathhascorners` 的值。

下面以装饰样式 zigzag 和 straight zigzag 为例, 看一下选项 amplitude, meta-amplitude, segment length 是如何起作用的。为了顺利理解下面的代码, 请先阅读 §98.

在程序库 decorations.pathmorphing 的源文件 `pgflibrarydecorations.pathmorphing.code` 中对装饰路径 zigzag 的定义如下:

```
\pgfdeclaredecoration{zigzag}{up from center}{
  \state{up from center}[width=+.5\pgfdecorationsegmentlength, next state=big down]
  {
```

```

\pgfpathlineto{\pgfqpoint{.25\pgfdecorationsegmentlength}{\
  pgfdecorationsegmentamplitude}}
}
\state{big down}[switch if less than+.5\pgfdecorationsegmentlength to center finish,
  width+.5\pgfdecorationsegmentlength,
  next state=big up]
{
  \pgfpathlineto{\pgfqpoint{.25\pgfdecorationsegmentlength}{-\
    pgfdecorationsegmentamplitude}}
}
\state{big up}[switch if less than+.5\pgfdecorationsegmentlength to center finish,
  width+.5\pgfdecorationsegmentlength,
  next state=big down]
{
  \pgfpathlineto{\pgfqpoint{.25\pgfdecorationsegmentlength}{\
    pgfdecorationsegmentamplitude}}
}
\state{center finish}[width=0pt, next state=final]{
  \pgfpathlineto{\pgfpointorigin}
}
\state{final}
{
  \pgfpathlineto{\pgfpointdecoratedpathlast}
}
}

```

以上定义代码规定了 5 个状态，即 `up from center`，`big down`，`big up`，`center finish`，`final`。在用 `zigzag` 装饰路径时，可能出现以下几种状态组合：

- `final`
- `up from center` → `final`
- `up from center` → `big down` → `final`
- `up from center` → `big down` → `center finish` → `final`
- `up from center` → `big down` → `big up` → `final`
- `up from center` → `big down` → `big up` → `big down` → `final`
-

其中的典型组合是 `up from center` → `big down` → `big up` → `final`，在选项的初始值下这个组合画出的图形相当于

```

~ \tikz \draw (0,0) -- (2.5pt,2.5pt)--(7.5pt,-2.5pt)--(10pt,0pt);

```

从定义代码看，这个图形的宽度就是 `\pgfdecorationsegmentlength` 的值，即选项 `segment length` 的值：

这个图形的高度是 `\pgfdecorationsegmentamplitude` 的值的 2 倍，即“振幅”是选项 `amplitude` 的值。

文件 `pgflibrarydecorations.pathmorphing.code` 中对装饰路径 `straight zigzag` 的定义如下：

```
\pgfdeclaremetadecoration{straight zigzag}{line to}{
  \state{line to}[width=\pgfmetadecorationsegmentlength, next state=zigzag]
  {
    \decoration{curveto}
  }
  \state{zigzag}[width=\pgfmetadecorationsegmentlength, next state=line to]
  {
    \decoration{zigzag}
  }
  \state{final}
  {
    \decoration{curveto}
  }
}
```

从上面的定义代码看，`straight zigzag` 样式的典型形式由三段构成，第一段是 `curveto` 装饰样式，第二段是 `zigzag` 装饰样式，第三段是 `curveto` 装饰样式，这三段的宽度都是 `\pgfmetadecorationsegmentlength` 的值，即选项 `meta-segment length` 的值。

48.2 修饰路径的装饰样式

首先调用程序库 `\usepgflibrary{decorations.pathmorphing}`。

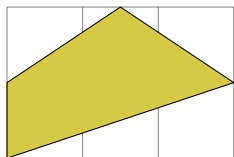
在外观上，修饰路径的装饰样式（`path morphing decoration`）会让被装饰路径变形。例如之字形装饰路径。这种装饰样式不会改变被装饰路径的子路径的个数，会针对每个子路径分别进行装饰。

48.2.1 由直线段构成的装饰路径

装饰样式 `lineto`

`lineto`

这个装饰样式实际上是 `decoration` 模块定义的，它用直线段代替被装饰路径，无论被装饰路径是曲线还是直线。



```
\begin{tikzpicture}[decoration=lineto]
  \draw [help lines] grid (3,2);
  \draw [decorate,fill=yellow!80!black]
    (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```

装饰样式 straight zigzag**straight zigzag**

这个装饰样式由三段已定义的装饰路径构成：曲线装饰路径、之字形装饰路径、曲线装饰路径，因此是 meta-decoration 类型的装饰样式。这个装饰样式中的之字形装饰路径有自己的振幅，所以它是沿着被装饰路径放置的，它的走势随着被装饰路径的弯曲而弯曲。

amplitude

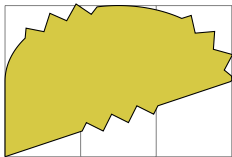
这个选项确定本装饰路径的振幅。

segment length

这个选项确定构成本装饰路径的，之字形路径的一个周期的宽度。

meta-segment length

这个选项确定的长度既是构成本装饰路径的各段的宽度（见前面的定义代码）。



```
\begin{tikzpicture}[decoration={straight zigzag,
    meta-segment length=1.1cm}]
  \draw [help lines] grid (3,2);
  \draw [decorate,fill=yellow!80!black]
    (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```

装饰样式 random steps**random steps**

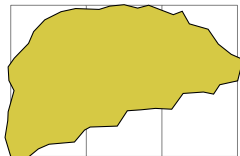
这个装饰样式由许多前后相接的直线段构成，原本每个直线段的终点都应该位于被装饰路径上，不过本装饰样式会使得每个直线段的终点随机地偏离被装饰路径，这个偏离包括水平方向的偏离 h 和竖直方向的偏离 v ， $h, v \in [-d, d]$ ，这里 d 由选项 **amplitude**= d 指定。

segment length

本选项确定构成本装饰路径的小线段的基本长度。

amplitude

本选项的作用如前述。



```
\begin{tikzpicture}
  [decoration={random steps,segment length=2mm}]
  \draw [help lines] grid (3,2);
  \draw [decorate,fill=yellow!80!black]
    (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```

装饰样式 saw

saw

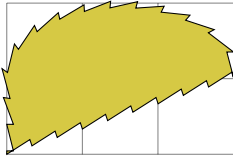
这个装饰路径是锯齿形状的。

amplitude

本选项确定锯齿的振幅。

segment length

本选项确定构成锯齿路径的一个锯齿的宽度。



```
\begin{tikzpicture}[decoration=saw]
  \draw [help lines] grid (3,2);
  \draw [decorate,fill=yellow!80!black]
    (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```

装饰样式 zigzag**zigzag**

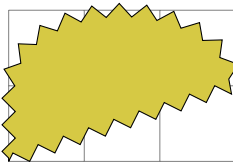
这个装饰路径是之字形路径。

amplitude

本选项确定之字形路径的振幅。

segment length

本选项确定之字形路径的一个周期的宽度。



```
\begin{tikzpicture}[decoration=zigzag]
  \draw [help lines] grid (3,2);
  \draw [decorate,fill=yellow!80!black]
    (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```

48.2.2 由曲线构成的装饰路径**装饰样式 bent****bent**

这个装饰路径由弯曲线条构成。设 $\text{aspect}=t$ ，当前子输入路径的未装饰部分的长度是 r ， $\text{amplitude}=a$ ，这个装饰样式就是控制曲线

```
<current point> .. controls ( $t \cdot r, a$ ) and ( $(1-t) \cdot r, a$ ) .. ( $r, 0$ )
```

amplitude

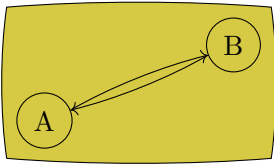
这个选项的值越大，装饰线条的就越是弯曲。若 $\text{amplitude}=0$ 则没有弯曲，等效于装饰样式 `lineto`。

aspect

这个选项的值影响控制曲线的两个支撑点在 x 轴方向的位置。



```
\begin{tikzpicture}
  [decoration={bent,aspect=1.5,amplitude=10mm}]
  \draw [red](0,0)--(2,0);
  \draw [decorate,cyan](0,0)--(2,0);
\end{tikzpicture}
```



```
\begin{tikzpicture}[decoration={bent,aspect=.3}]
  \draw [decorate,fill=yellow!80!black]
    (0,0) rectangle (3.5,2);
  \node[circle,draw] (A) at (.5,.5) {A};
  \node[circle,draw] (B) at (3,1.5) {B};
  \draw[->,decorate] (A) -- (B);
  \draw[->,decorate] (B) -- (A);
\end{tikzpicture}
```

装饰样式 bumps

bumps

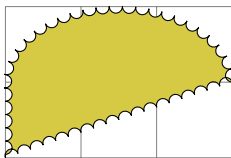
这个装饰样式的构成元素是前后相连的两个“半椭圆弧”。

amplitude

本选项的值确定半椭圆弧的“拱高”。

segment length

本选项的值是两个半椭圆弧的宽度。



```
\begin{tikzpicture}[decoration=bumps]
  \draw [help lines] grid (3,2);
  \draw [decorate,fill=yellow!80!black]
    (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```

装饰样式 coil

coil

这个装饰样式的构成元素是“螺线圈”。

amplitude

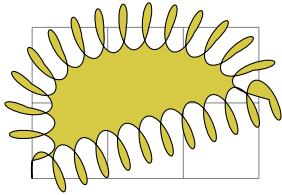
这个选项的值是螺线圈的振幅。

segment length

这个选项的值大约是，螺线转一圈时起止点之间的距离。

aspect

这个选项的值可以调节螺线圈的立体感，一般是它的值越大约具有立体感，但如果它的值过大就会导致装饰路径“走样”。如果它的值是 0，则螺线圈近似平面上的正弦曲线。



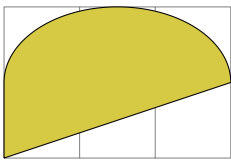
```
\begin{tikzpicture}[decoration={coil,aspect=0.4,
  segment length=3mm,amplitude=3mm}]
  \draw [help lines] grid (3,2);
  \draw [decorate,fill=yellow!80!black]
    (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```

装饰样式 curveto**curveto**

这个装饰样式是 decoration 模块定义的。文件《pgfmoduledecorations.code》中对装饰样式 curveto 的定义如下：

```
\pgfdeclaredecoration{curveto}{initial}{
  \state{initial}[width=\pgfdecoratedinputsegmentlength/100]
  {
    \pgfpathlineto{\pgfpointorigin}
  }
  \state{final}{\pgfpathlineto{\pgfpointdecoratedpathlast}}
}
```

从这个定义看出，curveto 实际上是用折线段来代替原来的路径，替换当前子输入路径的折线段的每个小线段的长度，约是当前子输入路径长度的 $\frac{1}{100}$ 。如果原来的路径是直线段，则替换后的外观还是直线段。如果原来的路径是曲线，而且长度不是过长，外观还是近似曲线。



```
\begin{tikzpicture}[decoration=curveto]
  \draw [help lines] grid (3,2);
  \draw [decorate,fill=yellow!80!black]
    (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```

装饰样式 snake**snake**

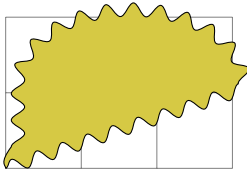
这个装饰样式主要是由正弦曲线构成的。

amplitude

这个选项的值决定振幅。

segment length

这个选项的值决定一个周期的宽度。



```
\begin{tikzpicture}[decoration=snake]
\draw [help lines] grid (3,2);
\draw [decorate,fill=yellow!80!black]
(0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```

48.3 替换路径的装饰样式

调用程序库 `\usepgflibrary{decorations.pathreplacing}`.

装饰样式 `border`

`border`

这个装饰样式在被装饰路径上画出一些小线段，这些小线段与被装饰路径之间有某个角度，这样被装饰路径就“被标记”了。

`segment length`

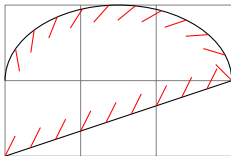
这个选项的值确定相邻两个小线段的间距。

`amplitude`

这个选项的值确定小线段的长度。

`angle`

这个选项的值确定小线段与被装饰路径之间的夹角。



```
\begin{tikzpicture}[decoration={border,amplitude=3mm}]
\draw [help lines] grid (3,2);
\draw [postaction={decorate,draw,red}]
(0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```

装饰样式 `brace`

`brace`

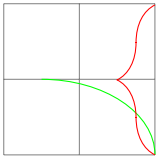
这个装饰样式是一个括号，并且只有一个括号，括号的起点位于被装饰路径的起点，括号的方向是被装饰路径在起点处的切线方向；括号的跨度是被装饰路径的总长度。所以当被装饰路径是直线段时，本装饰样式的效果较好。

`amplitude`

这个选项的值确定括号的“拱高”。

`aspect`

这个选项的值影响括号尖点的位置，最好保持默认值 0.5。



```
\begin{tikzpicture}[decoration={brace,amplitude=5mm}]
  \draw [help lines] grid (-2,2);
  \draw [postaction={decorate,draw,red}][green]
    (0,0) arc (0:90:1.5 and 1);
\end{tikzpicture}
```

装饰样式 `expanding waves`

`expanding waves`

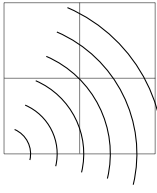
这个装饰样式是“逐渐扩散的波形”，装饰片段是圆弧，沿着被装饰路径摆放一些圆弧，圆弧的尺寸越来越大，用以模仿“波动”。

`segment length`

这个选项的值确定相邻两个圆弧的间距。

`angle`

这个选项的值是圆弧角度值的一半。



```
\begin{tikzpicture}[decoration={expanding waves,angle=40}]
  \draw [help lines] grid (2,2);
  \draw [decorate] (0,0) -- (2,1);
\end{tikzpicture}
```

装饰样式 `moveto`

`moveto`

这个装饰样式是 `decoration` 模块定义的，它直接跳到被装饰路径的终点，常用在选项 `pre=moveto` 或 `post=moveto` 中。

装饰样式 `ticks`

`ticks`

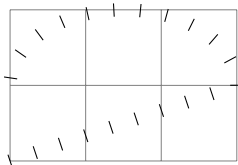
这个装饰样式是——沿着被装饰路径添加“刻度线”。

`segment length`

这个选项值确定相邻两个刻度线的间距。

`amplitude`

这个选项值确定刻度线的长度。



```
\begin{tikzpicture}[decoration=ticks]
  \draw [help lines] grid (3,2);
  \draw [decorate] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```

装饰样式 waves**waves**

这个装饰样式类似 `expanding waves`，都是由圆弧构成的，只是这个装饰样式中的圆弧尺寸保持不变。

segment length

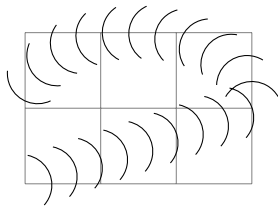
这个选项值确定相邻两个圆弧的间距。

angle

这个选项的值是圆弧角度值的一半。

radius

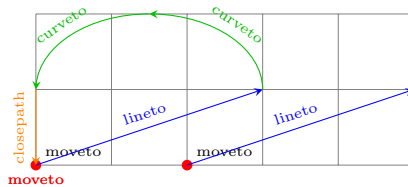
这个选项的值是圆弧的半径。



```
\begin{tikzpicture}[decoration={waves,radius=4mm,angle=60}]
  \draw [help lines] grid (3,2);
  \draw [decorate] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```

装饰样式 show path construction**show path construction**

被装饰路径可能由数种不同类型的“子输入路径”构成，例如被装饰路径可能含有 `moveto`, `lineto`, `curveto`, `closepath` 操作，不同操作构建不同类型的子输入路径。这个装饰样式可以针对各种类型的子输入路径分别进行装饰。



```
\begin{tikzpicture}[>=stealth, every node/.style={midway, sloped, font=\tiny},
  decoration={show path construction,
  moveto code={
    \fill [red] (\tikzinputsegmentfirst) circle (2pt)
    node [fill=none, below] {moveto};
    \pgftext[at={\pgfpointdecoratedinputsegmentfirst},bottom,left] {\tikz\node{\tiny moveto}};},
  lineto code={
    \draw [blue,->] (\tikzinputsegmentfirst) -- (\tikzinputsegmentlast)
    node [above] {lineto}};},
  curveto code={
```

```

\draw [green!75!black,->] (\tikzinputsegmentfirst) .. controls
  (\tikzinputsegmentsupporta) and (\tikzinputsegmentsupportb) ..
  (\tikzinputsegmentlast) node [above] {curveto};},
closepath code={
  \draw [orange,->] (\tikzinputsegmentfirst) -- (\tikzinputsegmentlast)
    node [above] {closepath};}
}]
\draw [help lines] grid (5,2);
\path [decorate] (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle (2,0) -- (5,1);
\end{tikzpicture}

```

使用下面的选项来分别为各个类型子输入路径设置装饰路径。

`/pgf/decoration/moveto code=<code>` (无默认值, 初始值 `{}`)

这个选项的 `<code>` 针对的是 `moveto` 操作的点。如前面的例子所示, 使用命令 `\node` 或者 `node` 算子给 `moveto` 操作的点加 `node` 时, 所加的 `node` 只能位于原点附近, 即所加的 `node` 的指向点只能是原点。要想使得所加的 `node` 跟随 `moveto` 操作的点, 应当使用命令 `\pgftext`。

`/pgf/decoration/lineto code=<code>` (无默认值, 初始值 `{}`)

这个选项的 `<code>` 针对的是 `lineto` 操作的点。

`/pgf/decoration/curveto code=<code>` (无默认值, 初始值 `{}`)

这个选项的 `<code>` 针对的是 `curveto` 操作的点。

`/pgf/decoration/closepath code=<code>` (无默认值, 初始值 `{}`)

这个选项的 `<code>` 针对的是 `closepath` 操作的点。

在以上选项的 `<code>` 中, 可以使用下面的宏来引用需要的点。

`\pgfpointdecoratedinputsegmentfirst`

`\tikzinputsegmentfirst`

这两个宏保存的是当前子输入路径的第一个“构造点”, 注意不是子路径的第一个“构造点”。第一个宏用在 PGF 命令中, 第二个宏用在 TikZ 命令中。

`\pgfpointdecoratedinputsegmentlast`

`\tikzinputsegmentlast`

这两个宏保存的是当前子输入路径 (不是子路径) 的最后一个“构造点”, 第一个宏用在 PGF 命令中, 第二个宏用在 TikZ 命令中。

`\pgfpointdecoratedinputsegmentsupporta`

`\tikzinputsegmentsupporta`

这两个宏保存的是当前的, 由 `curveto` 操作构建的子输入路径的第一个支撑点, 第一个宏用在 PGF 命令中, 第二个宏用在 TikZ 命令中。

`\pgfpointdecoratedinputsegmentsupportb`

`\tikzinputsegmentsupportb`

这两个宏保存的是当前的，由 `curveto` 操作构建的子输入路径的第二个支撑点，第一个宏用在 PGF 命令中，第二个宏用在 TikZ 命令中。

48.4 标记装饰

这种装饰样式是沿着被装饰路径放置标记符号。由于历史的原因，有 3 个不同程序库提供多种标记装饰样式。后文逐次介绍这三个程序库。

48.5 自选标记装饰

48.5.1 程序库 `decorations.markings`

先调用程序库 `\usepgflibrary{decorations.markings}`，这个程序库提供 `markings` 装饰样式，允许你自己选择或自己定义一种标记 (mark) 类型来装饰路径。

装饰样式 `markings`

`markings`

这个装饰样式允许你自己定义一个标记 (mark)，也就是说，你可以用绘图代码自己画一个标记，用于装饰路径。绘制标记的代码被放入一个局部域中来执行。

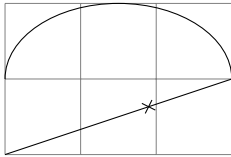
例如，可以用下面的代码

```
\draw (-2pt,-2pt) -- (2pt,2pt);
\draw (2pt,-2pt) -- (-2pt,2pt);
```

定义一个叉号来作为标记。放置标记时，使用某个选项来确定被装饰路径上的一系列点： P_1, P_2, \dots 标记就放在这些点上。在点 P_i 放置标记时，PGF 会开启一个 $\text{T}_\text{E}_\text{X}$ 分组，将绘制标记的代码放入这个 $\text{T}_\text{E}_\text{X}$ 分组中执行。绘制标记的代码需要在一个坐标系内实现，这个坐标系类似路径在点 P_i 处的“自然坐标系”，程序会使用（顶层的）坐标变换使得这个坐标系的原点位于点 P_i 处（平移），其 x 轴沿着路径在点 P_i 处的切线方向， y 轴与 x 轴成右手系（旋转）。因此如果绘制标记的代码中含有 `node` 并且其选项中有 `transform shape`，那么该 `node` 就会接受这个（顶层的）坐标变换。

你可以将标记装饰选项作为 `postaction` 的值，从而在装饰过程结束后还能显示被装饰路径。

下面是个例子。



```

\begin{tikzpicture}[decoration={
  markings,% 选定装饰样式 markings
  mark= % 确定标记位置
    at position 2cm
    with
    {
      \draw (-2pt,-2pt) -- (2pt,2pt);
      \draw (2pt,-2pt) -- (-2pt,2pt);
    }
}]
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,1)
  arc (0:180:1.5 and 1);
\end{tikzpicture}

```

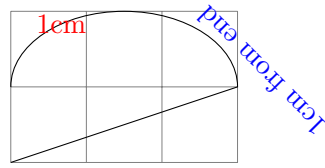
`/pgf/decoration/mark=at position <pos> with <code>`

如上面的例子所示，这里 `<code>` 就是绘制标记的代码，其中可以使用 `node` 算子。`<pos>` 用于决定标记在路径上的位置，`<pos>` 可以有 4 种形式：

1. `<pos>` 可以是非负值的尺寸，例如 `0pt`，或者 `5cm/2`。此时程序会从被装饰路径的起点开始，沿着被装饰路径行进 `<pos>` 指定的长度从而确定点 `p`，这个点 `p` 就是放置标记的位置。
2. `<pos>` 可以是负值的尺寸，例如 `-2pt`，或者 `-1sp`。此时程序会从被装饰路径的终点开始，沿着被装饰路径的反方向行进 `<pos>` 指定的尺寸长度从而确定点 `p`，这个点 `p` 就是放置标记的位置。
3. `<pos>` 可以是非负数字或数字运算表达式，例如 `1/2`，或者 `0.33+2*0.1`。此时程序会从被装饰路径的起点开始计算其总长度 `L`，然后由 `<pos>` 与 `L` 的乘积确定一个长度从而确定一个点 `p`，这个点 `p` 就是放置标记的位置。如果 `<pos>` 是 `0.5`，那么标记就放在被装饰路径的中间 (`L/2`) 处。
4. `<pos>` 可以是负的数字或数字运算表达式，例如 `-1/2`，或者 `-0.33-2*0.1`。此时程序会从被装饰路径的终点开始计算其总长度 `L`，然后由 `<pos>` 与 `L` 的乘积确定一个长度从而确定一个点 `p`，这个点 `p` 就是放置标记的位置。

注意如果 `<pos>` 是绝对值过大的数字或数字运算表达式，例如 `1.2`，会导致标记位置超出被装饰路径，此时没有标记画出。

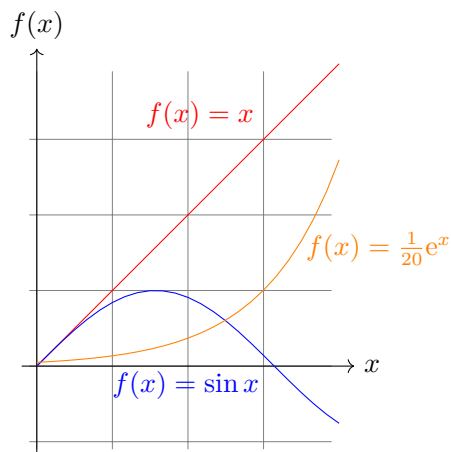
使用这个选项一次只能决定一个标记位置，你可以多次使用这个选项添加多个标记。假如多次使用这个选项，设第 i 次使用该选项确定的位置是点 P_i ，第 $i+1$ 次使用该选项确定的位置是点 P_{i+1} ，那么从被装饰路径的行进方向看，最好确保点 P_i 在点 P_{i+1} 之前，否则可能造成混乱。下面的例子中就出现了混乱：



mid

```
\begin{tikzpicture}[decoration={
  markings,% switch on markings
  mark=at position 6cm with \node[red]{1cm};,
  mark=at position 1cm with \node[green]{mid};,
  mark=at position -4cm with {\node[blue,transform shape]{1cm from end};}]
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```

下面的例子展示了如何用 `markings` 装饰样式给路径加标签。



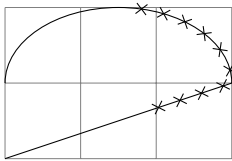
```
\begin{tikzpicture}[domain=0:4,label/.style={postaction={
  decorate,
  decoration={
    markings,
    mark=at position .75 with \node #1;}}}]
\draw[very thin,color=gray] (-0.1,-1.1) grid (3.9,3.9);
\draw[->] (-0.2,0) -- (4.2,0) node[right] {$x$};
\draw[->] (0,-1.2) -- (0,4.2) node[above] {$f(x)$};
\draw[red,label={[above left]{$f(x)=x$}}] plot (\x,\x);
\draw[blue,label={[below left]{$f(x)=\sin x$}}] plot (\x,{sin(\x r)});
\draw[orange,label={[right]{$f(x)=\frac{1}{20}\mathrm{e}^x$}}] plot (\x,{0.05*exp
```



```
(\x));
\end{tikzpicture}
```

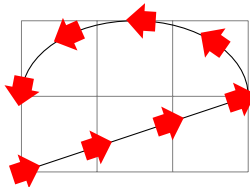
`/pgf/decoration/mark=betweenpositions <start pos> and <end pos> step <stepping> with <code>`

这个选项可以确定被装饰路径上的数个位置, 这些位置用来放置标记。这里对 `<start pos>`, `<end pos>`, `<stepping>` 这 3 个参数的规定类似前面选项的 `<pos>`. `<start pos>` 和 `<end pos>` 分别指定被装饰路径上的点 P_1 和 P_2 , 从 P_1 到 P_2 这一部分路径是需要被装饰的。 `<stepping>` 指定相邻两个标记的间距。



```
\begin{tikzpicture}[decoration={markings,
  mark=between positions 0.3 and 0.7 step 3mm
  with { \draw (-2pt,-2pt) -- (2pt,2pt);
\draw (2pt,-2pt) -- (-2pt,2pt); }} ]
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,1)
  arc (0:180:1.5 and 1);
\end{tikzpicture}
```

下面的例子中使用形状 `single arrow` 来装饰路径。

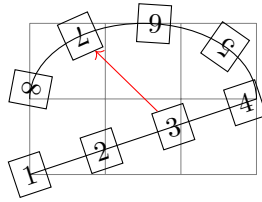


```
\begin{tikzpicture}[decoration={markings,
  mark=between positions 0 and 1 step 1cm
  with {
    \node [single arrow,fill=red,
      single arrow head extend=3pt,
      transform shape] {};}]}
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,1)
  arc (0:180:1.5 and 1);
\end{tikzpicture}
```

在前两个选项的 `<code>` 中可以使用以下选项。

`/pgf/decoration/mark info/sequence number`

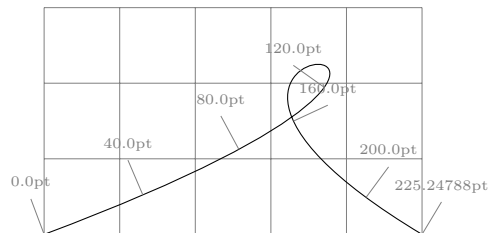
当用前两个 `mark` 选项为装饰路径设置一个或数个标记时, 程序会按照这些标记被添加的次序为它们编号。添加的第一个标记编号是 1, 添加的第二个标记编号是 2……当前标记的编号就保存在这个键 (key) 中, 可以使用命令 `\pgfkeysvalueof{/pgf/decoration/mark info/sequence number}` 来引用或者输出当前标记的编号。



```
\begin{tikzpicture}[decoration={markings, mark=between positions 0 and 1 step 1cm
with {
\node [draw,name=mark-\pgfkeysvalueof{/pgf/decoration/mark info/sequence number},
transform shape]
{\pgfkeysvalueof{/pgf/decoration/mark info/sequence number}};}}]
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\draw [red,->] (mark-3) -- (mark-7);
\end{tikzpicture}
```

`/pgf/decoration/mark info/distance from start`

这个键 (key) 保存的是当前标记与被装饰路径的起点之间的路径长度，即沿着被装饰路径，从被装饰路径的起点到当前标记位置的长度，这个长度的单位是 `pt`。



```
\begin{tikzpicture}[decoration={markings,
mark=between positions 0 and 1 step 40pt with
{
\draw [help lines] (0,0) -- (0,0.5)
node[above,font=\tiny]{\pgfkeysvalueof{/pgf/decoration/mark info/distance from
start}};
},
mark=at position -0.1pt with
{
\draw [help lines] (0,0) -- (0,0.5)
node[above,font=\tiny]{\pgfkeysvalueof{/pgf/decoration/mark info/distance from
start}};
}}]
\draw [help lines] grid (5,3);
\draw [postaction={decorate}] (0,0) .. controls (8,3) and (0,3) .. (5,0) ;
```

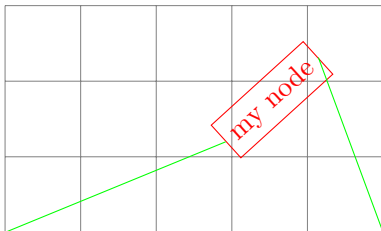
```
\end{tikzpicture}
```

在选项 `mark` 的值中可以使用以下选项。

```
/pgf/decoration/reset marks
```

```
/pgf/decoration/mark connection node=<node name> (无默认值, 初始值 empty)
```

先看一个例子。在下面的例子中, `mark` 选项的 `<code>` 中设置了一个名称为 `my node` 的 node, 这个名称也是选项 `mark connection node` 的值, 这样装饰路径就是“直线段——`my node`——直线段”。



```
\begin{tikzpicture}[decoration={markings,
  mark connection node=my node,
  mark=at position .5 with
  {\node [draw,red,transform shape] (my node) {my node};}}]
\draw [help lines] grid (5,3);
\draw [decorate,green] (0,0) .. controls (8,3) and (0,3) .. (5,0) ;
\end{tikzpicture}
```

但如果装饰选项 `decorate` 换成 `postaction={decorate}`, 就没有直线段, 只有“`my node`”。



```
\begin{tikzpicture}[decoration={markings,
  mark connection node=my node,
  mark=at position .5 with
  {\node [draw,red,transform shape] (my node) {my node};}}]
\draw [help lines] grid (5,3);
\draw [postaction={decorate,green}] (0,0) .. controls (8,3) and (0,3) .. (5,0) ;
\end{tikzpicture}
```

在前两个选项的 `<code>` 中可以使用以下两个箭头命令。

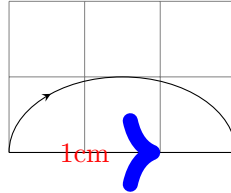
```
\arrow[<options>]{<arrow end tip>}
```

其中 `<arrow end tip>` 是箭头类型的名称, `<options>` 是箭头选项。在 `<code>` 的坐标系中, 箭头的尖点位于坐标系原点, 箭头方向指向右侧, 因此添加箭头后, 箭头的尖点位于指定的标记位置点上, 箭

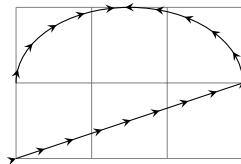
头方向沿着路径的切线方向。

`\arrowreversed[<options>]{<arrow end tip>}`

这里参数 `<arrow end tip>`，`<options>` 与上一命令相同，只是本命令添加一个“反向箭头”。



```
\begin{tikzpicture}[decoration={
  markings,
  mark=at position 1cm with {\node[red]{1cm};},
  mark=at position 2cm with {\arrow[blue,line width=2mm]{>}},
  mark=at position -1cm with {\arrowreversed[black]{stealth}}}
]
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,0) arc (0:180:1.5 and 1);
\end{tikzpicture}
```



```
\begin{tikzpicture}[decoration={
  markings,
  mark=between positions 0 and .75 step 4mm with {\arrow{stealth}},
  mark=between positions .75 and 1 step 4mm with {\arrowreversed{stealth}}}
]
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```

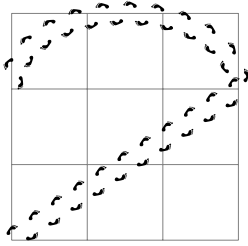
48.5.2 脚印标记

先调用程序库 `\usepgflibrary{decorations.footprints}`，这个程序库提供“脚印”装饰样式——一串沿着被装饰路径放置的脚印，就像沿着路径走过一样。

装饰样式 `footprints`

`footprints`

这个选项指定脚印装饰样式。



```
\begin{tikzpicture}[decoration={footprints,
  foot length=5pt, stride length=10pt}]
  \draw [help lines] grid (3,3);
  \fill [decorate] (0,0) -- (3,2) arc (0:180:1.5 and 1);
\end{tikzpicture}
```

下面是调节脚印装饰样式外观的选项。

`/pgf/decoration/foot length=<dimension>` (初始值 10pt)

这个选项值调节脚印的尺寸。



```
\begin{tikzpicture}[decoration={footprints,
  foot length=30pt}]
  \fill [decorate] (0,0) -- (3,0);
\end{tikzpicture}
```

`/pgf/decoration/stride length=<dimension>` (初始值 30pt)

这个选项值调节“步长”，即前后两个脚印的间距。



```
\begin{tikzpicture}[decoration={footprints,
  stride length=50pt}]
  \fill [decorate] (0,0) -- (3,0);
\end{tikzpicture}
```

`/pgf/decoration/foot sep=<dimension>` (初始值 4pt)

这个选项值调节左右脚印的横向间距。



```
\begin{tikzpicture}[decoration={footprints,
  foot sep=30pt}]
  \fill [decorate] (0,0) -- (3,0);
\end{tikzpicture}
```

`/pgf/decoration/foot angle=<angle>` (初始值 10)

这个选项值调节脚印的角度，如果这个选项的值是 60，就是“严重外八字脚”。



```
\begin{tikzpicture}[decoration={footprints,
  foot angle=60}]
  \fill [decorate] (0,0) -- (3,0);
\end{tikzpicture}
```

`/pgf/decoration/foot of=<name>` (初始值 human)

这个选项的值 `<name>` 用来选择脚印类型,初始值的“人类”的脚印,另外还有“矮人”(gnome),“鸟”(bird),“猫”(felis silvestris) 这 3 种可选。



```
\begin{tikzpicture}[decoration={footprints,
    foot of=felis silvestris}]
\fill [decorate] (0,0) -- (3,0);
\end{tikzpicture}
```

48.5.3 形状装饰

这个程序库可以使用各种已定义的“形状”(shape)来装饰路径,由于历史的原因保留这个程序库,不过使用 `markings` 程序库更好。

先调用程序库 `\usepgflibrary{decorations.shapes}`。

已定义的“形状”(shape)见程序库 `shapes.symbols`, `shapes.geometric`, `shapes.callouts`, `shapes.misc`, `shapes.arrows`, `shapes.multipart`, 你也可以自定义一个形状,见 §101.5。

下面 3 个选项是适用于各个形状装饰的选项。

`/pgf/decoration/shape width=<dimension>` (无默认值,初始值 2.5pt)

这个选项设置形状的宽度。

`/pgf/decoration/shape height=<dimension>` (无默认值,初始值 2.5pt)

这个选项设置形状的高度。

`/pgf/decoration/shape size=<dimension>` (无默认值)

这个选项同时设置形状的宽度和高度。

装饰样式 `crosses`

`crosses`

这个选项指定使用叉号形状 `crosses` 来装饰路径。下面的选项可以调节这个形状的外观。

`segment length=<dimension>`

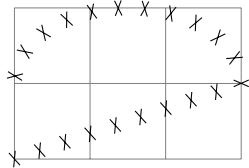
这个选项值确定相邻两个叉号的中心距离。

`shape height=<dimension>`

这个选项值确定叉号的高度。

`shape width=<dimension>`

这个选项值确定叉号的宽度。



```
\begin{tikzpicture}[decoration={crosses,shape height=2mm}]
\draw [help lines] grid (3,2);
\draw [decorate] (0,0) -- (3,1)
    arc (0:180:1.5 and 1);
\end{tikzpicture}
```

装饰样式 triangles

triangles

这个选项指定使用三形状 triangles 来装饰路径。下面的选项可以调节这个形状的外观。

segment length=<dimension>

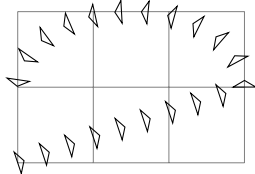
这个选项值确定相邻两个三角的间距。

shape height=<dimension>

这个选项值确定三角的竖直边的高度。

shape width=<dimension>

这个选项值确定三角的宽度。

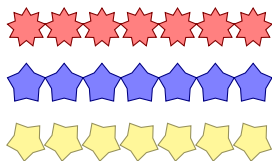


```
\begin{tikzpicture}[decoration={triangles,shape height=3mm}]
  \draw [help lines] grid (3,2);
  \draw [decorate] (0,0) -- (3,1)
    arc (0:180:1.5 and 1);
\end{tikzpicture}
```

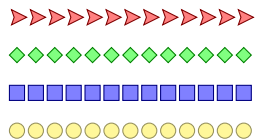
装饰样式 shape backgrounds

shape backgrounds

这个选项开启“形状背景路径”装饰样式。这里的“形状”（shapae）是已经用命令 `\pgfdeclareshape` 定义的形状，“背景路径”指的是形状的边界轮廓，其中包含各种锚位置，但不包含盛放文字的盒子，也不能为背景路径命名。有的特殊形状有自己的特殊选项来调节其外观，例如程序库 `shapes.geometric` 提供的形状 `star` 具有选项 `star points`, `star point height`, 这些选项在“形状背景路径”中也是保留的，可以用来调整形状背景路径的外观。



```
\tikzset{
  paint/.style={draw=#1!50!black, fill=#1!50},
  my star/.style={decorate,decoration={shape backgrounds,shape=star}, star points
    =#1}
}
\begin{tikzpicture}[decoration={shape sep=.5cm, shape size=.5cm}]
  \draw [my star=9, paint=red] (0,1.5) -- (3,1.5);
  \draw [my star=5, paint=blue] (0,.75) -- (3,.75);
  \draw [my star=5, paint=yellow, shape border rotate=30] (0,0) -- (3,0);
\end{tikzpicture}
```



```
\tikzset{paint/.style={ draw=#1!50!black, fill=#1!50 },
  decorate with/.style=
    {decorate,decoration={shape backgrounds,shape=#1,shape size=2mm}}}
\begin{tikzpicture}
  \draw [decorate with=dart, paint=red] (0,1.5) -- (3,1.5);
  \draw [decorate with=diamond, paint=green] (0,1) -- (3,1);
  \draw [decorate with=rectangle, paint=blue] (0,0.5) -- (3,0.5);
  \draw [decorate with=circle, paint=yellow] (0,0) -- (3,0);
\end{tikzpicture}
```

使用这个装饰样式时，下面的选项能影响背景路径的外观。

`/pgf/decoration/anchor=<anchor>` (无默认值, 初始值 `center`)

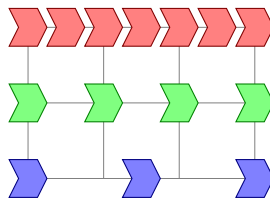
PGF 会根据有关选项自动确定被装饰路径上的点来放置形状背景路径，这些点就是形状背景路径的“指向点”。初始之下，形状背景路径的锚位置 `center` 位于这些指向点上。这个选项会把形状背景路径的锚位置 `<anchor>` 放在这些指向点上。

`/pgf/decoration/shape=<shape name>` (无默认值, 初始值 `circle`)

这个选项确定用哪个形状的背景路径来做装饰。

`/pgf/decoration/shape sep=<spacing>` (无默认值, 初始值 `.25cm, between centers`)

这个选项确定相邻两个形状背景路径的间距,初始之下使用形状中心之间的间距,即 `shape sep={.25cm, between centers}`, 也可以换成形状边界间距, `shape sep={.25cm, between borders}`。



```
\begin{tikzpicture}[
  decoration={shape backgrounds,shape size=.5cm,shape=signal},
  signal from=west, signal to=east,
  paint/.style={decorate, draw=#1!50!black, fill=#1!50}]
\draw [help lines] grid (3,2);
\draw [paint=red, decoration={shape sep=.5cm}]
  (0,2) -- (3,2);
\draw [paint=green, decoration={shape sep={1cm, between centers}}]
  (0,1) -- (3,1);
```

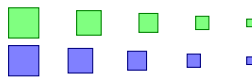


```
\draw [paint=blue, decoration={shape sep={1cm, between borders}}]
  (0,0) -- (3,0);
\end{tikzpicture}
```

`/pgf/decoration/shape evenly spread={<number>, by centers 或 by borders}` (默认 by centers)

这里 <number> 是个正整数。这个选项确定使用 <number> 个形状背景路径来做装饰，并且任意两个相邻形状背景路径在被装饰路径上的间距相等。这个选项有两个可选参数 `by centers` 和 `by borders`，这两个参数确定计算两个相邻形状背景路径的间距的方式，即“中心到中心”和“边界到边界”，默认 `by centers`。

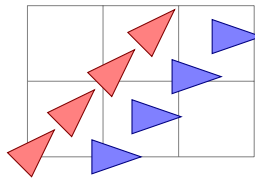
这个选项会抑制选项 `shape sep`。



```
\tikzset{
  paint/.style={draw=#1!50!black, fill=#1!50},
  spreading/.style={
    decorate,decoration={shape backgrounds, shape=rectangle, shape start size=4
      mm,shape end size=1mm,shape evenly spread={#1}}
  }
}
\begin{tikzpicture}
  \fill [paint=green,spreading={5, by borders}, decoration={shape scaled}] (0,2)
    -- (3,2);
  \fill [paint=blue,spreading={5, by centers}, decoration={shape scaled}] (0,1.5)
    -- (3,1.5);
\end{tikzpicture}
```

`/pgf/decoration/shape sloped=<boolean>` (无默认值, 初始值 true)

在默认下，绘制装饰路径的坐标系类似被装饰路径的“自然坐标系”，因此用于装饰的形状背景路径会随着被装饰路径的切线变化而旋转。如果本选项设置为 `shape sloped=false`，则形状背景路径不会出现旋转。



```
\tikzset{
  paint/.style={draw=#1!50!black, fill=#1!50}
}
\begin{tikzpicture}[decoration={
```

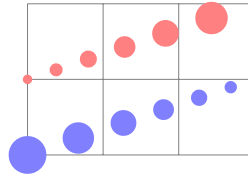
```

    shape width=.65cm, shape height=.45cm,
    shape=isosceles triangle, shape sep=.75cm,
    shape backgrounds}]
\draw [help lines] grid (3,2);
\draw [paint=red,decorate] (0,0) -- (2,2);
\draw [paint=blue,decorate,decoration={shape sloped=false}] (1,0) -- (3,2);
\end{tikzpicture}

```

`/pgf/decoration/shape scaled=<boolean>` (无默认值, 初始值 `false`)

这个选项需要下面的选项配合才会有用。如果被装饰路径上的第一个形状背景路径的尺寸与最后一个形状背景路径的尺寸不同, 那么本选项的值设为 `true` 后, 被装饰路径上形状背景路径的尺寸会逐渐变化。



```

\tikzset{
  bigger/.style={decoration={shape start size=.125cm, shape end size=.5cm}},
  smaller/.style={decoration={shape start size=.5cm, shape end size=.125cm}},
  decoration={shape backgrounds, shape sep={.25cm, between borders},shape
    scaled}
}
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\fill [decorate, bigger, red!50] (0,1) -- (3,2);
\fill [decorate, smaller, blue!50] (0,0) -- (3,1);
\end{tikzpicture}

```

`/pgf/decoration/shape start width=<length>` (无默认值, 初始值 `2.5pt`)

这个选项设置第一个形状背景路径的宽度, 它能够改写选项 `shape width` 的设置。

`/pgf/decoration/shape start height`

这个选项设置第一个形状背景路径的高度, 它能够改写选项 `shape height` 的设置。

`/pgf/decoration/shape start size=<length>` (无默认值)

本选项同时设置 `shape start width` 和 `shape start height`。

`/pgf/decoration/shape end width=<length>` (无默认值, 初始值 `2.5pt`)

这个选项设置最后一个形状背景路径的宽度, 它能够改写选项 `shape width` 的设置。

`/pgf/decoration/shape end height`

这个选项设置最后一个形状背景路径的高度，它能够改写选项 `shape height` 的设置。

`/pgf/decoration/shape end size=<length>` (无默认值)

本选项同时设置 `shape end width` 和 `shape end height`。

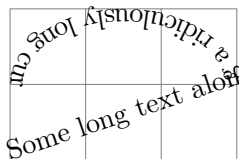
48.6 文字装饰

调用程序库 `\usepgflibrary{decorations.text}`，这个程序库提供文字装饰样式，有两种类型：`text along path` 和 `text effects along path`。

48.6.1 装饰样式 `text along path`

`text along path`

在默认下，沿着被装饰路径的方向，从被装饰路径的起点开始，文字会被放置在路径左侧。下面是个例子。



```
\begin{tikzpicture}[decoration={text along path,
  text={Some long text along a ridiculously long curve that}}]
  \draw [help lines] grid (3,2);
  \draw [decorate] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```

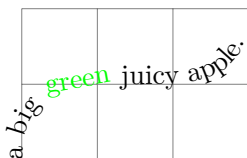
关于这个装饰样式要注意以下几点：

- 每个文字字符都被放入一个单独的 `\hbox` 中。
- 默认把字符的基线中点放在被装饰路径上，可以使用变换选项来改变文字与被装饰路径的相对位置。
- 文字沿着被装饰路径放置，文字或符号之间可能重叠。
- 文字可以是数学模式下的文字，数学模式的上下标要用花括号括起来，例如 `{^y_z}`；各种算符，例如 `\times`，`\cdot` 也要用花括号括起来。但是如果数学式子太复杂会影响装饰效果。
- 在子输入路径的边界位置上可能出现文字位置偏差，此时需要手工纠正。

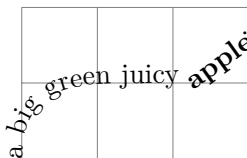
这个装饰样式有以下选项可用。

`/pgf/decoration/text={<text>}` (无默认值，初始值 `empty`)

这里 `<text>` 是需要沿着被装饰路径放置的文字。文字中多余的空格会被忽略，因此需要适当使用命令 `_` 或者 `\space`。也可以使用字体命令，如 `\tt`，`\it`，`\bf` 等设置文字字体，也可以使用颜色命令 `\color` 设置文字颜色。注意，一旦把多个字符放入一个盒子或 `TEX` 分组内，这些字符就不再随着被装饰路径的弯曲而旋转。



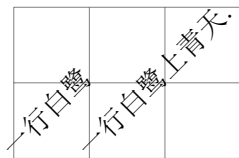
```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\path [decorate,decoration={text along path,
text={a big {\color{green}green} juicy apple.}}]
(0,0) .. controls (0,2) and (3,0) .. (3,2);
\end{tikzpicture}
```



```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\path [decorate,decoration={text along path,
text={a big green juicy {\bf apple}.}}]
(0,0) .. controls (0,2) and (3,0) .. (3,2);
\end{tikzpicture}
```

通常，用于装饰的文字 `<text>` 保存在宏 `\pgfdecorationtext` 中。如果被装饰路径太短而文字太多，文字超出被装饰路径的端点，那么有的文字就不能显示出来，这些不能显示出来的文字保存在宏 `\pgfdecorationrestoftext` 中。

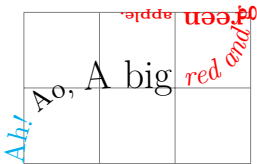
下面例子中，第一个被装饰路径长度是 $\sqrt{2}$ ，装饰文字过多，所以文字没有全部显示。第二个被装饰路径使用了选项 `scale=2` 把路径长度变为原来的 2 倍，于是装饰文字都显示出来了：



```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\path [decorate,decoration={text along path,
text={一行白鹭上青天.}}]
(0,0)--(1,1);
\path [scale=2, decorate,decoration={text along path,
text={一行白鹭上青天.}}]
(0.5,0)--(1.5,1);
\end{tikzpicture}
```

`/pgf/decoration/text format delimiters={<before>}{<after>}` (无默认值，初始值 `{|}{}`)

这个选项设置定界符，用于界定命令作用范围，如果 `<after>` 是空的，那么就把 `<before>` 同时用作开定界符和闭定界符。初始之下把“|”作为开定界符和闭定界符。与定界符配合使用的还有“+”，如下面的例子所示。



```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\path [decorate,
      decoration={text along path, text format delimiters={[]{}},
      text={[\color{cyan}]Ah! [] Ao, [\Large]A big
            [\color{red}\it]red and [+ \bf]green [+ \tiny]apple.}}]
(0,0) .. controls (0,2) and (3,0) .. (3,2)--(0,2);
\end{tikzpicture}
```

上面例子中，将 “[” 和 “]” 分别作为开定界符和闭定界符。定界符的用法是：定界符要成对使用；一对定界符内只能使用各种命令，所以定界符不能套嵌使用；把某些命令放在一对定界符内，这些命令就对之后的文字起作用；使用一对内容为空（即不含任何命令）的定界符将文字还原为默认状态；加号 “+” 的作用是把之前一对定界符内的命令添加到 “+” 所在位置，因此 “+” 要放在定界符内；定界符的符号类别是 11 或 12。

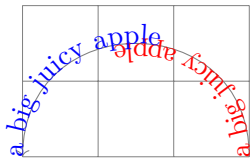
使用定界符命令后，文字的外观会被改变，而且还会随着被装饰路径的弯曲而旋转。

`/pgf/decoration/text color=<color>` （无默认值，初始值 black）

本选项设置文字颜色。

`/pgf/decoration/reverse path=<boolean>` （无默认值，初始值 false）

这个选项使得被装饰路径的方向变成反方向，因此文字会沿着反方向放置在被装饰路径的另一侧。



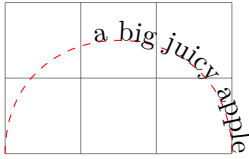
```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw [gray, ->]
  [postaction={decorate, decoration={text along path,
    text={a big juicy apple}, text color=red}}]
  [postaction={decorate, decoration={text along path,
    text={a big juicy apple}, text color=blue, reverse path}}]
(3,0) .. controls (3,2) and (0,2) .. (0,0);
\end{tikzpicture}
```

`/pgf/decoration/text align={<alignment options>}`

这个选项会给 <alignment options> 加上前缀 `/pgf/decoration/text align/` 来执行，因此 <alignment options> 可以是，例如，`left`，`align=left`，`center`，`fit to path=true` 等，具体参考下面的选项。

`/pgf/decoration/text align/align=<alignment>` （无默认值，初始值 left）

<alignment> 可以选择 `left`，`right`，`center` 三个之一。左对齐 `left` 一般指的与路径的起点对齐。



```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw [red, dashed]
  [postaction={decorate, decoration={text along path,
    text={a big juicy apple}, text align={align=right}}}]
  (0,0) .. controls (0,2) and (3,2) .. (3,0);
\end{tikzpicture}
```

`/pgf/decoration/text align/left`

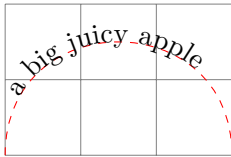
本选项等效于 `/pgf/decoration/text align/align=left`.

`/pgf/decoration/text align/right`

`/pgf/decoration/text align/center`

`/pgf/decoration/text align/left indent=<length>` (无默认值, 初始值 0pt)

本选项规定文字左对齐, 不过一开始有一段 (沿着路径) 长度为 `<length>` 的空白。



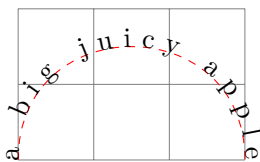
```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw [red, dashed]
  [postaction={decorate, decoration={text along path,
    text={a big juicy apple}, text align={left indent=.8cm}}}]
  (0,0) .. controls (0,2) and (3,2) .. (3,0);
\end{tikzpicture}
```

`/pgf/decoration/text align/right indent=<length>` (无默认值, 初始值 0pt)

本选项规定文字右对齐, 不过一开始有一段 (沿着路径) 长度为 `<length>` 的空白。

`/pgf/decoration/text align/fit to path=<boolean>` (无默认值, 初始值 false)

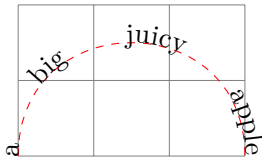
这个选项使得各个文字字符 (包括有效空格) 在被装饰路径上均匀分布。但如果文字太多而被装饰路径太短, 以致于文字超出被装饰路径的端点, 那么本选项无效。



```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw [red, dashed]
  [postaction={decorate, decoration={text along path,
    text={a big juicy apple},
    text align=fit to path}}]
  (0,0) .. controls (0,2) and (3,2) .. (3,0);
\end{tikzpicture}
```

`/pgf/decoration/text align/fit to path stretching spaces=<boolean>` (无默认值, 初始值 `false`)

这个选项的作用类似前一个选项, 只不过单个单词的各个字母之间的间距不会被改变, 改变的是单词之间的空格长度, 注意由 `\space` 生成的空格会被拉长或压缩, 但由 `_` 生成的空格不会被改变。



```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw [red, dashed]
  [postaction={decorate, decoration={text along path,
    text={a big juicy apple},
    text align={fit to path stretching spaces}}}]
  (0,0) .. controls (0,2) and (3,2) .. (3,0);
\end{tikzpicture}
```

下面的例子利用装饰样式 `text along path` 制作一个动画。先建一个 `tex` 文档, 将该文档命名为 `yuantu`, 并编译下面的代码:

```
\documentclass[tikz]{standalone}
\usepackage{ctex}
\usepackage{animate}
\usepackage{graphicx}
\usepackage{tikz}
\usetikzlibrary[calc, math, decorations.text, shapes.geometric]

\begin{document}

\pgfdeclareradialshading{sphere}{\pgfpoint{0.5cm}{-0.5cm}}%
{color(0cm)=(red!20); color(0.2cm)=(red!40); color(0.4cm)=(red!60); color(0.6cm)=(red
!70);
color(0.8cm)=(red!80); color(1cm)=(red); color(2cm)=(red)
}

\foreach \i in {0,-2,...,-24}
{
\begin{tikzpicture}
\path (-2.5,-1.5) rectangle (4,3);
\begin{scope}[x=(\i:1),y=(90+2*\i:1)]
\fill [shading=sphere,](-1,-1)----(2,0)----(0,2)----(-2,0)--cycle;
\draw [line width=0.1pt,draw=gray!50](-1,1)--(0,0)--(1,1);
\end{scope}
\end{tikzpicture}
}
```

```

    \fill [yellow,rotate=\i](0,0) ellipse (2pt and 2pt+\i/50*2pt);
  \end{scope}
\end{tikzpicture}
}

\foreach \j in {270,260,...,90}
{
\begin{tikzpicture}
  \path (-2.5,-1.5) rectangle (4,3);
  \begin{scope}[x=(-24:1),y=(42:1)]
    \fill [shading=sphere](-1,-1)---+(2,0)---+(0,2)---+(-2,0)--cycle;
    \coordinate (o') at ($(0,1)+(\j:1)$);
    \filldraw [fill=red,line width=0.1pt,draw=gray!50](-1,1)--(o')--(1,1)--cycle;
    \fill [yellow,rotate=-24](o') ellipse (2pt and 2pt-24/50*2pt);
  \end{scope}
\end{tikzpicture}
}

\foreach \i in {0,0.2,...,2}
{
\begin{tikzpicture}
  \path (-2.5,-1.5) rectangle (4,3);
  \begin{scope}[x=(-24:1),y=(42:1)]
    \coordinate (o) at (0,1);
    \fill [shading=sphere,](-1,-1)---+(2,0)---+(0,2)---+(-2,0)--cycle;
    \coordinate (o') at ($(0,1)+(90:1)$);
    \filldraw [fill=red,line width=0.1pt,draw=gray!50](-1,1)--(o')--(1,1)--cycle;
    \fill [yellow,rotate=-24](o') ellipse (2pt and 2pt-24/50*2pt);
  \end{scope}
  \begin{scope}[inner sep=0.01pt]
    \node[star,ball color=brown,minimum size=3*\i mm]at($(o)+(20:\i)$){};
    \node[star,ball color=cyan,minimum size=1*\i mm]at($(o)+(0:\i)$){};
    \node[star,ball color=pink,minimum size=2*\i mm]at($(o)+(60:0.8*\i)$){};
    \node[star,ball color=olive,minimum size=3*\i mm]at($(o)+(80:1.2*\i)$){};
    \node[star,ball color=yellow,minimum size=2*\i mm]at($(o)+(120:\i)$){};
    \node[star,ball color=orange,minimum size=2*\i mm]at($(o)+(150:1.2*\i)$){};
  \end{scope}
\end{tikzpicture}
}

```



```

}

\foreach \i in {18,...,0}
{
\begin{tikzpicture}
  \path (-2.5,-1.5) rectangle (4,3);
  \begin{scope}[x=(-24:1),y=(42:1)]
    \coordinate (o) at (0,1);
    \fill [shading=sphere,](-1,-1)---+(2,0)---+(0,2)---+(-2,0)--cycle;
    \coordinate (o') at ($(0,1)+(90:1)$);
    \filldraw [fill=red,line width=0.1pt,draw=gray!50](-1,1)--(o')--(1,1)--cycle;
    \fill [yellow,rotate=-24](o') ellipse (2pt and 2pt-24/50*2pt);
  \end{scope}
  \path [evaluate={\j=\i*12;},decorate,
    decoration={reverse path,text along path,text={|\kaishu\large|有朋自远方来不亦乐乎!
      },
    text align={left indent={\j pt}}}]
    (o)..controls +(20:1) and +(-30:1)..(3,2)--(-1.5,2);
\end{tikzpicture}
}

\end{document}

```

如果顺利，编译上面的代码得到一个 62 页的 PDF 文件，每个页面都是一个图形，每个图形的尺寸都是 tikz 命令 `\path(-2.5,-1.5) rectangle (4,3);` 规定的矩形尺寸。然后（在同一文件目录中）另建一个 tex 文件，编译下面的代码（可能需要编译两次）：

```

\documentclass{standalone}
\usepackage{animate}
\usepackage{graphicx}

\begin{document}
\animategraphics{8}{yuantu}{}{}
\end{document}

```

得到一个 PDF 文件，这是个动画，用鼠标点击这个文件中的图形开始演示，动画如下：

48.6.2 装饰样式 text effects along path


text effects along path

这个文字装饰类型与 `text along path` 类似，不过在这个文字装饰类型中，每个字符都是作为 TikZ 的 node 添加到被装饰路径上的，故每个字符都是一个小“图形”，关于 node 的各种选项，例如 `text`, `scale`, `opacity`, `fill`, `draw`, `shift` 等都是可用的，能产生多种“effects”。

注意区分“字母字符”和“非字母字符”，字母字符构成单词，而非字母字符有其他作用，例如空格可以分隔单词。

与 `text along path` 不同的是，定界符在 `text effects along path` 样式中无效，但是有其它选项能调整装饰文字外观。能用于这个装饰样式，调整装饰文字外观的选项很多，注意有的选项的前缀是 `/tikz/`，这种前缀的选项是 `tikz` 的选项。

本装饰样式的编译过程可能会慢一些。



```

\bfseries\large
\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!}, text align=center,
  text effects/.cd,
  character count=\i, character total=\n,
  characters={evaluate={\c=\i/\n*100;}, text along path,
  text=red!\c!orange},
  character widths={text along path, xslant=0, yscale=1}}]
\path [postaction={decorate}, preaction={decorate,
  text effects={characters/.append={yscale=-1.5,
  opacity=0.5, text=gray, xslant=(\i/\n-0.5)*3}}}]
(0,0) .. controls ++(2,1) and ++(-2,-1) .. (3,4);
\end{tikzpicture}

```

上面例子中的选项 `evaluate={...}` 是数学程序库定义的选项。

下面介绍能用于这个装饰样式的选项。

`/pgf/decoration/text={<text>}`

这个选项设置用于装饰的文字。<text> 中的字符或命令可以用花括号括起来，花括号在编译时才会展开，例如 `gr{"\o}{\ss}eren`。注意 <text> 中不能使用定界符。

```
/pgf/decoration/text align=<align>
```

这个选项确定文字的对齐方式，<align> 可以是 `left`，`right`，`center` 之一。

```
/tikz/text effects={<options>}
```

<options> 是某些能用于本装饰样式的选项，这些选项会被冠以前缀 `/pgf/decoration/ text effects/` 来执行。

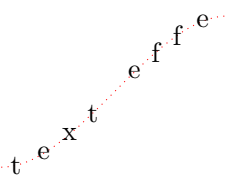
```
/pgf/decoration/text effects/every character (style)
```

这个选项是样式 (style)，用法如 `every character/.style={<options>}`，选项 <options> 会用于每个装饰文字 node，因此选项 <options> 应当都是 TikZ 的关于 node 的选项。这个样式设置的选项会用在各个字符 node 选项的开头，所以本样式之后的选项可以修改本样式的设置。例如以下两个样式

```
every letter/.style={fill=green}, every character/.style={fill=red}
```

样式 `every character` 先把所有字符 node 的填充色设为红色（尽管这个样式在后），样式 `every letter` 又把所有字母字符 node 的填充色由红色改为绿色（尽管这个样式在前）。

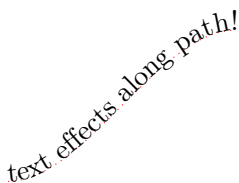
初始之下本样式为空，文字 node 只是沿着路径放置，本身并没有什么“effects”，比如，文字 node 不会随着被装饰路径的弯曲而旋转。



```
\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!}}]
  \path [draw=red, dotted, postaction={decorate}]
    (0,0) .. controls ++(1,0) and ++(-1,0) .. (3,2);
\end{tikzpicture}
```

```
/pgf/decoration/text effects/text along path
```

这个选项会使得文字 node 随着被装饰路径的弯曲而旋转，效果就像装饰样式 `text along path` 那样，但仍然不能使用定界符。



```
\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!}}]
  \path [draw=red, dotted, postaction={decorate},
    text effects={text along path}]
    (0,0) .. controls ++(1,0) and ++(-1,0) .. (3,2);
\end{tikzpicture}
```

```
/pgf/decoration/text effects/characters={<effects>}
```

这个选项是 `every character` 的简洁形式。

```
/pgf/decoration/text effects/character <number> (style)
```

这个选项是个样式，用法如 `character <number>/.``style={<options>}`，其中 `<number>` 是个正整数，代表第 `<number>` 个字符；`<options>` 是针对第 `<number>` 个字符的 `tikz` 选项。

```
/pgf/decoration/text effects/every letter (style)
```

这个选项是个样式，它设置的选项针对所有的“字母字符”。

```
/pgf/decoration/text effects/letter <number> (style)
```

这个选项是个样式，它设置的选项针对任何一个单词的第 `<number>` 个“字母字符”。

```
/pgf/decoration/text effects/every first letter (style)
```

这个选项是个样式，它设置的选项针对任何一个单词的第一个“字母字符”。

```
/pgf/decoration/text effects/every last letter (style)
```

这个选项是个样式，它设置的选项针对任何一个单词的最后一个“字母字符”。

```
/pgf/decoration/text effects/every word (style)
```

这个选项是个样式，它设置的选项针对任何一个单词的任何一个“字母字符”。

```
/pgf/decoration/text effects/word <number> (style)
```

这个选项是个样式，它设置的选项针对第 `<number>` 个单词的“字母字符”。

```
/pgf/decoration/text effects/word <m> letter <n> (style)
```

这个选项是个样式，它设置的选项针对第 `<m>` 个单词的第 `<n>` 个“字母字符”。

```
/pgf/decoration/text effects/every word separator (style)
```

这个选项是个样式，它设置的选项针对单词之间的分隔符。

```
/pgf/decoration/text effects/word separator=<character> (无默认值, 初始值 space)
```

这个选项用于设置单词分隔符，初始之下，单词分隔符是空格。这里用作分隔符的 `<character>` 必须是单个字符，例如 `a` 或 `-`。

```
/pgf/decoration/text effects/every character width (style)
```

这个选项是个样式，它设置所有字符 `node` 的宽度。本样式设置的选项应当是能够影响 `node` 的宽度各种 `tikz` 选项，例如 `inner xsep`, `text width`, `minimum width` 等。

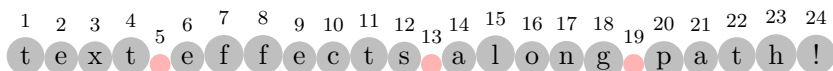
```
/pgf/decoration/text effects/character widths={<effects>}
```

这个选项是样式 `every character width` 的简洁形式。

```
/pgf/decoration/text effects/character count=<macro>
```

这个选项将字符 `node` 的编号保存在宏 `<macro>` 中，然后可以在代码中用宏 `<macro>` 来引用字符 `node`。字符 `node` 的编号从 1 开始。

注意这个选项不能用在前面介绍的各个样式 (`style`) 中。



```

\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!},
  text effects/.cd,
  path from text,
  character count=\i, every word separator/.style={fill=red!30},
  characters={text along path, shape=circle, fill=gray!50}}]
\path [decorate, text effects={characters/.append={label=above:\footnotesize\i}}]
  (0,0);
\end{tikzpicture}

```

上面例子中，共有 24 个字符 node，其中包括 3 个空格（用于分隔单词），一个叹号。选项 `path from text` 的作用见后文。宏 `\i` 保存字符 node 的编号，整个命令相当于使用了 `\foreach` 语句

```

\foreach \i in {1,...,24}
.....

```

`/pgf/decoration/text effects/character total=<macro>`

这个选项将字符 node 的总数保存在宏 `<macro>` 中。

注意这个选项不能用在前面介绍的各个样式（style）中。

text effects along path

```

\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!},
  text effects/.cd,
  character count=\i, character total=\n,
  characters={text along path, evaluate={\c=\i/\n*100;},
  text=orange!\c!blue, scale=\i/\n+0.5}}]
\path [decorate]
  (0,0) .. controls ++(1,0) and ++(-1,0) .. (3,2);
\end{tikzpicture}

```

上面代码中的选项 `evaluate` 是数学程序库中的选项，见 §56.1。

`/pgf/decoration/text effects/letter count=<macro>`

这个选项将字母字符 node 的编号保存在宏 `<macro>` 中。当使用这个选项后，字母字符 node 的编号从 1 开始，用作单词分隔符号的 node 的编号则统一编为 0。

注意这个选项不能用在前面介绍的各个样式（style）中。

```

  1 2 3 4 0 1 2 3 4 5 6 7 0 1 2 3 4 5 0 1 2 3 4 5
  t e x t  e f f e c t s  a l o n g  p a t h !

```

```

\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!},
  text effects/.cd,

```

```

    path from text, letter count=\i, every word separator/.style={fill=red!30},
    characters={text along path, shape=circle, fill=gray!50}}]
\path [decorate, text effects={characters/.append={label=above:\footnotesize\i}}]
    (0,0);
\end{tikzpicture}

```

`/pgf/decoration/text/effctcs/letter total=<macro>`

这个选项将字母字符 node 的总个数保存在宏 `<macro>` 中。当使用这个选项后，用作单词分隔符号的 node 的编号统一编为 0。

注意这个选项不能用在前面介绍的各个样式 (style) 中。

`/pgf/decoration/text effects/word count=<macro>`

这个选项将单词的编号保存在宏 `<macro>` 中，编号从 1 开始。例如，假设第一个单词是 `text`，这个单词有 4 个字母符号，那么这 4 个字母 node 的编号都是 1，换句话说，单词 `text` 由 4 个编号都是 1 的字母符号 node 组成。当使用这个选项后，用作单词分隔符号的 node 的编号与它前面的单词的编号相同。如果整个文字以单词分隔符号开头，那么这个单词分隔符 node 的编号是 0。

注意这个选项不能用在前面介绍的各个样式 (style) 中。

```

\begin{tikzpicture}[decoration={text effects along path,
    text={text effects along path!},
    text effects/.cd,
    path from text, word count=\i, every word separator/.style={fill=red!30},
    characters={text along path, shape=circle, fill=gray!50}}]
\path [decorate, text effects={characters/.append={label=above:\footnotesize\i}}]
    (0,0);
\end{tikzpicture}

```

`/pgf/decoration/text effects/word total=<macro>`

这个选项将单词的总数保存在宏 `<macro>` 中。

注意这个选项不能用在前面介绍的各个样式 (style) 中。

`/pgf/decoration/text effects/style characters={<characters>} with {<effects>}`

在用文字做装饰时，装饰的文字中可能含有某个 (某些) 符号，例如字符 `t` 以及符号 `-`，当然装饰的文字中可能含有数个 `t` 或数个 `-`，可以用本选项对这些字符 `t`，`-` 做某种特别设置。

这里 `<characters>` 是需要设置的某个或某些符号，这些符号依次列出，之间不需要 (用逗号或空格) 分隔。`<effects>` 是所期望的外观设置 (即 `tikz` 选项设置)。

Falsches Üben von Xylophonmusik quält jeden größeren Zwerg

```
\begin{tikzpicture}[decoration={text effects along path,
  text={Falsches {"U}ben von Xylophonmusik qu{"a}lt jeden gr{"o}{\ss}eren Zwerg
  },
  text effects/.cd,
  path from text,
  style characters=aeiou{"U}{\a}{\o} with {text=red},
  characters={text along path}}]
\path [decorate] (0,0);
\end{tikzpicture}
```

`/pgf/decoration/text effects/path from text=<true or false>` (默认值 true)

当被装饰路径只含有一个点时 P ，将本选项的值设为 true，那么 PGF 会计算装饰文字的总宽度 d ，并且假设一个水平向右的直线段 $|PQ| = d$ ，把 PQ 当作被装饰路径，因此装饰文字就沿着水平向右的方向显示。

text effects along path!

```
\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!},
  text effects/.cd,
  path from text,
  character count=\i, character total=\n,
  characters={text along path, scale=\i/\n+0.5}}]
\path [decorate] (0,0);
\end{tikzpicture}
```

`/pgf/decoration/text effects/path from text angle=<angle>`

这个选项与前一个选项 `path from text` 配合使用，本选项会使得假想的被装饰直线段 PQ 围绕起点 P 旋转 $\langle \text{angle} \rangle$ 角度。

text effects along path!

```
\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!},
  text effects/.cd,
  path from text, path from text angle=60,
  character count=\i, character total=\n,
  characters={text along path, scale=\i/\n+0.5}}]
\path [decorate] (0,0);
\end{tikzpicture}
```

`/pgf/decoration/text effects/fit text to path=<true or false>` (默认值 true)

这个选项会使得装饰字符 node 在被装饰路径上均匀分布。

text effects along path!
text effects along path!

```
\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!},
  text effects/every character/.style={text along path}}]
\path [draw=gray, postaction={decorate}, rotate=90]
(0,0) .. controls ++(2,0) and ++(-1,0) .. (5,-1);
\path [draw=gray, postaction={decorate}, rotate=90,
  yshift=-1cm, text effects={fit text to path}]
(0,0) .. controls ++(2,0) and ++(-1,0) .. (5,-1);
\end{tikzpicture}
```

`/pgf/decoration/text effects/scale text to path=<true or false>` (默认值 true)

这个选项会根据被装饰路径的长度对装饰字符 node 做放缩，使得整个被装饰路径从头到尾全被装饰起来。

text effects along path!
text effects along path!
text effects along path!

```
\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!},
  text effects/every character/.style={text along path}}]
\path [draw=gray, postaction={decorate}, rotate=90,
  yshift=0.8cm, text effects={scale text to path}]
(0,0) .. controls ++(1,0) and ++(-0.5,0) .. (2.5,-0.5);
\path [draw=gray, postaction={decorate}, rotate=90]
(0,0) .. controls ++(2,0) and ++(-1,0) .. (5,-1);
\path [draw=gray, postaction={decorate}, rotate=90,
  yshift=-0.8cm, text effects={scale text to path}]
(0,0) .. controls ++(2,0) and ++(-1,0) .. (5,-1);
\end{tikzpicture}
```

`/pgf/decoration/text effects/reverse text`

这个选项使得各个字符 node 按照倒序排布，如果原来的装饰文字是“从左向右”阅读的，那么使用本选项后，装饰文字是“从右向左”阅读的，不过 PGF 先将各个字符 node 倒序排布后再对它们做处理，因此各个字符 node 的编号仍然是“从左向右”的，选项的作用也是“从左向右”的。

!htap gnola stceffe txet
!htap gnola stceffe txet

```
\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!},
  text effects/.cd,
  path from text, path from text angle=60,
  character count=\i, character total=\n,
  characters={text along path, scale=\i/\n+0.5}}]
\path [decorate, text effects={reverse text}] (0,0);
\path [red, decorate, decoration={reverse path},
  text effects={characters/.append={scale=-1}}] (1,0);
\end{tikzpicture}
```

/pgf/decoration/text effects/group letters

这个选项把连续的数个字母 node（即一个单词）看作一个“分组”，并把一个“分组”作为一个字符 node 来处理。如果要同时使用选项 `reverse text` 和 `group letters`，那么一定要注意二者的次序。

!htap gnola stceffe txet
!htap gnola stceffe txet

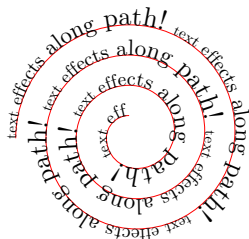
```
\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!},
  text effects/.cd,
  path from text, path from text angle=90,
  every word separator/.style={fill=none},
  character count=\i, character total=\n,
  characters={text along path, fill=gray!50, scale=\i/\n+0.5}}]
\path [decorate, text effects={reverse text, group letters}] (0,0);
\path [decorate, text effects={group letters, reverse text,
  characters/.append={fill=red!20}}] (1,0);
\end{tikzpicture}
```

/pgf/decoration/text effects/repeat text

/pgf/decoration/text effects/repeat text=<times>

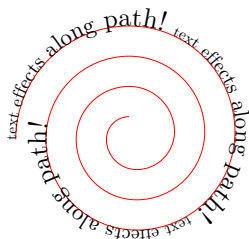
如果被装饰路径过长，而装饰文字过短，那么装饰文字就只能装饰一部分路径，此时可使用这个选项，让装饰文字沿着被装饰路径重复。如果使用选项 `repeat text`，那么装饰文字会不断重复直到把被装饰路径全部装饰完毕。如果使用选项 `repeat text=<times>`，那么在首次添加完毕装饰文字后，再重复添加 `<times>` 次，故装饰文字总共被添加 `<times>+1` 次。

下面的例子设置选项 `repeat text`：



```
\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!\ },
  text effects/.cd,
  repeat text,
  character count=\m, character total=\n,
  characters={text along path, scale=0.5+\m/\n/2}}]
\path [draw=red, ultra thin, postaction=decorate]
(180:2) \foreach \a in {0,...,12}{ arc (180-\a*90:90-\a*90:1.5-\a/10) };
\end{tikzpicture}
```

将上面的例子修改为选项 `repeat text=<times>`:



```
\begin{tikzpicture}[decoration={text effects along path,
  text={text effects along path!\ },
  text effects/.cd,
  repeat text=2,
  character count=\m, character total=\n,
  characters={text along path, scale=0.5+\m/\n/2}}]
\path [draw=red, ultra thin, postaction=decorate]
(180:2) \foreach \a in {0,...,12}{ arc (180-\a*90:90-\a*90:1.5-\a/10) };
\end{tikzpicture}
```

`/pgf/decoration/text effects/character command=<macro>`

本选项所针对的对象可能是一个字符，或者数个字符，或者针对各个单词，等等。这里 `<macro>` 是个 \TeX 宏，这个宏至多可以带有一个参数。假如本选项针对各个单词，那么在输出任何一个单词时，都会把该单词作为宏 `<macro>` 的操作对象，如下面的例子所示。

text₁ effects₂ along₃ path!₄

```
\def\mycommand#1{#1$_\n$}
\begin{tikzpicture}[decoration={text effects along path,
text effects along path!},
text effects/.cd,
path from text, path from text angle=60, group letters,
word count=\n,
every word/.style={character command=\mycommand},
characters={text along path}}]
\path [decorate] (0,0);
\end{tikzpicture}
```

`/pgf/decoration/text effects/replace characters=<characters> with {<code>}`

这里 <characters> 是个字符串，其中相邻两个字符之间不必用空格或其它符号分隔。本选项的 <characters> 声明了某些字符，一旦装饰文字中出现了这些被声明的字符就会产生某种动作，即本选项会用 <code> 代替装饰文字中被声明的字符。<code> 可以是绘图命令、字符、数学公式。

text effects long path!

```
\begin{tikzpicture}[decoration={text effects along path,
text effects along path!},
text effects/.cd,
path from text, path from text angle=10,
replace characters=e with {\fill [red!20] (0,1mm) circle [radius=1mm];},
replace characters=a with {\fill [black!20] (0,1mm) circle [radius=1mm];},
character count=\i, character total=\n,
characters={text along path, scale=\i/\n+0.5}}]
\path [decorate] (0,0);
\end{tikzpicture}
```

48.7 分形装饰

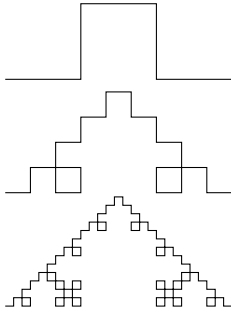
调用程序库 `\usepgflibrary{decorations.fractals}`，这个程序库提供几种分形装饰样式，这些装饰样式主要用于被装饰路径是直线形的情况，而且要“套嵌装饰”才能显示出效果。

装饰样式 Koch curve type 1

Koch curve type 1

这个装饰样式是“第 1 种 Koch 曲线”，这种曲线的基本迭代方式是：将任何直线段“——”替换为折线段“┌┐”，其 Hausdorff 维数是 $\frac{\log 5}{\log 3}$ 。

下面的例子中使用了“套嵌装饰”操作。

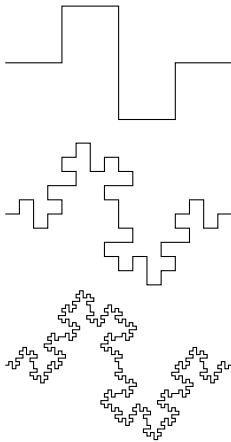


```
\begin{tikzpicture}[decoration=Koch curve type 1]
  \draw decorate{ (0,0) -- (3,0) };
  \draw decorate{ decorate{ (0,-1.5) -- (3,-1.5) } };
  \draw decorate{ decorate{ decorate{ (0,-3) -- (3,-3) } } };
\end{tikzpicture}
```

装饰样式 Koch curve type 2

Koch curve type 2

这个装饰样式是“第 2 种 Koch 曲线”，这种曲线的基本迭代方式是：将任何直线段“——”替换为折线段“┌┐”，其 Hausdorff 维数是 $\frac{3}{2}$ 。

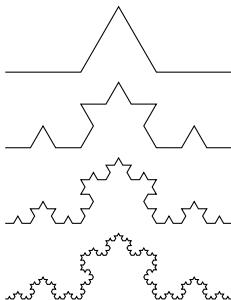


```
\begin{tikzpicture}[decoration=Koch curve type 2]
  \draw decorate{ (0,0) -- (3,0) };
  \draw decorate{ decorate{ (0,-2) -- (3,-2) } };
  \draw decorate{ decorate{ decorate{ (0,-4) -- (3,-4) } } };
\end{tikzpicture}
```

装饰样式 Koch snowflake

Koch snowflake

这个装饰样式的基本迭代方式是：将任何直线段“——”替换为折线段“_^_”，其 Hausdorff 维数是 $\frac{\log 4}{\log 3}$ 。

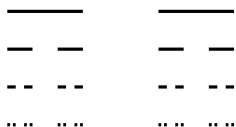


```
\begin{tikzpicture}[decoration=Koch snowflake]
  \draw decorate{ (0,0) -- (3,0) };
  \draw decorate{ decorate{ (0,-1) -- (3,-1) } };
  \draw decorate{ decorate{ decorate{ (0,-2) -- (3,-2) } } };
  \draw decorate{ decorate{ decorate{ decorate{
    (0,-3) -- (3,-3) } } } };
\end{tikzpicture}
```

装饰样式 Cantor set

Cantor set

这个装饰样式不必介绍，其 Hausdorff 维数是 $\frac{\log 2}{\log 3}$.



```
\begin{tikzpicture}[decoration=Cantor set,very thick]
  \draw decorate{ (0,0) -- (3,0) };
  \draw decorate{ decorate{ (0,-.5) -- (3,-.5) } };
  \draw decorate{ decorate{ decorate{ (0,-1) -- (3,-1) } } };
  \draw decorate{ decorate{ decorate{ decorate{
    (0,-1.5) -- (3,-1.5) } } } };
\end{tikzpicture}
```

25 变换

25.1 各种坐标系统

绘图时，提供一个坐标，如 (1,2)，然后指定对该坐标的变换，直到最后在屏幕上显示结果，是一个很长的处理过程。其中可能涉及以下操作：

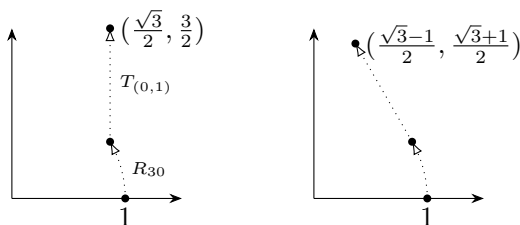
1. PGF 将 (1,2) 解释为它的 xy 坐标系中的位置 P1.
2. PGF 对 P1 应用一个变换，得到位置 P2.
3. 驱动程序（如 dvips 或 pdftex）将 P2 变换为 T_EX 的页面坐标系中的位置 P3.
4. PDF（或 PostScript）对 P3 应用画布变换调整它对应的页面上的位置，得到位置 P4.
5. 预览器将 P4 变换为显示器上的像素位置 P5.

在 TikZ 中只涉及 xy 坐标系以及坐标变换矩阵。PGF 也支持画布变换，但是一旦作出画布变换，PGF 就不再正确地计算坐标位置，例如与 node 有关的形状、锚位置，各种与 bounding box 相关的位置，在将来这一点可能会改进。

下面介绍各种变换选项，这些选项可以用作环境选项，也可以做命令选项。如果这些选项用作环境选项，则这些选项会被加到环境中每个绘图命令的方括号选项序列中，并且排在方括号内的其它（命令本身带有的）变换选项之前。各个变换选项按顺序依次起作用，变换效果是累计的。

从几何的角度看，TikZ 中使用的 xy 坐标系和 xyz 坐标系实际上是“仿射坐标系”，由坐标向量 v_x , v_y , v_z 构成“仿射标架”，(1,2,3) 意味着 $v_x + 2v_y + 3v_z$ 。在初始之下，坐标系是笛卡尔坐标系。注意带有长度单位的坐标 (1,2pt) 不属于 xy 坐标系或 xyz 坐标系，它属于 canvas 坐标系 (§13.2)。

本节所讲的变换对应数学上的仿射变换以及平移，但与数学中的相应变换差别很大。例如，在二维实空间中，设变换 $R_{30}(x)$ 表示将点 x 绕原点旋转 30° ，变换 $T_{(0,1)}(x)$ 表示将点 y 沿着方向 (0,1) 平移一个单位长度，那么这两个变换的复合效果，例如 $T_{(0,1)}(R_{30}((1,0)))$ ，在数学中的理解就是下面左侧图中的点 $(\frac{\sqrt{3}}{2}, \frac{3}{2})$



但是命令 `\fill [rotate=30,shift={(0,1)}](1,0) circle (1pt)`; 先将坐标点和标架一起旋转 30° , 再相对于新标架来平移点, 得到的是上面右侧图中的点 $(\frac{\sqrt{3}-1}{2}, \frac{\sqrt{3}+1}{2})$, 可见与数学中的理解差别很大。

对于一个绘图命令而言, 它带有的变换选项通常分为两种情况。下面分别解释这两种情况, 所做解释仅是为了理解变换效果, 并不代表实际的程序处理过程。

情况一, 绘图命令的变换选项都在一个方括号里, 例如

```
\draw [x={(-60:1)},y={{-sqrt(3)},-2}}] ... ;
```

情况二: 绘图命令的变换选项分属数个方括号, 例如

```
\draw [shift={(1,2)}] [x={(-60:1)},y={{-sqrt(3)},-2}}] [shift={(-1,-2)}] ... ;
```

先看第一种情况。

假设有两类点集, 一类点集构成“不变画布”, 记为 \mathcal{C} , 它的位置和形态都不变; 另一类点集构成“可变画布”, 记为 \mathcal{S} , \mathcal{S} 是叠放在 \mathcal{C} 上的, 每个命令都开启一个“可变画布”并在其上绘图。可以用“皮影戏”来做比方, 皮影戏的白色屏幕相当于“不变画布”, 皮影戏的角色影像相当于“可变画布”, 角色影像在屏幕上会有平移、旋转、镜像翻转、扭曲、拉伸、放缩等变换形态。这里区别不变画布与可变画布只是为了便于理解变换的作用。

不变画布与可变画布都有自己的标架系, 从而可以把点与坐标对应起来, 而且点集(画布)与自己的标架系是固连在一起的。可变画布就像是有无限弹性和无限延展性的特殊画布, 再打个比方, 把 \mathcal{C} 看作一个房间, 把 \mathcal{S} 看作一个气球, \mathcal{S} 的标架系看作是画在气球上的网格, 气球充气或放气会改变气球的形态, 而气球形态的改变与它上面的网格形态的改变是等价的, 所以可以用网格的变化来定义气球的变化。而气球上网格的变化又是以房间(\mathcal{C})为参考系来衡量的。如果气球形态的变化是具有某种“均匀”性的, 那么在气球的网格坐标系下, 无论气球形态怎么变化, 气球上任意一个点 D 的坐标都是不变的; 若以房间为参考系, 则点 D 的坐标是变化的。

假设某个绘图命令中的变换(包括继承自环境选项的, 命令之后的方括号里的)依次是 T_1, T_2, T_3, \dots , 它们的作用如下。

首先, 命令开启一个可变画布 \mathcal{S}_0 , 初始之下 \mathcal{S}_0 与 \mathcal{C} 重合, 二者完全一样。 \mathcal{S}_0 有自己的标架系 $V_0 = \{v_1^0, v_2^0, v_3^0\}$, V_0 就是 \mathcal{C} 中的三个点的坐标。设点 $\mathbf{Q}_0 \in \mathcal{S}_0$, \mathbf{Q}_0 可以看作图形(路径)中的任意点。 \mathbf{Q}_0 在 V_0 中的坐标记为 P_0 , 由于 \mathcal{S}_0 与 \mathcal{C} 重合, 故 \mathbf{Q}_0 在 \mathcal{C} 中的坐标当然也是 P_0 。所有变换的总效果是将标架系 V_0 改变为新的标架系 V_n , 然后在 V_n 中画出坐标为 P_0 的点 \mathbf{Q}_n , 记点 \mathbf{Q}_n 在不变画布 \mathcal{C} 中的坐标为 $\mathbf{Q}_n = P_n \in \mathcal{C}$, 那么变换的效果是 $\mathbf{Q}_0 \rightarrow \mathbf{Q}_n, P_0 \rightarrow P_n$ 。

从数学的角度看, 可变画布与其标架系是等价的, 为了方便叙述, 约定可以不区分它们。

注意在变换过程中, 涉及的标架系都是“定点标架系”, 也就是说, 如果平移一个标架系, 那么其原点就处于平移后的位置。坐标向量相同但原点位置不同的两个标架系, 认为是不同标架系。

把变换 T_1 写成 $T_1[V](x) = y$ 这种形式, 其中 V 是变换 T_1 所相对的参考系, x 是变换 T_1 的操作对象, y 是变换结果, 那么变换 T_1 的作用是

$$T_1V_0 = V_1,$$

其中 V_1 是 T_1 变换 V_0 的结果, 变换的参考系是 V_0 本身(即 \mathcal{C})。经 T_1 变换后, 可变画布由 V_0 变成 V_1 , 或者说 V_1 成为当前可变画布。

类似地, 变换 T_2 的作用是

$$T_2V_1 = V_2,$$

T_2 变换 V_1 为 V_2 , 注意参考系是当前可变画布 V_1 , 而不是 \mathcal{C} . 经 T_2 变换后, 当前可变画布变成 V_2 .

一般地, 变换 T_n 的作用是

$$T_nV_{n-1} = V_n,$$

也就是说, 这些变换按次序变换标架, 最后得到标架系 V_n , 其中与数学习惯的最大不同在于, 每次变换都以当前可变画布为参考系, 并且会改变当前可变画布. 当方括号里的变换选项都处理完毕后, 在 V_n 中画出坐标为 P_0 的点 Q_n , Q_n 在不变画布 \mathcal{C} 中的坐标为 $P_n \in \mathcal{C}$, 那么 Q_n 的集合就是变换后的图形, 然后把可变画布返回到初始状态 \mathcal{C} .

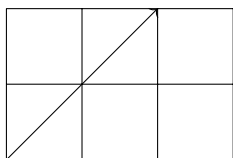
再看第二个情况.

把这些方括号按照从右向左的次序 (倒序), 记为第 1 个、第 2 个……单个方括号的作用属于情况一, 第 1 个方括号得到图形 G_1 , 并把可变画布返回为初始状态 \mathcal{C} , 然后第 2 个方括号对 G_1 作用得到 G_2 , 并把可变画布返回为初始状态 \mathcal{C} ……当这些方括号都处理完毕后, 得到最终的变换图形 G' , 可变画布返回为初始状态 \mathcal{C} . 每个方括号最后都会把可变画布返回为初始状态 \mathcal{C} , 这就比较接近数学上的习惯了.

25.2 xy 坐标系和 xyz 坐标系

`/tikz/x=<value>` (无默认值, 初始值 1cm)

如果 `<value>` 是尺寸 (带单位), 这个选项设置 x 轴的单位向量为 `(<value>, 0pt)`. 注意, 如果 `<value>` 不带长度单位, 例如 `x=1`, 则 `(1, 0pt)` 就是 `(1pt, 0pt)`. 这个选项不仅规定 x 轴单位向量的方向是水平向右的, 也规定这个向量的绝对长度, 因此它可以在水平方向造成放缩效果.



```
\tikz {
\draw[->] (0,0) -- (2,2);
\draw[x=1.5cm] (0,0) grid (2,2);}
```

上面例子中, 第 2 个命令设置 x 轴单位向量的实际长度为 1.5cm, 故点 (2,2) 的横标 2 代表的实际长度是 3cm, 在默认下, 算子 `grid` 的步长选项 `step=1cm`, 所以画出的网格中有 3 条竖线. 也就是说, 由于选项 `step=1cm` 所规定的尺寸是带单位的绝对长度, 故不接受 `x=1.5cm` 的影响.

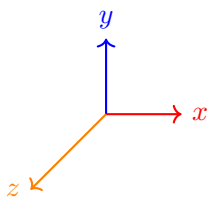
如果 `<value>` 是坐标, 则要用花括号括起来, 如 `x={(a,b)}`, 则 `{(a,b)}` 就是 x 轴的单位向量.

`/tikz/y=<value>` (无默认值, 初始值 1cm)

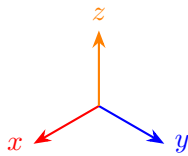
类似 `x=<value>`. 如果 `<value>` 是单个值, 这个选项设置 y 轴的单位向量为 `(0pt, <value>)`.

`/tikz/z=<value>` (无默认值, 初始值 -3.85mm)

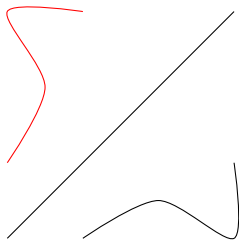
类似 `x=<value>`. 如果 `<value>` 是单个值, 这个选项设置 z 轴的单位向量为 `(<value>, <value>)`.



```
\begin{tikzpicture}[z=-1cm,->,thick]
\draw[color=red] (0,0,0) -- (1,0,0) node[right]{$x$};
\draw[color=blue] (0,0,0) -- (0,1,0) node[above]{$y$};
\draw[color=orange] (0,0,0) -- (0,0,1) node[left]{$z$};
\end{tikzpicture}
```



```
\tikzmath {\a=0.5*\sqrt{3);}
\begin{tikzpicture}[x={(-\a cm,-0.5cm)}, y={(\a cm,-0.5cm)},
z={(0cm,1cm)}, -Stealth,thick]
\draw[color=red] (0,0,0) -- (1,0,0) node[left]{$x$};
\draw[color=blue] (0,0,0) -- (0,1,0) node[right]{$y$};
\draw[color=orange] (0,0,0) -- (0,0,1) node[above]{$z$};
\end{tikzpicture}
```

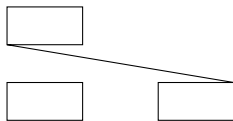


```
\begin{tikzpicture}[smooth]
\draw plot coordinates{(1,0) (2,0.5) (3,0) (3,1)};
\draw[x={(0cm,1cm)},y={(1cm,0cm)},color=red]
plot coordinates{(1,0) (2,0.5) (3,0) (3,1)};
\draw (0,0) -- (3,3);
\end{tikzpicture}
```

25.3 坐标变换

如 $(1,0)$, $(1\text{cm},1\text{pt})$, $(30:2\text{cm})$ 这样的坐标对应坐标系中的位置，位置是用横向和纵向的有单位的长度确定的。PGF 和 TikZ 可以对坐标应用坐标变换矩阵。坐标变换矩阵只对坐标有效，对线宽、虚线样式、颜色渐变的方向等项目没有影响。一般而言，坐标变换矩阵不能用于不涉及坐标的项目，即使该项目间接地涉及坐标。

坐标变换的有效范围受到 $\text{T}_{\text{E}}\text{X}$ 分组的限制。环境是个分组，一个绘图命令也是个分组。当在一个路径之内用方括号列出变换选项时，该变换只对其后的子路径有效，对其前的子路径无效。



```
\tikz \draw
(0,0) rectangle (1,0.5)
{[xshift=2cm] (0,0) rectangle (1,0.5)}
--(0,1) rectangle (1,1.5) ;
```

注意坐标变换由 $\text{T}_{\text{E}}\text{X}$ 完成，限于 $\text{T}_{\text{E}}\text{X}$ 的计算能力，当计算过程涉及变换矩阵的逆矩阵时，不良条件数的矩阵或者奇异矩阵会导致意外的结果。

25.3.1 变换选项

`/tikz/shift={<coordinate>}`

平移变换，平移向量为 $\langle \text{coordinate} \rangle$ 。

`/tikz/shift only`

如果一个变换过程可以看作复合变换，则此选项只保留其中的平移成分。

`/tikz/xshift=<dimension>`

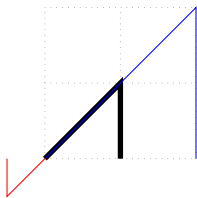
沿着 x 轴方向平移, $\langle \text{dimension} \rangle$ 的值可以是负的。注意 $\langle \text{dimension} \rangle$ 应当带长度单位, 否则默认其单位为 pt.

`/tikz/yshift= $\langle \text{dimension} \rangle$`

沿着 y 轴方向平移, $\langle \text{dimension} \rangle$ 的值可以是负的。注意 $\langle \text{dimension} \rangle$ 应当带长度单位, 否则默认其单位为 pt.

`/tikz/scale= $\langle \text{factor} \rangle$`

以原点为中心的位似变换, 即放缩, $\langle \text{factor} \rangle$ 是放缩比例。如果 $\langle \text{factor} \rangle$ 是负值, 就先将图形以原点为中心做中心对称, 再以原点为中心放缩。



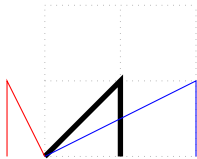
```
\begin{tikzpicture}
\draw[help lines,dotted] (0,0) grid (2,2);
\draw [line width=2pt](0,0) -- (1,1) -- (1,0);
\draw[scale=2,blue] (0,0) -- (1,1) -- (1,0);
\draw[scale=-0.5,red] (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}
```

`/tikz/scale around= $\langle \text{factor} \rangle$: $\langle \text{coordinate} \rangle$`

与 scale 类似, 只是变换的中心改为 $\langle \text{coordinate} \rangle$.

`/tikz/xscale= $\langle \text{factor} \rangle$`

以原点为中心, 在 x 轴方向做放缩, $\langle \text{factor} \rangle$ 为放缩比例。如果 $\langle \text{factor} \rangle$ 是负值, 则先以 y 轴为对称轴将图形做反射变换, 再以原点为中心做放缩。



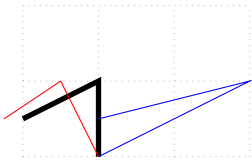
```
\begin{tikzpicture}
\draw[help lines,dotted] (0,0) grid (2,2);
\draw [line width=2pt](0,0) -- (1,1) -- (1,0);
\draw[xscale=2,blue] (0,0) -- (1,1) -- (1,0);
\draw[xscale=-0.5,red] (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}
```

`/tikz/yscale= $\langle \text{factor} \rangle$`

类似 xscale.

`/tikz/xslant= $\langle \text{factor} \rangle$`

这个变换是 $(x, y) \rightarrow (x + y \cdot \langle \text{factor} \rangle, y)$.



```
\begin{tikzpicture}
\draw [help lines,dotted] (0,0) grid (3,2);
\draw [line width=2pt](0,0.5) -- (1,1) -- (1,0);
\draw [xslant=2,blue] (0,0.5) -- (1,1) -- (1,0);
\draw [xslant=-0.5,red] (0,0.5) -- (1,1) -- (1,0);
\end{tikzpicture}
```

`/tikz/yslant=<factor>`

这个变换是 $(x, y) \rightarrow (x, y + x \cdot \langle \text{factor} \rangle)$.

`/tikz/rotate=<degree>`

以原点为中心做旋转。

`/tikz/rotate around={<degree>:<coordinate>}`

以 `<coordinate>` 为中心做旋转。

`/tikz/rotate around x=<angle>`

围绕 x 轴做旋转，以右手握 x 轴，拇指指向 x 轴的正向，手指螺旋方向为旋转的正向。

`/tikz/rotate around y=<angle>`

围绕 y 轴做旋转，以右手螺旋为正。

`/tikz/rotate around z=<angle>`

围绕 z 轴做旋转，以右手螺旋为正。

`/tikz/cm={<a>,,<c>,<d>,<coordinate>}`

这个变换是一般的变换形式，假设 `<coordinate>` 是向量 $\begin{pmatrix} t_x \\ t_y \end{pmatrix}$ ，针对点 $\begin{pmatrix} x \\ y \end{pmatrix}$ 做变换，则变换结果是

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}.$$

注意这个算式是数学学习中的结果，不需要用可变画布来理解。

`/tikz/reset cm`

将变换矩阵设为单位矩阵。取消当前分组内的所有变换，继承自外层环境的变换也被取消。

25.3.2 注意的问题

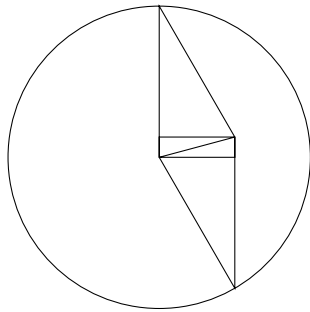
关于坐标变换要注意以下问题。

- 变换对预定义的 `circle` 路径无效，对预定义的 `rectangle` 路径只有横向（x 方向）或纵向（y 方向）的作用。
- 前面提到，变换所针对的（以及相对的）坐标系都是可变画布，这一点并不符合数学上的习惯，有的

情况下可能需要一定的手工计算来确定变换数据。举例说, 如果只是使用 $x={(-60:1)}$ 选项, 那么可变画布就是仿射标架

$$\left\{ \left(\frac{1}{2}, \frac{0}{1} \right), \left(\frac{0}{1}, \frac{0}{1} \right) \right\} = \left\{ \left(\cos(-60^\circ), \frac{0}{1} \right), \left(\frac{0}{1}, \frac{0}{1} \right) \right\},$$

在这个标架下画出点线正方形图, circle 路径, rectangle 路径如下



```
\begin{tikzpicture}[scale=2]
\draw [x={(-60:1)}]
(0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle
(0,0) circle (1cm)
(0,0) rectangle (1,1)
(0,0) -- (1,1);
\end{tikzpicture}
```

现在要把标架转为如下左手单位直角标架

$$\left\{ \left(\frac{1}{2}, -\frac{\sqrt{3}}{2} \right), \left(-\frac{\sqrt{3}}{2}, -\frac{1}{2} \right) \right\} = \left\{ \left(\cos(-60^\circ), \sin(-60^\circ) \right), \left(\sin(-60^\circ), -\cos(-60^\circ) \right) \right\},$$

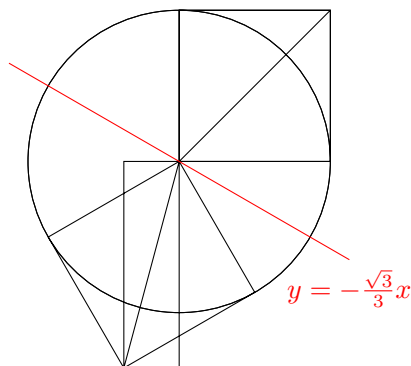
就需要先在标架

$$\left\{ \left(\frac{1}{2}, \frac{0}{1} \right), \left(\frac{0}{1}, \frac{0}{1} \right) \right\}$$

中计算向量 $(-\frac{\sqrt{3}}{2}, -\frac{1}{2})$ 的坐标, 它的坐标是 $(-\sqrt{3}, -2)$, 将这个坐标设为 $y=$ 的值, 如下

```
[x={(-60:1)},y={{-sqrt(3)},-2}]
```

重新画上面的图形



```
\begin{tikzpicture}[scale=2]
\draw (0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle
(0,0) circle (1cm)
(0,0) rectangle (1,1)
(0,0) -- (1,1);
\end{tikzpicture}
```

```

\draw [x={(-60:1)},y={{-sqrt(3)},-2}}
(0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle
(0,0) circle (1cm)
(0,0) rectangle (1,1)
(0,0) -- (1,1);
\draw [red]({-0.65*sqrt(3)},0.65)--({0.65*sqrt(3)},-0.65)
node [below] {$y=-\frac{\sqrt{3}}{3}x$};
\end{tikzpicture}

```

其中第 2 个 `\draw` 命令就是在标架 $\left\{\left(\frac{1}{2}, -\frac{\sqrt{3}}{2}\right), \left(-\frac{\sqrt{3}}{2}, -\frac{1}{2}\right)\right\}$ 下画图形的。由于这个标架与初始标架 $\{(1,0), (0,1)\}$ 关于直线 $y = -\frac{\sqrt{3}}{3}x$ 对称, 故上面画出的两个正方形关于该直线对称, 此直线的方向是 $(\cos(-30^\circ), \sin(-30^\circ))$ 。

注意下面两个命令中的变换并不等价

```

\draw [x={(-60:1)},y={{-sqrt(3)},-2}}
(0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle;

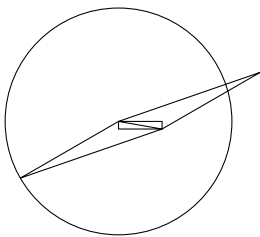
\draw [y={{-sqrt(3)},-2}][x={(-60:1)}]
(0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle;

```

第一个命令如上图, 其变换选项确定一个左手直角标架。第二个命令的变换选项分成两个方括号, 其总的变换效果是:

$$(0,0) \rightarrow (0,0), \quad (1,0) \rightarrow \left(\frac{5}{4}, \frac{\sqrt{3}}{4}\right), \quad (1,1) \rightarrow \left(\frac{5-2\sqrt{3}}{4}, \frac{\sqrt{3}-2}{4}\right), \quad (0,1) \rightarrow \left(-\frac{\sqrt{3}}{2}, -\frac{1}{2}\right),$$

得到的是个菱形, 如下图

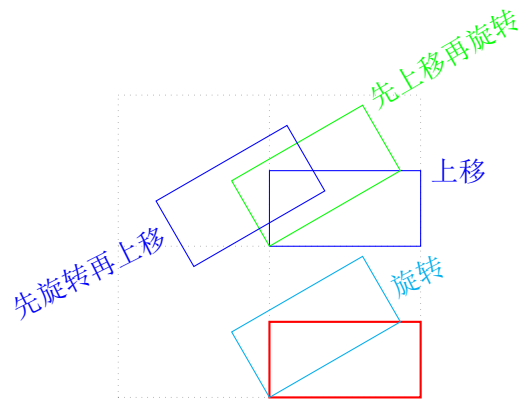


```

\begin{tikzpicture}[scale=1.5]
\draw [y={{-150:1}}][x={(-60:1)}]
(0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle
(0,0) circle (1cm)
(0,0) rectangle (1,1)
(0,0) -- (1,1);
\end{tikzpicture}

```

- 不同的变换次序对结果可能有不同影响。

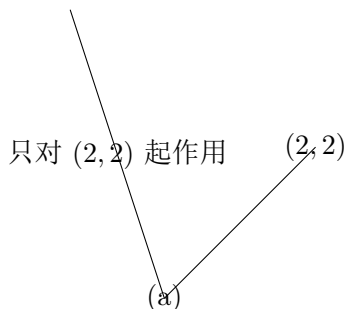


```
\begin{tikzpicture}[scale=2]
\draw [help lines,dotted] (-1,0) grid (1,2);
\draw [red, thick] (0,0) rectangle (1,0.5);
\draw [yshift=1cm] [blue] (0,0) rectangle (1,0.5) node[right]{上移};
\draw [rotate=30] [cyan] (0,0) rectangle node[sloped,right=1cm]{旋转} (1,0.5) ;
\draw [yshift=1cm,rotate=30] [green]
    (0,0) rectangle (1,0.5) node[rotate=30,right]{先上移再旋转};
\draw [rotate=30,yshift=1cm] [blue]
    (0,0) rectangle node[sloped,left=1cm]{先旋转再上移} (1,0.5);
\end{tikzpicture}
```

- 前面提到，坐标变换矩阵不能用于不涉及坐标的项目，即使该项目间接地涉及坐标。例如，假设写出

```
\coordinate (a) at (1,1);
% 或者
\tikzmath{
    \bx=(3-sqrt(3))/2; \by=0; \cx=3/2; \cy=-1/2;
coordinate \b, \c;
    \b=(\bx, \by);
    \c=(\cx, \cy);
}
```

尽管其中的名称 (a), (\b), (\c) 代表坐标点，但是坐标变换对“名称”无效，也就是说，如果路径中用到了名称 (a)，该名称代表的点 (1,1) 在坐标变换下保持不动；其它的凡是以 (x, y) 或 (d:1) 形式，或者以坐标运算形式 (\$... \$)、(\$...!...!...\$) 提供的点，都接受变换选项的作用，其中 x, y, d, 1 可以是命令 \tikzmath{} 提供的实数型数值名称（下面有这种例子）。



```

\begin{tikzpicture}
\coordinate (a) at (0,0);
\coordinate (b) at (2,2);
\draw [shift={(-1,1)},rotate=50]
(a) -- node{只对 $(2,2)$ 起作用} (2,2);
\draw (0,0) node{(a)}--(2,2) node{$(2,2)$};
\end{tikzpicture}

```

- 可以在路径中使用多个方括号，这种变换方式与数学上的习惯接近但仍有差别。

25.3.3 平面上的轴对称

下面分析平面上的轴对称作图，从中可以展示坐标变换中的“当前性”和“变换次序”对结果的影响。

给定一个线段 AB ，一个图形 G ，我们作出 G 关于直线 AB 对称的图形 G' 。对不同情况可以有不同方法，下面列举几种方法。

方法一： 如果图形 G 是由几个点决定的直线形或简单曲线，可以先找出这几个点的对称点，然后把这些对称点连接起来，即可得到对称图形。可以使用表达式 $(A)!(B)!(C)$ 和 $(A)!x!(B)$ 确定对称点。

方法二： 使用 `cm` 变换选项。先对一个点做一般分析。设点 $\mathbf{A} = (a_1, a_2)$ ， $\mathbf{B} = (b_1, b_2)$ 确定直线 AB ，点 $\mathbf{P} = (p_1, p_2)$ 关于直线 AB 的对称点是 $\mathbf{P}' = (p'_1, p'_2)$ ，用 \mathbf{A} ， \mathbf{B} ， \mathbf{P} 表达 \mathbf{P}' ：

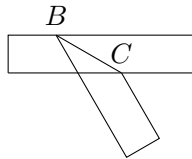
$$\begin{aligned}
 \mathbf{P}' &= \mathbf{P} - \mathbf{A} - 2(\mathbf{e}, \mathbf{P} - \mathbf{A})\mathbf{e} + \mathbf{A} \\
 &= (\mathbf{I} - 2(\mathbf{e}, \mathbf{e}^T))(\mathbf{P} - \mathbf{A}) + \mathbf{A} \\
 &= \begin{pmatrix} \frac{(a_1 - b_1)^2 - (a_2 - b_2)^2}{(a_1 - b_1)^2 + (a_2 - b_2)^2} & \frac{2(a_1 - b_1)(a_2 - b_2)}{(a_1 - b_1)^2 + (a_2 - b_2)^2} \\ \frac{2(a_1 - b_1)(a_2 - b_2)}{(a_1 - b_1)^2 + (a_2 - b_2)^2} & \frac{(a_2 - b_2)^2 - (a_1 - b_1)^2}{(a_1 - b_1)^2 + (a_2 - b_2)^2} \end{pmatrix} \begin{pmatrix} p_1 - a_1 \\ p_2 - a_2 \end{pmatrix} + \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \\
 &= \frac{1}{(a_1 - b_1)^2 + (a_2 - b_2)^2} \begin{pmatrix} (a_1 - b_1)^2 - (a_2 - b_2)^2 & 2(a_1 - b_1)(a_2 - b_2) \\ 2(a_1 - b_1)(a_2 - b_2) & (a_2 - b_2)^2 - (a_1 - b_1)^2 \end{pmatrix} \begin{pmatrix} p_1 - a_1 \\ p_2 - a_2 \end{pmatrix} + \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \\
 &= \begin{pmatrix} a & c \\ b & d \end{pmatrix} \begin{pmatrix} p_1 - a_1 \\ p_2 - a_2 \end{pmatrix} + \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \\
 &= \begin{pmatrix} p'_1 \\ p'_2 \end{pmatrix}
 \end{aligned}$$

其中 \mathbf{e} 是直线 AB 的单位法向量， $\begin{pmatrix} p_1 - a_1 \\ p_2 - a_2 \end{pmatrix}$ 是以 \mathbf{A} 为平移向量对 \mathbf{P} 做的平移。只要计算出上式中的各个元素就可以用 `cm` 选项作轴对称变换。可以调用数学程序库进行计算。

假设图形 G 的绘图代码是 `\draw <G-code>`，那么对称图形 G' 可以用下面的代码画出：

```
\draw [cm={a,b,c,d,(a1,a2)}] [shift={(-a1,-a2)}] <G-code>;
```

注意代码中用了两个方括号，二者次序不能调换。首先是 `[shift={(-a1,-a2)}` 对 `<G-code>` 中的点以 $-\mathbf{A}$ 为平移向量做的平移，然后是 `cm` 选项的作用。还要注意，`<G-code>` 中不能出现坐标名称，因为变换对坐标名称无效。下面是一个例子，将一个矩形纸条沿着 BC 折叠：



```
\tikzmath{
  \bx=(3-sqrt(3))/2; \by=0; \cx=3/2; \cy=-1/2;
  \ff=((\bx-\cx)^2+(\by-\cy)^2)^(-1);
  \fa=(\bx-\cx)^2-(\by-\cy)^2; \fd=-\fa; \fbc=2*(\bx-\cx)*(\by-\cy);
  \a=\ff*\fa; \b=\ff*\fbc; \c=\b; \d=-\a;
}

\begin{tikzpicture}
\coordinate (a) at (0,0);
\coordinate [label=above:$B$] (b) at (\bx,\by);
\coordinate [label=above:$C$] (c) at (\cx,\cy);
\coordinate (d) at (2.5,0);
\coordinate (e) at (2.5,-0.5);
\coordinate (f) at (0,-0.5);
\draw (b)--(a)--(f);
\draw (f)--(c);
\draw (b)--(c);
\draw (b)--(d)--(e)--(c);
\draw [cm={\a,\b,\c,\d,(\bx,\by)}] [shift={(-\bx,-\by)}]
  (0:\bx) -- (2.5,0) -- (2.5,-0.5) -- (\cx,\cy);
\end{tikzpicture}
```

方法三：利用 `x={}`，`y={}` 和 `shift={}` 选项。

在不变画布（初始坐标系） xOy 中，设 \mathbf{P} 是原图形 G 上的任一点，要找出点 \mathbf{P} 关于直线 \mathbf{AB} 的对称点 \mathbf{P}' ，按以下步骤分析：

1. 先做平移： $\mathbf{P} \rightarrow \mathbf{P} - \mathbf{A}$.
2. 找出 xOy 坐标系关于直线 $t \cdot (\mathbf{A} - \mathbf{B})$ 对称的坐标系 $x'Oy'$.
3. 找出点 \mathbf{Q} ，该点在 $x'Oy'$ 中的坐标为 $\mathbf{Q}_{x'Oy'} = \mathbf{P} - \mathbf{A}$ ，在 xOy 中的坐标记为 $\mathbf{Q}_{xOy} = (q_1, q_2)$.
4. 在 xOy 坐标系中做平移： $\mathbf{Q}_{xOy} \rightarrow \mathbf{Q}_{xOy} + \mathbf{A} = \mathbf{P}'$ 得到要找的点 \mathbf{P}' .

算式

$$(\mathbf{I} - 2(\mathbf{e}, \mathbf{e}^T))(\mathbf{P} - \mathbf{A}) = \mathbf{Q},$$

表示点 $\mathbf{P} - \mathbf{A}$ 关于参数直线 $t \cdot (\mathbf{A} - \mathbf{B})$ 的对称点。

坐标系 $x'Oy'$ 是左手系, 可设其单位向量在 xOy 坐标系内的表达式是

$$x' \text{轴单位向量} : \begin{pmatrix} \cos \varphi \\ -\sin \varphi \end{pmatrix} = (-\varphi : 1), \quad y' \text{轴单位向量} : \begin{pmatrix} -\sin \varphi \\ -\cos \varphi \end{pmatrix} = (-\varphi - 90^\circ : 1)$$

设 $\mathbf{P} - \mathbf{A}$ 在 xOy 坐标系内的坐标是 (α, β) , 则

$$\begin{aligned} xOy \text{系} : \mathbf{P} - \mathbf{A} &= \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ x'Oy' \text{系} : \mathbf{Q}_{x'Oy'} &= \alpha \begin{pmatrix} \cos \varphi \\ -\sin \varphi \end{pmatrix} + \beta \begin{pmatrix} -\sin \varphi \\ -\cos \varphi \end{pmatrix} \end{aligned}$$

再找出 \mathbf{A}, \mathbf{B} 与 $(-\varphi : 1)$ 和 $(-\varphi - 90^\circ : 1)$ 的关系。

参数直线 $t \cdot (\mathbf{A} - \mathbf{B})$ 的方向是 $\mathbf{A} - \mathbf{B}$, 如果能求得这个方向与 xOy 系的单位向量之间的夹角, 用对称的办法就能容易地知道系 $x'Oy'$ 的单位向量的坐标。设从方向向量 $\mathbf{A} - \mathbf{B}$ 到单位向量 $(1, 0)$ 的夹角是 θ_1 , 从方向向量 $\mathbf{A} - \mathbf{B}$ 到单位向量 $(0, 1)$ 的夹角是 θ_2 , $|\theta_i| \leq 180^\circ, i = 1, 2$, 计算外积

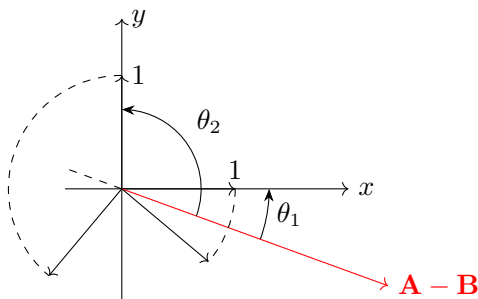
$$(\mathbf{A} - \mathbf{B}) \times (1, 0) = |\mathbf{A} - \mathbf{B}| \sin \theta_1 = b_2 - a_2$$

$$(\mathbf{A} - \mathbf{B}) \times (0, 1) = |\mathbf{A} - \mathbf{B}| \sin \theta_2 = a_1 - b_1$$

注意反三角函数的值域 $0^\circ \leq \arccos t \leq 180^\circ$, $-90^\circ \leq \arcsin t \leq 90^\circ$, 可以分以下几种情况分析

(i) 如果 $b_2 - a_2 > 0$ 且 $a_1 - b_1 > 0$, 则 $\mathbf{A} - \mathbf{B}$ 在第四象限, 此时

$$\theta_1 = \arcsin \frac{b_2 - a_2}{|\mathbf{A} - \mathbf{B}|}, \quad \theta_2 = \theta_1 + 90^\circ$$



坐标系 xOy 变为坐标系 $x'Oy'$, 则 x 轴的单位向量 $(1, 0)$ 变为 x' 轴的单位向量

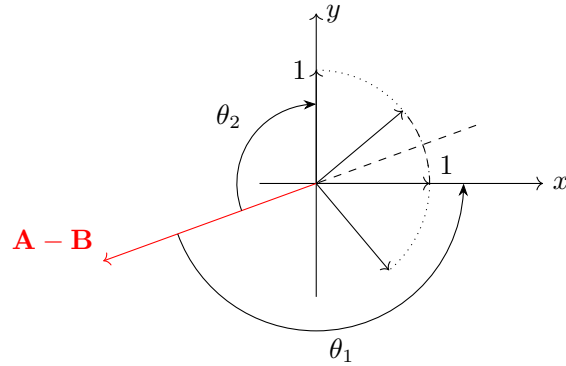
$$(\cos(-2\theta_1), \sin(-2\theta_1)) = (\cos 2\theta_1, -\sin 2\theta_1)$$

因为坐标系 $x'Oy'$ 是左手系, 故 y 轴的单位向量 $(0, 1)$ 变为 y' 轴的单位向量

$$(-\sin 2\theta_1, -\cos 2\theta_1)$$

(ii) 如果 $b_2 - a_2 > 0$ 且 $a_1 - b_1 < 0$, 则 $\mathbf{A} - \mathbf{B}$ 在第三象限, 此时

$$\theta_1 = 180^\circ - \arcsin \frac{b_2 - a_2}{|\mathbf{A} - \mathbf{B}|}, \quad \theta_2 = \theta_1 - 270^\circ$$



x 轴的单位向量 $(1, 0)$ 变为 x' 轴的单位向量

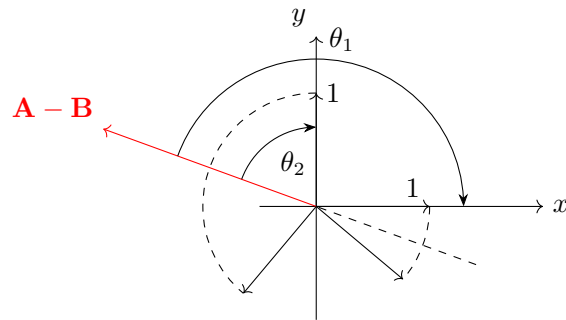
$$(\cos 2(180^\circ - \theta_1), \sin 2(180^\circ - \theta_1)) = (\cos 2\theta_1, -\sin 2\theta_1)$$

y 轴的单位向量 $(0, 1)$ 变为 y' 轴的单位向量

$$(-\sin 2\theta_1, -\cos 2\theta_1)$$

(iii) 如果 $b_2 - a_2 < 0$ 且 $a_1 - b_1 < 0$, 则 $\mathbf{A} - \mathbf{B}$ 在第二象限, 此时

$$\theta_2 = \arcsin \frac{a_1 - b_1}{|\mathbf{A} - \mathbf{B}|}, \quad \theta_1 = \theta_2 - 90^\circ = -180^\circ - \arcsin \frac{b_2 - a_2}{|\mathbf{A} - \mathbf{B}|}$$



x 轴的单位向量 $(1, 0)$ 变为 x' 轴的单位向量

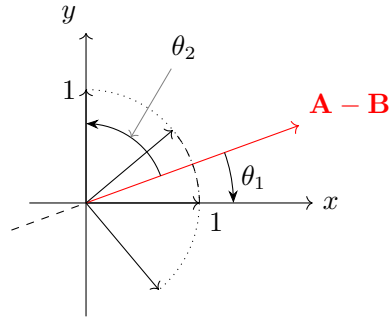
$$(\cos(-2(180^\circ + \theta_1)), \sin(-2(180^\circ + \theta_1))) = (\cos 2\theta_1, -\sin 2\theta_1)$$

y 轴的单位向量 $(0, 1)$ 变为 y' 轴的单位向量

$$(-\sin 2\theta_1, -\cos 2\theta_1)$$

(iv) 如果 $b_2 - a_2 < 0$ 且 $a_1 - b_1 > 0$, 则 $\mathbf{A} - \mathbf{B}$ 在第一象限, 此时

$$\theta_2 = \arcsin \frac{a_1 - b_1}{|\mathbf{A} - \mathbf{B}|}, \quad \theta_1 = \theta_2 - 90^\circ = \arcsin \frac{b_2 - a_2}{|\mathbf{A} - \mathbf{B}|}$$



x 轴的单位向量 $(1, 0)$ 变为 x' 轴的单位向量

$$(\cos(-2\theta_1), \sin(-2\theta_1)) = (\cos 2\theta_1, -\sin 2\theta_1)$$

y 轴的单位向量 $(0, 1)$ 变为 y' 轴的单位向量

$$(-\sin 2\theta_1, -\cos 2\theta_1)$$

(v) 对于 $b_2 - a_2 = 0$ 或 $a_1 - b_1 = 0$ 的情况, 显然, x 轴的单位向量 $(1, 0)$ 变为 x' 轴的单位向量

$$(\cos(-2\theta_1), \sin(-2\theta_1)) = (\cos 2\theta_1, -\sin 2\theta_1)$$

y 轴的单位向量 $(0, 1)$ 变为 y' 轴的单位向量

$$(-\sin 2\theta_1, -\cos 2\theta_1)$$

总结起来, 有

$\mathbf{A} - \mathbf{B}$	$-180^\circ \leq \theta_1 \leq 180^\circ$	反正弦值	与 $-2\theta_1$ 终边相同的角度
第一象限	$\theta_1 = \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} }$	$-90^\circ \leq \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} } \leq 0^\circ$	$-2 \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} }$
第二象限	$\theta_1 = -180^\circ - \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} }$	$-90^\circ \leq \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} } \leq 0^\circ$	$2 \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} }$
第三象限	$\theta_1 = 180^\circ - \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} }$	$0^\circ \leq \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} } \leq 90^\circ$	$2 \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} }$
第四象限	$\theta_1 = \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} }$	$0^\circ \leq \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} } \leq 90^\circ$	$-2 \arcsin \frac{b_2 - a_2}{ \mathbf{A} - \mathbf{B} }$

并且总有

xOy 系的标架

$x'Oy'$ 系的标架

$$(1, 0) \longrightarrow (\cos 2\theta_1, -\sin 2\theta_1) = (-2\theta_1 : 1)$$

$$(0, 1) \longrightarrow (-\sin 2\theta_1, -\cos 2\theta_1) = (-2\theta_1 - 90^\circ : 1)$$

令

$$\xi = \frac{a_1 - b_1}{|a_1 - b_1|} \arcsin \frac{b_2 - a_2}{|\mathbf{A} - \mathbf{B}|},$$

则 -2ξ 与 $-2\theta_1$ 的终边相同, 所以

xOy 系的标架

$x'Oy'$ 系的标架

$$(1, 0) \longrightarrow (\cos 2\xi, -\sin 2\xi) = (-2\xi : 1)$$

$$(0, 1) \longrightarrow (-\sin 2\xi, -\cos 2\xi) = (-2\xi - 90^\circ : 1)$$

还有

$\mathbf{A} - \mathbf{B}$	ξ	θ_1
第一象限	$-90^\circ \leq \xi \leq 0^\circ$	$\theta_1 = \xi$
第二象限	$0^\circ \leq \xi \leq 90^\circ$	$\theta_1 = \xi - 180^\circ$
第三象限	$-90^\circ \leq \xi \leq 0^\circ$	$\theta_1 = \xi + 180^\circ$
第四象限	$0^\circ \leq \xi \leq 90^\circ$	$\theta_1 = \xi$

由此容易得到 ξ 的正弦值是

$$\sin \xi = \frac{a_1 - b_1}{|a_1 - b_1|} \frac{b_2 - a_2}{|\mathbf{A} - \mathbf{B}|},$$

再用 $\cos^2 t + \sin^2 t = 1$ 可以得到 ξ 的余弦值是

$$\cos \xi = \frac{a_1 - b_1}{|\mathbf{A} - \mathbf{B}|},$$

所以

$$\begin{aligned} \cos(-2\theta_1) &= \cos(-2\xi) = \frac{(a_1 - b_1)^2 - (a_2 - b_2)^2}{(a_1 - b_1)^2 + (a_2 - b_2)^2} \\ \sin(-2\theta_1) &= \sin(-2\xi) = -2 \frac{a_1 - b_1}{|a_1 - b_1|} \frac{(a_1 - b_1)(a_2 - b_2)}{(a_1 - b_1)^2 + (a_2 - b_2)^2} \end{aligned}$$

可以调用数学程序库计算 ξ ，例如，

```
\tikzmath{
  coordinate \dirvec;
  \dirvec=(a1 - b1, a2 - b2); %方向向量 A-B
  \xival=sign(a1 - b1) * asin(-\dirvecy / veclen(\dirvec));
}
```

由于变换选项会改变当前可变画布，所以为了能转换到 $x'Oy'$ 系的标架，需要计算 $(-\sin 2\theta_1, -\cos 2\theta_1)$ 在标架 $\{(\cos 2\theta_1, -\sin 2\theta_1), (0, 1)\}$ 下的坐标，先计算逆矩阵

$$T = \begin{bmatrix} \cos 2\theta_1 & 0 \\ -\sin 2\theta_1 & 1 \end{bmatrix}, \quad T^{-1} = \begin{bmatrix} \frac{1}{\cos 2\theta_1} & 0 \\ \frac{\sin 2\theta_1}{\cos 2\theta_1} & 1 \end{bmatrix}$$

需要计算的坐标就是

$$T^{-1} \begin{pmatrix} -\sin 2\theta_1 \\ -\cos 2\theta_1 \end{pmatrix} = \begin{pmatrix} -\frac{\sin 2\theta_1}{\cos 2\theta_1} \\ -\frac{\sin^2 2\theta_1}{\cos 2\theta_1} - \cos 2\theta_1 \end{pmatrix} = \frac{-1}{\cos 2\theta_1} \begin{pmatrix} \sin 2\theta_1 \\ 1 \end{pmatrix} = \frac{-1}{\cos 2\xi} \begin{pmatrix} \sin 2\xi \\ 1 \end{pmatrix},$$

即

$$\left(-2 \frac{a_1 - b_1}{|a_1 - b_1|} \frac{(a_1 - b_1)(a_2 - b_2)}{(a_1 - b_1)^2 - (a_2 - b_2)^2}, -\frac{(a_1 - b_1)^2 - (a_2 - b_2)^2}{(a_1 - b_1)^2 + (a_2 - b_2)^2} \right).$$

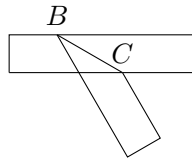
还是假设图形 G 的绘图代码是 `\draw <G-code>`，那么对称图形 G' 可以用下面的代码画出：

```
\draw [shift={(a1, a2)}]
  [x={(-2*\xival : 1)}, y={{(1/\cos(2*\xival))}*(\sin(2*\xival)}, 1)}]
  [shift={{-1*(a1, a2)}}]
  <G-code> ;
```

% 注意 <G-code> 中不能用 `coordinate` 名称

注意，这里必须用三个方括号分别设置选项，而且这三个方括号的次序不能调换，第二个方括号内的两个选项次序也不能换。首先是第三个方括号的选项作用于路径，然后是第二个方括号的选项作用，最后是第一个方括号的选项作用。第三个方括号表示对路径上的点作 xOy 系内的平移 $-\mathbf{A}$ 。第二个方括号表示转换到 $x'Oy'$ 系画图，是个反射变换。第一个方括号表示 xOy 系内的平移 $+\mathbf{A}$ 。

用这个方法重画前面折叠矩形纸条的例子：



```
\tikzmath{
  \bx=(3-sqrt(3))/2; \by=0; \cx=3/2; \cy=-1/2;
  coordinate \dirvec;
  \dirvec=(\bx - \cx, \by - \cy); %方向向量 A-B
  \xival=sign(\bx-\cx)*asin(-\dirvecy / veclen(\dirvec));
}

\begin{tikzpicture}
\coordinate (a) at (0,0);
\coordinate [label=above:$B$] (b) at (\bx,\by);
\coordinate [label=above:$C$] (c) at (\cx,\cy);
\coordinate (d) at (2.5,0);
\coordinate (e) at (2.5,-0.5);
\coordinate (f) at (0,-0.5);
\draw (b)--(a)--(f);
\draw (f)--(c);
\draw (b)--(c);
\draw (b)--(d)--(e)--(c);
\draw [shift={(\bx,\by)}
  [x={(-2*\xival:1)}, y={{-sin(2*\xival)/cos(2*\xival)}, -1/cos(2*\xival)}}]
  [shift={{-\bx,-\by}}]
  (0:\bx) -- (2.5,0) -- (2.5,-0.5) -- (\cx,\cy);
\end{tikzpicture}
```

这个方法计算量较大。

方法四：利用 $y={}$ ， $shift={}$ 和 $rotate={}$ 选项。

继续沿用前面设定的符号。用下面的思路找出点 \mathbf{P} 的对称点 \mathbf{P}' ：

1. 先做平移： $\mathbf{P} \rightarrow \mathbf{P} - \mathbf{A}$ 。

2. 转换标架系

$$\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\} \rightarrow \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right\}.$$

3. 假设向量 $\mathbf{A} - \mathbf{B}$ 到单位向量 $(1, 0)$ 的夹角是 θ_1 , 将标架系 $\left\{ \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right\}$ 旋转 $-2\theta_1$ 角度, 相当于旋转 -2ξ 角度:

$$\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right\} \rightarrow \left\{ \begin{pmatrix} \cos 2\xi \\ -\sin 2\xi \end{pmatrix}, \begin{pmatrix} -\sin 2\xi \\ -\cos 2\xi \end{pmatrix} \right\}.$$

由此得到的坐标系 $x'Oy'$ 与坐标系 xOy 关于直线 $t \cdot (\mathbf{A} - \mathbf{B})$ 对称.

4. 在 $x'Oy'$ 中找出坐标为 $\mathbf{Q}_{x'Oy'} = \mathbf{P} - \mathbf{A}$ 的点 \mathbf{Q} , 它在 xOy 中的坐标记为 $\mathbf{Q}_{xOy} = (q_1, q_2)$.

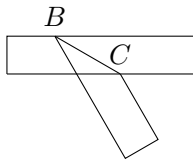
5. 在 xOy 坐标系中做平移: $\mathbf{Q}_{xOy} \rightarrow \mathbf{Q}_{xOy} + \mathbf{A} = \mathbf{P}'$.

还是利用数学程序库计算 ξ , 绘图代码可以是

```
\tikzmath{
  coordinate \dirvec;
  \dirvec=(a1 - b1, a2 - b2); %方向向量 A-B
  \xival=sign(a1 - b1) * asin(-\dirvecy / veclen(\dirvec));
}

\draw [shift={(a1, a2)}]
  [y={(0, -1)}, rotate=-2*\xival]
  [shift={($-1*(a1, a2)$)}]
  <G-code> ;
% 注意 <G-code> 中不能用 coordinate 名称 >;
```

用这个思路重画前面折叠矩形纸条的图形:



```
\tikzmath{
  \bx=(3-sqrt(3))/2; \by=0; \cx=3/2; \cy=-1/2;
  coordinate \dirvec;
  \dirvec=(\bx - \cx, \by - \cy); %方向向量 A-B
  \xival=sign(\bx-\cx)*asin(-\dirvecy / veclen(\dirvec));
}

\begin{tikzpicture}
\coordinate (a) at (0,0);
\coordinate [label=above:$B$] (b) at (\bx,\by);
```

```

\coordinate [label=above:$C$] (c) at (\cx,\cy);
\coordinate (d) at (2.5,0);
\coordinate (e) at (2.5,-0.5);
\coordinate (f) at (0,-0.5);
\draw (b)--(a)--(f);
\draw (f)--(c);
\draw (b)--(c);
\draw (b)--(d)--(e)--(c);
\draw [shift={(\bx,\by)}]
      [y={(0, -1)}, rotate=-2*\xival]
      [shift={(-\bx,-\by)}]
      (0:\bx) -- (2.5,0) -- (2.5,-0.5) -- (\cx,\cy);
\end{tikzpicture}

```

可见，与前两种方法相比，由于用了旋转变换选项，这个方法的代码稍微简洁一些。

25.4 画布变换

参考 §94.4. 打个比方说，当气球充气时，气球上的图画变化就可以比喻成“画布变换”。线条、文字以及其它的项目都发生变化。当做画布变换时，PGF 不再跟踪 node 的位置，也不再计算图形的尺寸，因为它目前还不能将画布变换的结果纳入计算之内。在将来可能会改进这一点。

画布变换会作用于整个路径，不能只作用于某一段子路径，这与坐标变换不同。

应尽量避免使用画布变换。

```
/tikz/transform canvas=<options>
```

这个选项引入画布变换，其中 <options> 所包含的选项，就是前面所讲的关于坐标变换的选项。画布变换与坐标变换会叠加。

50 Externalization Library

调用 external 程序库：`\usetikzlibrary{external}`。

这个程序库可以将 tex 文件中，用 tikz 语句绘制的图形输出为单独的图形，输出图形的格式可以是 pdf, eps 等。

50.1 Overview

50.2 Requirements

激活程序库的输出功能后，当程序遇到 `\begin{tikzpicture}` 时开始收集绘图内容，直到遇到随后第一个 `\end{tikzpicture}` 时中止收集，并将收集的内容处理用于输出。所以，如果在 `{tikzpicture}` 环境内直接套嵌 `{tikzpicture}` 环境，输出图形时极有可能得到意外结果。因此，套嵌的内层 `{tikzpicture}` 环境应该放在花括号分组内，或者作为其它命令的调用内容。

程序库会假定 L^AT_EX 的 `picture` 环境在遇到 `\end{tikzpicture}` 时结束。

50.3 A Word About ConTEXt And Plain TEX

50.4 输出图形

在调用 `external` 程序库后，在导言区使用命令

```
\tikzexternalize
\tikzexternalize[<optional arguments>]
```

命令 `\tikzexternalize` 用以激活图形输出功能。在默认之下，图形输出到当前文件夹。程序会为输出的图形命名，默认名称依次是 `<mani file name>-figure0`，`<mani file name>-figure1`，`<mani file name>-figure2` 等等，其中 `<mani file name>` 是当前 `tex` 文件的名称，末尾的编号从 0 开始。

```
\tikzexternalrealjob
```

在调用程序库后，这个宏总是被赋值为当前的主文件名 `<mani file name>`。

```
\pgfactualjobname
```

在调用命令 `\tikzexternalize` 后，这个宏总是被赋值为当前输出的文件名（图形名）。

```
\jobname
```

根据不同的设置，这个宏的值是 `\tikzexternalrealjob` 或 `\pgfactualjobname` 二者之一。例如，如果激活 `auxiliary` 文件的支持功能，那么 `\jobname=\tikzexternalrealjob`（因为这是 `auxiliary` 文件的名称）。

```
/tikz/external/system call={<template>} (无默认值)
```

`<template>` 是模板字符串，用来产生系统调用（`system calls`）。在 `{<template>}` 中，当 `\texsource` 含有主文件名 `<mani file name>`（实际上，这个宏含有 `\input{<mani file name>}`）时，宏 `\image` 可以用作（待生成的图形的）占位符。

该选项的设置依赖宏 `\pgfsysdriver` 的值，例如，如果定义

```
\def\pgfsysdriver{pgfsys-pdf.tex.def}
```

那么该选项就是

```
\tikzset{external/system call={pdf\latex \tikzexternalcheckshellescape
-halt-on-error-interaction=batchmode -jobname "\image" "\texsource"}}
```

仅在当前文档用 `-shell-escape` 排版时，宏 `\tikzexternalcheckshellescape` 才会插入选项 `shell escape` 的值。

其它驱动需要不同的 `<template>` 来做系统调用，支持 `lualatex`，`xelatex`，`dvips` 等驱动。在编辑文档时，准确的 `<template>` 会被写入 `.log` 文件。

定义驱动（参考 §10）的语句如下：

```
\def\pgfsysdriver{pgfsys-dvipdfm.def}
\def\pgfsysdriver{pgfsys-dvipdfmx.def}
```

```

\def\pgfsysdriver{pgfsys-dvips.def}
\def\pgfsysdriver{pgfsys-pdfTeX.def}
\def\pgfsysdriver{pgfsys-tex4ht.def}
\def\pgfsysdriver{pgfsys-textures.def}
\def\pgfsysdriver{pgfsys-vTeX.def}
\def\pgfsysdriver{pgfsys-xeTeX.def}

```

在载入 PGF 之前定义驱动。

程序会用 `\edef` 将 `{<template>}` 展开，因此诸多控制序列都会被展开。在展开过程中，“`\`”变成普通符号“`\`”，双引号，单引号，分号，连字符都会（由宏）变成普通符号（如果有某个宏包将它们作为宏的话），这能确保与 `german` 等宏包兼容。

`/tikz/external/shell escape={<command-line arg>}`（无默认值，初始值 `-shell-escape`）

这个选项的值是 L^AT_EX 命令行的参数选项，它将激活 `\write18` 特性。对于 T_EX-Live 来说，该选项的值是 `-shell-escape`。

50.4.1 Support for Labels and References In External Files

50.4.2 设置输出图形的名称

输出图形的默认名称是 `<real file name>-figure_<number>` 这种形式，这里 `<number>` 从 0 开始编号。如果不为某个图形指定名称，就采用这种默认格式为该图形命名。图形名称包括：前缀、名称、后缀、编号这 4 个部分，其中前缀、后缀可以不设置，除了用命令 `\tikzsetnextfilename` 设置的名称外，其余名称都会带有编号（从 0 开始编号）。

`/tikz/external/prefix={<file name prefix>}`（无默认值，初始值 `empty`）

这个选项定义当前 tex 文件中，所有输出图形名称的前缀。它与下面的命令等效。

```
\tikzsetexternalprefix{<file name prefix>}
```

例如

```
\tikzsetexternalprefix{figures/}
```

将为所有输出图形名称加上前缀 `figures/`，注意如果没有安装 PGF，那么只能用这个命令给图形名加前缀。

这个设置可以放在命令

```
\tikzsetfigurename{<name>}
```

之前或之后。

```
\tikzsetnextfilename{<file name>}
```

设置下一个图形的名称，前缀会自动在这个名称之前。该命令命名结束后，就恢复之前的命名设置。

`/tikz/external/figure name={<name>}`

该选项为之后的图形设置名称。等效于下一个命令。

```
\tikzsetfigurename{<name>}
```


该命令设置其后的所有图形的名称，直到遇到 `\tikzsetnextfilename{<file name>}` 或下一个 `\tikzsetfigurename{<name>}`。注意这个命令的有效范围受到环境、分组的限制。这个命令设置的是图形的“主名称”，该命令产生的各个图形名称会以后缀编号来相互区别，编号从 0 开始。

```
\tikzset{external/figure name/.add={prefix_}{_suffix_}}
```

这个选项设置输出图形的前缀为“prefix_”，后缀为“_suffix_”，注意这个设置要用在命令 `\tikzsetfigurename{<name>}` 之后。

```
\tikzappendtofigurename{<suffix>}
```

这个命令设置图形名称的后缀，注意它要用在命令 `\tikzsetfigurename{<name>}` 之后。
举例说明。

```
\documentclass{article}
% 主文件，名称为 main.tex
\usepackage{tikz}
\usetikzlibrary{external}
\tikzexternalize[prefix=figures/] % 激活并给图形文件名加前缀figures/
\begin{document}
\tikzsetnextfilename{trees} % 设置下一图形的名称为 trees
\begin{tikzpicture} % 该环境生成的图形名称将是 figures/trees.pdf
\node {root}
  child {node {left}}
  child {node {right}}
    child {node {child}}
    child {node {child}}
  };
\end{tikzpicture}
\tikzsetnextfilename{simple} % 设置下一图形的名称为 simple
A simple image is \tikz \fill (0,0) circle(5pt);. % 该图形名称将是 figures/simple.pdf
\begin{tikzpicture} % 该环境生成的图形名称将是 figures/main-figure0.pdf
\draw[help lines] (0,0) grid (5,5);
\end{tikzpicture}
\end{document}
```

下面是另一个例子。

```
\documentclass{article}
% 主文件，名称为 main.tex
\usepackage{tikz}
\usetikzlibrary{external}
\tikzexternalize % 激活
```

```

\begin{document}
\begin{tikzpicture} % 该环境生成的图形名称将是 main-figure0.pdf
\node {root}
  child {node {left}}
  child {node {right}}
    child {node {child}}
    child {node {child}}
  };
\end{tikzpicture}
{
\tikzsetfigurename{subset_}
A simple image is \tikz \fill (0,0) circle(5pt);. % 该图形名称将是 subset_0.pdf'
\begin{tikzpicture} % 该环境生成的图形名称将是 subset_1.pdf
\draw[help lines] (0,0) grid (5,5);
\end{tikzpicture}
} % 名称 subset_ 至此失效, 后面将用原来的名称并继续编号
\begin{tikzpicture} % 该环境生成的图形名称将是 main-figure1.pdf
\draw (0,0) -- (5,5);
\end{tikzpicture}
\end{document}

```

50.4.3 Remaking Figures or Skipping Figures

```

\tikzpicturedependsonfile{<file name>}
\tikzexternalfiledependsonfile{<external graphics>}{<file name>}
/tikz/external/disable dependency files
/tikz/external/force remake={<boolean>} (默认 true)
/tikz/external/remake next={<boolean>} (默认 true)
/tikz/external/export next={<boolean>} (默认 true)

```

指示不输出下一个图形。

```

/tikz/external/export={<boolean>} (默认 true)

```

指示该选项之后的图形不输出, 其有效范围受到分组的限制。

```

\tikzexternaldisable

```

该命令关闭图形输出功能, 其后的图形不输出 (它之前的输出)。其有效范围受到分组的限制。

```

\tikzexternalenable

```

这个命令用在命令 `\tikzexternaldisable` 之后, 重启图形输出功能。其有效范围受到分组的限制。

53 定点算术程序库

L^AT_EX 的宏包 `fp` 提供了较强的定点算术功能，定点算术程序库提供了使用该宏包的一个“界面”。首先调用宏包 `fp`，然后再载入定点算术程序库 `fixedpointarithmetic`。

```
\usepackage{fp}
\usetikzlibrary{fixedpointarithmetic}
```

53.1 Overview

PGF 的数学引擎在解析表达式时，有较快的速度和弹性，但是精度有时较低。受到 T_EX 的计算能力的限制，数学引擎能计算的数据范围不超过 ± 16383.99999 。而宏包 `fp` 提供了更高的精度和更广的数据计算范围（大约是 $\pm 9.999 \times 10^{17}$ ），但是它工作起来较慢并缺少弹性。

本程序库将宏包 `fp` 的计算精度与 PGF 数学引擎的弹性结合了起来。使用本程序库时，注意以下几点：

- 宏包 `fp` 支持很大的数值计算，例如， 2^{20} 或 $1.2e10+3.4e10$ ，但是 PGF 和 TikZ 并不支持很大的数值，故宏包 `fp` 所计算的大数值不能直接用于绘图中，需要做一下变通。
- 长度，例如，`10pt`，`10pt+2mm`，仍然由 PGF 的数学引擎来处理，故关于长度的计算仍然不能超过 ± 16383.99999 pt。所以，表达式 `3*10000cm` 不能接受，但 `3cm*10000` 却可以接受。
- §89 中介绍了许多函数，使用本程序库后，其中多数函数都能具备 `fp` 的计算能力，但其行为可能与不使用本程序库时的行为有所不同，具体参考宏包 `fp` 的说明文档。
- 在 PGF 中，三角函数，例如，`sin`，`cos`，其参数默认为角度制下的数值；反三角函数，例如，`asin`，`acos`，其函数值也是默认为角度制下的数值。尽管宏包 `fp` 总是使用弧度制，所幸的是 PGF 会自动做好相应的转换，以维持 PGF 偏好角度制的习惯。
- 使用本程序库后，总体上说，PGF 的运行速度变慢了。为了利用宏包 `fp` 的精度，不得不牺牲一点速度。

53.2 在 PGF 和 TikZ 中使用定点算术

通过使用以下 key，可以在 PGF 和 TikZ 中使用宏包 `fp`。

```
/pgf/fixed point arithmetic=<options>
/tikz/fixed point arithmetic=<options>
```

这个选项会把 `<options>` 中的选项冠以前缀

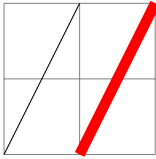
```
/pgf/fixed point
```

来执行。本选项最好用作环境选项，例如，用作环境 `{tikzpicture}`，`{pgfpicture}`，`{scope}` 的选项。当本选项用作环境选项时，就限制在该环境中使用宏包 `fp`，故在该环境之外，PGF 的数学引擎恢复到原本的行为状态，能节省一些时间。

目前，`<options>` 中能使用的选项很少，列举如下：

```
/pgf/fixed point/scale results=<factor>
```

先看一个例子：



```
\tikz[fixed point arithmetic={scale results=10^-6}]{
\draw [help lines] grid (2,2);
\draw (0,0) -- (1,2);
\draw [red, line width=4pt] (*1.0e6,0) -- (*2.0e6,*2.0e6);}
```

在上面的代码中，环境选项中用了 `fixed point arithmetic={scale results=10-6}`，表示在该环境中使用宏包 `fp`，并且设置一个比例因子 10^{-6} 。在绘图命令中用了很大的坐标数值，每个大数值都前缀一个星号“*”，这个星号会引导程序使用宏包 `fp` 来处理这个大数值，即给大数值乘上前面选项设置的比例因子 10^{-6} ，从而把大数值变通为 PGF 数学引擎能处理的“小数值”。

如果一个数值前面不带星号“*”，就不会使用宏包 `fp` 来处理该数值。

有时候会把数据点保存在一个外部文件中，绘图时调用该文件，将文件中的数据点变成可视的图形，例如，§22 中的算子 `plot[<options>]file{<file name>}`，§75 中介绍的可视化数据点格式。在编辑包含数据点的外部文件时，可以给大数值前面带上星号“*”，然后在绘图环境的选项中，或在绘图命令的选项中调用宏包 `fp` 来处理大数值。如果手工给大数值添加前缀“*”太麻烦，还可以使用下面的选项：

```
/pgf/fixed point/scale file plot x=<factor>
```

本选项将外部数据文件的第一列数值乘上比例因子 `<factor>`。

```
/pgf/fixed point/scale file plot y=<factor>
```

本选项将外部数据文件的第二列数值乘上比例因子 `<factor>`。

```
/pgf/fixed point/scale file plot z=<factor>
```

本选项将外部数据文件的第三列数值乘上比例因子 `<factor>`。

54 浮点单元程序库

首先调用程序库 `fpu`。

Floating Point Unit (`fpu`) 程序库能够为 PGF 的科学计算提供足够大的数值范围，其核心是 PGF 中处理尾数的数学程序，并在精度与速度之间实现某种平衡。本程序库不需要第三方宏包或外部程序的支持。

54.1 Overview

`fpu` 能提供足够大的数值范围以及合理的数据精度。它至少能提供 IEEE 标准下的双精度数值范围，即从 $-1 \cdot 10^{324}$ 到 $1 \cdot 10^{324}$ 。大于 0 的最小数值是 $1 \cdot 10^{-324}$ 。`fpu` 的相对精度至少是 $1 \cdot 10^{-4}$ ，而且对于像加法那样的运算，相对精度是 $1 \cdot 10^{-6}$ 。

注意本程序库还没有与绘图命令一起进行测试，`fpu` 的计算结果应当转换到通常情况下 PGF 能接受的数值范围内，即 ± 16383.99999 ， $\text{T}_{\text{E}}\text{X}$ 允许的数值范围，然后再用于绘图，PGF 可以自己做到这一点。

首先注意本程序库所使用的表达浮点数的格式。`fpu` 使用一种低层次的表数格式，该格式包括：标记、尾数、幂指数、分隔符号，例如，`1Y2.1e4`，有的命令会把这种格式的浮点数保存在 `\pgfmathresult` 中，有的命令只接受这种格式的浮点数作为其参数。可以为命令手工编写这种格式的参数。具体说明如下：

- 标记是个数字符号，标记 0 代表数值 $\pm 0 \cdot 10^0$ ；标记 1 代表正号；标记 2 代表负号；标记 3 代表“非数”；标记 4 代表 $+\infty$ ；标记 5 代表 $-\infty$ 。
- 尾数是 1 到 10 之间的实数，而且总是带有小数点，至少有一个小数数字。
- 幂指数是一个整数。
- 标记与尾数之间用一个大写字母“Y”分隔。
- 尾数之后是小写字母“e”，代表单词 exponent。
- 小写字母“e”之后是幂指数。
- 幂指数之后是右方括号“]”，表示数值表达格式的结束。

后文把 fpu 程序库使用的表达浮点数的格式简称为“浮点数格式”，提到的浮点数都是这种格式的。可以把浮点数格式的数值转换为适合于 PGF 处理的格式，fpu 提供了相应的转换办法。

54.2 用法

使用 fpu 程序库的方式一般有两种，一种方式是使用程序库的命令，这些命令在 §54.4 中列出；另一种方式是通过选项设置，将 fpu 程序库的作用植入其它程序库的命令或者绘图命令中。这里介绍第 2 种方式。

`/pgf/fpu={<boolean>}` (默认值 true)

这个选项决定是否在当前分组中启用 fpu 的功能。如果启用，那么 PGF 的那些标准的数学解析器会被换成 fpu 的形式，例如，`\pgfmathadd` 换成 `\pgfmathfloatadd`，所有的数值都会用 `\pgfmathfloatparsenumber` 来解析，解析结果仍然保存在 `\pgfmathresult` 中（不带单位的数值结果）。

1Y2.0e0]

```
\pgfkeys{/pgf/fpu}
\pgfmathparse{1+1} \pgfmathresult
```

用 `fpu=false` 取消 fpu 的作用，恢复到 PGF 原来的状态。注意 fpu 的作用受到 TeX 分组的限制，故若只是在组内使用 fpu，无需使用 `fpu=false`。

注意，如果先启用了 fpu，而后再启用定点算术程序库 fixed point arithmetics，那么 fpu 会被自动取消。

`/pgf/fpu/output format=float|sci|fixed` (无默认值，初始值 float)

这个选项设置保存在 `\pgfmathresult` 中的数值格式。

1Y2.17765411e23]

```
\pgfkeys{/pgf/fpu, /pgf/fpu/output format=float}
\pgfmathparse{exp(50)*42} \pgfmathresult
```

2.17765411e23

```
\pgfkeys{/pgf/fpu, /pgf/fpu/output format=sci}
\pgfmathparse{exp(50)*42} \pgfmathresult
```

217765411000000000000000.0

```
\pgfkeys{/pgf/fpu, /pgf/fpu/output format=fixed}
\pgfmathparse{exp(50)*42} \pgfmathresult
```

注意 `fixed` 格式会被选项 `scale results={<scale>}` 强制选定。

```
/pgf/fpu/scale results={<scale>}}
```

这个选项的用法与程序库 `fixed point arithmetics` 中选项 `/pgf/fixed point/scale results=<factor>` 的用法类似。这里 `<scale>` 是个比例因子，在表达式或绘图命令中，如果给某个数值加前缀“*”，那么在处理该数值时，就会把该数值换成它与 `<scale>` 的乘积。使用这个选项后，表达式的计算结果会被以 `fixed` 格式保存在 `\pgfmathresult` 中，这便于 PGF 对计算结果做进一步的处理。

```
/pgf/fpu/scale file plot x={<scale>}
```

```
/pgf/fpu/scale file plot y={<scale>}
```

```
/pgf/fpu/scale file plot z={<scale>}
```

以上 3 个选项与程序库 `fixed point arithmetics` 中的相应选项类似。

```
\pgflibraryfpuifactive{<true-code>}{<false-code>}
```

如果已经启用 `fpu`，则执行 `<true-code>`；如果没有启用 `fpu` 或者它已经被取消，则执行 `<false-code>`。



```
\pgflibraryfpuifactive
  {\tikz\draw circle(5mm);}
  {\tikz\fill circle(5mm);}
```

下面的例子说明选项 `fpu` 的作用受到 \TeX 分组的限制：

开启；关闭

```
{\pgfkeys{/pgf/fpu}
 \pgflibraryfpuifactive {开启}{关闭};
 \pgflibraryfpuifactive {开启}{关闭}}
```

54.3 与定点算术程序库的比较

`fpu` 至少支持 IEEE 标准下的双精度数值范围，而 `fp` 覆盖的数值范围是 $\pm 1 \cdot 10^{17}$ 。

`fpu` 有一致的相对精度，使用 4 个或 5 个纠正数字。定点算术程序库使用绝对精度，当计算过程处于数值范围的极限附近时，计算可能失败。

`fpu` 使用 PGF 的数学程序（使用 \TeX 的寄存器）来处理数值的尾数，它的运行速度有可能比 `fp` 更快。

54.4 命令与编程参考

54.4.1 浮点数的创建与转换

```
\pgfmathfloatparsenumber{<x>}
```

这个命令是 `fpu` 读取数值的主要命令。参数 `<x>` 可以是任意数量级和任意精度的数值，本命令读取 `<x>` 并做适当处理，以某种纯文本格式保存在命令 `\pgfmathresult` 中。前面提到，默认以 `float` 格式保存数值。

当手工输入参数 `<x>` 时，`<x>` 的格式可以是定点数格式，也可以是使用 `e` 或 `E` 的科学计数法格式，也可以是浮点数格式，其中的幂指数应处于 \TeX 允许的整数范围内，即不超过 31 位的整数。

标记: 1; 尾数 5.21513; 幂指数 -11.

```
\pgfmathfloatparsenumber{5.21513e-11}
\pgfmathfloattomacro{\pgfmathresult}{\F}{\M}{\E}
标记: \F; 尾数 \M; 幂指数 \E.
```

上面例子中, 命令 `\pgfmathfloatparsenumber` 读取数值并保存在 `\pgfmathresult` 中。然后用命令 `\pgfmathfloattomacro` 读取 `\pgfmathresult` 中的值, 将标记保存在宏 `\F` 中, 将尾数保存在宏 `\M` 中, 将幂指数保存在宏 `\E` 中。然后用这 3 个宏分别输出标记、尾数、幂指数。

```
/pgf/fpu/handlers/empty number={<input>}{<unreadable part>}
/pgf/fpu/handlers/invalid number={<input>}{<unreadable part>}
/pgf/fpu/handlers/wrong lowlevel format={<input>}{<unreadable part>}
\pgfmathfloatqparsenumber{<x>}
```

类似 `\pgfmathfloatparsenumber`, 只是不会执行详细的校验, 因此速度快一些。

```
\pgfmathfloattofixed{<x>}
```

这里的参数 `<x>` 是浮点数格式的数值, 本命令将 `<x>` 转换为定点数格式, 并以纯文本形式保存在 `\pgfmathresult` 中。

1Y5.2e-4]; 0.00052;

```
\pgfmathfloatparsenumber{0.00052} \pgfmathresult; \quad
\pgfmathfloattofixed{\pgfmathresult} \pgfmathresult;
```

1Y1.23456e6]; 1234560.00000000;

```
\pgfmathfloatparsenumber{123.456e4} \pgfmathresult; \quad
\pgfmathfloattofixed{\pgfmathresult} \pgfmathresult;
```

```
\pgfmathfloattoint{<x>}
```

这里的参数 `<x>` 是浮点数格式的数值, 本命令将 `<x>` 的小数部分直接去掉 (无舍入), 只保留整数部分, 并以纯文本形式保存在 `\pgfmathresult` 中。

1Y1.23456e6]; 1234560;

```
\pgfmathfloatparsenumber{123.456e4} \pgfmathresult; \quad
\pgfmathfloattoint{\pgfmathresult} \pgfmathresult;
```

```
\pgfmathfloattosci{<float>}
```

这里的参数 `<float>` 是浮点数格式的数值, 本命令将 `<float>` 变成科学计数法格式, 并以纯文本形式保存在 `\pgfmathresult` 中。

```
\pgfmathfloatvalueof{<float>}
```

```
\pgfmathfloatcreate{<flags>}{<mantissa>}{<exponent>}
```

本命令创建一个浮点数值, `<flags>` 是所要创建的浮点数的标记, `<mantissa>` 是尾数, `<exponent>` 是幂指数。所创建的浮点数保存在 `\pgfmathresult` 中。

标记: 1; 尾数 1.0; 指数 327

```
\pgfmathfloatcreate{1}{1.0}{327}
\pgfmathfloattomacro{\pgfmathresult}{\F}{\M}{\E}
标记: \F; 尾数 \M; 指数 \E
```

`\pgfmathfloatifflags{<floating point number>}{<flag>}{<true-code>}{<false-code>}`

如果浮点数 <floating point number> 的标记等于 <flag>, 则执行 <true-code>, 否则执行 <false-code>.

<flag> 有以下选择:

- 0 对应数值 0.
- 1 对应正数.
- + 对应正数.
- 2 对应负数.
- 对应负数.
- 3 对应“非数”.
- 4 对应 $+\infty$.
- 5 对应 $-\infty$.

It' s not zero!

```
\pgfmathfloatparsenumber{42}
\pgfmathfloatifflags{\pgfmathresult}{0}{It' s zero!}{It' s not zero!}
```

`\pgfmathfloattomacro{<x>}{<flags macro>}{<mantissa macro>}{<exponent macro>}`

参数 <x> 是浮点数格式的, 本命令创建 3 个宏: <flags macro>, <mantissa macro>, <exponent macro>, 这 3 个宏分别保存 <x> 的标记、尾数、幂指数。

`\pgfmathfloattoregisters{<x>}{<flags count>}{<mantissa dimen>}{<exponent count>}`

参数 <x> 是浮点数格式的, <flags count> 是已定义的整数寄存器, <mantissa dimen> 是已定义的尺寸寄存器, <exponent count> 是已定义的整数寄存器, 在本命令的处理下, 这 3 个寄存器分别保存 <x> 的标记、尾数、幂指数。注意本命令会按照 $\text{T}_{\text{E}}\text{X}$ 的精度对尾数做舍入, 然后保存在尺寸寄存器 <mantissa dimen> 中, 因为寄存器都是 $\text{T}_{\text{E}}\text{X}$ 的寄存器。

`\pgfmathfloattoregisterstok{<x>}{<flags count>}{<mantissa toks>}{<exponent count>}`

参数 <x> 是浮点数格式的, <flags count> 是已定义的整数寄存器, <mantissa toks> 是已定义的 token 寄存器, <exponent count> 是已定义的整数寄存器, 这 3 个寄存器分别保存 <x> 的标记、尾数、幂指数。与上一个命令不同, 本命令会保存 <x> 的尾数的完整精度。

`\pgfmathfloatgetflags{<x>}{<flags count>}`

参数 <x> 是浮点数格式的, <flags count> 是已定义的整数寄存器, <x> 的标记会被保存在 <flags count> 中。

`\pgfmathfloatgetflagstomacro{<x>}{<macro>}`

参数 $\langle x \rangle$ 是浮点数格式的，本命令定义宏 $\langle \text{macro} \rangle$ ，并将 $\langle x \rangle$ 的标记保存在宏 $\langle \text{macro} \rangle$ 中。

`\pgfmathfloatgetmantissa{<x>}{<mantissa dimen>}`

参数 $\langle x \rangle$ 是浮点数格式的， $\langle \text{mantissa dimen} \rangle$ 是已定义的尺寸寄存器， $\langle x \rangle$ 的尾数会被保存在 $\langle \text{mantissa dimen} \rangle$ 中。

`\pgfmathfloatgetmantissatok{<x>}{<mantissa toks>}`

参数 $\langle x \rangle$ 是浮点数格式的， $\langle \text{mantissa toks} \rangle$ 是已定义的 token 寄存器， $\langle x \rangle$ 的尾数会被保存在 $\langle \text{mantissa toks} \rangle$ 中。

`\pgfmathfloatgetexponent{<x>}{<exponent count>}`

参数 $\langle x \rangle$ 是浮点数格式的， $\langle \text{exponent count} \rangle$ 是已定义的整数寄存器， $\langle x \rangle$ 的幂指数会被保存在 $\langle \text{exponent count} \rangle$ 中。

54.4.2 符号舍入操作

程序会把数值处理成纯文本形式，对数值的舍入操作其实是对文本符号的操作，因此程序可以接受任意大的输入数值。

`\pgfmathroundto{<x>}`

按选项 `/pgf/number format/precision` (见 §92) 设置的输出精度，对 $\langle x \rangle$ 做舍入，小数部分中多余的 0 字符会被去掉，并把结果保存在 `\pgfmathresult` 中，保存结果的格式（按有关选项的设置，见 §92）是定点数格式或科学计数法格式。

仅当该命令保存在 `\pgfmathresult` 中的结果包含小数点时，该命令会把 $\text{T}_{\text{E}}\text{X}$ 条件判断命令 `\ifpgfmathfloatroundhasperiod` 的值设为 true，这个条件判断命令是全局布尔变量。

如果该命令保存在 `\pgfmathresult` 中的结果的幂指数大于 $\langle x \rangle$ ，那么该命令会把 $\text{T}_{\text{E}}\text{X}$ 条件判断命令 `\ifpgfmathfloatroundmayneedrenormalize` 的值设为 true，否则这个条件判断命令的值保持为 false。

20000

```
\pgfmathroundto{19999.9996}
\pgfmathresult
```

`\pgfmathroundtozerofill{<x>}`

按选项 `/pgf/number format/precision` (见 §92) 设置的输出精度，对 $\langle x \rangle$ 做舍入，如果小数部分的位数少于精度规定的位数，则用 0 补足，并把结果保存在 `\pgfmathresult` 中，保存结果的格式（按有关选项的设置，见 §92）是定点数格式或科学计数法格式。

20000.00

```
\pgfmathroundtozerofill{19999.9996}
\pgfmathresult
```

`\pgfmathfloatround{<x>}`

参数 $\langle x \rangle$ 时浮点数格式的，按选项 `/pgf/number format/precision`（见 §92）设置的输出精度，调用命令 `\pgfmathroundto` 对 $\langle x \rangle$ 做舍入，小数部分中多余的 0 字符会被去掉，并把结果保存在 `\pgfmathresult` 中，保存的结果仍然是浮点数格式。

仅当该命令保存在 `\pgfmathresult` 中的结果包含小数点时，该命令会把 $\text{T}_{\text{E}}\text{X}$ 条件判断命令 `\ifpgfmathfloatroundhasperiod` 的值设为 `true`，这个条件判断命令是全局布尔变量。

5.3e1	<pre>\pgfmathfloatparsenumber{52.965} \pgfmathfloatround{\pgfmathresult} \pgfmathfloattosci{\pgfmathresult} \pgfmathresult</pre>
1e0	<pre>\pgfmathfloatparsenumber{1} \pgfmathfloatround{\pgfmathresult} \pgfmathfloattosci{\pgfmathresult} \pgfmathresult</pre>

`\pgfmathfloatroundzerofill{<x>}`

类似 `\pgfmathfloatround`，不同的是，如果小数部分的位数少于精度规定的位数，则用 0 补足。

1.00e0	<pre>\pgfmathfloatparsenumber{1} \pgfmathfloatroundzerofill{\pgfmathresult} \pgfmathfloattosci{\pgfmathresult} \pgfmathresult</pre>
--------	---

54.4.3 数学运算命令

只需要载入 `fpu` 程序库，就可以使用以下命令。

`\pgfmathfloat<op>`

这是个一般的格式， $\langle op \rangle$ 是数学引擎能够接受的函数名称，参考 §89, §90, §91，例如，

```
\pgfmathfloatadd
\pgfmathfloatneg
\pgfmathfloatabs
\pgfmathfloatcos
\pgfmathfloatceil
\pgfmathfloatnotequal
\pgfmathfloatveclen
```

等等，都是数学命令的 `fpu` 版本。注意，`fpu` 版本的数学命令只接受浮点数格式的参数，它们保存在 `\pgfmathresult` 中的结果也是浮点数格式的。

```
2Y3.056789e2] \pgfmathfloatadd{1Y2.0e1]}{2Y3.256789e2]}
\pgfmathresult
```

```
1Y4.5476e-1]; 2Y9.9619e-1]; 2Y5.4143001e-1];
```

```
\pgfmathfloatsin{1Y2.705e1]} \edef\s{\pgfmathresult} \s; \quad
\pgfmathfloatcos{2Y1.75e2]} \edef\c{\pgfmathresult} \c; \quad
\pgfmathfloatadd{\s}{\c} \pgfmathresult;
```

`\pgfmathfloattoextendedprecision{<x>}`

`\pgfmathfloatsettextprecision{<shift>}`

`\pgfmathfloatlessthan{<x>}{<y>}`

参数 `<x>`, `<y>` 都是浮点数, 如果 `<x><y>`, 则本命令设置 `\pgfmathresult` 的值是 1.0, 否则设置 `\pgfmathresult` 的值是 0.0.

```
0.0 \pgfmathfloatlessthan{1Y2.1e6]}{1Y2.1e6]}
\pgfmathresult
```

`\pgfmathfloatmultiplyfixed{<float>}{<fixed>}`

`<float>` 是浮点数, `<fixed>` 是定点数, 本命令计算二者的乘积, 并把结果保存在 `\pgfmathresult` 中, 保存的结果是浮点数格式。

`\pgfmathfloatifaproxequalrel{<a>}{}{<>true-code>}{<>false-code>}`

本命令会调用命令 `\pgfmathfloatparsenumber` 来读取 `<a>`, ``, 故 `<a>`, `` 的格式不限。本命令会参考选项 `rel thresh` (见下文) 的设置来判断 `<a>` 与 `` 的近似程度。本命令计算 `<a>` 与 `` 的相对误差, $\frac{|a-b|}{|b|}$, 如果相对误差小于选项 `rel thresh` 设置的值, 则执行 `<>true-code>`, 否则执行 `<>false-code>`。

`/pgf/fpu/rel thresh={<number>}` (无默认值, 初始值 `1e-4`)

本选项为命令 `\pgfmathfloatifaproxequalrel` 设置一个阈值 (threshold)。

```
按阈值近似 \pgfmathfloatifaproxequalrel{1Y1.123e-5]}{0}
{按阈值近似} {按阈值不近似}
```

注意上面例子中, `` 是 0, 但仍然能计算。

`\pgfmathfloatshift{<x>}{<num>}`

`<x>` 是浮点数, `<num>` 是整数。本命令将 `<x> · 10<num>` 保存在 `\pgfmathresult` 中, 保存的结果是浮点数格式。

```
2Y2.0e5] \pgfmathfloatshift{2Y2.0e3]}{2}
\pgfmathresult
```

`\pgfmathfloatabserror{<x>}{<y>}`

$\langle x \rangle$, $\langle y \rangle$ 都是浮点数, 本命令计算 $\langle x \rangle$ 与 $\langle y \rangle$ 的绝对误差 $|x - y|$, 并保存在 `\pgfmathresult` 中, 保存的结果是浮点数格式。

`\pgfmathfloatreerror{<x>}{<y>}`

$\langle x \rangle$, $\langle y \rangle$ 都是浮点数, 本命令计算 $\langle x \rangle$ 与 $\langle y \rangle$ 的相对误差 $\frac{|x-y|}{|y|}$, 并保存在 `\pgfmathresult` 中, 保存的结果是浮点数格式。

`\pgfmathfloatint{<x>}`

$\langle x \rangle$ 是浮点数, 本命令将 $\langle x \rangle$ 的小数部分直接去掉 (无舍入), 将整数部分以浮点数格式保存在 `\pgfmathresult` 中。

`\pgfmathlog{<x>}`

目前, 这里的 $\langle x \rangle$ 必须是数值, 不能是表达式, $\langle x \rangle$ 的格式任意。本命令计算 $\langle x \rangle$ 的自然对数值, 等效于 `\pgfmathln`, 不同的是, 本命令将 $\langle x \rangle$ 读取为浮点数, 利用等式

$$\ln(m \cdot 10^e) = \ln(m) + e \cdot \ln(10),$$

来计算结果。 $\ln(10)$ 是个常数, 使用 PGF 的标准数学程序计算 $\ln(m)$, 这里 $1 \leq m < 10$, 最后的计算结果保存在 `\pgfmathresult` 中, 保存的格式是定点数格式或整数。

如果 $\langle x \rangle$ 不是有效的数值, 例如 $\langle x \rangle$ 不是正实数, 则 `\pgfmathresult` 的值是空值 (empty), 而且没有错误提示信息。

9.21031

```
\pgfmathlog{1Y10e3}
\pgfmathresult
```

-15.7452

```
\pgfmathlog{1.452e-7}
\pgfmathresult
```

54.4.4 用于编程的原始数学程序

55 Lindenmayer System 分形图程序库

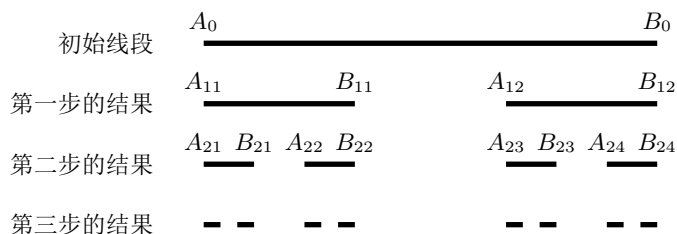
Cantor 集合

分形图有很多种, Lindenmayer System 是绘制某一类型分形图的方法。程序库 `lindenmayersystems` 提供了实现简单 Lindenmayer System 的命令。为了顺利理解本程序库的命令, 先看一下 Lindenmayer System 是什么样子。

先看一个例子——Cantor 集合。

给定一个线段 A_0B_0 , 我们对线段 A_0B_0 做以下操作: 第一步, 将 A_0B_0 做 3 等分, 去掉中间的那一段, 留下两边的线段, 得到 $A_{11}B_{11}$ 和 $A_{12}B_{12}$; 第二步, 将 $A_{1i}B_{1i}$, $i = 1, 2$ 进行 3 等分, 去掉中间的一段, 留下的线段记为 $A_{2j}B_{2j}$, $j = 1, 2, 3, 4$; 第三步, 将 $A_{2j}B_{2j}$, $j = 1, 2, 3, 4$ 进行 3 等分, 去掉中间的一段, 留下的线段记为 $A_{3k}B_{3k}$, $k = 1, 2, 3, 4, 5, 6, 7, 8, \dots$ 像这样, 每一次操作都是针对上一步的结果, 操作内容都

是将各个连续线段分别进行 3 等分，去掉中间的一段，留下两边的线段；原来线段 A_0B_0 上的很多点被去掉了，但是还有很多点保留了下来，令操作步骤达到无穷，在极限状态之下，那些保留下来的点构成的集合叫作 Cantor 集合。下面的图形表示了从初始线段到第三步操作的结果。



假设 A_0B_0 的长度是 1，那么初始线段可以这样得到：设想拿一个画笔 p，令 p 右移 1，移动时画线，这就画出了 A_0B_0 。如果令字母 F 表示“右移 1 且在移动时画线”，那么初始线段就可以表示为：F。

第一步的结果也可以这样得到：先令画笔 p 右移 $\frac{1}{3}$ ，移动时画线；再令 p 右移 $\frac{1}{3}$ ，移动时不画线；再令 p 右移 $\frac{1}{3}$ ，移动时画线，这就画出了第一步的结果。如果令字母 F 表示“右移 $\frac{1}{3}$ 且在移动时画线”，令字母 f 表示“右移 $\frac{1}{3}$ 且在移动时不画线”，那么第一步的结果就可以表示为：FfF。

类似地，如果令字母 F 表示“右移 $\frac{1}{9}$ 且在移动时画线”，令字母 f 表示“右移 $\frac{1}{9}$ 且在移动时不画线”，那么第二步的结果就可以表示为：FfFffffFfF。

如果令字母 F 表示“右移 $\frac{1}{27}$ 且在移动时画线”，令字母 f 表示“右移 $\frac{1}{27}$ 且在移动时不画线”，那么第三步的结果就可以表示为：FfFffffFfFffffffffffFfFffffFfF。

暂时忽略画笔移动的距离，看看表示各个步骤结果的字母符号之间有什么关系。事实上，表示各个步骤结果的字母符号可以通过一些列的符号替换操作得到，这里使用的替换规则只有两条：

rule1 $F \rightarrow FfF$ ，即将 F 替换为“FfF”；

rule2 $f \rightarrow fff$ ，即将 f 替换为“fff”。

这里使用 $F \rightarrow FfF$ 这样的格式表达替换规则，箭头前的 F 叫作“前任”，箭头后的 FfF 叫作“继任”。给定初始线段对应的字母 F，按照替换规则做替换，前 3 个替换步骤的结果是：

步骤	替换结果	使用的规则
初始	F	给定的初始符号
第一步	FfF	rule1
第二步	FfFffffFfF	rule1, rule2
第三步	FfFffffFfFffffffffffFfFffffFfF	rule1, rule2

以 n 代表各个步骤， $n = 0$ 代表初始状态。在第 n 步的结果中，令 F 表示“向右移动 $\frac{1}{3^n}$ 且在移动时画线”，令 f 表示“向右移动 $\frac{1}{3^n}$ 且在移动时不画线”，那么第 n 步的字母符号串恰好表达了第 n 步的图形。

我们将 F, f, 以及替换规则 rule1, rule2 看作一个体系，这就是一个简单的 Lindenmayer System. 简单地说，给定一组符号、符号替换规则，就给出了一个 Lindenmayer System，简称为“L-S”。

L-S 的分类

一、确定与随机

L-S 有多种类型。在某个 L-S 中做符号替换时，如果有随机化的处理，例如，随机地选择符号替换规则，随机地选择画笔移动距离，随机地选择画笔的旋转角度，那么这个 L-S 就是“随机的”L-S，否则就是“确定的”L-S。

二、是否含有 X, Y

L-S 也可以分为“不含 X, Y 的”与“含 X, Y 的”。这里 X, Y 是体系使用的符号。在替换符号时会用到 X 和 Y ，但是 X 和 Y 不对应任何画图动作，即它们对“画笔”的动作无影响。显然，如果一个 L-S 含有 X, Y ，那么它具有更高的灵活性和构图能力。

下面的例子是一个含 X, Y 的体系。用下面的条目规定一个 L-S:

- 变量: X, Y ;
- 常量: $F, +, -$;
- 初始符号: X ;
- 规则: $X \rightarrow X - YF, Y \rightarrow FX + Y$;
- 角度: 90° .

先解释一下上面的各个条目。在初始之下，画笔位于原点，以水平向右的方向为画笔的“前方”。 F 表示“画笔向前移动并画线”。 $+$ 和 $-$ 表示旋转画笔，旋转的幅度由条目 90° 来规定。 $+$ 表示将画笔的“前方”方向逆时针旋转 90° 。 $-$ 表示将画笔的“前方”方向顺时针旋转 90° 。符号替换规则只有两条，前三次替换是：

$$\begin{aligned} n = 0 & \quad X \\ n = 1 & \quad X - YF \\ n = 2 & \quad X - YF - FX + YF \\ n = 3 & \quad X - YF - FX + YF - FX - YF + FX + YF \end{aligned}$$

令第 n 个步骤中的 F 表示“向前移动 $\frac{1}{2^n}$ 并画线”，上面各个步骤的符号对应的图形分别是：

- $n = 0$ ，符号 X 不引起画笔动作。
- $n = 1$ ，符号 X, Y 不引起画笔动作，只有 $-$ 和 F 引起画笔动作



- $n = 2$ ，如下图



- $n = 3$ ，如下图

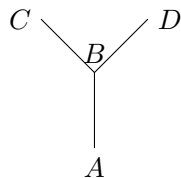


上面图形中的箭头只是为了表明画图时画笔的行动方向，不属于图形本身。

三、是否含有方括号

有的 L-S 含有方括号，即左方括号“ $[$ ”与右方括号“ $]$ ”。左方括号“ $[$ ”会使得程序将当前的画笔状态（包括画笔的位置、画笔的前方方向）压入堆栈（即“保存起来”），继续执行“ $[$ ”后的画图动作，直到遇到右方括号“ $]$ ”，将画笔恢复到左方括号“ $[$ ”所“保存”的画笔状态。

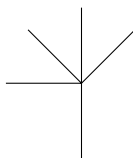
举例来说，令 F 表示画笔向前移动 1 单位长度并画线， F 的前方方向初始为“向右”；“ $+$ ”表示左传 45° ；“ $-$ ”表示右转 45° ，那么符号串 $++F[+F]-F$ 画出的图形就是：



对这个画图过程解释如下：

1. 画笔一开始处于 A 点，开头的两个“+”使得画笔的前方转到“向上”，然后画出线段 AB ；
2. 然后读取 “[”，将画笔的状态，即位置 B 和前方“向上”，压入堆栈保存起来；
3. 然后执行 “+F”，画笔左传 45° ，画出线段 BC ；
4. 然后读取 “]”，调出堆栈中保存的画笔状态，即令画笔状态恢复到——位置 B 和前方“向上”；
5. 然后执行 “-F”，画笔右传 45° ，画出线段 BD 。

方括号可以套嵌使用，例如，符号 F ， $+$ ， $-$ 的意义如前一个例子，那么符号串 $++F[[[++F]+F]F]-F$ 画出的图形就是



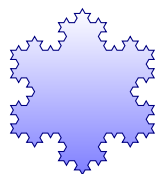
四、其它类型

L-S 还有其它类型，不过这里的程序库只涉及以上几种。

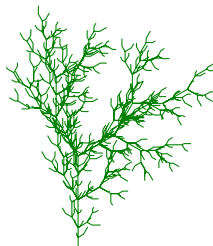
55.1 Overview

首先调用程序库 `lindenmayersystems`。

本程序库提供构造平面 L-S 的方法。受到 $\text{T}_\text{E}_\text{X}$ 内存的限制，能画出的 L-S 分形图形都不是非常复杂。下面是两个例子。



```
\begin{tikzpicture}
\pgfdeclarelindenmayersystem{Koch curve}{
  \rule{F -> F-F++F-F}
}
\shadedraw [top color=white, bottom color=blue!50, draw=blue!50!black]
  [l-system={Koch curve, step=2pt, angle=60, axiom=F++F++F, order=3}]
  lindenmayer system -- cycle;
\end{tikzpicture}
```



```
\begin{tikzpicture}
\draw [green!50!black, rotate=90]
      [l-system={rule set={F -> FF-[-F+F]+[+F-F]}, axiom=F, order=4, step=2pt,
        randomize step percent=25, angle=30, randomize angle percent=5}]
      lindenmayer system;
\end{tikzpicture}
```

55.1.1 声明一个 L-S

下面的命令声明（定义）一个 L-S:

```
\pgfdeclarelindenmayersystem{<name>}{<specification>}
```

<name> 是所定义（声明）的 L-S 的名称，<specification> 是 L-S 的定义，其中通常使用命令 `\symbol` 和 `\rule` 来定义，也就是说，定义的 L-S 一般只包括“画图符号”和“符号替换规则”，这两个命令见下文。

本命令的声明全局有效。

```
\symbol{<name>}{<code>}
```

<name> 是一个或一串由字母或者数字构成的符号，<code> 是对该 <name> 的定义，使得 <name> 能执行某种画图动作。例如

```
\symbol{A}{\pgflsystemdrawforward}
```

使得符号 A 执行命令 `\pgflsystemdrawforward`，即令画笔从当前点向前移动且在移动时画线

在程序库中，符号 F, f, +, -, [,] 是具有某种作用的预定义符号，它们的作用分别是：

- F, 执行命令 `\pgflsystemdrawforward`，令画笔从当前点向前移动且在移动时画线。
- f, 执行命令 `\pgflsystemmoveforward`，令画笔从当前点向前移动且在移动时不画线。
- +, 执行命令 `\pgflsystemturnleft`，将画笔的“前方”向左旋，即逆时针转动。
- -, 执行命令 `\pgflsystemturnright`，将画笔的“前方”向右旋，即顺时针转动。
- [, 执行命令 `\pgflsystemsavestate`，保存画笔的当前状态。
-], 执行命令 `\pgflsystemrestorestate`，调出 “[” 所保存的画笔状态。

下面的选项用来设置各个符号的代表的 <code>.

```
/pgf/lindenmayer system/step=<length>    (无默认值, 初始值 5pt)
```

这个选项设置画笔向前移动的距离。<length> 会被保存在 $\text{T}_{\text{E}}\text{X}$ 尺寸宏 `\pgflsystemstep` 中。

```
/pgf/lindenmayer system/randomize step percent=<percentage>    (无默认值, 初始值 0)
```


`<percentage>` 是百分比下的数值,从 0 到 100,该值会被保存在 T_EX 宏 `\pgflsystemrandomizesteppercent` 中。

使用该选项会引起画笔移动长度的随机化,在程序处理过程中,程序中的命令 `\pgflsystemrandomizestep` 会利用本选项设置的值。

`/pgf/lindenmayer system/left angle=<angle>` (无默认值, 初始值 90)

`<angle>` 会被保存在 T_EX 宏 `\pgflsystemrleftangle` 中。这个选项设置逆时针旋转画笔“前方”方向的角度。

`/pgf/lindenmayer system/right angle=<angle>` (无默认值, 初始值 90)

`<angle>` 会被保存在 T_EX 宏 `\pgflsystemrrrightangle` 中。这个选项设置顺时针旋转画笔“前方”方向的角度。

`/pgf/lindenmayer system/angle=<angle>`

同时设置 `left angle=<angle>` 和 `right angle=<angle>`。

`/pgf/lindenmayer system/randomize angle percent=<percentage>` (无默认值, 初始值 0)

`<percentage>` 是百分比下的数值,从 0 到 100,该值会被保存在 T_EX 宏 `\pgflsystemrandomizeanglepercent` 中。

使用该选项会引起画笔的“前方”旋转角度的随机化,在程序处理过程中,程序中的命令 `\pgflsystemrandomizelefta` 或 `\pgflsystemrandomizerightangle` 会利用本选项设置的值。

当符号执行 `<code>` 时, 下面的命令可用。

`\pgflsystemcurrentstep`

这个宏的值是画笔在当前点移动的长度, 在初始之下, 这个宏的值就是 T_EX 尺寸宏 `\pgflsystemstep` 的值, 即由选项 `step=<length>` 设定的尺寸, 该尺寸的初始值是 5pt。

如果使用了命令 `\pgflsystemrandomizestep`, 那么画笔移动的长度会被随机化, 即这个宏的值是随机的。

`\pgflsystemcurrentleftangle`

这个宏的值是画笔的“前方”方向在当前点逆时针旋转的角度, 在初始之下, 这个宏的值就是 T_EX 宏 `\pgflsystemrleftangle` 的值, 即由选项 `left angle=<angle>` 设定的角度, 该角度的初始值是 90。

如果使用了命令 `\pgflsystemrandomizeleftangle`, 那么画笔的“前方”方向逆时针旋转的角度会被随机化, 即这个宏的值是随机的。

`\pgflsystemcurrentrightangle`

这个宏的值是画笔的“前方”方向在当前点顺时针旋转的角度, 在初始之下, 这个宏的值就是 T_EX 宏 `\pgflsystemrrrightangle` 的值, 即由选项 `right angle=<angle>` 设定的角度, 该角度的初始值是 90。

如果使用了命令 `\pgflsystemrandomizerightangle`, 那么画笔的“前方”方向顺时针旋转的角度会被随机化, 即这个宏的值是随机的。

当自定义符号时, 下面的命令可能会用得上。

\pgflsystemrandomizestep

本命令会参考选项 `randomize step percent=<percentage>` 设置的数值，将保存在 `\pgflsystemcurrentstep` 中的值随机化。

\pgflsystemrandomizeleftangle

本命令会参考选项 `randomize angle percent=<percentage>` 设置的数值，将保存在 `\pgflsystemcurrentleftangle` 中的值随机化。

\pgflsystemrandomizerightangle

本命令会参考选项 `randomize angle percent=<percentage>` 设置的数值，将保存在 `\pgflsystemcurrentrightangle` 中的值随机化。

\pgflsystemdrawforward

本命令将画笔从当前点向前移动且在移动时画线，移动距离由 `\pgflsystemcurrentstep` 规定，本命令会调用 `\pgflsystemrandomizestep`。

\pgflsystemmoveforward

本命令将画笔从当前点向前移动且在移动时不画线，移动距离由 `\pgflsystemcurrentstep` 规定，本命令会调用 `\pgflsystemrandomizestep`。

\pgflsystemturnleft

本命令将画笔的前方方向逆时针旋转，旋转角度由 `\pgflsystemcurrentleftangle` 规定，本命令会调用 `\pgflsystemrandomizeleftangle`。

\pgflsystemturnright

本命令将画笔的前方方向顺时针旋转，旋转角度由 `\pgflsystemcurrentrightangle` 规定，本命令会调用 `\pgflsystemrandomizerightangle`。

\pgflsystemsavestate

将画笔的当前状态压入堆栈。

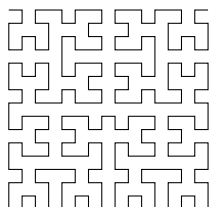
\pgflsystemrestorestate

从堆栈调出画笔状态。

\rule{<head>-><body>}

这个命令声明符号替换规则。

下面的例子中，命令 `\rule` 的声明中使用的符号 A, B 未经命令 `\symbol` 定义，它们不影响画笔动作，只用于符号替换。故所定义的 L-S 属于“含 X, Y”的类型。



```

\pgfdeclarelindenmayersystem{Hilbert curve}{
  \symbol{X}{\pgflsystemdrawforward} % 画线符号
  \symbol{+}{\pgflsystemturnright} % 顺时针转
  \symbol{-}{\pgflsystemturnleft} % 逆时针转
  \rule{A -> +BX-AXA-XB+} % 替换规则, A 和 B 不影响画笔
  \rule{B -> -AX+BXB+XA-} % 替换规则
}
\tikz\draw[lindenmayer system={Hilbert curve, axiom=A, order=4, angle=90}] % 初始符号是 A
  lindenmayer system;

```

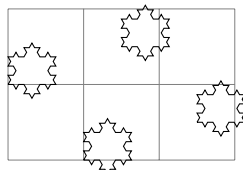
55.2 使用 L-S

55.2.1 在 PGF 中使用 L-S

`\pgflindenmayersystem{<name>}{<axiom>}{<order>}`

<name> 是已经定义（声明）的 L-S，前面的命令 `\pgfdeclarelindenmayersystem` 可以定义（声明）一个 L-S。选项 <axiom> 规定初始符号。选项 <order> 规定符号替换的次数。

本命令将按照 <name> 的定义，计算第 <order> 次符号替换的结果，得到一串符号，并将这串符号变成图形。



```

\begin{tikzpicture}
  \draw [help lines] grid (3,2);
  \pgfset{lindenmayer system/.cd, angle=60, step=2pt}
  \foreach \x/\y in {0cm/1cm, 1.5cm/1.5cm, 2.5cm/0.5cm, 1cm/0cm}{
    \pgftransformshift{\pgfqpoint{\x}{\y}}
    \pgfpathmoveto{\pgfpointorigin}
    \pgflindenmayersystem{Koch curve}{F++F++F}{2}
    \pgfusepath{stroke}
  }
\end{tikzpicture}

```

55.2.2 在 TikZ 中使用 L-S

在 TikZ 中使用 L-S 的句法比较灵活，可以使用路径命令画出先前定义的 L-S，也可以在路径命令中用选项临时定义一个 L-S 并画出其图形。

`\path . . . lindenmayer system [<keys>] . . . ;`

这个路径句中，算子 `lindenmayer system` 会调用本程序库。其中 `<keys>` 可以是本程序库提供的选项，也可以是 TikZ 的选项（如 `draw`, `color`, `line width`）。这个句法与下面的句法是等效的：

```
\draw lindenmayer system [lindenmayer system={Hilbert curve, axiom=4, order=3}];
\draw [lindenmayer system={Hilbert curve, axiom=4, order=3}] lindenmayer system;
\tikzset{lindenmayer system={Hilbert curve, axiom=4, order=3}}
\draw lindenmayer system;
```

```
\path . . . l-system [<keys>] . . . ;
```

这个句法与上一个句法等效。

本程序库还提供了下面的选项，它们只能用在 TikZ 中。它们的路径都是 `/pgf/lindenmayer system`。

```
/pgf/lindenmayer system={<keys>}
```

```
/tikz/lindenmayer system={<keys>}
```

这个选项用于临时定义一个 L-S，定义内容由 `<keys>` 指定，`<keys>` 中使用下文的选项。`<keys>` 中的选项的路径会被修改为 `/pgf/lindenmayer systems` 并执行。

```
/pgf/l-system={<keys>}
```

```
/tikz/l-system={<keys>}
```

与上一个选项等效。

```
/pgf/lindenmayer system/name={<name>}
```

`<name>` 是先前已经定义的 L-S 的名称，用 `<name>` 调用这个 L-S 的定义，画出其图形。

```
/pgf/lindenmayer system/axiom={<string>}
```

设置 L-S 的初始符号串。

```
/pgf/lindenmayer system/order={<integer>}
```

指定符号替换的次数，获得第 `<integer>` 次符号替换的结果——一串符号，程序将这串符号转换成图形。

```
/pgf/lindenmayer system/rule set={<list>}
```

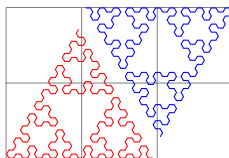
指定符号替换规则，各规则之间用逗号分隔，每个规则都使用“`<前任> -> <继任>`”的格式。

```
/pgf/lindenmayer system/anchor=<anchor>
```

如果不使用这个选项，当启用 L-S 画笔开始画分形图时，画笔会从当前点开始画起，“当前点”就是启用画笔时程序处理过程恰好达到的点。

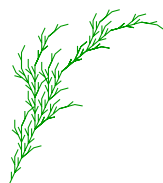
使用这个选项后，所画的分形图会被放入一个矩形 node 中，当前点就成为这个 node 的指向点，程序会把该 node 的 `<anchor>` 位置放在这个点上。

下面的例子中，先定义一个 L-S，然后在路径命令中调用这个定义，画出其图形。



```
\begin{tikzpicture}[l-system={step=1.75pt, order=5, angle=60}]
  \pgfdeclarelindenmayersystem{Sierpinski triangle}{
    \symbol{X}{\pgfsystemdrawforward}
    \symbol{Y}{\pgfsystemdrawforward}
    \rule{X -> Y-X-Y}
    \rule{Y -> X+Y+X}
  }
  \draw [help lines] grid (3,2);
  \draw [red] (0,0) l-system [l-system={Sierpinski triangle, axiom=+++X, anchor=south west}]; % 锚位置 south west 在 (0,0) 点上
  \draw [blue] (3,2) l-system [l-system={Sierpinski triangle, axiom=X, anchor=north east}]; % 锚位置 north east 在 (3,2) 点上
\end{tikzpicture}
```

下面是一个用选项定义 L-S 并画出其图形的例子。



```
\tikz[rotate=65]\draw [green!60!black] l-system
  [l-system={rule set={F -> F[+F]F[-F]}, axiom=F, order=4,
    angle=25,step=3pt}];
```

56 数学程序库

调用程序库 math.

这个程序库定义了一种简单的数学语言，能执行一些基本的数学运算。

56.1 Overview

PGF 的数学引擎用起来有点繁琐，尤其是在为多个变量赋值时。TikZ 的 calc 程序库也能执行计算，但一般情况下 calc 只用于 TikZ 的绘图命令。当调用程序库 math 以及 calc 后，可以在程序库 math 的语句中使用 calc 的句法。程序库 math 提供赋值语句、循环语句、条件语句、自定义函数语句、输出语句等句法来实现相应的运算。

程序库 math 使用 PGF 的数学引擎来完成各种赋值、解析、计算工作，受限于 $\text{T}_\text{E}_\text{X}$ 的计算能力，程序库 math 的计算精度有限。可以用程序库 fp 或 fpu 提高计算精度。

本程序库提供命令 `\tikzmath` 和选项 `/tikz/evaluate` 来使用本程序库的功能。

```
\tikzmath{<statements>}
```

<statements> 是程序库提供的各种语句，注意

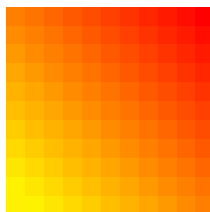
- 每个语句都要以分号 “;” 结束，否则报错。
- <statements> 中不能有空行，否则报错。

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765,

```
\tikzmath{
  function fibonacci(\n) { % 定义函数 fibonacci 并以 \n 为参数
    if \n == 0 then { % 使用条件语句
      return 0;
    }
    else {
      return fibonacci2(\n, 0, 1);
    }; % 条件语句结束
  }; % 函数定义结束
  function fibonacci2(\n, \p, \q) { % 定义函数 fibonacci2 并以 \n, \p, \q 为参数
    if \n == 1 then { % 使用条件语句
      return \q;
    }
    else {
      return fibonacci2(\n-1, \q, \p+\q);
    }; % 条件语句结束
  }; % 函数定义结束
  int \f, \i; % 声明整数变量
  for \i in {0,1,...,20}{ % 使用循环语句
    \f = fibonacci(\i);
    print {\f, }; % 输出计算结果
  }; % 循环语句结束
}
```

`/tikz/evaluate=<statements>`

相当于执行 `\tikzmath{<statements>}`。



```
\tikz[x=0.25cm,y=0.25cm,
  evaluate={
    int \i, \j;
    for \i in {0,...,10}{
```

```

        for \j in {0,...,10}{
            \a{\i,\j} = (\i+\j)*5;
        };
    };
}
]
\foreach \i in {0,...,10}
    \foreach \j in {0,...,10}
        \fill [red!\a{\i,\j}!yellow] (\i,\j) rectangle ++(1, 1);

```

程序库 math 提供的语句中, 有一些关键词 (keyword), 如上面例子中的 `function`, `if`, `return`, `int`, 注意用空格把关键词分隔起来, 否则程序不能识别关键词。

56.2 赋值语句

本程序库提供两种赋值方式。第一种是使用等号 “=” 赋值:

`\<macro> = <expression>;`

数学引擎会解析 `<expression>` 的值, 得到最终的计算结果, 并将结果赋予宏 `\<macro>`, 例如

26.0, 2.0, 11, 225.0pt

```

\newcount\mycount
\newdimen\mydimen
\tikzmath{
    \a = 4*5+6;
    \b = sin(30)*4;
    \mycount = log10(2048) / log10(2);
    \mydimen = 15^2;
}
\a, \b, \the\mycount, \the\mydimen

```

在数学中经常有 $x_1 = 0$, $x_2 = 1$, 这种“字母”缀“标号”的写法, 本程序库也提供类似的赋值格式, 即给 $\text{T}_{\text{E}}\text{X}$ 宏 (不是 $\text{T}_{\text{E}}\text{X}$ 寄存器) 加后缀, 例如加数字后缀:

7.0, 70.0, 700.0

```

\tikzmath{
    \x1 = 3+4; \x2 = 30+40; \x3 = 300+400;
}
\x1, \x2, \x3

```

也可以加文字后缀, 不过需要把后缀文字用花括号括起来:

340, 1435, 6100

```

\tikzmath{
    \c{air} = 340; \c{water} = 1435; \c{steel} = 6100;
}
\c{air}, \c{water}, \c{steel}

```

给变量宏加后缀时，程序会有一些特殊的操作，看下面的例子：

7.0, 70.0,

```
\tikzmath{
  \x = 5; \x1 = 3+4; \x2 = 30+40;
}
\x, \x1, \x2,
```

在这个例子中，宏 \x 并没有值，这是因为带后缀的宏 $\x1$ 和 $\x2$ 与前面不带后缀的宏 \x 的名称的“主要部分”相同，后面的宏清取消了 \x 的原来定义，使之没有任何作用。对比下面的例子：

5, 7.0, 70.0,

```
\tikzmath{
  \y=5; \x1 = 3+4; \x2 = 30+40;
}
\y, \x1, \x2,
```

反过来也有类似的效果：

51, 52, 5,

```
\tikzmath{
  \x1 = 3+4; \x2 = 30+40; \x=5;
}
\x1, \x2, \x,
```

显然并没有输出所需要的 $\x1$ 和 $\x2$ 的值，输出的只是在数字“5”后面缀上数字“1”或“2”。这是因为带后缀的宏 $\x1$ 和 $\x2$ 与后面不带后缀的宏 \x 的名称的“主要部分”相同，后面的宏取消了前面宏的原来定义，使之没有任何作用。

另一种赋值方式是使用关键词 `let`：

let <variable> = <expression>;

<variable> 是 \TeX 宏，在赋值时不解析 <expression>，而且 <expression> 会被用 `\edef` 展开，然后以纯文本形式保存在 <variable> 中。<expression> 前面的空格会被忽略，后面（直到分号之前）的空格会被认为是 <expression> 的一部分，一并赋予 <variable>。

$(5*4)+1-2$, “blue”

```
\tikzmath{
  let \x = (5*4)+1-2;
  let \c1 = blue;
}
\x, “\c1”
```

从上面的例子看出，所做的赋值在命令 `\tikzmath` 之外也是有效的。

56.3 声明变量类型

在默认下，数学引擎解析表达式，解析结果会带有小数点，而且小数部分至少有一个数字，然后将结果赋予变量。本程序库支持 3 种变量类型：整数、实数、坐标。


```
integer <variable>;
integer <variable1>, <variable2>, ...;
```

这里 <variable>, <variable1>, <variable2>……都是 T_EX 宏, 将它们声明为整数变量, 当把某个数值 (或者表达式的解析结果) 赋予它们时, 把小数部分 (包括小数点) 直接去掉 (不做舍入), 只保留数值的整数部分。

$x = 26, y = 2, z = 9$

```
\tikzmath{
  integer \x, \y, \z;
  \x = 4*5+6;
  \y = sin(30)*4;
  \z = log10(512) / log10(2);
  print {$x=\x$, $y=\y$, $z=\z$};
}
```

可以给整数变量宏加后缀来表示不同的整数变量, 采用如下办法:

7, 70, 700

```
\tikzmath{
  integer \x;
  \x1 = 3+4; \x2 = 30+40; \x3 = 300+400;
}
\x1, \x2, \x3
```

即先声明一个宏, 然后这个宏带上不同的后缀, 表示不同的整数变量。

```
int <variable>;
int <variable1>, <variable2>, ...;
```

等效于关键词 `integer`.

```
real <variable>;
real <variable1>, <variable2>, ...;
```

这里 <variable>, <variable1>, <variable2>……都是 T_EX 宏, 将它们声明为实数变量, 它们保存的值都带有小数点, 小数部分至少有一个数字。

给实数变量宏加后缀的办法如同整数变量宏。

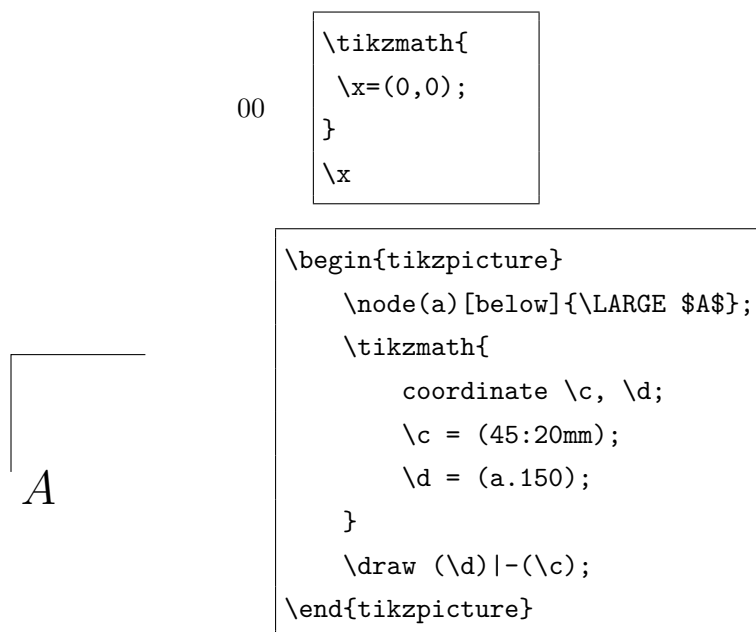
```
coordinate <variable>;
coordinate <variable1>, <variable2>, ...;
```

这里 <variable>, <variable1>, <variable2>……都是 T_EX 宏, 将它们声明为坐标变量。可以将数值构成的坐标, 如 (2cm,3pt), (25:3mm), 或 node 的各种“部位”, 如 (my node.60), (my node.east), 等等, 赋予 <variable>.

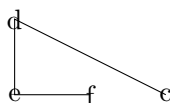
当使用坐标变量 <variable> 绘图时, 注意给 <variable> 加上圆括号。

如果需要把某个坐标保存在一个宏中, 应当先使用关键词 `coordinate` 声明坐标变量, 否则程序不能识

别坐标赋值，观察下面的例子：



如果还调用了程序库 `calc`，那么在为坐标变量赋值时，还能使用 §13.5 中介绍的各种坐标计算句法，有的还可以省略 `$`。



```

\tikzmath{
  coordinate \c, \d, \e, \f;
  \c = (-1,2)+(1,-1);
  \d = (\c)-(2,-1);
  \e = (\c -| \d);
  \f = ($(\c)!0.5!(\e)$);
}
\tikz \draw (\c) node {c} -- (\d) node {d} -- (\e) node {e} -- (\f) node {f};

```

当使用关键词 `coordinate` 声明坐标变量时，例如 `coordinate \c`；程序不仅会定义 $\text{T}_{\text{E}}\text{X}$ 宏 `\c`，而且还会定义另外两个宏：`\cx` 和 `\cy`，前者保存 `\c` 的 x 分量，后者保存 `\c` 的 y 分量。

-56.90549pt, 28.45273,

```

\tikzmath{
  coordinate \c, \d, \e, \f;
  \c = (-1,2)+(1,-1);
  \d = (\c)-(2,-1);
  \e = (\c -| \d);


```

```

\ f = (\ c)!0.5!(\ e$);
}
\ ex, % 输出坐标 \ e 的 x 分量, 带单位
\ pgfmathparse{scalar(\ fy)} \ pgfmathresult, % 输出坐标 \ f 的 y 分量, 不带单位

```

给坐标变量宏加后缀的办法如同整数变量宏, 看下面的例子:



```

\tikzmath{
coordinate \ c;
\ c1 = (30:20pt);
\ c2 = (210:20pt);
}
\tikz\draw (\ cx1,\ cy1) -- (\ cx2,\ cy1) -- (\ cx2,\ cy2) -- (\ cx1,\ cy2);

```

在这个例子中需要注意的是, 坐标 $\ c1$ 的 x 分量保存在宏 $\ cx1$ 中, 而不是 $\ c1x$ (程序没有定义这个宏)。可以这样比方: $\ c$ 是个坐标点集合, 带后缀的 $\ c1$ 是用数字“1”来索引集合 $\ c$; 各个坐标点的 x 分量组成一个集合 $\ cx$, 带后缀的 $\ cx1$ 是用数字“1”来索引集合 $\ cx$ 。

56.4 循环语句

```
for <variable> in {<list>}{<expressions>;}
```

这个语句是 `\foreach` 命令的删减版。

关于这个语句需要注意:

- $\ <list>$ 是用逗号分隔的列表, 其中的条目会被数学引擎解析。如果其中某个条目本身含有逗号, 应当用花括号把该条目括起来, 例如, $\ \{\text{mod}(5,2)\}$ 。

$x = 5, v = 32$

```

\tikzmath{
int \ x, \ v;
\ v=1;
for \ x in {1,...,\ random(3,10)}{
\ v=\ v*2;
};
print {\ $x=\ x, v=\ v$};
}

```

- 因为 $\ <list>$ 中的条目用数学引擎来解析, 故条目中不能使用 TikZ 的坐标。
- 目前关键词 `for` 之后只能使用一个变量。
- 可以在 $\ <list>$ 中使用省略号“...”, 但注意是本程序库对省略号的处理没有 PGF 中的 `\foreach` 命令那么智能。
- 在循环体内所做的“最终赋值”的有效范围超出循环体, 故可在循环体外利用其中的“最终赋值”做计算。

$$x_1 = 5, x_2 = 50, y = 2250$$

```
\tikzmath{
  int \x, \y;
  \y = 0;
  for \x1 in {1,...,5}{
    for \x2 in {10,20,...,50}{
      \y = \y+\x1*\x2;
    };
  };
}
$x_1=\x1, x_2=\x2, y=\y$
```

56.5 条件语句

if <condition> then {<if-non-zero-statements>;}

仅当 <condition> 的执行结果非 0 时，执行 <if-non-zero-statements>.

if <condition> then {<if-non-zero-statements>} else {<if-zero-statements>;}

若 <condition> 的执行结果非 0 时，则执行 <if-non-zero-statements>，否则执行 <if-zero-statements>.



```
\begin{tikzpicture}
\tikzmath{
  int \x;
  for \k in {0,10,...,350}{
    if \k>260 then { let \c = orange; } else {
      if \k>170 then { let \c = blue; } else {
        if \k>80 then { let \c = red; } else {
          let \c = green; }; }; }; % 结束条件语句
    % 下面是 for 语句中的非 0 语句，因为它是个绘图命令，不是本程序库的语句，所以要用花括号把它括起来，且在闭花括号后加分号
    {
      \path [fill=\c!50, draw=\c] (\k:0.5cm) -- (\k:1cm) -- (\k+5:1cm) -- (\k+5:0.5cm)
        -- cycle;
    }; % 注意这里加分号
  }; % 结束 for 语句
}
\end{tikzpicture}
```

56.6 声明函数

function <name> (<arguments>) {<definition> };

这个关键词的用法与 PGF 数学引擎中的选项 `declare function` 类似。

<name> 是当前环境中没有用过的函数名称。

<arguments> 是函数的变量参数列表，各个参数采用 \TeX 宏的形式，之间用逗号分隔，目前不接受省略号（可变个数的参数形式）。如果函数没有变量参数，可以省去圆括号。目前，PGF 数学引擎中的“数组”不能作为这里的函数的参数。

<definition> 应当是 `\tikzmath` 能够处理的语句，其中的变量不能超出 <arguments> 所列举的变量。<definition> 中可以使用关键词 `return`（见下文）。在 <definition> 中尽量不要在一个函数定义中定义新的函数，即不要在一个 `function` 的辖域内再使用 `function`。

$$6 \times 39 = 234$$

```
\tikzmath{
  function product(\x,\y) {
    return \x*\y;
  };
  int \i, \j, \k;
  \i = random(1,10);
  \j = random(20, 40);
  \k = product(\i, \j);
  print { $\i\times \j = \k$ };
}
```

return <expression>;

这个关键词用于函数声明中的定义中，<expression> 就是定义函数的表达式。

56.7 在命令 `\tikzmath` 的辖域内执行代码

要想在命令 `\tikzmath` 的辖域内执行某些代码，例如，输出代码，绘图代码等，需要相应的措施。

如果是要输出代码，则使用关键词 `print`：

print {<code>;}

<code> 会被放入一个 \TeX 分组内来处理，<code> 可以是任何 \TeX 代码，其中可以使用本程序库的赋值结果、运算结果。

$$4^0 = 1, 4^1 = 4, 4^2 = 16, 4^3 = 64, 4^4 = 256, 4^5 = 1024, 4^6 = 4096,$$

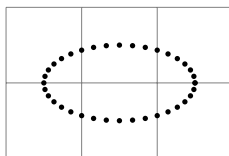
```
\tikzmath{
  int \x, \y, \z;
  \x = random(2, 5);
  for \y in {0,...,6}{
```

```

    \z = \x^\y;
    print {\$ \x^\y=\z$, };
};
}

```

如果要执行某些命令，则需要将执行命令用花括号括起来，且闭花括号之后要加上分号。花括号里的内容会被放入一个 $\text{T}_{\text{E}}\text{X}$ 分组内来处理。

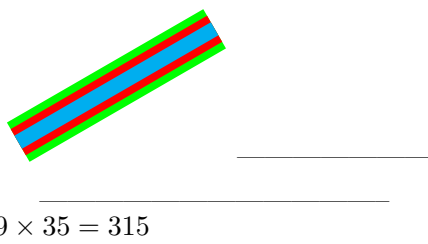


```

\begin{tikzpicture}
\draw [help lines] grid (3,2);
\tikzmath{
  coordinate \c;
  for \x in {0,10,...,360}{
    \c = (1.5cm, 1cm) + (\x:1cm and 0.5cm);
    { \fill (\c) circle [radius=1pt]; };
  };
}
\end{tikzpicture}

```

用这种方式，几乎可以在 \tikzmath 的辖域内的任何位置插入一个命令：



```

\begin{tikzpicture}
\tikzmath{
  function product(\x,\y) {
    return \x*\y;
    {\draw [green,line width=6mm](\i pt,\j pt)---+(30:3)};};
    {\draw [red,line width=4mm](\x pt,\y pt)---+(30:3)};};
};
int \i, \i, \k;
\i = random(1,10);
\j = random(20, 40);

```

```

\k = product(\i, \j);
print { $\i\times \j = \k$};
{\draw [cyan,line width=2mm](\i pt,\j pt)--+(30:3)};
}
\end{tikzpicture}-----\\
-----\\
-----\\

```

注意在上面的例子中，由于是在 `{tikzpicture}` 环境中使用命令 `\tikzmath`，所以插入在 `\tikzmath` 的辖域内的命令是绘图命令，这些绘图命令使用同一个坐标系。其中关键词 `print` 输出一串数学模式下的符号，但是这一串符号位于基线以下。

65 shadings 程序库

首先调用程序库 `\usepgflibrary{shadings}`。

这个程序库定义了数种颜色渐变模式，其中的 `axis`, `ball`, `radial` 这三种模式在不调用本程序库时也可以直接使用。

关于颜色渐变的详细介绍参考 §109。

以下选项可以作为绘图命令或者绘图环境的选项。

shade 启用颜色渐变效果

shading=<name> 设置颜色渐变的模式

shading angle=<degrees> 初始值 0，设置颜色渐变的方向

shading=axis

这个选项设置轴向（横向或纵向）颜色渐变。纵向渐变应当设置上、下、中三个部位的颜色，或者设置上、下两个部位的颜色，此时中间的颜色自动设为上部颜色与下部颜色的“中间色”。横向渐变应当设置左、右、中三个部位的颜色，或者设置左、右两个部位的颜色，此时中间的颜色自动设为左侧颜色与右侧颜色的“中间色”。

渐变方向由下面的选项确定。

/tikz/top color=<color>

这个选项设置垂直方向渐变的上部颜色。当使用这个选项时，系统会有以下反应：

1. 选定 `shade` 选项。
2. 选定 `shading=axis` 选项。
3. 中间颜色首先会被设定为上、下颜色的中间颜色。
4. 渐变的旋转角度设为 0 度。



```
\tikz \draw[top color=red] (0,0) rectangle (2,1);
```

/tikz/bottom color=<color>

这个选项设置纵向渐变的下部颜色，作用与选项 `top color` 类似。

`/tikz/middle color=<color>`

这个选项设置纵向渐变或者横向渐变的中部颜色，作用与选项 `top color` 类似。

注意，由于选项 `top color`, `bottom color` 会重设中间颜色，所以选项 `middle color` 应该用在这两个选项之后。



```
\tikz \draw[top color=white,bottom color=black,middle color=red]
(0,0) rectangle (2,1);
```

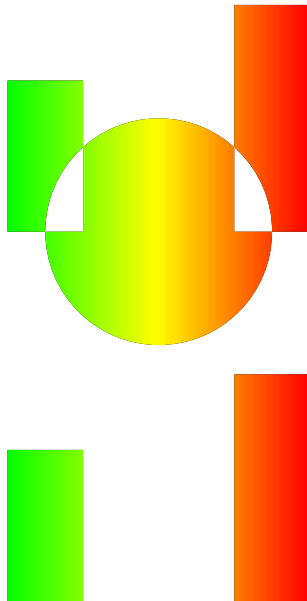
`/tikz/left color=<color>`

这个选项设置横向渐变的左侧颜色，作用与 `top color` 类似。

`/tikz/right color=<color>`

这个选项设置横向渐变的右侧颜色，作用与 `top color` 类似。

当用颜色渐变填充路径时，在整个路径边界盒子内都充满渐变效果。下面图形中的空白是由于判断区域内外部的规则所导致的，见《手册》172 页。



```
\tikz \fill [left color=green,right color=red,
middle color=yellow]
(0,0)rectangle(1,2)
(2,0)circle(1.5cm)
(3,0)rectangle(4,3);
```

```
\tikz \fill [left color=green,right color=red,
middle color=yellow]
(0,0)rectangle(1,2)
(3,0)rectangle(4,3);
```

`shading=ball`

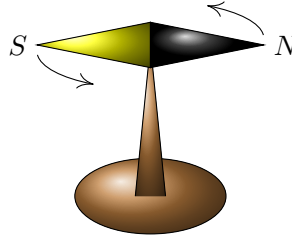
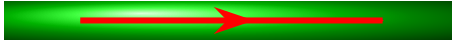
本模式参照素描中物体的光影色调变化原理来变化颜色，得到立体视觉效果，可以用于填充各种路径（不只可以填充圆形），渐变颜色默认用蓝色，可以用下面选项更换渐变颜色。

`/tikz/ball color=<color>`

这个选项设置球形渐变的颜色，也会自动设置 `shade` 和 `shading=ball` 选项。



```
\begin{tikzpicture}
  \shade[ball color=white] (0,0) circle (2ex);
  \shade[ball color=red] (1,0) circle (2ex);
  \shade[ball color=black] (2,0) circle (2ex);
\end{tikzpicture}
```



```
\begin{tikzpicture}
  \fill [ball color=green] (-3,0) -- ++ (6,0) -- ++(0,0.5) -- ++(-6,0) -- cycle;
  \filldraw [ball color=brown] (0,-3) ellipse (1 and 0.5);
  \filldraw [ball color=brown] (-0.2,-3) -- (0,-1) -- (0.2,-3) (-0.2,-3);
  \filldraw [ball color=yellow ]
    (-1.5,-1) node [left] {$S$} -- (0,-0.7) -- (0,-1.3) -- cycle;
  \filldraw [ball color=black]
    (1.5,-1) node [right] {$N$} -- (0,-0.7) -- (0,-1.3) -- cycle;
  \draw [line width=0.08cm,red, -{Stealth[width=10pt,length=15pt,sep=1.7cm]}
    Stealth[width=0.001pt 0 0 ,length=0.001pt 0 0]]
    (-2,0.25) -- (2,0.25);
  \draw [-{>[bend, width=6pt,length=6pt]}] (0,-1) ++(5:1.5) arc (5:60:1.5 and 0.5);
  \draw [-{>[bend, width=6pt,length=6pt]}] (0,-1) ++(185:1.5) arc (185:240:1.5 and
    0.5);
\end{tikzpicture}
```

shading=bilinear interpolation

双线性插值。本模式通过指定矩形四个角的颜色，在该矩形内产生渐变效果。四个角的名称是 `lower left`, `lower right`, `upper left`, `upper right`，这四个名称也是四个选项，修改这四个选项的颜色可以改变渐变的颜色。

`/tikz/lower left=<color>` (无默认值, 初始值 `white`)

这个选项设置左下角的颜色，也会自动设置 `shade` 和 `shading=bilinear interpolation` 选项。

`/tikz/upper left=<color>` (无默认值, 初始值 `white`)

类似上一选项。

`/tikz/lower right=<color>` (无默认值, 初始值 `white`)

类似上一选项。

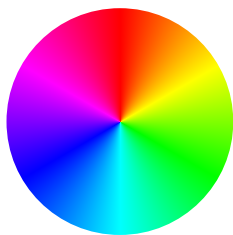
`/tikz/upper right=<color>` (无默认值, 初始值 `white`)

类似上一选项。



```
\tikz\shade[upper left=red,upper right=green,
lower left=blue,lower right=yellow]
(0,0) rectangle (3,2);
```

`shading=color wheel`



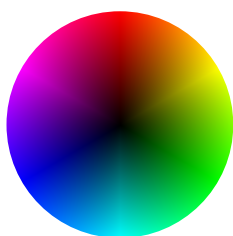
```
\tikz \shade[shading=color wheel] (0,0) circle (1.5);
```

下面的例子中使用了 `even odd rule` 规则 (见手册 172 页) 来填充颜色。



```
\tikz \shade[shading=color wheel] [even odd rule]
(0,0) circle (1.5)
(0,0) circle (1);
```

`shading=color wheel black center`



```
\tikz \shade[shading=color wheel black center]
(0,0) circle (1.5);
```

`shading=color wheel white center`



```
\tikz \shade[shading=color wheel white center]
(0,0) circle (1.5);
```

`shading=Mandelbrot set`

此模式会产生一个 Mandelbrot 分形集, 是由 PDF 渲染器计算出来的, 可以任意放缩, 不是 bit 图。



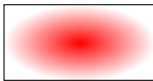
```
\tikz \shade[shading=Mandelbrot set] (0,0) rectangle (2,2);
```

shading=radial

本选项确定辐射渐变模式，辐射中心在被填充路径的边界盒子的中心，在默认下，内部中心颜色是灰色，外部边界颜色是白色。可以用下面的选项修改渐变颜色。

```
/tikz/inner color=<color>
```

这个选项设置辐射渐变的内部中心的颜色，也会自动设置 `shade` 和 `shading=radial` 选项。



```
\tikz \draw[inner color=red] (0,0) rectangle (2,1);
```

```
/tikz/outer color=<color>
```

这个选项设置辐射渐变的外部边界的颜色，也会自动设置 `shade` 和 `shading=radial` 选项。



```
\tikz \draw[outer color=red,inner color=white]
(0,0) rectangle (2,1);
```

66 shadows 程序库

调用程序库 `\usepgflibrary{shadows}`.

本程序库可以为路径或者 `node` 添加透明阴影。

66.1 Overview

阴影 (shadow) 通常是黑色或者灰色的，并且相对于路径有一定的位置偏移或者尺寸放缩。本程序库提供一些选项来实现阴影效果，实际上这是使用选项 `preaction`，两次利用路径的结果，第一用路径画阴影，第二次画出路径，阴影相对于路径有一定偏移。

注意只能针对“路径”使用本程序库，不能针对整个图形画阴影。

66.2 一般的阴影选项

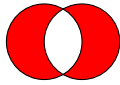
```
/tikz/general shadow=<shadow options>
```

这个选项只能作为路径或者 `node` 的选项。本选项的作用是，先执行 `<shadow options>`，利用路径完成阴影，然后再画出路径。这个选项实际上使用选项 `preaction` 来工作。

在 `<shadow options>` 中可以使用以下选项。

```
fill=<color>
```

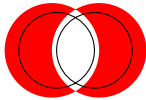
其中 `<color>` 可以是颜色表达式。这个选项会导致对阴影的填充，填充颜色是 `<color>`。如果不用这个选项指定填充色，那么就默认不填充颜色，也就没有阴影效果。



```
\tikz [even odd rule]
\draw [general shadow={fill=red}]
(0,0) circle (.5) (0.5,0) circle (.5);
```

`/tikz/shadow scale=<factor>` (无默认值, 初始值 1)

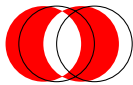
本选项设置阴影相对于原路径的尺寸比例。注意 PGF 用画布变换实现此选项的作用。



```
\tikz [even odd rule]
\draw [general shadow={fill=red,shadow scale=1.25}]
(0,0) circle (.5) (0.5,0) circle (.5);
```

`/tikz/shadow xshift=<dimension>` (无默认值, 初始值 0pt)

本选项设置阴影相对于原路径在水平方向的偏移量。注意 PGF 用画布变换实现此选项的作用。



```
\tikz [even odd rule]
\draw [general shadow={fill=red,shadow xshift=-5pt}]
(0,0) circle (.5) (0.5,0) circle (.5);
```

`/tikz/shadow yshift=<dimension>` (无默认值, 初始值 0pt)

本选项设置阴影相对于原路径在垂直方向的偏移量。注意 PGF 用画布变换实现此选项的作用。

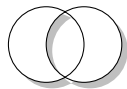
66.3 预定义的特殊阴影

66.3.1 Drop Shadows

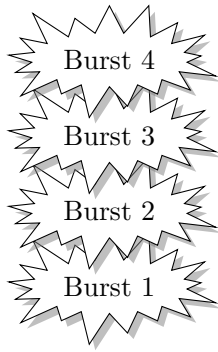
`/tikz/drop shadow=<shadow options>`

本选项给路径或 node 添加 drop shadow. 本选项实际使用 `general shadow` 来工作, `<shadow options>` 是关于图形外观的选项设置。在执行 `<shadow options>` 之前会先执行以下选项:

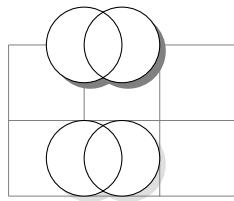
```
shadow scale=1, shadow xshift=.5ex, shadow yshift=-.5ex,
opacity=.5, fill=black!50, every shadow
```



```
\tikz [even odd rule]
\filldraw [drop shadow,fill=white]
(0,0) circle (.5) (0.5,0) circle (.5);
```



```
\begin{tikzpicture}
  \foreach \i in {1,...,4}
    \node[starburst,drop shadow,fill=white,draw]
      at (0,\i) {Burst \i};
\end{tikzpicture}
```



```
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (3,2);
  \filldraw [drop shadow={opacity=1},fill=white]
    (1,2) circle (.5) (1.5,2) circle (.5);
  \filldraw [drop shadow={opacity=0.25},fill=white]
    (1,.5) circle (.5) (1.5,.5) circle (.5);
\end{tikzpicture}
```

`/tikz/every shadow` (style, 初始值 empty)

为每个阴影设置样式。

66.3.2 Copy Shadows

一个 copy shadow 实际上不是阴影，而是原路径的复制品，只是被原路径遮挡了，并且相对于原路径有一定偏移。

`/tikz/copy shadow=<shadow options>` (默认值 empty)

<shadow options> 是关于图形外观的选项设置。在执行 <shadow options> 之前会先执行以下选项：

shadow scale=1, shadow xshift=.5ex, shadow yshift=-.5ex, every shadow



```

\begin{tikzpicture}
  \node [copy shadow,fill=blue!20,draw=blue,thick]
    {Hello World!};
  \node at (0,-1) [copy shadow={shadow xshift=1ex,
    shadow yshift=1ex},fill=blue!20,draw=blue,thick]
    {Hello World!};
  \node at (0,-2) [copy shadow={opacity=.5},tape,
    fill=blue!20,draw=blue,thick]
    {Hello World!};
  \node at (0,-3) [copy shadow={left color=blue!50},
    left color=blue!50,draw=blue,thick]
    {Hello World!};
\end{tikzpicture}

```

`/tikz/double copy shadow=<shadow options>` (默认值 `empty`)

将原路径赋值两次来制作阴影。



```

\begin{tikzpicture}
  \node [double copy shadow,fill=blue!20,draw=blue,thick]
    {Hello World!};
  \node at (0,-1) [double copy shadow={shadow xshift=1ex,
    shadow yshift=1ex},fill=blue!20,draw=blue,thick]
    {Hello World!};
  \node at (0,-2) [double copy shadow={opacity=.5},tape,
    fill=blue!20,draw=blue,thick]
    {Hello World!};
  \node at (0,-3) [double copy shadow={left color=blue!50},
    left color=blue!50,draw=blue,thick]
    {Hello World!};
\end{tikzpicture}

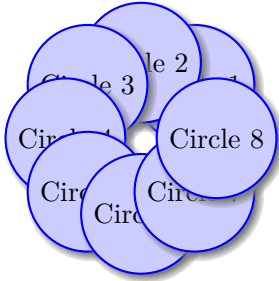
```

66.4 针对圆形的阴影

下面的阴影对圆形路径或圆形 node 的效果较好，如果用于其它路径则可能会显得很奇怪。

`/tikz/circular drop shadow=<shadow options>`

`<shadow options>` 是关于图形外观的选项设置。在执行 `<shadow options>` 之前会先执行以下选项：
`shadow scale=1.1, shadow xshift=.3ex, shadow yshift=-.3ex,`
`fill=black, path fading={circle with fuzzy edge 15 percent},`
`every shadow,`

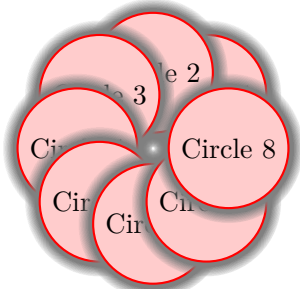


```
\begin{tikzpicture}
\foreach \i in {1,...,8}
\node[circle,circular drop shadow,draw=blue,
fill=blue!20,thick]
at (\i*45:1) {Circle \i};
\end{tikzpicture}
```

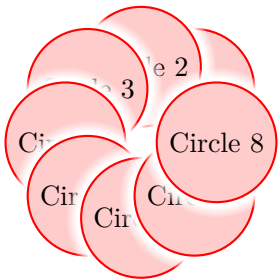
`/tikz/circular glow=<shadow options>`

<shadow options> 是关于图形外观的选项设置。在执行 <shadow options> 之前会先执行以下选项：

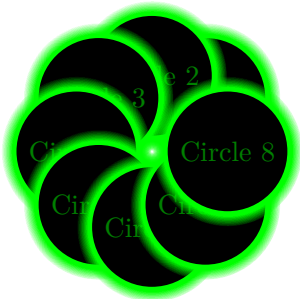
```
shadow scale=1.25, shadow xshift=0pt, shadow yshift=0pt,
fill=black, path fading={circle with fuzzy edge 15 percent},
every shadow,
```



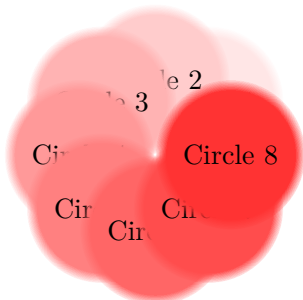
```
\begin{tikzpicture}
\foreach \i in {1,...,8}
\node[circle,circular glow,
fill=red!20,draw=red,thick]
at (\i*45:1) {Circle \i};
\end{tikzpicture}
```



```
\begin{tikzpicture}
\foreach \i in {1,...,8}
\node[circle,circular glow={fill=white},
fill=red!20,draw=red,thick]
at (\i*45:1) {Circle \i};
\end{tikzpicture}
```



```
\begin{tikzpicture}
\foreach \i in {1,...,8}
\node[circle,circular glow={fill=green},
fill=black,text=green!50!black]
at (\i*45:1) {Circle \i};
\end{tikzpicture}
```



```
\begin{tikzpicture}
\foreach \i in {1,...,8}
\node[circle,circular glow={fill=red!\i0}]
at (\i*45:1) {Circle \i};
\end{tikzpicture}
```

74 数据可视化简介

数据可视化 (data visualization) 是关于数据点 (data points) 的处理的。PGF 中的数据可视化系统十分强大, 但它的底层不易用。

74.1 数据点

例如做一个关于汽车的试验, 考虑的物理量有时间、速度大小、运动方向 (一个角度值)、加速度这 4 个, 那么试验数据就需要用到 4 维空间中的点来描述, 一个数据点有 4 个分量。

渲染管线 (rendering pipeline) 会从数据点中抽取一个或数个需要的数据用于某个可视化过程, 不同的可视化过程可能会从同一个数据点中抽取不同的数据。

74.2 可视化管线 (visualization Pipeline)

数据点会作为一个很长的数据点流 (a long stream of complex data points) 提供给可视化管线, 可视化管线是对数据的一些列处理, 处理过程中会多次传递数据。第一次传递叫作 “survey phase”, 会确定数据的最大值和最小值, 从而确定绘图区域。最主要的传递过程叫作 “visualization phase”, 会把数据转换成可见的线条或点。

75 数据可视化的基本概念

75.1 Overview

首先调用程序库 `\usetikzlibrary{datavisualization}`, 然后才能使用命令 `\datavisualization` 创建可视化图形。这个程序库提供了一些基本的可视化样式, 还有其它程序库也提供了一些特别样式。

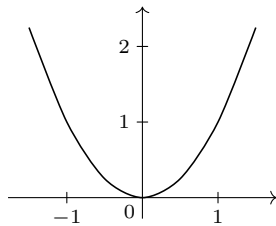
通过以下 3 方面创建可视化图形:

1. 给命令 `\datavisualization` 带上选项来确定可视化的样式, 例如, `school book axes` (教科书式), `scientific axes` (科研式)。
2. 用 `data` 算子提供数据点。
3. 用其它选项修改图形外观细节, 例如坐标轴的刻度线, 网格, 标签, 图例, 颜色, 字体等。对于这些细节, 在初始之下可视化引擎使用 TikZ 风格。

75.2 数据点与数据格式

将数据点作为 `data` 算子的参数提供出来，数据点可以是内置的 (`inline`)，即在文档中手工编辑数据，也可以放在一个外部文件中，将文件名提供给 `data` 算子。

数据点按照一定的格式编辑出来，格式有数种，并且可以自己定义。常用的格式是逗号分隔式 (`comma separated values format`)，一个数据点单独占一行，一个点内的各个分量之间用逗号分隔，例子如下

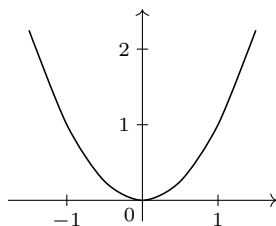


```
\begin{tikzpicture}
\datavisualization [school book axes,
visualize as smooth line]
data {
x, y
-1.5, 2.25
-1, 1
-.5, .25
0, 0
.5, .25
1, 1
1.5, 2.25
};
\end{tikzpicture}
```

必须注意，在逗号分隔式的第一行中，即声明变量标识符“`x, y`”的行中，逗号之前不能有空格。

再一种常用数据格式是键值式 (`key-value format`)，用于函数式绘图。

首先调用程序库 `datavisualization.formats.functions`，然后使用这个程序库提供的 `function` 功能，例子如下



```
\begin{tikzpicture}
\datavisualization [school book axes,
visualize as smooth line]
data [format=function] {
var x : interval [-1.5:1.5] samples 7;
func y = \value x*\value x;
};
\end{tikzpicture}
```

$\text{T}_{\text{E}}\text{X}$ 提供的数值最多包含 13 字节的整数和 16 字节的小数，数据可视化所需的计算精度和数值范围要超出 $\text{T}_{\text{E}}\text{X}$ ，因此数据可视化引擎并不使用标准的 PGF 的数值表示方式、 $\text{T}_{\text{E}}\text{X}$ 尺寸，也不使用标准的解析器，而是使用 `fpu` 程序库，参考 §54。

限于目前 `fpu` 程序库的能力，在编辑数据点时要注意以下问题：

1. 在数据点中可以写下像 10000000000000 或 0.00000000001 这样的数值。

2. 目前 fpu 程序库不支持高级解析，不能把算式 $2+3$ 作为数据点的数据，否则导致错误。
3. 如果数据点中的某个数据放在圆括号里，则该数据会用通常的解析器来解析。
 - 100000 是可接受的。
 - $2+3$ 导致错误。
 - $(2+3)$ 产生数值 5.
 - (100000) 导致错误，因为 100000 超出通常的解析器的解析范围。

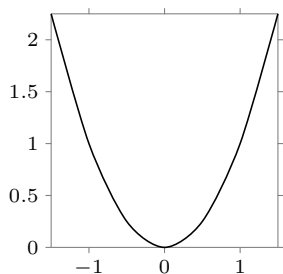
所以主要的原则就是算式放入圆括号、长数值不放入圆括号。将来这一点可能会改进。

§76 会详细讲解数据格式。

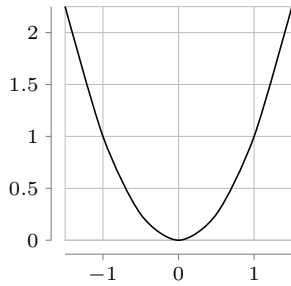
75.3 轴，刻度线，网格

多数图形有 2 个或 3 个坐标轴，可视化引擎还会使用极坐标系。利用可视化引擎可以细致地调整刻度线、网格的外观，它们有预定义的处理方式，除非必要，不必手动调整。

§77 详细讲解坐标轴、刻度线、网格。



```
\begin{tikzpicture}
  \datavisualization [
    scientific axes,
    x axis={length=3cm, ticks=few},
    visualize as smooth line
  ]
  data [format=function] {
    var x : interval [-1.5:1.5] samples 7;
    func y = \value x*\value x;
  };
\end{tikzpicture}
```

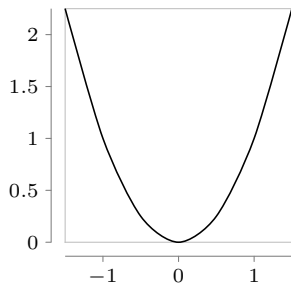


```
\begin{tikzpicture}
  \datavisualization [
    scientific axes=clean,
    x axis={length=3cm, ticks=few},
    all axes={grid},
    visualize as smooth line
  ]
  data [format=function] {
    var x : interval [-1.5:1.5] samples 7;
    func y = \value x*\value x;
  };
\end{tikzpicture}
```

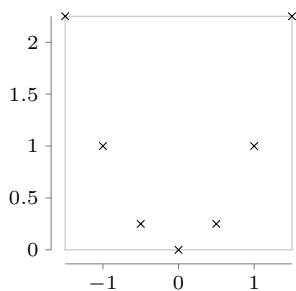
75.4 显像器 (visualizer)

最常用的显像器是 line visualizer, 会把数据转为线条图形。scatter plot visualizer 会把数据变成散点图。还有其它的显像器, 例如用于绘制统计图形的显像器。

§78 讲解显像器。



```
\begin{tikzpicture}
  \datavisualization [
    scientific axes=clean,
    x axis={length=3cm, ticks=few},
    visualize as smooth line
  ]
  data [format=function] {
    var x : interval [-1.5:1.5] samples 7;
    func y = \value x*\value x;
  };
\end{tikzpicture}
```



```
\begin{tikzpicture}
  \datavisualization [
    scientific axes=clean,
    x axis={length=3cm, ticks=few},
    visualize as scatter
  ]
  data [format=function] {
    var x : interval [-1.5:1.5] samples 7;
    func y = \value x*\value x;
  };
\end{tikzpicture}
```

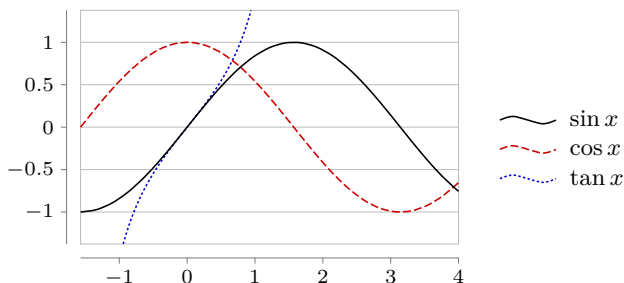
75.5 样式表和图例

在一个可视化图形中可能有多个线条，不同线条由不同显像器生成，分别有特殊的样式来相互区别。当然可以手工为每个线条设置样式，但如果线条太多就显得比较麻烦而且容易混淆。为此，可视化系统提供了“样式表”（style sheets）的概念。

样式表是由数个样式组成的列表，第 i 个线条使用样式表中的第 i 个样式。如果线条太多，样式还会被循环使用。

还可以添加图例或标签来为图形做注解。

§79 讲解样式表和图例。



```
\begin{tikzpicture}[baseline]
  \datavisualization [ scientific axes=clean,
    y axis=grid,
    visualize as smooth line/.list={sin,cos,tan},
    style sheet=strong colors,
    style sheet=vary dashing,
    sin={label in legend={text=$\sin x$}},
    cos={label in legend={text=$\cos x$}},
    tan={label in legend={text=$\tan x$}},
    data/format=function ]

  data [set=sin] {
```

```

    var x : interval [-0.5*pi:4];
    func y = sin(\value x r);
}
data [set=cos] {
    var x : interval [-0.5*pi:4];
    func y = cos(\value x r);
}
data [set=tan] {
    var x : interval [-0.3*pi:.3*pi];
    func y = tan(\value x r);
};
\end{tikzpicture}

```

75.6 用法

在 `{tikzpicture}` 环境或 `{scope}` 环境中可以使用命令 `\datavisualization` 绘制数据可视化图形，把绘图环境套嵌起来，可以在一个 `{tikzpicture}` 环境中绘制多个可视化图形。

`\datavisualization[<data visualization options>]<data specification>;`

这个命令只能用在 `{tikzpicture}` 环境或 `{scope}` 环境中。

`<data visualization options>` 用于设置图形的外观，其中的选项会被加上 key 路径前缀 `/tikz/data visualization` 来处理，这意味着一般的 TikZ 选项，例如 `draw`, `red`, `thin` 等不能在这里。

你至少要写出两个选项，第一，指定坐标轴样式，例如 `school book axes` 或 `scientific axes`；第二，指定显像器，例如选项 `visualize as line` 画出直线形。

与命令 `\path` 类似，`<data visualization options>` 不必紧跟在命令之后，可以放在语句的其它位置。

`<data specification>` 提供绘图数据。

命令以分号结束。

`\datavisualization . . . data[<options>]{<inline data>} . . . ;`

`data` 算子提供绘图数据，在一个 `\datavisualization` 内可以多次使用该算子。

如果没有 `<inline data>`，那么就提供包含绘图数据的外部文件名称，将文件名称用在 `<options>` 中以调用之，`<inline data>` 外围的花括号也必须去掉。

如果有 `<inline data>`，须用花括号将数据括起来，此时程序不会读取外部文件，即使在 `<options>` 中有外部文件名称。

`<options>` 中的选项会被冠以前缀 `/pgf/data` 来执行，以下选项可用：

`/pgf/data/read from file=<filename>` (无默认值，初始值 `empty`)

这个选项用来调用文本数据文件。注意如果调用文本文件就不要使用 `<inline data>` 及其外围的花括号。

```
\datavisualization ...
  data [read from file=file1.csv]
  data [read from file=file2.csv];
```

如果 read from file 的值留空，则应提供 `<inline data>` 并用花括号括起来。

```
\datavisualization ...
  data {
    x, y
    1, 2
    2, 3
  };
```

`/pgf/data/format=<format>` (无默认值, 初始值 `table`)

这个选项决定按照什么格式规则来读取数据，读取规则应当与数据格式一致。

默认的格式是 `table`，即“逗号分隔式”（comma-separated values）。数据格式的第一行是变量名（或者说是标识符，attribute），不同变量名用逗号分隔；后续各行，每一行都是一个数据点，一个数据点内的各个数据用逗号分隔；每个数据都与第一行相应的变量名对应，可以看作是是该变量的一个值。

为多个 data 算子统一设置选项 当 data 算子有数个时，为它们统一设置某些选项可能会比较合适。

`/tikz/every data` (style, 无值)

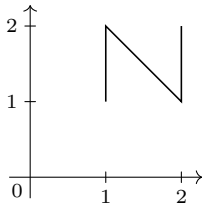
这个选项设置的样式会赋予每个 data 算子。

另外，每当前缀路径为 `/tikz/data visualization/data` 的选项被使用时，该选项的前缀路径也会被映射为 `/pgf/data`，这意味着，可以把以 `/pgf/data` 为前缀路径的选项，例如 `data/format=table` 用在命令 `\datavisualization` 的选项中，通过该命令，这个选项会传递给命令之内的各个 data 算子。

关于数据格式中的换行 数据格式中，每个数据点占一行，这当然用到“回车”符号来换行，但是 $\text{T}_{\text{E}}\text{X}$ 会取消回车的换行作用，仅把回车当作空格。因此在处理 inline 数据点时，可视化系统暂时（局部地）重定义了回车符号，使之具有换行作用。但是如果在可视化系统处理 inline 数据点之前 $\text{T}_{\text{E}}\text{X}$ 就处理了 inline 数据点，那么就会取消回车的换行作用，可视化系统无法正确读取数据点。因此，如果某个宏不能处理“脆弱”（fragile）代码，就不能把 `\datavisualization` 命令语句作为这个宏的参数，例如，当可视化命令用在 beamer 文类的帧（frame）内时，要给帧命令加 `fragile` 选项。如果可视化命令调用外部数据文件，就不会出现这个问题。

`/tikz/data visualization/data point=<options>` (无默认值)

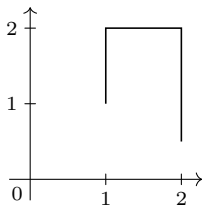
这个选项的值使用 key-value 的形式，例如 `data point={x=1, y=1}`，这样定义一个（只定义一个）数据点，它可以用在样式（style）中。



```
\tikzdatavisualizationset{
  horizontal/.style={
    data point={x=#1, y=1}, data point={x=#1, y=2}},
}
\tikz \datavisualization
[ school book axes, visualize as line,
horizontal=1,
horizontal=2 ];
```

`\datavisualization . . . data point[<options>] . . . ;`

这个算子与作为选项 (key) 的 `data point={}` 的作用一样, 用于定义一个数据点。<options> 中的选项会被冠以前缀路径 `/data point` 来执行, 因此其中的选项必须使用 key-value 的赋值形式, 并且只能定义一个数据点。



```
\tikz \datavisualization
[school book axes, visualize as line]
data point [x=1, y=1] data point [x=1, y=2]
data point [x=2, y=2] data point [x=2, y=0.5];
```

`\datavisualization . . . data group[<options>]{<name>}={<data specifications>} . . . ;`

<data specifications> 是用 `data` 算子或 `data point` 算子定义的一组数据点, 算子 `data group` 将这组数据点看作一个整体, 将其命名为 <name>, 可以用 <name> 引用这组数据点, 并且 <name> 是全局 (全文档) 有效的, 可以在命令之外引用。如果先前已经存在一个名称同样为 <name> 的数据点组, 那么旧的数据点组会被清除, 代之以新的数据点组。数据点组并不传递给渲染管线, 而是被立即解析, 这里的 <options> 会与解析的 <data specifications> 一起保存起来。

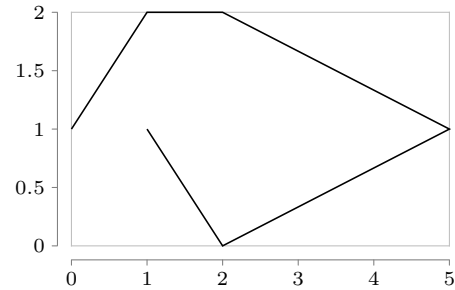
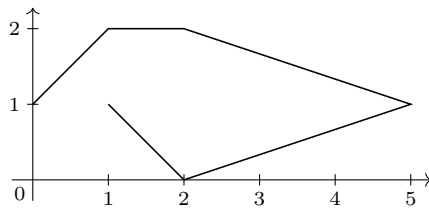
`\datavisualization . . . data group[<options>]{<name>}+={<data specifications>} . . . ;`

这里 <name> 是已有的数据点组名称, “+” 的意思是把 <data specifications> 添加到原来的 <name> 中, 从而扩充之。

`\datavisualization . . . data group[<options>]{<name>} . . . ;`

这里 {<name>} 之后没有 “+” 或 “=”, 程序会在当前位置插入名称为 <name> 的数据点组, <options> 一同被执行。

下面是个例子, 先创建一个数据点组, 然后引用之



```

\tikz \datavisualization data group {points} = {
  data {
    x, y
    0, 1
    1, 2
    2, 2
    5, 1
    2, 0
    1, 1
  }
};
\tikz \datavisualization [school book axes, visualize as line] data group {points};
\qqquad
\tikz \datavisualization [scientific axes=clean, visualize as line] data group {
  points};

```

```

\datavisualization . . . scope[<options>]{<data specification>} . . . ;

```

```

\datavisualization . . . info[<options>]{<code>} . . . ;

```

这个算子所辖的 `<code>` 是通常的 TikZ 绘图代码，在数据可视化过程结束（但命令未结束）时执行 `<code>`，而这里 `<options>` 中的选项会被冠以 `/tikz/` 来执行，也就是说，这些选项是通常的 TikZ 选项。这个句法的作用是把 TikZ 绘图代码引入到可视化命令之内，在绘图代码中可以使用可视化图形中的坐标。可视化图形有自己的坐标系，该坐标系中的点只能在可视化命令之内引用，在可视化命令之外不能引用。

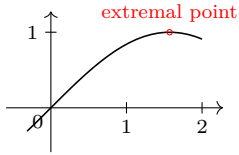
可视化坐标系

```

(visualization cs:<list of attribute-value pairs>)

```

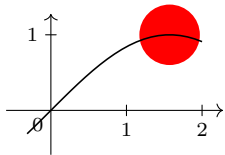
这个语句定义可视化图形坐标系中的一个点（坐标轴就是图形的坐标轴），“visualization cs:”用于引起可视化坐标系。对于每个“变量名 = 值”（`<attribute>=<value>`），都会作成 `/datapoint/<attribute>`。注意，一定要写出正确的 `<attribute>`，否则该点可能会被忽略为 `(0,0)`。



```
\begin{tikzpicture}[baseline]
\datavisualization [ school book axes, visualize as line ]
  data [format=function] {
    var x : interval [-0.1*pi:2];
    func y = sin(\value x r);
  }
  info {
    \draw [red] (visualization cs: x={(.5*pi)}, y=1)
      circle [radius=1pt]
      node [above,font=\footnotesize] {extremal point};
  };
\end{tikzpicture}
```

`\datavisualization . . . info' [<options>]{<code>} . . . ;`

这个选项与 `info` 类似，只是在可视化过程开始时执行 `<code>`，所画出的图形会被可视化图形遮挡，所以可以用这个选项画出可视化图形的背景。



```
\begin{tikzpicture}[baseline]
\datavisualization [ school book axes, visualize as line ]
  data [format=function] {
    var x : interval [-0.1*pi:2];
    func y = sin(\value x r);
  }
  info' {
    \fill [red] (visualization cs: x={(.5*pi)}, y=1)
      circle [radius=4mm];
  };
\end{tikzpicture}
```

预定义的 node

`data visualization bounding box`

这个预定义的矩形 node 保存的是“当前”可视化图形(包括坐标轴、图例、标签等)的边界盒子(bounding box)，在 `info` 所辖的 `<code>` 中可能会比较有用。

`data bounding box`

这个预定义的矩形 node 保存的是当前可视化图形中的“数据点”的边界盒子，这当然就不包括坐标轴、刻度线、图例、网格、标签等等。

75.7 在数据可视化过程中执行用户自定义的代码

下面的选项用在命令 `\datavisualization` 中，可以在可视化过程的不同阶段执行相应的自定义代码。

```
/tikz/data visualization/before survey=<code>
/tikz/data visualization/at start survey=<code>
/tikz/data visualization/at end survey=<code>
/tikz/data visualization/after survey=<code>
/tikz/data visualization/before visualization=<code>
/tikz/data visualization/at start visualization=<code>
/tikz/data visualization/at end visualization=<code>
/tikz/data visualization/after visualization=<code>
```

75.8 创建新对象

76 用于数据可视化的数据格式

76.1 Overview

76.2 简介

编辑数据点时要按照一定的格式，即数据格式（data format），数据按照格式编辑成一个矩阵，或者说一个数据表格。常用的是例如逗号分隔式（csv format，即 comma separated values），key-value format，还有比较复杂的 pdb-format。你可以指定某个格式解析器（用选项 `/pgf/data/format`），否则使用默认格式解析器（默认格式是 table，即逗号分隔式）。

举例来说，假设给出如下数据点

```
x, y, z
0, 0, 0
1, 1, 0
1, 1, 0.5
0, 1, 0.5
```

这是逗号分隔式数据点，对应的格式解析器是 `table`，第一行声明标识符（attribute，相当于变量名），标识符用逗号分隔。一个数据点占用一行，一个点内的各个数据用逗号分隔。数据用于给标识符赋值，例如，对于最后一行，`/data point/x` 被设置为 0，`/data point/y` 被设置为 1，`/data point/z` 被设置为 0.5。

必须注意，在逗号分隔式的第一行中，逗号之前不能有空格。否则会导致错误，错误信息是：

```
! Package PGF Math Error: Sorry, an internal routine of the floating point unit got an
ill-formatted floating point number `'. The unreadable part was near ''
```

76.3 内置格式

系统预设了数种数据格式。

如果不指定一种数据格式，系统就默认数据格式为 `table`。

format=table

即逗号分隔式。第一个非空行（头行，headline）用于声明标识符（attribute），标识符用来储存数据。这在前文已经解释过。再看一个例子。

```
angle, radius
0, 1
45, 2
90, 3
135, 4
```

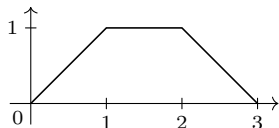
这个表格的首行设置两个标识符（即 key，参考 §82）/data point/angle 和 /data point/radius，每行的第一个数据赋予 angle，第二个赋予 radius。

注意，在默认下，标识符使用 x, y, z 这三个符号，如果使用其它符号，则需要对“轴”做相应设置，否则不能顺利绘图，参考 §77。

对于 table 格式，可以自定义分隔数据的符号（分列符）：

/pgf/data/seperator=<character> （无默认值，初始值 ,）

这个选项初始值是逗号，所设置的符号用于分隔一行内的不同列表项目。如果要把分列符设为空格，可以把该选项的值留空或者写下 separator=\space。注意，如果先将分列符设为（例如）分号 separator=;，之后又想改回逗号，则必须把逗号用花括号括起来 separator={,}。

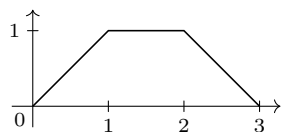


```
\begin{tikzpicture}
  \datavisualization [school book axes,
    visualize as line]
    data [separator=\space] {
      x y
      0 0
      1 1
      2 1
      3 0
    };
\end{tikzpicture}
```

注意当使用这个选项时，就默认为 table 格式。

/pgf/data/headline=<headline>

这个选项设置数据格式的首行（headline），然后在数据表格中就可以省去设置标识符的首行。

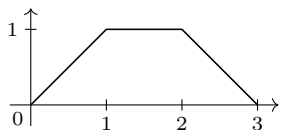


```
\begin{tikzpicture}
  \datavisualization [school book axes,
    visualize as line]
  data [headline={x, y}] {
    0, 0
    1, 1
    2, 1
    3, 0
  };
\end{tikzpicture}
```

注意当使用这个选项时，就默认为 table 格式。

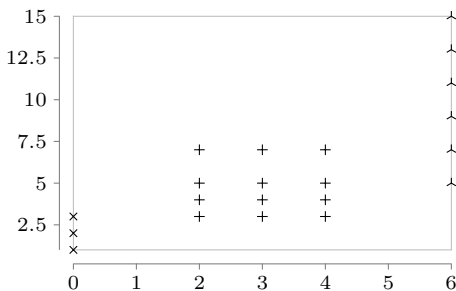
format=named

这个格式中，一个数据点仍然占一行，一行内的数据仍然用逗号分隔，只不过每个数据使用 `<attribute>=<value>` 这种赋值形式写出来，例如 `x=5, lo=500`，这会把 `/data point/x` 设为 5，把 `/data point/lo` 设为 500。



```
\begin{tikzpicture}
  \datavisualization [school book axes,
    visualize as line]
  data [format=named] {
    x=0, y=0
    x=1, y=1
    x=2, y=1
    x=3, y=0
  };
\end{tikzpicture}
```

在给标识符赋值时，不仅可以赋予单个数值，还可以赋予一个数组，而且数组中可以使用省略号构建等差数组，如 `x={1, 2, 3}`，`x={1,..., 5}` 这种形式。观察下图：



```
\tikz \datavisualization [scientific axes=clean, visualize as scatter/.list={a,b,c},
  style sheet=cross marks]
```

```
data [format=named] {
  x=0, y={1,2,3}, set=a
  x={2,3,4}, y={3,4,5,7}, set=b
  x=6, y={5,7,...,15}, set=c
};
```

数据格式的第 2 行实际构造了一个 4×3 矩阵。

`format=TeX code`

76.4 函数格式

首先调用程序库 `datavisualization.formats.functions`，这个程序库允许用户利用函数式提供数据点，从而绘制函数图形。

`format=function`

在这个格式下，把变量和函数式提供给 `data` 算子，程序根据函数式自动搜集样本点并绘制函数图形。

函数涉及自变量和因变量，下面讲解声明自变量和因变量的句法，其中用到 `var`，`<variable>`，`func`，`<attribute>`，这些都是起到标示作用的符号或符号串，因此它们与其它符号之间必须有空格分隔。

变量声明 提供变量用到“变量声明”（variable declaration），变量声明的句法如下：

1. `var <variable> : interval[<low>:<high>] samples <number>;`

注意 `var` 与 `<variable>` 之间有空格，`<variable>` 与冒号“:”之间有空格。如果没有 `samples`，那么样本点的数量由下一选项的值决定：

`/pgf/data/samples=<number>` （无默认值，初始值 25）

`<variable>` 是变量名，即标识符 `attribute`，它的 key 路径是 `/data point/<variable>`。

`interval` 设置变量的变化区间。

2. `var <variable> : interval[<low>:<high>] step <step>;`

注意 `var` 与 `<variable>` 之间有空格，`<variable>` 与冒号“:”之间有空格。当计算样本点时，`step` 设置样本点的自变量的变化步长，即样本点的自变量值从 `<low>` 开始，以 `<step>` 为公差增长。

3. `var <variable> : {<values>;}`

注意 `var` 与 `<variable>` 之间有空格，`<variable>` 与冒号“:”之间有空格。`<values>` 是个数值列表，它直接规定样本点的自变量值，该列表可以使用省略号来构造等差数列。

4. 预定义的特殊变量 `set`，参考 §79.5.

在一个 `data` 算子中，可以定义多个变量，各个变量独立变化，例如写下

```
var x : interval [0:1]
var y : interval [0:1]
```

对于每个变量来说，其样本点数量默认为 25，故由 `x`，`y` 构成的点 `(x, y)` 的个数就是 625。

函数声明 声明函数的句法是：

```
func <attribute> = <expression>;
```

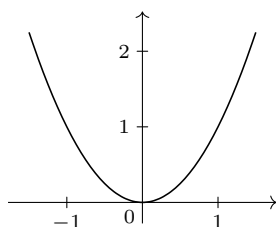
注意 `func` 与 `<attribute>` 之间有空格，`<attribute>` 与等号“=”之间有空格。`<expression>` 是函数表达式，会用标准的 TikZ 数学解析器来解析之。应该用变量声明中所声明的变量来构造 `<expression>`，程序根据自变量的样本点值来计算函数值，函数值储存在 `/data point/<attribute>` 中，`<attribute>` 是标识符，即因变量。

`<expression>` 中的自变量必须使用

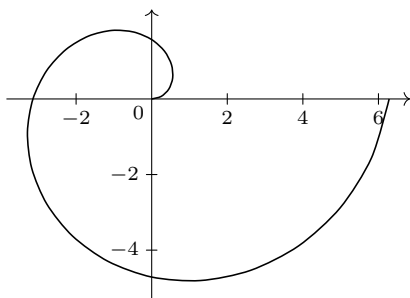
```
\value{<variable>}
```

这种形式，自变量会被展开为 `/data point/<variable>` 的值来计算。

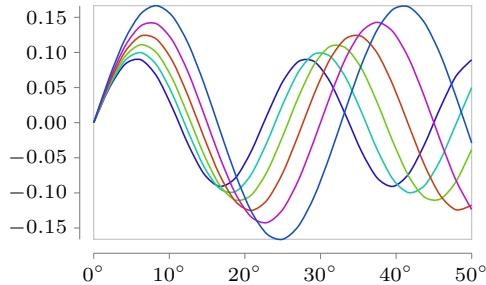
在一个 `data` 算子中，可以声明多个函数。



```
\tikz
  \datavisualization
    [school book axes, visualize as smooth line]
  data [format=function] {
    var x : interval [-1.5:1.5];
    func y = \value x * \value x;
  };
```



```
\tikz \datavisualization [
  school book axes,
  all axes={unit length=5mm, ticks={step=2}},
  visualize as smooth line]
data [format=function] {
  var t : interval [0:2*pi];
  func x = \value t * cos(\value t r);
  func y = \value t * sin(\value t r);
};
```



```
\tikz \datavisualization [
  scientific axes=clean,
  y axis={ticks={style={
    /pgf/number format/fixed,
    /pgf/number format/fixed zerofill,
    /pgf/number format/precision=2}}},
  x axis={ticks={tick suffix=${}^\circ}},
  visualize as smooth line/.list={1,2,3,4,5,6},
  style sheet=vary hue]
data [format=function] {
  var set : {1,...,6};
  var x : interval [0:50];
  func y = sin(\value x * (\value{set}+10))*
    (\value{set}+5)^(-1);
};
```

76.5 数据处理过程

76.6 定义新格式

77 坐标轴

77.1 Overview

对于平面绘图来说，一般需要 2 个维度来描述数据，笛卡尔系是横坐标和纵坐标，极坐标系是角度和极径，在可视化引擎看来，两种坐标系都涉及 2 个“轴”。

一般情况下，数据可视化系统中不能直接使用 TikZ 的绘图命令或选项，如命令 `\draw`，关于线型、颜色的选项，关于 node 的各种选项等等。但是对于绘图来说，TikZ 的命令或选项十分有力，因此可视化系统定义了能够沟通可视化命令与 TikZ 的算子和选项。例如 §75.6 中的 `info`，`info'` 算子，§77.4.7 中介绍的 `style`，`node style` 选项。这两个选项中的值（选项）都是以 `/tikz` 开头的 key，但是在这两个选项中，并非所有的 TikZ 选项都有效。这两个选项可以用在很多绘图要素中，例如，刻度 ticks，网格 grid，可视化轴 `visualize axis` 等等。

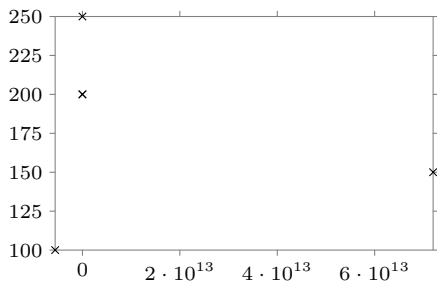
77.2 轴的基本设置

可视化图形中可能会画出坐标轴，坐标轴上还可能会有刻度线。但是图形中的坐标轴并不是这里所说的“轴”，这里说的“轴”其实是一个数据处理模式、计算过程，举例来说，写出数据点

```
x, y
-56000000000000, -1
0, 0
```

```
7200000000000, 1
```

其中给出两个变量名 (attribute): x , y , 它们的 key 路径是 `/data point/x` 和 `/data point/y`. 注意 x , y 不是“轴”的名称, 只是变量名, 每个变量应该对应一个“轴”, 例如默认下 x 对应的轴是 `x axis`, y 对应的轴是 `y axis`, 也就是说, 轴也有自己的名称, 有名称就便于加以引用、设置. x 的值集是区间 $[-5.6 \cdot 10^{12}, 7.2 \cdot 10^{12}]$ 的子集, 这个区间跨度很大, 如果不适当处理一下就不便于在页面上显示. 为了能在页面上适当显示图形, 例如, 使得图形的横坐标轴长度为 5cm, 此时需要对轴 `x axis` 做相应地设置, 例如, 将区间 $[-5.6 \cdot 10^{12}, 7.2 \cdot 10^{12}]$ 变换到 5cm 长, 在默认下, 轴会自动检查自变量值的范围, 并把这个范围转变成为一个长度比较“合理的”区间以便于绘图. 如果需要对数坐标轴, 就需要设置相应的选项了.



```
\tikz \datavisualization [scientific axes,
    visualize as scatter]
    data {
        x, y
        -5600000000000, 100
        1910, 200
        1950, 200
        1960, 250
        72000000000000, 150
    };
```

77.2.1 用法

用下面的选项创建一个“轴”并为其命名:

```
/tikz/data visualization/new axis base=<axis name>
```

创建一个轴, `<axis name>` 是轴名称, 默认使用笛卡尔坐标系样式的轴, 轴的名称是 `x axis`, `y axis`, `z axis`, 分别对应的变量 (attribute) 名称是 `x`, `y`, `z`. 轴会自动或按选项的设置, 将变量值范围映射到某个合理的区间内. `<axis name>` 会被做成一个 key, 可以用下面的选项来设置这个 key 的某些属性:

```
/tikz/data visualization/<axis name>=<options>
```

用选项 `new axis base=<axis name>` 创建一个轴后, 再用这个选项对这个轴做设置. `<options>` 会被冠以路径前缀 `/tikz/data visualization/axis options` 来执行.

```
/tikz/data visualization/all axes=<options>
```

`<options>` 对当前环境内的所有的轴有效.

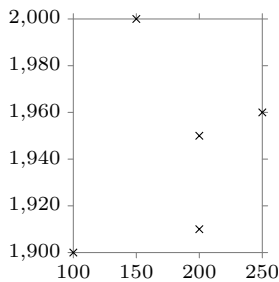
以上两个选项中的 `<options>` 之内可以用的选项将在下文介绍.

77.2.2 与轴对应的变量

每个轴都对应一个变量, 默认下, `x axis`, `y axis`, `z axis` 这 3 个轴对应的变量名称应该分别是 `x`, `y`, `z`, 变量名应该在数据点列表的首行给出. 如果数据点列表的首行给出的变量名是其它符号, 就需要用下面的选项把轴名称跟变量名对应起来.

`/tikz/data visualization/axis options/attribute=<attribute>`

这个选项用作轴名称 `<axis name>=<options>` 的选项，将这个轴与变量 `<attribute>` 对应起来，这个轴会监测该变量的取值，将变量值映射到一个合理的区间中。



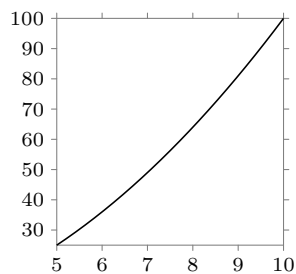
```
\tikz \datavisualization [scientific axes,
    x axis={attribute=people, length=2.5cm, ticks=few},
    y axis={attribute=year},
    visualize as scatter]
data {
    year, people
    1900, 100
    1910, 200
    1950, 200
    1960, 250
    2000, 150
};
```

77.2.3 变量值的范围

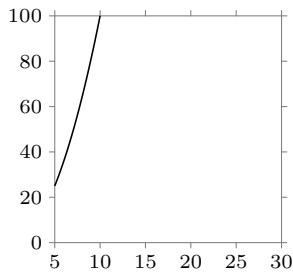
变量的当前值储存在 `/data point/<attribute>` 中，可视化引擎会检查变量的最大值与最小值从而确定变量的变化范围，称之为“变量值集”（attribute range interval）。变量值集一般由数据点列表决定，从而决定可视化图形中坐标轴的刻度值。可以用下面的选项向变量值集中添加数据，从而改变可视化图形中坐标轴的显示范围。

`/tikz/data visualization/axis options/include value=<list of value>`

这个选项用作轴名称 `<axis name>=<options>` 的选项，向该轴对应的变量值集中添加数据。



```
\tikz \datavisualization [scientific axes,
    all axes={length=3cm},
    visualize as line]
data [format=function] {
    var x : interval [5:10];
    func y = \value x * \value x;
};
```



```
\tikz \datavisualization [scientific axes,
    all axes={length=3cm},
    visualize as line,
    x axis={include value={12,30}},
    y axis={include value=0}]
data [format=function] {
    var x : interval [5:10];
    func y = \value x * \value x;
};
```

`/tikz/data visualization/axis options/min value=<value>`

这个选项用作轴名称 `<axis name>=<options>` 的选项，指定该轴对应的变量值集的最小值，如果 `<value>` 小于原来变量值集的最小值，则相当于向该变量值集中添加数据 `<value>`。如果 `<value>` 大于原来变量值集的最小值，那么变量值集中小于 `<value>` 的数据将被排除在可视化范围之外。

`/tikz/data visualization/axis options/max value=<value>`

这个选项用作轴名称 `<axis name>=<options>` 的选项，作用类似于 `min value=`。

77.2.4 轴对数据的变换

射影变换 前面提到，轴会监测变量值的范围，并将这个范围变换到一个合理的区间内以利于绘图，在默认下，轴的这种变换是数轴间的射影变换。设想有两个数轴 \mathcal{L}_1 和 \mathcal{L}_2 ， $s_1, s_2 \in \mathcal{L}_1$ ， $t_1, t_2 \in \mathcal{L}_2$ ，用点变换

$$s_1 \rightarrow t_1, \quad s_2 \rightarrow t_2,$$

来规定一个射影变换，在这个变换下有点变换：

$$\mathcal{L}_1 \ni s \rightarrow t_1 + (s - s_1) \frac{t_2 - t_1}{s_2 - s_1} \in \mathcal{L}_2.$$

例如，

$$-100 \rightarrow 0, \quad 100 \rightarrow 2, \quad s \rightarrow \frac{s + 100}{100},$$

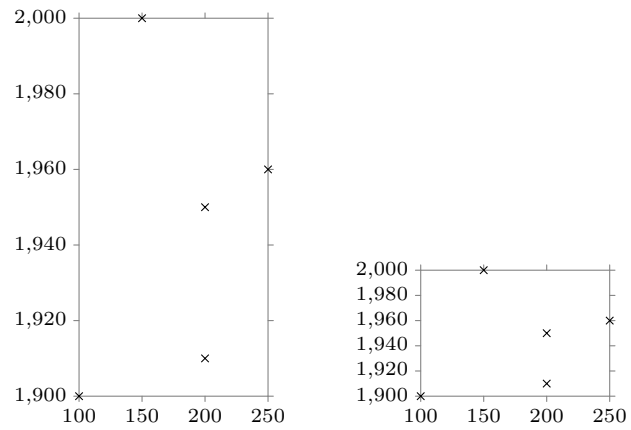
在这个变换下，不仅改变了单位区间的长度（长度比例 100 : 1），还改变了原点的位置。

轴对数据的射影变换用如下选项设置：

`/tikz/data visualization/axis options/scaling=<s1> at <t1> and <s2> at <t2>`

这个选项用作轴名称 `<axis name>=<options>` 的选项，其中 `<s1>`，`<s2>` 是变换前的数值，`<t1>`，`<t2>` 是变换后的数值。注意 `<s1>`，`<s2>` 可以不属于变量值集合。`<s1>` 可以是 `min`（代表变量的最小值），`<s2>` 可以是 `max`（代表变量的最大值）。对于数值变换 $s \rightarrow t$ ，变换前的数值 s 仍然被储存在 `/data point/<attribute>` 中，而变换后的数值 t 会被储存在 `/data point/<attribute>/scaled` 中。

下面例子中的两个图形，由于 `<s1>`，`<s2>` 的选择不同，导致纵轴的高度不同。



```

\tikz \datavisualization
  [scientific axes,
   x axis={attribute=people, length=2.5cm, ticks=few},
   y axis={attribute=year, scaling=1900 at 0cm and 2000 at 5cm},
   visualize as scatter]
data {
  year, people
  1900, 100
  1910, 200
  1950, 200
  1960, 250
  2000, 150
};
\quad
\tikz \datavisualization
  [scientific axes,
   x axis={attribute=people, length=2.5cm, ticks=few},
   y axis={attribute=year, scaling=1800 at 0cm and 2100 at 5cm},
   visualize as scatter]
data {
  year, people
  1900, 100
  1910, 200
  1950, 200
  1960, 250
  2000, 150
};

```

函数变换 首先用选项 `scaling=<s1> at <t1> and <s2> at <t2>` 规定一个变换（否则使用默认的射影变换），然后再用下面的选项

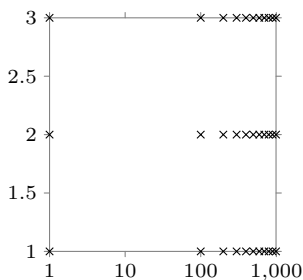
`/tikz/data visualization/axis options/function=<code>`

这个选项用作轴名称 `<axis name>=<options>` 的选项，其中的 `<code>` 定义一个函数 f ，则有点变换

$$t = t_1 + (f(s) - f(s_1)) \frac{t_2 - t_1}{f(s_2) - f(s_1)}.$$

例如定义以 10 为底的常用对数变换的句法是：

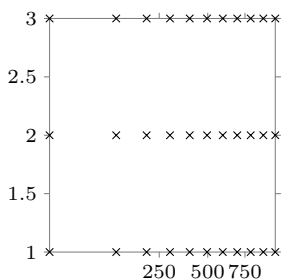
`<axis name>={function=\pgfdvmathln{\pgfvalue}{\pgfvalue}}`



```
\tikz \datavisualization
  [scientific axes,
   x axis={ticks={major={at={1,10,100,1000}}}},
   scaling=1 at 0cm and 1000 at 3cm,
   function=\pgfdvmathln{\pgfvalue}{\pgfvalue}},
  visualize as scatter]
data [format=named] {
  x={1,100,...,1000}, y={1,2,3}
};
```

定义开平方变换的句法是：

`<axis name>={function=\pgfdvmathunaryop{\pgfvalue}{sqrt}{\pgfvalue}}`



```
\tikz \datavisualization
  [scientific axes,
   x axis={ticks=few,
   scaling=1 at 0cm and 1000 at 3cm,
   function=\pgfdvmathunaryop{\pgfvalue}{sqrt}{\pgfvalue}},
  visualize as scatter]
data [format=named] {
  x={1,100,...,1000}, y={1,2,3}
};
```

设置默认的射影变换

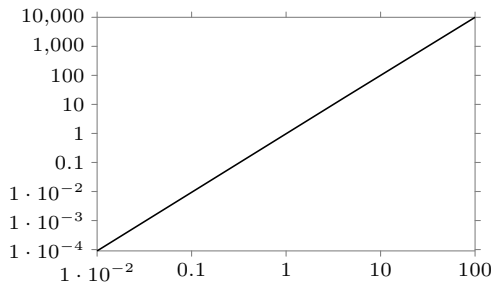
`/tikz/data visualization/axis options/scaling/default=<text>`

如果不明显地指定射影变换，则默认的射影变换由这个选项指定。

77.2.5 对数轴

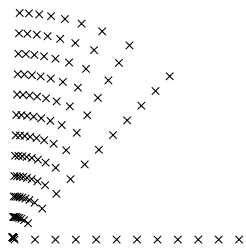
/tikz/data visualization/axis options/logarithmic (无值)

这个选项用作轴名称 <axis name>=<options> 的选项，将该轴设为对数轴，不必再使用 function= 选项来定义对数代码，而且默认的射影变换也会被使用。

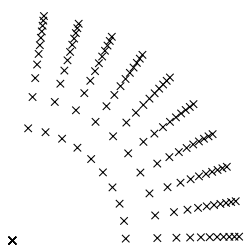


```
\tikz \datavisualization [scientific axes,
    x axis={logarithmic},
    y axis={logarithmic},
    visualize as line]
data [format=function] {
    var x : interval [0.01:100];
    func y = \value x * \value x;
};
```

这个选项对极坐标下的轴也有效，下面的例子用了 new polar axes 选项，使用该选项需要调用程序库 datavisualization.polar，该选项用于声明极坐标系，在默认下坐标系的角度轴的名称是 angle axis，极径轴的名称是 radius axis:



```
\tikz \datavisualization
    [new polar axes,
    angle axis={logarithmic, scaling=1 at 0 and 90 at 90},
    radius axis={scaling=0 at 0cm and 100 at 3cm},
    visualize as scatter]
data [format=named] {
    angle={1,10,...,90}, radius={1,10,...,100}
};
```



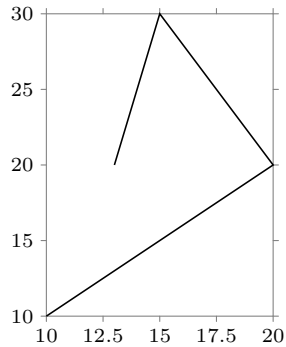
```
\tikz \datavisualization
    [new polar axes,
    angle axis={degrees},
    radius axis={logarithmic, scaling=1 at 0cm and 100 at 3cm},
    visualize as scatter]
data [format=named] {
    angle={1,10,...,90}, radius={1,10,...,100}
};
```

77.2.6 设置坐标轴的长度和单位长度

/tikz/data visualization/axis options/length=<dimension>

这个选项用作轴名称的选项，设置 scaling=min at 0cm and max at <dimension>, 指定坐标轴的长

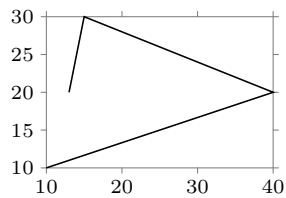
度。



```
\tikz \datavisualization [scientific axes,
  x axis={length=3cm},
  y axis={length=4cm},
  all axes={ticks=few},
  visualize as line]
data {
  x, y
  10, 10
  20, 20
  15, 30
  13, 20
};
```

`/tikz/data visualization/axis options/unit length=<dimension>`

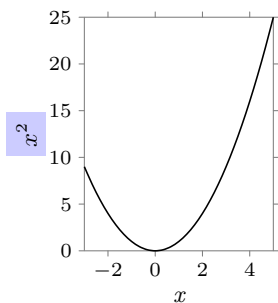
这个选项用作轴名称的选项，设置 `scaling=0` at 0cm and 1 at <dimension>，指定一个长度单位的实际长度。



```
\tikz \datavisualization [scientific axes,
  all axes={ticks=few, unit length=1mm},
  visualize as line]
data {
  x, y
  10, 10
  40, 20
  15, 30
  13, 20
};
```

`/tikz/data visualization/axis options/unit length=<dimension>per<number>units`

这个选项用作轴名称的选项，指定 <number> 个长度单位的实际长度。



```
\tikz \datavisualization
  [scientific axes,
   x axis = {label, length=2.5cm},
   y axis = {label={[\node style={fill=blue!20}]{x^2$}}},
   visualize as smooth line]
  data [format=function] {
    var x : interval [-3:5];
    func y = \value x * \value x;
  };
```

注意，尽管使用了 `node style` 选项，但这里的选项 `label` 并不创建一个 `node`，而是生成一个与标签对应的 `key`，即 `visualize label`，参考 §77.5.5。

77.2.8 仿射轴

默认下的轴是笛卡尔直角坐标系样式的，用选项 `new axis base` 设置的轴也是笛卡尔直角坐标系样式的，可以设置“非直角”坐标系样式的轴。

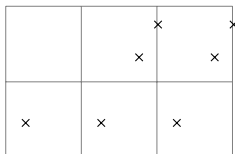
`/tikz/data visualization/new Cartesian axis=<name>`

这个选项创建一个 Cartesian 轴，然后用下面的选项指定这个轴的单位向量：

`/tikz/data visualization/axis options/unit vector=<coordinate>`（无默认值，初始值 `(1pt,0pt)`）

注意到这个选项的初始值是 `(1pt,0pt)`，所以用 `new Cartesian axis=<name>` 创建的轴会对变量值集做变换。

这个选项可以与 `scaling=<s1> at <t1> and <s2> at <t2>` 一起使用。



```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\datavisualization
  [new Cartesian axis=x axis,
   x axis={attribute=x},
   new Cartesian axis=y axis,
   y axis={attribute=y},
   x axis={unit vector=(0:1pt)},
   y axis={scaling=0 at 10pt and 10 at 200pt, unit vector=(60:1.5pt)},
   visualize as scatter]
  data {
```



```

x, y
0, 0
1, 0
2, 0
1, 1
2, 1
1, 1.5
2, 1.5
};
\end{tikzpicture}

```

这样创建的仿射轴是不可见的，可以将它变成可见的，参考 §77.5.

77.3 轴系统

77.3.1 用法

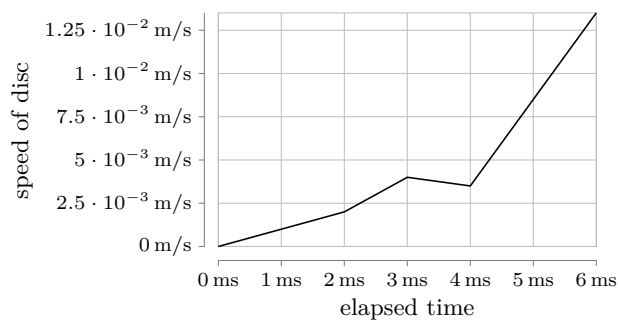
`x axis = {attribute = time}` 将轴与变量对应起来

`all axes={grid}` 给坐标系加网格

`all axes={ticks=few}` 设置坐标轴的刻度线

`x axis={ticks={tick unit=cm}}`, `y axis={ticks={tick unit=m/s^2}}` 刻度线的间隔

`x axes={label={time t (ms)}}`, `y axis={label={distance d (mm)}}` 给坐标轴加标签



```

\tikz \datavisualization
  [scientific axes=clean,
  x axis={attribute=time, ticks={tick unit=ms}, label={elapsed time}},
  y axis={attribute=v, ticks={tick unit=m/s}, label={speed of disc}},
  all axes=grid,
  visualize as line]
data {
  time, v
  0, 0
  1, 0.001

```

```

2, 0.002
3, 0.004
4, 0.0035
5, 0.0085
6, 0.0135
};

```

77.3.2 科学坐标系

`/tikz/data visualization/scientific axes=<options>`

作为命令的选项，将图形中的坐标系设为科学坐标系，自动将变量值集映射到合理的区间上。<options> 中的选项将被冠以前缀 `/tikz/data visualization/scientific axes` 来执行，以此为前缀的 key 都可以用在 <options> 中。

用下面的选项来对科学坐标系中的坐标轴做设置。

`/tikz/data visualization/scientific axes/width=<dimension>` (无默认值, 初始值 5cm)

设置科学坐标系的横轴的宽度，初始值 5cm，横轴的默认名称是 x axis。横轴对应的变量的最小值在横轴最左端，对应的变量的最大值在横轴最右端。

`/tikz/data visualization/scientific axes/height=<dimension>`

设置科学坐标系的纵轴的高度，默认下，纵轴高度与横轴宽度之比为黄金分割比例。纵轴对应的变量的最小值在纵轴最下端，对应的变量的最大值在纵轴最上端。

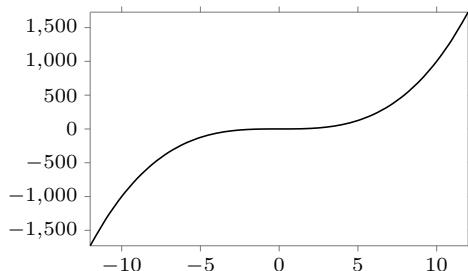
图形中，数据区域的外围有一个矩形外框，框线是灰色 (black!50)。

`/tikz/data visualization/every scientific axes` (style, 无值)

设置科学坐标系的各个轴的风格。

`/tikz/data visualization/scientific axes/outer ticks` (无值)

使得科学坐标系的轴的刻度线指向图形外部，这是默认的。



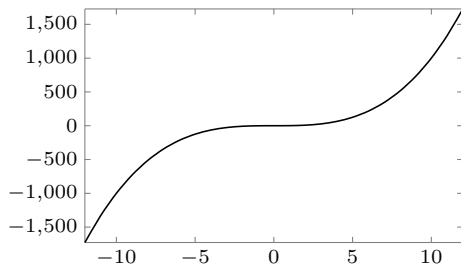
```

\begin{tikzpicture}
\datavisualization [scientific axes=outer ticks,
visualize as smooth line]
data [format=function] {
var x : interval [-12:12];
func y = \value x*\value x*\value x;
};
\end{tikzpicture}

```

`/tikz/data visualization/scientific axes/inner ticks`

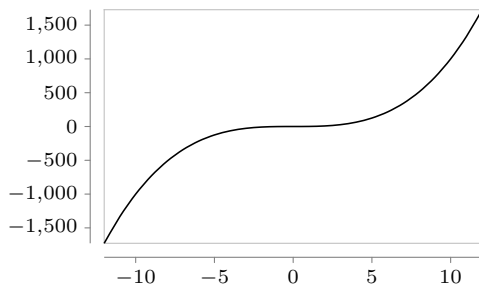
使得科学坐标系的轴的刻度线指向图形内部。



```
\begin{tikzpicture}
\datavisualization [scientific axes=inner ticks,
visualize as smooth line]
data [format=function] {
var x : interval [-12:12];
func y = \value x*\value x*\value x;
};
\end{tikzpicture}
```

`/tikz/data visualization/scientific axes/clean`

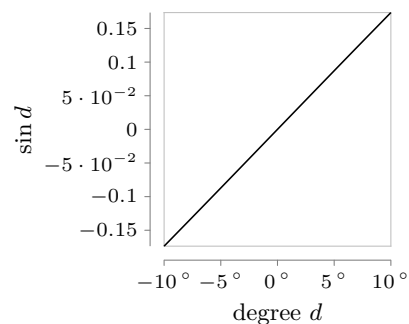
使得坐标轴以及刻度线脱离数据绘图区域，而绘图区域会用灰色框线标示出来，这样轴线、刻度线就不会与数据点重叠，以免干扰读图（尤其是对散点图）。



```
\tikz \datavisualization
[scientific axes=clean,
visualize as smooth line]
data [format=function] {
var x : interval [-12:12];
func y = \value x*\value x*\value x;
};
```

`/tikz/data visualization/scientific axes/standard labels`

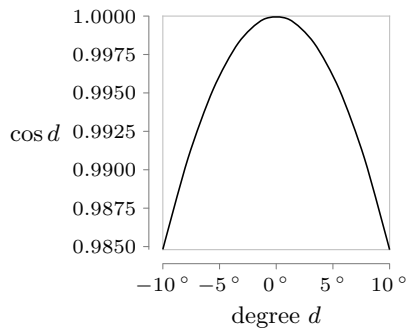
这个选项是默认的。横轴的标签位于横轴中间的下部，标签方向是从左向右的。纵轴的标签位于纵轴中间的左侧，并且标签方向是从下向上的。



```
\tikz \datavisualization
[scientific axes={clean, standard labels},
visualize as smooth line,
x axis={label=degree $d$,
ticks={tick unit={}\^\circ},
length=3cm},
y axis={label=$\sin d$}]
data [format=function] {
var x : interval [-10:10] samples 10;
func y = sin(\value x);
};
```

`/tikz/data visualization/scientific axes/upright labels`

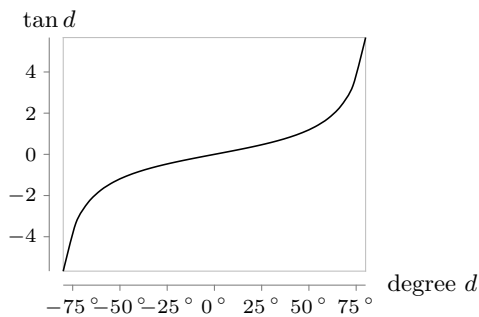
与 `standard labels` 类似，只是纵轴的标签方向是从左向右的。



```
\tikz \datavisualization
  [scientific axes={clean, upright labels},
   visualize as smooth line,
   x axis={label=degree $d$,
    ticks={tick unit={}\^\circ},
    length=3cm},
   y axis={label=$\cos d$, include value=1,
    ticks={style={
      /pgf/number format/precision=4,
      /pgf/number format/fixed zerofill}}}]
  data [format=function] {
    var x : interval [-10:10] samples 10;
    func y = cos(\value x);
  };
```

`/tikz/data visualization/scientific axes/end labels`

将坐标轴的标签放在坐标轴的末端中间。



```
\tikz \datavisualization
  [scientific axes={clean, end labels},
   visualize as smooth line,
   x axis={label=degree $d$,
    ticks={tick unit={}\^\circ},
    length=4cm},
   y axis={label=$\tan d$}]
  data [format=function] {
    var x : interval [-80:80];
    func y = tan(\value x);
  };
```

77.3.3 教科书坐标系

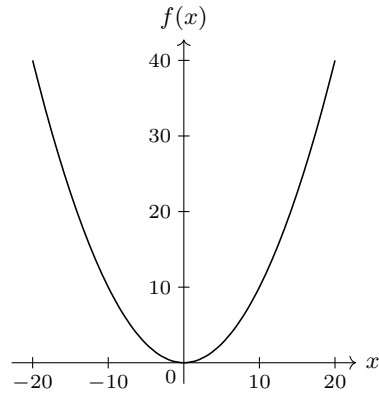
教科书坐标系的特点是，两个坐标轴交于原点，两个坐标轴的单位长度都是 1cm，当然轴的单位长度也可以另作设置。

`/tikz/data visualization/school book axes=<options>` (无默认值)

将坐标系设为教科书坐标系。<options> 中使用以下选项：

`/tikz/data visualization/school book axes/unit=<value>`

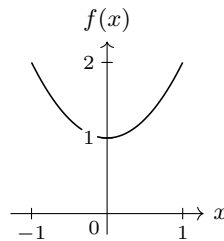
效果相当于 `scaling=0 at 0cm and <value> at 1cm`，刻度值也会随之变化。



```
\begin{tikzpicture}
\datavisualization [school book axes={unit=10}, visualize as smooth line,
                    clean ticks, x axis={label=$x$}, y axis={label=$f(x)$}]
data [format=function] {
  var x : interval [-20:20];
  func y = \value x*\value x/10;
};
\end{tikzpicture}
```

/tikz/data visualization/school book axes/standard labels

这个选项使得横轴的标签位于横轴右端中间，纵轴标签位于纵轴上端中间。目前，教科书坐标系仅支持这种轴标签位置。



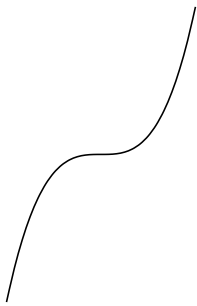
```
\begin{tikzpicture}
\datavisualization [school book axes={standard labels}, visualize as smooth line,
                    clean ticks, x axis={label=$x$}, y axis={label=$f(x)$}]
data [format=function] {
  var x : interval [-1:1];
  func y = \value x*\value x + 1;
};
\end{tikzpicture}
```

77.3.4 底层的笛卡尔坐标系

前面的图形中都使用科学坐标系或者教科书坐标系，在图形中都有相应的坐标系画出。这两种坐标系的底层使用 `xy Cartesian` 坐标系统，这个坐标系统创建两个轴：`x axis` 和 `y axis`，这两个轴是不画出的，即隐形的，科学坐标系或者教科书坐标系通过设置一些列 key 来画出坐标轴，并给坐标轴添上刻度线，标签等内容。关于画出坐标轴以及添加刻度、标签的方法，参考 §77.5。

`/tikz/data visualization/xy Cartesian`

当可视化命令使用这个选项后，就创建了一个隐形的坐标系，其中有两个轴 `x axis`，`y axis`，图形中没有坐标轴画出，在默认下，坐标轴的一个长度单位的实际长度是 1cm。



```
\begin{tikzpicture}
\datavisualization [xy Cartesian, visualize as smooth line]
  data [format=function] {
    var x : interval [-1.25:1.25];
    func y = (\value x)^3;
  };
\end{tikzpicture}
```

使用这个选项后，可以用下面的选项设置坐标轴：

`/tikz/data visualization/xy axes=<options>`

`<options>` 用于轴 `x axis`，`y axis`。

`/tikz/data visualization/xyz Cartesian cabinet`

这个选项与 `xy Cartesian` 类似，创建三个轴 `x axis`，`y axis`，`z axis`，第 3 个轴指向左下方，其一个长度单位的实际长度是 $\frac{1}{2} \sin 45^\circ$ ，这也称为“斜二侧投影”（cabinet projection）。

使用这个选项后，可以用下面的选项设置坐标轴：

`/tikz/data visualization/xyz axes=<options>`

`<options>` 用于轴 `x axis`，`y axis`，`z axis`。

`/tikz/data visualization/uv Cartesian`

这个选项与 `xy Cartesian` 类似，创建 2 个轴 `u axis`，`v axis`。

使用这个选项后，可以用下面的选项设置坐标轴：

`/tikz/data visualization/uv axes=<options>`

`<options>` 用于轴 `u axis`，`v axis`。

`/tikz/data visualization/uvw Cartesian cabinet`

这个选项与 `xyz Cartesian` 类似，创建三个轴 `u axis`，`v axis`，`w axis`，第 3 个轴指向左下方，其一个长度单位的实际长度是 $\frac{1}{2} \sin 45^\circ$ ，这也称为“斜二侧投影”（cabinet projection）。

使用这个选项后，可以用下面的选项设置坐标轴：

`/tikz/data visualization/uvw axes=<options>`

<options> 用于轴 u axis, v axis, w axis.

77.4 坐标轴的刻度和网格

77.4.1 概略

图形中如果网格，一般情况下网格与坐标轴的刻度线是重合的，有很多选项对刻度和网格是通用的。

刻度包括两个方面：刻度线和刻度值。刻度线分为 3 种：主刻度线 (major, 线较长)，副刻度线 (minor, 线较短)，次副刻度线 (subminor, 线最短)，通常只有主刻度线才有刻度值标签。

网格线也分为 3 种：major, minor, subminor, 主网格线最粗，副网格线较细，次副网格线最细。

与单词 major 有关的选项一般是关于主刻度线（网格线）的，与单词 minor 有关的选项一般是关于副刻度线（网格线）的。

有数种自动添加刻度的策略，也可以手工添加刻度。如果需要的刻度较多，则使用自动添加刻度的策略比较简便。但是刻度线的位置应该恰当，例如一般情况下，能用整数就不用小数，如果用小数则 2.5 要比 2.317 更好。自动添加刻度的策略需要做某些计算来确定“好”的刻度线的位置。

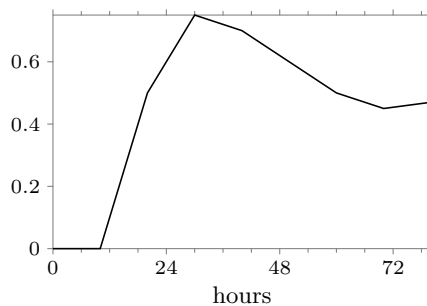
横轴的网格线是一组竖直线，纵轴的网格线是一组水平线，所以要想得到网格，需要给两个轴都带上网格选项，或者给可视化命令带上选项 `all axes=grid`。

77.4.2 刻度和网格的主要选项

`/tikz/data visualization/axis options/ticks=<options>` (默认值 `some`)

作为轴名称的选项，可用于 <options> 的选项在后文介绍，默认为 `some` (设置大约 5 个刻度线)。本选项只是一种选定刻度线的机制，并不实际画出刻度线，只有在作为 key 的 `visualize ticks` 被创建后才画出刻度线 (参考 §77.5.4)。

<options> 中的选项将被冠以 `/tikz/data visualization/` 来执行。如果在一个轴中多次使用 `ticks` 选项，则选项作用会累加。



```
\tikz \datavisualization [scientific axes, visualize as line,
    x axis={ticks={step=24, minor steps between steps=3}, label=hours}]
data {
  x, y
  0, 0
```

```

10, 0
20, 0.5
30, 0.75
40, 0.7
50, 0.6
60, 0.5
70, 0.45
80, 0.47
};

```

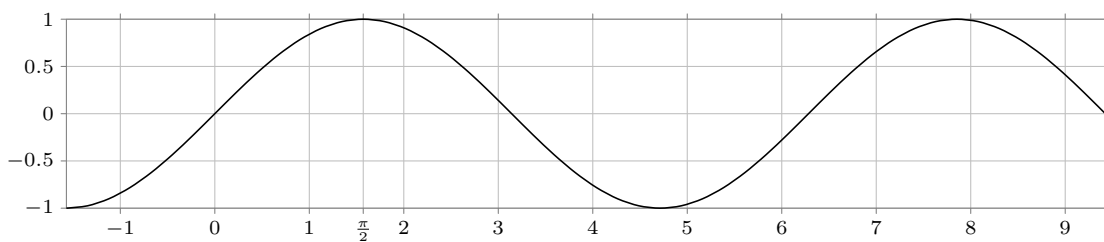
`/tikz/data visualization/axis options/grid=<options>` (默认为 `at default ticks`)

与选项 `ticks` 类似, 可用于 `ticks` 的选项也能用于本选项, 当然本选项的值只对网格线有效。本选项并不实际画出网格, 只有在作为 key 的 `visualize grid` 被创建后才画出网格 (参考 §77.5.3)。

本选项的默认值是一个文本 `at default ticks`, 这会使得网格线与刻度线重合。

`/tikz/data visualization/axis options/ticks and grid=<options>`

`<options>` 会被传递给 `ticks` 和 `grid`。



```

\tikz \datavisualization[scientific axes, visualize as smooth line,
    all axes= {grid, unit length=1.25cm}, y axis={ ticks=few },
    x axis={ ticks=many, ticks and grid={ major also at={(\pi/2) as $\frac{\pi}{2}$} }
}];
data [format=function] {
    var x : interval [-pi/2:3*pi] samples 50;
    func y = sin(\value x r);
};

```

77.4.3 计算刻度线和网格线位置的半自动机制

以下针对刻度做讲解, 对网格线也是有效的。

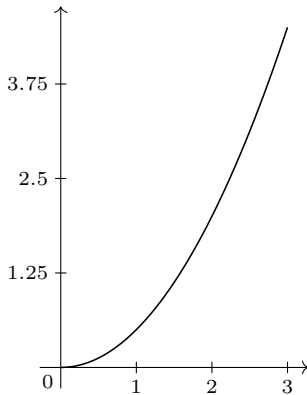
预定义的自动添加刻度的策略有两个: `linear steps` 策略和 `exponential steps` 策略 (§77.4.13)。`linear steps` 策略创建的刻度是等差数列, 假设某个坐标轴使用 `linear steps` 策略, 并且需要显示的区间是 $[a, b]$, 该策略创建的刻度就是 $i \cdot s + p \in [a, b]$, $i \in \mathbb{Z}$, 其中 s 是公差 (即步长 `step`), p 是初值 (phase), i 取不同的值得到不同的刻度, 这些刻度是主刻度; 而 `exponential steps` 策略则适用于对数坐标轴, 创建的刻度是以 $10^{i \cdot s + p}$ 为公比的等比数列, 这些刻度也是主刻度。

linear steps 策略是默认的策略。

可以手工调整上述两个策略中的 step, phase, 以及副刻度线, 用到下面的选项:

`/tikz/data visualization/step=<value>` (无默认值, 初始值 1)

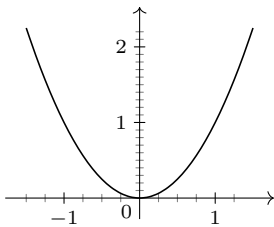
设置 linear steps 策略和 exponential steps 策略的 step.



```
\tikz \datavisualization [school book axes,
    visualize as smooth line,
    y axis={ticks={step=1.25}}]
data [format=function] {
    var x : interval [0:3];
    func y = \value x*\value x/2;
};
```

`/tikz/data visualization/minor steps between steps=<number>` (默认值 9)

设置两个主刻度线之间的副刻度线的数目。对于 linear steps 策略来说, 副刻度线会等分两个主刻度线之间的区间。对于 exponential steps 策略来说, 副刻度线仍然是按等比间隔分布的, 一般是疏密不均的。



```
\begin{tikzpicture}
\datavisualization [school book axes,
    visualize as smooth line,
    x axis={ticks={minor steps between steps=3}},
    y axis={ticks={minor steps between steps}}]
data [format=function] {
    var x : interval [-1.5:1.5];
    func y = \value x*\value x;
};
\end{tikzpicture}
```

`/tikz/data visualization/phase=<value>` (无默认值, 初始值 0)

设置 linear steps 策略和 exponential steps 策略的 phase.

在需要显示的区间 $[a, b]$ 已经确定的情况下, 假如给出 step 和 phase 的值, 就把主刻度线决定了, 但这些刻度线未必处于“好”的位置上, 也就是说, 刻度值未必是简洁或者是利于读图的值。

77.4.4 计算刻度线和网格线位置的自动机制

以下针对刻度做讲解, 对网格线也是有效的。

手工设置刻度的 step 和 phase 的值可能导致“不好”的刻度线位置, 为了避免这一点, 可以使用自动

机制。将下面的选项作为 `ticks=` 的值。

```
/tikz/data visualization/about=<number> (无默认值)
```

这个选项意思是“设置大约 `<number>` 个主刻度线”，实际得到的主刻度线可能少于也可能多于 `<number>` 个，实际的个数决定于计算过程，计算过程依赖于轴所选择的自动计算刻度的策略。

linear steps 策略下的计算 假设需要显示的区间是 $[a, b]$ ，首先将该区间按整数 `<number>` 做等分，得到步长 $s = m \cdot 10^k$ ， $1 \leq m \leq 10$ ， $k \in \mathbb{Z}$ ，注意 s 的写法。由于 `phase` 的初始值是 0，因此确定了步长就确定了主刻度线的个数，但这样得到刻度值未必足够“好”。一般而言 m 决定了步长 s 是否为一个简洁的数字，如果 m 不够简洁，就需要选择一种机制将 m 换成另外一个比较简洁的数字 m' 。有数种预定义的机制可供使用，默认的机制是“standard about strategy”，此机制下 m' 的值只允许是 1, 2, 2.5, 5 这四个值中的某一个。也可以用下一个选项自定义一种机制：

```
/tikz/data visualization/about strategy=<list> (无默认值)
```

`<list>` 是个逗号分隔的列表，列表项的形式是 `<threshold>/<value>`，斜线“/”之前的值是“界限值”，斜线之后的值是该界限值对应的转换值。找到与 m 最接近且大于 m 的那个界限值，把 m 换成相应的转换值，从而得到 m' 的值。举例说，给某个轴加上如下选项

```
about strategy={1.5/1.0,2.3/2.0,4/2.5,7/5,11/10}
```

假如 $m = 3.141$ ，那么在列表中与 3.141 最接近且大于 3.141 的那个界限值是 4，界限值 4 对应的转换值是 2.5，于是 $m' = 2.5$ ；对于 $m = 6.3$ ，得到 $m' = 5$ 。

exponential steps 策略下的计算 假设需要显示的区间是 $[a, b]$ ，将该区间对数化 $[\lg a, \lg b]$ ，将对数化后的区间按整数 `<number>` 做等分，得到步长 $s = m \cdot 10^k$ ， $1 \leq m \leq 10$ ， $k \in \mathbb{Z}$ ，注意 s 的写法。有了这个步长就可以得到区间 $[\lg a, \lg b]$ 内的刻度值，然后再以 10 为底做指数运算，得到原来区间 $[a, b]$ 内的刻度值，因此如果 s 是小数，例如 $s = 2.5$ ，就有可能得到无理数刻度值 $10^{2.5} = 100 \cdot \sqrt{10}$ ，显然不够简洁。因此在 `exponential steps` 策略下没有一种自动计算并转换步长的机制，而是使用一种固定的机制来把 m 换成 m' ， m' 的值只允许是 1, 3, 6, 10 这四个值中的某一个。

linear steps 策略下预定义的计算刻度线的机制 下面的预定义的计算刻度线的机制是针对 `linear steps` 策略的。

```
/tikz/data visualization/standard about strategy
```

这是默认的机制， m' 的值只允许是 1, 2, 2.5, 5 这四个值中的某一个。

```
/tikz/data visualization/euro about strategy
```

字面意思是欧元硬币 (Euro coins) 面值， m' 的值只允许是 1, 2, 5 这三个值中的某一个。

```
/tikz/data visualization/half about strategy
```

m' 的值只允许是 1, 5 这两个值中的某一个。

```
/tikz/data visualization/decimal about strategy
```

m' 的值只允许是 1。

`/tikz/data visualization/quarter about strategy`

m' 的值只允许是 1, 2.5, 5 这三个值中的某一个。

`/tikz/data visualization/int about strategy`

m' 的值只允许是 1, 2, 3, 4, 5 这五个值中的某一个。

`/tikz/data visualization/many`

是 `about=10` 的简写。

`/tikz/data visualization/some`

是 `about=5` 的简写。

`/tikz/data visualization/few`

是 `about=3` 的简写。

`/tikz/data visualization/none`

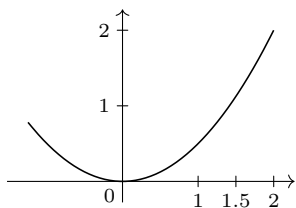
关闭步长计算，不会在坐标轴上自动添加刻度（包括刻度线和刻度值），除非另用 `step=` 显式地做设置。

77.4.5 手工确定刻度线和网格线的位置

刻度和网格有主、副、次副三个等级，可以手工方式来设置它们。

`/tikz/data visualization/major=<options>`

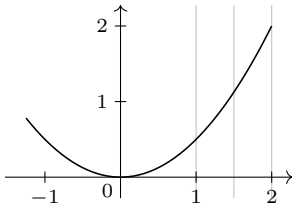
这个选项可用做 `ticks` 或 `grid` 的选项，指定主刻度（网格线）的位置，在 `<options>` 中使用选项 `at=` 或 `also at=` 来指定主刻度的位置。



```
\tikz \datavisualization [ school book axes,
    visualize as smooth line,
    x axis={ticks={major={at={1, 1.5, 2}}}}]
data [format=function] {
    var x : interval [-1.25:2];
    func y = \value x * \value x / 2;
};
```

`/tikz/data visualization/minor=<options>`

这个选项可用做 `ticks` 或 `grid` 的选项，指定副刻度（网格线）的位置。



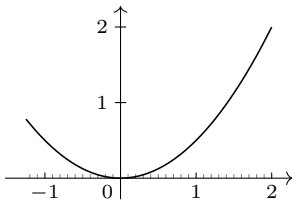
```
\tikz \datavisualization [ school book axes,
    visualize as smooth line,
    x axis={grid={minor={at={1, 1.5, 2}}}}]
data [format=function] {
    var x : interval [-1.25:2];
    func y = \value x * \value x / 2;
};
```

`/tikz/data visualization/subminor=<options>`

这个选项可用做 ticks 或 grid 的选项，指定次副刻度（网格线）的位置。

`/tikz/data visualization/common=<options>`

这个选项可用做 ticks 或 grid 的选项，其中的 <options> 对 major, minor, subminor 这 3 个 key 都有效，因此 <options> 中不能使用位置选项 at= 或 also at=.



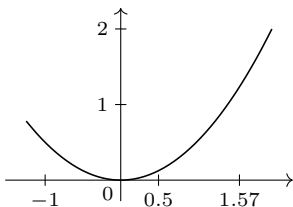
```
\tikz \datavisualization [ school book axes,
    visualize as smooth line,
    x axis={ticks={minor steps between steps,
        common={low=0}}} ]
data [format=function] {
    var x : interval [-1.25:2];
    func y = \value x * \value x / 2;
};
```

以上 major, minor, subminor, common 选项中可以带有设置外观的选项，例如 style= 选项 (§77.4.7), low=, high= 选项 (§77.5.2), 也可带有位置选项，如下。

下面的位置选项 at= 或 also at= 指定主（副、次副）刻度（网格线）的位置。

`/tikz/data visualization/at=<list>`

<list> 是个列表，将被宏 \foreach 处理，因此列表的形式比较灵活，其中可以使用省略号来构造等差项。空列表将被忽略。



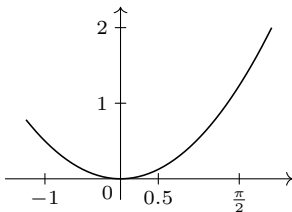
```
\tikz \datavisualization [ school book axes,
    visualize as smooth line,
    x axis={ticks={major={at={-1,0.5,(pi/2)}}}}]
data [format=function] {
    var x : interval [-1.25:2];
    func y = \value x * \value x / 2;
};
```

如果用这个选项给出主（副、次副）刻度（网格线）的位置列表，那么此选项之前的以各种（自动或手工）方式确定的主（副、次副）刻度（网格线）的位置将被压制。

使用该选项的一般句法是，例如，对于主刻度：

```
<axis name>= {ticks = {major = {at = {...}}}}
```

列表中的数字可以带有算子 `as` 来决定该数字的外观，例如 `1.2 as{\$x\$}`，那么刻度值 1.2 将被显示为数学模式下的字母 x ，而不是一个数字。注意 `as` 是算子，其后的花括号里是该算子的参数。



```
\tikz \datavisualization [ school book axes,
    visualize as smooth line,
    x axis={ticks={major={at={-1,0.5,
        (pi/2)as{\frac{\pi}{2}}}}}]}
    data [format=function] {
    var x : interval [-1.25:2];
    func y = \value x * \value x / 2;
    };
```

关于本选项，有以下稍微简捷一些的形式：

```
/tikz/data visualization/major at=<list>
```

这个选项是 `major={at={<list>}}` 的简写。

```
/tikz/data visualization/minor at=<list>
```

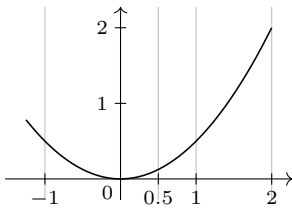
这个选项是 `minor={at={<list>}}` 的简写。

```
/tikz/data visualization/subminor at=<list>
```

这个选项是 `subminor={at={<list>}}` 的简写。

```
/tikz/data visualization/also at=<list>
```

这个选项“添加”某些主（副、次副）刻度（网格线）位置，并不清除此选项之前的以各种（自动或手工）方式确定的主刻度（网格线）的位置，因此如果多次使用该选项添加位置，那么这些位置会累计并显示在图形中。但是如果在 `also at=` 之后使用 `at=`，那么只有 `at=` 设置的位置会被显示。



```
\tikz \datavisualization [ school book axes,
    visualize as smooth line,
    x axis={grid, ticks and grid={
        major={also at={0.5}}}]
    data [format=function] {
    var x : interval [-1.25:2];
    func y = \value x * \value x / 2;
    };
```

关于本选项，也有以下稍微简捷一些的形式：

```
/tikz/data visualization/major also at=<list>
```

这个选项是 `major={also at={<list>}}` 的简写。

```
/tikz/data visualization/minor also at=<list>
```

这个选项是 `minor={also at={<list>}}` 的简写。

```
/tikz/data visualization/subminor also at=<list>
```

这个选项是 `subminor={also at={<list>}}` 的简写。

77.4.6 刻度与网格线的样式：概略

77.4.7 刻度与网格线的样式：style 与 node Style

前面讲了如何确定刻度线和网格线的位置，下面讲如何设置刻度线、刻度值、网格线的外观样式。

刻度值是作为 node 标签来创建的。

数据可视化系统中的 key 都有前缀 `/tikz/data visualization`，一般情况下以 `/tikz` 开头的 key 不能直接用在可视化命令中。Tikz 中有许多针对形状、旋转、颜色、透明度等外观做设置的 key，这些 key 可以通过间接的办法在可视化图形中起作用，例如 §75.6 中的 `info`，`info'` 算子，还有下面的选项 `style`，`node style` 等选项。这些选项的值是以 `/tikz` 开头的 key，可以设置对刻度和网格的外观。

```
/tikz/data visualization/style=<TikZ options>
```

`<TikZ options>` 是以 `/tikz` 开头的 key，这个 `style` 选项可以多次使用，效果累加。这些 `<TikZ options>` 不会被立即执行，而是在可视化引擎调用 TikZ 后才会被执行，或者用下面的 `styling` 选项使得它们立即被执行。这个选项可用于设置刻度或网格线的外观，例如

```
ticks={style=blue} 所有刻度值标签（不包括刻度线）是蓝色
```

```
ticks={major={style=blue}} 所有主刻度线、主刻度值都是蓝色
```

```
grid={style=red} 所有网格线都是红色
```

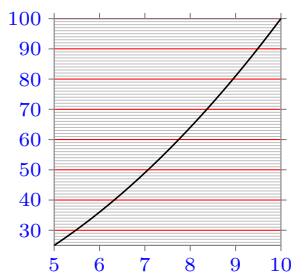
```
grid={major={style=red}} 主网格线是红色
```

```
grid={major={low=0.5, high=2}} 主网格线的起止位置，参考 \S77.5.2.
```

```
grid={major={style=red}, major also at={65 as[style=cyan]}} 画一条特殊的主网格线
```

```
ticks={major also at={65 as[low=-1.5em,style=cyan]$K$}} 画一个特殊的主刻度线和主刻度值
```

下面例子中，用 `style` 设置 y 轴的主网格线的颜色以及刻度值标签的颜色：



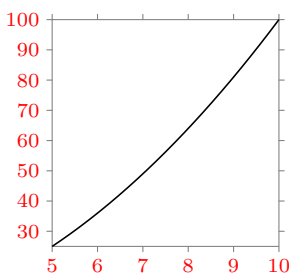
```
\tikz \datavisualization [scientific axes,
    all axes={ticks={style=blue}, length=3cm},
    y axis={grid, grid={minor steps between steps,
        major={style=red}}},
    visualize as line]
data [format=function] {
    var x : interval [5:10];
    func y = \value x * \value x;
};
```

/tikz/data visualization/styling

这个选项只能直接用作命令 `\datavisualization` 的选项。`style` 选项调用 `<TikZ options>`，如果要让这些 `<TikZ options>` 被立即执行，就可以给 `\datavisualization` 命令带上这个选项，一般情况下不必使用这个选项。

/tikz/data visualization/node style=<TikZ options>

这个选项类似 `style=` 选项，这个选项设置刻度值标签的外观。对于预定义的坐标系统，在默认下，下横轴的刻度值标签位于刻度线的下端点之下，也就是说，作为 `node` 的刻度值标签的指向点是刻度线的下端点，标签的 `anchor` 位置是 `north`；上横轴的刻度值标签位于刻度线的上端点之上，即作为 `node` 的刻度值标签的指向点是刻度线的上端点，标签的 `anchor` 位置是 `south`；左纵轴的刻度值标签位于刻度线的左端点之左，右纵轴的刻度值标签位于刻度线的右端点之右。标签的 `anchor` 位置可用本选项修改。



```
\tikz \datavisualization [scientific axes,
    all axes={ticks={node style=red}, length=3cm},
    visualize as line]
data [format=function]{
    var x : interval [5:10];
    func y = \value x * \value x;
};
```

/tikz/data visualization/node styling

类似 `styling` 选项。

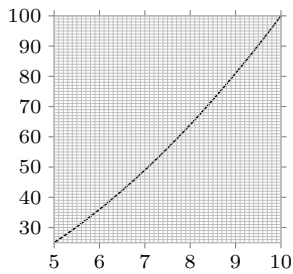
77.4.8 网格线的样式

下面的 `key` 用于设置网格线的样式，其中 `grid layer` 和 `styling` 所存储的样式将被冠以 `/tikz` 来执行。

/tikz/data visualization/grid layer (style, 初始值 on background layer)

这个选项是个样式 (`style`)，其中所存储的样式会被冠以 `/tikz` 来执行。这个选项指定网格线所在的“层”，默认是背景层，也就是说，图形会遮挡网格线。

下面的例子中重设网格线所在的层，但是没有指定是哪个层，就把网格线改到了顶层：



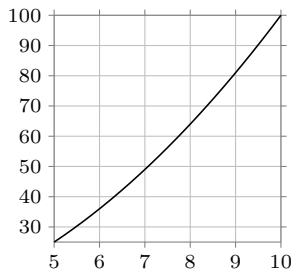
```
\tikz \datavisualization [scientific axes,
    all axes={length=3cm, grid,
        grid={minor steps between steps}},
    grid layer/.style=, % 这导致网格线遮挡图形
    visualize as line]
data [format=function] {
    var x : interval [5:10];
    func y = \value x * \value x;
};
```

`/tikz/data visualization/every grid` (style, 无值)

对所有网格线作设置，默认值是

```
low=min, high=max
```

选项 low, high 将在 §77.5.4 中讲解。可以在这个选项样式中使用 `style=` 选项来调用 TikZ 的 key:



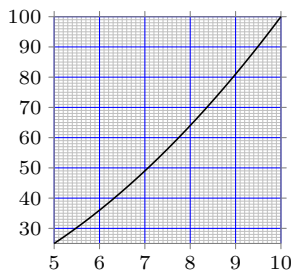
```
\tikz \datavisualization [scientific axes,
    all axes={length=3cm, grid},
    every grid/.append style={style=densely dashed},
    visualize as line]
data [format=function] {
    var x : interval [5:10];
    func y = \value x * \value x;
};
```

`/tikz/data visualization/every major grid` (style, 无值)

这个选项对所有的主网格线作设置，它的默认是

```
style = {help lines, thin, black!25}
```

可以在这个选项样式中使用 `style=` 选项来调用 TikZ 的 key:



```
\tikz \datavisualization [scientific axes,
    all axes={length=3cm, grid,
        grid={minor steps between steps}},
    every major grid/.style = {style={blue, thin}},
    visualize as line]
data [format=function] {
    var x : interval [5:10];
    func y = \value x * \value x;
};
```


`/tikz/data visualization/every minor grid` (style, 无值)

这个选项对所有的副网格线作设置，它的默认是

```
style = {help lines, thin, black!25}
```

可以在这个选项样式中使用 `style=` 选项来调用 TikZ 的 key.

`/tikz/data visualization/every subminor grid` (style, 无值)

这个选项对所有的次副网格线作设置，它的默认是

```
style = {help lines, thin, black!10}
```

可以在这个选项样式中使用 `style=` 选项来调用 TikZ 的 key.

77.4.9 刻度线与刻度值标签的样式

`/tikz/data visualization/every ticks` (style, 无值)

这个样式对所有的刻度线和刻度值标签作设置，可以在这个样式中使用选项 `style=`, `node style=` 来分别设置刻度线和刻度值标签。刻度值标签的默认设置是

```
node style={
  font=\footnotesize,
  inner sep=1pt,
  outer sep=.1666em,
  rounded corners=1.5pt
}
```

`/tikz/data visualization/every major ticks` (style, 无值)

这个样式对所有的主刻度线和主刻度值标签作设置，可以在这个样式中使用选项 `style=`, `node style=` 来分别设置主刻度线和主刻度值标签。主刻度线的默认设置是

```
style={line cap=round}, tick length=2pt
```

`/tikz/data visualization/every minor ticks` (style, 无值)

这个样式对所有的副刻度线作设置，可以在这个样式中使用选项 `style=` 设置副刻度线，默认设置是

```
style={help lines, thin, line cap=round}, tick length=1.4pt
```

`/tikz/data visualization/every subminor ticks` (style, 无值)

这个样式对所有的次副刻度线作设置，可以在这个样式中使用选项 `style=` 设置次副刻度线，默认设置是

```
style={help lines, line cap=round}, tick length=0.8pt
```

`/tikz/data visualization/tick layer` (style, 初始值 on background layer)

类似 `grid layer` 选项，指定刻度线所在的“层”，这个样式中的选项被冠以 `/tikz` 来执行。

`/tikz/data visualization/tick node layer` (style, 初始值空)

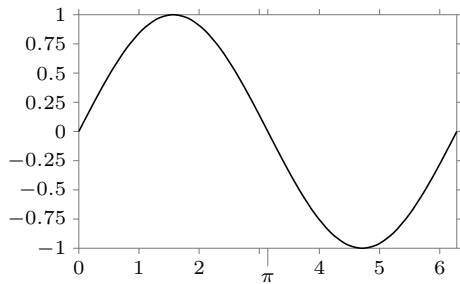
类似 `grid layer` 选项，指定刻度值标签所在的“层”，这个样式中的选项被冠以 `/tikz` 来执行。

77.4.10 设置个别刻度的样式

有时候图形中的某些点有特殊的意义，这些点对应的刻度值或刻度线需要个别设置，此时可以用下面的选项：

`/tikz/data visualization/options at=<value> as[<options>]`

<value> 指定坐标轴上相应的刻度位置，`as` 是个算子，将 <options> 添加到这些位置上的刻度线或刻度值标签中来执行，改变刻度线或刻度值标签的外观。



```
\tikz \datavisualization [scientific axes,
  visualize as smooth line,
  x axis={ticks={major={
    options at = 3 as [no tick text],
    also at = (pi) as
      [{tick text padding=1ex}
        $\pi$}}}]}
  data [format=function] {
    var x : interval[0:2*pi];
    func y = sin(\value x r);
  };
```

`/tikz/data visualization/no tick text at=<value>`

这是 `options at=<value> as [no tick text]` 的简写。

77.4.11 其它刻度值标签选项

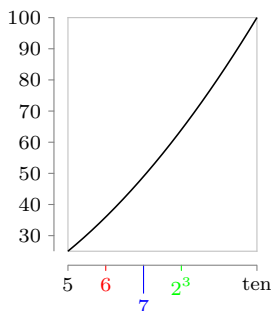
下面的选项可用于设置刻度值标签的样式。

`<value> as [<local options>]`

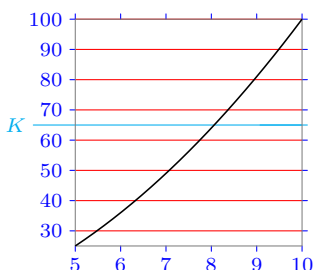
`<value> as [<local options>] <text>`

这里 `as` 是个算子。当使用 `at={<value>...}`, `also at={<value>...}` 指定刻度位置时，在 <value> 后加 `as` 算子，可以改变相应的刻度值标签的样式。<local options> 是针对标签样式的选项。如果 `as` 算子之后有 <text>，则标签将显示为 <text>。

<local options> 中可用选项 `style=`, `node style=` 来分别设置该位置处的刻度线、刻度值标签、网格线的样式，可以用 `low`, `high` 选项。



```
\tikz \datavisualization [scientific axes=clean,
  x axis={length=2.5cm, ticks={major at={
    5,
    6 as [style=red],
    7 as [{style=blue, low=-1em}],
    8 as [style=green] $2^3$,
    10 as ten
  }}}},
  visualize as line]
data [format=function] {
  var x : interval [5:10];
  func y = \value x * \value x;
};
```



```
\tikz \datavisualization [scientific axes,
  all axes={ticks={major={style=blue}}, length=3cm},
  y axis={grid,
    grid={major={style=red},
      major also at={65 as[style=cyan]}},
    ticks={major also at={
      65 as[low=-1.5em,style=cyan]$K$}}},
  visualize as line]
data [format=function] {
  var x : interval [5:10];
  func y = \value x * \value x;
};
```

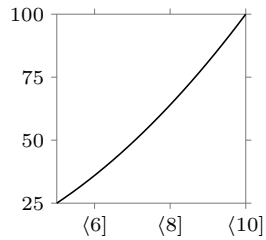
上面的例子中，用选项 `major also at=`，算子 `as` 设置了一个特殊的主网格线、主刻度。

如果数值 `0.000000015` 作为刻度值标签，一般情况下，它会被显示为 $1.5 \cdot 10^{-9}$ ，也就是说，可视化系统并不是简单地将数值排版为文本标签，而是先将标签数值传递给一个 `typesetter`，它会利用 `TeX` 的排版功能来编辑标签，除非使用 `as[...]<text>` 来指定标签文本。

刻度值标签通常有 3 个部分，前缀、标签文本、后缀（例如物理单位）：

`/tikz/data visualization/tick prefix=<text>` （无默认值，初始值 `empty`）

这个选项设置刻度值标签的前缀。



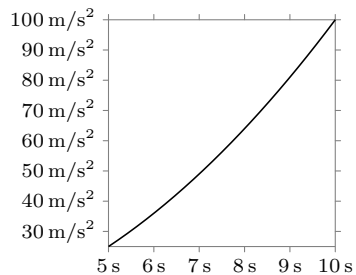
```
\tikz \datavisualization [scientific axes,
    all axes={ticks=few, length=2.5cm},
    x axis={ticks={tick prefix=$\langle$,
        tick suffix=$\rangle$}},
    visualize as line]
data [format=function] {
    var x : interval [5:10];
    func y = \value x * \value x;
};
```

`/tikz/data visualization/tick suffix=<text>` (无默认值, 初始值 empty)

这个选项设置刻度值标签的后缀。

`/tikz/data visualization/tick unit=<roman math text>`

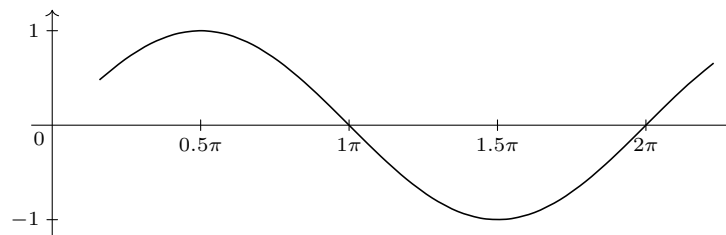
这是 `tick suffix={\rm<roman math text>}` 的简写, `<roman math text>` 自动进入数学模式。



```
\tikz \datavisualization [scientific axes,
    all axes={length=3cm},
    x axis={ticks={tick unit=s}},
    y axis={ticks={tick unit=m/s^2}},
    visualize as line]
data [format=function] {
    var x : interval [5:10];
    func y = \value x * \value x;
};
```

`/tikz/data visualization/tick typesetter`

用于定义数值标签编辑器 `typesetter`, 在默认下该选项使用命令 `\pgfmathprintnumber` 将数值标签显示为科学计数法格式。

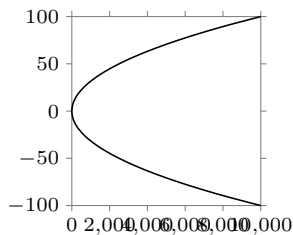


```
\def\mytypesetter#1{%
    \pgfmathparse{#1/pi}%
    \pgfmathprintnumber{\pgfmathresult}$\pi$%
}
```

```
\tikz \datavisualization [school book axes,
  all axes={unit length=1.25cm},
  x axis={ticks={step=(0.5*pi), tick typesetter/.code=\mytypesetter{##1}}},
  y axis={include value={-1,1}},
  visualize as smooth line]
data [format=function] {
  var x : interval [0.5:7];
  func y = sin(\value x r);
};
```

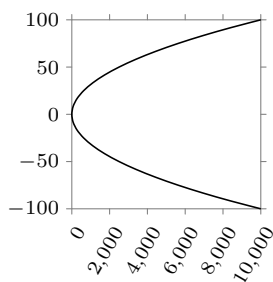
77.4.12 交错叠放刻度值标签

有时候刻度值标签过长以致发生重叠，例如



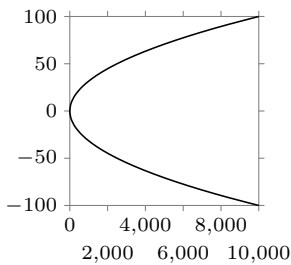
```
\tikz \datavisualization [scientific axes,
  all axes={length=2.5cm},
  visualize as smooth line]
data [format=function] {
  var y : interval[-100:100];
  func x = \value y*\value y;
};
```

解决这种问题的办法有两个，一个是旋转标签：



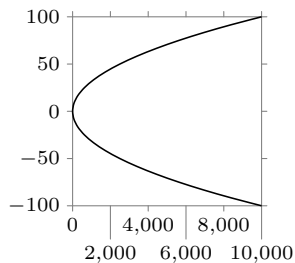
```
\tikz \datavisualization [scientific axes,
  all axes={length=2.5cm},
  x axis={ticks={node style={rotate=60,
    anchor=north east}}},
  visualize as smooth line]
data [format=function] {
  var y : interval[-100:100];
  func x = \value y*\value y;
};
```

另一个办法是将标签交错叠放：



```
\tikz \datavisualization [scientific axes,
    all axes={length=2.5cm},
    x axis={ticks={major at={0,4000,8000,
        2000 as [node style={yshift=-1em}],
        6000 as [node style={yshift=-1em}],
        10000 as [node style={yshift=-1em}]}}}],
    visualize as smooth line]
data [format=function] {
    var y : interval[-100:100];
    func x = \value y*\value y;
};
```

实现这种叠放效果的便捷办法是使用 `stack` 选项:



```
\tikz \datavisualization [scientific axes,
    all axes={length=2.5cm},
    x axis={ticks=stack},
    visualize as smooth line]
data [format=function] {
    var y : interval[-100:100];
    func x = \value y*\value y;
};
```

刻度值标签按照奇数、偶数位置交错叠放。程序会考虑所有的主刻度、副刻度、次副刻度，按它们在代码中出现的次序来确定刻度的奇数、偶数位置，而不是按照它们在坐标轴上的左右位置来确定刻度的奇偶位置。例如，

```
ticks={major at={1,2,3,4}, major also at={0,-1,-2}, minor at={9,8,7}}
```

其中的 1,3,0,-2,9,7 处于奇数位置。

当刻度值标签偏移后，相应的刻度线也会变长。

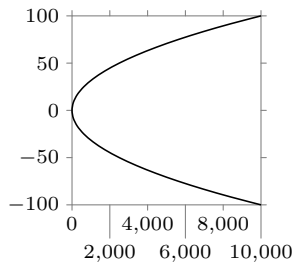
在绘图时，在图形的上部和下部都可能出现横轴，上下横轴都可能带有刻度。横轴刻度线的下端点对应单词“low”，刻度线的上端点对应单词“high”；对于预定义的坐标系统，在默认下，下横轴的刻度值标签位于刻度线的下端点之下，也就是说，作为 node 的刻度值标签的指向点是刻度线的下端点，标签的 anchor 位置是 north；上横轴的刻度值标签位于刻度线的上端点之上，即作为 node 的刻度值标签的指向点是刻度线的上端点，标签的 anchor 位置是 south。

图形中也可能会有左纵轴和右纵轴，情况是类似的。纵轴刻度线的左端点对应单词“low”，纵轴刻度线的右端点对应单词“high”；对于预定义的坐标系统，在默认下，左纵轴的刻度值标签位于刻度线的左端点之左，右纵轴的刻度值标签位于刻度线的右端点之右。

横轴刻度线的下端点相对于横轴的偏移尺寸由 `low=<dimension>` 确定；横轴刻度线的上端点相对于横轴的偏移尺寸由 `high = <dimension>` 确定。纵轴刻度线的左端点相对于纵轴的偏移尺寸由 `low=<dimension>` 确定；纵轴刻度线的右端点相对于纵轴的偏移尺寸由 `high = <dimension>` 确定。

`/tikz/data visualization/tick text low even padding=<dimension>` (无默认值, 初始值 0pt)

这个选项针对下横轴或者左纵轴, 因为这两个轴的刻度值标签处于刻度线的“low”端点的 low 侧。该选项确定轴的偶数位置刻度线的“low”端点相对于轴的偏移尺寸, 同时使得偶数位置的刻度值标签随之偏移, 从而使得奇偶位置的刻度值标签发生错位。例如对于下横轴来说, 若 `<dimension>` 是足够大的带单位的负值尺寸, 则该选项使得偶数位刻度值标签处于奇数刻度值标签之下。如果刻度还有 `low=` 选项, 则 `<dimension>` 会叠加到 `low=` 的值上。



```
\tikz \datavisualization [scientific axes,
  all axes={length=2.5cm},
  x axis={ticks={tick text low even padding=-1em}},
  visualize as smooth line]
data [format=function] {
  var y : interval[-100:100];
  func x = \value y*\value y;
};
```

`/tikz/data visualization/tick text low odd padding=<dimension>` (无默认值, 初始值 0pt)

`/tikz/data visualization/tick text high even padding=<dimension>` (无默认值, 初始值 0pt)

这个选项针对上横轴或右纵轴, 因为这两个轴的刻度值标签处于刻度线的“high”端点的 high 侧。该选项指定轴的偶数位置刻度线的“high”端点相对于轴的偏移尺寸, 同时使得偶数位置的刻度值标签随之偏移, 从而使得奇偶位置的刻度值标签发生错位。

`/tikz/data visualization/tick text high odd padding=<dimension>`

`/tikz/data visualization/tick text odd padding=<dimension>`

同时设置 `tick text odd low padding` 和 `tick text odd high padding`.

`/tikz/data visualization/tick text even padding=<dimension>`

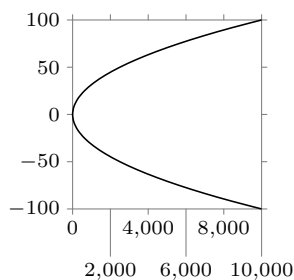
同时设置 `tick text even low padding` 和 `tick text even high padding`.

`/tikz/data visualization/tick text padding=<dimension>`

同时设置所有刻度值标签的 `text padding`, 相当于给 (有刻度值标签的) 刻度线附加一个长度尺寸, 改变刻度线的长短。

`/tikz/data visualization/stack=<dimension>` (默认值 1em)

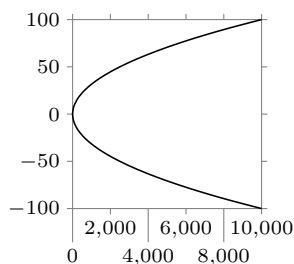
这是 `tick text even padding=<dimension>` 的简写。



```
\tikz \datavisualization [scientific axes,
  all axes={length=2.5cm},
  x axis={ticks={stack=1.5em}},
  visualize as smooth line]
data [format=function] {
  var y : interval[-100:100];
  func x = \value y*\value y;
};
```

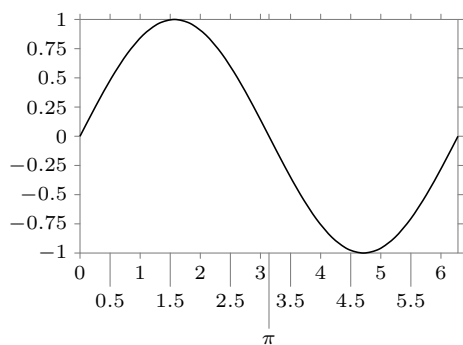
`/tikz/data visualization/stack'=<dimension>` (无默认值)

这是 `tick text odd padding=<dimension>` 的简写。



```
\tikz \datavisualization [scientific axes,
  all axes={length=2.5cm},
  x axis={ticks=stack'},
  visualize as smooth line]
data [format=function] {
  var y : interval[-100:100];
  func x = \value y*\value y;
};
```

个别设置的有特殊意义的刻度值标签也可以有交错叠放效果:



```
\tikz \datavisualization [scientific axes,
  x axis={ticks={stack, many, major also at=
    {(pi) as [{tick text padding=2.5em}]
      $\pi$}},
  visualize as smooth line]
data [format=function] {
  var x : interval[0:(2*pi)];
  func y = sin(\value x r);
};
```

77.4.13 自动添加刻度的策略

在 §77.4.3 中提到两种自动添加刻度的策略: `linear steps` 和 `exponential steps`.

`/tikz/data visualization/axis options/linear steps`

这是默认的策略。

`/tikz/data visualization/axis options/exponential steps`

77.4.14 定义新的添加刻度的策略

77.5 创建新的轴系统

§77.2.8 中讲到用选项 `new Cartesian axis=` 创建（不显示的）仿射轴，§77.3.4 中讲到底层的（默认的、不显示的）笛卡尔轴系统，本小节讲解如何将这些不显示的轴变为可见，以及如何调整这些轴的外观，例如，刻度，刻度值标签，网格线，轴标签等等。

首先注意，一个轴系统并不直接与数据的变化范围相关联，也没有刻度、网格线等可视化内容，数据范围、刻度等内容都是用其它算子或选项添加到轴系统中的。

创建一个新的轴系统并将它可视化，主要包括 4 方面：

1. 坐标轴的可视化。首先区分“轴”与“可视化的轴”。轴是一套处理过程，并不是可见的。以直角坐标系的横轴为例，假设这个轴对应一个变量 x ， x 的范围就是该轴的范围。最初该轴是不可见的，打个比方，有个“显像器”可以决定横轴的哪一部分能够被可视化。在默认下，“显像器”将整个横轴可视化为一个线段，线段左端点对应变量 x 的最小值，线段右端点对应变量 x 的最大值。但是可以用某些选项来改变“显像器”的作用范围，把横轴的某个子集可视化为一个线段，其余部分尽管不显示，但在图形中仍然占据空白位置，因为“显像器”不改变可视化数据区域，也就是说，轴可以是“间断地显示的”。在这个意义上，“轴”与“可视化的轴”是不同的。

自定义坐标轴的可视化由选项 `visualize axis` 来设置，涉及轴的显示范围、可视化轴的位置、颜色、箭头、线宽等内容。

2. 网格线的可视化。自定义坐标轴的网格线的可视化由选项 `visualize grid` 来辖制，该选项指定默认的网格线的外观，例如网格线的方向、起点、终点、线型、颜色、线宽等。在可视化绘图命令中，要画出网格线只需给轴使用 `grid` 选项，画出的网格线的外观会按选项 `visualize grid` 所指定的样式，也可以临时修改。
3. 刻度的可视化。自定义坐标轴的刻度包括刻度线和刻度值标签两方面，其可视化由选项 `visualize ticks` 来辖制，该选项指定默认的刻度线、标刻度签线的外观，例如刻度的方向、起点、终点、颜色、标签样式等。在可视化绘图命令中，要出刻度只需给轴使用 `ticks` 选项，画出的刻度的外观会按选项 `visualize ticks` 所指定的样式，也可以临时修改。
4. 坐标轴标签的可视化。自定义坐标轴的标签的可视化由选项 `visualize label` 来辖制，该选项指定默认的轴标签的外观，例如标签的位置、样式等。在可视化绘图命令中，要出轴标签只需给轴使用 `label` 选项，画出的刻度的外观会按选项 `visualize label` 所指定的样式，也可以临时修改。

在前文多处涉及选项 `low` 和 `high`，这两个选项在后文会做解释，它们在不同的地方有不同的意义和作用，其意义主要有 3 个：

1. 对轴的可视化来说，即用作选项 `visualize axis` 的值时，`low` 对应可视化轴的起点，`high` 对应可视化轴的终点。此时二者的格式是 `low=<value>`，`high=<value>`，其中 `<value>` 是该轴对应的变量值区间内的值。
2. 对于网格线的可视化来说，即用作选项 `visualize grid` 的值时，`low` 对应网格线的起点，`high` 对应网格线的终点。此时二者的格式是 `low=<value>`，`high=<value>`，其中 `<value>` 是网格线的“方向轴”所对应的变量值区间内的值。
3. 对于刻度线的可视化来说，即用作选项 `visualize ticks` 的值时，`low` 对应刻度线的起点，`high` 对应刻度线的终点。此时二者的格式是 `low=<dimension>`，`high=<dimension>`，其中 `<dimension>` 是

带单位的正值或负值尺寸。

77.5.1 创建一个轴系统

下面通过一个例子来说明如何创建轴系统。下面例子中，轴系统是笛卡尔式的，名称是 `our system`，其中包括 3 个轴，一个水平轴 `x axis`，一个左侧纵轴 `left axis`，一个右侧纵轴 `right axis`。

一个轴系统是作为 key 被创建的，其前缀是 `/tikz/data visualization`，例如：

```
\tikzset{
  data visualization/our system/.style={
    ...
  }
}
```

这个句子创建一个名称为 `our system` 的轴系统。然后在该轴系统中定义 3 个轴：

```
\tikzset{
  data visualization/our system/.style={
    new Cartesian axis=x axis,
    new Cartesian axis=left axis,
    new Cartesian axis=right axis,
    x axis={attribute=x},
    left axis={unit vector={(0cm,1pt)}},
    right axis={unit vector={(0cm,1pt)}},
  }
}
```

其中的 `x axis` 是系统预定义的笛卡尔系的横轴名称（见 §77.3.4）；用单位向量来分别规定左右纵轴 `left axis` 和 `right axis`；轴 `x axis` 对应变量名称是 `x`；轴 `left axis` 和 `right axis` 尚未对应变量。

再规定轴的长度，例如使用科学坐标系的轴长度：

```
x axis ={length=\pgfkeysvalueof{/tikz/data visualization/scientific axes/width}},
left axis ={length=\pgfkeysvalueof{/tikz/data visualization/scientific axes/height}},
right axis={length=\pgfkeysvalueof{/tikz/data visualization/scientific axes/height}}
```

其中 `/tikz/data visualization/scientific axes/width` 是科学坐标系的横轴的长度，初始值为 5cm；`/tikz/data visualization/scientific axes/height` 是科学坐标系的纵轴的高度，与横轴长度成黄金分割比。

通过以上设置就定义了一个轴系统，可以调用这个轴系统来画图了。其中各个轴对应的变量名称，单位向量，轴长度等项目，都可以在绘图时做临时修改。

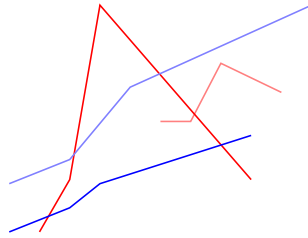
```
\tikzset{
  data visualization/our system/.style={
    new Cartesian axis=x axis,
    new Cartesian axis=left axis,
    new Cartesian axis=right axis,
```

```
x axis={attribute=x, length=\pgfkeysvalueof{/tikz/data visualization/scientific
axes/width}},
left axis={unit vector={(0cm,1pt)}, length=\pgfkeysvalueof{/tikz/data
visualization/scientific axes/height}},
right axis={unit vector={(0cm,1pt)}, length=\pgfkeysvalueof{/tikz/data
visualization/scientific axes/height}},
}
}

\tikz \datavisualization data group {people and money} = {
  data [set=people 1] {
    time, people
    1900, 1000000000
    1920, 1500000000
    1930, 2000000000
    1980, 3000000000
  }
  data [set=people 2] {
    time, people
    1900, 2000000000
    1920, 2500000000
    1940, 4000000000
    2000, 5700000000
  }
  data [set=money 1] {
    time, money
    1910, 1.1
    1920, 2
    1930, 5
    1980, 2
  }
  data [set=money 2] {
    time, money
    1950, 3
    1960, 3
    1970, 4
    1990, 3.5
  }
}
```

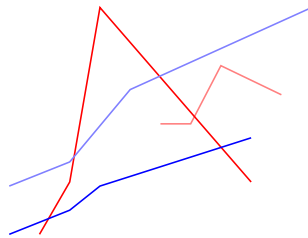
```
};
```

用上面代码定义的 `our system` 画出这个 `data group`, 并且临时修改轴对应的变量名、轴的长度:



```
\tikz \datavisualization [
  our system,
  x axis={attribute=time, length=4cm},
  left axis ={attribute=money, length=3cm},
  right axis={attribute=people, length=3cm},
  visualize as line/.list={people 1, people 2, money 1, money 2},
  people 1={style={visualizer color=blue}},
  people 2={style={visualizer color=blue!50}},
  money 1={style={visualizer color=red}},
  money 2={style={visualizer color=red!50}}]
data group {people and money};
```

也可以将轴系统的定义放在命令 `\datavisualization` 的选项中:



```
\tikz \datavisualization [
  our system/.style={
    new Cartesian axis=x axis,
    new Cartesian axis=left axis,
    new Cartesian axis=right axis,
    left axis={unit vector={(0cm,1pt)}},
    right axis={unit vector={(0cm,1pt)}},
  },
  our system,
  x axis={attribute=time, length=4cm},
  left axis ={attribute=money, length=3cm},
```

```
right axis={attribute=people, length=3cm},
visualize as line/.list={people 1, people 2, money 1, money 2},
people 1={style={visualizer color=blue}},
people 2={style={visualizer color=blue!50}},
money 1={style={visualizer color=red}},
money 2={style={visualizer color=red!50}}]
data group {people and money};
```

77.5.2 坐标轴的可视化

如果要将坐标轴的某个要素，例如坐标轴的位置，可视化区间，坐标轴的颜色、线型、线宽等（但不包括刻度、网格线），可视化，就必须将该要素作为选项 `visualize axis` 的值，也就是说这个选项辖制轴的可视化内容，但是不辖制像 `axis layer, every axis` 这种 `style` 选项。

在一个可视化过程中，如果某个轴的选项中多次使用 `visualize axis` 选项，那么它们的效果会累计。

```
/tikz/data visualization/axis options/visualize axis=<options>
```

这个 `key` 作为选项传递给轴，直接使得轴可视化。`<options>` 中可以使用选项 `style=` 来设置可视化轴的外观，也可用下面的选项来决定轴画在什么位置以及轴的长度。轴的位置是利用两个轴的相对位置来确定的。例如对于前面的定义的轴系统 `our system` 来说，可以规定轴 `left axis` 与轴 `x axis` 的相对位置为：

```
left axis={ visualize axis={ x axis={ goto=min } }
```

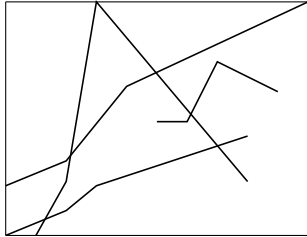
这个句子指定轴 `left axis` 经过轴 `x axis` 上刻度值为 `min` 的点，即轴 `x axis` 对应的变量值集的最小值点。

可视化坐标轴相对的位置，轴的可视化区间 如前述，轴的位置相互确定。

```
/tikz/data visualization/axis options/goto=<value>
```

这个选项的作用如前述。`<value>` 是个属于变量值集的数字，也可以是下述之一：

- `min`，轴所对应的变量值集的最小值点。
- `max`，轴所对应的变量值集的最大值点。
- `padded min`，在 `min` 的基础上再附加一个偏移尺寸所确定的点，偏移尺寸由选项 `padding min=<dimension>` 确定，见后文。
- `padded max`，类似 `padded min`。

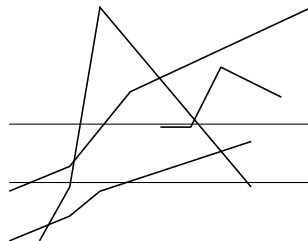


```
\tikzset{
  data visualization/our system/.append style={
    left axis={visualize axis={x axis={goto=min}}},
    right axis={visualize axis={x axis={goto=max}}},
    x axis={visualize axis={left axis={goto=min}},
            visualize axis={left axis={goto=max}}},
  }
}
\tikz \datavisualization [
  our system,
  x axis={attribute=time, length=4cm},
  left axis={attribute=money},
  right axis={attribute=people},
  visualize as line/.list={people 1, people 2,
                           money 1, money 2}]
data group {people and money};
```

上面的例子中，横轴 `x axis` 里面两次使用 `visualize axis` 选项，使得横轴画了两次，一次经过纵轴 `left axis` 的上端，一次经过纵轴 `left axis` 的下端。

`/tikz/data visualization/axis options/goto pos=<fraction>`

这个选项类似 `goto`，如果设置 `scaling=min at 0 and max at 1`，即“合理区间”是 $[0,1]$ ，那么 `<fraction>` 决定的位置是 $0 + \text{fraction} * (1 - 0)$ ，也就是说，`<fraction>` 是针对合理区间来计算的，与变量值集无关。`<fraction>` 可以是任意小数，不必限于 0 到 1 之间。



```
\tikzset{
  data visualization/our system/.append style={
    x axis={visualize axis={left axis={goto pos=0.25}},
            visualize axis={left axis={goto pos=0.5}}},
  }
}
\tikz \datavisualization [
  our system,
  x axis={attribute=time, length=4cm},
```

```

left axis ={attribute=money},
right axis={attribute=people},
visualize as line/.list={people 1, people 2,
                        money 1, money 2}]
data group {people and money};

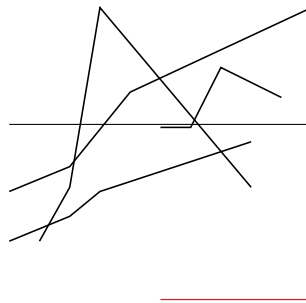
```

```
/tikz/data visualization/low=<value>
```

这个选项用作 `visualize axis` 的值，或者用于其它可视化过程中。“轴”与“可视化的轴”的区别在本节开头已经解释过了，在这里单词 `low` 指的是可视化轴的左端点，`<value>` 是该左端点对应的变量值，故本选项指定轴上的可视化轴的左端点。本选项与 `min value=<value>` 不同 (§77.2.3)，本选项不会改变“合理区间”（即不改变可视化数据的区域），只是用来决定哪一部分区间能被可视化。

其中的 `<value>` 可以是 `min`, `max`, `padded min`, `padded max` 等，也可以是某个变量值（不是“合理区间”中的点），默认为 `low=min`。

继续沿用前面的例子，但是将 `x axis` 轴的显示区间由 `[1900,2000]` 改为 `[1950,2000]`，位置在纵轴下端之下 (`goto pos=-0.25`)，颜色为红色，如下所示：



```

\tikzset{
  data visualization/our system/.append style={
    x axis= {visualize axis={left axis={goto pos=-0.25}, style=red, low=1950},
            visualize axis={left axis={goto pos=0.5}}},
  }
}
\tikz \datavisualization [
  our system,
  x axis={attribute=time, length=4cm},
  left axis ={attribute=money},
  right axis={attribute=people},
  visualize as line/.list={people 1, people 2, money 1, money 2}]
data group {people and money};

```

从上面的例子可见，对于横轴来说，`low=` 引起横轴的水平方向上可视化区间的左端点变化。类似地，对于纵轴来说，`low=` 引起纵轴的竖直方向上可视化区间的下端点变化。

选项 `low=` 还可以用于规定网格线、刻度线的起点，在 §77.4.11 中讲解 `as` 算子时有个例子，在 §77.5.3

中还有例子。

```
/tikz/data visualization/high=<value>
```

与 `low` 类似，只是决定轴的可视化部分的另一端点的变化，默认为 `high=max`。

```
/tikz/data visualization/padded
```

该选项同时设置 `low=padded min` 和 `high=padded max`。

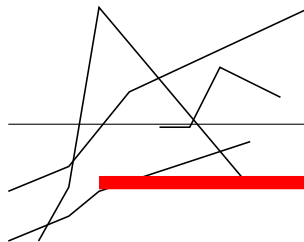
对于仿射坐标系，轴的可视化区间线段是利用 `\pgfpathdvmoveto` 和 `\pgfpathdvlineto` 来构造的。对于极坐标系，某一半径的角度轴是用 `arc` 构造的。

轴的样式 可以用 `style=` 选项 (§77.4.7) 设置轴的颜色、线型、箭头等外观，只需将 `style={<TikZ options>}` 放在 `visualize axis={}` 中。此外还有以下两个样式选项：

```
/tikz/data visualization/axis layer (style, 初始值 on background layer)
```

与 `grid layer` 类似，这个选项决定可视化的轴画在那个“层”上。因为这个选项针对所有的轴，所以只能直接用作命令 `\datavisualization` 的选项或者用在 `\tikzset{...}` 中，不受选项 `visualize axis` 的辖制，例如

```
\tikzset{
  data visualization/our system/.append style={
    axis layer/.style=,
    .....
  }
}
```



```
\tikzset{
  data visualization/our system/.append style={
    x axis= {visualize axis={left axis={goto pos=0.25}, style={red,line width=2pt
    }, low=1930},
    visualize axis={left axis={goto pos=0.5}}},
  }
}
\tikz \datavisualization [
  our system,
  axis layer/.style=, % 坐标轴处于顶层，遮挡图形
  x axis={attribute=time, length=4cm},
  left axis ={attribute=money},
  right axis={attribute=people},
```

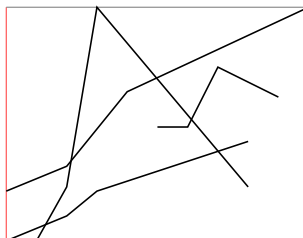


```
visualize as line/.list={people 1, people 2, money 1, money 2}]
data group {people and money};
```

`/tikz/data visualization/every axis` (style, 无值)

用来设置某个轴的初始样式，直接用作命令 `\datavisualization` 的选项或者用在 `\tikzset{...}` 中，不受选项 `visualize axis` 的辖制，例如

```
every axis/.style={style=black!50}
```



```
\tikzset{
  data visualization/our system/.append style={
    every axis/.style={style=black!50},
    left axis={visualize axis={x axis={goto=min}, style=red!75}},
    right axis={visualize axis={x axis={goto=max}, style=blue!75}},
    x axis={visualize axis={left axis={goto=min}},
    visualize axis={left axis={goto=max}}}},
  }
}
\tikz \datavisualization [
  our system,
  x axis={attribute=time, length=4cm},
  left axis={attribute=money},
  right axis={attribute=people},
  visualize as line/.list={people 1, people 2, money 1, money 2}]
data group {people and money};
```

选项 `styling` 也是可用的，参考 §77.4.7.

坐标轴的偏移 对一般的 `scientific axes` 坐标系 (§77.3.2)，坐标轴是相交的；对于 `scientific axes = clean`，坐标轴是相离的，即轴发生偏移。前面的选项 `goto` 用变量值设置两个轴的相对位置，`goto pos` 用一个小数设置两个轴的相对位置，在这两个选项设置的基础上，选项 `padded min`，`padded max`，`padded` 进一步指定轴做偏移。用带单位的尺寸指定轴的偏移量有时要比用变量值更方便，所以这些选项是相互补充、各有优点的。

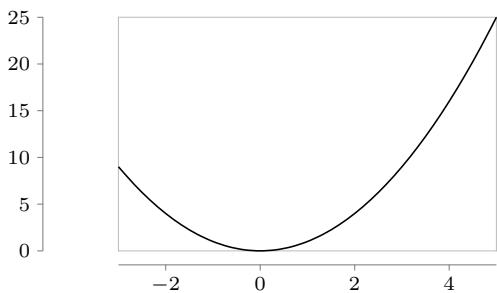
为了方便，约定一个称呼：如果一个轴的位置是参照另一个轴（用选项 `goto` 等）确定的，这“另一个轴”称为该轴的“参照轴”。

两个坐标轴的相对偏移尺寸用下面的选项设置。

`/tikz/data visualization/axis options/padding min=<dimension>`

这个选项指定一个偏离尺寸 `<dimension>`，用于来配合选项 `goto=padded min`，指的是轴相对于 `min` 位置偏离的尺寸。轴的偏移方向与 `<dimension>` 的正负符号、参照轴的方向有关（按通常的数学上的理解）。通常 `<dimension>` 应当为带单位的负值尺寸。

对于标准的预定义坐标系统 `scientific axes=clean`，有个默认的偏离尺寸，如果再使用这个选项指定一个尺寸 `<dimension>`，则偏离会叠加，而且纵轴与横轴不相交，有灰色框线来标示绘图区域。但是对于自定义坐标系，坐标轴偏移后，如果没有其它设置，纵轴与横轴仍然相交，也没有灰色框线来标示绘图区域。



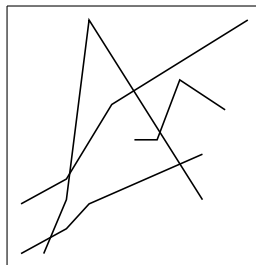
```
\begin{tikzpicture}
\datavisualization [scientific axes=clean,
    x axis={padding min=-1cm},
    visualize as smooth line]
data [format=function] {
    var x : interval [-3:5];
    func y = \value x * \value x;
};
\end{tikzpicture}
```

`/tikz/data visualization/axis options/padding max=<dimension>`

这个选项与 `padding min=` 类似，指定一个偏离尺寸 `<dimension>`，用于来配合选项 `goto=padded max`，指的是轴相对于 `max` 位置偏离的距离。注意 `<dimension>` 应当为带单位的正值尺寸，这样才会使得两个轴相离，否则可能相交。

`/tikz/data visualization/axis options/padding=<dimension>`

这里 `<dimension>` 应当是带单位的正值尺寸，这个选项会同时设置 `padding max=<dimension>` 和 `padding min=-<dimension>`。



```
\tikzset{
    data visualization/our system/.append style={
        all axes= {padding=.5em},
        left axis= {visualize axis={x axis= {goto=padded min}, padded}},
    }
}
```

```

    right axis={visualize axis={x axis= {goto=padded max}, padded}},
    x axis= {visualize axis={left axis={goto=padded min}, padded},
    visualize axis={left axis={goto=padded max}, padded}},
  }
}
\tikz \datavisualization [
  our system,
  x axis={attribute=time, length=3cm},
  left axis ={attribute=money},
  right axis={attribute=people},
  visualize as line/.list={people 1, people 2, money 1, money 2}]
data group {people and money};

```

77.5.3 可视化网格线

选项 `visualize grid` 辖制网格线的可视化，例如网格线的位置、颜色、线型等。如果在某个轴的选项中多次使用该选项，则它们的效果会被累计。

```
/tikz/data visualization/axis options/visualize grid=<options>
```

这个 key 作为轴的选项，导致该轴的网格线可视化，`<options>` 决定网格线的默认外观，如起止点、颜色等，其中可用 `style=` 选项。该选项完成设置后，在可视化命令中给轴使用 `grid` 选项会按设置画出网格线，也可临时修改网格线的外观。

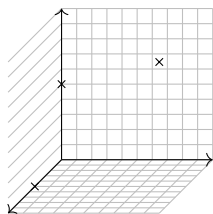
网格线的方向 “网格线”应该是对坐标空间的一种分割，对于不同的坐标系“网格线”有不同的含义。对于平面仿射坐标系，各个轴的网格线就是一组平行线段。对于平面极坐标系，网格线就是一组射线或者一组同心圆弧。对于 3 维直角坐标系，分割空间的是平面，所以一个轴的“网格线”应该是一组与该轴正交的平面，但是在绘图时最好还是用线段来画网格线。对于 3 维球坐标系，分割空间的是纬平面和经平面，但最好用经纬网做网格线。

在一个可视化过程中，如果某个轴的选项中多次使用 `visualize grid` 选项，那么它们的效果会累计。

对于 3 维直角坐标系的 x 轴，其网格线有两个方向，一个方向是 y 轴方向，另一个方向是 z 轴方向，用下面的 key 指定网格线的方向：

```
/tikz/data visualization/direction axis=<axis name>
```

这个选项规定网格线的方向与轴 `<axis name>` 的方向一致。



```

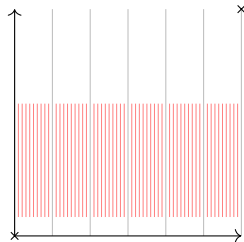
\begin{tikzpicture}
\tikz \datavisualization [
  xyz Cartesian cabinet,
  all axes={visualize axis={low=0, style=->}, grid=many},
  x axis={visualize grid={direction axis=z axis},
    visualize grid={direction axis=y axis}},
  y axis={visualize grid={direction axis=x axis},
    visualize grid={direction axis=z axis}},
  z axis={visualize grid={direction axis=x axis}},
  visualize as scatter]
data {
  x, y, z
  0, 0, 1
  0, 1, 0
  2, 2, 2
};
\end{tikzpicture}

```

网格线的外观 参考 §77.4.7, §77.4.8.

主网格线、副网格线、次副网格线的外观、起止点 §77.4.7 中讲了可以在 `grid=` 中使用 `style=` 设置网格线的外观。§77.4.8 中讲了 `grid layer`, `every grid`, `every major grid`, `every minor grid`, `every subminor grid`, 这些都是样式 (`style`), 应该直接用作命令 `\datavisualization` 的选项或者用在 `\tikzset{...}` 中。在默认下, 这些样式已经对网格线的初始状态做了比较合理地设置, 如无特别需要, 可以不做修改。

选项 `major`, `minor`, `subminor`, `common` (§77.4.5) 可以用做 `grid=` 的值, 也可以用做 `visualize grid=` 的值, 分别设置主网格线、副网格线、次副网格线的外观、起止位置。设置网格线的起止位置要用到选项 `low=<value>` 和 `high=<value>`, 这里的 `<value>` 是网格线的方向轴 (与网格线平行的轴) 上的变量值, 注意此时的 `<value>` 不能换成带单位的尺寸 `<dimension>`, 否则无法读取尺寸, 导致错误。



```

\tikz \datavisualization [
  xy Cartesian,
  all axes={visualize axis={low=0, style=->}, grid={some, minor steps between steps}},

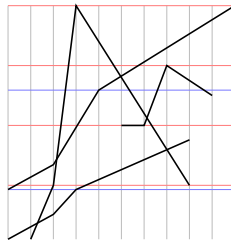
```

```

x axis= {visualize grid={direction axis=y axis, minor={low=0.25, high=1.75, style=red
!50}}}, % 注意这里对横轴的副网格线的规定
visualize as scatter]
data {
  x, y
  0, 0
  3, 3
};

```

再用前面定义的 `our system` 作一个图:



```

\tikzset{
  data visualization/our system/.append style={
    x axis= {visualize grid={direction axis=left axis}},
    left axis= {visualize grid={direction axis=x axis, common={style=red!50}}},
    right axis={visualize grid={direction axis=x axis, common={style=blue!50}}},
  }
}
\tikz \datavisualization [
  our system,
  x axis={attribute=time, length=3cm, grid=many},
  left axis ={attribute=money, grid=some},
  right axis={attribute=people, grid=few},
  visualize as line/.list={people 1, people 2, money 1, money 2}]
data group {people and money};

```

77.5.4 刻度线、刻度值标签的可视化

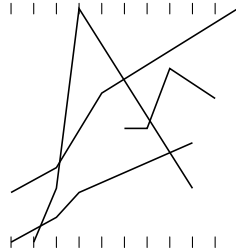
刻度包括刻度线、刻度值标签 (node) 两方面, 需要用到选项 `visualize ticks`, 这个选项辖制刻度的可视化, 例如刻度的位置、颜色等。如果在某个轴的选项中多次使用该选项, 则它们的效果会被累计。

```
/tikz/data visualization/axis options/visualize ticks=<options>
```

与选项 `visualize grid` 类似。可以在 `<options>` 使用选项 `style=` 设置刻度线的外观; 使用选项 `node style=` 设置刻度值标签的外观, `anchor` 位置等。在可视化命令中, 给轴使用选项 `ticks` 可以按设置画出刻度, 也可以临时修改刻度的外观。

刻度线可以看作是“迷你版”的网格线，但是与网格线不同的是，刻度线总是直的，不会弯曲，而极坐标网格、经纬网中的网格线都有弯曲。

再用前面定义的 `our system` 作图：



```
\tikzset{
  data visualization/our system/.append style={
    x axis={visualize ticks={direction axis=left axis, left axis={goto=min}},
           visualize ticks={direction axis=left axis, left axis={goto=max}},
    }
  }
}
\tikz \datavisualization [
  our system,
  x axis={attribute=time, length=3cm, ticks=many},
  left axis ={attribute=money},
  right axis={attribute=people},
  visualize as line/.list={people 1, people 2, money 1, money 2}]
data group {people and money};
```

上面例子中，图形的上部和下部都有横轴 `x axis`，它们的刻度线一样的。如果想要把刻度线改短，需要用下面的选项：

```
/tikz/data visualization/direction axis=<axis name>
```

作为 `visualize ticks` 的选项，这个选项指定刻度线的方向与轴 `<axis name>` 一致。

```
/tikz/data visualization/low=<dimension>
```

这个选项在前面已经出现过多次，在 §77.5.2 中该选项用于指定坐标轴的可视化区的一个起始端点，在 §77.5.3 中该选项用于指定网格线的起点。在这里，单词 `low` 指的是刻度线的起点，`<dimension>` 是带单位的（或正或负的）尺寸，若 `<dimension>` 为正值，则刻度线起点沿着刻度线方向（由选项 `direction axis=` 指定）偏移；若 `<dimension>` 为负值，则刻度线起点沿着刻度线方向的反方向偏移；偏移的距离是相对于刻度线所在坐标轴的中心线来计算的。例如某一横轴的刻度线的偏移选项是 `low=10pt`，那么刻度线的起点将偏移到横轴之上 10pt 处；如果偏移选项是 `low=-10pt`，那么刻度线的起点将偏移到横轴之下 10pt 处。

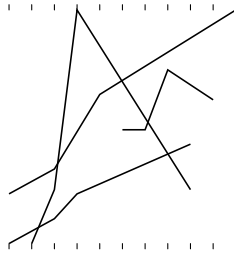
```
/tikz/data visualization/high=<dimension>
```

与 `low=` 类似，这个选项指定刻度线的终点。

`/tikz/data visualization/tick length=<dimension>`

同时指定 `low=-<dimension>` 和 `high=<dimension>`.

下面再将刻度线改短:



```
\tikzset{
  data visualization/our system/.append style={
    x axis={visualize ticks={direction axis=left axis,high=0pt,left axis={goto=min}},
      visualize ticks={direction axis=left axis,low=0pt,left axis={goto=max}},
    }
  }
}
\tikz \datavisualization [
  our system,
  x axis={attribute=time, length=3cm, ticks=many},
  left axis ={attribute=money},
  right axis={attribute=people},
  visualize as line/.list={people 1, people 2, money 1, money 2}]
data group {people and money};
```

上面例子中，对上部横轴使用了选项 `low=0pt`，对下部横轴使用了选项 `high=0pt`，可见在默认下，横轴的刻度线的 `low=` 与 `high=` 的值都不是 `0pt`。

设置刻度线的外观，参考 §77.4.9，§77.4.10，§77.4.11。

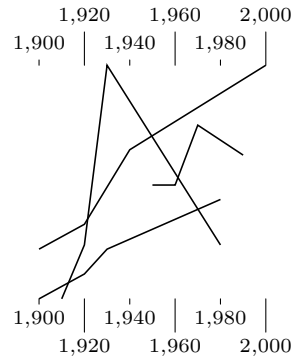
为刻度线添加刻度值标签用到下面的选项。

`/tikz/data visualization/tick text at low=<true or false>` (默认值 `true`)

把这个选项作为选项 `visualize ticks` 的值后，会在刻度线的 `low` 端即起点之前添加刻度值标签。也就是说，对于横轴，该选项会把刻度值标签加在刻度线的下方；对于纵轴，该选项会把刻度值标签加在刻度线的左方。默认值是 `true`。

`/tikz/data visualization/tick text at high=<true or false>` (默认值 `true`)

把这个选项作为选项 `visualize ticks` 的值后，会在刻度线的 `high` 端即终点之后添加刻度值标签，默认值是 `true`。



```

\tikzset{
  data visualization/our system/.append style={
    x axis={
      visualize ticks={direction axis=left axis, left axis={goto=min}, high=0pt,
        tick text at low, stack},
      visualize ticks={direction axis=left axis, left axis={goto=max}, low=0pt, tick
        text at high, stack}
    }
  }
}
\tikz \datavisualization [
  our system,
  x axis={attribute=time, length=3cm, ticks=some},
  left axis ={attribute=money},
  right axis={attribute=people},
  visualize as line/.list={people 1, people 2, money 1, money 2}]
data group {people and money};
/tikz/data visualization/no tick text

```

这个选项作为选项 `visualize ticks` 的值，会同时设置 `tick text at low=false` 和 `tick text at high=false`，即不添加刻度值标签。

设置刻度值标签的外观，参考 §77.4.9, §77.4.10, §77.4.11, §77.4.12.

77.5.5 坐标轴标签的可视化

要使轴带有显示的标签，需要给轴加 `label` 或 `label={ [options] <text> }` 选项 (§77.2.7):

```

/tikz/data visualization/axis options/label={ [options] <text> } (默认为选项 visualize label 的设置)

```

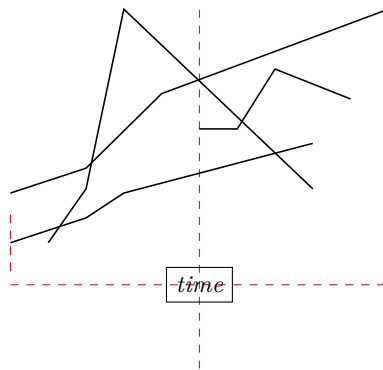
在预定义的坐标系统中，某个轴带有这个选项后，它设置轴标签。但对于自定义的轴来说，这个选项并不直接导致标签可视化，需要先用选项 `visualize label` 做有关设置后，才会令选项 `label` 有可视化效果。

`/tikz/data visualization/axis options/visualize label=<options>`

`<options>` 主要用于确定轴标签的位置，外观样式等，其中可以用选项 `node style=`。轴标签位置的确定方式与轴位置的确定方式类似。以下横轴为例，下横轴标签在水平方向的位置可以用选项 `goto` 和 `goto pos=<fraction>` 确定，这两个选项指定横轴标签在水平方向上（初始之下是沿着横轴）的偏移位置。`goto pos=0.5` 会把轴标签放在下横轴的中间位置；下横轴标签在垂直方向的偏移位置可以用其参照轴 `left axis` 来确定，如

```
x axis={visualize label={
    x axis={goto pos=.5},
    left axis={padding=1.5em, goto=padded min}}}
```

上面的代码对轴 `x axis` 的标签位置做设置，规定标签在水平方向的偏移位置是轴 `x axis` 的中间点，垂直方向的偏移位置是：轴 `left axis` 对应的变量的最小值点还向下平移 1.5em 点处；这两个偏移位置共同确定一个点，该点就是轴标签 `node` 的指向点，默认轴标签 `node` 的中心处于该点。



```
\tikzdatavisualizationset{
  our system/.append style={
    x axis={
      visualize label={
        node style=draw,
        x axis={goto pos=.5}, %横轴标签的列位置
        left axis={padding=1.5em, goto=padded min}}, %横轴标签的行位置
      visualize axis={
        style={red,dashed}, %画一个红色虚线横轴
        left axis={padding=1.5em, goto=padded min}},
      visualize grid={direction axis=left axis},
      visualize ticks={major={low=-1em,high=1em,style={red,dashed}}} %给横轴定义
        主刻度线样式
    }
  }
}
\tikz \datavisualization [
```

```

our system,
x axis={attribute=time, label, ticks={major at={min,max}}, %在横轴始末位置画刻
      度线
      grid={major at=1950 as[{{low=-1,high=5,style={red,dashed}}]}}, %在横轴中间画
      一根网格线
left axis ={attribute=money},
right axis={attribute=people},
visualize as line/.list={people 1, people 2, money 1, money 2}]
data group {people and money};

```

由上面的代码可见，用两个偏移位置确定轴标签位置的句法有些繁琐。还有另一种简捷一些的直接确定轴标签位置的方法，即使用 `node style={at=...}` 选项，其中 `at=...` 按 TikZ 的句法确定一个点，作为轴标签位置的参考点，当然这个点应当与当前的可视化图形有关。考虑这样一个坐标系：以预定义的 `data visualization bounding box` (§75.6) 的左下角为原点创建一个直角坐标系，以 `cm` 为单位长度，这个坐标系中的点坐标就与当前的可视化图形有关了，在 `node style={at=...}` 中涉及的点的坐标就是这个坐标系中的坐标。

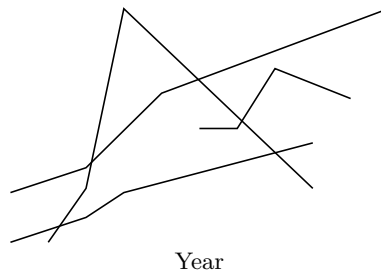
在 `node style={at=...}` 中可以使用多种 TikZ 句法来确定一个点，比较灵活，例如：

```

node style={at={{(data visualization bounding box.south)+(1,-1)}}} %坐标计算式
node style={at={{(0,0)!0.5!(-30:4)}}}
node style={at={{(0,0 |- data visualization bounding box.south)}}, below} %加平移选项
node style={at={{(0,0)+(1,-1)}}}, below right=-1cm and 2cm}

```

还需要注意的是，因为选项 `x axis={goto pos=<fraction>}` 指定轴标签在水平方向的偏移位置，所以如果某个轴同时带有 `x axis={goto pos=<fraction>}` 和 `node style={at=...}` 两个选项，则这两个选项的作用会叠加。



```

\tikzdatavisualizationset{
  our system/.append style={
    x axis={visualize label={
      x axis={goto pos=.5},
      node style={at={{(0,0 |- data visualization bounding box.south)}}, below
    }}}}
\tikz \datavisualization [
  our system,

```

```
x axis={attribute=time, ticks=some, label=Year},
left axis ={attribute=money},
right axis={attribute=people},
visualize as line/.list={
people 1, people 2, money 1, money 2}]
data group {people and money};
```

`/tikz/data visualization/axis option/anchor at min`

如果某个轴带有这个选项，则该轴的标签会位于轴的起点，即 `min` 或 `padded min` 处对应的点，并且标签的 `anchor` 位置还会使得标签看上去处于轴的起始点的“之前”。例如通常情况下，对于横轴，标签会处于横轴左侧；对于纵轴，标签会处于纵轴下方。

`/tikz/data visualization/axis option/anchor at max`

类似 `anchor at min`.

77.5.6 完整的定义代码

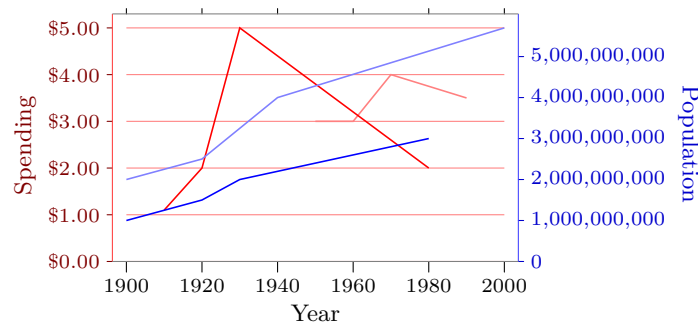
下面是前文定义轴系统 `our system` 的完整代码：

```
\tikzdatavisualizationset{ our system/.style={
% 声明 3 个坐标轴
new Cartesian axis=x axis, new Cartesian axis=left axis, new Cartesian axis=right
axis,
% 指定轴的偏移、方向
all axes={padding=.5em}, left axis={unit vector={{(0cm,1pt)}}}, right axis={unit vector
={{(0cm,1pt)}}},
% 指定 x axis 对应的变量
x axis={attribute=x},
% 指定轴的长度
x axis ={length=\pgfkeysvalueof{/tikz/data visualization/scientific axes/width}},
left axis ={length=\pgfkeysvalueof{/tikz/data visualization/scientific axes/height}},
right axis={length=\pgfkeysvalueof{/tikz/data visualization/scientific axes/height}},
% 设置轴的样式——颜色
every axis/.style={style=black!50}, % 设为默认颜色
% 将轴可视化
left axis= {visualize axis={x axis= {goto=padded min}, style=red!75, padded}},
right axis={visualize axis={x axis= {goto=padded max}, style=blue!75,padded}},
x axis= {visualize axis={left axis={goto=padded min}, padded},
visualize axis={left axis={goto=padded max}, padded}},
% 设置可视化网格线的默认项目
x axis= {visualize grid={direction axis=left axis}},
```

```

left axis={visualize grid={direction axis=x axis, common={style=red!50}}},
right axis={visualize grid={direction axis=x axis, common={style=blue!50}}},
% 设置可视化刻度线的默认项目
left axis={visualize ticks={style={red!50!black}, direction axis=x axis, x axis={goto
=padded min}, high=0pt, tick text at low}},
right axis={visualize ticks={style={blue!80!black}, direction axis=x axis, x axis={
goto=padded max}, low=0pt, tick text at high}},
x axis={visualize ticks={direction axis=left axis, left axis={goto=padded min}, high
=0pt, tick text at low},
visualize ticks={direction axis=left axis, left axis={goto=padded max}, low=0pt
}},
% 默认各个轴都有刻度线
all axes={ticks},
% 设置可视化的轴标签的默认项目
x axis={visualize label={x axis={goto pos=.5}, node style={
at={{(0,0 |- data visualization bounding box.south)}}, below}}},
left axis={visualize label={left axis={goto pos=.5}, node style={
at={{(0,0 -| data visualization bounding box.west)}}, rotate=90, anchor=south, red
!50!black}}},
right axis={visualize label={right axis={goto pos=.5}, node style={
at={{(0,0 -| data visualization bounding box.east)}}, rotate=-90, anchor=south,
blue!80!black}}},
}}

```



```

\tikz \datavisualization [
our system,
x axis={attribute=time, label=Year, ticks={tick text padding=2pt, style={/pgf/number
format/set thousands separator=}}},
left axis={attribute=money, label=Spending, padding min=0, include value=0, grid,
ticks={tick prefix=\$, style={/pgf/number format/fixed, /pgf/number format/fixed
zerofill, /pgf/number format/precision=2}}},

```

```

right axis={attribute=people, label=Population, padding min=0, include value=0, ticks
  ={style=/pgf/number format/fixed}},
visualize as line/.list={
  people 1, people 2, money 1, money 2},
  people 1={style={visualizer color=blue}},
  people 2={style={visualizer color=blue!50}},
  money 1={style={visualizer color=red}},
  money 2={style={visualizer color=red!50}} ]
data group {people and money};

```

下面是另一个例子。

先定义坐标系。

%定义一个坐标系, #1 是坐标轴与绘图区域的间距, #2 是横轴的长度, #3 是纵轴的长度

```

\tikzdatavisualizationset{
  AB system/.style n args={3}{
    new Cartesian axis=x axis, new Cartesian axis=y axis, new Cartesian axis=xx
      axis,
    y axis={unit vector={(0cm,1pt)}},
    all axes={padding=#1},
    x axis={
      length=#2,
      attribute=expense,
      visualize axis={y axis={goto=padded min}, padded, style={->}},
      visualize ticks={direction axis=y axis, low=0pt, high=4pt, y axis={goto=
        padded min},tick text at low},
      visualize label={x axis={goto=padded max}, node style={anchor=west}}
    },
    y axis={
      length=#3,
      attribute=price,
      visualize axis={x axis={goto=padded min}, padded, style={->}},
      visualize ticks={direction axis=x axis, low=0pt, high=4pt, x axis={goto=
        padded min},tick text at low},
      visualize label={y axis={goto=padded max}, node style={anchor=south}}
    },
    xx axis={
      length=#2,
      visualize label={node style={at={((data visualization bounding box.south))},
        below=1.5em}}
    }
  }
}

```

```

    }
  }
}

```

再定义一个数据点组。

```

\tikz \datavisualization
% B county 曲线上的点
data group [set=lineB]{e-p1BL} = {
  data {
    expense,price
    0,100
    100,100
    100,15
    117747,15
    117747,100}}
% B county 曲线上的标记为三角的点
data group [set=scatterB]{e-p1BS} = {
  data {
    expense,price
    100,90
    100,70
    100,55
    100,40
    100,22
    19600,15
    39200,15
    58800,15
    78400,15
    98000,15
    117747,30
    117747,50
    117747,70
    117747,90}}
% A county 曲线对应的点
data group [set=lineA]{e-p1AL} = {
  data {
    expense,price
    100,30
    114336,30

```

```

114336,100
146000,100}}
% A county 曲线上标记为圆点的点
data group [set=scatterA]{e-p1AS} = {
  data {
    expense,price
    100,97
    100,80
    100,65
    100,48
    100,30
    9600,30
    29200,30
    48800,30
    68400,30
    100000,30
    114336,40
    114336,60
    114336,75
    114336,96
    126000,100
    136000,100}
};

```

画出图形。

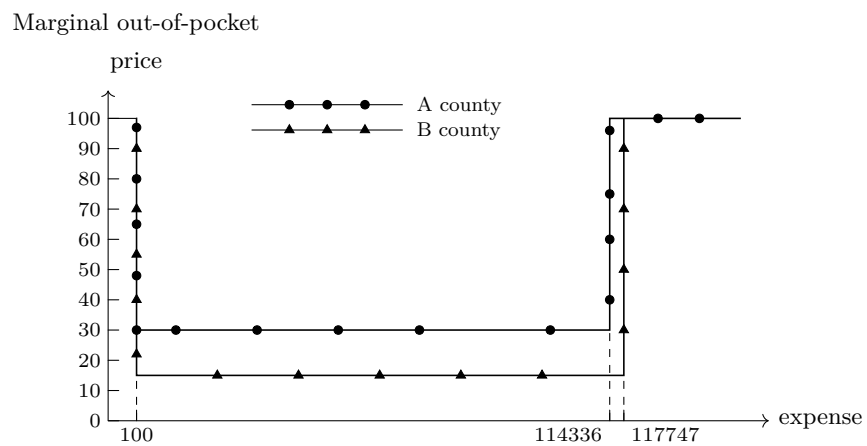


Figure 2: A county and B county in 2011

```

\begin{tikzpicture}
\datavisualization [

```

```

AB system={1em}{8cm}{4cm},
visualize as line=lineA,
visualize as scatter=scatterA,
visualize as line=lineB,
visualize as scatter=scatterB,
scatterA={style={mark=otimes*,mark options={color=black,scale=0.8}}},
scatterB={style={mark=triangle*,mark options={color=black}}},
x axis={
  label={expense},
  ticks={major at={
    100,
    114336 as [node style={anchor=north east}]$114336$,
    117747 as [node style={anchor=north west}]$117747$}}
},
y axis={
  label={\tikz\node[align=center]{Marginal out-of-pocket\ price}};,
  padding min=0, include value=0,
  ticks={major at={0,10,...,100}}
},
new legend entry={
  text=A county,
  visualizer in legend={\draw (0,0)--(-2,0) plot[mark=otimes*,mark options={color=
    black,scale=0.8},only marks] coordinates {(-1.5,0) (-1,0) (-0.5,0)};};
},
new legend entry={
  text=B county,
  visualizer in legend={\draw (0,0)--(-2,0);\draw plot[mark=triangle*,mark options
    ={color=black},only marks] coordinates {(-1.5,0) (-1,0) (-0.5,0)};};
},
legend={at values={expense=58000, price=100}},
xx axis={label={Figure 2: A county and B county in 2011}}
]

data group {e-p1AL}
data group {e-p1AS}
data group {e-p1BL}
data group {e-p1BS}
info{
  \draw (visualization cs: expense=100,price=100)--++(-1em,0);

```



```

\draw [dashed](visualization cs: expense=100,price=0)--(visualization cs: expense
=100,price=15);
\draw [dashed](visualization cs: expense=114336,price=0)--(visualization cs:
expense=114336,price=30);
\draw [dashed](visualization cs: expense=117747,price=0)--(visualization cs:
expense=117747,price=15);
};
\end{tikzpicture}

```

77.5.7 Using the New Axis System Key

78 Visualizers

78.1 Overview

显像器 (visualizer) 将数据点可视化, 并决定以什么方式显示数据点。例如, 显像器决定将数据点显示为散点或曲线, 散点的标记符号是什么样式, 曲线的线型、线宽、颜色, 散点或曲线是否带有标签, 等等。如果需要手工设置数据点的显示样式、外观, 可以在显像器的参数中使用本节介绍的选项。自动设置数据点的显示样式、外观的方法是使用 `style sheet`, 将在下节介绍。

有多种不同的显像器, 例如, `line visualizer`, 将数据点用直线段连接起来, 这个显像器或者是在实际提供的数据点之间插入一些点构成线段, 或者是按数据点出现的次序将它们连接起来。再如, `scatter visualizer` 或 `mark visualizer`, 用某种标记符号来标记数据点。在一个可视化过程中可以使用多个显像器, 例如, 一个可视化过程中有多组数据点, 对各组数据使用不同的显像器, 使它们的表现形式相互区别。对一个数据点可以使用多个显像器。

78.2 用法

78.2.1 使用一个显像器

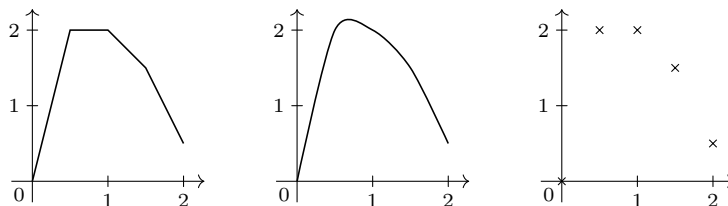
在命令 `\datavisualization` 的选项中使用

`visualize as line` 用直线段连接数据点

`visualize as smooth line` 用直线段连接数据点并使得连接点处平滑

`visualize as scatter` 将数据点显示为散点

这样的选项。



% 定义数据组

```
\tikz \datavisualization data group {example} = {
  data {
    x, y
    0, 0
    0.5, 2
    1, 2
    1.5, 1.5
    2, 0.5
  };
\tikz \datavisualization [school book axes, visualize as line] data group {example};
\qqquad
\tikz \datavisualization [school book axes, visualize as smooth line] data group {
  example};
\qqquad
\tikz \datavisualization [school book axes, visualize as scatter] data group {example};
```

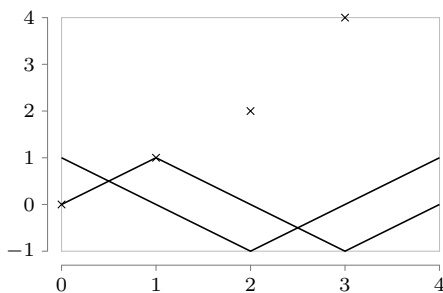
如果在命令 `\datavisualization` 的选项中只使用一个显像器选项，例如 `visualize as line`，那么命令之内的所有数据点（包括所有 `data` 算子定义的数据点）都会被用直线段连起来，因此只能得到一个曲线。

78.2.2 使用多个显像器

为了能使每一组数据点都产生一个单独的曲线，需要使得每一组数据点都应用一个显像器。可以先给显像器命名，然后把显像器名称引入到数据点组中。一个显像器作为一个 `key`，可以带有一个参数，或者说可以被赋予一个值（名称），例如，`visualize as line=<name>`，然后将这个 `<name>` 添加到数据点中，作为一个标识，表示该数据点用 `visualize as line` 来显示。`<name>` 对应的标识名称（变量名，attribute）是 `set`，而各个 `<name>` 则是 `set` 的值。

`/data point/set`

这个 `key` 的路径与表示数据点分量的标识（变量名）`/data point/x`，`/data point/y` 是一样的，在数据点列表中，这个标识符号代表的是显像器的名称 `<name>`。当在数据点列表的头行中写出 `set` 时，每个数据点都要带上相应的显像器名称 `<name>` 作为一个元素数据，指示该点用名称为 `<name>` 的显像器来显示。



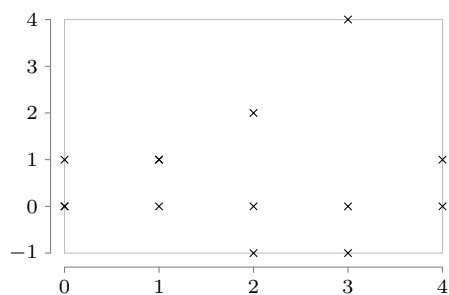
```
\tikz \datavisualization [scientific axes=clean,
```

```

        visualize as line=sin,
        visualize as line=cos,
        visualize as scatter=tan]
data {
    x, y, set
    0, 0, sin
    1, 1, sin
    2, 0, sin
    3, -1, sin
    4, 0, sin
    0, 1, cos
    1, 0, cos
    0, 0, tan
    1, 1, tan
    2, 2, tan
    3, 4, tan
    2, -1, cos
    3, 0, cos
    4, 1, cos
};

```

注意，如果在数据点列表中不给出标识符号“set”，那么就默认该数据点列表使用之前最近出现的显像器来显示。将下面的例子对比上面的例子：



```

\tikz \datavisualization [scientific axes=clean,
        visualize as line=sin,
        visualize as line=cos,
        visualize as scatter=tan]
data {
    x, y, % 缺少 set, 故使用 visualize as scatter=tan 作成散点图
    0, 0, sin
    1, 1, sin
    2, 0, sin

```

```

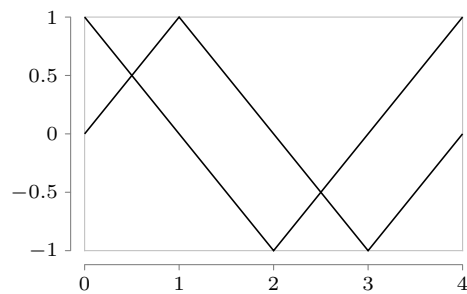
3, -1, sin
4, 0, sin
0, 1, cos
1, 0, cos
0, 0, tan
1, 1, tan
2, 2, tan
3, 4, tan
2, -1, cos
3, 0, cos
4, 1, cos
};

```

上一个选项的用法有些繁琐，使用下面的选项也可以有相同的效果，而用法相对简捷一些。

`/pgf/data/set=<name>`

这是 `/data point/set=<name>` 的另写。这个选项作为 `data` 算子的选项，指示该 `data` 算子设置的数据点组用对应 `<name>` 的显像器来显示。



```

\tikz \datavisualization [scientific axes=clean,
    visualize as line=sin,
    visualize as line=cos]
data [set=sin] {
    x, y
    0, 0
    1, 1
    2, 0
    3, -1
    4, 0
}
data [set=cos] {
    x, y
    0, 1
    1, 0

```

```

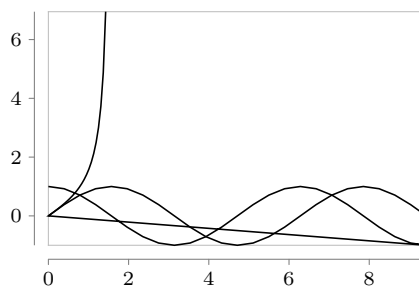
    2, -1
    3, 0
    4, 1
};

```

当需要给一个显像器赋予多个不同名称时，可以使用 `/.list` 手柄，例如

```
visualize as line/.list={sin, cos, tan}
```

这样就可以令 `visualize as line` 依次取值 `sin`, `cos`, `tan`，这些值应当依次用作各个 `data` 算子的选项，并且每个值最好只用一次，不要重复使用。如果重复使用这些值，例如，如果两个 `data` 算子都有 `set=cos` 选项，那么这两个算子定义的数据点之间会用（由显像器 `visualize as line` 产生的）直线段连接起来，这样一来，两个数据点组看上去就像是一个数据点组。



```

\tikz \datavisualization [scientific axes=clean,
    visualize as line/.list={sin, cos, tan}]
data [set=sin, format=function] {
    var x : interval[0:3*pi];
    func y = sin(\value x r);
}
data [set=cos, format=function] {
    var x : interval[0:3*pi];
    func y = cos(\value x r);
}
data [set=cos, format=function] { % 用了 set=cos, 导致与 cos 曲线连起来了
    var x : interval[0:pi/2.2];
    func y = tan(\value x r);
};

```

78.2.3 设置显像器的外观效果

```
/tikz/data visualization/<visualizer name>=<options>
```

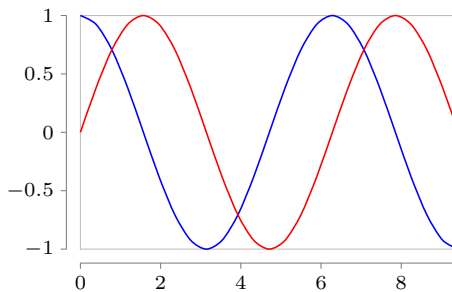
该选项可用作命令 `\datavisualization` 的选项，或者 `\tikzset` 的参数。用选项 `visualize as...=<visualizer name>` 给显像器命名时，程序会自动创建 `key` 路径

```
/tikz/data visualization/<visualizer name>
```

并且还会给 `<visualizer name>` 定义一个初始值，这个初始值是个正整数（参考 §79.4.2）。显像器有了名称，就可以同时把 `<visualizer name>=<options>` 作为可视化命令的选项来规定显像器的显示效果，`<options>` 中的选项会被冠以前缀

```
/tikz/data visualization/visualizer options/
```

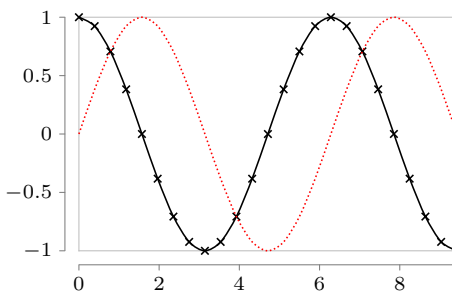
来执行。`<options>` 中可以使用 `style`, `label in legend`, `label in data` 等选项来设置显像器的外观效果。



```
\tikz \datavisualization [scientific axes=clean,
    visualize as smooth line/.list={sin, cos},
    sin={style=red},
    cos={style=blue}]
data [set=sin, format=function] {
    var x : interval[0:3*pi];
    func y = sin(\value x r);
}
data [set=cos, format=function] {
    var x : interval[0:3*pi];
    func y = cos(\value x r);
};
```

```
/tikz/data visualization/visualizer options/style=<options>
```

用在 `<visualizer name>=<options>` 的 `<options>` 中，设置显像器的外观效果，`style=<options>` 中的选项应该是 TikZ 的选项。



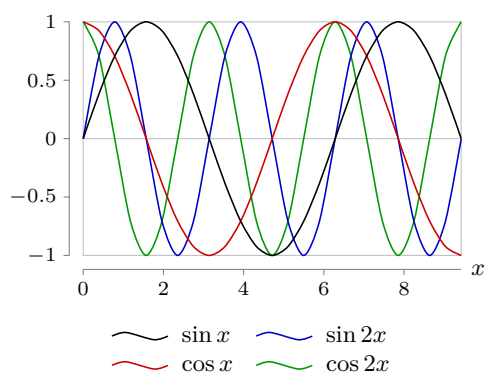
```
\tikz \datavisualization [scientific axes=clean,
    visualize as smooth line=sin,
```

```

sin={style={red, densely dotted}},
visualize as smooth line=cos,
cos={style={mark=x}}]
data [set=sin, format=function] {
  var x : interval[0:3*pi];
  func y = sin(\value x r);
}
data [set=cos, format=function] {
  var x : interval[0:3*pi];
  func y = cos(\value x r);
};

```

可以在命令 `\datavisualization` 的选项中使用 `style sheet` 选项。



```

\tikz \datavisualization [scientific axes={clean, end labels},
  x axis={label=$x$}, y axis={grid={major also at=0}},
  visualize as smooth line/.list={sin,cos,sin 2,cos 2},
  legend={below, rows=2},
  sin={label in legend={text=$\sin x$}},
  cos={label in legend={text=$\cos x$}},
  sin 2={label in legend={text=$\sin 2x$}},
  cos 2={label in legend={text=$\cos 2x$}},
  style sheet=strong colors]
data [set=sin, format=function] {
  var x : interval[0:3*pi];
  func y = sin(\value x r);
}
data [set=cos, format=function] {
  var x : interval[0:3*pi];
  func y = cos(\value x r);
}

```

```

data [set=sin 2, format=function] {
  var x : interval[0:3*pi];
  func y = sin(2*\value x r);
}
data [set=cos 2, format=function] {
  var x : interval[0:3*pi];
  func y = cos(2*\value x r);
};

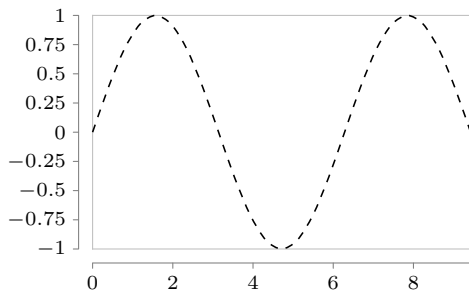
```

`/tikz/data visualization/visualizer options/ignore style sheets`

用在 `<visualizer name>=<options>` 的 `<options>` 中，使得 style sheets 不包含该显像器。

`/tikz/data visualization/every visualizer (style, 无值)`

这个样式中的选项应当是 TikZ 的选项。



```

\tikz \datavisualization
  [scientific axes=clean,
  every visualizer/.style={dashed},
  visualize as smooth line]
data [format=function] {
  var x : interval[0:3*pi];
  func y = sin(\value x r);
};

```

78.3 基本的显像器

78.3.1 直线段或曲线显像器

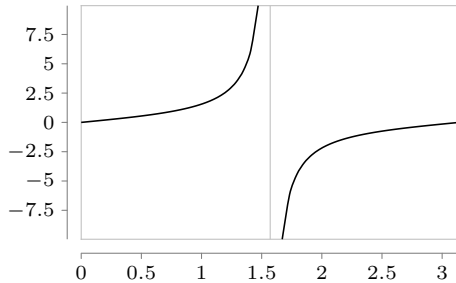
`/tikz/data visualizers/visualize as line=<visualizer name> (默认值 line)`

如果不给出 `<visualizer name>`，就默认为 `line`。这个显像器会把带有 `<visualizer name>` 的数据点之间或带有 `set=<visualizer name>` 选项的数据点组之间用直线段连接起来。

`/data point/outlier=<value> (默认值 true, 初始值 empty)`

这个选项的前缀是 `/data point/`，它用作 `data point` 的选项，当 `<value>` 非空时创建一个特殊的属性为 `outlier` 的数据点。当显像器遇到这个特殊点时，不会把这个点与它之前或之后的点连起来，在图形上得到间断（破缺、不连续）效果。比方说，如果显像器 `visualize as line` 遇到 3 个点 A, B, C，其中点 B 是 `data point[outlier]`，那么显像器就不会在 A, C 之间连线，于是 A, C 之间出现间断。这个选项常用来画函数的间断点。

下面的例子画的是 $[0, \pi]$ 之间的 $\tan x$ 的图像，以 $\frac{\pi}{2}$ 为间断点：



```
\tikz \datavisualization
  [scientific axes=clean,
   x axis={grid={major at=(pi/2)}},
   visualize as smooth line]
data [format=function] {
  var x : interval[0:pi/2-0.1];
  func y = tan(\value x r);
}
data point [outlier]
data [format=function] {
  var x : interval[pi/2+0.1:pi];
  func y = tan(\value x r);
};
```

`/tikz/data visualizers/visualize as smooth line=<visualizer name>` (默认值 `line`)

这是

```
visualize as line=<visualizer name>, <visualizer name>=smooth line
```

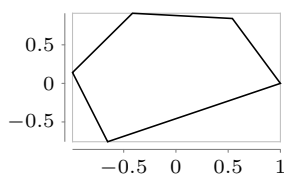
的简捷用法。

`/tikz/data visualization/visualizer options/straight line`

用于 `<visualizer name>=<options>` 中, 使得显像器的作用是用直线段连接数据点。

`/tikz/data visualization/visualizer options/straight cycle`

用于 `<visualizer name>=<options>` 中, 使得显像器的作用是将数据点连接为多边形。



```
\tikz [scale=.55] \datavisualization
  [scientific axes=clean, all axes={ticks=few},
   visualize as smooth line=my data,
   my data={straight cycle}]
data [format=function] {
  var t : interval [0:4] samples 5;
  func x = cos(\value t r);
  func y = sin(\value t r);
};
```

`/tikz/data visualization/visualizer options/polygon`

等效于前一个选项。

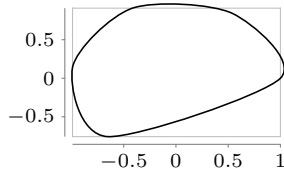
`/tikz/data visualization/visualizer options/smooth line`

用于 `<visualizer name>=<options>` 中, 使得显像器的作用是用直线段连接数据点并使得连接点处平

滑。

`/tikz/data visualization/visualizer options/smooth cycle`

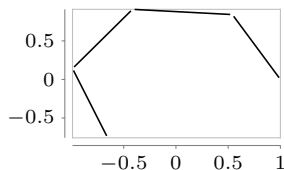
用于 `<visualizer name>=<options>` 中，使得显像器的作用是将数据点连接为多边形并使得连接点处平滑。



```
\tikz [scale=.55] \datavisualization
  [scientific axes=clean, all axes={ticks=few},
  visualize as smooth line=my data,
  my data={smooth cycle}]
data [format=function] {
  var t : interval [0:4] samples 5;
  func x = cos(\value t r);
  func y = sin(\value t r);
};
```

`/tikz/data visualization/visualizer options/gap line`

用于 `<visualizer name>=<options>` 中，使得显像器的作用是用直线段连接数据点，但直线段并不接触数据点位置，而是留有一段空缺。这个效果用命令 `\pgfplotshandlergaplineto` 实现。



```
\tikz [scale=.55] \datavisualization
  [scientific axes=clean, all axes={ticks=few},
  visualize as smooth line=my data,
  my data={gap line}]
data [format=function] {
  var t : interval [0:4] samples 5;
  func x = cos(\value t r);
  func y = sin(\value t r);
};
```

`/tikz/data visualization/visualizer options/gap cycle`

作用类似 `gap line`，只是将数据点连接为多边形。

`/tikz/data visualization/visualizer options/no lines`

用于 `<visualizer name>=<options>` 中，使得显像器不画线。

78.3.2 散点显像器

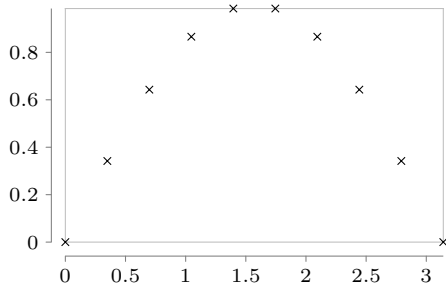
`/tikz/data visualizers/visualize as scatter=<visualizer name>` (默认值 `scatter`)

如果不给出 `<visualizer name>`，就默认为 `scatter`。它是

```
visualize as line=<visualizer name>,
```

```
<visualizer name>=no lines,
style={mark=...}
```

的简捷形式。



```
\tikz \datavisualization
  [scientific axes=clean,
   visualize as scatter]
data [format=function] {
  var x : interval [0:pi] samples 10;
  func y = sin(\value x r);
};
```

78.4 创建新的显像器

79 Style Sheets and Legends

79.1 Overview

一个可视化图形中可能有多个曲线，为了区别它们，每个曲线应该有单独的样式。为多个曲线设置单独样式的便捷方法是使用 style sheet，即“样式表”，由多个样式按次序组成的列表。在可视化命令中使用 style sheet 选项后，样式表中的第 1 个样式用于第 1 个曲线，第 2 个样式用于第 2 个曲线……，这都是自动完成的，无需手工设置曲线样式。

有的样式表专门设置颜色，有的样式表专门设置线型，所以在一个可视化命令中可以同时使用颜色 style sheet 和线型 style sheet。例如

```
\datavisualization [...style sheet=strong colors, style sheet=vary dashing]...;
```

其中用了设置颜色的样式表 strong colors 和设置线型的样式表 vary dashing。样式表 strong colors 中的颜色是那些“在白色背景下，具有最大对比度”的颜色，所以一般情况下其颜色对比效果比较好。样式表 vary dashing 中的虚线则是那些尽量让虚线的实线段穿过数据点的虚线，尽量不让数据点处于虚线的空缺部分，所以一般情况下其效果会较好。

另外也有预定义的散点样式表，其中的散点样式都是易于相互区分的，即使两个点标记重合。所以推荐使用预定义的 style sheet。

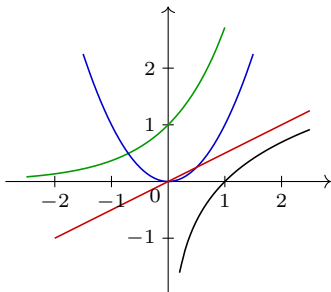
在 §78.2.3 中提到，选项 `<visualizer name>={style={<TikZ options>}}` 也可以设置显像器的外观效果，如果同时使用这个选项和样式表来规定显像器的外观效果，那么二者对显像器的规定会被累计。所以在使用样式表的时候，也可以使用选项 `<visualizer name>={style={<TikZ options>}}` 来对显像器做某些调整。

79.2 Style Sheets 的例子

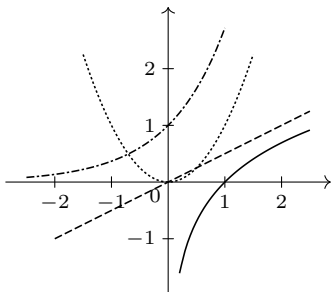
style sheet 可以与数据点标识符关联，即与 `/data point/<attribute>` 关联，在默认情况下与 `/data point/<visualizer>` 关联，使得有不同名称的显像器的效果对应 style sheet 中的不同样式。

下面的例子中，先定义一个数据点组，其中每个 data 算子都使用单独的显像器名称。

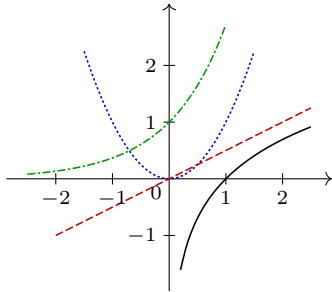
```
\tikz \datavisualization data group {function classes} = {
  data [set=log, format=function] {
    var x : interval [0.2:2.5];
    func y = ln(\value x);
  }
  data [set=lin, format=function] {
    var x : interval [-2:2.5];
    func y = 0.5*\value x;
  }
  data [set=squared, format=function] {
    var x : interval [-1.5:1.5];
    func y = \value x*\value x;
  }
  data [set=exp, format=function] {
    var x : interval [-2.5:1];
    func y = exp(\value x);
  }
};
```



```
\tikz \datavisualization [
  school book axes, all axes={unit length=7.5mm},
  visualize as smooth line/.list={log, lin, squared, exp},
  style sheet=strong colors]
data group {function classes};
```



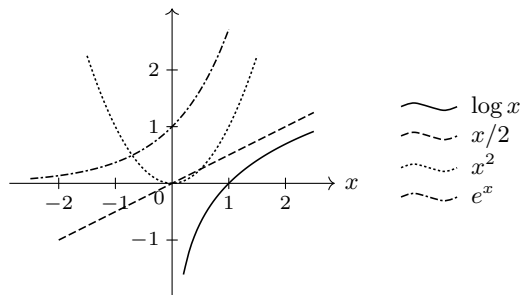
```
\tikz \datavisualization [
  school book axes, all axes={unit length=7.5mm},
  visualize as smooth line/.list={log, lin, squared, exp},
  style sheet=vary dashing]
data group {function classes};
```



```
\tikz \datavisualization [
  school book axes, all axes={unit length=7.5mm},
  visualize as smooth line/.list={log, lin, squared, exp},
  style sheet=vary dashing,
  style sheet=strong colors]
data group {function classes};
```

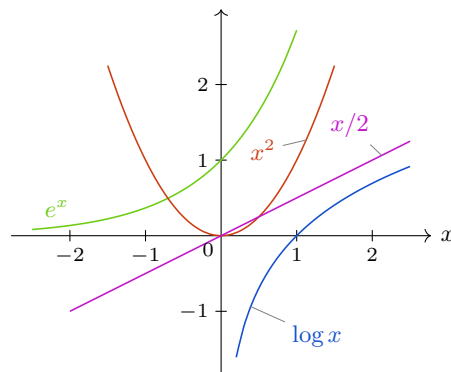
79.3 Legends 的例子

Legends, 即“图例”, 是对图形的注释。在可视化图形中, 添加图例的方式比较灵活。下面的例子里, 在 `<visualizer name>=<options>` 中使用 `label in legend={...}` 添加图例。



```
\tikz \datavisualization [
  school book axes, all axes={unit length=7.5mm},
  x axis={label=$x$},
  visualize as smooth line/.list={log, lin, squared, exp},
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$}},
  squared={label in legend={text=$x^2$}},
  exp= {label in legend={text=$e^x$}},
  style sheet=vary dashing]
data group {function classes};
```

下面的例子里, 在 `<visualizer name>=<options>` 中使用 `pin in data={...}` 和 `label in data={...}` 添加图例。



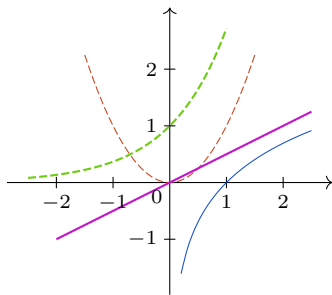
```
\tikz \datavisualization [
  school book axes,
  x axis={label=$x$},
  visualize as smooth line/.list={log, lin, squared, exp},
  every data set label/.append style={text colored},
  log= {pin in data={text'=$\log x$, when=y is -1}},
  lin= {pin in data={text=$x/2$, when=x is 2,
pin length=1ex}},
  squared={pin in data={text=$x^2$, when=x is 1.1,
pin angle=230}},
  exp= {label in data={text=$e^x$, when=x is -2}},
  style sheet=vary hue]
data group {function classes};
```

79.4 Style Sheet 的用法

79.4.1 引入一个 Style Sheet

```
/tikz/data visualization/style sheet=<style sheet>
```

该选项指定可视化命令使用的样式表，其中 `<style sheet>` 是某个样式表的名称。可以同时使用多个样式表。在默认下，样式表与显像器名称标识符 `set` 相关联，使得不同名称的显像器对应样式表中不同的样式。



```
\tikz \datavisualization [
  school book axes, all axes={unit length=7.5mm},
  visualize as smooth line/.list={
    log, lin, squared, exp},
  style sheet=vary thickness and dashing,
  style sheet=vary hue]
data group {function classes};
```

```
/data point/<attribute>/.style sheet=<style sheet>
```

这个手柄将数据点标识符 `<attribute>` 与样式表 `<style sheet>` 关联起来，主要用于自定义样式表。这个手柄选项的默认设置是

```
/data point/set/.style sheet=strong colors
```

即将样式表 `strong colors` 与显像器名称标识符 `set` 相关联。

79.4.2 创建新的样式表

样式表作为 `key` 被创建，故要使用命令 `\pgfkeys`，用下面的语句创建样式表

```
\pgfkeys{
  /pgf/data visualization/style sheets/<style sheet name>/<style 1 name>/.style={<TikZ
    set>},
  /pgf/data visualization/style sheets/<style sheet name>/<style 2 name>/.style={<TikZ
    set>},
  /pgf/data visualization/style sheets/<style sheet name>/<style 3 name>/.style={<TikZ
    set>},
  .....
}
```

其中的 `<style sheet name>` 是要创建的样式表的名称；`<style 1 name>`，`<style 2 name>` ... 是样式表名称 `<style sheet name>` 的子键（subkey），也是各个样式的名称；`<style 1 name>` 是样式表中第 1 个样式的名称，样式使用手柄 `/.style` 来规定，样式设置采用 TikZ 的设置。

创建样式表后，为了使用这个样式表，需要把 `<style sheet name>` 与某个 `<attribute>` 关联起来：

```
/data point/<attribute>/.style sheet=<style sheet name>
```

当 `<attribute>=<style 1 name>` 时，应用样式表中第 1 个样式；当 `<attribute>=<style 2 name>` 时，应用样式表中第 2 个样式……

还可以设置新建样式表的默认样式：

```
/pgf/data visualization/style sheets/<style sheet name>/default style=<value> (style,
无默认值)
```

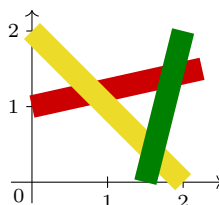
这个选项设置样式表 `<style sheet name>` 的默认样式。当 `<attribute>=<值>` 的 `<值>` 不是样式表名称的子键时，即样式表中没有定义名称为 `<值>` 的样式时，就使用默认样式。

注意在默认下，`<style sheet name>` 与 `set` 关联，而 `set` 的值与样式表中各个样式名称的关联方式可以比较灵活，下面举例来说明。

首先定义一个名称为 `traffic light` 的样式表：

```
\pgfkeys{
  /pgf/data visualization/style sheets/traffic light/.cd, % 样式表名称为 traffic light
  1/.style={green!50!black}, % 第 1 个样式
  2/.style={yellow!90!black}, % 第 2 个样式
  3/.style={red!80!black}, % 第 3 个样式
  default style/.style={black} %默认样式
}
```

然后用这个样式表绘图。



```
\tikz \datavisualization [
```

```

school book axes,
visualize as line=1, % 显像器名称与第 1 个样式一致, 故使用第 1 个样式
visualize as line=2, % 显像器名称与第 2 个样式一致, 故使用第 2 个样式
visualize as line=3, % 显像器名称与第 3 个样式一致, 故使用第 3 个样式
every visualizer/.style={line width=3mm},
style sheet=traffic light] % 使用样式表 traffic light
data point [x=1.5, y=0, set=1] % 用第 1 个显像器画绿色线
data point [x=2, y=2, set=1]
data point [x=0, y=2, set=2] % 用第 2 个显像器画黄色线
data point [x=2, y=0, set=2]
data point [x=0, y=1, set=3] % 用第 3 个显像器画红色线
data point [x=2.25, y=1.5, set=3];

```

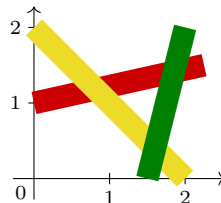
可见样式表中的第 1 个样式效果出现在图形的最上层, 会遮挡其余样式的效果。

在 §78.2.3 中提到, 当用选项 `visualize as...=<visualizer name>` 给显像器命名时, 程序会自动创建 key 路径

```
/tikz/data visualization/<visualizer name>
```

并且还会给 `<visualizer name>` 定义一个初始值, 这个初始值是个正整数, 第 1 个显像器名称的初始值是 1, 第 2 个显像器名称的初始值是 2, 依次类推。

由于 `<visualizer name>` 是个 key, 故可以为它赋值, 使其值等于样式表中的某个样式名称, 从而使得该显像器使用相应的样式。对于前面例子, 也可以如下得到。



```

\begin{tikzpicture}
\datavisualization data group {lines} = {
  data point [x=1.5, y=0, set=normal] % 用第 1 个显像器
  data point [x=2, y=2, set=normal]
  data point [x=0, y=2, set=heated] % 用第 2 个显像器
  data point [x=2, y=0, set=heated]
  data point [x=0, y=1, set=critical] % 用第 3 个显像器
  data point [x=2.25, y=1.5, set=critical]};
\datavisualization [
  school book axes,
  visualize as line=normal, % 第 1 个显像器名称
  visualize as line=heated, % 第 2 个显像器名称
  visualize as line=critical, % 第 3 个显像器名称

```



```

every visualizer/.style={line width=3mm},
/data point/set/normal/.initial=1, % 指定第 1 个显像器名称的初始值为 1, 与样式表的第 1
    个样式名称一致, 故该显像器使用第 1 个样式
/data point/set/heated/.initial=2, % 指定第 2 个显像器名称的初始值为 2, 与样式表的第 2
    个样式名称一致, 故该显像器使用第 2 个样式
/data point/set/critical/.initial=3, % 指定第 3 个显像器名称的初始值为 3, 与样式表的第
    3 个样式名称一致, 故该显像器使用第 3 个样式
style sheet=traffic light] % 使用样式表 traffic light
data group {lines};
\end{tikzpicture}

```

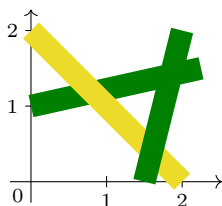
由于程序会自动设置显像器名称的初始值为 1, 2, 3,..., 故上面例子的代码也可以省略如下:

```

\begin{tikzpicture}
\datavisualization data group {lines} = {
    data point [x=1.5, y=0, set=normal] % 用第 1 个显像器
    data point [x=2, y=2, set=normal]
    data point [x=0, y=2, set=heated] % 用第 2 个显像器
    data point [x=2, y=0, set=heated]
    data point [x=0, y=1, set=critical] % 用第 3 个显像器
    data point [x=2.25, y=1.5, set=critical]};
\datavisualization [
    school book axes,
    visualize as line=normal, % 第 1 个显像器名称
    visualize as line=heated, % 第 2 个显像器名称
    visualize as line=critical, % 第 3 个显像器名称
    every visualizer/.style={line width=3mm},
    style sheet=traffic light] % 使用样式表 traffic light
data group {lines};
\end{tikzpicture}

```

修改上面的例子, 使得第 3 个显像器的样式与第 1 个显像器相同。



```

\begin{tikzpicture}
\datavisualization data group {lines} = {
    data point [x=1.5, y=0, set=normal] % 用第 1 个显像器
    data point [x=2, y=2, set=normal]

```

```

data point [x=0, y=2, set=heated] % 用第 2 个显像器
data point [x=2, y=0, set=heated]
data point [x=0, y=1, set=critical] % 用第 3 个显像器
data point [x=2.25, y=1.5, set=critical]];
\datavisualization [
  school book axes,
  visualize as line=normal, % 第 1 个显像器名称
  visualize as line=heated, % 第 2 个显像器名称
  visualize as line=critical, % 第 3 个显像器名称
  /data point/set/critical/.initial=1, % 将第 3 个显像器名称的初始值设为 1, 与第 1 个样式
    名称一致, 故使用第 1 个样式
  every visualizer/.style={line width=3mm},
  style sheet=traffic light] % 使用样式表 traffic light
data group {lines};
\end{tikzpicture}

```

还有一个专门用于创建样式表的命令:

```
\pgfdeclarestylesheet{<style sheet name>}{<keys>}
```

例如, 前面用命令 `\pgfkeys` 创建的样式表 `traffic light` 也可以如下得到

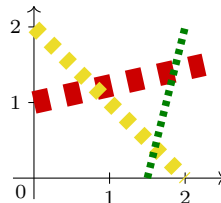
```

\pgfdeclarestylesheet{traffic light}{
  1/.style={green!50!black},
  2/.style={yellow!90!black},
  3/.style={red!80!black},
  default style/.style={black}
}

```

这个命令会自动添加路径前缀 `/pgf/data visualization/style sheets/`.

另外, 在定义样式表时, 其中各个样式的代码中可以含有一个变量, 这个变量以显像器名称的值为自己的值, 而显像器名称有初始值。



```

\begin{tikzpicture}
\datavisualization data group {lines} = {
  data point [x=1.5, y=0, set=normal] % 用第 1 个显像器
  data point [x=2, y=2, set=normal]
  data point [x=0, y=2, set=heated] % 用第 2 个显像器

```

```

data point [x=2, y=0, set=heated]
data point [x=0, y=1, set=critical] % 用第 3 个显像器
data point [x=2.25, y=1.5, set=critical]};

\pgfdeclareshape{my dashings}{
  default style/.style={dash pattern={on #1*2pt off #1*2pt}, line width=#1mm}
} % 定义样式表 my dashings, 其中只有默认样式, 代码中使用了一个变量, 使得所定义的虚线外观随
   着该变量值的变化而变化

\datavisualization [
  school book axes,
  visualize as line=normal,
  normal={style=green!50!black}, % 为该显像器的外观添加特殊颜色
  visualize as line=heated,
  heated={style=yellow!90!black}, % 为该显像器的外观添加特殊颜色
  visualize as line=critical,
  critical={style=red!80!black}, % 为该显像器的外观添加特殊颜色
  style sheet=my dashings]
data group {lines};
\end{tikzpicture}

```

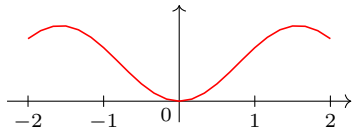
在上面例子中, 样式表 `my dashings` 的默认样式的代码中, 变量 `#1` 会以显像器名称的初始值为值。第 1 个显像器与绿色对应, 其初始值为 1, 故它画出的线较细。

79.4.3 创建新的颜色样式表

创建颜色样式表时, 最好用选项 `visualizer color=<color>` 来规定颜色, 而不是像 §79.4.2 中定义的样式表 `traffic light` 那样直接在手柄 `/.style` 中使用 `<color>`, 因为 `visualizer color=<color>` 不仅规定颜色, 还有其它的作用。在 §79.8.1 中介绍的选项 `text colored` 会调用样式表中相应样式的 `visualizer color=<color>`。涉及颜色的各种预定义样式表中, 各个样式的颜色都用选项 `visualizer color` 规定, 因此都能与选项 `text colored` 相配合。

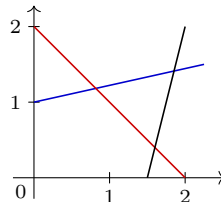
`/tikz/visualizer color=<color>`

这个选项可以直接用作可视化命令 `\datavisualization` 的选项, 也可以用作选项 `style={}` 的参数, 也可以用作自定义样式表中手柄 `/.style` 的参数, 但不能直接用作显像器名称的参数, 因为这个 key 的前缀是 `/tikz`。



```
\begin{tikzpicture}
\datavisualization [ school book axes,
visualize as line, visualizer color=red]
data [format=function]{
var x : interval [-2:2];
func y = sin(\value x r)^2;
}\end{tikzpicture}
```

下面的例子用 `visualizer color` 定义一个颜色样式表，并且画出前面定义的数据点组 `data group {lines}`。



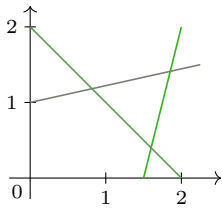
```
\begin{tikzpicture}
\pgfdeclarestylesheet{my colors}{
default style/.style={visualizer color=black}, % 直接用选项 visualizer color 规定
颜色
1/.style={visualizer color=black},
2/.style={visualizer color=red!80!black},
3/.style={visualizer color=blue!80!black},
}

\datavisualization [
school book axes,
visualize as line=normal,
visualize as line=heated,
visualize as line=critical,
style sheet=my colors]
data group {lines};
\end{tikzpicture}
```

还有一个专门用于创建颜色样式表的命令：

```
\tikzdvdeclarestylesheetcolorseries{<name>}{<color model>}{<initial color>}{<step>}
```

这个命令实际上使用命令 `\pgfdeclarestylesheet` 来工作。下面是个例子：



```
\tikzdeclarestylesheetcolorseries{greens}{hsb}
  {0.3,1.3,0.8} {0,-.4,-.1}
\tikz \datavisualization [
  school book axes,
  visualize as line=normal,
  visualize as line=heated,
  visualize as line=critical,
  style sheet=greens]
data group {lines};
```

这个例子中，颜色样式表的名称是 `greens`，颜色模式是 `hsb`，该颜色模式下的第 1 个颜色由参数（向量）`{0.3,1.3,0.8}` 规定，第 2 个颜色由第 1 个颜色的参数（向量）加上步长（向量）`{0,-.4,-.1}` 得到。

79.5 预定义的线型样式表

预定义的线型样式表有 `vary thickness`, `vary dashing`, `vary dashing and thickness`. 为了展示这些预定义的线型样式表，先定义一组显像器以及一组函数数据。

```
\tikzdatavisualizationset {
  example visualization/.style={
    scientific axes=clean,
    y axis={ticks={style={
      /pgf/number format/fixed,
      /pgf/number format/fixed zerofill,
      /pgf/number format/precision=2}}},
    x axis={ticks={tick suffix=${}^{\circ}$}},
    1={label in legend={text=${\frac{1}{6}}\sin 11x$}},
    2={label in legend={text=${\frac{1}{7}}\sin 12x$}},
    3={label in legend={text=${\frac{1}{8}}\sin 13x$}},
    4={label in legend={text=${\frac{1}{9}}\sin 14x$}},
    5={label in legend={text=${\frac{1}{10}}\sin 15x$}},
    6={label in legend={text=${\frac{1}{11}}\sin 16x$}},
    7={label in legend={text=${\frac{1}{12}}\sin 17x$}},
    8={label in legend={text=${\frac{1}{13}}\sin 18x$}}
  }
}

\tikz \datavisualization data group {sin functions} = {
```

```

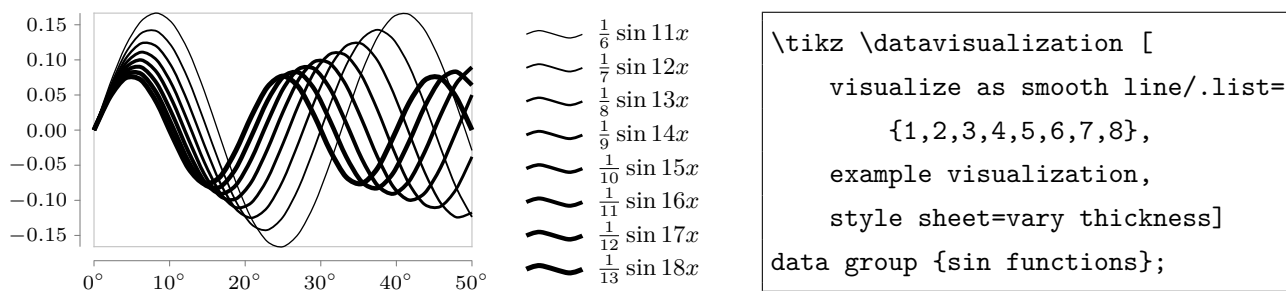
data [format=function] {
  var set : {1,...,8};
  var x : interval [0:50];
  func y = sin(\value x * (\value{set}+10))/(\value{set}+5);
}
};

```

上面的代码中，`var set : {1,...,8}` 规定显像器名称标识符 `set` 的取值。

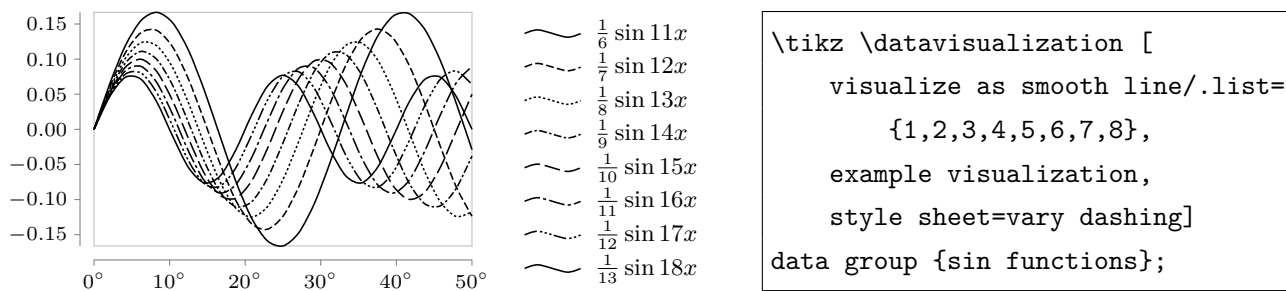
style sheet=vary thickness

这个样式表中的各个样式具有不同的线宽，第一个样式线宽最细。



style sheet=vary dashing

这个样式表中的各个样式具有不同的线型，第一个样式是实线，第二个样式是虚线，第三个样式是点线，第四个样式是点划线，等等。这个样式表中共有 7 个线型样式，如果显像器个数超过 7 个，则第 8 个及以后的显像器都自动用实线，此时需要手工设置这些显像器的线型。



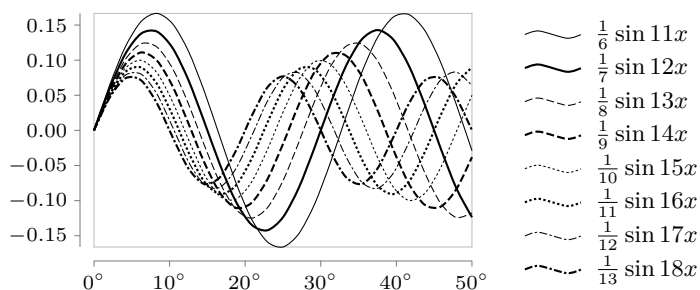
style sheet=vary dashing and thickness

这个样式表中的各个样式在线型和线宽两方面相互区别，共有 14 个样式。注意这个样式表不是

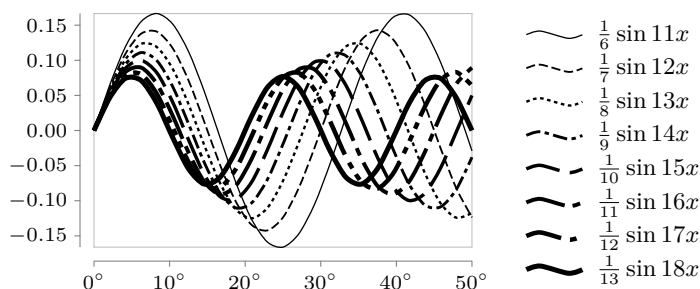
```
style sheet=vary thickness
```

```
style sheet=vary dashing
```

这两个样式表的简单叠加。



```
\tikz \datavisualization [
  visualize as smooth line/.list=
    {1,2,3,4,5,6,7,8},
  example visualization,
  style sheet=vary thickness
  and dashing]
data group {sin functions};
```



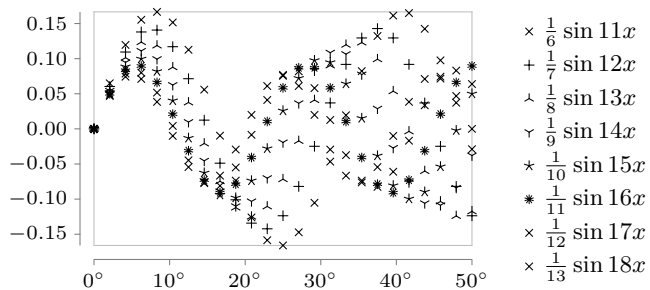
```
\tikz \datavisualization [
  visualize as smooth line/.list=
    {1,2,3,4,5,6,7,8},
  example visualization,
  style sheet=vary thickness,
  style sheet=vary dashing]
data group {sin functions};
```

79.6 预定义的散点样式表

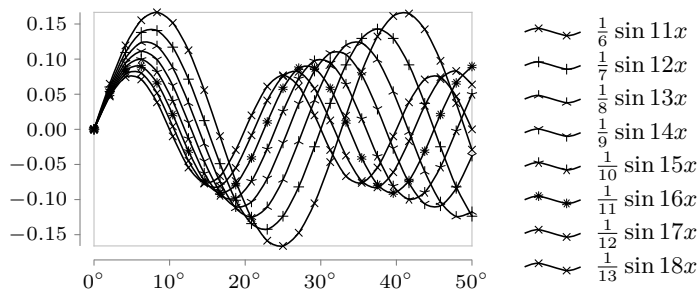
为了能使用多种点标记符号，在使用散点样式表之前，先在导言区调用程序库 `plotmarks`。

`style sheet=cross marks`

这个样式表中的各个样式的区别在于标记散点的标记符号不同，这些标记符号不是随意选择的。当两个样式的标记符号重叠时，仍然能分辨它们，所以在使用多个散点显像器时，建议使用这个样式表。这个样式表共有 6 个样式，分别提供一种散点标记符号。第 1 个样式是叉号，第 2 个样式是加号。如果显像器个数多于 6 个，则第 7 个以及以后的显像器都自动使用第一个样式的散点标记符号，此时可能需要手工设置这些显像器的标记符号。



```
\tikz \datavisualization [
  visualize as scatter/.list=
    {1,2,3,4,5,6,7,8},
  example visualization,
  style sheet=cross marks]
data group {sin functions};
```

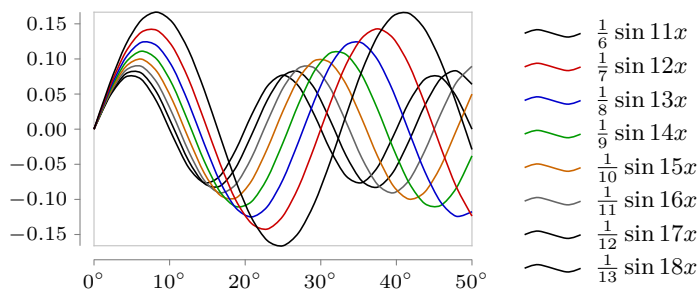


```
\tikz \datavisualization [
  visualize as smooth line/.list=
    {1,2,3,4,5,6,7,8},
  example visualization,
  style sheet=cross marks]
data group {sin functions};
```

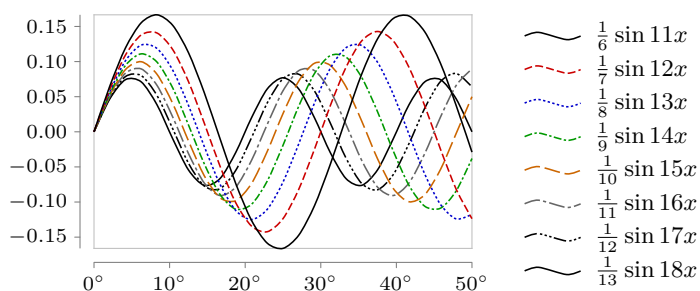
79.7 预定义的颜色样式表

style sheet=strong colors

前面已经提到，这个样式表中各个样式使用的颜色具有（在白色背景下）最大的视觉对比效果。这个样式表共有 6 个样式，第 1 个样式是黑色，第 2 个样式是红色。如果显像器个数多于 6 个，则第 7 个以及以后的显像器都自动使用第一个样式的颜色。



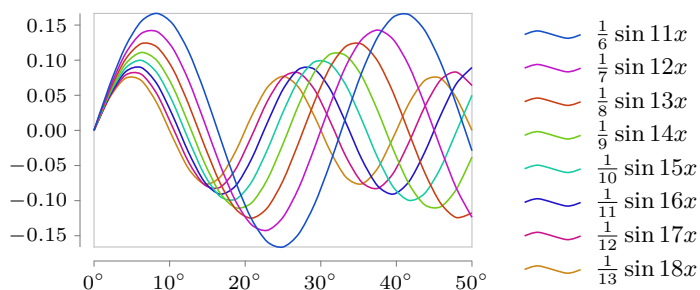
```
\tikz \datavisualization [
  visualize as smooth line/.list=
    {1,2,3,4,5,6,7,8},
  example visualization,
  style sheet=strong colors]
data group {sin functions};
```



```
\tikz \datavisualization [
  visualize as smooth line/.list=
    {1,2,3,4,5,6,7,8},
  example visualization,
  style sheet=strong colors,
  style sheet=vary dashed]
data group {sin functions};
```

style sheet=vary hue

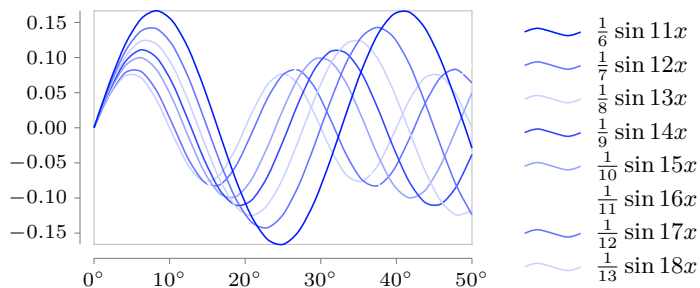
单词 hue 的意思是：样子，色度，色彩，叫声，色调。这个样式表中的各个样式的区别在于 hue 不同。



```
\tikz \datavisualization [
  visualize as smooth line/.list=
    {1,2,3,4,5,6,7,8},
  example visualization,
  style sheet=vary hue]
data group {sin functions};
```

style sheet=shades of blue

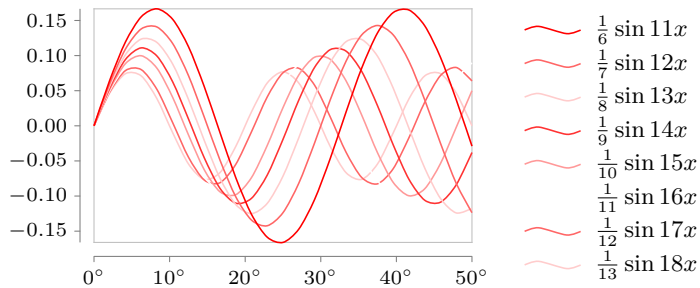
这个样式表中的各个样式都使用蓝色，区别在于蓝色的深浅不同。



```
\tikz \datavisualization [
  visualize as smooth line/.list=
    {1,2,3,4,5,6,7,8},
  example visualization,
  style sheet=shades of blue]
data group {sin functions};
```


style sheet=shades of red

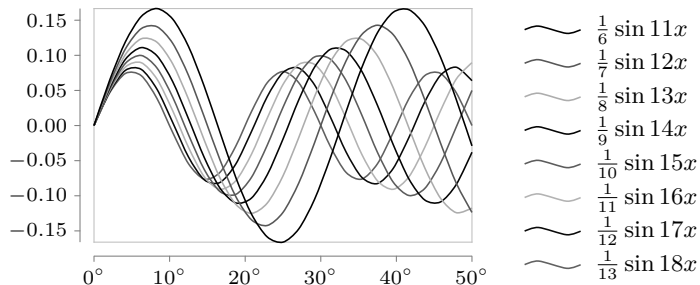
这个样式表中的各个样式都使用红色，区别在于红色的深浅不同。



```
\tikz \datavisualization [
  visualize as smooth line/.list=
    {1,2,3,4,5,6,7,8},
  example visualization,
  style sheet=shades of red]
data group {sin functions};
```

style sheet=gray scale

这个样式表中的各个样式都使用灰色，区别在于灰色的深浅不同。



```
\tikz \datavisualization [
  visualize as smooth line/.list=
    {1,2,3,4,5,6,7,8},
  example visualization,
  style sheet=gray scale]
data group {sin functions};
```

79.8 显像器的标签

一组数据点对应一个显像器名称，相应的显像器将该数据点组可视化为一个图形，在二维之下这个图形通常是一组散点或一个曲线。由于一个显像器处理一组数据点，可以把一组数据点看作是属于相应显像器的数据点。显像器决定这组数据点的可视化外观，其中包括这组数据点的标签，也就是这组数据点的图形的标签。因为一组数据点对应一个显像器，数据点的标签也可以看作是显像器的标签。例如对于曲线来说，这种标签会放在该曲线上某一点的旁边，起到注释曲线的作用。

设置这种标签的选项一般应作为显像器的参数。

79.8.1 给一组数据点设置标签

下一选项设置某个显像器的标签。

```
/tikz/data visualization/visualizer options/label in data=<options>
```

这个选项作为显像器的参数，给一组数据点设置标签。可以在一个显像器的参数中多次使用这个选项，给显像器设置多个标签。这里 <options> 是关于标签的设置，其中的选项会被冠以前缀

```
/tikz/data visualization/visualizer label options
```

来执行。在 <options> 中可以使用下面介绍的选项。

```
/tikz/data visualization/visualizer label options/text=<text>
```

这个选项设置标签的文字内容，该文字标签通常位于图形的左侧。该选项的定义中使用了 auto 选项。

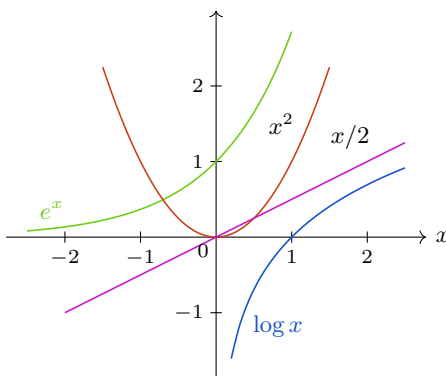
`/tikz/data visualization/visualizer label options/text'=<text>`

这个选项设置标签的文字内容,该文字标签通常位于图形的右侧。该选项的定义中使用了 `auto,swap` 选项。

假设一组数据点可视化为一个曲线,其中各个数据点的先后次序决定曲线的方向。在给数据点添加标签时,通常是采用某个方法选定某个数据点 A,记该点之后的数据点是 B,从 A 到 B 决定一个有向线段,这个线段有“左侧”和“右侧”,标签就放在其左侧(对应选项 `text`)或者右侧(对应选项 `text'`)且位于 A 点附近,即标签以 A 为指向点。所选的指向点 A 会被储存在 `(label visualizer coordinate)` 中,而其后的点 B 会被储存在 `(label visualizer coordinate')` 中。标签的样式则可以使用 `node style` 来设置。

`/tikz/data visualization/visualizer label options/when=<attribute> is <number>`

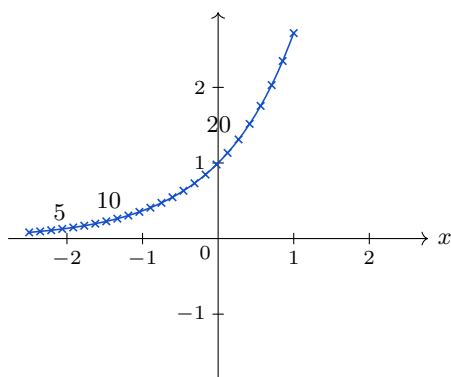
`<attribute>` 是数据的标识符或者说变量名。以绘制函数曲线来讲,一般默认使用 25 个样本点(即数据点),假如设置 `when=x is 5.3`,则规定曲线标签的指向点是某个样本点(`(label visualizer coordinate)`),该样本点的坐标的 `x` 分量值不小于 `x=5.3`。如果不小于 `x=5.3` 的样本点找不到,就把最后一个数据点(样本点)作为指向点。



```
\tikz \datavisualization [
  school book axes,
  x axis={label=$x$},
  visualize as smooth line/.list={log, lin, squared, exp},
  log= {label in data={text'=$\log x$, when=y is -1, text colored}},
  lin= {label in data={text=$x/2$, when=x is 2}},
  squared={label in data={text=$x^2$, when=x is 1.1}},
  exp= {label in data={text=$e^x$, when=x is -2, text colored}},
  style sheet=vary hue]
data group {function classes};
```

`/tikz/data visualization/visualizer label options/index=<number>`

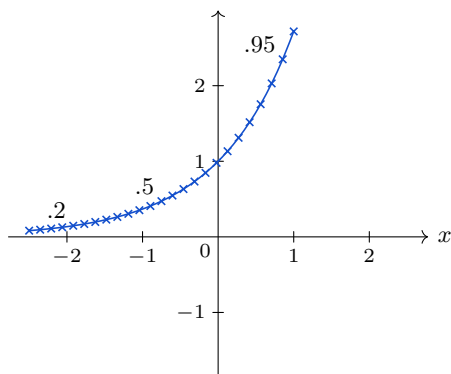
这里 `<number>` 是个正整数,作为序号来索引数据点。该选项选定第 `<number>` 个数据点(样本点)作为标签的指向点。



```
\tikz \datavisualization [
  school book axes,
  x axis={label=$x$},
  visualize as smooth line/.list={exp},
  exp={label in data={text=$5$, index=5},
    label in data={text=$10$, index=10},
    label in data={text=$20$, index=20},
    style={mark=x}},
  style sheet=vary hue]
data group {function classes};
```

`/tikz/data visualization/visualizer label options/pos=<fraction>`

`<fraction>` 是个小数，记该选项所在的显像器的数据点总数为 `<total>`，则该选项使得索引序号 `index=<number>` 不小于 `<fraction>` 与 `<total>` 的乘积。



```
\tikz \datavisualization [
  school book axes,
  x axis={label=$x$},
  visualize as smooth line=exp,
  exp= {label in data={text=$.2$, pos=0.2},
    label in data={text=$.5$, pos=0.5},
    label in data={text=$.95$, pos=0.95},
    style={mark=x}},
  style sheet=vary hue]
data group {function classes};
```

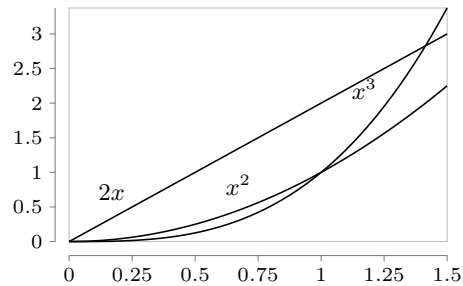
`/tikz/data visualization/visualizer label options/auto`

这个选项是默认执行的选项，即把 `label in data` 作为显像器的参数后，如果不用其它选项指定标签的指向点，就用该选项来指定。

该选项作为某个显像器的参数，对应某组数据点 `<data set>`。在可视化过程中可能会有多组数据点，设共有 `<total number of data sets>` 组数据点，而数据点组 `<data set>` 的索引序号是 `<data set's index>`（这里 `<data set's index>` 是个正整数），也就是说数据点组 `<data set>` 是第 `<data set's index>` 组数据点。该选项确定数据点组 `<data set>` 的标签的指向点时所用的算式是：

$$\text{pos} = (\text{<data set's index>} - 1/2) / \text{<total number of data sets>}$$

假设数据点共有 10 组，分别对应 10 个显像器，则该选项确定第 1 个显像器的标签的指向点是 $\text{pos} = (1 - 1/2) / 10 = 0.05$ ；该选项确定第 10 个显像器的标签的指向点是 $\text{pos} = (10 - 1/2) / 10 = 0.95$ ；这样可以尽量避免标签重叠，当然效果可能欠佳。



```
\tikz \datavisualization [
  scientific axes=clean,
  visualize as smooth line/.list={linear, squared, cubed},
  linear = {label in data={text=$2x$}},
  squared = {label in data={text=$x^2$}},
  cubed = {label in data={text=$x^3$}}
data [set=linear, format=function] {
  var x : interval [0:1.5];
  func y = 2*\value x;
}
data [set=squared, format=function] {
  var x : interval [0:1.5];
  func y = \value x * \value x;
}
data [set=cubed, format=function] {
  var x : interval [0:1.5];
  func y = \value x * \value x * \value x;
};
```

以上选项用于确定标签的位置，用下面的选项可以设置标签的样式外观。

`/tikz/data visualization/visualizer label options/node style=<options>`

该选项的 `<options>` 会被传递给 `/tikz/data visualization/node style`。 `<options>` 中的选项应该是 TikZ 的选项。

`/tikz/data visualization/visualizer label options/text colored`

要想使用这个选项就必须同时使用涉及颜色的样式表，并且该样式表中各个样式都使用 `visualizer color` 来规定颜色（见 §79.4.3）。各种预定义的涉及颜色的样式表都使用选项 `visualizer color` 来规定颜色。这个选项会在选项 `node style` 的参数中引用相应样式的 `visualizer color` 的值，使得标签的颜色为选项 `visualizer color` 规定的颜色，即使得标签与数据点的显示颜色一致。

注意尽管

```
<visualizer name>=\{style=\{visualizer color=<color>\}\}, label in data=\{text=<
  text>, text colored\}
```

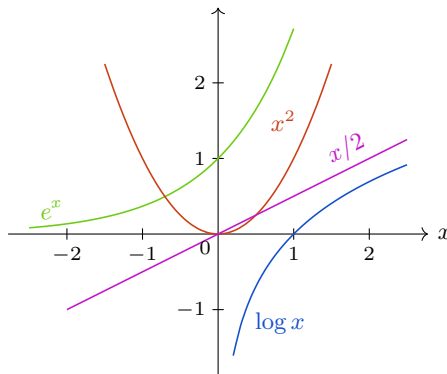
是有效的句法，但是其中的 `visualizer color` 不能被 `text colored` 调用。

另外，如果自定义的样式表中，某个样式的颜色没有使用 `visualizer color` 来规定，而是直接使用 `<color>` 选项，那么这个样式传递给显像器，显像器会使得其中的标签文字也带有这个颜色 `<color>`。

`/tikz/data visualization/every data set label (style)`

这是个样式 (style)，其中可以用 `node style` 来设置各组数据点的标签的样式。一般情况下，如果没有手工设置各标签的样式，那么各标签就使用预定义的默认样式。因为标签的默认样式有比较好的适用性，最好不要修改，所以通常为默认样式附加某些样式来做适当调整。

下面的例子用该选项为默认样式附加其它样式，其中使用的手柄是 `/.append style`：



```
\tikz \datavisualization [
  school book axes,
  x axis={label=$x$},
  visualize as smooth line/.list={log, lin, squared, exp},
  every data set label/.append style={text colored},
  log= {label in data={text'=$\log x$, when=y is -1}},
  lin= {label in data={text=$x/2$, node style=sloped, when=x is 2}},
  squared={label in data={text=$x^2$, when=x is 1.1}},
  exp= {label in data={text=$e^x$, node style=sloped, when=x is -2}},
  style sheet=vary hue]
data group {function classes};
```

`/tikz/data visualization/every label in data (style)`

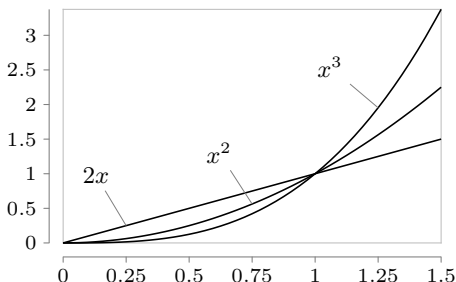
这是个样式 (style)，类似 `every data set label`，二者的区别是，在各个样式表 (style sheet) 被执行后才会执行该选项，这就多了一次修改各个标签样式的机会。

79.8.2 给一组数据点设置大头针标签

如果把 `pin in data` 作为显像器的参数，就产生大头针标签。

`/tikz/data visualization/visualizer options/pin in data=<options>`

这个选项与 `label in data` 类似，`<options>` 中也是使用前文介绍的 `text`, `text'` 选项来设置标签的文字内容（作为大头针的“头”），可以使用 `pos` 等选项来确定标签的指向点（针尖的指向位置）。



```
\tikz \datavisualization [
  scientific axes=clean,
  visualize as smooth line/.list={linear, squared, cubed},
  linear ={pin in data={text=$2x$}},
  squared={pin in data={text=$x^2$}},
  cubed ={pin in data={text=$x^3$}}
data [set=linear, format=function] {
  var x : interval [0:1.5];
  func y = \value x;
}
data [set=squared, format=function] {
  var x : interval [0:1.5];
  func y = \value x * \value x;
}
data [set=cubed, format=function] {
  var x : interval [0:1.5];
  func y = \value x * \value x * \value x;
};
```

上面这个例子就是前文中的例子，二者比较来说，使用大头针标签更清楚明确。

此外还有以下选项可用在 `<options>` 中。

`/tikz/data visualization/visualizer label options/pin angle=<angle>`

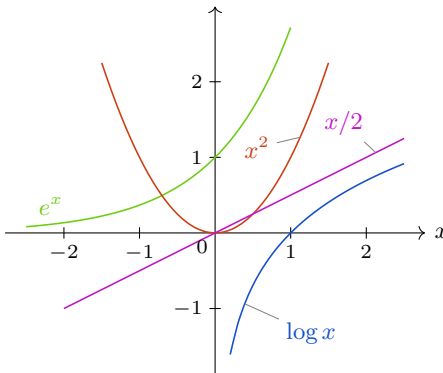
这个选项会使得“大头针”绕它的针尖（标签的指向点）旋转，`<angle>` 用于规定旋转的角度。但是具体的旋转机制并不明显，一般正值角度对应逆时针旋转，负值角度对应顺时针旋转。

如果不指定 `<angle>`，则默认为：标签的“针”与线段 (`label visualizer coordinate`)--(`label visualizer coordinate'`) 垂直，垂足为 (`label visualizer coordinate`)；如果使用选项 `text = <text>`，则标签在该线段左侧；如果使用选项 `text'=<text>`，则标签在该线段右侧。

`/tikz/data visualization/visualizer label options/pin length=<dimension>`

这个选项指定大头针标签的“针”的长度，在大头针围绕针尖的指向点旋转时会保持“针”的长度

不变。注意 $\langle \text{dimension} \rangle$ 是带单位的尺寸。如果不指定 $\langle \text{dimension} \rangle$ ，那么“针”的长度与选项 $\text{pin angle}=\langle \text{angle} \rangle$ 有关。



```
\tikz \datavisualization [
  school book axes,
  x axis={label=$x$},
  visualize as smooth line/.list={log, lin, squared, exp},
  every data set label/.append style={text colored},
  log= {pin in data={text'=$\log x$, when=y is -1}},
  lin= {pin in data={text=$x/2$, when=x is 2, pin length=1ex}},
  squared={pin in data={text=$x^2$, when=x is 1.1, pin angle=230}},
  exp= {label in data={text=$e^x$, when=x is -2}},
  style sheet=vary hue]
data group {function classes};
```

在默认下，大头针标签的“针”的样式是 `help lines`，这个 key 的初始值是 `line width=0.2pt,gray`。在 §17.10.3 中讲到 `node` 的大头针标签时，“针”的样式可以用样式 `every pin edge` 和选项 `pin edge` 来修改。但是在可视化命令中这两种 key 都难以起作用，很难修改单个大头针标签的“针”的样式。如果要统一修改大头针标签的“针”的样式，可以用以下形式：

```
\tikzset {every pin edge/.style={...}}
或者
\tikz [every pin edge/.style={...}] \datavisualization...;
或者
\begin{tikzpicture}[every pin edge/.style={...}]
  ...
\end{tikzpicture}
```

79.9 使用图例

图例要比标签复杂。一个可视化图形中可有多个图例，一般情况下，一个图例中含有数个条目，或者称为条目标签，条目是对某一显像器产生的数据点图形的注释。也就是说，条目直接与显像器对应。使用图例的通常做法是：首先创建一个或者数个自命名的图例；然后，给需要图例条目注释的显像器使用参数 `label`

in legend=<options>, 来产生一个对应该显像器的条目, 并在 <options> 中使用 legend=<name> 来指定该条目属于那个图例; 如果不创建自命名的图例, 就默认使用名称为 main legend 的图例, 并且不必在显像器的参数 label in legend=<options> 中使用 legend=main legend 来指定该条目属于默认图例。

一个条目一般包含两部分, 文字标签和图示标签。文字标签是用文字做的注释, 通常由选项 text=<text> 产生。图示标签是用图形做的注释, 由“条目显像器”产生, 在默认下是个之字形 (zig-zag) 线。图例在整体上是个矩阵 (matrix) node, 它的条目就是它的元素 (关于矩阵的内容参考 §20, §57)。关于图例的设置就分为矩阵整体的设置和各个条目的设置。图例涉及的 key 比较多, 这里先概略梳理一下。

以下 3 个 key:

```
/tikz/data visualization/new legend=<legend name>
/tikz/data visualization/<legend name>=<options>
/tikz/data visualization/legend=<options>
```

其前缀路径都是 /tikz/data visualization/, 故它们都直接用作可视化命令的选项。第 1 个 key 声明一个图例, 可以用这个选项声明多个图例, 即可以在图形中使用多个图例。第 2 个 key 引起对图例 <legend name> 的整体设置或者其中所有条目的统一设置。第 3 个 key 引起对默认图例的整体设置或者其中所有条目的统一设置。

针对图例整体做设置的 key 一般都有前缀路径:

```
/tikz/data visualization/legend options/
```

它们都用作 <legend name> 或 legend 的参数。

引起单个条目设置的 key 是:

```
/tikz/data visualization/visualizer options/label in legend=<options>
```

按照这个 key 的路径, 它只能用作显像器名称的参数。它引起与所在显像器相关联的图例条目的设置。

针对单个条目做设置的 key 都有前缀路径:

```
/tikz/data visualization/legend entry options/
```

它们可以作 label in legend 的参数来设置单个条目; 也可以用作 <legend name> 的参数来对该图例中的所有条目做统一设置; 也可以用作 legend 的参数来对默认图例中的所有条目做统一设置。

另外还有一些以 /tikz/data visualization/ 为前缀的样式也用于设置图例, 它们直接用作可视化命令的选项。

79.9.1 创建图例, 图例中的条目

在默认之下是不给各组数据点创建图例标签的, 要想使用图例就要用相关选项做设置。

```
/tikz/data visualization/new legend=<legend name> (默认值 main legend)
```

这个选项创建一个名称为 <legend name> 的作为矩阵整体的图例, 其默认名称为 main legend. 如果用这个选项创建两个名称相同的图例, 则第 2 个无效, 也就是说一个图例名称只能用一次。

当使用这个选项后, 程序会自动创建下面的键:

```
/tikz/data visualization/<legend name>
```

可以赋予这个键选项值

```
/tikz/data visualization/<legend name>=<options>
```

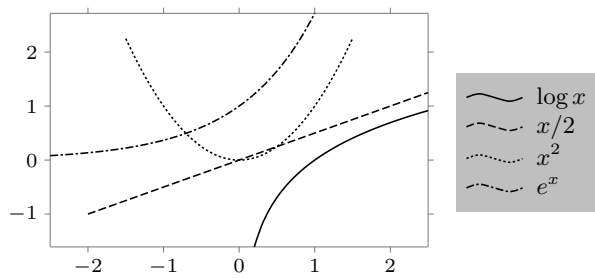

这里 `<options>` 中的选项会被冠以前缀

```
/tikz/data visualization/legend options
```

来执行，用以确定图例的位置，外观，元素条目的排布方式等等。这里的情况类似显像器，使用显像器时，给显像器命名，然后给显像器名称赋选项值，设置显像器的外观效果。

```
/tikz/data visualization/legend options/matrix node style=<options>
```

这个选项设置图例的样式，针对作为 `matrix node` 的图例整体，而不是其中的各个条目。



```
\tikz \datavisualization [
  scientific axes,
  visualize as smooth line/.list={log, lin, squared, exp},
  legend={matrix node style={fill=black!25}}, % 图例的填充色为 25% 黑
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$}},
  squared={label in legend={text=$x^2$}},
  exp= {label in legend={text=$e^x$}},
  style sheet=vary dashed]
data group {function classes};
```

```
/tikz/data visualization/legend options/every new legend (style)
```

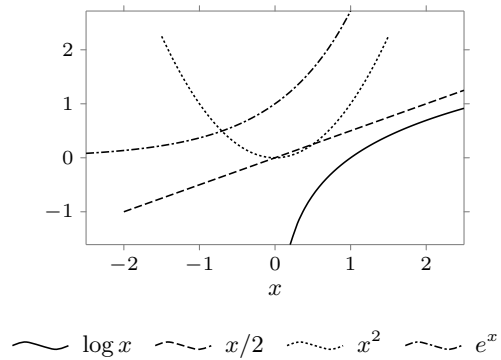
这个样式的默认设置是: `east outside, label style=text right`，即图例位于绘图区域之外且在绘图区的东方（右侧），各个条目中的文字在条目的右侧，如前面的例子所示。

```
/tikz/data visualization/legend=<options>
```

这个选项是

```
new legend=main legend, main legend=<options>
```

的简化，即这个选项创建一个名称为 `main legend` 的图例，并将 `<options>` 赋予这个图例。

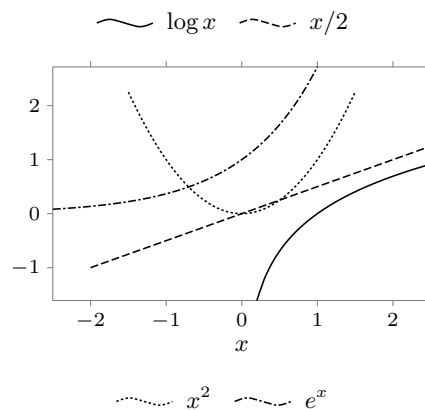


```

\tikz \datavisualization [
  scientific axes, x axis={label=$x$},
  visualize as smooth line/.list={log, lin, squared, exp},
  legend=below,
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$}},
  squared={label in legend={text=$x^2$}},
  exp= {label in legend={text=$e^x$}},
  style sheet=vary dashed]
data group {function classes};

```

上面的例子中使用选项 `legend=below` 创建名称为 `main legend` 的图例，并将它放在绘图区的下方。



```

\tikz \datavisualization [
  scientific axes, x axis={label=$x$},
  visualize as smooth line/.list={log, lin, squared, exp},
  new legend={upper legend},
  new legend={lower legend},
  upper legend=above,
  lower legend=below,
  log= {label in legend={text=$\log x$, legend=upper legend}},

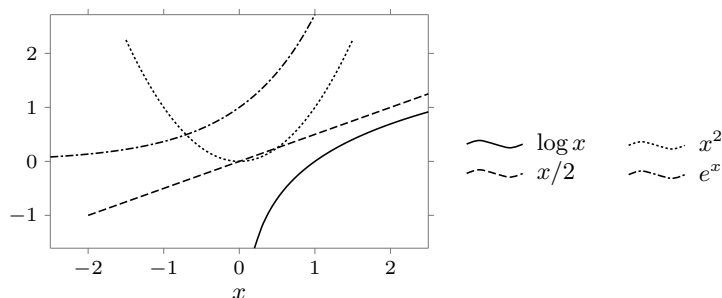
```

```

lin= {label in legend={text=$x/2$, legend=upper legend}},
squared={label in legend={text=$x^2$, legend=lower legend}},
exp= {label in legend={text=$e^x$, legend=lower legend}},
style sheet=vary dashed]
data group {function classes};

```

上面的例子中，设置了两个图例 upper legend 和 lower legend，它们的位置分别是 above 和 below。然后在显像器的参数中使用 legend=upper legend 选项将相应数据点的标签放在图例 upper legend 中。对比下面的例子。



```

\tikz \datavisualization [
  scientific axes, x axis={label=$x$},
  visualize as smooth line/.list={log, lin, squared, exp},
  log= {label in legend={text=$\log x$, legend=above}},
  lin= {label in legend={text=$x/2$, legend=above}},
  squared={label in legend={text=$x^2$, legend=below}},
  exp= {label in legend={text=$e^x$, legend=below}},
  style sheet=vary dashed]
data group {function classes};

```

在这个例子中只有一个图例，即选项 legend 所默认规定的图例，其名称为 main legend，显像器中的图例选项 legend=above 和 legend=below 只是使得标签分为两列。

/tikz/data visualization/visualizer options/label in legend=<options>

这个选项用在显像器的参数中，为该显像器的数据点生成图例条目，并放在某个指定的图例盒子 (matrix node) 中。在 <options> 中的选项会被冠以前缀

```
/tikz/data visualization/legend entry options
```

来执行。

在 <options> 中可以使用以下选项。

/tikz/data visualization/legend entry options/legend=<name> (无默认值，初始值 main legend)

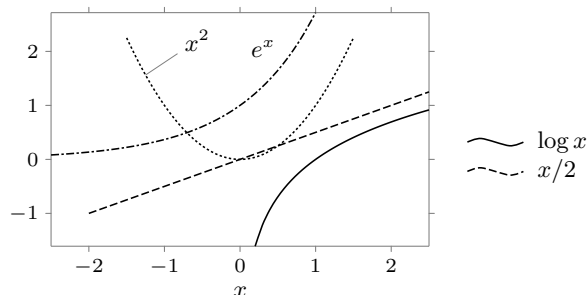
这个选项规定所生成的图例条目标签属于名称为 <name> 的图例 (即放在这个图例盒子中)。这里的 <name> 可以是选项 new legend=<name> 设置的图例名称。如果没有用选项 new legend 声

明图例名称，就使用该选项的默认名称 `main legend`；如果打算使用默认的图例名称 `main legend`，就无需明确写出 `legend=main legend`。

如果 `legend=<name>` 是之前没有被声明的图例名称，则程序会自动创建这个图例名称。

`/tikz/data visualization/legend entry options/text=<text>`

这个选项设置条目标签的文字内容。



```
\tikz \datavisualization [
  scientific axes, x axis={label=$x$},
  visualize as smooth line/.list={log, lin, squared, exp},
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$}},
  squared={pin in data = {text=$x^2$, pos=0.1}},
  exp= {label in data = {text=$e^x$}},
  style sheet=vary dashing]
data group {function classes};
```

79.9.2 图例中条目的行列排布

一个图例是个 `matrix node`，其中的条目作为它的元素排成行、列。图例条目有多个排布规则，每个规则对应一个 `key`，这些 `key` 一般应作为图例名称的参数，即选项 `new legend=<legend name>` 设置的图例名称的参数，放在 `<legend name>=<options>` 中的 `<options>` 里。

如果不设置图例名称，那么可以使用默认的图例名称 `main legend`，即把这些 `key` 放在 `main legend = <options>` 中的 `<options>` 里；也可以把这些 `key` 作为选项 `legend` 的参数，即把这些 `key` 放在 `legend = <options>` 中的 `<options>` 里，因为选项 `legend` 会默认图例名称为 `main legend`，所以这两个方式是等效的。

为了展示这些 `key` 的效果，先定义一个样式 `legend example`：

```
\tikzdatavisualizationset {
  legend example/.style={
    scientific axes, all axes={length=1cm, ticks=none},
    1={label in legend={text=1}},
    2={label in legend={text=2}},
    3={label in legend={text=3}},
```

```

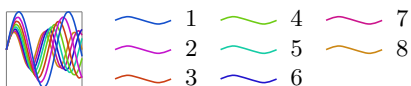
4={label in legend={text=4}},
5={label in legend={text=5}},
6={label in legend={text=6}},
7={label in legend={text=7}},
8={label in legend={text=8}}
}
}

```

`/tikz/data visualization/legend options/down then right`

这个选项确定图例条目的排布方式是：先自上而下纵向排成一列，当该列达到某个行数后，转到该列的右侧再自上而下排一列，如此继续。

这个选项是默认选项。



```

\tikz \datavisualization [
    visualize as smooth line/.list={1,2,3,4,5,6,7,8},
    legend example, style sheet=vary hue,
    main legend={down then right, columns=3}]
data group {sin functions};

```

`/tikz/data visualization/legend options/down then left`

这个选项确定图例条目的排布方式是：先自上而下纵向排成一列，当该列达到某个行数后，转到该列的左侧再自上而下排一列，如此继续。



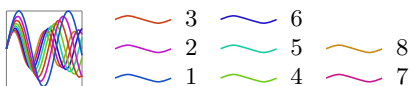
```

\tikz \datavisualization [
    visualize as smooth line/.list={1,2,3,4,5,6,7,8},
    legend example, style sheet=vary hue,
    legend={down then left, columns=3}]
data group {sin functions};

```

`/tikz/data visualization/legend options/up then right`

这个选项确定图例条目的排布方式是：先自下而上纵向排成一列，当该列达到某个行数后，转到该列的右侧再自下而上排一列，如此继续。



```

\tikz \datavisualization [
    visualize as smooth line/.list={1,2,3,4,5,6,7,8},
    legend example, style sheet=vary hue,
    legend={up then right, columns=3}]
data group {sin functions};

```

/tikz/data visualization/legend options/up then left

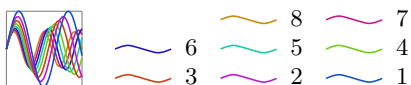
这个选项确定图例条目的排布方式是：先自下而上纵向排成一列，当该列达到某个行数后，转到该列的左侧再自下而上排成一列，如此继续。



```
\tikz \datavisualization [
  visualize as smooth line/.list={1,2,3,4,5,6,7,8},
  legend example, style sheet=vary hue,
  legend={up then left, columns=3}]
data group {sin functions};
```

/tikz/data visualization/legend options/left then up

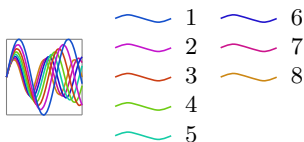
这个选项确定图例条目的排布方式是：先自右而左横向排成一行，当该行达到某个列数后，转到该行的上方再自右而左排成一行，如此继续。



```
\tikz \datavisualization [
  visualize as smooth line/.list={1,2,3,4,5,6,7,8},
  legend example, style sheet=vary hue,
  legend={left then up, columns=3}]
data group {sin functions};
```

/tikz/data visualization/legend options/left then down**/tikz/data visualization/legend options/right then up (no****/tikz/data visualization/legend options/right then down****/tikz/data visualization/legend options/max rows=<number>**

这个选项规定图例矩阵的最大行数。



```
\tikz \datavisualization [
  visualize as smooth line/.list={1,2,3,4,5,6,7,8},
  legend example, style sheet=vary hue,
  main legend={max rows=5}]
data group {sin functions};
```

/tikz/data visualization/legend options/max columns=<number>

这个选项规定图例矩阵的最大列数。注意只有使用“先左右，后上下”的条目排布方式时，这个选项才有效。



```
\tikz \datavisualization [
  visualize as smooth line/.list={1,2,3,4,5,6,7,8},
  legend example, style sheet=vary hue,
  legend={right then down, max columns=2}]
data group {sin functions};
```

`/tikz/data visualization/legend options/ideal number of columns=<number>`

这个选项设置一个“理想的”列数，程序会尽量将条目排成这个列数，但实际的排布列数还要参照选项 `max rows` 来确定，也就是说，选项 `max rows` 要比这个选项更有优先地位。

`/tikz/data visualization/legend options/rows=<number>`

这是 `ideal number of rows=<number>` 的简写。

`/tikz/data visualization/legend options/ideal number of rows=<number>`

这个选项设置一个“理想的”列数，程序会尽量将条目排成这个列数，但实际的排布列数还要参照选项 `max columns` 来确定，也就是说，选项 `max columns` 要比这个选项更有优先地位。

`/tikz/data visualization/legend options/columns=<number>`

这是 `ideal number of columns=<number>` 的简写。

79.9.3 确定图例位置的一般方法

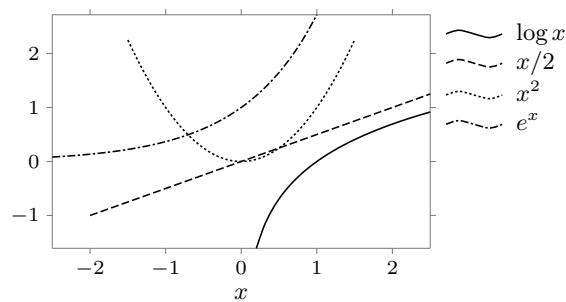
图例是个 `matrix node`，作为一个 `node` 它有自己的各种“部位”、“锚位”，也可以使用 `at` 选项来确定它的指向点。

`/tikz/data visualization/legend options/anchor=<anchor>`

这个选项用作图例名称的参数或者 `legend` 的参数，该选项把图例的锚位 `<anchor>` 放在其指向点上。

`/tikz/data visualization/legend options/at=<coordinate>`

这个选项用作图例名称的参数或者 `legend` 的参数，该选项确定图例的指向点。不过注意这里的 `<coordinate>` 必须是与预定义的 `data bounding box` 或 `data visualization bounding box` 有关的点，不能直接使用数值坐标。这两个预定义 `node` 见 §75.6 的结尾。



```
\tikz \datavisualization [
  scientific axes, x axis={label=$x$},
```

```

visualize as smooth line/.list={log, lin, squared, exp},
legend={anchor=north west, at=(data visualization bounding box.north east)},
log= {label in legend={text=$\log x$}},
lin= {label in legend={text=$x/2$}},
squared={label in legend={text=$x^2$}},
exp= {label in legend={text=$e^x$}},
style sheet=vary dashed]
data group {function classes};

```

at=<coordinate> 还可以使用如下的坐标计算格式:

```

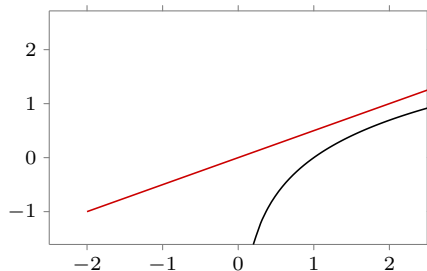
at=( $(data visualization bounding box.east)+(90:1)$ )
at=( $(data visualization bounding box.east)+(10pt, 1cm)$ )

```

在这种计算格式中的坐标点的默认长度单位都是 cm，并不是可视化命令中轴系统中的坐标点。

/tikz/data visualization/legend options/at values={<data point>}

与 at= 选项不同，这里的 <data point> 是可视化命令中轴系统中的坐标点，例如 x=1, y=2，使用这个选项选定数据点后，默认图例的中心点处于这个数据点位置上。注意这里使用 <attribute>=<value> 的形式，要写出正确的 <attribute> 名称。



$\log x$

 $x/2$

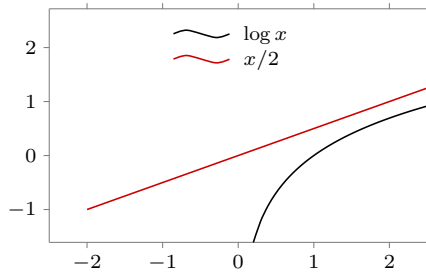
```

\tikz \datavisualization [
  scientific axes,
  visualize as smooth line/.list={log, lin},
  legend={at values={x=5, y=5}},
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$}},
  style sheet=strong colors]
data group {function classes};

```

/tikz/data visualization/legend options/right of=<data point>

这里的 <data point> 是可视化命令中轴系统中的坐标点，这个选项会使得图例处于该数据点的右侧，即图例的锚位 west 处于数据点 <data point> 的位置上。



```
\tikz \datavisualization [
  scientific axes,
  visualize as smooth line/.list={log, lin},
  legend={right of={x=-1, y=2}},
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$}},
  style sheet=strong colors]
data group {function classes};
```

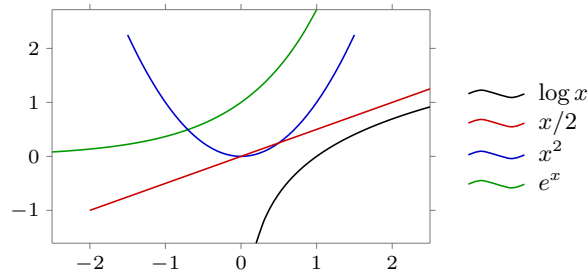
```
/tikz/data visualization/legend options/above right of=<data point>
/tikz/data visualization/legend options/above of=<data point>
/tikz/data visualization/legend options/above left of=<data point>
/tikz/data visualization/legend options/left of=<data point>
/tikz/data visualization/legend options/below left of=<data point>
/tikz/data visualization/legend options/below of=<data point>
/tikz/data visualization/legend options/below right of=<data point>
```

79.9.4 在绘图区域之外放置图例

下面的选项作为图例名称的参数或者 legend 的参数，在绘图区域之外放置图例。

```
/tikz/data visualization/legend options/east outside
```

将图例放在绘图区域之外的右侧，这是默认的放置方式。



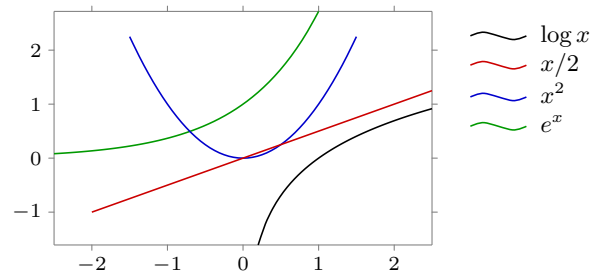
```
\tikz \datavisualization [
  scientific axes,
  visualize as smooth line/.list={log, lin, squared, exp},
  legend=east outside,
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$}},
  squared={label in legend={text=$x^2$}},
  exp= {label in legend={text=$e^x$}},
  style sheet=strong colors]
data group {function classes};
```

`/tikz/data visualization/legend options/right`

等价于 `east outside`.

`/tikz/data visualization/legend options/north east outside`

将图例放在绘图区域之外的右上角。



```
\tikz \datavisualization [
  scientific axes,
  visualize as smooth line/.list={log, lin, squared, exp},
  legend=north east outside,
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$}},
  squared={label in legend={text=$x^2$}},
  exp= {label in legend={text=$e^x$}},
  style sheet=strong colors]
data group {function classes};
```

`/tikz/data visualization/legend options/south east outside`

将图例放在绘图区域之外的右下角。

`/tikz/data visualization/legend options/west outside`

将图例放在绘图区域之外的左侧。

`/tikz/data visualization/legend options/left`

等价于 `west outside`.

`/tikz/data visualization/legend options/north west outside`

将图例放在绘图区域之外的左上角。

`/tikz/data visualization/legend options/south west outside`

将图例放在绘图区域之外的左下角。

`/tikz/data visualization/legend options/north outside`

将图例放在绘图区域之外的上部。

`/tikz/data visualization/legend options/above`

等价于 `north outside`.

```
/tikz/data visualization/legend options/south outside
```

将图例放在绘图区域之外的下部。

```
/tikz/data visualization/legend options/below
```

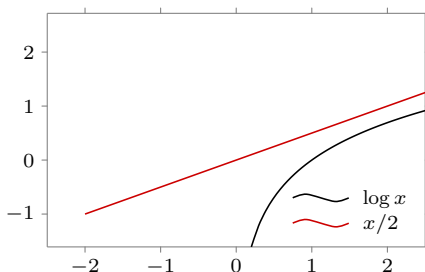
等价于 `south outside`.

79.9.5 在绘图区域之内放置图例

下面的选项作为图例名称的参数或者 `legend` 的参数，在绘图区域之内放置图例。在默认下，绘图区域之内的图例的外观样式与绘图区域之外的图例的外观样式有所不同。绘图区域之内的图例（作为 `matrix node`）会被用不透明的白色填充，因此可能会遮挡数据点，边界是圆角矩形，条目中文字的尺寸是脚注尺寸。

```
/tikz/data visualization/legend options/south east inside
```

将图例放在绘图区域之内的右下角。



```
\tikz \datavisualization [
  scientific axes,
  visualize as smooth line/.list=
    {log, lin},
  legend=south east inside,
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$}},
  style sheet=strong colors]
data group {function classes};
```

```
/tikz/data visualization/legend options/every legend inside (style)
```

这个样式的默认设置是 `opaque` 并且标签文字尺寸是脚注尺寸。

```
/tikz/data visualization/legend options/opaque=<color> (默认值 white)
```

设置图例的填充颜色为 `<color>`，默认颜色是白色，并且还使得边界为圆角。

```
/tikz/data visualization/legend options/transparent
```

使得图例透明，即没有填充色：`opaque=none`.

```
/tikz/data visualization/legend options/east inside
```

```
/tikz/data visualization/legend options/north east inside
```

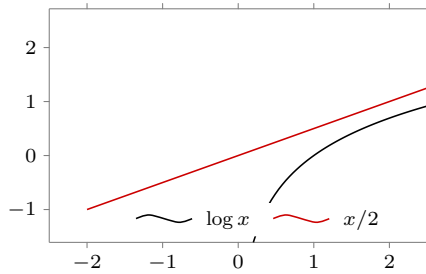
```
/tikz/data visualization/legend options/south west inside
```

```
/tikz/data visualization/legend options/west inside
```

```
/tikz/data visualization/legend options/north west inside
```

```
/tikz/data visualization/legend options/south inside
```

这个选项会使得条目排成一行。



```
\tikz \datavisualization [
  scientific axes,
  visualize as smooth line/.list={log, lin},
  legend=south inside,
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$}},
  style sheet=strong colors]
data group {function classes};
```

`/tikz/data visualization/legend options/north inside`

这个选项会使得条目排成一行。

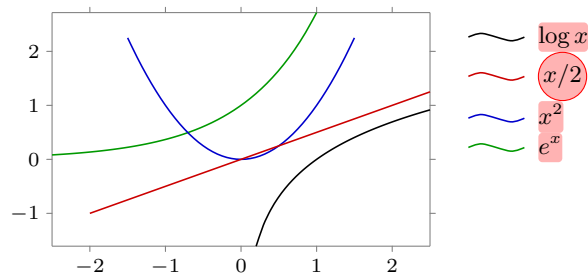
79.9.6 图例条目的一般样式

`/tikz/data visualization/every label in legend (style)`

这个样式针对所有图例中所有条目。该样式中的选项会被冠以前缀

`/tikz/data visualization/legend entry options`

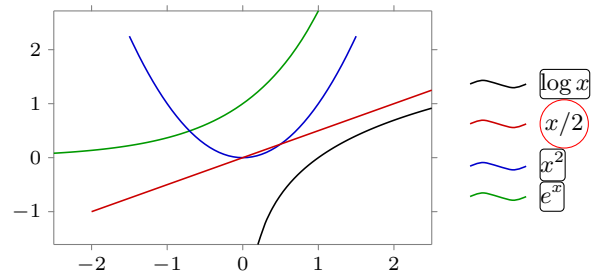
来执行。其中可以用选项 `node style` 来设置文字标签的样式。



```
\tikz \datavisualization [
  scientific axes,
  every label in legend/.style={node style={fill=red!30}},
  visualize as smooth line/.list={log, lin, squared, exp},
  legend=north east outside,
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$},
  node style={circle, draw=red}},
  squared={label in legend={text=$x^2$}},
  exp= {label in legend={text=$e^x$}},
  style sheet=strong colors]
data group {function classes};
```

`/tikz/data visualization/legend options/label style=<options>`

这个选项规定的样式会添加到样式 `every label in legend` 中。

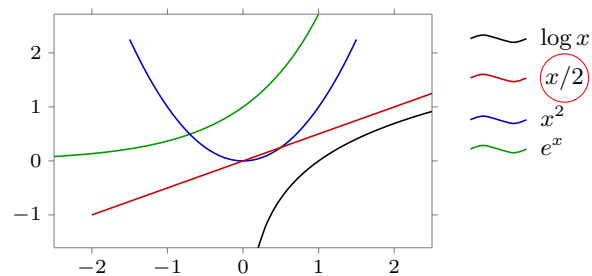


```
\tikz \datavisualization [
  scientific axes,
  visualize as smooth line/.list={log, lin, squared, exp},
  legend={label style={node style=draw}},
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$, node style={circle, draw=red}}},
  squared={label in legend={text=$x^2$}},
  exp= {label in legend={text=$e^x$}},
  style sheet=strong colors]
data group {function classes};
```

79.9.7 图例条目中的文字标签

`/tikz/data visualization/legend entry options/node style=<options>`

这个选项用于设置条目中的文字标签的样式。

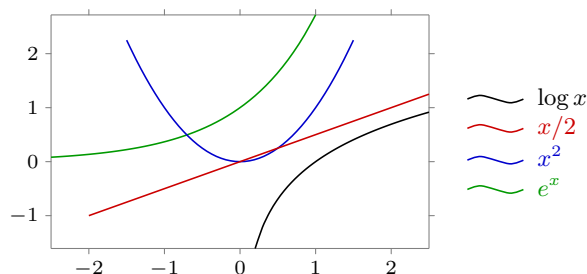


```
\tikz \datavisualization [
  scientific axes,
  visualize as smooth line/.list={log, lin, squared, exp},
  legend=north east outside,
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$,
  node style={circle, draw=red}}},
  squared={label in legend={text=$x^2$}},
  exp= {label in legend={text=$e^x$}},
```

```
style sheet=strong colors]
data group {function classes};
```

/tikz/data visualization/legend entry options/text colored

这个选项的作用与标签的 `text colored` 选项一样，将与显像器关联的选项 `visualizer color=<color>` 与条目文字标签的选项 `node style` 联系起来，使得条目文字标签颜色是 `<color>`，与显像器所显示的数据点的颜色一致。



```
\tikz \datavisualization [
  scientific axes,
  visualize as smooth line/.list={log, lin, squared, exp},
  legend={label style=text colored},
  log= {label in legend={text=$\log x$}},
  lin= {label in legend={text=$x/2$}},
  squared={label in legend={text=$x^2$}},
  exp= {label in legend={text=$e^x$}},
  style sheet=strong colors]
data group {function classes};
```

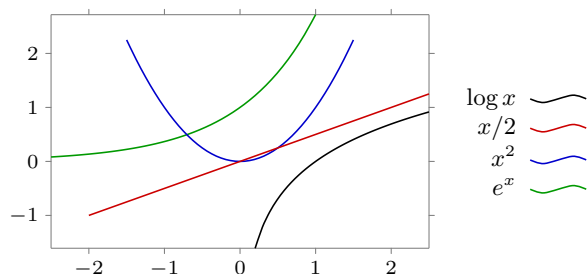
79.9.8 条目中文字标签与图示标签的相对位置

/tikz/data visualization/legend entry options/text right

这个选项使得条目中文字标签位于图示标签的右侧，在多数情况下这是默认的排列方式，但也有例外。

/tikz/data visualization/legend entry options/text left

这个选项使得条目中文字标签位于图示标签的左侧。



```
\tikz \datavisualization [
```

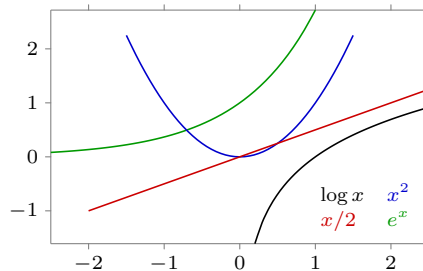
```

scientific axes,
visualize as smooth line/.list={log, lin, squared, exp},
legend={label style=text left},
log= {label in legend={text=$\log x$}},
lin= {label in legend={text=$x/2$}},
squared={label in legend={text=$x^2$}},
exp= {label in legend={text=$e^x$}},
style sheet=strong colors]
data group {function classes};

```

/tikz/data visualization/legend entry options/text only

这个选项只设置文字标签，取消图示标签，同时默认选项 `text colored` 有效，以利于把条目和数据点图形对应起来。



```

\tikz \datavisualization [
scientific axes,
visualize as smooth line/.list={log, lin, squared, exp},
legend={south east inside, rows=2,
label style=text only},
log= {label in legend={text=$\log x$}},
lin= {label in legend={text=$x/2$}},
squared={label in legend={text=$x^2$}},
exp= {label in legend={text=$e^x$}},
style sheet=strong colors]
data group {function classes};

```

79.9.9 手工添加条目

一般情况下，给某个显像器使用参数 `label in legend=<options>`，产生一个对应该显像器的条目。也可以手工添加一个不与任何显像器对应的条目，并指定该条目属于哪个图例。

/tikz/data visualization/new legend entry=<options>

这个选项创建一个条目，`<options>` 设置该条目的具体内容和样式。`<options>` 中的选项会被冠以前缀

```
/tikz/data visualization/legend entry options/
```

来执行。可以在 `<options>` 中使用 `legend=<name>` 来指定该条目属于哪个图例。条目会按照先后次序添加到图例中。如果需要把该条目放入默认的图例 `main legend` 中，就不必写出 `legend=main legend`。

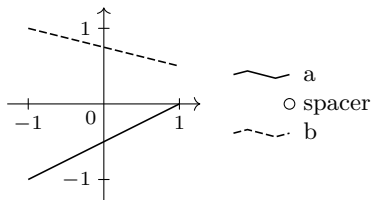
在 `<options>` 中仍然用

```
/tikz/data visualization/legend entry options/text
```

设置条目的文字标签，图示标签则下一选项设置：

```
/tikz/data visualization/legend entry options/visualizer in legend=<code>
```

这里的 `<code>` 是绘图代码，绘制图示标签。



```
\tikz \datavisualization [
  school book axes, visualize as line/.list={a,b}, style sheet=vary dashing,
  a={label in legend={text=a}},
  new legend entry={
    text=spacer,
    visualizer in legend={\draw[solid] (0,0) circle[radius=2pt];}
  },
  b={label in legend={text=b}}]
data point [x=-1, y=-1, set=a] data point [x=1, y=0, set=a]
data point [x=-1, y=1, set=b] data point [x=1, y=0.5, set=b];
```

```
/tikz/data visualization/legend entry options/setup={<set up>}
```

这个选项常用语为样式表 (style sheet) 设置标识符 (attribute)。

```
/tikz/data visualization/legend entry options/visualizer in legend style={<options>} (
style)
```

`<options>` 中的选项会被冠以 `/tikz` 来执行，即其中的选项都是 TikZ 的选项。

图例的条目是“元素图形” (cell picture, 见 §20.3)，在画出元素图形之前，下面的样式会被依次处理：

1. `/tikz/data visualization/every data set label`, 见 §79.8.1, 该样式中的选项会被冠以

```
/tikz/data visualization
```

来执行。

2. `/tikz/data visualization/every label in legend`, 见 §79.9.6, 该样式中的选项会被冠以

```
/tikz/data visualization/legend entry options
```

来执行。

3. `<options>`.

4. `/tikz/data visualization/legend entry options/setup`

5. A styling signal is emitted.

6. 处理文字标签的样式, 即 `node style` 的当前值。

7. `/tikz/data visualization/legend entry options/visualizer in legend style={<options>}`
`/tikz/data visualization/visualizer options/label in legend options=<options>`

这个选项用作显像器的参数, 设置该显像器的条目的内容、样式。

在显像器的参数值中使用选项 `label in legend=<options>` 来产生与该显像器对应的条目, 并设置条目的内容和样式, 而这个选项实际上是调用 `new legend entry` 来设置条目的。还有以下选项会影响显像器的条目样式:

1. 显像器自己的样式。
2. `/tikz/data visualization/every label in legend`
3. `/tikz/every label`
4. Setting setup to `/data point/set=<name of the visualizer>`
5. 保存在显像器中的其它选项, 这些选项可以用 `/tikz/data visualization/visualizer options/label in legend options=<options>` 修改。

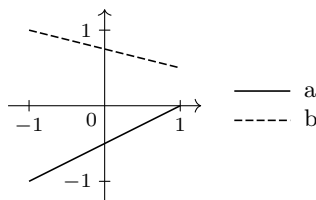
79.9.10 条目显像器

在显像器的参数值中使用选项 `label in legend=<options>` 来产生与该显像器对应的条目, 条目中的图示标签有自己的默认形式, 可以在使用下面介绍的选项来改变其默认形式。

画出图示标签的是“条目显像器”(label in legend visualizer), 它的效果受到它所对应的显像器的影响。参照 §78.3, 预定义显像器有多种, 例如 `visualize as line`, `visualize as smooth line`, `visualize as scatter`, 另外还有影响显像器的显示外观的选项, 例如, `straight line`, `straight cycle`, `polygon`, `smooth line`, `smooth cycle`, `gap line`, `gap cycle`, `no lines`. 显像器及其外观选项都会影响条目的图示标签。这是显然的, 图示标签应当与它所对应的数据点图形一致。

`/tikz/data visualization/legend entry options/default label in legend path (style)`

当显像器使用选项 `straight line`, `smooth line`, `gap line` 时, 该显像器对应的条目会使用这个样式。这个样式的默认设置是选项 `zig zag label in legend line`, 使得图示标签是之字形的, 可以改为其它形式, 例如改用选项 `straight label in legend line`, 即直线段形式的图示标签。



```
\tikz \datavisualization [
  school book axes, visualize as line/.list={a,b},
  legend entry options/default label in legend path/.style=straight label in legend
  line,
  style sheet=vary dashing,
  a={label in legend={text=a}}, b={label in legend={text=b}}]
```

```
data point [x=-1, y=-1, set=a] data point [x=1, y=0, set=a]
data point [x=-1, y=1, set=b] data point [x=1, y=0.5, set=b];
```

`/tikz/data visualization/legend entry options/default label in legend closed path (style)`

当显像器使用选项 `smooth cycle`, `straight cycle` 时, 该显像器对应的条目会使用这个样式。这个样式的默认设置是选项 `circular label in legend line`, 使得图示标签是环形的。

`/tikz/data visualization/legend entry options/default label in legend mark (style)`

当显像器使用选项 `no lines`, 并且使用散点来标记数据点时, 该显像器对应的条目会使用这个样式。这个样式的默认设置是选项 `label in legend line one mark`, 只有一个点标记符号, 可以改为其它形式, 例如改用选项 `label in legend line three marks`, 即使用 3 个点标记符号。

```
\tikz \datavisualization [
  visualize as scatter/.list={a,b,c},
  style sheet=cross marks,
  legend entry options/default label in legend mark/.style=
    label in legend three marks,
  a={label in legend={text=example a}},
  b={label in legend={text=example b}},
  c={label in legend={text=example c}}];
```

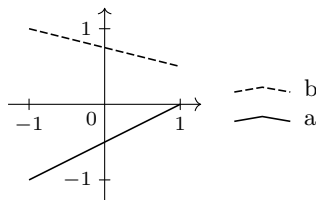
x x x example a
+ + + example b
^ ^ ^ example c

一个条目的图示标签通常是一段曲线 (可能是环形), 或者是一个或者数个散点标记符号, 或者是带有标记符号的曲线。程序会开启一个 TikZ 坐标系, 在该坐标系中绘制作为图示标签的曲线、散点标记。在默认下, 文字标签位于图示标签的右侧, 而绘制图示标签的坐标系的原点 (0,0) 位于文字标签的左侧, 并在文字基线之上 0.5ex 处。

`/tikz/data visualization/legend entry options/label in legend line coordinates={<list of coordinates>}`

<list of coordinates> 是个点坐标列表, 其中的点坐标使用 TikZ 的格式, 如 (0,1)。在这里, 程序会开启一个 TikZ 坐标系, 在其中绘制作为图示标签的曲线。条目显像器会画出经过这些点的曲线, 将其作为图示标签的曲线部分 (图示标签还可能包含散点, 需要用下面的选项画出), 曲线的样式与显像器画出的数据点曲线的样式相同。例如, 如果显像器使用选项 `smooth line` 和 `style=red`, 则数据点曲线和图示标签曲线就都是红色、光滑的。

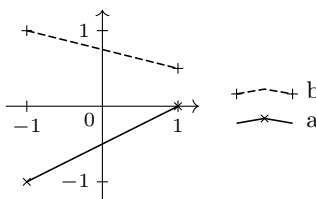
由于绘制图示标签的坐标系的原点 (0,0) 位于文字标签的左侧, 并在文字基线之上 0.5ex 处, 所以 <list of coordinates> 中的坐标的横坐标分量应当使用非正数, 否则图示标签曲线可能会与文字标签重叠。当使用选项 `text left` 时, 文字标签位于图示标签的左侧, 但此时绘制图示标签曲线的坐标系也会被翻过来 (即受到一个反射变换), 所以 <list of coordinates> 中的坐标的横坐标分量仍然应当使用非正数。



```
\tikz \datavisualization [
  school book axes, visualize as line/.list={a,b},
  legend={up then right}, style sheet=vary dashing,
  a={label in legend={text=a, label in legend line coordinates={
    (-2em,-.2ex), (-1em,.2ex), (0,-.2ex)}}},
  b={label in legend={text=b, label in legend line coordinates={
    (-2em,-.2ex), (-1em,.2ex), (0,-.2ex)}}}]
data point [x=-1, y=-1, set=a] data point [x=1, y=0, set=a]
data point [x=-1, y=1, set=b] data point [x=1, y=0.5, set=b];
```

`/tikz/data visualization/legend entry options/label in legend mark coordinates=<list of coordinates>`

与上面的选项 `label in legend line coordinates` 类似，只是用于画出图示标签的散点部分，也就是说，这两个选项可以同时使在一个条目中。

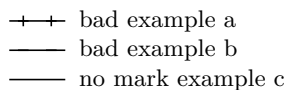


```
\tikz \datavisualization [
  school book axes, visualize as line/.list={a,b}, legend={up then right},
  style sheet=vary dashing, style sheet=cross marks,
  a={label in legend={text=a, label in legend line coordinates={(-2em,-.2ex), (-1em
    ,.2ex), (0,-.2ex)},
    label in legend mark coordinates={(-1em,.2ex)}}},
  b={label in legend={text=b, label in legend line coordinates={(-2em,-.2ex), (-1em
    ,.2ex), (0,-.2ex)},
    label in legend mark coordinates={(-2em,-.2ex), (0,-.2ex)}}}]
data point [x=-1, y=-1, set=a] data point [x=1, y=0, set=a]
data point [x=-1, y=1, set=b] data point [x=1, y=0.5, set=b];
```

上面两个选项都需要手工指定一个 `<list of coordinates>` 来绘制图示标签，手工指定坐标列表有时会比较麻烦，例如，如果显像器使用选项 `gap cycle`，那么图示标签曲线当然也是“`gap cycle`”形式的，为了使得整个条目足够美观并且与其它条目大小相称，在手工指定一个坐标列表时就得仔细考虑一番，选择合适的坐标。如果想省去手工指定坐标列表的麻烦，可以使用下面的选项。

/tikz/data visualization/legend entry options/straight label in legend line

这个选项将图示标签曲线指定为直线段，并且直线段上还有两个散点标记符号，不过只有在指定点标记符号时（用选项 `style={mark=<mark specification>}`），直线段上才会出现这两个点标记。



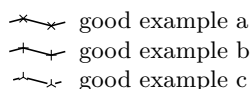
 + + bad example a
 — bad example b
 — no mark example c

```
\tikz \datavisualization [visualize as line/.list={a,b,c},
  legend entry options/default label in legend path/.style=
    straight label in legend line,
  a={style={mark=+}, label in legend={text=bad example a}},
  b={style={mark=-}, label in legend={text=bad example b}},
  c={label in legend={text=no mark example c}}];
```

上面这个例子表明，有的点标记符号在直线段上是辨认不出来的。

/tikz/data visualization/legend entry options/zig zag label in legend line

这个选项将图示标签曲线指定为之字形曲线，这是默认的形式。

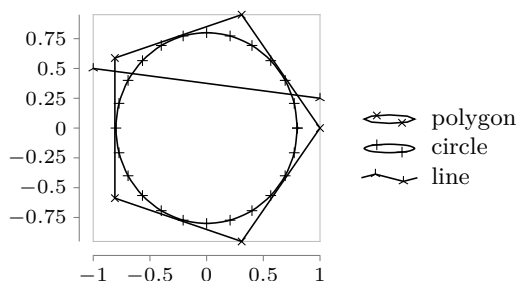


 x x good example a
 + + good example b
 - - good example c

```
\tikz \datavisualization [
  visualize as line/.list={a,b},
  visualize as smooth line=c,
  style sheet=cross marks,
  a={label in legend={text=good example a}},
  b={label in legend={text=good example b}},
  c={gap line, label in legend={text=good example c}}];
```

/tikz/data visualization/legend entry options/circular label in legend line

这个选项将图示标签曲线指定为环形，当显像器使用选项 `polygon`, `smooth cycle`, `straight cycle` 时，可以使用这个选项，该选项是专为这个情况定制的。



```
\tikz \datavisualization [
  scientific axes={clean}, all axes={length=3cm},
  visualize as line/.list={a,b,c}, a={polygon}, b={smooth cycle},
  style sheet=cross marks,
  a={label in legend={text=polygon}},
  b={label in legend={text=circle}},
```

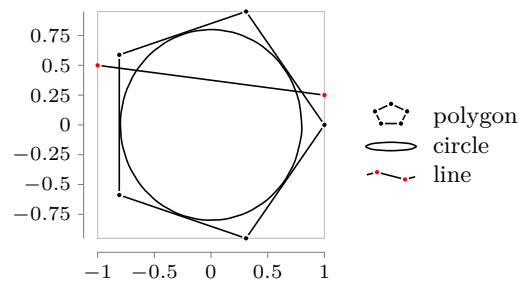
```

c={label in legend={text=line}}}
data [format=function, set=a] {
  var t : {0,72,...,359};
  func x = cos(\value t);
  func y = sin(\value t);
}
data [format=function, set=b] {
  var t : [0:2*pi];
  func x = .8*cos(\value t r);
  func y = .8*sin(\value t r);
}
data point [x=-1, y=0.5, set=c]
data point [x=1, y=0.25, set=c];

```

/tikz/data visualization/legend entry options/gap circular label in legend line

这个选项将图示标签曲线指定为带空隙的环形，当显像器使用选项 `gap cycle`, `gap line` 时，可以使用这个选项，该选项是专为这个情况定制的。



```

\tikz \datavisualization [
  scientific axes={clean}, all axes={length=3cm},
  visualize as line/.list={a,b,c}, a={gap cycle}, b={smooth cycle}, c={gap line},
  a={style={mark=*, mark size=0.5pt}, label in legend={text=polygon}},
  b={label in legend={text=circle}},
  c={style={mark=*, mark size=0.5pt, mark options=red}, label in legend={text=line
    }}]
data [format=function, set=a] {
  var t : {0,72,...,359};
  func x = cos(\value t);
  func y = sin(\value t);
}
data [format=function, set=b] {
  var t : [0:352];
  func x = .8*cos(\value t);

```

```

func y = .8*sin(\value t);
}
data point [x=-1, y=0.5, set=c]
data point [x=1, y=0.25, set=c];

```

/tikz/data visualization/legend entry options/label in legend one mark

这个选项适用于显像器绘制散点图，或者显像器使用 `no lines` 选项的情况，该选项只把一个点标记符号作为图示标签。

```

x example a
+ example b
^ example c

```

```

\tikz \datavisualization [visualize as scatter/.list={a,b,c},
style sheet=cross marks,
a={label in legend={text=example a}},
b={label in legend={text=example b}},
c={label in legend={text=example c}}];

```

/tikz/data visualization/legend entry options/label in legend three marks

这个选项适用于显像器绘制散点图，或者显像器使用 `no lines` 选项的情况，该选项把 3 个点标记符号作为图示标签。

```

x x x example a
+ + + example b
^ ^ ^ example c

```

```

\tikz \datavisualization [visualize as scatter/.list={a,b,c},
style sheet=cross marks,
a={label in legend={text=example a,
label in legend three marks}},
b={label in legend={text=example b,
label in legend three marks}},
c={label in legend={text=example c,
label in legend three marks}}];

```

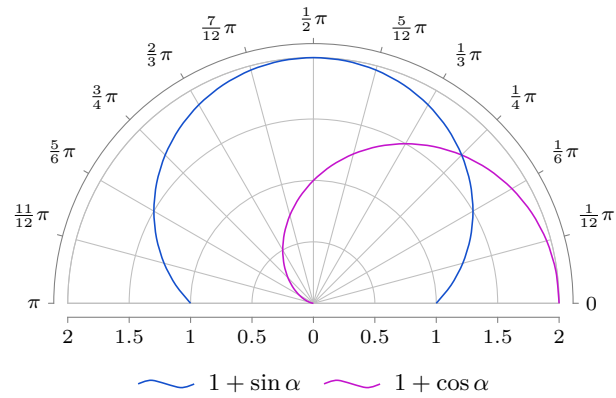
80 极坐标系

80.1 Overview

首先调用程序库 `datavisualization.polar`.

在极坐标系统中，数据点的一个数据标识符对应角度，另一个对应半径。极坐标系统的角度轴可以使用角度制、弧度制，可以做对数化处理，或者在两个角度区间之间转换（映射）。

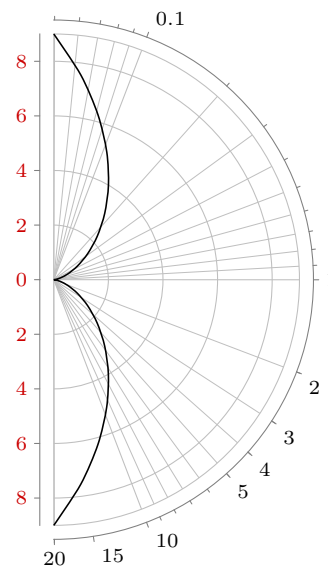
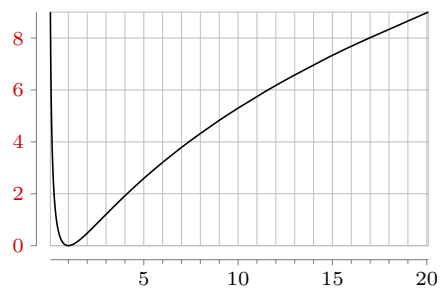
在可视化命令中使用选项 `scientific polar axes` 就可以开启极坐标系。这个选项可以带有一些值（选项）来对极坐标系做某些设置。极坐标系统中的“轴”尽管比笛卡尔坐标系统中的“轴”抽象一些，但在可视化系统中二者都是通常的轴，在很多方面的处理上都是类似地，能用于笛卡尔坐标轴的那些选项也能用于极坐标轴。例如关于轴标签、刻度线、刻度值标签、网格线、轴的长度、`style` 等的选项。当然极坐标系有自己的特殊选项。



```

\tikz \datavisualization [scientific polar axes={0 to pi, clean}, all axes=grid,
    style sheet=vary hue, legend=below]
    [visualize as smooth line=sin,
    sin={label in legend={text=$1+\sin \alpha$}}]
data [format=function] {
    var angle : interval [0:pi];
    func radius = sin(\value{angle}r) + 1;
}
    [visualize as smooth line=cos,
    cos={label in legend={text=$1+\cos \alpha$}}]
data [format=function] {
    var angle : interval [0:pi];
    func radius = cos(\value{angle}r) + 1;
};

```



```

\tikz[baseline] \datavisualization [
  scientific axes={clean},
  x axis={attribute=angle, ticks={minor steps between steps=4}},
  y axis={attribute=radius, ticks={some, style=red!80!black}},
  all axes=grid,
  visualize as smooth line=sin]
data [format=function] {
  var t : interval [-3:3];
  func angle = exp(\value t);
  func radius = \value{t}*\value{t};
};
\quad
\tikz[baseline] \datavisualization [
  scientific polar axes={right half clockwise, clean},
  angle axis={logarithmic, ticks={minor steps between steps=8,
    major also at/.list={2,3,4,5,15,20}}},
  radius axis={ticks={some, style=red!80!black}},
  all axes=grid,
  visualize as smooth line=sin]
data [format=function] {
  var t : interval [-3:3];
  func angle = exp(\value t);
  func radius = \value{t}*\value{t};
};

```

80.2 科学极坐标系统

`/tikz/data visualization/scientific polar axes=<options>`

这个选项开启极坐标系统，其中两个轴的名称分别默认为 `angle axis`, `radius axis`. `<options>` 中的选项会被冠以前缀

```
/tikz/data visualization/scientific polar axes
```

来执行。

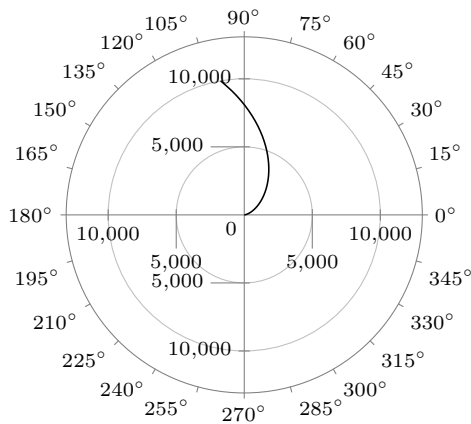
与笛卡尔坐标系统一样，要把轴于数据点的数据标识符（变量名）对应起来，使用选项 `attribute`（见 §77.2.2）。

对于半径轴 `radius axis`，对数化选项 `logarithmic`（见 §77.2.5）无效，因为任何极径都包含极点，在对半径轴的实际长度做变换时都会涉及 0，而对数函数在 0 处无定义。除此之外，半径轴都与笛卡尔坐标系统中的轴类似。在指定半径轴的实际长度时，可用 §77.2.6 中的选项，例如

```
radius axis={scaling=0 at 0 and 999 at 2}
```



```
radius axis={length=2cm}
```



```
\tikz \datavisualization [scientific polar axes,
  radius axis={attribute=distance,
    ticks={step=5000, stack}, padding=1.5em,
    length=1.8cm, grid},
  visualize as smooth line]
data [format=function] {
  var angle : interval [0:100];
  func distance = \value{angle}*\value{angle};
};
```

80.2.1 角度轴的刻度线

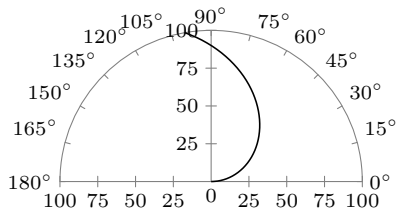
角度轴的刻度值标签处于“外侧”，刻度线有“向内”、“向外”两种选择。

```
/tikz/data visualization/scientific polar axes/outer ticks
```

这个选项使得角度轴的刻度线指向“外侧”，这是默认的。

```
/tikz/data visualization/scientific polar axes/inner ticks
```

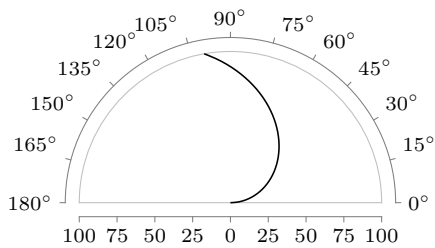
这个选项使得角度轴的刻度线指向“内侧”。



```
\tikz \datavisualization [
  scientific polar axes={inner ticks, 0 to 180},
  radius axis={length=2cm},
  visualize as smooth line]
data [format=function] {
  var angle : interval [0:100];
  func radius = \value{angle};
};
```

```
/tikz/data visualization/scientific polar axes/clean
```

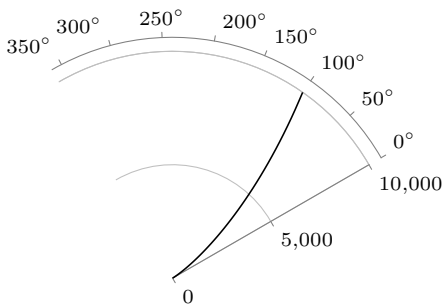
这个选项使得可视化的半径轴线段和可视化的角度轴圆弧脱离绘图区域，向外偏移一段距离，并且没有竖直的纵轴。这样可视化的轴就不会干扰读图。



```
\tikz \datavisualization [
  scientific polar axes={clean, 0 to 180},
  radius axis={length=2cm},
  visualize as smooth line]
data [format=function] {
  var angle : interval [0:100];
  func radius = \value{angle};
};
```

80.2.2 角度轴的角度范围

当说到角度轴的角度范围时，可能有两个意思：角度轴的刻度值标签所标示的角度范围，或者角度轴在页面上所占用的角度范围。对于第 2 个意思，可在角度轴的选项中使用 `scaling=<a1> at <A1> and <a2> at <A2>`（见 §77.2.6）来设置。



```
\tikz \datavisualization [
  scientific polar axes=clean,
  radius axis={attribute=distance,
    ticks={step=5000}, length=3cm, grid},
  angle axis={scaling=0 at 30 and 360 at 120,
    ticks={about=6}},
  visualize as smooth line]
data [format=function] {
  var angle : interval [0:100];
  func distance = \value{angle}*\value{angle};
};
```

对于第 1 个意思，系统有多个预定义的选项，这些选项有 3 种类型

- 所规定的范围都是 90° 的整数倍。
- 所规定的范围都是 $\frac{\pi}{2}$ 的整数倍。
- 以角度轴处理的各数据点的最小角度为水平或竖直方向，并且各数据点的最大角度也是水平或竖直方向。

将这些预定义的选项简单列举如下，具体效果参考手册：

```
0 to 90
-90 to 0
0 to 180
-90 to 90
0 to 360
-180 to 180
0 to pi half 范围同 0 to 90，但角度值标签使用弧度并带有  $\pi$ 
```

```

-pi half to 0
0 to pi
-pi half to pi half
0 to 2pi
-pi to pi
quadrant
quadrant clockwise
fourth quadrant
fourth quadrant clockwise
upper half
upper half clockwise
lower half
lower half clockwise
left half
left half clockwise
right half
right half clockwise

```

80.3 创建新的极坐标系统

```
/tikz/data visualization/new polar axes={<angle axis name>}{<radius axis name>}
```

这个选项开启一个新的极坐标系统，并且分别为角度轴和半径轴命名。

```
/tikz/data visualization/axis options/unit vectors={<unit vector 0 degrees>}{<unit
vector 90 degrees>} (无默认值，初始值 {(1pt,0pt)}{(0pt,1pt)})
```

该选项作为角度轴的选项。在初始之下，极坐标系统将 $(1pt, 0pt)$ 作为角度度量的始边方向，即 0 度方向；将 $(0pt, 1pt)$ 作为 90 度方向。用这个选项可以修改这种初始行为，其中 $\langle \text{unit vector 0 degrees} \rangle$ 和 $\langle \text{unit vector 90 degrees} \rangle$ 都是 TikZ 的坐标形式。



```

\tikz \datavisualization [
  new polar axes={angle axis}{radius axis},
  radius axis={unit length=1cm},
  angle axis={unit vectors={({10:1pt}){(60:1pt)}}},
  visualize as scatter]
data [format=named] {
  angle={0,90}, radius={0.25,0.5,...,2}
};

```

```
/tikz/data visualization/axis options/degrees
```

该选项作为角度轴的选项。这个选项决定角度的度量和计算采用角度制。

```
\tikz \datavisualization [
  new polar axes={angle axis}{radius axis},
  radius axis={unit length=1cm},
  angle axis={degrees},
  visualize as scatter]
data [format=named] {
  angle={10,90}, radius={0.25,0.5,...,2}
};
```

/tikz/data visualization/axis options/radians

该选项作为角度轴的选项。这个选项决定角度的度和计算采用弧度制。

```
\tikz \datavisualization [
  new polar axes={angle axis}{radius axis},
  radius axis={unit length=1cm},
  angle axis={radians},
  visualize as scatter]
data [format=named] {
  angle={0,1.5}, radius={0.25,0.5,...,2}
};
```

81 The Data Visualization Backend

82 Key Management

本节讲解 `pgfkeys` 宏包，本宏包可以独立于 PGF 来使用，PGF 和 TikZ 会自动加载该宏包。

82.1 简介

82.1.1 与其它类似宏包的比较

`pgfkeys` 宏包定义了一种键值管理系统，在某些方面与 `keyval` 宏包或其改进版 `xkeyval` 宏包类似。`pgfkeys` 宏包与 `xkeyva` 宏包的主要区别在于：

- `pgfkeys` 宏包将键（key）组织为“树”，而 `xkeyva` 宏包将键组织为“族”（family）。
- `pgfkeys` 宏包没有保存堆栈影响。
- `pgfkeys` 宏包比 `xkeyval` 宏包稍慢，但也慢不了很多。
- `pgfkeys` 宏包支持“样式”（style，即用手柄 `/.style` 定义的键），也就是说，可以把多个键的作用可以保存在一个键中，使用这一个键就可以实现多个键的作用。
- `pgfkeys` 宏包可以定义有多个变量的键。
- `pgfkeys` 宏包支持“手柄”（handler）。

82.1.2 快速引导

pgfkeys 宏包将键 (key) 组织为“树”，从树的根节点到某个键所包含的各个层次的键叫作一个“键路径”，一个路径包含键名称和分隔键名称的斜线，一个完整的键路径以斜线开头，例如

```
/tikz/cs/x
```

这是坐标系中代表 x 分量的键的路径，有 3 个层次。键路径中，一个键名称之前的所有符号叫作该键的“前缀路径”，例如上面的键 x 的前缀就是“/tikz/cs/”。当某个键正在被处理时，该键的名称保存在宏 `\pgfkeyscurrentname` 中，该键的前缀路径保存在宏 `\pgfkeyscurrentpath` 中。

可以用命令 `\pgfkeys`，`\pgfkeyssetvalue`，`\pgfkeysdef` 等自定义一个键。

命令 `\pgfkeys` 可以用于定义键，也可以用于执行键。用命令 `\pgfkeys` 定义键时常用到“手柄”，见下面的例子。在执行键时，通常使用“键值对”(key-value pairs) 的形式，这是一种赋值形式，如，`<key>=<value>` 或者只写出 `<key>`，如果将这种形式作为命令 `\pgfkeys` 的参数，就表示执行键（而不是定义键），例如

<pre>The value is ' hi' .</pre>	<pre>\pgfkeys{/my key/.code=The value is ' #1' .} \pgfkeys{/my key=hi!}</pre>
---------------------------------	---

这个例子的第 1 行定义键 `/my key`，用手柄 `/.code` 将其定义为含有一个变量的可执行键。第 2 行给这个键赋值并执行，得到一个句子。

再如定义包含两个变量的键：

<pre>The values are ' a1' and ' a2' .</pre>	<pre>\pgfkeys{/my key/.code 2 args=The values are \par ' #1' and ' #2' .} \pgfkeys{/my key={a1}{a2}}</pre>
---	--

下面例子定义一个包含变量的键，并用手柄 `/.default` 指定这个键的默认值：

<pre>hello, world!</pre>	<pre>\pgfkeys{/my key/.code=#1} \pgfkeys{/my key/.default=hello} \pgfkeys{/my key, /my key={,\ world!}}</pre>
--------------------------	---

在使用键时，一般应写出完整的键路径，但是如果某个键有默认路径，就可以只写出键名称，省略路径前缀，在执行键时，默认的前缀路径会被自动添加。Tikz 中的键多数都以 `tikz` 为默认路径前缀，例如

```
/tikz/line width
```

在 `{tikzpicture}` 环境中绘图时，写出

```
\draw[/tikz/line width=10mm]...
```

```
\draw[line width=10mm]...
```

都是可以的。

可以用手柄 `/.cd` 来设置键的默认路径，句法格式是：

```
\pgfkeys{<前缀>/cd, <key1>=<value>, <key2>=<value>, ...}
```

这个句式中的键 `<key1>`，`<key2>` 等都会有相同的前缀。

用手柄 `/.style` 定义的键叫作“样式”，样式中可以使用自定义的键，即把键作为其它键的参数，也就是说此时可以把键套嵌使用：

(a:foo)(b:bar)(a:wow)

```
\pgfkeys{/a/.code=(a:#1)}
\pgfkeys{/b/.code=(b:#1)}
\pgfkeys{/my style/.style={/a=foo,/b=bar,/a=#1}}
\pgfkeys{/my style=wow}
```

执行命令 `\pgfkeys` 时，命令中的各个键应当以斜线 “/” 开头，否则，命令会给键加上斜线，例如上面的例子可以修改为：

(a:foo)(b:bar)(a:wow)

```
\pgfkeys{a/.code=(a:#1)}
\pgfkeys{b/.code=(b:#1)}
\pgfkeys{my style/.style={a=foo,b=bar,a=#1}}
\pgfkeys{my style=wow}
```

还需要注意的是，命令 `\pgfkeys` 定义的键的有效范围受到 $\text{T}_\text{E}_\text{X}$ 分组的限制，超出所在分组无效。 $\text{T}_\text{E}_\text{X}$ 分组是花括号、环境等限定的范围。

82.2 The Key Tree

前面提到可以给键定义默认路径，但是默认路径不同于搜索路径，`pgfkeys` 并不执行针对键路径的搜索。当一个键有默认路径时，可以写出该键的完整路径，也可以只写键名称，例如以下 2 个形式都可以接受：

```
/pgf/arrow keys/fill=red
fill=red
```



```
\tikz \draw[-{Stealth[fill=red,scale=5}}](0,0);
\tikz \draw[-{Stealth[/pgf/arrow keys/fill=red,scale=5}}](0,0);
```

一个键中可以储存某些记号，记号可以是字符、其它的键、命令、宏等。可以用命令 `pgfkeys` 或 `\tikzset` 向键中存储记号。下面的命令也可以向键中可以储存某些记号或者读取键中所存储的记号。

```
\pgfkeyssetvalue{<full key>}{<token text>}
```

定义键 `<full key>`，并将 `<token text>` 保存到 `<full key>` 中，`<full key>` 是完整的键路径。`<token text>` 可以是字符， $\text{T}_\text{E}_\text{X}$ 的 `if` 语句，或者其它命令（如绘图命令），其中可以含有变量，但变量不能被赋值。当执行 `<full key>` 后，输出 `<token text>` 的结果。可以用命令 `\pgfkeys` 或 `\pgfkeysvalueof` 执行 `<full key>`。



```
\pgfkeyssetvalue{/fw}{\tikz\draw circle(10pt);}
\pgfkeys{/fw}
```

此命令定义的键的有效范围受到 $\text{T}_\text{E}_\text{X}$ 分组的限制。

```
\pgfkeyslet{<full key>}{<macro>}
```

定义键 `<full key>`，并执行 `\let` 语句，使得 `<full key>` 指向宏 `<macro>`，也就是说，执行 `<full key>` 就相当于执行宏 `<macro>`。注意 `<macro>` 中不能有变量。如果 `<macro>` 是 `\relax`，则执行 `<full key>` 不会出错但没有任何结果。

`\pgfkeysgetvalue{<full key>}{<macro>}`

定义宏 `<macro>`，并将 `<full key>` 中存储的记号赋予宏 `<macro>`，如果宏 `<macro>` 不存在就创建它，因此执行宏 `<macro>` 就相当于执行键 `<full key>`。如果 `<full key>` 没有定义，则 `<macro>` 等于 `\relax`

`\pgfkeysvalueof{<full key>}`

执行 `<full key>`，将储存在其中的记号插入到当前位置。

`\pgfkeysifdefined{<full key>}{<if>}{<else>}`

检查 `<full key>` 是否已经用 `\pgfkeyssetvalue` 或 `\pgfkeyslet` 定义，如果已经定义，执行 `<if>`，否则执行 `<else>`。

false

```
\pgfkeyssetvalue{/ifstat}{\ifnum 1>10 true \else false \fi}
\pgfkeysvalueof{/ifstat}
```

Hello, world!

```
\def\helloworld{Hello, world!}
\pgfkeyslet{/my family/my key}{\helloworld}
\pgfkeysvalueof{/my family/my key}
```

Hello, world!

```
\pgfkeyssetvalue{/my family/my key}{Hello, world!}
\pgfkeysgetvalue{/my family/my key}{\helloworld}
\helloworld
```

82.3 Setting Keys

命令 `\pgfkeys` 执行键时处理的是键值对，键值对是 `<key>=<value>` 或 `<key>` 这种形式。`\pgfkeys` 处理键值对时会执行以下操作：

1. 某个命令以 `<value>` 为参数的命令会被执行，该命令被储存在 `<key>` 的一个特殊的子键（subkey）中。
2. 将 `<value>` 储存入 `<key>` 中。
3. 如果 `<key>` 的名称（最后一个斜线之后的符号）是手柄（handler），则按该类手柄的规则处理。
4. 如果 `<key>` 是未知的，则调用 unknown key handlers 来给出错误提示。

另外，如果只有键名称 `<key>` 而没有 `<value>`，则程序可能会尝试使用 `<key>` 的默认值。

下面是一些用于执行键的命令，其作用范围受到 $\text{T}_\text{E}_\text{X}$ 分组的限制。

`\pgfkeys{<key list>}`

`<key list>` 是用逗号分隔的键值对列表，键值对可以是 `<key>=<value>` 或 `<key>` 的形式。在 `<key>` 和 `<value>` 两侧的空格会被省略，也可以在 `<key>` 或 `<value>` 的两侧使用花括号。如果 `<value>` 中含有逗号、等号则必须用花括号将它括起来。键值对会按照它们出现的次序被依次处理。

在命令中，如果 `<key>=<value>` 中的 `<key>` 只是键名称，那么程序会把斜线“/”加到键名称之前。

在 `<value>` 中可以使用命令 `\pgfkeys`，即在执行 `<key>` 时，命令 `\pgfkeys` 可以套嵌使用，例如

```
(a:(b:1))
\pgfkeys{/a/.code=(a:#1)}
\pgfkeys{/b/.code=(b:#1)}
\pgfkeys{/a={\pgfkeys{b=1}}}
```

但是在定义 `<key>` 时命令 `\pgfkeys` 不能套嵌使用，例如

```
(a:)
\pgfkeys{/a/.code=(a:{\pgfkeys{/b/.code=(b:#1)}})}
\pgfkeys{/a=1}
```

`<key>=<value>` 中的 `<value>` 可以是字符，在允许的情况下也可以是命令，例如绘图命令。

```
○
\pgfkeys{/fw/.code={#1}}
\pgfkeys{/fw={\tikz\draw circle(10pt);}}
```

`\pgfqkeys{<default path>}{<key list>}`

等效于 `\pgfkeys{<default path>/.cd,<key list>}`，只是运行速度稍微快些。这个命令不能用于“用户代码”（user code）中，而可以用在 `\tikzset` 或 `\pgfset` 中。

`\pgfkeysalso{<key list>}`

`\pgfqkeysalso{<default path>}{<key list>}`

82.3.1 首字符句法检测

命令 `\pgfkeys` 执行 `key` 时，其参数是键值对，键值对之间用逗号分隔。键值对可以是以某特殊字符开头的字符串（即只有键名称，没有“`=<value>`”这部分），这个特殊字符引起“首字符句法检测”功能。这个功能的意思是，将某个字符与某个命令联系起来，如果一个参数以这个特殊字符开头，那么该命令就会处理这个参数，处理结果由命令 `\pgfkeys` 显示出来。

使用首字符句法检测功能需要 3 个步骤：

1. 开启首字符句法检测功能，使用下一选项，

`/handlers/first char syntax=<true or false>` （默认 `true`，初始值 `false`）

这个键（选项）决定是否开启首字符句法检测功能，其初始值为 `false`，默认为 `true`。

2. 选定要使用的特殊字符，用下一选项

`/handlers/first char syntax/<meaning of character>` （无默认值）

这里的 `<meaning of character>` 必须是“the character `<字符>`”这种形式，例如，将小于号作为特殊首字符，那么 `<meaning of character>` 就必须是 `the character <`，其原因在于下一步骤。

3. 可以用命令 `\pgfkeyssetValue` 或者手柄 `/.initial`，将某个宏与指定的字符联系起来，这个宏可以是已有的，也可以是稍后再定义的。

`/handlers/first char syntax/the character <字符>/.initial=<macro>`

这个选项使用了 $\text{T}_{\text{E}}\text{X}$ 的 `\let` 和 `\meaning` 命令。命令 `\let\<macro>=标识符` 的作用是将“标识符”的当前功能赋予宏 `\<macro>`，例如，将小于号的作用赋予宏 `\mycharacter`，


```
\let\mycharacter=<
```

命令 `\meaning<macro>` 的作用是显示 `\<macro>` 的定义,

the character <

```
\let\mycharacter=<
\meaning\mycharacter
```


命令 `\meaning<macro>` 的结果就是 the character < 字符 > 这种形式。

下面是使用首字符句法检测功能例子。

```
Quoted: "foo". Pointed: <bar>.
```

```
\pgfkeys{
  /handlers/first char syntax=true,
  /handlers/first char syntax/the character "/.initial=\myquotemacro,
  /handlers/first char syntax/the character </.initial=\mypointedmacro,
}
\def\myquotemacro#1{Quoted: #1. }
\def\mypointedmacro#1{Pointed: #1. }
\ttfamily \pgfkeys{"foo", <bar>}
```

下面的例子中, 单引号起到了给 node 加标签的作用:

' foo'



```
\pgfkeys{
  /handlers/first char syntax=true,
  /handlers/first char syntax/the character ' /.initial=
  \mysinglequotemacro
}
\def\mysinglequotemacro#1{\pgfkeysalso{label={#1}}}
\tikz \node [circle, ' foo' , draw] {bar};
```

注意对于上面的例子来说, 如果直接把命令 `\pgfkeysalso` 用作 `\node` 的选项:

```
\tikz \node [circle, \pgfkeysalso{label={' foo' }}, draw] {bar};
```

会导致错误: ! TeX capacity exceeded, sorry [input stack size=5000].

下面的例子中, 用左圆括号引起一个 if 判断:



```
\pgfkeys{
  /handlers/first char syntax=true,
  /handlers/first char syntax/the character (/ .initial=\myparamacro
}
\def\myparamacro#1{\myparaparser#1\someendtext}
\def\myparaparser(#1)#2\someendtext{
```

```

\pgfmathparse{#1}
\ifx\pgfmathresult\onetext
  \pgfkeysalso{#2}
\fi
}
\def\onetext{1}
\foreach \i in {1,...,4}
\tikz \node [draw, thick, rectangle, (pi>\i) circle, (pi>\i*2) draw=red] {x};

```

82.3.2 默认参数值

命令 `\pgfkeys` 执行 `key` 时，其参数是键值对，键值对可以是 `<key>=<value>` 或 `<key>` 的形式。如果键值对是 `<key>` 的形式，命令 `\pgfkeys` 会尝试为该键提供“默认值”，如果该键有默认值，则按 `<key>=<default value>` 来执行。手柄 `/.default` 用于定义键的默认值 (§82.4.2)。

当命令 `\pgfkeys` 遇到 `<key>` 形式的参数（即只有键名称）时，执行以下操作：

1. 输入的参数被替换为 `<key>=\pgfkeysnovalue`。
2. 如果 `<key>=\pgfkeysnovalue`，则检查子键（subkey）`<key>/.@def` 是否存在。
3. 如果子键 `<key>/.@def` 存在，就把子键中储存的记号作为 `<key>` 的 `<value>`。
4. 如果子键 `<key>/.@def` 不存在，则把 `\pgfkeysnovalue` 作为 `<key>` 的 `<value>`。
5. 如果 `<key>` 的 `<value>` 等效于 `\pgfkeysvaluerequired`，则执行命令

```
\pgfkeys{/errors/value required=<key>{}}
```

产生一个错误信息 (§82.5)。

82.3.3 定义一个键，用键为命令的变量赋值

用键来执行某个不带变量的命令的方式如 §82.2 所述。可以定义带变量的命令，例如

```
no \def\tf#1#2{\ifnum #1>#2 yes \else no \fi}
\tf{11}{13}
```

上面的例子中定义了一个有 2 个变量的命令，执行该命令时需要同时列出两个变量值。当某个命令带有变量时，可以用键来为命令的变量赋值，然后在执行该命令的时候就不必再同时给出变量值。下面讲两种用键来为命令的变量赋值的方式。

第一种方式，包括 3 个步骤：

1. 首先要用命令 `\def` 定义一个宏，并用这个宏保存所需要执行运算的命令代码，因此在该宏名称之后要列举代码中所有的变量，变量列举之后要加上命令 `\pgfeov`，其中 `eov` 的意思是“end of value”，这个命令提示变量列举完毕。例如

```
\def\mystore#1#2\pgfeov{\def\a{#1}\def\b{#2}}
```

2. 用命令 `\pgfkeyslet` 使得键 `<key>/.@cmd` 指向宏 `<macro>`。

```
\pgfkeyslet{/my key/.\@cmd}{\mystore}
```

3. 用命令 `\pgfkeys` 运算 `<key>=<value>`，完成赋值。

```
\pgfkeys{/my key={hello}{world}}
```

在这个过程中,程序会创建宏 `\pgfkeyscurrentkey`,用以保存展开 `<key>` 后的文本。程序会检查 `<key>/.\cmd` 是否存在, 如果存在就令该键保存定义宏 `<macro>` 的代码。`<value>` 就是宏的变量值。

在用命令 `\def` 定义宏时,如果宏中有多个变量,则需要依次列举变量为 `#1#2...` 等;相应地,在 `<key>=<value>` 的 `<value>` 中也要列举变量值, 变量的列举方式与变量值的列举方式要一致。要适当使用花括号将变量值括起来, 不能只用空格分隔变量值, 否则变量值的第 1 个字符被看作是第 1 个变量值:

```
\a is hello, \b is world.
\a is y, \b is es no.
```

```
\def\mystore#1#2\pgfeov{\def\a{#1}\def\b{#2}}
\pgfkeyslet{/my key/.\cmd}{\mystore}
\pgfkeys{/my key={hello}{world}}
\string\a \ is \a, \string\b \ is \b.

\pgfkeys{/my key=yes no} % 用空格分隔变量值, 出错
\string\a \ is \a, \string\b \ is \b.
```

`pgfkeys` 宏包允许的列举方式是比较灵活的, 实际上就是 \TeX 定义宏时所允许的变量列举方式。例如下面的例子中使用单词 `and`, 加号来分隔列举项, 此时可以不用花括号把变量值括起来, 但如果两个变量值之间没有其它分隔标识, 就应当用花括号分别括起来, 不能只用空格分隔:

```
\a is and, \b is and,
\c is y, \d is es no.
```

```
\def\mystore#1 and #2 + #3#4\pgfeov{
  \def\a{#1}\def\b{#2}\def\c{#3}\def\d{#4}}
\pgfkeyslet{/my key/.\cmd}{\mystore}
\pgfkeys{/my key=and and and + yes no }
\string\a \ is \a, \string\b \ is \b, \par
\string\c \ is \c, \string\d \ is \d.
```

第二种方式, 即将上一种方式改编为以下命令。

```
\pgfkeysdef{<key>}{<code>}
```

这个命令会用 `\def` 临时定义一个宏, 这个宏至多有一个变量, 即变量列举为 `#1\pgfeov`, 这个宏的定义内容是 `<code>`。 `<code>` 中至多使用 1 个变量, 当然也可以不使用变量。然后程序会令 `<key>/.\cmd` 等效于这个宏, 再然后就可以用命令 `\pgfkeys` 运算 `<key>=<value>` 来完成赋值, 并且同时执行 `<code>`, 得到结果。

```
hello, hi!
```

```
\pgfkeysdef{/my key}{hello, hi!}
\pgfkeys{/my key}
```

```
hello, hello.
```

```
\pgfkeysdef{/my key}{#1, #1.}
\pgfkeys{/my key=hello}
```

这个命令可以套嵌使用, 即可以在 `<code>` 中使用该命令。

```
\pgfkeysdef{<key>}{<code>}
```

这个命令类似 `\pgfkeysdef`，只不过本命令使用 `\edef` 来定义宏。

`\pgfkeysdefnargs{<key>}{<argument count>}{<code>}`

个命令类似 `\pgfkeysdef`，不过本命令允许在 `<code>` 中使用多个变量，至多 9 个，变量个数要在 `<argument count>` 中声明。但不同的是，在本命令之后用命令 `\pgfkeys` 运算 `<key>=<value>`，只是完成赋值，并不执行 `<code>`，要想得到 `<code>` 的执行结果，需要另外执行 `<code>`。

`\a is hello, \b is world.`

```
\pgfkeysdefnargs{/my key}{2}{\def\a{#1}\def\b{#2}}
\pgfkeys{/my key={hello}{world}}
\string\a \ is \a, \string\b \ is \b.
```

`\pgfkeysedefnargs{<key>}{<argument count>}{<code>}`

个命令类似 `\pgfkeysdefnargs`，不过本命令使用 `\edef` 来定义宏。

`\pgfkeysdefargs{<key>}{<argument pattern>}{<code>}`

个命令类似 `\pgfkeysdefnargs`，只是 `<argument pattern>` 是列举变量，列举变量时不能在变量最后使用命令 `\pgfeov`，否则出错。如前述，允许的列举方式比较灵活，注意这里的列举方式与 `<key>=<value>` 中变量值的列举方式要一致。

`\ais hello, \bis world.`

```
\pgfkeysdefargs{/my key}{#1+#2}{\def\a{#1}\def\b{#2}}
\pgfkeys{/my key=hello+world}
\string\a is \a, \string\b is \b.
```

`\pgfkeysedefargs{<key>}{<argument pattern>}{<code>}`

82.3.4 Keys That Store Values

前面讲解了用键为命令的变量赋值的方法，在这种赋值过程中，如果命令 `\pgfkeys` 处理 `<key>=<value>` 时，当前 `<key>` 和 `<key>/.\@cmd` 都找不到，也就是说 `<key>` 没有在当前做定义（没有在当前将某些代码或内容储存在 `<key>` 中），那么命令会检查这个 `<key>` 是否在之前（非当前）就已经定义了。若已经定义了，那么命令会把先前储存在 `<key>` 中记号替换为 `<value>`，然后继续处理下一个键值对。也就是说，可以用

```
\pgfkeys{<key>=<value>}
```

多次为同一个或一组命令中的变量赋值，对于不同的赋值可以得到不同的执行结果，这在前面已有例子。

82.3.5 定义手柄键

继续之前的讲解。命令 `\pgfkeys` 处理 `<key>=<value>` 时，如果当前以及之前的 `<key>` 和 `<key>/.\@cmd` 都找不到，即 `<key>` 没有在当前或之前做定义，命令 `\pgfkeys` 会检查键 `/handlers/<name>/.\@cmd` 是否存在。如果存在，就把需要执行的命令储存在这个键中，并按 §82.3.3 的方式执行之。当定义手柄键后，键 `/handlers/<name>/.\@cmd` 会自动生成。

用下面的命令定义手柄键，其中要用到 `\pgfkeyscurrentpath`：

`\pgfkeysdef{/handlers/.<handler name>}{\pgfkeysdef{\pgfkeyscurrentpath}{<code>}}`

宏 `\pgfkeyscurrentpath` 中保存的是当前键的前缀路径。<code> 中可以含有变量，但是至多使用一个变量。定义手柄后，可以把 <code> 赋予某个键，例如

代码

```
\pgfkeysdef{/handlers/.my code}{
  \pgfkeysdef{\pgfkeyscurrentpath}{代码}}
\pgfkeys{/my key/.my code={}}
\pgfkeys{/my key}
```

大于

```
\pgfkeysdef{/handlers/.my code}{
  \pgfkeysdef{\pgfkeyscurrentpath}{#1}}
\pgfkeys{/my key/.my code={\ifnum #1>3 大于 \else 小于 \fi}}
\pgfkeys{/my key=5}
```

大于

```
\pgfkeysdef{/handlers/.my code}{
  \pgfkeysdef{\pgfkeyscurrentpath}{#1}}
\pgfkeysdefnargs{/a}{2}{\ifnum #1>#2 大于 \else 小于 \fi}
\pgfkeys{/my key/.my code={#1}}
\pgfkeys{/my key={\pgfkeys{a={5}{3}}}} % 繁琐的套嵌
```

如果使用的变量个数多于 1 个，会导致错误：

```
\pgfkeysdef{/handlers/.my code}{
  \pgfkeysdefnargs{\pgfkeyscurrentpath}{2}{#1#2}}
\pgfkeys{/my key/.my code={#1#2}}
\pgfkeys{/my key={a}{b}}
```

错误提示信息是：

```
! Illegal parameter number in definition of \pgfkeys@temp.
```

前面提到，命令 `\pgfkeys` 处理 `<key>=<value>` 时，如果当前以及之前的 `<key>` 和 `<key>/.@cmd` 都找不到，命令 `\pgfkeys` 会检查键 `/handlers/<name>/.@cmd` 是否存在，如果存在则执行相应的操作。也就是说，当定义手柄 `/.<handler name>` 后，可以直接用命令

```
\pgfkeys{<key>/.<handler name>={<code>}}
```

定义一个键 `<key>`，这个键的内容就是 `<code>`，而且 `<key>/.@cmd` 也会自动产生，上面的例子就是这样的，这是程序的初始行为。可以把这个行为修改为：尽管定义了手柄 `/.<handler name>`，但是仅当 `<key>=<value>` 或 `<key>/.@cmd` 存在时，才会执行 `<key>/.<handler name>={<code>}`。这用到下面的选项：

`/handler config=all|only existing|full or existing` （无默认值，初始值 `all`）

all 这是初始值，初始行为如前述。

only existing 仅当 `<key>=<value>` 或 `<key>/.@cmd` 存在时，才会执行 `<key>/.<handler name>={<code>}`。

也就是说，即使定义了手柄 `/.<handler name>`，也不能直接用命令

```
\pgfkeys{<key>/.<handler name>={<code>}}
```

定义键 <key>, 否则 <key> 会被当作一个未知键, 输出未知键的提示信息。设置未知键的提示信息的方法见下文。

full or existing 类似 **only existing**, 只是所针对的键是那些 (只给出键名称) 没有给出完整键路径的键。例如下面的

```
\pgfkeys{/my path/.cd, key/.style={...}}
```

其中 `key/.style` 是不完整的路径, 所以对它使用 **only existing** 规则。而下面的

```
\pgfkeys{/my path/key/.style={...}}
```

给出完整路径, 不使用 **only existing** 规则。

初始定义; 重定义; 未知的 ‘other key/.code’ .

```
\pgfkeys{/the/key/.code={初始定义;}}
```

```
% 设置未知键的提示信息
```

```
\pgfkeys{/handlers/.unknown/.code={未知的 ‘\pgfkeyscurrentname’ .}}
```

```
% 输出 /the/key 的内容
```

```
\pgfkeys{/the/key}
```

```
% 改变行为
```

```
\pgfkeys{/handler config=only existing}
```

```
% 重定义 /the/key 的内容
```

```
\pgfkeys{/the/key/.code={重定义;}}
```

```
% 输出 /the/key 的内容
```

```
\pgfkeys{/the/key}
```

```
% 用手柄定义一个新 key, 产生未知键提示
```

```
\pgfkeys{/the/other key/.code={New definition.}}
```

```
/handler config/only existing/add exception={<key handler name>}
```

针对 `/handler config=only existing` 设置例外的手柄, 初始之下, 手柄 `/.cd`, `/.try`, `/.retry`, `/.lastretry`, `/.unknown` 是例外的。

82.3.6 设置未知键的提示信息

命令 `\pgfkeys` 处理 `<key>=<value>` 时, 如果当前以及之前的 `<key>` 和 `<key>/.@cmd` 都找不到, `<key>` 也不是由某个手柄定义的, 命令会检查 `<current path>/.unknown/.@cmd` 是否存在, 若存在则执行之, 输出未知键的提示信息。前面已经有一个例子。

设置未知键的提示选项的句法如下:

```
\pgfkeys{/handlers/.unknown/.code={<提示信息>}}
```

< 提示信息 > 中可以使用命令 `\pgfkeyscurrentname` 或 `\pgfkeyscurrentkey`, 分别是当前键名称, 当前键。

82.3.7 用手柄设置键名称的搜索路径

82.4 键手柄

下面讲解预定义的手柄。

82.4.1 设置键路径的手柄

`<key>/ .cd`

例如

```
\pgfkeys{<key>/ .cd, <key1>, <key2>, ...}
```

将 `<key>` 设为 `<key1>`, `<key2>` 的默认路径。

`<key>/ .is family`

当多个键被执行时, 将 `<key>` 作为其它键的默认路径。例如

```
\pgfkeys{/tikz/.is family}
\pgfkeys{tikz,line width=1cm}
```

这样就把 `/tikz` 作为 `line width=1cm` 的默认路径。

82.4.2 设置键的默认值的手柄

`<key>/ .default=<value>`

将 `<key>` 的默认值设为 `<value>`, 例如

```
\pgfkeys{/width/.default=1cm}
```

`<key>/ .value required`

当执行 `<key>` 但这个键没有被赋值时, 这个手柄引起 `/erros/value required` 被执行, 产生一个错误提示。例如

```
? \pgfkeys{/width/.code={?}}
  \pgfkeys{/width/.value required}
  \pgfkeys{/width=1}
```

`<key>/ .value forbidden`

这个手柄禁止给 `<key>` 赋值。当以 `<key>=<value>` 的形式执行 `<key>` 时, 这个手柄引起 `/erros/value forbidden` 被执行, 产生一个错误提示。

```
\pgfkeys{/my key/.code=I do not want an argument!}
\pgfkeys{/my key/.value forbidden}
\pgfkeys{/my key} % 可执行
\pgfkeys{/my key=foo} % 出错
```

82.4.3 定义键所储存的代码

`<key>/ .code=<code>`

用命令

```
\pgfkeys{<key>/ .code=<code>}
```

将 `<code>` 保存在 `<key>` 中, 执行 `<key>` 就相当于执行 `<code>`. 在 `<code>` 中可以至多使用一个变量, 此时执行 `<key>=<value>`, 否则执行 `<key>` 的默认值, 否则将 `<key>` 的值视为 `\pgfkeysnovalue`.

在 `<code>` 中可以使用其它的命令或者键, 但不能直接使用套嵌使用命令 `\pgfkeys`.

`<key>/ .ecode=<code>`

类似 `/.code`, 只是使用命令 `\pgfkeysedef`.

`<key>/ .code 2 args=<code>`

类似 `/.code`, 只是 `<code>` 中至多可以使用 2 个变量 (可以没有变量)。如果有两个变量, 则应该执行 `<key>={<value1>}{<value2>}`, 不过第 2 个变量值 `<value2>` 是可选的, 如果不给出 `<value2>` 就默认它为空。

注意对于这个手柄定义的键, 在执行键时, 键的两个变量值要用花括号分别括起来, 否则, 例如

f 分隔 first second

```
\pgfkeys{/fs/.code 2 args={#1 分隔 #2}}
\pgfkeys{/fs=first second}
```

这样会把第 1 个字符“f”当成第 1 个变量, 把“first second”当成第 2 个变量。

`<key>/ .ecode 2 args=<code>`

类似 `/.code 2 args`, 只是使用 `\edef` 做定义。

`<key>/ .code n args={<argument count>}{<code>}`

`<argument count>` 声明 `<code>` 中使用的变量的个数。对于这个手柄, 声明几个变量就必须使用几个变量, 不能多也不能少, 至多能用 9 个变量。执行 `<key>={<value1>}{<value2>}`... 时注意用花括号把各个变量值括起来。变量之间的空格会被忽略。

`<key>/ .ecode n args={<argument count>}{<code>}`

`<key>/ .code args={<argument pattern>}{<code>}`

`<argument pattern>` 列举 `<code>` 中使用的变量, 允许的列举方式比较灵活 (参考 §82.3.3), 例如可以是 `(#1/#2)`, 相应地应该执行 `<key>=<value1>/<value2>`. 在列举形式中, 变量之间的空格不会被忽略。

`<key>/ .ecode args={<argument pattern>}{<code>}`

`<key>/ .add code={<prefix code>}{<append code>}`

当用 `<key>/ .code=<code>` 定义 `<key>` 之后, 可以用这个手柄将 `<prefix code>` 添加到 `<key>/ .@cmd` 的开头, 将 `<append code>` 添加到 `<key>/ .@cmd` 的结尾, 二者任何一个都可以空置。`<prefix code>` 和 `<append code>` 中可以带有变量, 但变量不能超出 `<code>` 中的变量, 且变量的序号与 `<code>` 中

的变量序号一致。

比较 1 与 2, <

```
\pgfkeys{/ps/.code args={#1 and #2}{
  \ifnum #1>#2 > \else < \if}}
\pgfkeys{/ps/.add code={比较 #1 与 #2, }{}}
\pgfkeys{/ps=1 and 2}
```

`<key>/.prefix code={<prefix code>}`

这是 `<key>/.add code={<prefix code>}{ }` 的简写。

`<key>/.append code={<append code>}`

这是 `<key>/.add code={ }{<append code>}` 的简写。

82.4.4 定义样式的手柄

`<key>/.style=<key list>`

`<key list>` 是一系列键值的列表，其中的键应当是已经定义的。在 `<key list>` 中至多可以使用 1 个变量，有一个变量时，应当执行 `<key>=<value>`，否则执行 `<key>` 的默认值或者 `\pgfkeysnovalue`。

red box

blue box

```
\begin{tikzpicture}[outline/.style={draw=#1,fill=#1!20}]
  \node [outline=red] {red box};
  \node [outline=blue] at (0,-1) {blue box};
\end{tikzpicture}
```

`<key>/.estyle=<key list>`

类似 `/.style`，只是用 `\edef` 来定义键。

`<key>/.style 2 args=<key list>`

类似 `/.code 2 args`。

`<key>/.estyle 2 args=<key list>`

类似 `/.ecode 2 args`。

`<key>/.style args={<argument pattern>}{<key list>}`

类似 `/.code args`。

`<key>/.estyle args={<argument pattern>}{<key list>}`

类似 `/.ecode args`。

`<key>/.style n args={<argument count>}{<key list>}`

类似 `/.code n args`。

`<key>/.add style={<prefix key list>}{<append key list>}`

类似 `/.add code`，

```
<key>/.prefix style=<prefix key list>
```

```
<key>/.append style=<append key list>
```

82.4.5 Defining Value-, Macro-, If- and Choice-Keys

```
<key>/.initial=<value>
```

设置 <key> 的初始值，之后可以用 <key>=<value> 修改键的值。这个手柄的工作机制与其它手柄有所不同，这里并不创建子键 <key>/.@cmd，使用时通常需要手柄 /.get 的配合。

例如

```
\pgfkeys{/my key/.initial=red}
% 修改 /my key 的值
\pgfkeys{/my key=blue}
```

注意上面的命令 \pgfkeys{/my key=blue} 并不输出 blue，实际上这个命令会导致 /my key=\pgfkeysnovalue，要想输出 blue，需要手柄 /.get 的配合。

```
<key>/.get=<macro>
```

这个手柄执行 \let 命令，将储存在 <key> 中的值保存到宏 <macro> 中。

blue

```
\pgfkeys{/my key/.initial=red}
\pgfkeys{/my key=blue}
\pgfkeys{/my key/.get=\mymacro}
\mymacro
```

```
<key>/.add={<prefix value>}{<append value>}
```

将 <prefix value> 加到 <key> 的值的开头，将 <append value> 加到 <key> 的值的结尾。

```
<key>/.prefix={<prefix value>}
```

将 <prefix value> 加到 <key> 的值的开头。

```
<key>/.append={<append value>}
```

将 <append value> 加到 <key> 的值的结尾。

```
<key>/.link=<another key>
```

将 \pgfkeysvalueof{<another key>} 的值保存到 <key> 中，在展开 <key> 时，展开的是 <another key> 的值。

```
<key>/.store in=<macro>
```

这个手柄的作用是：当执行 <key>=<value> 时，程序会自动执行 \def<macro>{<value>}。

Hello Gruffalo!

```
\pgfkeys{/text/.store in=\mytext}
\def\a{world}
\pgfkeys{/text=Hello \a!}
\def\a{Gruffalo}
\mytext
```

<key>/.estore in=<macro>

类似 /.store in, 只是程序会自动执行 \edef<macro>{<value>}.

<key>/.is if=<TEX-if name>

这个手柄的作用是, 当执行 <key>=<value> 时, 命令会先检查 <value> 的真值是 true 还是 false, 如果只写出 <key> 就默认 <value> 的真值是 true. 如果无法判断 <value> 的真值, 就执行 /errors/boolean expected 给出错误提示. 然后将 <value> 的真值赋予 <TEX-if name>, 从而可以把 <key>=<value> 与 \if<TEX-if name> 引起的 if 语句联系起来.

Round?

```
\newif\iftheworldisflat
\pgfkeys{/flat world/.is if=theworldisflat}
\pgfkeys{/flat world=false}
\iftheworldisflat
  Flat
\else
  Round?
\fi
```

<key>/.is choice

这个手柄的作用是, 当执行 <key>=<value> 时, 命令会自动执行 <key>/<value>, 这要求键 <key>/<value> 已经存在, 否则, 若找不到这个键, 就执行 /errors/unknown choice value 给出错误提示.

82.4.6 键值的展开, 多重键值

把 <value> 赋予 <key> 时, <value> 可能是由多个命令套嵌起来定义的, 如果希望套嵌的命令先展开再参与对 <key> 的执行, 可以用下面的手柄.

<key>/.expand once=<value>

这个手柄使用命令 \expandafter 将 <value> 的最外层命令展开, 得到 <result>, 并将 <result> 赋予 <key>. 如果 <key> 中含有手柄, 则按通常的方式调用手柄.

```
Key 1: \c
Key 2: \b
Key 3: \a
Key 4: bottom
```

```

\def\a{bottom}
\def\b{\a}
\def\c{\b}
\pgfkeys{/key1/.initial=\c}
\pgfkeys{/key2/.initial/.expand once=\c} % 将 \c 展开一次, /key2=\b
\pgfkeys{/key3/.initial/.expand twice=\c} % 将 \c 展开两次, /key3=\a
\pgfkeys{/key4/.initial/.expanded=\c} % 将 \c 完全展开, /key4=bottom
\def\a{{\ttfamily\string\a}}
\def\b{{\ttfamily\string\b}}
\def\c{{\ttfamily\string\c}}
\begin{tabular}{ll}
Key 1:& \pgfkeys{/key1} \\
Key 2:& \pgfkeys{/key2} \\
Key 3:& \pgfkeys{/key3} \\
Key 4:& \pgfkeys{/key4}
\end{tabular}

```

<key>/ .expand twice=<value>

将 <value> 展开两次，即展开外层的两个层次，等效于

<key>/ .expand once/.expand once=<value>

<key>/ .expanded=<value>

使用命令 `\edef` 将 <value> 完全展开并赋予 <key>.

<key>/ .list={<comma-separated list of values>}

其中 <comma-separated list of values> 是可以赋予 <key> 的值的列表，各个值之间用逗号分隔，注意整个列表要用花括号括起来，否则 \TeX 会把键值当作 <key>. 这个手柄会使得 <key> 依次从列表中取值并被执行，得到一系列结果。程序会用 `\foreach` 语句处理键值列表，所以键值列表中可以使用省略号。

(a)(b)(0)(1)(2)(3)(4)(5)

```

\pgfkeys{/foo/.code=#1}
\pgfkeys{/foo/.list={a,b,0,1,...,5}}

```

82.4.7 键值和执行结果的转送

<key>/ .forward to=<another key>

这个手柄的作用是，当执行 `<key>=<value>` 时，同时会执行 `<another key>=<value>`，即这两个键按共同的值 <value> 来执行。如果 <key> 在先前已经定义，程序还会把这两个键的执行结果并列起来。如果 <key> 尚未定义，就直接在当前定义 <key>，并把 <value> 赋予 <another key>，并执行 `<another key>=<value>`。这里 <another key> 必须是一个完整的路径。

(a:), (b)(a:),

```
(c:1)(a:1),
(d:5:(d:6:
```

```
\pgfkeys{
  /a/.code=(a:#1),
  /b/.code=(b),
  /c/.code=(c:#1),
  /b/.forward to=/a,
  /c/.forward to=/a,
  /d/.code n args={2}{(d:#1:#2)},
  /e/.forward to=/d,
  /f/.forward to=/d
}
\pgfkeys{/a}, \pgfkeys{/b}, \
\pgfkeys{/c=1}, \
\pgfkeys{/e=5, /f=6}
```

从上面的例子看出，使用这个手柄时，其中键的定义代码里不宜有太多变量。

```
<key>/.search also={<path list>}
```

82.4.8 测试键的手柄

82.4.9 解释键的手柄

```
<key>/.show value
```

这个手柄使用命令 `\show` 展示 `<key>` 的值。

```
\pgfkeys{/my/obscure key/.show value}
```

```
<key>/.show code
```

这个手柄使用命令 `\show` 展示定义 `<key>` 的代码，即储存在 `<key>/.@cmd` 中的代码。

```
\pgfkeys{/my/obscure key/.show code}
```

```
/utils/exec=<code>
```

这是个预定义的键，用于直接执行 `<code>`。

```
\pgfkeys{\pgfkeys{some key=some value,/utils/exec=\show\hallo,obscure key=obscure}}
```

82.5 提示错误的键

```
/errors/value required={<offending key>}{<value>}
```

当这个键被执行时，编译会中断并产生一个错误提示，提示需要赋值的键 `<offending key>` 没有被赋值。

```
/errors/value forbidden={<offending key>}{<value>}
```

当这个键被执行时，编译会中断并产生一个错误提示，提示不需要赋值的键 `<offending key>` 被赋值。

```
/errors/boolean expected={<offending key>}{<value>}
```

当这个键针对手柄 `/.is if`，该键被执行时，编译会中断并产生一个错误提示，提示键 `<offending key>` 的值应当是逻辑值。

```
/errors/unknown choice value={<offending key>}{<value>}
```

当这个键针对手柄 `/.is choice`，该键被执行时，编译会中断并产生一个错误提示，提示键 `<offending key>` 不在可选序列内。

```
/errors/unknown key={<offending key>}{<value>}
```

当该键被执行时，编译会中断并产生一个错误提示，提示键 `<offending key>` 是未定义的。

82.6 键筛选

83 重复操作：foreach 句法

实现 `foreach` 句法的是宏包 `pgffor`，这个宏包会被 TikZ 自动加载，但 PGF 并不自动加载这个宏包，这个宏包可以独立于 PGF 使用。所以要想在 PGF 下使用 `foreach` 句法，应当先调用这个宏包。

这个宏包定义了命令 `\foreach` 和 `\breakforeach`。

```
\foreach <variables> [<options>] in <list> <commands>
```

首先注意，`\foreach` 语句各个组成部分之间不能有空行。

`<variables>` 是以反斜线开头的 TeX 命令形式，例如 `\x`，`\point`。

`<list>` 是以逗号分隔的列表，列举的条目可以是数字，宏，字符串等等。注意 `<list>` 中的空格不会被忽略，所以不要在列举的条目中随意使用空格，也不要为了增强代码的可读性而随意添加空格。

`<commands>` 通常是用花括号括起来的一组命令。

`foreach` 句法会将 `<list>` 中的条目，作为值依次赋予 `<variables>` 中的变量，对每次赋值，用 `<commands>` 执行一次。每次执行 `<commands>` 时，`<commands>` 都会被放入一个 TeX 分组中。因此，如果 `<commands>` 中有受限于分组的命令，该命令的效果都会限于一次执行中，不会被累计。例如，假设 `<commands>` 中有将某个计数器加 1 的命令，在每次执行 `<commands>` 之前该计数器的值都是 `n`，那么每执行 `<commands>` 时该计数器的值就是 `n+1`；当各次执行结束后，该计数器的值还是 `n`。如果希望某个命令不受分组限制，可以使用 `\global`。

```
[1][2][3][0]
```

```
\def\mylist{1,2,3,0}
\foreach \x in \mylist {[ \x]}
```

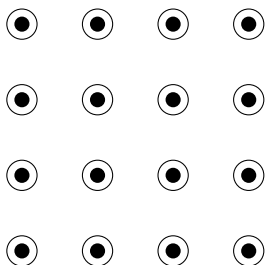
83.1 `<commands>` 的句法

通常 `<commands>` 是用花括号括起来的一组命令。如果不用花括号将命令括起来，那么 `\foreach` 会检测分号，并以检测到的第一个分号为标志来结束 `<commands>`。



```
\tikz \foreach \x in {0,1,2,3}
\draw (\x,0) circle (0.2cm);
```

另外还有 `\foreach` 语句套嵌的情况。当两个 `\foreach` 语句直接套嵌时, 内层的 `\foreach` 语句作为外层 `\foreach` 的 `<commands>`, 可以不处于花括号内。



```
\begin{tikzpicture}
\foreach \x in {0,1,2,3}
\foreach \y in {0,1,2,3}
{
\draw (\x,\y) circle (0.2cm);
\fill (\x,\y) circle (0.1cm);
}
\end{tikzpicture}
```

当两个 `\foreach` 语句套嵌时, 两个语句中的变量是相互独立的。所以在上面的例子中, `(\x,\y)` 代表 16 个点。

另外需要注意的是, 如果 `<commands>` 中不出现所设定的变量 `<variables>`, 那么程序就仅仅把 `<commands>` 执行数次, 执行的次数是 `<list>` 中列表项的个数。

```
0.48586, 0.48692, 0.80148, 0.60384, 0.71593, 0.60799, 0.98637, 0.26668, 0.23878, 0.78108,
```

```
\foreach \x in {1,...,10} {\pgfmathparse{rnd}\pgfmathresult, }
```

83.2 `<list>` 中的省略号

在 `<list>` 中将某个列举条目使用省略号代替会导致一种“递推”构造, 有以下几种类别:

1. 构造公差为 1 或 -1 的等差数列:

```
\foreach \x in {1,...,6} {\x, } 得到 1, 2, 3, 4, 5, 6,
\foreach \x in {9,...,3.5} {\x, } 得到 9, 8, 7, 6, 5, 4,
```

2. 用两项规定公差, 构造等差数列:

```
\foreach \x in {1,2,...,6} {\x, } 得到 1, 2, 3, 4, 5, 6,
\foreach \x in {1,2,3,...,6} {\x, } 得到 1, 2, 3, 4, 5, 6,
\foreach \x in {1,3,...,11} {\x, } 得到 1, 3, 5, 7, 9, 11,
\foreach \x in {1,3,...,10} {\x, } 得到 1, 3, 5, 7, 9, 注意不包括 10
\foreach \x in {0,0.1,...,0.5} {\x, } 得到 0, 0.1, 0.20001, 0.30002, 0.40002
```

3. 字母递推:

```
\foreach \x in {a,...,m} {\x, } 得到 a, b, c, d, e, f, g, h, i, j, k, l, m,
\foreach \x in {Z,X,...,M} {\x, } 得到 Z, X, V, T, R, P, N,
```

4. 参数递推:

```
\foreach \x in {2^1,2^...,2^7} {\x$} 得到 21, 22, 23, 24, 25, 26, 27,
\foreach \x in {0\pi,0.5\pi,...\pi,2\pi} {\x$} 得到 0π, 0.5π, 1π, 1.5π, 2π,
\foreach \x in {A_1,..._1,H_1} {\x$} 得到 A1, B1, C1, D1, E1, F1, G1, H1,
```

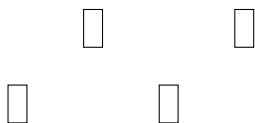
注意参数递推只能用于这种比较简单的表达式, 只针对 1 个参数使用省略号。

5. 多种混合

```
\foreach \x in {a,b,9,8,...,1,2,2.125,...,2.5} {\x,} 得到 a, b, 9, 8, 7, 6, 5, 4,
3, 2, 1, 2,
2.125, 2.25, 2.375, 2.5,
```

83.3 在 <list> 中使用花括号包裹列举条目

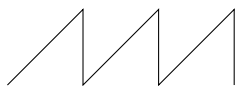
如果 <list> 中的某个列举条目中含有逗号或其他特殊符号, 应当用花括号把该列举条目括起来; 但如果该列举条目本身以开圆括号“(”开头, 以闭圆括号“)”结尾, 则可以不用花括号包裹该列举条目, 例如 TikZ 的坐标。



```
\tikz
\foreach \position in {(0,0), (1,1), (2,0), (3,1)}
\draw \position rectangle +(.25,.5);
```

83.4 在路径中使用 foreach 语句

在路径中, foreach 被作为一个算子, 其作用与命令 \foreach 相同。在路径中使用 foreach 语句时, foreach 所辖的 <commands> 必须是用于构造路径的代码, foreach 算子会参照变量值多次写出这些代码用于构成路径。



```
\tikz
\draw (0,0)
foreach \x in {1,...,3}
{ -- (\x,1) -- (\x,0) };
```

注意算子 node 和 pic 也支持 foreach 语句。

83.5 多个相互关联的变量

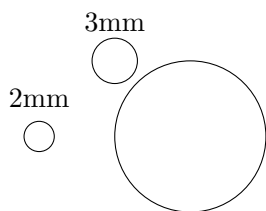
假设有 n 个状态点: $\{(x_1, y_1, z_1), \dots, (x_n, y_n, z_n)\}$, 每个状态点对应一个结果: $f((x_i, y_i, z_i)), i = 1, 2, \dots, n$. 可以用 foreach 语句得到这 n 个结果, 此时要用斜线“/”来分隔变量和变量值, 即在 <variables> 中设置数个变量并用斜线“/”分隔它们, 而在 <list> 中一个列举条目就是各变量的一个值, 构成一个状态点, 在条目中用斜线“/”分隔对应各个变量的值。在这里, 状态点是 3 维向量, 因此需要设置 3 个变量, 例如 \x/\y/\z, 这 3 个变量不是相互独立的, 也就是说, 当 \x 取某个值时, 另外两个变量不能随意取值, 它们要关联起来构成这 n 个 3 维向量。下面举例来说明。


```
\foreach \x / \y in {1/2,a/b} { "\x\ and \y" }
```

得到 “1 and 2” “a and b”，这里 \x 与 \y 构成的向量是 (1,2), (a,b)

关于斜线 “/” 需要注意以下 3 点:

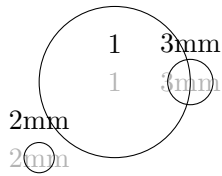
- 在 <variables> 中斜线 “/” 的前后可以有空格; 在 <list> 中, 斜线 “/” 的前后如果有空格, 则空格会被当作变量值的一部分, 有时会导致错误。
- 有的斜线 “/” 的前面或后面缺少变量值, 就当作 “空值” 处理, 在不同情况下程序会有不同反应。



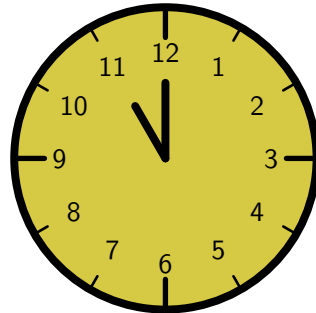
```
\begin{tikzpicture}
  \foreach \x / \diameter / \y in
    {0 / 2mm / 0 , 1 / 3mm / 1 , 2 / / 0}
  {\draw (\x,\y) circle (\diameter);
   \node [label={\diameter},text opacity=0]
     at (\x,\y) {\diameter};}
\end{tikzpicture}
```

- 如果在 <list> 中, 某个列表项缺少斜线 “/”, 那么程序会自动把该列表项补充完整, 补足过程如下:
 - (1) 如果列表项中的斜线个数不足, 且列表项不以变量值开头, 这个情况下该列表项的第 1 个变量值缺失, 程序会把第 1 个变量的值设为 1, 然后进入下一步。否则直接进入下一步。
 - (2) 检查第 1 个变量值后面是否有斜线, 如果没有斜线, 自然说明第 1 个变量值后的各斜线和变量值都缺失, 于是就补上一个 “/”, 然后令第 2 个变量值等于第 1 个变量值, 然后再补上一个 “/”, 然后令第 3 个变量值等于第 2 个变量值……直到构造出整个列表项。
如果第 1 个变量值后面有斜线, 然后检查是否有第 2 个变量值。如果有第 2 个变量值则检查第 2 个变量值后面是否有斜线; 如果没有第 2 个变量值则令第 2 个变量值等于第 1 个变量值, 然后再检查第 2 个变量值后面是否有斜线。
 - (3) 如果第 2 个变量值后面没有斜线, 自然说明第 2 个变量值后的各斜线和变量值都缺失, 于是就补上一个 “/”, 然后令第 3 个变量值等于第 2 个变量值, 然后再补上一个 “/”, 然后令第 4 个变量值等于第 3 个变量值……直到构造出整个列表项。
如果第 2 个变量值后面有斜线, 然后检查是否有第 3 个变量值。如果有第 3 个变量值则检查第 3 个变量值后面是否有斜线; 如果没有第 3 个变量值则令第 3 个变量值等于第 2 个变量值, 然后再检查第 3 个变量值后面是否有斜线。
 - (4) ……

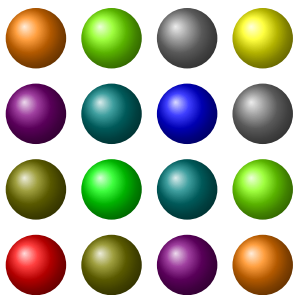
简单地说, 如果某个列表项缺少斜线 “/”, 那么该列表项中一定缺少变量值。如果缺少第 1 个变量值, 就把第 1 个变量值设为 1; 如果缺少其它变量值 (除第 1 个变量值以外), 程序会把所缺少的变量值之前的一个变量值拿来作补充, 并自动补充斜线, 直到列表项中的斜线个数 “够数”。



```
\begin{tikzpicture}
  \foreach \x / \diameter / \y in
    {0 / 2mm / 0 , 2 / 3mm / 1 , /1 }
    {\draw (\x,\y) circle (\diameter);
     \node [label={\diameter},text opacity=0.3]
       at (\x,\y) {\diameter};}
\end{tikzpicture}
```



```
\begin{tikzpicture}[line cap=round,line width=3pt]
  \filldraw [fill=yellow!80!black] (0,0) circle (2cm);
  \foreach \angle / \label in
    {0/3, 30/2, 60/1, 90/12, 120/11, 150/10, 180/9, 210/8, 240/7, 270/6, 300/5,
     330/4}
  {
    \draw[line width=1pt] (\angle:1.8cm) -- (\angle:2cm);
    \draw (\angle:1.4cm) node{\textsf{\label}};
  }
  \foreach \angle in {0,90,180,270}
  {
    \draw[line width=2pt] (\angle:1.6cm) -- (\angle:2cm);
    \draw (0,0) -- (120:0.8cm); % 时针
    \draw (0,0) -- (90:1cm); % 分针
  }
\end{tikzpicture}
```



```
\tikz[shading=ball]
\foreach \x / \cola in {0/red,1/green,2/blue,3/yellow}
\foreach \y / \colb in {0/red,1/green,2/blue,3/yellow}
\shade[ball color=\cola!50!\colb] (\x,\y) circle (0.4cm);
```

下面的例子用 foreach 语句粗略地实现一种文字效果:

好

```
\tikz [scale=3]
{
  \foreach \i in {0,0.02,...,1.5}
    \node [transform shape,evaluate={\c=\i*60;}, text=red!\c,
          opacity=\i]at(-\i pt,-0.5*\i pt){好};
}
```

83.6 针对变量的选项

/pgf/foreach/var=<variable>

这个选项用于声明变量名称, 举例来说, 以下两个叙述等效:

```
\foreach \x/\y
\foreach [var=\x,var=\y]
```

注意如果使用这个选项, 那么该选项应该放在其它变量选项之前, 因为其它选项需要变量名称。

/pgf/foreach/evaluate=<variable>

在 <list> 中, 列表项可能算式, 例如,

```
\foreach \x in {2^0,2^...,2^4}{\x$, }
```

得到 2^0 , 2^1 , 2^2 , 2^3 , 2^4 , 尽管算式 2^0 的值是 1, 但是命令会把算式 2^0 赋予变量 \x , 而不是把 1 赋予变量 \x . 如果想把 <list> 中列表项算式的值赋予变量, 就使用这个选项。

<variable> 用于指定该选项针对哪个变量。

1.0, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0, 128.0, 256.0,

```
\foreach \x [evaluate=\x] in {2^0,2^...,2^8}{\x$, }
```

[1.0, 3^0], [2.0, 3^1], [4.0, 3^2],

```
\foreach \x / \y [evaluate=\x] in {2^0/3^0, 2^1/3^1, 2^2/3^2} {\x, \y$, }
```

/pgf/foreach/evaluate=<variable> as <macro>

这个选项定义一个宏 <macro>, 计算列表项中关于变量 <variable> 的算式的值, 并把这个值赋予宏 <macro>, 而变量 <variable> 的值仍然是列表项中的算式。宏 <macro> 储存了算式的值, 可以用在 <commands> 中参与各种运算。

$2^0 = 1.0$, $2^1 = 2.0$, $2^2 = 4.0$, $2^3 = 8.0$, $2^4 = 16.0$, $2^5 = 32.0$, $2^6 = 64.0$, $2^7 = 128.0$, $2^8 = 256.0$,

```
\foreach \x [evaluate=\x as \xeval] in {2^0,2^...,2^8} {\x=\xeval$, }
```

/pgf/foreach/evaluate=<variable> as <macro> using <formula>

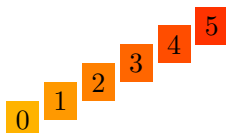
这里的 <formula> 是用 <variable> 构造的表达式。这个选项定义一个宏 <macro>, 利用 <variable> 的值计算 <formula> 的结果, 并把这个结果赋予宏 <macro>, 而变量 <variable> 的值仍然是列表项中的算式。宏 <macro> 储存了 <formula> 的结果, 可以用在 <commands> 中参与各种运算。



```
\tikz\foreach \x [evaluate=\x as \shade using \x*10] in {0,1,...,10}
  \node [fill=red!\shade!yellow, minimum size=0.65cm] at (\x,0) {\x};
```

`/tikz/evaluate={<formula>}`

这个是程序库 `math` 提供的选项, 其中的 `<formula>` 应当是数学引擎或数学程序库能够接受的表达式或者语句。



```
\tikz\foreach \x [count=\a,
  evaluate={\i=0.5*\x;\j=10*\x+30;}
  in {0,1,...,5}
  \node [fill=red!\j!yellow] at (0.5*\a,0.5*\i) {\x};
```

上面例子中, 选项 `evaluate=` 的参数是数学程序库中的赋值语句。

注意很多情况下 `foreach` 语句输出的数值结果默认是带小数点的, 如果需要修改输出数值的格式可以参考 §89, §92, 例如:

1, 2, 4, 8, 16, 32, 64, 128, 256,

```
\foreach \x [evaluate=\x as \xeval using int(\x)] in {2^0,2^... ,2^8}{\xeval, }
```

1, 2, 4, 8, 16, 32, 64, 128, 256,

```
\foreach \x [evaluate=\x as \xeval] in {2^0,2^... ,2^8}
  {\pgfkeys{/pgf/number format/int trunc} \pgfmathprintnumber{\xeval}, }
```

`/pgf/foreach/remember=<variable> as <macro> (initially <value>)`

假设变量 `\x` 的值域是 $\{a_1, a_2, \dots\}$, 而 `<commands>` 的作用相当于一个二元操作 $f(a_{n-1}, a_n)$, 也就是说, `<commands>` 是针对 `\x` 的当前值和前一个值的操作, 此时可以使用本选项。

这个选项定义宏 `<macro>`, 并将 `\x` 的当前值的前一个值赋予宏 `<macro>`. 其中 `initially <value>` 是可选的, 规定这个宏的初值为 `<value>`. 可以在 `<commands>` 中使用宏 `<macro>`.

```
\foreach \x [remember=\x as \lastx (initially A)] in {B,...,F}
```

等价于

```
\foreach \x / \lastx in {B/A,...,F/E}
```

$\overrightarrow{AB}, \overrightarrow{BC}, \overrightarrow{CD}, \overrightarrow{DE}, \overrightarrow{EF},$
初值 $\overrightarrow{C}, \overrightarrow{CD}, \overrightarrow{DE}, \overrightarrow{EF},$

```
\foreach \x [remember=\x as \lastx (initially A)]
  in {B,...,F}{\overrightarrow{\lastx\x}$, }
\foreach \x [remember=\x as \lastx
  (initially {\text{初值}})]
  in {C,...,F}{\overrightarrow{\lastx\x}$, }
```

/pgf/foreach/count=<macro> from <value>

这个选项定义宏 <macro>, 并把变量的当前值在 <list> 中的序号赋予 <macro>, 可以在 <commands> 中使用宏 <macro>. 其中的 from <value> 是可选的, 当使用这个词组后, 宏 <macro> 的初始值就是 <value>, 即第 1 次执行 <commands> 时宏 <macro> 的值是 <value>, 第 2 次执行 <commands> 时宏 <macro> 的值是 <value>+1, ……

```
\foreach \x [count=\xi from -2] in {a,...,e}
```

等价于

```
\foreach \x / \xi in {a/-2,...,e/3}
```

aa	bb	cc	dd	ee
ab	bc	cd	de	
ac	bd	ce		
ad	be			
ae				

```
\tikz[x=0.75cm,y=0.75cm]
```

```
\foreach \x [count=\xi] in {a,...,e}
```

```
\foreach \y [count=\yi] in {\x,...,e}
```

```
\node [draw, top color=white, bottom color=blue!50, minimum size=0.666cm]
at (\xi,-\yi) {$\mathstrut\x\y$};
```

\breakforeach

这个命令用在 \foreach 语句的 <commands> 中. \foreach 语句会执行 <commands> 数次, 在每次执行 <commands> 的过程中, 如果执行了 \breakforeach, 就会终止本次执行, 进入下一次执行. 这是对 <commands> 的处理过程作出的限制。



```
\begin{tikzpicture}
\foreach \x in {1,...,4}
\foreach \y in {1,...,4}
{
\fill[red!50] (\x,\y) ellipse (3pt and 6pt);
\ifnum \x<\y
\breakforeach
\fi
}
\end{tikzpicture}
```

86 扩展颜色支持

调用宏包 `\usepackage{xxcolor}`.

宏包 `xxcolor` 是 PGF 的一个子宏包,它扩展了宏包 `color` 的特性.宏包 `xxcolor` 提供环境 `{colormixin}` 和命令 `\colorcurrentmixin`.

```
\begin{colormixin}{<mix-in specification>}
  <environment contents>
\end{colormixin}
```

宏包 `xxcolor` 定义了类似 `black!25!white` 这样的颜色表达式,意思是将 25% 的黑色与 75% 的白色混合.将这种颜色表达式的第一个颜色及第一个感叹号“!”去掉就是这里的 `<mix-in specification>`,例如 `25!white`,我们以此为例继续介绍.

在环境内容 `<environment contents>` 中可能有多种内容,如文字,表格线,图形,从外部插入的图形等等,这些内容都有自己原本的颜色.例如,假设文字的颜色原本是黑色 `black`,那么本环境就会把文字的颜色变成 `black!25!white`;如果某个线条的颜色是自定义颜色 `<self-def-color>`,那么这个线条的颜色就变成 `<self-def-color>!25!white`.也就是说,某个颜色成分与环境内容的原色相混合,修改了内容的颜色,为了方便,称这个颜色成分为“修改色”.

Red text, washed-out red
text, washed-out blue
text, dark washed-out
blue text, dark
washed-out green text,
back to washed-out blue
text, ● and back to red.

```
\begin{minipage}{4cm}\raggedright
  \color{red}Red text,%
  \begin{colormixin}{25!white}
    washed-out red text,
    \color{blue} washed-out blue text,
    \begin{colormixin}{25!black}
      dark washed-out blue text,
      \color{green} dark washed-out green text,%
    \end{colormixin}
    back to washed-out blue text,%
    \tikz \fill [green] circle(4pt);
  \end{colormixin}
  and back to red.
\end{minipage}%
```

注意本环境未必能改变所有内容的颜色,但一定能改变命令 `\color` 规定的颜色.另外命令 `\pgfuseimage` 和 `\pgfuses shading` 支持本环境,见 §105, §109.

以下的代码为例,

```
\begin{colormixin}{25!white}
  \begin{pgfpicture}
    \pgftext[at=\pgfpoint{1cm}{5cm},left,base] {\pgfuseimage{image}}
    \pgfusepath{stroke}
```

```
\end{pgfpicture}
\end{colormixin}
```

上面的代码中，在 `{colormixin}` 环境内使用了 `{pgfpicture}` 环境，命令 `\pgfuseimageimage` 指定插入名称为 `image` 的图形，但实际上 PGF 会先查找并插入名称为 `image!``25!white` 的图形，这个名称是原图名称与环境 `{colormixin}` 的参数用感叹号 “!” 连接起来的符号串。

当套嵌使用 `{colormixin}` 环境时，各环境的参数 `<mix-in specification>` 会按照次序，由感叹号 “!” 连接起来，构成 “修改色” 来修改环境内容的颜色。当多个 `{colormixin}` 环境套嵌时，“修改色” 可能不太容易梳理清楚，可以用命令 `\colorcurrentmixin` 返回当前的 “修改色”。

`\colorcurrentmixin`

本命令返回当前的 “修改色”。

```
!75!white should be “!75!white”
!75!black!75!white should be “!75!black!75!white”
!50!white!75!black!75!white should be “!50!white!75!black!75!white”
!50!green!75!black!75!white should be “!50!green!75!black!75!white”
```

```
\begin{minipage}{\linewidth-6pt}\tt \raggedright
\begin{colormixin}{75!white}
\colorcurrentmixin\ should be “!75!white” \par
\begin{colormixin}{75!black}
\quad\colorcurrentmixin\ should be “!75!black!75!white” \par
\begin{colormixin}{50!white}
\quad\quad\colorcurrentmixin\ should be “!50!white!75!black!75!white” \par
\end{colormixin}
\begin{colormixin}{50!green}
\quad\quad\colorcurrentmixin\ should be “!50!green!75!black!75!white” \par
\end{colormixin}
\end{colormixin}
\end{colormixin}
\end{minipage}
```

如果在套嵌的 `{colormixin}` 环境中使用命令 `\pgfuseimageimage`，也会影响图形文件的搜索名称。例如，

```
\begin{colormixin}{25!white}
\begin{colormixin}{50!green}
\begin{pgfpicture}
\pgftext[at=\pgfpoint{1cm}{5cm},left,base] {\pgfuseimage{image}}
\pgfusepath{stroke}
\end{pgfpicture}
\end{colormixin}
\end{colormixin}
```

```
\end{colormixin}
```

上面代码中，命令 `\pgfuseimageimage` 指定插入名称为 `image` 的图形，但实际上 PGF 会先查找并插入名称为 `image.!50!green!25!white` 的图形。

经测试，这个查找并插入图形的机制未能实现。

88 数学引擎概略

PGF 的数学引擎是个宏包，PGF 会自动载入数学引擎，数学引擎也可以独立于 PGF 使用

```
\usepackage{pgfmath}
```

当在某个高层次的宏中使用数字或者带单位的尺寸时，就默认使用数学引擎来解析输入。

数学引擎有 3 个层次：

- 顶层，提供命令 `\pgfmathparse`，还有一些能够设置尺寸或计数器的函数。
- 计算层。
- 实现层。

目前，数学引擎完全在 \TeX 中做成，由于 \TeX 是个排版程序而不是专门的数学程序，所以某些情况下数学引擎的计算精度和效率可能较差。

89 数学表达式

在解析数学表达式时，在任何时刻都应保证计算范围不超过 ± 16383.99999 ，这是 \TeX 允许的尺寸。为了顺利理解下文中的命令，先解释一下 \TeX 的寄存器（register）。 \TeX 的寄存器分为多种，每一种都有自己的名称和编号，每个寄存器都可以保存某种特别类型的值。例如，

- 有 256 个整数寄存器：`\count0`，`\count1`，……`\count255`，每个都可以保存一个整数，但整数范围限制为 ± 2147483647 。用 `\count0` 可以引用保存在该寄存器中的整数值。从 `\count0` 到 `\count22` 这 23 个整数寄存器是系统自己使用的，最好不要随意改变它们的值。
- 有 256 个（刚性）尺寸寄存器：`\dimen0`，`\dimen1`，……`\dimen255`，每个都可以保存一个尺寸。
- 有 256 个弹性尺寸寄存器：`\skip0`，`\skip1`，……`\skip255`，每个都可以保存一个弹性尺寸。
- 有 256 个弹性数学尺寸寄存器：`\muskip0`，`\muskip1`，……`\muskip255`，每个都可以保存一个弹性数学尺寸。
- 有 256 个盒子寄存器：`\box0`，`\box1`，……`\box255`，每个都可以保存一个盒子。
- 有 256 个记号列（token list）寄存器：`\toks0`，`\toks1`，……`\toks255`，每个都可以保存一个记号（token）。

介绍几个命令。

`\wd` 这个命令的句法是 `\wd<number>=<dimension>`，将编号为 `<number>` 的盒子的宽度设为 `<dimension>`，如果不给出“`<dimension>`”就默认其值为盒子的自然宽度。而 `\wd<number>` 得到编号为 `<number>` 的盒子的宽度。

`\dp` 这个命令与 `\wd` 类似，只是针对盒子深度。

`\ht` 这个命令与 `\wd` 类似，只是针对盒子高度。

`\dimexpr` 这个命令的句法是 `\dimexpr<dimension expression>`, `<dimension expression>` 是关于 (带单位的) 尺寸的算术表达式 (即只用加减乘除构造的算式), 该命令计算这个表达式的值。

`\numexpr` 这个命令的句法是 `\numexpr<integer expression>`, `<integer expression>` 是关于整数的算术表达式, 该命令计算这个表达式的值。

`\muexpr` 这个命令的句法是 `\muexpr<mu expression>`, `<mu expression>` 是关于数学长度 (带有单位 `mu` 的尺寸) 的算术表达式, 该命令计算这个表达式的值。

`\the` 这个命令的句法是 `\the<命令>`, 可以输出保存在 `<命令>` 中的数值、尺寸、代码等。

`\countdef` 这个命令的句法是 `\countdef<command>=<number>`. 这个句法创建命令 `<command>`, 故 `<command>` 应是以反斜线开头的命令形式。这个句法将命令 `<command>` 等同于编号为 `<number>` 的整数寄存器 `\count<number>`, 故对 `<command>` 进行操作等同于对 `\count<number>` 进行操作。

`\dimendef` 这个命令的句法是 `\dimendef<command>=<number>`. 这个句法创建命令 `<command>`, 故 `<command>` 应是以反斜线开头的命令形式。这个句法将命令 `<command>` 等同于编号为 `<number>` 的尺寸寄存器 `\dimen<number>`, 故对 `<command>` 进行操作等同于对 `\dimen<number>` 进行操作。

`\skipdef` 这个命令的句法是 `\skipdef<command>=<number>`. 这个句法创建命令 `<command>`, 故 `<command>` 应是以反斜线开头的命令形式。这个句法将命令 `<command>` 等同于编号为 `<number>` 的弹性尺寸寄存器 `\skip<number>`, 故对 `<command>` 进行操作等同于对 `\skip<number>` 进行操作。

`\muskipdef` 这个命令的句法是 `\muskipdef<command>=<number>`. 这个句法创建命令 `<command>`, 故 `<command>` 应是以反斜线开头的命令形式。这个句法将命令 `<command>` 等同于编号为 `<number>` 的弹性数学尺寸寄存器 `\muskip<number>`, 故对 `<command>` 进行操作等同于对 `\muskip<number>` 进行操作。

`\setbox` 这个命令的句法是 `\setbox<number>=<box>`. 这个句法将盒子 `<box>` 中的内容赋予编号为 `<number>` 的盒子寄存器 `\box<number>`.

`\toksdef` 这个命令的句法是 `\toksdef<命令>=<number>`. 这个句法将第 `<number>` 个记号列寄存器 `\toks<number>` 等同于 `<命令>`.

`\newbox` 这个命令的句法是 `\newbox<command>`, 用于自定义盒子命令 `<command>`, 故 `<command>` 应是以反斜线开头的命令形式。这个句法将 `<command>` 创建一个盒子, 系统会自动为它分配一个盒子寄存器, 即它会占用一个盒子寄存器。自定义盒子命令占用的盒子寄存器是从 `\box26` 到 `box233`, 共 208 个, 即最多可以自定义 208 个盒子命令, 过多的自定义盒子命令会导致错误信息: `No room for a new \box.`

`\newtoks` 这个命令的句法是 `\newtoks<command>`, 将 `<command>` 创建一个记号列寄存器, 即它会占用一个记号列寄存器。

`\multiply` 这个命令的句法是 `\multiply<a> by `, 本命令计算 `<a>` 与 `` 的乘积, 其中 `<a>` 可以是数值、长度、代表数值或长度的命令, `` 应该是数值、整数、代表数值或整数的命令。

89.1 解析一个表达式

89.1.1 命令

`\pgfmathparse{<expression>}`

这个宏解析 `<expression>`, 如果 `<expression>` 中有长度单位, 就先把所有长度都转换为以 `pt` 为单位的

长度;如果 `<expression>` 中没有长度单位,就把所有数字当作以 pt 为单位的长度,再计算 `<expression>` 的结果。然后将计算结果中的单位 pt 去掉,仅仅把计算结果中的数字(作为一个十进制数字)保存在宏 `\pgfmathresult` 中,使用这个宏可以输出结果数字。

例如,执行 `\pgfmathparse{2pt+3.5pt}` 后,可以用 `\pgfmathresult` 输出 5.5。

5.5 5.5

```
\pgfmathparse{2+3.5pt} \pgfmathresult \quad
\pgfmathparse{2pt+3.5pt} \pgfmathresult
```

关于这个宏需要注意以下几点:

- 在各个情况下,保存在宏 `\pgfmathresult` 中的总是十进制无单位的数值。
- 解析器能够识别并处理 $\text{T}_{\text{E}}\text{X}$ 的寄存器和盒子尺寸,类似 `\mydimen` (表示一个自定义的尺寸), `0.5\mydimen` (表示自定义尺寸的一半), `\wd\mybox`, `0.5\dp\mybox`, `\mycount\mydimen` 这样的尺寸或者计数器数值都是可以用在表达式 `<expression>` 中的。
- 解析器能识别并处理 ϵ - $\text{T}_{\text{E}}\text{X}$ 扩展的命令 `\dimexpr`, `\numexpr`, `\glueexpr`, `\muexpr`, 只需要在这些命令前面加上 `\the` 就可以用这些命令的结果参与运算。
- 在表达式 `<expression>` 中可以使用圆括号来规定运算次序。
- 在表达式 `<expression>` 中可以使用多种函数,函数的参数也可以是表达式。
- 表达式 `<expression>` 接受科学计数法,如 `1.234e+4`,其中的指数符号可用小写 `e` 或大写 `E`。
- 在表达式 `<expression>` 中,如果一个整数以 0 开头,就默认这个整数是八进制数,会被自动转为十进制数。
- 在表达式 `<expression>` 中,如果一个整数以 `0x` 或 `0X` 开头,就默认这个整数是十六进制数,会被自动转为十进制数。十六进制数中的字母符号可以是大写也可以是小写。
- 在表达式 `<expression>` 中,如果一个整数以 `0b` 或 `0B` 开头,就默认这个整数是二进制数,会被自动转为十进制数。
- 在表达式 `<expression>` 中,如果有一串符号被双引号括起来,就忽略这一串符号。

`\pgfmathqparse{<expression>}`

这个宏与 `\pgfmathparse` 类似,解析 `<expression>` 并将结果作成无单位的数值保存在宏 `\pgfmathresult` 中。注意:(1) `\pgfmathqparse` 不解析函数、科学计数法、非十进制数,也不把双引号、问号、冒号当作具有特殊作用的符号。前面提到,表达式中双引号引起来的部分会被忽略。问号“?”和冒号“:”用于构成条件句。(2)除了像 `0.5\pgf@x` 这种式子,`<expression>` 中所有数值后面都要带单位。因为这两个限制,`\pgfmathqparse` 的速度是 `\pgfmathparse` 的两倍。

`\pgfmathpostparse`

这个命令通常用在解析过程结束之后,它能修改 `\pgfmathresult` 中的数值。在默认下该命令等于 `\relax`。

下面介绍几个能给寄存器或计数器赋值或增值的命令。对于这几个命令,如果其中的 `<expression>` 以加号“+”开头,那么就不会解析 `<expression>`,仅仅是 $\text{T}_{\text{E}}\text{X}$ 的赋值或增值。这些命令的有效范围受到 $\text{T}_{\text{E}}\text{X}$ 分组的限制。

`\pgfmathsetlength{<register>}{<expression>}`

<register> 代表一个 T_EX 的寄存器。

如果 <expression> 以加号 “+” 开头就只是给 <register> 赋值，举例来说：

1.0pt plus 1.0fil

```
\skipdef\myskip=0 % 定义弹性尺寸命令 \myskip
\pgfmathsetlength{\myskip}{+1pt plus 1fil} \the\myskip
```

上面例子中，先定义弹性尺寸命令 `\myskip`，该命令占用寄存器 `\skip0`，然后为它赋以弹性尺寸。

如果 <expression> 不以加号 “+” 开头，就用命令 `\pgfmathparse` 解析表达式，解析结果的数值部分会保存在 `\pgfmathresult` 中。在解析 <expression> 时，如果解析器遇到数学单位 `mu`，解析器就会默认 <register> 是 `\muskip` 类型的寄存器，会把 `\pgfmathresult` 中的数值带上单位 `mu`，赋予 <register>。在解析 <expression> 时，如果解析器没有遇到数学单位 `mu`，解析器就会默认 <register> 是刚性尺寸寄存器或弹性尺寸寄存器，会把 `\pgfmathresult` 中的数值带上单位 `pt`，赋予 <register>。

13.0mu

```
\muskipdef\mymuskip=0
\pgfmathsetlength{\mymuskip}{1mu+3*4mu} \the\mymuskip
```

13.0pt

```
\dimendef\mydimen=0
\pgfmathsetlength{\mydimen}{1pt+3*4pt} \the\mydimen
```

13.0pt

```
\skipdef\myskip=0
\pgfmathsetlength{\myskip}{1+3*4} \the\myskip
```

注意，下面的命令

```
\pgfmathsetlength{\myskip}{1pt plus 1fil}
```

是无效的，因为目前解析器还不支持 `fil`。

`\pgfmathaddtlength{<register>}{<expression>}`

该命令的注意事项同 `\pgfmathsetlength`。该命令将 <expression> 加到 <register> 中。

`\pgfmathsetcount{<count register>}{<expression>}`

该命令针对整数寄存器，注意事项同 `\pgfmathsetlength`。首先解析 <expression>，如果解析结果是小数，就直接去掉小数部分（没有“四舍五入”），将整数部分作为整数寄存器 <count register> 的值。

`\pgfmathaddtocount{<count register>}{<expression>}`

该命令针对整数寄存器，先解析 <expression>，如果解析结果是小数，就直接去掉小数部分（没有“舍入”），将整数部分加到整数寄存器 <count register> 中。

`\pgfmathsetcounter{<counter>}{<expression>}`

该命令针对 L^AT_EX 计数器，先解析 <expression>，如果解析结果是小数，就直接去掉小数部分（没有“四舍五入”），将整数部分作为计数器 <counter> 的值。

`\pgfmathaddtocounter{<counter>}{<expression>}`

该命令针对 L^AT_EX 计数器，先解析 <expression>，如果解析结果是小数，就直接去掉小数部分（没有“四舍五入”），将整数部分加到计数器 <counter> 中。

`\pgfmathsetmacro{<macro>}{<expression>}`

该命令定义宏 <macro>，并把解析 <expression> 的结果，即保存在 `\pgfmathresult` 中的数值赋予 <macro>，注意解析结果是无单位的数值。

`\pgfmathsetlengthmacro{<macro>}{<expression>}`

该命令定义宏 <macro>，并把解析 <expression> 的结果，即保存在 `\pgfmathresult` 中的数值带上单位 pt 赋予 <macro>。

`\pgfmathtruncatemacro{<macro>}{<expression>}`

该命令定义宏 <macro>，并把解析 <expression> 的结果去掉小数部分（无“舍入”），只把整数部分赋予 <macro>。

89.2 长度单位的“显”、“隐”

`\ifpgfmathunitsdeclared`

这是个 T_EX 的条件判断命令，如果在该命令之前曾经使用 `\pgfmathparse` 解析表达式，并且表达式中出现了长度单位，那么 `\ifpgfmathunitsdeclared` 的真值就是 true，否则它的真值就是 false，这个条件判断命令是全局有效的。

`scalar(<exp>)`

`\pgfmathscalar{<exp>}`

这里 <exp> 是个表达式，这个函数将 <exp> 的解析结果变成一个无单位的数值 (scalar)，故它使得 `\ifpgfmathunitsdeclared` 忽略 <exp> 中的长度单位，而且还将 `\ifpgfmathunitsdeclared` 的真值设为 false。

0.5 without unit

```
\pgfmathparse{scalar(1pt/2pt)} \pgfmathresult\
\ifpgfmathunitsdeclared with \else without \fi unit
```

注意对于 `\ifpgfmathunitsdeclared` 来说，`1pt+scalar(1pt)` 与 `scalar(1pt)+1pt` 是不同的表达式：

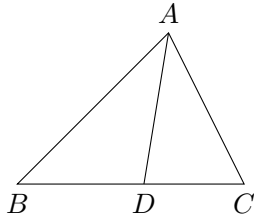
2.0 without unit

```
\pgfmathparse{1pt+scalar(1pt)} \pgfmathresult\
\ifpgfmathunitsdeclared with \else without \fi unit
```

2.0 with unit

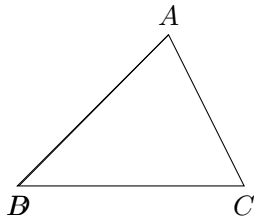
```
\pgfmathparse{scalar(1pt)+1pt} \pgfmathresult\
\ifpgfmathunitsdeclared with \else without \fi unit
```

一般情况下，在 `\tikzmath{}` 中所做的赋值，如 `\a=1+2;` 是不带单位的，保存在 `\pgfmathresult` 中的值是无单位的，其它情况下表达式的计算结果通常是带单位的，当引用带单位的计算结果时，如果只需要结果的数值部分参与运算，就可以使用 `scalar` 函数。比较下面两个例子：



```
\tikz{
  \coordinate["$A$"] (A) at (2,2);
  \coordinate["$B$" below] (B) at (0,0);
  \coordinate["$C$" below] (C) at (3,0);
  \draw (A) -- (B) -- (C) -- cycle;
  \path
    let \p1 =($A)-(B$), \p2 =($A)-(C$),
        \n1 = {veclen(\x1,\y1)}, \n2 = {veclen(\x2,\y2)}
    in coordinate ["$D$" below]
        (D) at ($ (B)!scalar(\n1/(\n1+\n2))!(C) $);
  \draw (A) -- (D);
}
```

在上面的例子中， $\text{\n1}/(\text{\n1}+\text{\n2})$ 是带单位的，用 `scalar` 将其包裹起来得到无单位的数值，否则构造的坐标算式无效，如下



```
\tikz{
  \coordinate["$A$"] (A) at (2,2);
  \coordinate["$B$" below] (B) at (0,0);
  \coordinate["$C$" below] (C) at (3,0);
  \draw (A) -- (B) -- (C) -- cycle;
  \path
    let \p1 =($A)-(B$), \p2 =($A)-(C$),
        \n1 = {veclen(\x1,\y1)}, \n2 = {veclen(\x2,\y2)}
    in coordinate ["$D$" below]
        (D) at ($ (B)!\n1/(\n1+\n2)!(C) $);
  \draw (A) -- (D);
}
```

TEX 的数学单位 “mu” (math units) 会被特殊处理。

`\ifpgfmathmathunitsdeclared`

这个 TEX 条件判断命令与 `\ifpgfmathunitsdeclared` 类似，只是针对数学单位 mu，并且函数 `scalar` 对该命令无影响。

89.3 数学表达式中的算子

`x + y` 中置算子，等效于 `add` 函数

`x - y` 中置算子，等效于 `subtract` 函数

`- x` 前置算子，等效于 `neg` 函数

`x * y` 中置算子，等效于 `multiply` 函数

x / y 中置算子, 等效于 `divide` 函数

如果 y 是 0 会导致错误。

$x \wedge y$ 中置算子, 等效于 `pow` 函数

计算 x^y 。

$x!$ 后置算子, 等效于 `factorial` 函数

计算阶乘。

xr 后置算子, 等效于 `deg` 函数

这个式子假定 x 为弧度数, 并将 x 转为角度。

$x ? y : z$ 条件算子, 等效于 `ifthenelse` 函数

如果式子 x 的计算结果是非 0 值, 则认为其值是 `true`。

$x == y$ 中置算子, 等效于 `equal` 函数

$x > y$ 中置算子, 等效于 `greater` 函数

$x < y$ 中置算子, 等效于 `less` 函数

$x != y$ 中置算子, 等效于 `notequal` 函数

$x >= y$ 中置算子, 等效于 `notless` 函数

$x <= y$ 中置算子, 等效于 `notgreater` 函数

$x <= y$ 中置算子, 等效于 `notgreater` 函数

$x \&\& y$ 中置算子, 等效于 `and` 函数

$x || y$ 中置算子, 等效于 `or` 函数

$!x$ 前置算子, 等效于 `not` 函数

(x) 组算子

圆括号有两个用处, 一是用来规定运算次序, 一是用来标示数学函数的参数, 例如 `sin(30*10)` 或 `mod(72,3)`. 注意, `sin 30` (留意其中的空格) 等效于 `sin(30)`, 而 `sin 30*10` (留意其中的空格) 等效于 `sin(30)*10`.

{x} 组算子

在数学表达式中, 花括号可以构造数组, 例如, `{1,2,3}`. 花括号还可以套嵌起来构成多维数组, 例如, `{1, {2,3}, {4,5}, 6}`, 这是个 2 维数组。可以用 $\text{T}_{\text{E}}\text{X}$ 的定义命令将数组保存在一个宏中, 以便于在别处引用, 例如,

```
\def\myarray{1,2,3}
```

1, two, 3.0, IV, cinq, sechs, 7.0,

```
\def\myarray{1,"two",2+1,"IV","cinq","sechs",sin(\i*5)*14}
```

```
\foreach \i in {0,...,6}{\pgfmathparse{\myarray[\i]}\pgfmathresult, }
```

[x] 数组索引算子

方括号可用以索引数组, 数组中元素的编号从 0 开始, 如果索引号过大或过小都会导致错误。

```
5 \pgfmathparse{{1,2,3,{4,5}}[3][1]} \pgfmathresult
```

下面再看一个索引多维数组的例子。索引多维数组时，可以连续使用多个方括号括起来的索引号，一个方括号确定一个“地址”。

```
1 0 0
0 1 0
0 0 1
```

```
\def\print#1{\pgfmathparse{#1}\pgfmathresult}
\def\identitymatrix{{{1,0,0},{0,1,0},{0,0,1}}}
\tikz[x=0.5cm,y=0.5cm]\foreach \i in {0,1,2} \foreach \j in {0,1,2}
  \node at (\j,-\i) [anchor=base] {\print{\identitymatrix[\i][\j]}};
```

"x" 组算子

双引号引起来的部分表示“引用”，如果在双引号之内有宏，那么在数学引擎解析整个表达式之前，这些宏会被展开，这类似命令 `\edef` 的作用。如果不希望这些宏被展开，就在这些宏的前面加上 `\noexpand`，例如，`\noexpand\Huge`。

5 is Bigger than 0. 5 is smaller than 10.

```
\def\x{5}
\foreach \y in {0,10}{
  \pgfmathparse{\x > \y ? "\noexpand\Large Bigger" : "\noexpand\tiny smaller"}
  \x is \pgfmathresult\ than \y.
}
```

89.4 数学表达式中的函数

每个函数都有对应的 PGF 命令版本，注意 PGF 命令一般不用做命令 `\pgfmathparse` 的参数，而是单独使用，它们都把结果保存在命令 `\pgfmathresult` 中。例如：

```
3.0 \pgfmathadd{1}{2} \pgfmathresult
```

89.4.1 基本算术函数

首先介绍关于取整函数、除法余数的概念，参考

https://en.wikipedia.org/wiki/Floor_and_ceiling_functions

<https://en.wikipedia.org/wiki/Truncation>

https://en.wikipedia.org/wiki/Modulo_operation

⇒ 对一个实数有 3 种取整方式：

Floor 向下取整，如 $\lfloor 2.7 \rfloor = 2$ ， $\lfloor -2.3 \rfloor = -3$ 。

Ceil 向上取整，如 $\lceil 2.3 \rceil = 3$ ， $\lceil -2.7 \rceil = -2$ 。

向 0 取整 即直接去掉小数部分，只保留整数部分。这个取整方式可以由前两种取整方式定义，即对正实数向下取整，对负实数向上取整。

⇒ 可以用取整函数定义针对小数的“截尾”操作，即把某个位置之后的小数数字全部去掉，不做舍入。

如果 $x \in \mathbb{R}_+$ ，设 $n \in \mathbb{N}_0$ ，对 x 的截尾可以是

$$\text{trunc}(x, n) = \frac{\lfloor 10^n \cdot x \rfloor}{10^n}.$$

如果 $x \in \mathbb{R}_-$ ，设 $n \in \mathbb{N}_0$ ，对 x 的截尾可以是

$$\text{trunc}(x, n) = \frac{\lceil 10^n \cdot x \rceil}{10^n}.$$

⇒ 在做整数除法时，通常用的是欧几里得辗转相除法，例如计算 $a \div b$ ，最终的结果表现为：

$$a = b \cdot q + r, \quad q \in \mathbb{Z}, \quad |r| < |b|.$$

在数学上，一般规定 $r \geq 0$ ，但在程序计算上有多种选择：

1. 令 r 的符号非负。此时， $0 \leq r < b$ ，

$$\frac{a}{|b|} = \begin{cases} q + \frac{r}{b}, & b > 0, \\ -q - \frac{r}{b}, & b < 0, \end{cases} \quad \text{故} \quad \left\lfloor \frac{a}{|b|} \right\rfloor = \begin{cases} q, & b > 0, \\ -q, & b < 0, \end{cases}$$

所以

$$r = a - \text{sign}(b) \cdot b \cdot \left\lfloor \frac{a}{|b|} \right\rfloor.$$

2. 令 r 的符号与 $a \div b$ 相同。
3. 令 r 的符号与 a 相同，这个情况叫作 truncated division，
 - 若 $a > 0$ ，则 $r > 0$ 且 $|r| < |b|$ ，此时

$$\frac{a}{|b|} = \begin{cases} q + \frac{r}{b}, & b > 0, \\ -q - \frac{r}{b}, & b < 0, \end{cases} \quad \text{故} \quad \left\lfloor \frac{a}{|b|} \right\rfloor = \begin{cases} q, & b > 0, \\ -q, & b < 0, \end{cases}$$

所以

$$r = a - \text{sign}(b) \cdot b \cdot \left\lfloor \frac{a}{|b|} \right\rfloor.$$

- 若 $a < 0$ ，则 $r < 0$ 且 $|r| < |b|$ ，此时

$$\frac{a}{|b|} = \begin{cases} q + \frac{r}{b}, & b > 0, \\ -q - \frac{r}{b}, & b < 0, \end{cases} \quad \text{故} \quad \left\lceil \frac{a}{|b|} \right\rceil = \begin{cases} q, & b > 0, \\ -q, & b < 0, \end{cases}$$

所以

$$r = a - \text{sign}(b) \cdot b \cdot \left\lceil \frac{a}{|b|} \right\rceil.$$

4. 令 r 的符号与 b 相同，这个情况叫作 floored division，

$$\frac{a}{b} = q + \frac{r}{b}, \quad \text{故} \quad \left\lfloor \frac{a}{b} \right\rfloor = q,$$

所以

$$r = a - b \cdot \left\lfloor \frac{a}{b} \right\rfloor.$$

举例来说，因为 $-4 = 3 \cdot (-1) - 1$ ，所以按 truncated division 方式计算 $-4 \div 3$ 的余数就是 -1 ；另一方面 $-4 = 3 \cdot (-2) + 2$ ，所以按 floored division 方式计算 $-4 \div 3$ 的余数就是 2 。

`add(x,y)`

`\pgfmathadd{x}{y}`

加法

81.0

`\pgfmathparse{add(75,6)} \pgfmathresult`

`subtract(x,y)`

`\pgfmathsubtract{x}{y}`

减法

`neg(x)`

`\pgfmathneg{x}`

相反数

`multiply(x,y)`

`\pgfmathmultiply{x}{y}`

乘法

`divide(x,y)`

`\pgfmathdivide{x}{y}`

除法，计算 $x \div y$

12.5

`\pgfmathparse{divide(75,6)} \pgfmathresult`

`div(x,y)`

`\pgfmathdiv{x}{y}`

除法，并将结果四舍五入，即变成最靠近的整数。

8

`\pgfmathparse{div(75,9)} \pgfmathresult`

`factorial(x)`

`\pgfmathfactorial{x}`

计算阶乘

`sqrt(x)`

`\pgfmathsqrt{x}`

计算平方根

`pow(x,y)`

`\pgfmathpow{x}{y}`

计算 x^y ，当 y 是整数时精度较好，如果 y 不是整数则近似计算 $e^{y \ln x}$ 。

e

`\pgfmathe`

自然对数底常数，约等于 2.718281828.

`exp(x)``\pgfmathexp{x}`

计算 e^x .

`ln(x)``\pgfmathln{x}`

近似计算自然对数。

4.99997 `\pgfmathparse{ln(exp(5))}` `\pgfmathresult`

`log10(x)``\pgfmathlogten{x}`

近似计算以 10 为底的常用对数。

1.99997 `\pgfmathparse{log10(100)}` `\pgfmathresult`

`log2(x)``\pgfmathlogtwo{x}`

近似计算以 2 为底的对数。

6.99994 `\pgfmathparse{log2(128)}` `\pgfmathresult`

`abs(x)``\pgfmathabs{x}`

绝对值

`mod(x,y)``\pgfmathmod{x}{y}`

使用 truncated division 方式计算除法 $\frac{x}{y}$ 的余数，但是余数的符号与 $\frac{x}{y}$ 相同。

`Mod(x,y)``\pgfmathMod{x}{y}`

使用 floored division 方式计算除法 $\frac{x}{y}$ 的余数，但是余数的符号总是非负。

`sign(x)``\pgfmathsign{x}`

返回 x 的符号。

89.4.2 舍入函数

`round(x)`

`\pgfmathround{x}`

四舍五入。

`floor(x)`

`\pgfmathfloor{x}`

向下取整。

`ceil(x)`

`\pgfmathceil{x}`

向上取整。

`int(x)`

`\pgfmathint{x}`

返回 x 的整数部分，即向 0 取整。

`frac(x)`

`\pgfmathfrac{x}`

返回 x 的小数部分。

`real(x)`

`\pgfmathreal{x}`

声明（确保） x 是个十进制小数，带有小数点。

4.0

<code>\pgfmathparse{real(4)} \pgfmathresult</code>
--

89.4.3 几个整数运算函数

`gcd(x,y)`

`\pgfmathgcd{x}{y}`

计算 x 与 y 的最大公因子。

`isodd(x)`

`\pgfmathisodd{x}`

如果 x 的整数部分是奇数就返回 1，否则返回 0。

`iseven(x)`

`\pgfmathiseven{x}`

如果 x 的整数部分是偶数就返回 1，否则返回 0。

`isprime(x)`

`\pgfmathisprime{x}`

如果 x 的整数部分是素数就返回 1，否则返回 0.

89.4.4 三角函数

三角函数的参数都默认为角度制的数。

`pi`

`\pgfmathpi`

圆周率常数，约等于 3.141592654.

179.99962

<code>\pgfmathparse{pi r} \pgfmathresult</code>

`rad(x)`

`\pgfmathrad{x}`

假定 x 是角度制的数，并将它转换为弧度制的数。

`deg(x)`

`\pgfmathdeg{x}`

假定 x 是弧度制的数，并将它转换为角度制的数。

`sin(x)`

`\pgfmathsin{x}`

正弦函数，默认 x 是角度制的数。

`cos(x)`

`\pgfmathcos{x}`

`tan(x)`

`\pgfmathatan{x}`

`sec(x)`

`\pgfmathsec{x}`

`cosec(x)`

`\pgfmathcosec{x}`

`cot(x)`

`\pgfmathcot{x}`

`asin(x)`

`\pgfmathasin{x}`

反正弦函数，值域是 $[-90^\circ, 90^\circ]$.

`acos(x)`

`\pgfmathacos{x}`

反余弦函数，值域是 $[0^\circ, 180^\circ]$.

`atan(x)`

`\pgfmathatan{x}`

反正切函数，计算的函数值默认为角度制的数。

`atan2(y,x)`

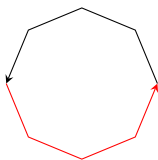
`\pgfmathatantwo{y}{x}`

计算实数 $y \div x$ 所对应的角度制的反正切数值。

-45.0 `\pgfmathparse{atan2(-4,4)} \pgfmathresult`

`/pgf/trig format=deg|rad` (无默认值, 初始值 deg)

在涉及角度的地方, 例如三角函数的参数, 极坐标点的坐标 (如 (60:2pt)), node 的角度部位 (如 a.30) 等等, 都使用角度制数值, 这是因为 PGF 的初始设置为 `trig format=deg`, 使用 `trig format=rad` 可以将这些数值转换到弧度制。



```
\begin{tikzpicture}
\draw[-stealth] (0:1) -- (45:1) -- (90:1) -- (135:1) -- (180:1);
\draw[-stealth,trig format=rad,red]
(pi:1) -- (5/4*pi:1) -- (6/4*pi:1) -- (7/4*pi:1) -- (2*pi:1);
\end{tikzpicture}
```

但是注意, PGF 的某些个算子、命令、程序库可能依赖初始设置 `trig format=deg`, 因此最好局部地使用 `trig format=rad`.

89.4.5 比较函数与逻辑函数

`equal(x,y)`

`\pgfmathequal{x}{y}`

如果 $x = y$ 则返回 1, 否则返回 0.

1 `\pgfmathparse{equal(20,20)} \pgfmathresult`

`greater(x,y)`

`\pgfmathgreater{x}{y}`

如果 $x > y$ 则返回 1, 否则返回 0.

`less(x,y)`

`\pgfmathless{x}{y}`

如果 $x < y$ 则返回 1, 否则返回 0.

`notequal(x,y)`

`\pgfmathnotequal{x}{y}`

如果 $x \neq y$ 则返回 1, 否则返回 0.

`notgreater(x,y)`

`\pgfmathnotgreater{x}{y}`

如果 $x \leq y$ 则返回 1, 否则返回 0.

`notless(x,y)`

`\pgfmathnotless{x}{y}`

如果 $x \geq y$ 则返回 1, 否则返回 0.

`and(x,y)`

`\pgfmathand{x}{y}`

如果 x 和 y 的解析结果都是非 0 值, 则返回 1, 否则返回 0.

`or(x,y)`

`\pgfmathor{x}{y}`

如果 x 和 y 的解析结果不都是 0 值, 则返回 1, 否则返回 0.

`not(x)`

`\pgfmathnot{x}`

如果 x 的解析结果是 0 值, 则返回 1, 否则返回 0.

`ifthenelse(x,y,z)`

`\pgfmathifthenelse{x}{y}{z}`

如果 x 的解析结果不是 0 值, 则执行 y , 否则执行 z .

`true`

`\pgfmathtrue`

执行结果是 1.

yes

```
\pgfmathparse{true ? "yes" : "no"} \pgfmathresult
```

`false`

`\pgfmathfalse`

执行结果是 0.

89.4.6 伪随机函数

`rnd`

`\pgfmathrnd`

产生一个在 0 到 1 之间的服从均匀分布的伪随机数。

0.54295, 0.02867, 0.93219, 0.86432, 0.30621, 0.7772, 0.4322, 0.02435, 0.48743, 0.26953,

```
\foreach \x in {1,...,10} {\pgfmathparse{rnd}\pgfmathresult, }
```

`rand`

`\pgfmathrand`

产生一个在 -1 到 1 之间的服从均匀分布的伪随机数。

`random(x,y)`

`\pgfmathrandom{x,y}`

这个函数可以采用 3 种形式：

- `random()`, `\pgfmathrandom{}`, 产生一个在 0 到 1 之间的服从均匀分布的伪随机数。
- `random(x)`, 产生一个在 1 到 x 之间的服从均匀分布的伪随机“整数”。
- `random(x,y)`, 产生一个在 x 到 y 之间的服从均匀分布的伪随机“整数”。

89.4.7 基本的转换函数

下面的函数将十进制数转换为二进制数、八进制数、十六进制数，转换结果不能再参与进一步的计算，因为解析器只能对十进制数做计算。

`hex(x)`

`\pgfmathhex{x}`

假定 x 是十进制数，并将它转换为十六进制数，其中的字母使用小写，转换结果不能再参与计算。

fff `\pgfmathparse{hex(65535)} \pgfmathresult`

`Hex(x)`

`\pgfmathHex{x}`

假定 x 是十进制数，并将它转换为十六进制数，其中的字母使用大写，转换结果不能再参与计算。

`oct(x)`

`\pgfmathoct{x}`

假定 x 是十进制数，并将它转换为八进制数，转换结果不能再参与计算。

`bin(x)`

`\pgfmathbin{x}`

假定 x 是十进制数，并将它转换为二进制数，转换结果不能再参与计算。

89.4.8 其它函数

`min(x1,x2,...,xn)`

`\pgfmathmin{x1,x2,...}{... ,xn 1,xn}`

返回某一组数值中的最小值，由于历史的原因，命令 `\pgfmathmin` 的参数要分成两组，每一组的个数任意。

-8.0 `\pgfmathparse{min(3,-2,-8,100)} \pgfmathresult`

```
max(x1,x2,...,xn)
\pgfmathmax{x1,x2,...}{... ,xn 1,xn}
```

返回某一组数值中的最大值。

```
veclen(x,y)
\pgfmathveclen{x}{y}
```

计算 $\sqrt{x^2 + y^2}$ 。

```
array(x,y)
\pgfmatharray{x}{y}
```

这里 x 是个数组, y 是个索引数, 本函数索引数组 x 中编号为 y 的元素, 元素编号从 0 开始。

```
17 \pgfmathparse{array({9,13,17,21},2)} \pgfmathresult
```

```
sinh(x)
\pgfmathsinh{x}
```

计算双曲正弦值。

```
cosh(x)
\pgfmathcosh{x}
```

计算双曲余弦值。

```
tanh(x)
\pgfmathtanh{x}
```

计算双曲正切值。

```
width("x")
\pgfmathwidth{"x"}
```

这里 x 代表能够放在一个 $\text{T}_\text{E}_\text{X}$ 的水平盒子里的内容, 本函数返回该盒子的宽度, 宽度数值的默认单位为 pt , 双引号防止 x 被解析。注意在整个表达式被解析之前, 双引号内的宏会被展开。

```
82.6919 \pgfmathparse{width("Some Lovely Text")} \pgfmathresult
```

```
height("x")
\pgfmathheight{"x"}
```

这里 x 代表能够放在一个 $\text{T}_\text{E}_\text{X}$ 的水平盒子里的内容, 本函数返回该盒子的高度, 高度数值的默认单位为 pt , 双引号防止 x 被解析。注意在整个表达式被解析之前, 双引号内的宏会被展开。

```
depth("x")
\pgfmathdepth{"x"}
```

这里 x 代表能够放在一个 $\text{T}_\text{E}_\text{X}$ 的水平盒子里的内容, 本函数返回该盒子的深度, 深度数值的默认单位为 pt , 双引号防止 x 被解析。注意在整个表达式被解析之前, 双引号内的宏会被展开。

90 其它数学命令

90.1 基本算术函数

`\pgfmathreciprocal{<x>}`

这个命令计算 $\frac{1}{x}$ ，即倒数，当 x 的值很小时，该命令能有较高的精确度。

90.2 比较与逻辑函数

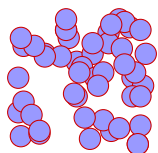
`\pgfmathapproxequalto{<x>}{<y>}`

如果 $\langle x \rangle$ 与 $\langle y \rangle$ 的绝对值之差满足 $|\langle x \rangle - \langle y \rangle| < 0.0001$ ，该命令会把 1.0 保存在 `\pgfmathresult` 中；除了这个情况外，把 0.0 保存在 `\pgfmathresult` 中。该命令还会使得 TeX 的条件判断命令 `\ifpgfmathcomparison` 的真值被相应地设定，这个条件判断命令是全局命令。

0.0	<code>\pgfmathapproxequalto{0.01}{0.002} \pgfmathresult \\\</code>
false	<code>\ifpgfmathcomparison true \else false \fi</code>

`\pgfmathrandominteger{<macro>}{<minimum>}{<maximum>}`

定义宏 `<macro>`，其中保存一个从 `<minimum>`（含）到 `<maximum>`（含）的随机整数。



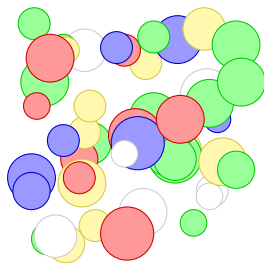
```
\begin{pgfpicture}
\foreach \x in {1,...,50}{
\pgfmathrandominteger{\a}{1}{50}
\pgfmathrandominteger{\b}{1}{50}
\pgfpathcircle{\pgfpoint{+\a pt}{+\b pt}}{+4pt}
\color{blue!40!white}
\pgfsetstrokecolor{red!80!black}
\pgfusepath{stroke, fill}
}
\end{pgfpicture}
```

`\pgfmathdeclarerandomlist{<list name>}{<item-1>}{<item-2>}...`

这个命令创建一个列表，列表的名称是 `<list name>`，注意各个列表项要用花括号括起来，而不是用逗号分隔。该命令一般与下一个命令一起使用。

`\pgfmathrandomitem{<macro>}{<list name>}`

其中的 `<list name>` 是用上一个命令创建的列表名称，`<macro>` 是本命令定义的宏。本命令从 `<list name>` 中随机选择一个列表项，并把选出的列表项保存在宏 `<macro>` 中。



```

\begin{pgfpicture}
  \pgfmathdeclarerandomlist{color}{\red}{\blue}{\green}{\yellow}{\white}}
  \foreach \a in {1,...,50}{
    \pgfmathrandominteger{\x}{1}{85}
    \pgfmathrandominteger{\y}{1}{85}
    \pgfmathrandominteger{\r}{5}{10}
    \pgfmathrandomitem{\c}{color}
    \pgfpathcircle{\pgfpoint{+\x pt}{+\y pt}}{+\r pt}
    \color{\c!40!white}
    \pgfsetstrokecolor{\c!80!black}
    \pgfusepath{stroke, fill}
  }
\end{pgfpicture}

```

`\pgfmathsetseed{<integer>}`

显式地设置随机数生成器的种子。

90.3 基本的进位制转换

目前, PGF 能把 0 到 $2^{31} - 1$ 之间的十进制整数转换为 p 进制数, 这里 $2 \leq p \leq 36$. 如果 $p > 10$, 那么数值中的字母可以使用大写也可以使用小写。

`\pgfmathbasetodec{<macro>}{<number>}{<base>}`

定义宏 <macro>; 将 <number> 看作基数是 <base> 的数, 把它转换为 10 进制数, 保存在宏 <macro> 中。

4223	<code>\pgfmathbasetodec\mynumber{107f}{16} \mynumber</code>
------	---

`\pgfmathdectobase{<macro>}{<number>}{<base>}`

定义宏 <macro>; 将 <number> 看作 10 进制数, 把它转换成基数为 <base> 的数, 其中的字母使用小写, 并保存在宏 <macro> 中。

fff	<code>\pgfmathdectobase\mynumber{65535}{16} \mynumber</code>
-----	--

`\pgfmathdectoBase{<macro>}{<number>}{<base>}`

定义宏 `<macro>`；将 `<number>` 看作 10 进制数，把它转换成基数为 `<base>` 的数，其中的字母使用大写，并保存在宏 `<macro>` 中。

```
FFFF \pgfmathdectoBase\mynumber{65535}{16} \mynumber
```

`\pgfmathbasetobase{<macro>}{<number>}{<base-1>}{<base-2>}`

定义宏 `<macro>`；将 `<number>` 看作基数是 `<base-1>` 的数，把它转换成基数为 `<base-2>` 的数，其中的字母使用小写，并保存在宏 `<macro>` 中。

`\pgfmathbasetoBase{<macro>}{<number>}{<base-1>}{<base-2>}`

定义宏 `<macro>`；将 `<number>` 看作基数是 `<base-1>` 的数，把它转换成基数为 `<base-2>` 的数，其中的字母使用大写，并保存在宏 `<macro>` 中。

`\pgfmathsetbasenumberlength{<integer>}`

进进制转换时，结果值的位数可以用这个命令设置。如果结果值的位数小于 `<integer>`，就用 0 做前缀来补足。

```
00001111 \pgfmathsetbasenumberlength{8}
           \pgfmathdectobase\mynumber{15}{2} \mynumber
```

`\pgfmathtodigitlist{<macro>}{<number>}`

定义宏 `<macro>`；在 `<number>` 的相邻两个位置的数字之间插入逗号，作成 一个列表，并保存在宏 `<macro>` 中。

```
1,1,1,1,0,0 \pgfmathdectobase{\binary}{60}{2}
1            \pgfmathtodigitlist{\digitlist}{\binary}
             \digitlist \par
             \pgfmathparse{{\digitlist}[3]} \pgfmathresult
```

从上面的例子看出，本命令得到的列表并不是“数组”，要想把它做成数组还需要在它外围加上花括号。而下面的例子表明，本命令得到的列表处于一个 `TEX` 分组内，可见尽管数组与 `TEX` 分组都可以是用花括号括起来的内容，但二者实际上是不同的：



```
\pgfmathsetbasenumberlength{8}
\begin{tikzpicture}[x=0.25cm, y=0.25cm]
  \foreach \n [count=\y] in {0, 60, 102, 102, 126, 102, 102, 102, 0}{
    \pgfmathdectobase{\binary}{\n}{2}
    \pgfmathtodigitlist{\digitlist}{\binary}
```

```

\foreach \digit [count=\x, evaluate={\c=\digit*50+15;}] in \digitlist
  \fill [fill=black!\c] (\x, -\y) rectangle ++(1,1);
}
\end{tikzpicture}

```

90.4 角度计算

下面的两个命令必须与 PGF 的内核一起使用才有效，它们都把结果保存在命令 `\pgfmathresult` 中。

`\pgfmathanglebetweenpoints{<p>}{<q>}`

这里 `<p>` 与 `<q>` 是 PGF 命令规定的坐标点，设想一条起始点为 `<p>` 且方向向右的水平射线，一条起始点为 `<p>` 且过点 `<q>` 的射线，两条射线构成一个角，从水平射线到第 2 条射线的（角度制下的）角就是本命令所返回的数值。

```

45.0
\pgfmathanglebetweenpoints
  {\pgfpoint{1cm}{3cm}} {\pgfpoint{2cm}{4cm}}
\pgfmathresult

```

`\pgfmathanglebetweenlines{<p1>}{<q1>}{<p2>}{<q2>}`

这里 `<p1>`, `<q1>`, `<p2>`, `<q2>` 都是 PGF 命令规定的坐标点，设想过点 `<p1>`, `<q1>` 的直线以及过点 `<p2>`, `<q2>` 的直线，从第 1 条直线到第 2 条直线的角度就是本命令所返回的数值。

```

270.0
\pgfmathanglebetweenlines
  {\pgfpoint{1cm}{3cm}}{\pgfpoint{2cm}{4cm}}
  {\pgfpoint{0cm}{1cm}}{\pgfpoint{1cm}{0cm}}
\pgfmathresult

```

91 自定义数学引擎中的函数

可以自定义一个函数，像使用 `add(x,y)`, `\pgfmathadd{x,y}` 那样使用它。这主要用到下面的命令。

`\pgfmathdeclarefunction{<name>}{<number of arguments>}{<code>}`

`\pgfmathdeclarefunction*{<name>}{<number of arguments>}{<code>}`

`<name>` 是自定义函数的名称，其中可以使用字母（大小写皆可）、数字、下划线（下标符号），但是注意不能以数字开头，也不能包含空格。函数名称应当是尚未使用过的名称，如果不确定某个函数名称是否已经被使用，可以使用带星号“*”版本的命令。若重复使用函数名称，则带星号“*”的命令会将函数的定义改写为“新版”。注意最好不要改变预定义的函数，因为在某些命令、程序库中可能会用到它们。最好只用不带星号的命令以避免出错。

`<number of arguments>` 声明函数的参数个数，可以是 0，正整数，或者是省略号“...”，省略号表示函数的参数个数是可变的。PGF 将常数，如 `pi`, `e` 当作是有 0 个参数的函数。如果一个函数的参数个数超过 9 个或者是可变的，就会被特殊处理。

`<code>` 是函数的定义内容，它应当是这样的代码：解析 `<code>` 得到一个结果，去掉结果中的长度单位（如果有的话）后可以保存在命令 `\pgfmathresult` 中。函数的定义最好不要有其它副作用，例如不要改变全局变量。

下面用一个例子来展示本命令的用法。下面的例子定义一个函数 `double`，它将其参数变成原值的 2 倍：

```
\makeatletter
\pgfmathdeclarefunction{double}{1}{
  \begingroup % 开启一个 TeX 分组
    \pgf@x=#1pt\relax
    \multiply\pgf@x by2\relax
    \pgfmathreturn\pgf@x
  \endgroup % 结束 TeX 分组
}
\makeatother
\pgfmathparse{double(44.3)} \pgfmathresult
```

上面的代码输出 88.6。

在上面的代码中，定义函数的 `<code>` 是个 TeX 分组内容。`\pgf@x` 是 PGF 的尺寸寄存器，PGF 的尺寸寄存器还有 `\pgf@y`，整数寄存器有 `\c@pgf@counta`，`\c@pgf@countb` 等等。

命令 `\multiply` 计算乘积。

注意宏 `\pgfmathreturn<tokens>` 之后必须跟上 `\endgroup`。

`\pgfmathreturn<tokens>` 会把展开 `\pgf@x` 后的结果（去掉单位）保存在 `\pgfmathresult` 中，因此这里所用的 `<tokens>` 必须适当，这里使用的是 PGF 尺寸寄存器。

自定义函数的命令还会创建以下两个宏，它们是函数的 PGF 命令版本。

- `\pgfmath<function name>`

例如，对于上面定义的函数 `double`，有相应的 PGF 命令 `\pgfmathdouble`，因此

```
88.6 \pgfmathdouble{44.3} \pgfmathresult
```

这个宏是自定义函数 `<function name>` 的“公共”版，它只是使用 `<function name>` 的一个外在“界面”，实际上并不计算函数值。当用命令 `\pgfmath<function name>{<参数>}` 计算函数值时，该命令的参数会用 `\pgfmathparse` 做处理，其中的单位会被转换为 pt，然后把所有单位去掉，再传递给下面的宏，由下面的宏计算函数值：

- `\pgfmath<function name>@`

例如，对于上面定义的函数 `double`，有相应的 PGF 命令 `\pgfmathdouble@`。

这个宏是自定义函数 `<function name>` 的“个人”版，计算函数值时，为了提高速度，解析器调用“个人”版宏，而不是调用“公共”版宏。定义这个宏的 `<code>` 就是定义函数的 `<code>`。这个宏接受“公共”版宏传递来的（无单位）参数做计算。如果要把参数手工传递给这个宏，则参数必须是不带单位的。这个宏按 `<code>` 执行计算，然后将计算结果（不带单位）保存在 `\pgfmathresult` 中。

88.6

```
\makeatletter
\pgfmathdouble@{44.3}
\makeatother
\pgfmathresult
```

对于自定义的函数，如果其参数个数不超过 9 个，则该函数的“公共”版宏和“个人”版宏都使用通常的定义 \TeX 宏的方式来定义，例如，

```
\def\pgfmathNoArgs{<code>} 定义一个无参数宏
```

```
\def\pgfmathThreeArgs#1#2#3{<code>} 定义一个 3 参数宏
```

如果自定义函数的参数个数超过 9 个，或使用省略号表示参数个数（可变个数），那么函数的“公共”版宏和“个人”版宏都定义为“好像是只有一个参数”的宏。此时使用“公共”版宏的句法是：

```
\pgfmathVariableArgs{1.1,3.5,-1.5,2.6}
```

即只是用逗号分隔各个参数。使用“个人”版宏的句法是：

```
\pgfmathVariableArgs@{{1.1}{3.5}{-1.5}{2.6}}
```

即用花括号把各个参数括起来。

注意，函数 `min`，`max` 的“公共”版宏使用两组花括号。

重定义一个函数使用下面的命令。

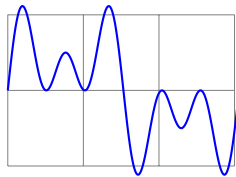
```
\pgfmathredeclarefunction{<function name>}{<algorithm code>}
```

重定义函数 `<function name>` 时，命令会用 `<algorithm code>` 重定义 `\pgfmath<function name>`，但是注意不能改变原来函数的参数个数，而且本命令的有效范围受到花括号分组的限制，因此可以局部地重定义函数。

下面的键（key）用于自定义函数。

```
/pgf/declare function=<function definitions>
```

这个键用来自定义一个局部有效的、不太复杂的函数，先看一个例子：



```
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (3,2);
  \draw [blue, thick, x=0.0085cm, y=1cm,
    declare function={
      sines(\t,\a,\b)=1 + 0.5*(sin(\t)+sin(\t*\a)+sin(\t*\b));
    }]
    plot [domain=0:360, samples=144, smooth] (\x,{sines(\x,3,5)});
\end{tikzpicture}
```

- `<function definitions>` 中定义一个函数所用的格式是:

```
<name>(<arguments>)=<definition>;
```

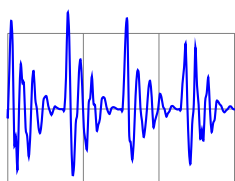
注意其中末尾的分号。其中 `<name>` 是所定义的函数名, `<arguments>` 是该函数的参数列表, `<definition>` 是定义函数的表达式。

- `<arguments>` 是用逗号分隔的“宏”列表, 例如

```
\x, \y
```

注意这里不能定义可变个数的参数。

- `<definition>` 必须是能被数学引擎解析的表达式, 其中应当使用 `<arguments>` 列出的宏。
- `<function definitions>` 中可以定义多个函数, 每个函数定义都以分号结束, 所用函数名在当前环境内不能重复, 后定义的函数可以调用前定义的函数。



```
\begin{tikzpicture}[
  declare function={
    excitation(\t,\w) = sin(\t*\w);
    noise = rnd - 0.5;
    source(\t) = excitation(\t,20) + noise;
    filter(\t) = 1 - abs(sin(mod(\t, 90)));
    speech(\t) = 1 + source(\t)*filter(\t);
  }
]
\draw [help lines] (0,0) grid (3,2);
\draw [blue, thick, x=0.0085cm, y=1cm] (0,1) --
  plot [domain=0:360, samples=144, smooth] (\x,{speech(\x)});
\end{tikzpicture}
```

92 输出数值的格式

输出的数值处在数学模式中, 所以使用某些允许自定义输出格式的选项时, 例如, `frac TeX=<\macro>`, `dec sep=<text>`, `mantissa sep=<text>` 等, 要记住是在数学模式中使用代码。

92.1 基本的命令与选项

```
\pgfmathprintnumber{<x>}
```

```
\pgfmathprintnumber[<options>]{<x>}
```

这是数值输出命令。它使用命令 `\pgfmathfloatparsenumber` 解析实数 $\langle x \rangle$ ，并输出到显示器上。

`\pgfmathprintnumberto{<x>}{<macro>}`

解析实数 $\langle x \rangle$ ，并将它保存在宏 $\langle macro \rangle$ 中，而不是输出到显示器上。

`/pgf/number format/fixed`

这个 key 针对 `\pgfmathprintnumber`，使得该命令输出的数值的小数部分具有固定位数（位数由选项 `precision=` 规定，该选项的初始设置是 2 位），多余位数的小数会被四舍五入，即使用定点小数。

4.57 0 0.1 24,415.98 123,456.12

```
\pgfkeys{/pgf/number format/.cd,fixed,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

上面的例子中使用 `\pgfkeys` 为之后的输出命令设置选项。也可以使用带选项的输出命令：

4.57 `\pgfmathprintnumber [fixed, precision=2] {4.568}`

`/pgf/number format/fixed zerofill={<boolean>}` （默认值 `true`）

该布尔选项决定：当输出数值为定点数且小数部分的位数小于选项 `precision=` 的规定时，是否用 0 来填充。

4.5600 `\pgfmathprintnumber [fixed, fixed zerofill, precision=4] {4.56}`

`/pgf/number format/sci`

这个选项使得输出的数值为科学计数法格式，该格式包括：符号、尾数（mantissa）、幂（exponent，以 10 为底）三部分。注意尾数的整数部分有且只有一位，即个位，个位上的数应当非 0（如果可能的话）。尾数会参照选项 `precision=` 或 `sci precision=` 的规定做舍入。

4.57 · 10⁰ 5 · 10⁻⁴ 1 · 10⁻¹ 2.44 · 10⁴ 1.23 · 10⁵

```
\pgfkeys{/pgf/number format/.cd,sci,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

`/pgf/number format/sci zerofill={<boolean>}` （默认值 `true`）

该布尔选项决定：当输出数值为科学计数法格式且小数部分的位数小于选项 `precision=` 的规定时，是否用 0 来填充。

4.5600 · 10⁰`\pgfmathprintnumber [sci, sci zerofill, precision=4] {4.56}``/pgf/number format/zerofill={<boolean>} (默认值 true)`同时设置 `fixed zerofill` 和 `sci zerofill`.`/pgf/number format/std (无值)``/pgf/number format/std=<lower e> (无默认值)``/pgf/number format/std=<lower e>:<upper e> (无默认值)`

这些选项用来设置输出命令。这些选项的工作机制是：假设输入的数值是 n ，在科学计数法之下 $n = s \cdot m \cdot 10^e$ （这里用 s 代表符号， m 代表尾数， e 代表幂指数），那么，当 $\text{<lower e>} \leq e \leq \text{<upper e>}$ 时，输出的数值为 `fixed` 格式；否则输出的数值为 `sci` 格式。

- 对于选项 `std` 来说，`<lower e>` 等于 $-\frac{p}{2}$ ，这里 p 是由选项 `precision=p` 规定的精度；`<upper e>` 等于 4。即当 $-\frac{p}{2} \leq e \leq 4$ 时，输出的数值为 `fixed` 格式；否则输出的数值为 `sci` 格式。

注意，如果输出命令不使用选项 `fixed` 或 `sci`，则默认使用该选项。

- 对于选项 `std=<lower e>` 来说，`<upper e>` 等于 4。

5 · 10⁻⁴`\pgfmathprintnumber [std, precision=2] {5e-4}`

0.0005

`\pgfmathprintnumber [std, precision=8] {5e-4}`

0.0005

`\pgfmathprintnumber [std=-4, precision=4] {5e-4}`

0.001

`\pgfmathprintnumber [std=-4, precision=3] {5e-4}`5.2 · 10⁵`\pgfmathprintnumber [std=-4:4, precision=1] {51.5e4}`

注意上面后两个例子，根据指定的精度，输出结果有所舍入。

`/pgf/number format/relative*=<exponent base 10>`

注意其中的星号“*”。这个选项设置输出命令。

这里的 `<exponent base 10>` 是个整数。假设 `<exponent base 10> = r`，待输出的数值是 n ，将 n 写成 $n = s \cdot M \cdot 10^r$ ，注意其中的幂指数是 r ， s 代表符号。此时按照选项 `precision=` 规定的精度对 M 做舍入，得到 \overline{M} ，于是 n 变成 $\overline{n} = s \cdot \overline{M} \cdot 10^r$ ，然后参照输出格式选项，例如 `fixed`，`sci`，`std`，输出 \overline{n} 。

1.23457 · 10⁸ 1.23457 · 10⁸`\pgfkeys{/pgf/number format/.cd,relative*={3},precision=0}``\pgfmathprintnumber{123456999}\hspace{1em}``\pgfmathprintnumber{123456999.12}`

在上面的例子中，指定的数量级是 10^3 ，记 $123456999 = 123456.999 \cdot 10^3$ ，精度为 0，即将小数部分向个位做舍入，得到 123457，然后按选项 `std` 的规定输出。

`/pgf/number format/every relative (style)`

这是个样式，该样式的初始设置是：

```
\pgfkeys{/pgf/number format/every relative/.style=std}
```

`/pgf/number format/relative style={<options>}`

等效于 `every relative/.append style={<options>}`.

`/pgf/number format/fixed relative`

本选项设置输出命令，其作用是：假设待输出的数值是 n ，所设置的输出数值的精度是 p ；从左向右考察 n 的各个位置上的数字，首先找出第一个非零数字，记为 w_1 ，记 w_1 右侧的数字为 w_2 ，记 w_2 右侧的数字为 w_3 ，……直到数字 w_p ，然后按“四舍五入”的原则，将 w_p 右侧的数字舍去，得到需要输出的数值 \bar{n} 。

0.0101 `\pgfmathprintnumber[fixed relative,precision=3]{0.010073452}`

在上面例子中，输出精度是 3，待输出数值 0.010073452 的第 1 个非零数字是 1，因此需要保留的是 0.0100，而将 73452 “四舍五入”，于是得到 0.0101。

`/pgf/number format/int detect`

本选项设置输出命令，其作用是：检查待输出的数值是否是整数，如果是整数就将它输出为不带小数点的整数，否则输出为科学计数法格式。

15 20 $2.04 \cdot 10^1$ $1 \cdot 10^{-2}$ 0

```
\pgfkeys{/pgf/number format/.cd,int detect,precision=2}
\pgfmathprintnumber{15}\hspace{1em}
\pgfmathprintnumber{20}\hspace{1em}
\pgfmathprintnumber{20.4}\hspace{1em}
\pgfmathprintnumber{0.01}\hspace{1em}
\pgfmathprintnumber{0}
```

`\pgfmathifisint<number constant>{<true code>}{<false code>}`

这个命令检查待输出数值 `<number constant>` 是否为整数，如果是整数就执行 `<true code>`，否则执行 `<false code>`。

本命令调用 `\pgfmathfloatparsenumber` 来解析 `<number constant>`，解析结果会保存在宏 `\pgfretval` 中。

15 is an int: 15. 15.5 is no int

```
15 \pgfmathifisint{15}{is an int: \pgfretval.}{is no int}\hspace{2em}
15.5 \pgfmathifisint{15.5}{is an int: \pgfretval.}{is no int}
```

`/pgf/number format/int trunc`

将待输出的数值的小数部分去掉，无舍入，只保留整数部分，输出之。

4 0 0 24,415 123,456

```
\pgfkeys{/pgf/number format/.cd,int trunc}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

/pgf/number format/frac

将数值输出为真分数或带分数。真分数、带分数的分数部分的样式,由下面的选项 `frac TeX={<\macro>}` 来规定。

$\frac{1}{3}$ $\frac{1}{2}$ $\frac{16}{75}$ $\frac{3}{25}$ $\frac{2}{75}$ $-\frac{1}{75}$ $\frac{18}{25}$ $\frac{1}{15}$ $\frac{2}{15}$ $-\frac{1}{75}$ $3\frac{1}{3}$ $1\frac{22657}{96620}$ 1 -6

```
\pgfkeys{/pgf/number format/frac}
\pgfmathprintnumber{0.3333333333333333}\hspace{1em}
\pgfmathprintnumber{0.5}\hspace{1em}
\pgfmathprintnumber{2.1333333333333325e-01}\hspace{1em}
\pgfmathprintnumber{0.12}\hspace{1em}
\pgfmathprintnumber{2.6666666666666646e-02}\hspace{1em}
\pgfmathprintnumber{-1.333333333333334e-02}\hspace{1em}
\pgfmathprintnumber{7.200000000000000e-01}\hspace{1em}
\pgfmathprintnumber{6.666666666666667e-02}\hspace{1em}
\pgfmathprintnumber{1.333333333333333e-01}\hspace{1em}
\pgfmathprintnumber{-1.333333333333333e-02}\hspace{1em}
\pgfmathprintnumber{3.333333}\hspace{1em}
\pgfmathprintnumber{1.2345}\hspace{1em}
\pgfmathprintnumber{1}\hspace{1em}
\pgfmathprintnumber{-6}
```

/pgf/number format/frac TeX={<\macro>} (无默认值, 初始值 \frac)

这个选项与上面的选项 `frac` 相配合, 如果要使用这个选项就必须同时使用选项 `frac`. 本选项将输出数值的格式规定为 $\text{T}_{\text{E}}\text{X}$ 宏 `\macro` 所表现的形式, 初始为 `\frac`.

这里 `\macro` 应当是带有两个参数的宏, 待输出的数值在选项 `frac` 的作用下表现为 `a \macro{m}{n}` 的实现形式。如果 `\macro` 是只有一个参数的宏, 那么待输出的数值在选项 `frac` 的作用下表现为 `a \macro{m} n` 的形式。

$$\frac{3}{5}, \frac{3}{5}, \sqrt{35},$$

$$1^3 5, {}_1 3 5, \ln_3 5,$$

```
\pgfmathprintnumber[frac]{.6},~
\pgfmathprintnumber[frac,frac TeX={\dfrac}]{.6},~
\pgfmathprintnumber[frac,frac TeX={\sqrt}]{.6}, \\\[10pt]
\pgfmathprintnumber[frac,frac TeX={^}]{1.6},~
\pgfmathprintnumber[frac,frac TeX={_}]{1.6},~
\pgfmathprintnumber[frac,frac TeX={\ln_}]{.6},
```

`/pgf/number format/frac denom=<int>` (无默认值, 初始值 empty)

这个选项与上面的选项 `frac` 相配合, 如果要使用这个选项就必须同时使用选项 `frac`. 本选项将输出数值的分数部分的分母设为整数 `<int>`.

$$\frac{1}{10} \quad \frac{5}{10} \quad 1\frac{2}{10} \quad -\frac{6}{10} \quad -1\frac{4}{10}$$

```
\pgfkeys{/pgf/number format/.cd,frac, frac denom=10}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{0.5}\hspace{1em}
\pgfmathprintnumber{1.2}\hspace{1em}
\pgfmathprintnumber{-0.6}\hspace{1em}
\pgfmathprintnumber{-1.4}\hspace{1em}
```

注意本选项设置的 `<int>` 最好是 10 的整数倍。

$$\frac{3}{5}, \frac{1}{2},$$

$$\frac{7}{12}, \frac{-12}{-20},$$

```
\pgfmathprintnumber[frac]{.6},~
\pgfmathprintnumber[frac,frac denom=2]{.6},\\\[10pt]
\pgfmathprintnumber[frac,frac denom=12]{.6},~
\pgfmathprintnumber[frac,frac denom=-20,frac TeX={\dfrac}]{.6},
```

`/pgf/number format/frac whole=true|false` (无默认值, 初始值 true)

这个选项与上面的选项 `frac` 相配合, 如果要使用这个选项就必须同时使用选项 `frac`.

当本选项的值为 `true` 时, 其作用是: 如果待输出数值无整数部分, 就将它输出为真分数; 如果待输出数值有整数部分, 就将它输出为带分数。

当本选项的值为 `false` 时, 其作用是: 如果待输出数值无整数部分, 就将它输出为真分数; 如果待输出数值有整数部分, 就将它输出为假分数。

注意本选项的初始值是 `true`.

$$\frac{201}{10} \quad \frac{11}{2} \quad \frac{6}{5} \quad -\frac{28}{5} \quad -\frac{7}{5}$$

```
\pgfkeys{/pgf/number format/.cd,frac, frac whole=false}
\pgfmathprintnumber{20.1}\hspace{1em}
\pgfmathprintnumber{5.5}\hspace{1em}
\pgfmathprintnumber{1.2}\hspace{1em}
\pgfmathprintnumber{-5.6}\hspace{1em}
```

```
\pgfmathprintnumber{-1.4}\hspace{1em}
```

`/pgf/number format/frac shift={<integer>}` (无默认值, 初始值 4)

`/pgf/number format/precision={<number>}`

本选项针对尾数 (mantissa), 设置输出数值的精度, 即保留原数值的小数点后的 <number> 位小数, 将其余小数做四舍五入。如果原数值没有小数点, 或小数部分的位数不足 <number> 位, 输出时本选项不会为它添加小数点或用 0 补足位数。如果原数值有小数点, 但没有小数数字或没有非 0 小数数字, 输出时本选项会把小数部分去掉。

本选项对 `fixed` 格式和 `sci` 格式都有效。

10 10 10 10.01 5·10⁶ 5.01·10⁶

```
\pgfkeys{/pgf/number format/precision=2}
\pgfmathprintnumber{10} \quad
\pgfmathprintnumber{10.} \quad
\pgfmathprintnumber{10.0} \quad
\pgfmathprintnumber{10.005} \quad
\pgfmathprintnumber{5.00e6} \quad
\pgfmathprintnumber{5.005e6}
```

`/pgf/number format/sci precision=<number or empty>` (无默认值, 初始值 `empty`)

本选项针对 `sci` 格式的尾数 (mantissa), 设置其精度, 即保留尾数的小数点后的 <number> 位小数, 将其余小数做四舍五入。对于 `sci` 格式的输出来说, 本选项要比选项 `precision=` 优先。

`/pgf/number format/read comma as period=true|false` (无默认值, 初始值 `false`)

这个选项影响数值解析器的行为。若设置本选项的值是 `true`, 数值解析器在读取待输出数值 (即输入的数值) 时, 会把其中的逗号当作是小数点。如果没有其它相关设置, 在输出数值时, 仍然使用点号作为小数点。

1,234.56

```
\pgfkeys{/pgf/number format/read comma as period}
\pgfmathprintnumber{1234,56}
```

92.2 输出数值的样式以及标点符号

`/pgf/number format/set decimal separator={<text>}`

将 <text> 作为输出数值中的小数点符号, 默认是点号。

1.5, ...

```
\pgfkeysgetvalue{/pgf/number format/set decimal separator}{\aspoint}
1\aspoint 5,\quad \aspoint \aspoint \aspoint
```

`/pgf/number format/dec sep={<text>}`

等效于 `set decimal separator={<text>}`。

/pgf/number format/set thousands separator={<text>}

为了便于读数，对于输出数值的整数部分，可以每隔 3 个数字放置一个分隔符，叫作“千位分隔符”。本选项将 <text> 作为千位分隔符，默认使用逗号。

a,b

```
\pgfkeysgetvalue{/pgf/number format/set thousands separator}\asthsep
a\asthsep b
```

如果要取消千位分隔符，就把 <text> 留空。如果 <text> 是一个逗号“,”，则输出时，逗号“,”后面会有一个“逗号空白”。如果 <text> 是两重花括号括起来的逗号“{,}”，则输出时，逗号“,”后面没有“逗号空白”。比较下面的输出：

```
1,234.56      \pgfmathprintnumber{1234.56} \par
1234.56      \pgfmathprintnumber[set thousands separator={}]{1234.56} \par
1,234.56      \pgfmathprintnumber[set thousands separator={,}]{1234.56} \par
1,234.56      \pgfmathprintnumber[set thousands separator={{,}}]{1234.56}
```

/pgf/number format/1000 sep={<text>}

等效于 `set thousands separator={<text>}`。

/pgf/number format/1000 sep in fractionals={<boolean>} (默认值 true, 初始值 false)

如果这个选项的值是 true，则在输出数值的整数部分和小数部分中都使用“千位分隔符”；如果这个选项的值是 false，则只在输出数值的整数部分中都使用“千位分隔符”。

注意本选项的默认值是 true，初始值是 false。

1.234&123&456&7 · 10³

```
\pgfkeys{/pgf/number format/.cd, std=-2:2,
precision=10, set thousands separator={\&},
1000 sep in fractionals }
\pgfmathprintnumber{1234.1234567}
```

/pgf/number format/min exponent for 1000 sep={<number>} (无默认值, 初始值 0)

这个选项的作用是：假设待输出的数值是 n ，将 n 写成科学计数法形式 $n = s \cdot m \cdot 10^e$ ，当 $e \geq \langle \text{number} \rangle$ 时，才会在输出 n 时使用千位分隔符。

如果 <number> 是 0，则取消该选项；如果 <number> 是负数，则忽略之。

```
1000      \pgfkeys{/pgf/number format/.cd, int detect,
1000 sep={\ } , min exponent for 1000 sep=5}
10000     \pgfmathprintnumber{1000} \par
100 000   \pgfmathprintnumber{10000} \par
           \pgfmathprintnumber{100000}
```

/pgf/number format/use period

这个选项是默认的，即默认小数点用点号“.”，千位分隔符用逗号“,”。

`/pgf/number format/use comma`

这个选项决定：小数点用逗号“,”，千位分隔符用点号“.”，恰好与上一个选项相反。

1.234,56 `\pgfmathprintnumber[use comma]{1234.56}`

`/pgf/number format/skip 0.={<boolean>}` (默认值 true, 初始值 false)

如果这个选项的值是 true，则 0.5 会被输出为 .5。

.56 `\pgfmathprintnumber[skip 0.]{0.56}`

`/pgf/number format/showpos={<boolean>}` (默认值 true, 初始值 false)

如果这个选项的值是 true，则输出非负数时会在数值前面添上正号“+”。

+12.3
+0
-12.3

```
\pgfkeys{/pgf/number format/showpos}
\pgfmathprintnumber{12.3} \par
\pgfmathprintnumber{0} \par
\pgfmathprintnumber{-12.3}
```

`/pgf/number format/print sign={<boolean>}`

与上一个选项 `showpos` 等效。

`/pgf/number format/sci 10e`

这个选项的作用是：输出数值时，按照默认设置或者某些手工设置的选项，如果需要将数值输出为科学计数法格式，那么该数值采用的科学计数法格式是 $s \cdot m \cdot 10^e$ ，这是默认的科学计数法格式。

`/pgf/number format/sci 10^e`

等效于上一个选项 `sci 10e`。

`/pgf/number format/sci e`

这个选项的作用是：输出数值时，按照默认设置或者某些手工设置的选项，如果需要将数值输出为科学计数法格式，那么该数值采用的科学计数法格式是 $s m e \pm r$ 。如 $1.5e-4$ 等于 $1.5 \cdot 10^{-4}$ ； $-1.5e+4$ 等于 $-1.5 \cdot 10^4$ 。

-1.23e+1 `\pgfkeys{/pgf/number format/.cd,sci,sci e}`
`\pgfmathprintnumber{-12.345}`

`/pgf/number format/sci E`

这个选项的作用是：输出数值时，按照默认设置或者某些手工设置的选项，如果需要将数值输出为科学计数法格式，那么该数值采用的科学计数法格式是 $s m E \pm r$ ，与上一个选项类似，只是这里使用大写“E”。

1.23E+1

```
\pgfkeys{/pgf/number format/.cd,sci,sci E}
\pgfmathprintnumber{12.345}
```

/pgf/number format/sci subscript

这个选项的作用是：输出数值时，按照默认设置或者某些手工设置的选项，如果需要将数值输出为科学计数法格式，那么该数值采用的科学计数法格式是 $s m_r$ ，即把幂指数 r 作为尾数的下标。

1.23₁

```
\pgfkeys{/pgf/number format/.cd,sci,sci subscript}
\pgfmathprintnumber{12.345}
```

/pgf/number format/sci superscript

这个选项的作用是：输出数值时，按照默认设置或者某些手工设置的选项，如果需要将数值输出为科学计数法格式，那么该数值采用的科学计数法格式是 $s m^r$ ，即把幂指数 r 作为尾数的上标。

1.23¹

```
\pgfkeys{/pgf/number format/.cd,sci,sci superscript}
\pgfmathprintnumber{12.345}
```

/pgf/number format/sci generic={<keys>}

这个选项用于自定义一种科学计数法格式，其中 <keys> 可以使用以下选项。

/pgf/number format/sci generic/mantissa sep={<text>} （无默认值，初始值 empty）

将 <text> 作为尾数与幂之间的分隔符号，默认的科学计数法格式中使用乘积点 `\cdot`。

/pgf/number format/sci generic/exponent={<text>} （无默认值，初始值 empty）

定义幂的格式，其中可以使用一个变量符号“#1”来表示幂指数。注意这里默认 <text> 处于数学模式中。

$$1.23 \times \text{拾}^1; \quad 1.23 \times \text{拾}^{-4}$$

```
\pgfkeys{/pgf/number format/.cd, sci,
  sci generic={mantissa sep=\times,exponent={\text{拾}^{\#1}}}
\pgfmathprintnumber{12.345}; \quad \pgfmathprintnumber{0.00012345}
```

实际上，在 `sci generic={<keys>}` 的 <keys> 中可以出现 3 个变量：#1 代表幂指数，#2 代表数值的符号，#3 代表尾数。

如果数值是正数，则 #2 的值是 1；如果数值是负数，则 #2 的值是 2。

#3 代表的尾数是未经格式化处理的，例如其中没有千位分隔符。

$$1.23 \text{ 玩一下}_1^{1.23} \text{ 拾}^1; \quad -1.23 \text{ 玩一下}_2^{1.23} \text{ 拾}^{-4}$$

```
\pgfkeys{/pgf/number format/.cd, sci,
  sci generic={mantissa sep={\text{\quad 玩一下}_\#2^{\#3}},exponent={\text{拾}^{\#1}}}
\pgfmathprintnumber{12.345}; \quad \pgfmathprintnumber{-0.00012345}
```


`/pgf/number format/@dec sep mark={<text>}`

将 `<text>` 放在输出数值的小数点位置的左侧，即使数值是没有小数点的整数也会在末尾添加这个 `<text>`，通常用作占位符。

<pre> \makeatletter \pgfkeys{/pgf/number format/@dec sep mark={\&}} \pgfmathprintnumber{12.345} \par \pgfmathprintnumber{12} \makeatother </pre>	<pre> 12&.35 12& </pre>
--	---------------------------------

`/pgf/number format/@sci exponent mark={<text>}`

这个选项针对科学计数法格式的输出数值，将 `<text>` 放在尾数与幂之间的分隔符的左侧，通常用作占位符。

<pre> \makeatletter \pgfkeys{/pgf/number format/@sci exponent mark={\&}} \pgfmathprintnumber[sci]{12.345} \par \pgfmathprintnumber[sci]{1} \makeatother </pre>	<pre> 1.23& · 10¹ 1& · 10⁰ </pre>
--	---

`/pgf/number format/assume math mode={<boolean>}` (默认值 `true`)

`/pgf/number format/verbatim`

用“抄录”形式输出数值，而不是用数学模式输出数值。这个选项会重置 `1000 sep`, `dec sep`, `print sign`, `skip 0.` 等选项的设置，但保留 `precision`, `fixed zerofill`, `sci zerofill`, `fixed`, `sci`, `int detect` 等选项的效果。

对于需要用科学计数法格式输出的数值，本选项还定义了一种比较简洁的格式。

1.23e1; 1.23e-4; 3.27e6

```

\pgfkeys{
  /pgf/fpu,
  /pgf/number format/.cd, sci, verbatim}
\pgfmathprintnumber{12.345};
\pgfmathprintnumber{0.00012345};
\pgfmathparse{exp(15)}
\pgfmathprintnumber{\pgfmathresult}

```

94 基本层 (basic layer) 概略

本节讲解 PGF 的基本层 (basic layer)，它以系统层 (system layer) 为基础而建立。系统层提供了绘图所必须的少量内容，基本层提供了大量命令来直接绘图。与系统层相比，基本层的绘图操作要便捷一些。

但是与前端层 (例如 TikZ) 相比, 基本层的绘图句法用起来也不是那么方便, 因此基本层通常会被其它程序利用。例如, beamer 宏包利用基本层来扩展其功能。

基本层的设计主要包括以下内容:

1. 内核和模块。
2. 能在 $\text{T}_\text{E}\text{X}$ 中使用的绘图命令。
3. 以路径 (path) 为核心的构图方式。
4. 坐标变换矩阵。

94.1 内核和模块

基本层的内核宏包叫作 pgfcore, 提供大量命令。用命令 `\usepgfmodule` 载入模块。

当执行载入命令: `\usepackage{pgf}` 或者 `\inputpgf.tex` 或者 `\usemodule[pgf]` 时, 模块 plot, 模块 shapes, 内核, 以及系统层都会被预先载入。

94.2 基本层的宏

基本层提供大量的宏, 通过这些宏来使用基本层的功能。这些宏都以 `\pgf` 开头, 大部分宏都必须用在 `{pgfpicture}` 环境中。注意 `{tikzpicture}` 环境会开启一个 `{pgfpicture}` 环境, 因此基本层的宏可以在 `{tikzpicture}` 环境中。基本层的宏可以做很多事情, 例如可以用命令使得当前点移动到别的地方, 通过计算确定一个点, 等等。



```
\newdimen\myypos % 定义一个尺寸, 没有给这个尺寸赋值, 默认为 0pt.
\begin{pgfpicture} % 开启 pgf 绘图环境
  \pgfpathmoveto{\pgfpoint{0cm}{\myypos}} % 指定横标为 0cm, 纵标为 \myypos 的一个点, 将
    该点作为当前点并在这个点开启一个路径。
  \pgfpathlineto{\pgfpoint{1cm}{\myypos}} % 指定横标为 1cm, 纵标为 \myypos 的一个点, 将
    当前点移动到该点并在移动时画线段。
  \advance \myypos by 1cm % 将尺寸 \myypos 增加 1cm.
  \pgfpathlineto{\pgfpoint{1cm}{\myypos}}
  \pgfpathclose % 使得路径成为闭合路径
  \pgfusepath{stroke} % 画出刚才构造的路径, 并结束该路径
\end{pgfpicture} % 结束 pgf 绘图环境
```

基本层的命令名称有以下规律:

1. 命令、环境都以 `pgf` 开头。
2. 指定坐标点的命令以 `\pgfpoint` 开头。
3. 开启或者延伸一个路径的命令以 `\pgfpath` 开头。
4. 设置或者修改图形参数的命令以 `\pgfset` 开头。
5. 针对刚刚创建的对象 (如一个路径) 进行操作的命令以 `\pgfuse` 开头。

6. 坐标变换命令以 `\pgftransform` 开头。
7. 与箭头有关的命令以 `\pgfarrows` 开头。
8. “快速” 延展路径或 “快速” 画出路径的命令以 `\pgfpathq` 或 `\pgfusepathq` 开头。
9. 与矩阵有关的命令以 `\pgfmatrix` 开头。

94.3 以路径为核心的构图方式

路径是 PGF 中最重要的概念。所有图形都是由一个或数个路径构成的。对路径可以进行多种操作，如画出 (stroke)，填充颜色 (fill)，带上阴影 (shade)，剪切 (clip) 等等。一个路径可以是闭合的，非闭合的，自交的，或者是由数个不相连的部分构成。

路径先被构建，然后才能被 “使用”。使用以 `\pgfpath` 开头的命令构建路径，每当使用这种命令，当前路径就会被延伸。当使用命令 `\pgfusepath` 时，就会结束刚才构建的路径，并且可以对刚才构建的路径进行某种操作，例如画出、填充、剪切等。

94.4 坐标变换与画布变换

PGF 提供两种变换：坐标变换与画布变换。PGF 本身具有坐标变换矩阵，PDF 和 PostScript 有画布变换矩阵。这两种变换很不一样。当对图形做画布变换时，图形中的一切要素，例如，线条的线宽，文字的笔画都会改变。打比方说，在一个气球上写下文字，画上图画，当气球充气时，文字的笔画、图画中的线条都会有变化。当对图形做坐标变换时，线条的线宽、文字的笔画（文字尺寸）都不会改变。

在默认下，只使用坐标变换。使用命令 `\pgfshowlevel` 可以引入画布变换。

95 层级结构：宏包，环境，子环境，文字

95.1 Overview

PGF 有两种层级结构：一种是关于宏包的，即有的宏包会包含 “子宏包”；另一种是关于绘图的，即绘图环境之内可以有 “子环境”。

95.1.1 宏包的层级结构

PGF 由几个层次构成：

System layer. 这是最低的层，或者可以称之为 “驱动层”、“后台层”，这个层次可以操作 `.dvi` 文件。系统层由宏包 `pgfsys` 实现，它能够按需要自动调用驱动文件。

Basic layer. 基本层由宏包 `pgfcore` 和数个模块组成，用命令 `\usepgfmodule` 载入模块。

Frontend layer. 前端层有多种，例如 `TikZ`，`pgfpict2e`，都是基本层的前端。

每个层次都会自动加载所需要的本层次中的文件。另外作为层次宏包的补充，还有许多程序库宏包，例如有的程序库宏包定义了多种样式的箭头，有的定义了绘图手柄。

95.1.2 图形的层级结构

一个图形可以具有层次，每个层次可以看作一个“组”，图形的各个要素被放入各个组中，可以对同一组中的要素做“统一处理”。例如可以把多个路径放入统一组中，并用红色线条画出它们，然后你可以把红色线条改成蓝色线条——只需改动一个参数。

这里的“组”对应“域”（scope）这个概念。一般情况下，一个域内的参数只在该域内有作用。有多种“开启”、“关闭”一个域的方法，但是这些方法之间可能会有冲突，同时使用多种方法也会造成混乱。我们使用两种域，即“环境”和“ \TeX 分组”，规则如下：

1. PGF 的最外层的域是 `{pgfpicture}` 环境，即图形要在这个环境中画出。一般来说，在 `{pgfpicture}` 环境之外，不能把某个绘图参数设为“全局参数”，例如，在文档开头使用命令 `\pgfsetlinewidth{1pt}` 并不能将所有 `{pgfpicture}` 环境中的线宽设为 1pt，你只能在每个环境中分别设置线宽。
2. 可以在 `{pgfpicture}` 环境中使用一个或数个 `{pgfscope}` 环境，即 `{pgfscope}` 环境是“子环境”。注意 \TeX 分组也是一种域，可以把 `{pgfscope}` 环境放入一个 \TeX 分组内，也可以把一个 \TeX 分组放入 `{pgfscope}` 环境中。凡是涉及“图形状态”（graphic state，见 §95.3.2）的参数都接受 `{pgfscope}` 环境的限制，而不接受 \TeX 分组的限制。有的图形参数（如箭头）、命令（如坐标变换），接受 \TeX 分组的限制。
3. 注意 `{pgfscope}` 环境会自动创建一个 \TeX 分组，因此接受 \TeX 分组的限制的图形参数、命令也会受到 `{pgfscope}` 环境的限制。而且，不能将 `{pgfscope}` 环境与 \TeX 分组交错使用。
4. 如果要对一个路径中的不同部分做不同的坐标变换，应当将各个部分分别放入一个 \TeX 分组内，分别做变换。
5. 命令 `\pgftext` 会创建一个域，这个域是通常的 \TeX 状态，因此各种 \TeX 的命令、模式、环境都可以用作命令 `\pgftext` 的参数，例如，可以把 `{pgfpicture}` 环境或者表格环境用作命令 `\pgftext` 的参数。

遵循以下原则会避免很多令人费解的问题：

- 绘图命令要放入 `{pgfpicture}` 环境中。
- 使用 `{pgfscope}` 环境来明确图形的层次。
- 在绘图环境中不要使用 \TeX 分组，除非要限制坐标变换。

95.2 宏包的层次

95.2.1 内核宏包

```
\usepackage{pgfcore} % LaTeX
\input pgfcore.tex % plain TeX
\usemodule[pgfcore] % ConTeXt
```

这个命令会载入 PGF 的基本层的内核，但不载入任何模块，也不载入任何前端层（如 TikZ），但会把系统层一并载入。使用命令 `\usepgfmodule` 载入模块。

```
\usepackage{pgf} % LaTeX
\input pgf.tex % plain TeX
\usemodule[pgf] % ConTeXt
```

这个命令会载入宏包 `pgfcore`，模块 `shapes` 和模块 `plot`。在 \LaTeX 中，这个命令有两个选项：

```
\usepackage[draft]{pgf}
```

带上这个选项后，所有图形都被方框代替，加快编译速度。

```
\usepackage[version=<version>]{pgf}
```

这个选项指示版本信息，如果 `<version>` 是 0.65，那么会载入很多“兼容命令”。如果 `<version>` 是 0.96，这些“兼容命令”不会被载入。如果不给出版本信息，那么所有版本的命令都会被载入。

95.2.2 模块

```
\usepgfmodule{<module names>}
\usepgfmodule[<module names>]
```

载入内核后，可以用这个命令进一步载入模块。在 `<module names>` 中可以列出多个模块名称，之间用逗号分隔。包裹 `<module names>` 的可以是花括号或者方括号。多次载入同一模块不会有特别的作用。例如

```
\usepgfmodule{matrix,shapes}
```

这个命令实际上会载入文件 `pgfmodule<module>.code.tex`，例如文件 `pgfmodulematrix.code.tex`，因此你可以自己编辑一个这种文件，放在 \TeX 能找到的地方，自己使用。

下面的模块能与 `pgfcore` 配合使用：

- `plot` 模块提供绘图命令，见 §107.
- `shapes` 模块提供绘图形状和 `node`，见 §101.
- `decorations` 模块提供装饰路径，见 §98.
- `matrix` 模块提供命令 `\pgfmatrix`，见 §102.

95.2.3 程序库宏包

程序库与模块的区别在于，程序库提供基本层的“附加”内容，基本层的功能可以作用于这些附加内容；而模块则提供新的功能。例如，程序库 `decoration` 提供多种装饰路径，而模块 `decoration` 则提供装饰功能来操作装饰路径。

```
\usepgflibrary{<list of libraries>}
\usepgflibrary[<list of libraries>]
```

`<list of libraries>` 中列出程序库名称，之间用逗号分隔。例如

```
\usepgflibrary{arrows}
```

这个命令实际上会载入文件 `pgflibrary<library>.code.tex`。

95.3 图形的层级

95.3.1 主要的环境


多数（不是全部）PGF 宏包的命令都要放在环境 `{pgfpicture}` 中。插入外部图形（如 `\pgfuseimage`）、插入阴影（如 `\pgfuseshading`）的命令要放在环境 `{pgfpicture}` 之外。但命令 `\pgfshadepath` 要放在环境内。

```
\begin{pgfpicture}
  <environment contents>
\end{pgfpicture}
```

这个环境会在当前位置插入一个 T_EX 盒子，把该环境构造的图形放入这个盒子中。

边界盒子的尺寸。 盛放图形的盒子的尺寸用这种方式确定：当 PGF 解析 `<environment contents>` 时，会跟踪图形的边界盒子，也就是说，在处理绘图代码的同时，不断刷新图形的边界盒子。边界盒子内只包含绘图命令中“直接”涉及的坐标点，例如画圆的命令所画的圆。

当完成对 `<environment contents>` 的解析后，图形的边界盒子就确定了，此时程序在页面的当前位置创建一个与边界盒子尺寸相同的 T_EX 盒子，把图形放入这个 T_EX 盒子中。

Hello  World!

```
Hello \begin{pgfpicture}
  \pgfpathrectangle{\pgfpointorigin}{\pgfpoint{2ex}{1ex}}
  \pgfusepath{stroke}
\end{pgfpicture} World!
```

如果你希望能控制边界盒子的尺寸，可以使用命令 `\pgfusepath{use as bounding box}`，见 §99.6。

边界盒子的基线。 当一个图形插入到上下文中时，图形的基线通常位于图形的最底部。下面的例子中，两个矩形的长宽一样，尽管画出它们的纵坐标不同，但仍然处于同一水平线上：

Rectangles  and .

```
Rectangles \begin{pgfpicture}
  \pgfpathrectangle{\pgfpointorigin}{\pgfpoint{2ex}{1ex}}
  \pgfusepath{stroke}
\end{pgfpicture} and \begin{pgfpicture}
  \pgfpathrectangle{\pgfpoint{0ex}{1ex}}{\pgfpoint{2ex}{1ex}}
  \pgfusepath{stroke}
\end{pgfpicture}.
```

如果想控制图形基线的位置，可以使用命令 `\pgfsetbaseline`，例如：

Rectangles  and .

```
Rectangles \begin{pgfpicture}
```

```

\pgfpathrectangle{\pgfpointorigin}{\pgfpoint{2ex}{1ex}}
\pgfusepath{stroke}
\pgfsetbaseline{0pt} % 将直线 y=0pt 作为基线
\end{pgfpicture} and \begin{pgfpicture}
\pgfpathrectangle{\pgfpoint{0ex}{1ex}}{\pgfpoint{2ex}{1ex}}
\pgfusepath{stroke}
\pgfsetbaseline{0pt}
\end{pgfpicture}.

```

在图形中插入文字、外部图形。不能在 `{pgfpicture}` 环境中直接插入文字，或者直接使用命令 `\includegraphics` 或 `\pgfimage` 插入外部图形。应当将文字和插入图形的命令作为 `\pgftext` 的参数，将它们插入到 `{pgfpicture}` 环境中，从而添加到图形中。

记住一个图形的位置并在后文中引用。当一个图形编辑完成后，PGF 和 $\text{T}_\text{E}\text{X}$ 会忘记它在页面中的位置。这意味着你不能在后文中引用前面图形中的坐标点，不能在前文图形中的 `node` 与后文图形中的 `node` 之间连线，尽管有时候你想这么做。

为了让 PGF 记住一个图形的位置，需要将 $\text{T}_\text{E}\text{X}$ -if 判断命令 `\ifpgfrememberpicturepositiononpage` 的真值设为 `true`。这个命令一定要用在 `{pgfpicture}` 环境内，且在环境的结尾处，而不是环境的开头处。将这个 $\text{T}_\text{E}\text{X}$ -if 的真值设为 `true`，PGF 就会记住该图形的位置。也可以全局地将这个 $\text{T}_\text{E}\text{X}$ -if 的真值设为 `true`，那么 PGF 就会记住文档中所有图形的位置。

在默认下，这个 $\text{T}_\text{E}\text{X}$ -if 的真值是 `false`，这是因为：第一，有的驱动不支持这一功能，目前这一功能只与 `pdf $\text{T}_\text{E}\text{X}$` 配合工作；第二，当使用这一功能时，需要编译 $\text{T}_\text{E}\text{X}$ 两次才会得到跨图连线的效果，在第一次编译时，会搞乱所有图形的位置；第三，每记住一个图形，就会在 `.aux` 文件中添加一行代码。

尽管有这些不便之处，当文档中的图形比较少时，全局地开启这一功能不会导致 $\text{T}_\text{E}\text{X}$ 变得很慢。

```

\pgfpicture
  <environment contents>
\endpgfpicture

```

这是在 `plain TeX` 中使用的绘图环境，这个版本也会开启一个 $\text{T}_\text{E}\text{X}$ 分组。

```

\startpgfpicture
  <environment contents>
\stoppgfpicture

```

这是在 `ConTeXt` 中使用的绘图环境。

```

\ifpgfrememberpicturepositiononpage

```

这个条件判断命令的真值决定是否让 PGF 记住图形在页面中的位置。注意，这个命令一定要用在 `{pgfpicture}` 环境内，且在环境的结尾处，而不是环境的开头处。使用这个选项后，尽管 PGF 能记住图形在页面中的位置，但图形中的坐标还是不能直接引用，目前只能引用图形中关于 `node` 的位置。

见 §101.3.2, §17.13.

`\pgfsetbaseline{<dimension>}`

在默认下，图形的基线在图形的底部。本命令将直线 $y = \langle \text{dimension} \rangle$ 作为图形的基线，注意 $\langle \text{dimension} \rangle$ 是带单位的尺寸。

`\pgfsetbaselinepointnow{<point>}`

直接指定图形的基线为过点 $\langle \text{point} \rangle$ 的水平线。

`\pgfsetbaselinepointlater{<point>}`

指定过点 $\langle \text{point} \rangle$ 的水平线为图形的基线，不过 $\langle \text{point} \rangle$ 在绘图过程的结尾处确定下来。

Hello ~~world.~~

```

Hello
\begin{pgfpicture}
  \pgfsetbaselinepointlater{\pgfpointanchor{X}{base}} % 注意此时还没有点 X
  \pgfnode{cross out}{center}{world.}{X}{\pgfusepath{stroke}} 创建一个 node, 名称为
  X, 内容是 world., 形状为叉号 cross out, 叉号位于文字的中心 center, 并且画出叉号
\end{pgfpicture}

```

95.3.2 绘图子环境

`\begin{pgfscope}`

`<environment contents>`

`\end{pgfscope}`

该环境内的“图形状态”参数只在该环境内起作用。“图形状态”参数包括以下内容：

- 线宽。
- 实线或虚线等线条样式。
- 线结合、线冠。
- 线条转角处的尖锐程度。
- 画布变换矩阵。
- 剪切路径。

其它类型的参数也会影响图形的外观，但是不属于“图形状态”参数。例如，箭头命令不属于图形状态参数，箭头命令的有效范围受到 TeX 分组的限制。当然，`{pgfscope}` 环境创建一个 TeX 分组。



```

\begin{pgfpicture}
  \begin{pgfscope}
    { % 开启一个 TeX 分组
      \pgfsetlinewidth{2pt} % 图形状态参数
      \pgfpathrectangle{\pgfpointorigin}{\pgfpoint{2ex}{2ex}}
    }
  \end{pgfscope}
\end{pgfpicture}

```



```

    \pgfusepath{stroke} % 画出刚才的矩形
  } % 结束 TeX 分组
  \pgfpathrectangle{\pgfpoint{3ex}{0ex}}{\pgfpoint{2ex}{2ex}}
  \pgfusepath{stroke}
\end{pgfscope}
\pgfpathrectangle{\pgfpoint{6ex}{0ex}}{\pgfpoint{2ex}{2ex}}
\pgfusepath{stroke}
\end{pgfpicture}

```

↗ // /

```

\begin{pgfpicture}
  \begin{pgfscope}
    {
      \pgfsetarrows{-to}
      \pgfpathmoveto{\pgfpointorigin}\pgfpathlineto{\pgfpoint{2ex}{2ex}}
      \pgfusepath{stroke}
    }
    \pgfpathmoveto{\pgfpoint{3ex}{0ex}}\pgfpathlineto{\pgfpoint{5ex}{2ex}}
    \pgfusepath{stroke}
  \end{pgfscope}
  \pgfpathmoveto{\pgfpoint{6ex}{0ex}}\pgfpathlineto{\pgfpoint{8ex}{2ex}}
  \pgfusepath{stroke}
\end{pgfpicture}

```

↗ ↗ ↗

```

\begin{pgfpicture}
  \pgfsetarrows{-Stealth}
  \begin{pgfscope}
    \pgfpathmoveto{\pgfpointorigin}\pgfpathlineto{\pgfpoint{2ex}{2ex}}
    \pgfusepath{stroke}
    {
      \pgfsetarrows{-to}
      \pgfpathmoveto{\pgfpoint{3ex}{0ex}}\pgfpathlineto{\pgfpoint{5ex}{2ex}}
      \pgfusepath{stroke}
    }
  \end{pgfscope}
  \pgfpathmoveto{\pgfpoint{6ex}{0ex}}\pgfpathlineto{\pgfpoint{8ex}{2ex}}
  \pgfusepath{stroke}
\end{pgfpicture}

```

在开启 `{pgfscope}` 环境时,当前路径必须是空的,也就是说,不能在构建路径的过程中开启 `{pgfscope}` 环境。

```
\pgfscope
  <environment contents>
\endpgfscope
```

这是 Plain TeX 的用法。

```
\startpgfscope
  <environment contents>
\stoppgfscope
```

这是 ConTeXt 的用法。

```
\begin{pgfinterruptpath}
  <environment contents>
\end{pgfinterruptpath}
```

在构建路径的过程中插入这个环境,可以把当前路径的构建状态暂时封存起来,待本环境结束后,再调出封存的路径状态,继续构建路径。

本环境不会开启 `{pgfscope}` 环境,也不会调用任何 `\pgfsys` 命令。

```
\pgfinterruptpath
  <environment contents>
\endpgfinterruptpath
```

Plain TeX 中的用法。

```
\startpgfinterruptpath
  <environment contents>
\stoppgfinterruptpath
```

ConTeXt 中的用法。

```
\begin{pgfinterruptpicture}
  <environment contents>
\end{pgfinterruptpicture}
```

这个环境用在 `{pgfpicture}` 环境中,它会暂时打断当前环境,可以在 `{pgfinterruptpicture}` 环境中插入一个新的 `{pgfpicture}` 环境。但这个环境不能直接放到 `{pgfpicture}` 环境中。本环境必须放入一个 TeX 盒子中,然后将该盒子作为命令 `\pgfqbox` 的参数,引入 `{pgfpicture}` 环境中。

```
|
| Sub-picture.
|
```

```
\begin{pgfpicture}
  \pgfpathmoveto{\pgfpoint{0cm}{0cm}}
  \newbox\mybox % 定义一个盒子
```

```

\setbox\mybox=\hbox{ % 设置盒子为一个水平盒子，
\begin{pgfinterruptpicture} % 将这个打断环境放入盒子中
  Sub-\begin{pgfpicture} % 插入一个新的绘图环境
    \pgfpathmoveto{\pgfpoint{1cm}{0cm}}
    \pgfpathlineto{\pgfpoint{1cm}{1cm}}
    \pgfusepath{stroke}
  \end{pgfpicture}-picture. % 结束插入新环境
\end{pgfinterruptpicture} % 结束打断环境
}
\pgfqbox{\mybox} % 引入新定义的盒子
\pgfpathlineto{\pgfpoint{0cm}{1cm}}
\pgfusepath{stroke}
\end{pgfpicture}\hskip3.9cm

```

```

\pgfinterruptpicture
  <environment contents>

```

```

\endpgfinterruptpicture

```

Plain TeX 中的用法。

```

\startpgfinterruptpicture
  <environment contents>

```

```

\stoppgfinterruptpicture

```

ConTeXt 中的用法。

```

\begin{pgfinterruptboundingbox}
  <environment contents>

```

```

\end{pgfinterruptboundingbox}

```

这个环境打断对于图形的边界盒子的计算，将边界盒子的计算状态暂时封存，然后创建一个针对本环境的 <environment contents> 的边界盒子。待本环境结束后，再调出封存的边界盒子的计算状态，继续计算原来的（外层环境的）边界盒子。这样做的效果是，{pgfinterruptboundingbox} 环境所绘的图形会被外层环境的边界盒子忽略。

```

\pgfinterruptboundingbox
  <environment contents>

```

```

\endpgfinterruptboundingbox

```

Plain TeX 中的用法。

```

\startpgfinterruptboundingbox
  <environment contents>

```

```

\stoppgfinterruptboundingbox

```

ConTeXt 中的用法。

95.3.3 插入文字和图形

将文字或者外部图形作为命令 `\pgftext` 的参数，可以插入到 `{pgfpicture}` 环境中。

`\pgftext[<options>]{<text>}`

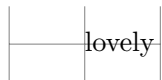
这个命令会开启一个盒子，在这个盒子内部是通常的 $\text{T}_{\text{E}}\text{X}$ 状态，本命令把 `<text>` 放入盒子中，因此 `<text>` 可以是 $\text{T}_{\text{E}}\text{X}$ 状态下的文字、环境、命令等。在默认下，这个盒子的中心位于坐标系的原点位置，可以通过选项来改变盒子的位置。

盒子本身有上 (top)、下 (bottom)、左 (left)、右 (right) 等“部位”，这些“部位”就是盒子边界上的点。如果把盒子中的文字看作是一种注释，那么注释应该有指向的目标点，姑且称之为“指向点”。

`/pgf/text/left`

将盒子的 left 点放在指向点上。

下面例子中，默认文字盒子的指向点是原点，盒子的 left 点放在原点上：



```
\tikz{\draw[help lines] (-1,-.5) grid (1,.5);
\pgftext[left] {lovely}}
```

`/pgf/text/right`

将盒子的 right 点放在指向点上。

`/pgf/text/top`

将盒子的 top 点放在指向点上。

`/pgf/text/bottom`

将盒子的 bottom 点放在指向点上。

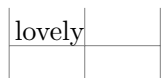
`/pgf/text/base`

将盒子中文字的基线的中点放在指向点上。



```
\tikz{\draw[help lines] (-1,-.5) grid (1,.5);
\pgftext[base] {lovely}}
```

以上选项可以配合使用，例如：



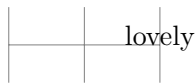
```
\tikz{\draw[help lines] (-1,-.5) grid (1,.5);
\pgftext[bottom,right] {lovely}}
```



```
\tikz{\draw[help lines] (-1,-.5) grid (1,.5);
\pgftext[base,right] {lovely}}
```

`/pgf/text/at=<point>`

指定盒子的指向点为 `<point>`。



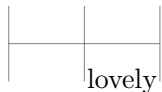
```
\tikz{\draw[help lines] (-1,-.5) grid (1,.5);
\pgftext[base,at={\pgfpoint{1cm}{0cm}}] {lovely}}
```

`/pgf/text/x=<dimension>`

规定盒子的指向点沿着 x 轴方向平移 <dimension>, 可以是负值尺寸。

`/pgf/text/y=<dimension>`

规定盒子的指向点沿着 y 轴方向平移 <dimension>, 可以是负值尺寸。

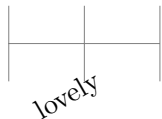


```
\tikz{\draw[help lines] (-1,-.5) grid (1,.5);
\pgftext[at={\pgfpoint{1cm}{0cm}},x=-0.5cm,y=-0.5cm] {lovely}}
```

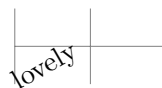
`/pgf/text/rotate=<degree>`

将盒子围绕指向点旋转, 旋转角度由 <degree> 指定。

注意以上选项 `left`, `top`, `x=`, `y=` 等都是平移选项, 选项 `rotate` 是旋转选项, 如果它们的先后次序不同会导致不同的结果, 参考 §25, 比较下面的例子:



```
\tikz{\draw[help lines] (-1,-.5) grid (1,.5);
\pgftext[rotate=30,left,x=-1cm,y=-0.5cm,] {lovely}}
```



```
\tikz{\draw[help lines] (-1,-.5) grid (1,.5);
\pgftext[left,x=-1cm,y=-0.5cm,rotate=30,] {lovely}}
```

95.4 错误信息与警告

96 指定坐标

96.1 Overview

在 `{pgfpicture}` 环境中提供的坐标只在本环境中有效, 一般情况下, 你不能引用页面上的一个绝对位置。使用命令 `\pgfpoint` 指定一个点。

有的命令不必用在 `{pgfpicture}` 环境中, 例如:

3.0pt

5.0pt

```
\newlength{\aaa} \newlength{\bbb} \newlength{\ccc}
\pgfmathsetlength{\aaa}{1pt} \pgfmathsetlength{\bbb}{2pt}
\pgfmathsetlength{\ccc}{3pt}
\pgfpointadd{\pgfpoint{\aaa}{\bbb}}{\pgfpoint{\bbb}{\ccc}}
\makeatletter
\the\pgf@x\ \the\pgf@y
\makeatother
```

96.2 基本的坐标命令

`\pgfpoint{<x coordinate>}{<y coordinate>}`

这个命令产生一个点，<x coordinate> 和 <y coordinate> 都是 TeX 尺寸，可以是带有长度单位量纲的函数表达式，如 $1\text{cm} + 2\text{pt} * \cos(30)$ 。

`\pgfpointorigin`

等效于 `\pgfpoint{0pt}{0pt}`。

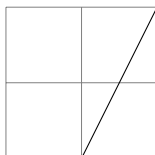
`\pgfpointpolar{<degree>}{<radius>/<y-radius>}`

96.3 XY-坐标系中的坐标

在通常情况下，x 轴的单位向量方向水平向右，长度是 1cm，y 轴的单位向量方向竖直向上，长度是 1cm。可以重设坐标轴的单位向量。

`\pgfpointxy{<sx>}{<sy>}`

该命令确定一个点，这个点在 x 轴的分量等于 <sx> 乘以 x 轴的单位向量，在 y 轴的分量等于 <sy> 乘以 y 轴的单位向量。<sx> 和 <sy> 可以是数学表达式。



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (2,2);
\pgfpathmoveto{\pgfpointxy{1}{0}}
\pgfpathlineto{\pgfpointxy{2}{2}}
\pgfusepath{stroke}
\end{tikzpicture}
```

`\pgfsetxvec{<point>}`

将点 <point> 作为 xyz 坐标系的 x 轴的单位向量。

`\pgfsetyvec{<point>}`

将点 <point> 作为 xyz 坐标系的 y 轴的单位向量。

`\pgfpointpolarxy{<degree>}{<radius>/<y-radius>}`

这个命令产生一个极坐标点，这里 <radius> 和 <y-radius> 都是数值，不是尺寸，是与坐标轴的单位向量相乘的因子。

96.4 三维坐标

`\pgfpointxyz{<sx>}{<sy>}{<sz>}`

这个命令产生一个点，该点的 x 轴分量是 <sx> 与 x 轴单位向量的乘积，该点的 y 轴分量是 <sy> 与 y 轴单位向量的乘积，该点的 z 轴分量是 <sz> 与 z 轴单位向量的乘积。

`\pgfsetzvec{<point>}`

将点 `<point>` 作为 xyz 坐标系的 z 轴的单位向量，这里 `<point>` 是二维点的形式。

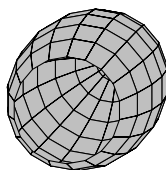
`\pgfpointcylindrical{<degree>}{<radius>}{<height>}`

这个命令产生一个圆柱坐标系点，等效于

```
\pgfpointadd{\pgfpointpolarxy{<degree>}{<radius>}}{\pgfpointxyz{0}{0}{<height>}}
```

`\pgfpointsspherical{<longitude>}{<latitude>}{<radius>}`

这个命令产生一个球坐标系点，`<longitude>` 是经度，`<latitude>` 是纬度，`<radius>` 是半径。



```
\begin{tikzpicture}
  \pgfsetfillcolor{lightgray}
  \foreach \latitude in {-90,-75,...,30}
  {
    \foreach \longitude in {0,20,...,360}
    {
      \pgfpathmoveto{\pgfpointsspherical{\longitude}{\latitude}{1}}
      \pgfpathlineto{\pgfpointsspherical{\longitude+20}{\latitude}{1}}
      \pgfpathlineto{\pgfpointsspherical{\longitude+20}{\latitude+15}{1}}
      \pgfpathlineto{\pgfpointsspherical{\longitude}{\latitude+15}{1}}
      \pgfpathclose
    }
    \pgfusepath{fill,stroke}
  }
\end{tikzpicture}
```

96.5 用已有坐标构建新的坐标

96.5.1 基本的坐标计算

`\pgfpointadd{<v1>}{<v2>}`

本命令得到向量 `<v1>` 与 `<v2>` 的和。

`\pgfpointscale{<factor>}{<coordinate>}`

本命令得到数值 `<factor>` 与向量 `<coordinate>` 的乘积。

`\pgfpointdiff{<start>}{<end>}`

本命令得到向量 `<end>` 减去 `<start>` 的差。

`\pgfpointnormalised{<point>}`

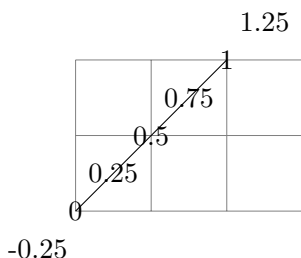
本命令将向量 `<point>` 单位化，即方向不变，长度变成 1pt。如果 `<point>` 是 0 向量或者是长度极短的向量，那么本命令得到的是方向竖直向上，长度是 1pt 的向量。

96.5.2 直线或曲线上的点

设想一个点在直线或曲线上移动，在时刻 $t = 0$ ，该点位于 p ；在时刻 $t = 1$ ，该点位于 q ；在时刻 $t = \frac{1}{2}$ ，该点位于 p 与 q 之间的某个位置，具体位置取决于该点的“速度”变化，详细内容参考关于 Bézier 曲线的资料。

`\pgfpointlineattime{<time t>}{<point p>}{<point q>}`

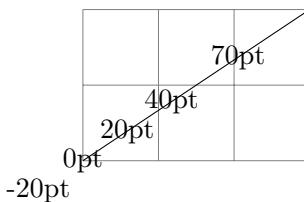
本命令假设 `<point p>` 是时刻 $t = 0$ 时动点所处的位置，`<point q>` 是时刻 $t = 1$ 时动点所处的位置，动点在这两点决定的直线上移动。本命令得到时刻 `<time t>` 时动点所处的位置。如果 `<time t>` 小于 0 或大于 1，则本命令得到的点将处于线段 pq 之外。



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\pgfpathmoveto{\pgfpointorigin}
\pgfpathlineto{\pgfpoint{2cm}{2cm}}
\pgfusepath{stroke}
\foreach \t in {-0.25,0,...,1.25}
{\pgftext[at=\pgfpointlineattime{\t}{\pgfpointorigin}{\pgfpoint{2cm}{2cm}}]{\t}}
\end{tikzpicture}
```

`\pgfpointlineatdistance{<distance>}{<start point>}{<end point>}`

`<distance>` 是带单位的尺寸。点 `<start point>` 与 `<end point>` 决定一个方向，沿着这个方向，与点 `<start point>` 的距离为 `<distance>` 的点就是本命令确定的点。`<distance>` 可以是负值尺寸。



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\pgfpathmoveto{\pgfpointorigin}
```



```

\pgfpathlineto{\pgfpoint{3cm}{2cm}}
\pgfusepath{stroke}
\foreach \d in {-20pt,0pt,20pt,40pt,70pt}
{\pgftext[at=\pgfpointlineatdistance{\d}{\pgfpointorigin}{\pgfpoint{3cm}{2cm}}]{\d
}}
\end{tikzpicture}

```

```

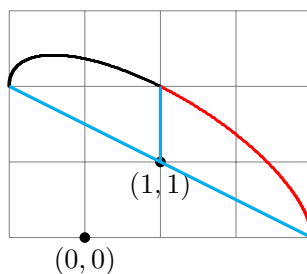
\pgfpointarcaxesattime{<time t>}{<center>}{<0-degree axis>}{<90-degree axis>}{<start
angle>}{<end angle>}

```

设想一个椭圆 E ，初始下， E 的中心在原点，以 $(1,0)$ 为横半轴，以 $(0,1)$ 为纵半轴，以逆时针方向为角度的正方向。以原点为始点，以 $\langle \text{start angle} \rangle$ 为方向的射线 l_1 交椭圆于点 P ；以原点为始点，以 $\langle \text{end angle} \rangle$ 为方向的射线 l_2 交椭圆于点 Q 。假设一个动点在 E 上匀速运动，在时刻 $t=0$ 时，动点位于点 P ，在时刻 $t=1$ 时，动点位于点 Q ，那么在时刻 $\langle \text{time } t \rangle$ 时动点的位置 M 是容易确定的，因为此时的椭圆就是个圆。

对椭圆、射线、各个点做变换：先将原点平移到点 $\langle \text{center} \rangle$ ，然后做一个仿射变换，使得椭圆的“横半轴”： $(1,0)$ 变成向量 $\langle 0\text{-degree axis} \rangle$ ；椭圆的“纵半轴”： $(0,1)$ 变成向量 $\langle 90\text{-degree axis} \rangle$ 。这样得到椭圆 E' ，射线 l'_1, l'_2 ，射线与椭圆的交点 P', Q' ，以及 M' ，这个点 M' 就是本命令确定的点。

在这个变换下，椭圆以 $\langle \text{center} \rangle$ 为中心，以 $\langle 0\text{-degree axis} \rangle$ 为度量角度的起始方向，以自 $\langle 0\text{-degree axis} \rangle$ 旋转到 $\langle 90\text{-degree axis} \rangle$ 的角度为正。



```

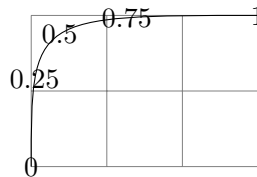
\begin{tikzpicture}
\draw [help lines] (-1,0) grid (3,3);
\fill circle (2pt) node [below] {$(0,0)$};
\fill (1,1) circle (2pt) node [below] {$(1,1)$};
\foreach \t in {0,0.005,...,1}
{
\pgftext[at=\pgfpointarcaxesattime{\t}{\pgfpoint{1cm}{1cm}}{\pgfpoint{-2cm}{1cm}
}{\pgfpoint{0cm}{1cm}}{0}{90}]{.} % 在该位置写一个点号“.”
\pgftext[at=\pgfpointarcaxesattime{\t}{\pgfpoint{1cm}{1cm}}{\pgfpoint{-2cm}{1cm}
}{\pgfpoint{0cm}{1cm}}{90}{180}]{\color{red}.} % 在该位置写一个红色点号“.”
}
\draw [cyan,line width=1.2pt](1,1)---+(0,1) (1,1)---+(-2,1) (1,1)---+(2,-1);

```

```
\end{tikzpicture}
```

```
\pgfpointcurveatime{<time t>}{<point p>}{<point s1>}{<point s2>}{<point q>}
```

设想一个以点 $\langle \text{point } p \rangle$ 为始点，以 $\langle \text{point } q \rangle$ 为终点，以 $\langle \text{point } s1 \rangle$ ， $\langle \text{point } s2 \rangle$ 为控制点的控制曲线，假设一个动点在该曲线上运动，在时刻 $t = 0$ 时，动点位于始点 $\langle \text{point } p \rangle$ ，在时刻 $t = 1$ 时，动点位于终点 $\langle \text{point } q \rangle$ ，那么在时刻 $\langle \text{time } t \rangle$ 时动点的位置 M 就是本命令确定的点。



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathcurveto{\pgfpoint{0cm}{2cm}}{\pgfpoint{0cm}{2cm}}{\pgfpoint{3cm}{2cm}}
  \pgfusepath{stroke}
  \foreach \t in {0,0.25,0.5,0.75,1}
  {\pgftext[at=\pgfpointcurveatime{\t}
    {\pgfpointorigin}
    {\pgfpoint{0cm}{2cm}}
    {\pgfpoint{0cm}{2cm}}
    {\pgfpoint{3cm}{2cm}}]{\t}}
\end{tikzpicture}
```

96.5.3 矩形或椭圆边界上的点

```
\pgfpointborderrectangle{<direction point>}{<corner>}
```

设想一个以原点为中心，以点 $\langle \text{corner} \rangle$ 为右上角的矩形，以及一条以原点为起点，方向为向量 $\langle \text{direction point} \rangle$ 的射线，二者交于点 P ，这个点 P 就是本命令确定的点。

注意向量 $\langle \text{direction point} \rangle$ 的长度应当接近 1pt，如果向量 $\langle \text{direction point} \rangle$ 的长度不是 1pt，那么程序会把它“单位化”，使其长度是 1pt。在“单位化”过程中可能会出现舍入误差，因此向量 $\langle \text{direction point} \rangle$ 的长度越是接近 1pt，舍入误差就越小。

```
\pgfpointborderellipse{<direction point>}{<corner>}
```

设一个以原点为中心，以点 $\langle \text{corner} \rangle$ 为右上角的矩形，在该矩形内有一个内切椭圆，还有一条以原点为起点，方向为向量 $\langle \text{direction point} \rangle$ 的射线，射线与椭圆的交点是 P ，这个点 P 就是本命令确定的点。

96.5.4 两直线的交点

```
\pgfpointintersectionoflines{<p>}{<q>}{<s>}{<t>}
```

设过点 $\langle p \rangle$ 和 $\langle q \rangle$ 的直线，与过点 $\langle s \rangle$ 和 $\langle t \rangle$ 的直线相交，交点是 P ，这个点 P 就是本命令确定的点。

96.5.5 两个圆的交点

`\pgfpointintersectionofcircles<p1>{<p2>}{<r1>}{<r2>}{<solution>}`

本命令假设一个以点 $\langle p1 \rangle$ 为圆心、以 $\langle r1 \rangle$ 为半径的圆，与一个以点 $\langle p2 \rangle$ 为圆心、以 $\langle r2 \rangle$ 为半径的圆相交，本命令确定二者的交点。如果 $\langle solution \rangle$ 是 1，则本命令确定两圆的第一个交点；否则本命令确定两圆的第二个交点。

96.5.6 两个路径的交点

首先调用程序库 `intersections`。

```
\usepgflibrary{intersections}
\usetikzlibrary{intersections}
```

这个程序库计算两个路径的交点。限于 $\text{T}_\text{E}_\text{X}$ 的精度，路径不能太复杂，尤其不要包含很多小线段，如装饰路径。

`\pgfintersectionofpaths<path 1>{<path 2>}`

本命令计算路径 $\langle path 1 \rangle$ 和 $\langle path 2 \rangle$ 的交点，将交点保存起来，可供稍后引用。规定 $\langle path 1 \rangle$ 和 $\langle path 2 \rangle$ 的代码被放入 $\text{T}_\text{E}_\text{X}$ 分组中处理，代码中可以使用坐标变换命令。

`\pgfintersectionsolutions`

使用命令 `\pgfintersectionofpaths` 后，所得到的交点的个数会被保存在这个宏中。

`\pgfpointintersectionsolution{<number>}`

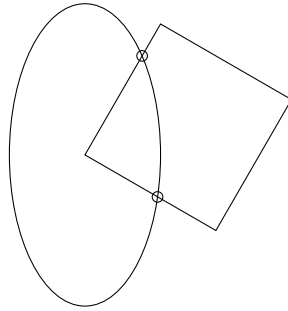
这里 $\langle number \rangle$ 是正整数。使用命令 `\pgfintersectionofpaths` 后，本命令确定第 $\langle number \rangle$ 个交点。如果第 $\langle number \rangle$ 个交点找不到，就返回原点。程序按照交点被找到的次序来规定各个交点的序号，但是寻找交点的算法比较复杂，所得到的交点次序也不易直观理解。

`\pgfintersectionsorthbyfirstpath`

使用命令 `\pgfintersectionofpaths` 后，本命令重排交点的次序，使得交点按照第一个路径的方向排序，即用第一个路径像穿珠子似的将交点串起来。

`\pgfintersectionsorthbysecondpath`

使用命令 `\pgfintersectionofpaths` 后，本命令重排交点的次序，使得交点按照第二个路径的方向排序。



```

\begin{pgfpicture}
  \pgfintersectionofpaths
  {
    \pgfpathellipse{\pgfpointxy{0}{0}}{\pgfpointxy{1}{0}}{\pgfpointxy{0}{2}}
    \pgfgetpath\temppath
    \pgfusepath{stroke}
    \pgfsetpath\temppath
  }{
    \pgftransformrotate{-30}
    \pgfpathrectangle{\pgfpointorigin}{\pgfpointxy{2}{2}}
    \pgfgetpath\temppath
    \pgfusepath{stroke}
    \pgfsetpath\temppath
  }
  \foreach \s in {1,...,\pgfintersectionsolutions}
  {\pgfpathcircle{\pgfpointintersectionsolution{\s}}{2pt}}
  \pgfusepath{stroke}
\end{pgfpicture}

```

96.6 坐标分量

可以得到一个坐标点的 x 分量和 y 分量。

`\pgfextractx{<dimension>}{<point>}`

<dimension> 是个已经声明的 $\text{T}_{\text{E}}\text{X}$ 宏，本命令将点 <point> 的 x 分量，即一个长度尺寸（单位转换为 pt）赋予宏 <dimension>。

注意：(1) <dimension> 必须提前声明；(2) 本命令可以不用在 `{pgfpicture}` 环境中。

56.9055pt

```

\newdimen\mydim
\pgfextractx{\mydim}{\pgfpoint{2cm}{4pt}}
\the\mydim

```

`\pgfextracty{<dimension>}{<point>}`

本命令将点 <point> 的 y 分量，即一个长度尺寸赋予宏 <dimension>，注意事项同上。

`\pgfgetlastxy{<macro for x>}{<macro for y>}`

将本命令之前最近出现的坐标点的 x 分量和 y 分量分别赋予宏 <macro for x> 和 <macro for y>，这两个宏不需要提前声明。

‘56.9055pt’ and ‘113.81102pt’ .

```
\pgfpoint{2cm}{4cm}
\pgfgetlastxy{\macrox}{\macroy}
‘\macrox’ and ‘\macroy’ .
```

96.7 坐标点命令的工作方式

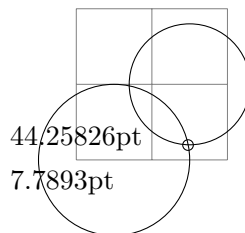
定义一个坐标点的命令，如 `\pgfpoint{1cm}{2pt}`，会把 TeX 尺寸寄存器 `\pgfx` 和 `\pgfy` 分别赋值为 1cm 和 2pt。这两个寄存器是分配给 PGF 的寄存器，参考 §111。以坐标点为参数的命令，例如 `\pgfpathmoveto`，会先确定 `\pgfx` 和 `\pgfy` 的值，然后将画笔移动到指定的坐标位置。

当给定一个坐标点，或者以某一组命令确定一个坐标点后，`\pgfx` 和 `\pgfy` 的值就是当前坐标点的分量值（带单位的尺寸）。例如：

28.45274pt	
2.0pt	

```
\begin{tikzpicture}
  \draw[help lines] (-1,-1) grid (1,1);
  \pgfmoveto{\pgfpoint{1cm}{2pt}}
  \makeatletter
  \pgftext[top]{\parbox{\widthof{\the\pgf@x}}
                {\the\pgf@x \ \ \the\pgf@y}}
  \makeatother
\end{tikzpicture}
```

再如：



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (2,2);
  \draw (0.5,0) circle (1);
  \draw (1.5,1) circle (.8);
  \pgfpathcircle{%
    \pgfpointintersectionofcircles{\pgfpointxy{.5}{0}}{\pgfpointxy{1.5}{1}}{1cm}{0.8cm}
  }{1}
```

```

}
{2pt}
\pgfusepath{stroke}
\makeatletter
\pgftext{\parbox{\widthof{\the\pgf@x}}
          {\the\pgf@x \ \the\pgf@y}}
\makeatother
\end{tikzpicture}
\pgf@process{<code>}

```

这是个使用率很高的内部命令，本命令执行 `<code>` 后，就全局性地给 `\pgfx` 和 `\pgfy` 赋了值（用于确定当前点）。因为当前坐标点是经常变化的，故 `\pgfx` 和 `\pgfy` 的值也是频繁变化的。如果你需要临时使用尺寸寄存器，可以使用 `\pgf@xa`，`\pgf@ya` 等等，参考 §111.

下面是命令 `\pgfpointadd` 的定义（见文件 `pgfcorepoints.code`）：

```

\def\pgfpointadd#1#2{%
  \pgf@process{#1}%
  \pgf@xa=\pgf@x%
  \pgf@ya=\pgf@y%
  \pgf@process{#2}%
  \advance\pgf@x by\pgf@xa%
  \advance\pgf@y by\pgf@ya}

```

97 构建路径

97.1 Overview

在 PGF 中，最基本的绘图概念是“路径”（path），一个路径可以由点、直线段、控制曲线组成，可以包含不相连的数个部分，可以是开的，也可以是闭的。

一个路径可以看作一个“点集”，构建一个路径就是指定一个点集。路径这个概念本身不包含线型、线宽、颜色、点标记类型等等内容，只有用某种标记符号来标记路径上的点时，路径才会显示出来。用“画笔”画出路径的操作是 `stroking` 或者 `drawing`，画笔可以描点，也可以画出线条。用某种颜色填充路径的操作是 `filling`，填充路径时程序会把路径当作是闭的。

在 `pgf` 中，创建路径的命令都以 `\pgfpath` 开头，使用路径的命令都以 `\pgfusepath` 开头。

创建路径的命令会跟踪两个“边界盒子”，一个是当前路径的边界盒子，另一个是所有路径（即整个的当前图形）的边界盒子，具体参考 §97.13.

创建路径的命令会延伸“当前路径”。“当前路径”是一个全局概念，不受 `TEX` 分组的限制。因此在构建一个路径的过程中可以插入 `TEX` 分组来完成某些工作。当遇到命令 `\pgfusepath` 时当前路径才会结束，也有其它结束当前路径的方式，参考 §115.

97.2 Move-To 路径操作

Move-To 算子将当前点移动到指定位置，有的路径命令自带 Move-To 功能，例如 `\pgfpathrectangle`。Move-To 算子常用于路径开头。

`\pgfpathmoveto{<coordinate>}`

本命令将当前点移动到 `<coordinate>` 位置，移动时不画线，其中 `<coordinate>` 可以使用 `\pgfpoint` 等命令指定。一般情况下，在路径的开头应当使用这个命令来开启一个路径创建过程，例如从原点开始创建路径：

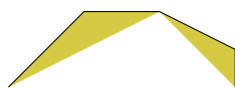
```
\pgfpathmoveto{\pgfpointorigin}
```

如果当前路径为“空”，使用该命令可以开启当前路径的构造。如果在一个路径之中使用该命令，会把路径分为不相连的数个部分（子路径）。

一般情况下，本命令会刷新当前路径和图形的边界盒子。



```
\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{1cm}{1cm}}
  \pgfpathlineto{\pgfpoint{2cm}{1cm}}
  \pgfpathlineto{\pgfpoint{3cm}{0.5cm}}
  \pgfpathlineto{\pgfpoint{3cm}{0cm}}
  \pgfsetfillcolor{yellow!80!black}
  \pgfusepath{fill,stroke}
\end{pgfpicture}
```



```
\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{1cm}{1cm}}
  \pgfpathlineto{\pgfpoint{2cm}{1cm}}
  \pgfpathmoveto{\pgfpoint{2cm}{1cm}}
  \pgfpathlineto{\pgfpoint{3cm}{0.5cm}}
  \pgfpathlineto{\pgfpoint{3cm}{0cm}}
  \pgfsetfillcolor{yellow!80!black}
  \pgfusepath{fill,stroke}
\end{pgfpicture}
```



```

\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{1cm}{1cm}}
  \pgfpathlineto{\pgfpoint{2cm}{1cm}}
  \pgfpathmoveto{\pgfpoint{2cm}{0.5cm}}
  \pgfpathlineto{\pgfpoint{3cm}{0.5cm}}
  \pgfpathlineto{\pgfpoint{3cm}{0cm}}
  \pgfsetfillcolor{yellow!80!black}
  \pgfusepath{fill,stroke}
\end{pgfpicture}

```

比较以上 3 个例子，可见填充一个路径时，程序会把路径的各个不相连部分（子路径）分别看作是“闭的”，对各个子路径分别填充颜色。

97.3 Line-To 路径操作

`\pgfpathlineto{<coordinate>}`

这个命令将当前坐标点移动到 `<coordinate>` 位置，移动时画线，延伸当前路径。

注意本命令的操作需要事先有当前坐标点存在，因此本命令不用于开启一个路径（即不作为一个路径的开头），如果以该命令开头来创建一个路径，而不以其它命令，如 `move-to` 操作开头，`TeX` 不会认为是个错误，但有的阅读器在渲染图形时，可能对此产生一个错误信息。如果以本命令开头构建“路径”，得到的不是真正路径，使用命令 `\pgfusepath{stroke}` 也不能画出这个“路径”，但是这个假“路径”会被算入边界盒子内。类似这个命令，还有一些命令的操作需要事先有当前坐标点存在，这些命令都不能用于开启一个路径。

本命令会先将当前的坐标变换矩阵（如果本命令之前有坐标变换命令的话）用于 `<coordinate>`，然后确定当前点的位置，也就是说，如果本命令之前有坐标变换命令，那么输入的是 `<coordinate>`，得到的是变换 `<coordinate>` 之后的点。

一般情况下，本命令会刷新当前路径和图形的边界盒子。



```

\begin{pgfpicture}
  \pgftransformrotate{-90}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{1cm}{1cm}}
  \pgfpathlineto{\pgfpoint{2cm}{1cm}}
  \pgfsetfillcolor{yellow!80!black}
  \pgfusepath{fill,stroke}
\end{pgfpicture}

```

97.4 Curve-To 路径操作

`\pgfpathcurveto{<support 1>}{<support 2>}{<coordinate>}`

本命令创建一段控制曲线来延伸当前路径。以本命令之前最近出现的点为始点，以 `<coordinate>` 为终点，以 `<support 1>` 和 `<support 2>` 分别为第一和第二控制点，构建一段控制曲线。与 `line-to` 命令一样，本命令一般不用于路径开头。

本命令会先将当前的坐标变换矩阵（如果本命令之前有坐标变换命令的话）用于 `<coordinate>`，然后利用变换后的坐标来构建控制曲线。

一般情况下，本命令会刷新当前路径和图形的边界盒子，注意边界盒子会把代码中出现的起点、终点、控制点都包括在内，这可能会导致边界盒子的边界与控制曲线之间的距离过大。



```
\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathcurveto
    {\pgfpoint{1cm}{1cm}}{\pgfpoint{2cm}{1cm}}
    {\pgfpoint{3cm}{0cm}}
  \pgfsetfillcolor{yellow!80!black}
  \pgfusepath{fill,stroke}
\end{pgfpicture}
```

`\pgfpathquadraticcurveto{<support>}{<coordinate>}`

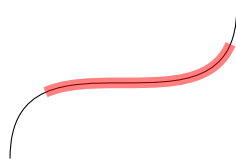
本命令创建一段 2 次 Bézier 曲线来延伸当前路径。以本命令之前最近出现的点为始点，以 `<coordinate>` 为终点，以 `<support>` 为控制点，构建一段 2 次控制曲线。在形式上本命令构造的是 2 次控制曲线，但实际上程序会自动重新选择控制点，把这个 2 次控制曲线转成 3 次控制曲线。而边界盒子会把重新选择的控制点包括在内，未必把 `<support>` 包括在内。

`\pgfpathcurvebetweentime{<time t1>}{<time t2>}{<point p>}{<point s1>}{<point s2>}{<point q>}`

设想一个控制曲线，以 `<point p>` 为始点，以 `<point q>` 为终点，以 `<point s1>` 和 `<point s2>` 分别为第一和第二控制点。有一个动点在这段控制曲线上移动，在时刻 $t = 0$ 该动点位于始点 `<point p>`，在时刻 $t = 1$ 该动点位于终点 `<point q>`。

本命令构建的曲线是动点在时刻 `<time t1>` 到 `<time t2>` 之间的轨迹。

本命令将 `move-to` 操作作为自己的开头，即使得“画笔”移动到始点 `<point p>`，因此本命令可以用在路径的开端处。



```
\begin{tikzpicture}
  \draw [thin] (0,0) .. controls (0,2) and (3,0) .. (3,2);
```

```

\pgfpathcurvebetweentime{0.25}{0.9}{\pgfpointxy{0}{0}}{\pgfpointxy{0}{2}}{\pgfpointxy{3}{0}}{\pgfpointxy{3}{2}}
\pgfsetstrokecolor{red}
\pgfsetstrokeopacity{0.5}
\pgfsetlinewidth{4pt}
\pgfusepath{stroke}
\end{tikzpicture}

```

```

\pgfpathcurvebetweentimecontinue{<time t1>}{<time t2>}{<point p>}{<point s1>}{<point s2>}{<point q>}

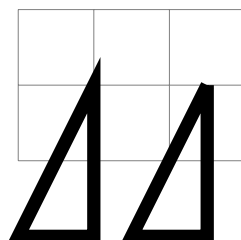
```

类似上一个命令，只是本命令不把 move-to 操作作为自己的开头。

97.5 Close 路径操作

`\pgfpathclose`

这个命令用在路径的某个子路径之后，在该子路径的起止点之间画一个直线段，将这个子路径作成闭合的（即首尾相接），并且首尾连接处看起来比较自然。注意起止点重合不等于“闭合”，二者在外观上有明显的不同，观察下面的例子：



```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\pgfsetlinewidth{5pt}
\pgfpathmoveto{\pgfpoint{1cm}{1cm}}
\pgfpathlineto{\pgfpoint{0cm}{-1cm}}
\pgfpathlineto{\pgfpoint{1cm}{-1cm}}
\pgfpathclose % 将前面的子路径作成闭合的
\pgfpathmoveto{\pgfpoint{2.5cm}{1cm}}
\pgfpathlineto{\pgfpoint{1.5cm}{-1cm}}
\pgfpathlineto{\pgfpoint{2.5cm}{-1cm}}
\pgfpathlineto{\pgfpoint{2.5cm}{1cm}} % 这一段子路径首尾相接，但不是“闭合”的
\pgfusepath{stroke} % 画出路径
\end{tikzpicture}

```

97.6 Arc, Ellipse, Circle 路径操作

`\pgfpatharc{<start angle>}{<end angle>}{<radius> and <y-radius>}`

这个命令以当前坐标点为始点构建一段圆弧或者椭圆弧。如果只给出 `<radius>` 则指示这是圆弧半径。如果还给出 `and <y-radius>` 则指示构建一段椭圆弧。

假设一个具有标准形式 $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ 的椭圆，椭圆的横半轴长度是 `<radius>`，纵半轴长度是 `<y-radius>`，在这个椭圆上取一段弧，弧的起点向量的方向角度是 `<start angle>`，弧的终点向量的方向角度是 `<end angle>`。这里规定：如果 `<start angle>` 小于 `<end angle>`，则按逆时针方向选取弧；如果 `<start angle>` 大于 `<end angle>`，则按顺时针方向选取弧。然后将这段弧平移，使得弧的起点与当前坐标点重合，从而将弧添加到当前路径上。这就是本命令的作用。

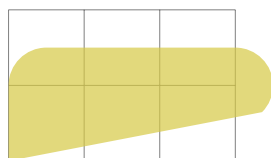
如果本命令之前有旋转变换，本命令先将弧围绕原点旋转，然后再将弧平移，使得弧的起点与当前坐标点重合。这种“绕原点旋转——平移”的效果，等于“平移——绕弧的始点旋转”。

由于本命令需要一个提前给出的点作为弧的始点，所以本命令不能用作路径的开头，也不用作一个孤立的子路径的开头。

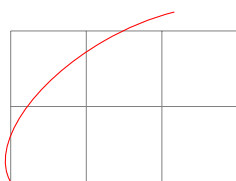
注意命令 `\pgfpatharc{0}{360}{1cm}` 给出的不是一个完整的圆，因为它不闭合，在它后面跟上命令 `\pgfpathclose` 使之闭合。

本命令会刷新边界盒子。

本命令的定义见文件“`pgfcorepathconstruct.code`”。



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{0cm}{1cm}}
  \pgfpatharc{180}{90}{.5cm}
  \pgfpathlineto{\pgfpoint{3cm}{1.5cm}}
  \pgfpatharc{90}{-45}{.5cm}
  \pgfsetfillcolor{yellow!80!black}
  \pgfsetfillopacity{0.7}
  \pgfusepath{fill}
\end{tikzpicture}
```

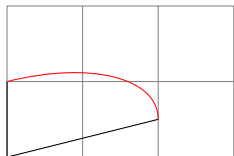


```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \pgftransformrotate{30}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpatharc{180}{60}{2cm and 1cm}
  \pgfsetstrokecolor{red}
  \pgfusepath{draw}
\end{tikzpicture}
```

`\pgfpatharcaxes{<start angle>}{<end angle>}{<first axis>}{<second axis>}`

这个命令与上一个命令类似，以当前坐标点为始点构建一段椭圆弧。这里 `<first axis>` 与 `<second axis>` 都是坐标（向量），用于指定椭圆的横半轴和纵半轴。

假设一个单位圆 $x^2 + y^2 = 1$ ，在这个圆上取一段弧，弧的起点向量的方向角度是 `<start angle>`，弧的终点向量的方向角度是 `<end angle>`。这里规定：如果 `<start angle>` 小于 `<end angle>`，则按逆时针方向选取弧；如果 `<start angle>` 大于 `<end angle>`，则按顺时针方向选取弧。然后做仿射变换，使得：(1,0) 变成向量 `<first axis>`，(0,1) 变成向量 `<second axis>`。这样圆弧就变成了椭圆弧，然后将这段弧平移，使得弧的起点与当前坐标点重合，从而将弧添加到当前路径上。这就是本命令的作用。



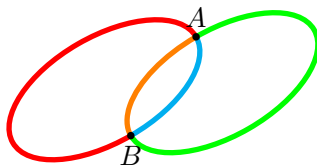
```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (2cm,5mm) (0,0) -- (0cm,1cm);
\pgfpathmoveto{\pgfpoint{2cm}{5mm}}
\pgfpatharcaxes{0}{90}{\pgfpoint{2cm}{5mm}}
{\pgfpoint{0cm}{1cm}}
\pgfsetstrokecolor{red}
\pgfusepath{draw}
\end{tikzpicture}
```

`\pgfpatharcto{<x-radius>}{<y-radius>}{<rotation>} {<large arc flag>}{<counterclockwise flag>}{<target point>}`

这个命令以当前的坐标点为始点，构建一段椭圆弧，延伸当前路径。这里 `<x-radius>` 与 `<y-radius>` 都是带单位的尺寸，`<rotation>` 是个代表角度的数值，`<large arc flag>` 与 `<counterclockwise flag>` 是整数，`<target point>` 是个坐标点。

假设一个标准形式的椭圆 $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ ，它的横半轴长度 a 等于尺寸 `<x-radius>`，纵半轴长度 b 等于尺寸 `<y-radius>`。给定两个点 A, B ，点 A 是当前的坐标点，点 B 是 `<target point>`。

令 θ 等于角度 `<rotation>`，将椭圆旋转 θ 角度，然后再做一个平移变换，我们要求变换后的椭圆经过点 A, B 。如果能满足这个要求，那么一般会有两个（形状一样的）椭圆满足这个要求（两个椭圆可能重合，但仍然看作是两个）。下图展示了这样的两个椭圆：

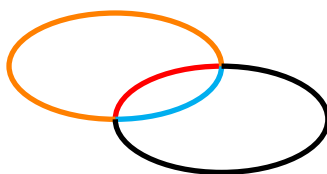


如上图所示，如果要在点 A, B 之间选择一段椭圆弧，那么就有 4 种选择。

本命令所确定的椭圆弧就是以上 4 种之一。如果参数 `<large arc flag>` 是非 0 整数（0 代表 false，非 0 值代表 true），则从当前坐标点到点 `<target point>` 的椭圆弧所张开的角度不小于 $|180^\circ|$ 。如果参数 `<counterclockwise flag>` 是非 0 整数（0 代表 false，非 0 值代表 true），则从当前坐标点到点 `<target`

point> 的椭圆弧是逆时针方向的。

本命令利用椭圆弧所张开的角度（绝对值）和椭圆弧的方向，从 4 个椭圆弧中选取一个弧。



```
\begin{tikzpicture}[scale=2]
  \pgfsetlinewidth{2pt}
  % Flags 0 0: 红色
  \pgfsetstrokecolor{red}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpatharcto{20pt}{10pt}{0}{0}{0}{\pgfpoint{20pt}{10pt}}
  \pgfusepath{stroke}
  % Flags 0 1: 青色
  \pgfsetstrokecolor{cyan}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpatharcto{20pt}{10pt}{0}{0}{1}{\pgfpoint{20pt}{10pt}}
  \pgfusepath{stroke}
  % Flags 1 0: 橙色
  \pgfsetstrokecolor{orange}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpatharcto{20pt}{10pt}{0}{1}{0}{\pgfpoint{20pt}{10pt}}
  \pgfusepath{stroke}
  % Flags 1 1: 黑色
  \pgfsetstrokecolor{black}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpatharcto{20pt}{10pt}{0}{1}{1}{\pgfpoint{20pt}{10pt}}
  \pgfusepath{stroke}
\end{tikzpicture}
```

注意：尽管本命令是个比较实用的工具，但实现本命令的数值计算过程可能不够稳定。椭圆弧的终点未必总是处于点 <target point> 上，有时会偏离数个 pt 距离，这是因为 T_EX 在数值计算方面较弱。

`\pgfpatharctoprecomputed{<center point>}{<start angle>}{<end angle>}{<end point>}{<x-radius>}{<y-radius>}{<ratio x-radius/y-radius>}{<ratio y-radius/x-radius>}`

这个命令构建一段椭圆弧，它的计算速度以及稳定性都较好，但是它的参数比较复杂。

`\pgfpatharctomaxstepsize`

命令 `\pgfpatharctoprecomputed` 使用 B 样条来逼近椭圆弧，逼近的程度由网格宽度 (mesh width) 来刻画，这个宏的值用来调节网格宽度。初始之下，网格宽度的设置是：

```
\def\pgfpatharctomaxstepsize{45}
```

这里设置的网格宽度值采用角度制。网格宽度越小，逼近状态就越好。

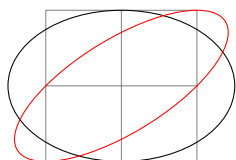
如果定义这个宏的值为空，就取消 3 次样条逼近。

注意赋予本宏的值必须是整数。

```
\pgfpathellipse{<center>}{<first axis>}{<second axis>}
```

本命令构建一个椭圆，它可以开启一个路径，或者延伸当前路径，本命令构建的椭圆是它所在路径的一个孤立部分，即椭圆本身是闭的。

坐标点 <center> 指定椭圆的中心，向量 <first axis> 和 <second axis> 分别指定椭圆的“横半轴向量”和“纵半轴向量”，如果这两个向量不是正交的，则意味着其中有仿射变换。



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (2,2);
\pgfpathellipse{\pgfpoint{1cm}{1cm}}
                {\pgfpoint{1.5cm}{0cm}}
                {\pgfpoint{0cm}{1cm}}
\pgfusepath{draw}
\color{red}
\pgfpathellipse{\pgfpoint{1cm}{1cm}}
                {\pgfpoint{1cm}{1cm}}
                {\pgfpoint{-1cm}{0cm}}
\pgfusepath{draw}
\end{tikzpicture}
```

如果本命令之前有坐标变换命令，那么本命令先按照参数构建椭圆，然后再把坐标变换施加于椭圆上的点，得到变换后的椭圆。

本命令会刷新边界盒子。

```
\pgfpathcircle{<center>}{<radius>}
```

本命令构建一个圆，它可以开启一个路径，或者延伸当前路径，本命令构建的圆是它所在路径的一个孤立部分，即圆本身是闭的。

坐标点 <center> 指定圆心，尺寸 <radius> 指定圆的半径。如果 <radius> 是负值尺寸，则当作正值尺寸看待。

97.7 Rectangle 路径操作

下面两个命令构建矩形，它们可以开启一个路径，或者延伸当前路径，它们构建的矩形是其所在路径的一个孤立部分，即所构建的矩形本身是闭的。

```
\pgfpathrectangle{<corner>}{<diagonal vector>}
```

以点 `<corner>` 和 `<corner> + <diagonal vector>` 为对角顶点确定矩形。

如果本命令之前有坐标变换命令，那么本命令先按参数构建矩形，然后将坐标变换矩阵用于矩形。

本命令刷新边界盒子。

`\pgfpathrectanglecorners{<corner>}{<opposite corner>}`

以点 `<corner>` 和 `<opposite corner>` 为对角顶点确定矩形。

如果本命令之前有坐标变换命令，那么本命令先按参数构建矩形，然后将坐标变换矩阵用于矩形。

本命令刷新边界盒子。

97.8 Grid 路径操作

网格是由平行于坐标轴的直线构成的，并且总是以原点为一个格点。

`\pgfpathgrid[<options>]{<first corner>}{<second corner>}`

本命令将网格添加到当前路径中。网格的范围由以点 `<first corner>` 与 `<second corner>` 为对角顶点的矩形限定。当本命令结束时，当前坐标点是 `<second corner>`。

如果本命令之前有坐标变换命令，那么本命令先按参数构建网格，然后将坐标变换矩阵用于网格。

本命令刷新边界盒子。

`<options>` 中可以使用以下选项：

`/pgf/stepx=<dimension>` (无默认值，初始值 1cm)

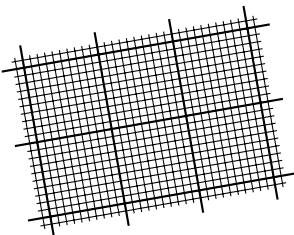
本选项指定网格的水平线之间的间距。

`/pgf/stepy=<dimension>` (无默认值，初始值 1cm)

本选项指定网格的竖直线之间的间距。

`/pgf/step=<vector>`

将向量 `<vector>` 作为一个网格单元的对角向量。



```
\begin{pgfpicture}
  \pgftransformrotate{10}
  \pgfsetlinewidth{0.8pt}
  \pgfpathgrid[step={\pgfpoint{1cm}{1cm}}]
    {\pgfpoint{-3mm}{-3mm}}{\pgfpoint{33mm}{23mm}}
  \pgfusepath{stroke}
  \pgfsetlinewidth{0.4pt}
  \pgfpathgrid[stepx=1mm,stepy=1mm]
    {\pgfpoint{-1.5mm}{-1.5mm}}{\pgfpoint{31.5mm}{21.5mm}}
  \pgfusepath{stroke}
\end{pgfpicture}
```

97.9 Parabola 路径操作

平面上开口向上或向下的抛物线的方程形式是

$$y = a(x - b)^2 + c,$$

它的顶点是 (b, c) ，所以如果指定它的顶点和另外一个点，这个抛物线就确定了。下面的命令就是针对这种情况。

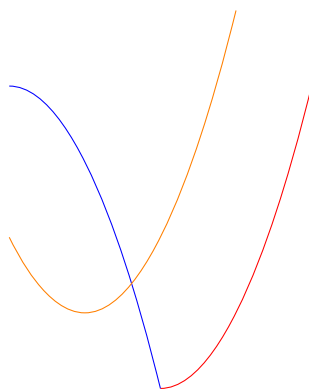
`\pgfpathparabola{<bend vector>}{<end vector>}`

本命令构建两段抛物线，第一段抛物线以当前坐标点为始点，以点 <bend vector> 为顶点以及终点；第二段抛物线以点 <bend vector> 为顶点以及始点，以 <end vector> 为终点。

如果 <end vector> 为空，则只构建第一段抛物线。如果 <bend vector> 为空，则本命令以当前坐标点为始点以及顶点，以 <end vector> 为终点构建一段抛物线。

不能用本命令构建一段不含顶点的抛物线。

本命令接受坐标变换矩阵，也会刷新边界盒子。



```
\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathparabola{\pgfpointorigin}{\pgfpoint{2cm}{4cm}}
  \color{red}
  \pgfusepath{stroke}

  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathparabola{\pgfpoint{-2cm}{4cm}}{\pgfpointorigin}
  \color{blue}
  \pgfusepath{stroke}

  \pgfpathmoveto{\pgfpoint{-2cm}{2cm}}
  \pgfpathparabola{\pgfpoint{1cm}{-1cm}}{\pgfpoint{2cm}{4cm}}
  \color{orange}
  \pgfusepath{stroke}
\end{pgfpicture}
```



```
\end{pgfpicture}
```

97.10 Sine 和 Cosine 路径操作

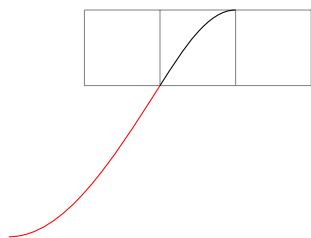
`\pgfpathsine{<vector>}`

本命令在当前坐标点与点 `<vector>` 之间构建一段正弦曲线，延伸当前路径。本命令首先构建一段“标准的”正弦曲线路径：

$$\sin x, \quad x \in [0, \frac{\pi}{2}],$$

然后对这一段正弦曲线做平移变换、伸缩变换、上下对称变换，但不做左右对称变换，也不做旋转变换，使得正弦曲线上原来的点 $(0, \sin 0) = (0, 0)$ 变成当前坐标点，原来的点 $(\frac{\pi}{2}, \sin \frac{\pi}{2}) = (\frac{\pi}{2}, 1)$ 变成点 `<vector>`。

本命令接受坐标变换矩阵，也会刷新边界盒子。



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,1);
  \pgfpathmoveto{\pgfpoint{1cm}{0cm}}
  \pgfpathsine{\pgfpoint{1cm}{1cm}}
  \pgfusepath{stroke}
  \color{red}
  \pgfpathmoveto{\pgfpoint{1cm}{0cm}}
  \pgfpathsine{\pgfpoint{-2cm}{-2cm}}
  \pgfusepath{stroke}
\end{tikzpicture}
```

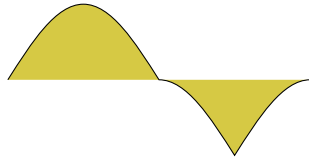
`\pgfpathcosine{<vector>}`

本命令在当前坐标点与点 `<vector>` 之间构建一段余弦曲线，延伸当前路径。本命令首先构建一段“标准的”余弦曲线路径：

$$\cos x, \quad x \in [0, \frac{\pi}{2}],$$

然后对这一段余弦曲线做平移变换、伸缩变换、上下对称变换，但不做左右对称变换，也不做旋转变换，使得余弦曲线上原来的点 $(0, \cos 0) = (0, 1)$ 变成当前坐标点，原来的点 $(\frac{\pi}{2}, \cos \frac{\pi}{2}) = (\frac{\pi}{2}, 0)$ 变成点 `<vector>`。

本命令接受坐标变换矩阵，也会刷新边界盒子。



```
\begin{pgfpicture}
  \pgfpathmoveto{\pgfpoint{0cm}{0cm}}
  \pgfpathsine{\pgfpoint{1cm}{1cm}}
  \pgfpathcosine{\pgfpoint{1cm}{-1cm}}
  \pgfpathcosine{\pgfpoint{1cm}{-1cm}}
  \pgfpathsine{\pgfpoint{1cm}{1cm}}
  \pgfsetfillcolor{yellow!80!black}
  \pgfusepath{fill,stroke}
\end{pgfpicture}
```

97.11 Plot 路径操作

具体参考 §107.

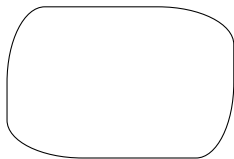
97.12 圆角 (Rounded Corners)

当在一段连续路径内部有“拐角”时，例如，折线段、多边形、先后相继的控制曲线，拐角一般是尖角。可以把尖角改成圆角。

`\pgfsetcornersarced{<point>}`

本命令将其后的所有尖角（不限于当前路径中的尖角）变成圆角。

假设线段或曲线 l 与 r 构成一段连续路径，二者在连接处有一个“尖的拐角”，并且 l 在前， r 在后。将 l 靠近连接点的一部分截去，也把 r 靠近连接点的一部分截去，然后用圆弧连接 l 与 r ，就把尖角变成了圆角。该命令的参数——点 $\langle \text{point} \rangle$ 的 x 分量决定 l 被截去的那一段有多长，点 $\langle \text{point} \rangle$ 的 y 分量决定 r 被截去的那一段有多长，从而决定圆弧的尺寸和形态。



```
\begin{tikzpicture}
  \pgfsetcornersarced{\pgfpoint{5mm}{10mm}}
  \pgfpathrectanglecorners{\pgfpointorigin}
  {\pgfpoint{3cm}{2cm}}
  \pgfusepath{stroke}
\end{tikzpicture}
```

从上面例子看出，矩形是按逆时针方向构建的。

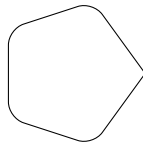


```

\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \pgfsetcornersarced{\pgfpoint{10mm}{5mm}}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{0cm}{2cm}}
  \pgfpathlineto{\pgfpoint{3cm}{2cm}}
  \pgfpathcurveto{\pgfpoint{3cm}{0cm}}
                  {\pgfpoint{2cm}{0cm}}
                  {\pgfpoint{1cm}{0cm}}
  \color{red}
  \pgfusepath{stroke}
\end{tikzpicture}

```

如果点 `<point>` 的 x 分量与 y 分量相同，则本命令构建的圆角是极其接近 90° 的圆弧（误差不大于 $9 \cdot 10^{-6}$ ）。如果点 `<point>` 的 x 分量与 y 分量不相同，则本命令构建的圆角与圆弧的接近程度要差一些。



```

\begin{pgfpicture}
  \pgfsetcornersarced{\pgfpoint{6pt}{6pt}}
  \pgfpathmoveto{\pgfpointpolar{0}{1cm}}
  \pgfpathlineto{\pgfpointpolar{72}{1cm}}
  \pgfpathlineto{\pgfpointpolar{144}{1cm}}
  \pgfpathlineto{\pgfpointpolar{216}{1cm}}
  \pgfpathlineto{\pgfpointpolar{288}{1cm}}
  \pgfpathlineto{\pgfpointpolar{0}{1cm}}
  \pgfusepath{stroke}
\end{pgfpicture}

```

在上面的例子中，表面上看路径是个多边形，但是它并不“封闭”，实际上是个折线段，所以它的起始边与终止边没有构成“角”，因此圆角命令不对这个地方起作用。

使用圆角命令后，如果想改回“尖角”，则可以使用命令

```
\pgfsetcornersarced{\pgfpointorigin}
```

注意，如果构成尖角的边的长度很短，那么使用圆角命令可能造成意外结果。

97.13 跟踪路径或图形的边界盒子

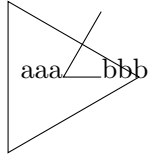
在构建路径时，程序会跟踪当前路径的边界盒子以及整个图形的边界盒子。

```
\pgfresetboundingbox
```

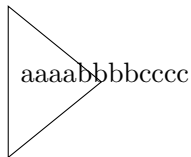
本命令重置图形的边界盒子。本命令使得程序“忘记”已经计算出的图形的边界盒子，并从当下开始，跟踪之后的坐标点，重新计算边界盒子。如果本命令用于绘图环境的末尾，那么程序就当图形没有边界，

并且使得图形的原点位于插入图形的位置。

本命令通常与 `\pgfusepath{use as bounding box}` 配合使用。



```
aaa\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointpolar{0}{0cm}}
  \pgfpathlineto{\pgfpointpolar{150}{2cm}}
  \pgfpathlineto{\pgfpointpolar{210}{2cm}}
  \pgfpathclose
  \pgfusepath{stroke}
  \pgfresetboundingbox % 重设边界盒子
  \pgfpathmoveto{\pgfpointpolar{120}{1cm}}
  \pgfpathlineto{\pgfpointpolar{0}{-1cm}}
  \pgfpathlineto{\pgfpointpolar{0}{-0.5cm}}
  \pgfusepath{stroke}
\end{pgfpicture}bbb
```



```
aaaabbbb\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointpolar{0}{-0.5cm}}
  \pgfpathlineto{\pgfpointpolar{150}{2cm}}
  \pgfpathlineto{\pgfpointpolar{210}{2cm}}
  \pgfpathclose
  \pgfusepath{stroke}
  \pgfresetboundingbox % 重设边界盒子
\end{pgfpicture}cccc
```

图形的边界盒子上的点不能用通常形式的宏来引用。下面的宏可以引用边界盒子上的点，它们都含有符号“@”。

`\pgf@pathminx`

考虑当前路径中涉及的坐标点，这些点的 x 分量的最小值就是这个宏的值。这个宏的初始值是 16000pt。

16000.0pt

```
\tikz;
\makeatletter
\the\pgf@pathminx
\makeatother
```

`\pgf@pathmaxx`

考虑当前路径中涉及的坐标点，这些点的 x 分量的最大值就是这个宏的值。这个宏的初始值是 -16000pt。

-16000.0pt

```

\tikz;
\makeatletter
\the\pgf@pathmaxx
\makeatother

```

\pgf@pathminy

考虑当前路径中涉及的坐标点, 这些点的 y 分量的最小值就是这个宏的值。这个宏的初始值是 16000pt.

\pgf@pathmaxy

考虑当前路径中涉及的坐标点, 这些点的 y 分量的最大值就是这个宏的值。这个宏的初始值是 -16000pt.

\pgf@picminx

考虑当前图形中涉及的坐标点, 这些点的 x 分量的最小值就是这个宏的值。这个宏的初始值是 -16000pt.

\pgf@picmaxx

考虑当前图形中涉及的坐标点, 这些点的 x 分量的最大值就是这个宏的值。这个宏的初始值是 16000pt.

\pgf@picminy

考虑当前图形中涉及的坐标点, 这些点的 y 分量的最小值就是这个宏的值。这个宏的初始值是 -16000pt.

\pgf@picmaxy

考虑当前图形中涉及的坐标点, 这些点的 y 分量的最大值就是这个宏的值。这个宏的初始值是 16000pt.

每当这些构建路径的命令时, 程序就会刷新以上 8 个宏的值, 从而改变边界盒子。如果想手工控制边界盒子的尺寸和位置, 可以使用下面的命令:

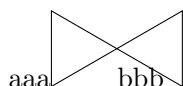
\pgf@protocolsizes{<x-dimension>}{<y-dimension>}

这里 <x-dimension> 和 <y-dimension> 都是尺寸, 二者分别作为横坐标和纵坐标, 决定一个坐标点。本命令使得边界盒子包含这个点。使用该命令时可能需要配合下面的命令。

一个点或者一个路径是否计入边界盒子, 由一个 T_EX-if 条件判断宏来决定: `\ifpgf@relevantforpicturesize`, 只有这个宏的值为 true, 点或者路径才计入边界盒子。或者说这个条件判断宏是个开关, 开启它时, 所涉及的坐标点就计入边界盒子; 关闭它时, 所涉及的坐标点就不计入边界盒子。例如, 当使用剪切路径时, 程序把该条件判断宏的值设为 true, 然后构建起“剪切”作用的路径, 当然这个路径会被计入边界盒子; 程序把该条件判断宏的值设为 false, 然后构建那些“被剪切”的路径, 当然那些路径不计入边界盒子。

\pgf@relevantforpicturesizefalse

这个命令将 T_EX-if 条件判断宏 `\ifpgf@relevantforpicturesize` 的值设为 false, 故此命令之后的路径不计入边界盒子。



```

aaa\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointpolar{0}{0cm}}
  \pgfpathlineto{\pgfpointpolar{150}{1cm}}
  \pgfpathlineto{\pgfpointpolar{210}{1cm}}
  \pgfpathclose
  \pgfusepath{stroke}
\makeatletter
\pgf@relevantforpicturesizefalse
\makeatother % 下面的路径不计入边界盒子
  \pgfpathmoveto{\pgfpointpolar{0}{0cm}}
  \pgfpathlineto{\pgfpointpolar{30}{1cm}}
  \pgfpathlineto{\pgfpointpolar{-30}{1cm}}
  \pgfpathclose
  \pgfusepath{stroke}
\end{pgfpicture}bbb

```

`\pgf@relevantforpicturesizetrue`

这个命令将 TeX-if 条件判断宏 `\ifpgf@relevantforpicturesize` 的值设为 `true`，故此命令之后的路径计入边界盒子。



```

aaa\begin{pgfpicture}
\makeatletter
\pgf@relevantforpicturesizefalse
\makeatother % 下面的路径不计入边界盒子
  \pgfpathmoveto{\pgfpointpolar{0}{0cm}}
  \pgfpathlineto{\pgfpointpolar{150}{1cm}}
  \pgfpathlineto{\pgfpointpolar{210}{1cm}}
  \pgfpathclose
  \pgfusepath{stroke}
\makeatletter
\pgf@relevantforpicturesizetrue
\makeatother % 下面的路径计入边界盒子
  \pgfpathmoveto{\pgfpointpolar{0}{0cm}}
  \pgfpathlineto{\pgfpointpolar{30}{1cm}}
  \pgfpathlineto{\pgfpointpolar{-30}{1cm}}
  \pgfpathclose
  \pgfusepath{stroke}
\end{pgfpicture}bbb

```

98 路径装饰

首先载入模块

```
\usepgfmodule{decorations}
```

这个模块定义了路径装饰功能，所以需要载入它才能装饰路径。各个路径装饰程序库会自动加载这个模块。

98.1 Overview

路径装饰在 §24 中已经介绍过。

98.2 装饰自动化 (Decoration Automata)

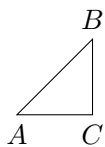
本小节讲解装饰自动化的基本思想。

98.2.1 约定路径名称

为了表述清楚，约定几个名称。

- 对于一个完整路径来说，它的一开始的一段子路径可能不需要装饰，之后的子路径是需要被装饰的。被装饰路径前面的（在当下）不需要装饰的子路径称为“前路径”（preexisting path），注意前路径是没有使用 `\pgfuse` 命令结尾的路径。前路径可以是未被装饰过的，也可以是装饰路径。
- 被装饰的路径称为“输入路径”（input path）。输入路径本身也可能被分成数个“子输入路径”。一般情况下，一个路径的“子路径”指的是被 `moveto` 操作分隔的各个部分，因此子路径是“连续的”。但是在这里划分“子输入路径”的依据是 PGF 计算路径长度的方式。计算某个路径的长度时，先把路径“拆分”为数个子路径，计算每个子路径的长度，然后加起来得到整个路径的长度。“子输入路径”是无需“拆分”就能够直接计算其长度的子路径。

用 `closepath` 操作产生的路径是“子输入路径”，例如下面图形中



```
\tikz \draw (0,0)node[below]{$A$}--
(1,1)node[above]{$B$}--
(1,0)node[below]{$C$}--cycle;
```

有向线段 CA 就是由 `closepath` 操作产生的子输入路径。

文件《pgflibrarydecorations.pathmorphing.code》对装饰样式 `bent` 的定义如下：

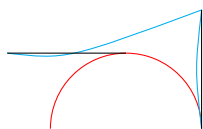
```
\pgfdeclaredecoration{bent}{bent}
{
  \state{bent}[width=+\pgfdecoratedinputsegmentremainingdistance]
  {
    \pgfpathcurveto
    {\pgfqpoint{\pgfdecorationsegmentaspect\
      pgfdecoratedinputsegmentremainingdistance}{\pgfdecorationsegmentamplitude}}
    {\pgfqpointadd{\pgfqpoint{\pgfdecoratedinputsegmentremainingdistance}{0pt}}
```

```

{\pgfpoint{-\pgfdecorationsegmentaspect\
  pgfdecoratedinputsegmentremainingdistance}{\pgfdecorationsegmentamplitude
  }}}
{\pgfpoint{\pgfdecoratedinputsegmentremainingdistance}{0pt}}
}
\state{final}
{}
}

```

由这个定义可知 `bent` 只有一个非空状态，这个状态的片段的宽度就是当前子输入路径的长度，因此，从下面的图形



```

\begin{tikzpicture}
\draw [red] (0,0) arc (0:180:1);
\draw [decorate,decoration=bent][cyan] (0,0) arc (0:180:1);
\draw (0,0)--++(90:{pi/2});
\draw (-1,1)--++(180:{pi/2});
\end{tikzpicture}

```


可以看出来，圆弧 `(0,0) arc (0:180:1)` 有两个子输入路径。用同样的方法可以测试出算子 `circle` 生成的圆有 4 个子输入路径；算子 `rectangle` 生成的矩形有 4 个子输入路径；每个画直线段的符号 “`--`” 创建一个子输入路径；控制曲线算子 “`<p1> .. controls <p2> and <p3> .. <p4>`”（即 `curve-to` 操作）创建一个子输入路径。

一般情况下，子输入路径就是由 `moveto`, `lineto`, `curveto`, `closepath` 操作构建的子路径，不同操作构建不同类型的子输入路径。圆弧、椭圆弧、抛物线、函数曲线都是由一段或数段控制曲线（即 `curve-to` 操作）构建的。由 `move-to` 操作产生的“跳跃”只包含一个点，故其构建的子路径长度是 0。

当装饰自动化过程结束后，输入路径就会被“忘记”，不能再用。

- 起到装饰作用的路径或者其它内容称为“输出”。输出的可能是路径，如程序库 `decorations.pathmorphing` 提供的装饰路径 `zigzag`；输出的也可能是一串符号，如程序库 `decorations.text` 提供的功能。

98.2.2 片段 (segment) 与状态 (state)

有的情况下，装饰路径只是某一个或数个片段 (segment) 的不断重复，如装饰路径 `zigzag` 的片段是：，由下面的代码规定：

```

\pgfpathlineto{\pgfpoint{5pt}{5pt}}
\pgfpathlineto{\pgfpoint{15pt}{-5pt}}
\pgfpathlineto{\pgfpoint{20pt}{0pt}}

```

沿着输入路径重复这一片段就得到装饰路径 `zigzag`。注意这里没有使用 `\pgfpathmoveto{\pgfpointorigin}`，因为这个命令会把一个片段分离为数个子片段，这里只是假定当前坐标点（即片段的起点）是坐标原点。



```

\tikz \draw decorate[decoration=zigzag] {(0,0) -- (2,0)};

```

上面例子中，输入路径是一个 2cm 长的线段，装饰路径是 `zigzag` 类型的，其中一开始有 5 个片段，之后是

一个不完整片段，最后是一段很小的线段。

一个装饰路径可以由一个片段重复构成，也可以由多个不同片段构成。决定什么情况下使用什么片段的规则叫作“状态”(state)，一个状态规定一个片段。装饰自动化实现的就是“有限状态自动化”(finite state automata)。通常，初始状态和终结状态所规定的片段都是必须添加的。初始状态是一个特殊的状态，它规定装饰路径的第一个片段。装饰路径的最后一个片段对应“终结状态”(名称为 final，这是个关键词，终结状态的名称必须使用这个词)，当这个状态完成后，装饰自动化过程就结束了，得到输出路径。

通常，程序根据状态规定，逐步选择片段添加到输入路径上，逐步装饰输入路径。也就是说，在装饰过程中输入路径分为“已装饰部分”和“未装饰部分”，通常程序会计算“未装饰部分”的长度，根据这个长度选择需要添加的片段，每添加一个片段就会刷新这个长度。程序计算“未装饰部分”的长度时会用到一个“参考点”，这个点在输入路径上移动。初始之下参考点与输入路径的始点重合，即初始之下整个输入路径都是“未装饰”的，“未装饰部分”的长度等于整个输入路径的长度。每个片段都被指定一个“宽度”，每添加一个片段，参考点会沿着输入路径移动一次，移动的长度是所添加片段的“宽度”，同时“未装饰部分”的长度也会减去这个片段的“宽度”。

输入路径也可能由数个子输入路径组成，此时对于输入路径的装饰就是分别对子输入路径进行装饰。程序会计算各个子输入路径的长度，加起来得到总的输入路径的长度。在装饰一个子输入路径时，前一个片段的终点与后一个片段的起点之间的联系方式决定于后一片段的定义代码。注意，前后相继的两个片段的“连接点”不是计算未装饰部分长度的“参考点”，这是理解路径装饰过程的关键。

如果添加某个片段会导致参考点超出输入路径，那么就不添加这个片段，而是直接添加终结状态(final)的片段，所以 final 片段的代码中应当包含输入路径的终点。完成 final 片段后，参考点就与输入路径的终点重合，输入路径的“未装饰部分”的长度为 0。

打比方说，一条公路可以看作是输入路径，路边的路灯投下来的光看作是片段，这些片段(光)可能有重叠，也可能没有重叠。如果你注意到路灯间距与灯光照射的宽度之间的区别，就容易理解这个比方。两个路灯的光是否重叠决定于路灯间距和路灯本身的构造，但是不管路灯本身的构造如何，总可以调整路灯间距来使得相邻路灯的光有重叠或无重叠。当然(根据各种情况)路灯间距也可能不是固定的。在判断公路是否被路灯“装饰”完毕时，一般是考虑路灯间距、路灯数量、公路长度三者的关系。这里路灯间距就相当于指定的片段的“宽度”。

假设输入路径是 p ，按 p 的路径方向，有 p 上的点： P_0, P_1, \dots, P_n ，其中 P_0 是 p 的起点， P_n 是 p 的终点。假设给 p 添加的装饰片段依次是 S_1, \dots, S_n ，其中 S_1 是初始片段， S_n 是终结片段。一开始，参考点位于 P_0 ，在创建片段 S_1 时，程序会开启一个“片段坐标系”，在片段坐标系中绘制出片段 S_1 ，这个坐标系的原点就是参考点 P_0 ，故 S_1 以 P_0 为“当前参考点”。当添加初始片段 S_1 后，参考点移动到 P_1 ，从 P_0 到 P_1 的路径长度就是“指定的 S_1 的宽度”， P_1 是创建 S_2 时的“当前参考点”。当添加片段 S_2 后，参考点位于 P_2 ，从 P_1 到 P_2 的路径长度就是“指定的 S_2 的宽度”……如此等等，注意 P_{n-1} 是创建 S_n 时的“当前参考点”。

98.3 自定义装饰路径

有多个装饰程序库，它们提供了多种装饰路径，见 §48，这些是预定义的装饰路径。你也可以自定义一种装饰路径，下面介绍自定义装饰路径的使用的方法，以此展示装饰自动化过程。

```
\pgfdeclaredecoration{<name>}{<initial state>}{<states>}
```

这个命令用于自定义一种装饰路径。`<name>` 是自定义装饰路径的名称。`<initial state>` 是初始状态的名称，规定装饰路径的第一个片段。`<states>` 是一个或数个“状态”，决定在什么情况下使用什么片段。通常，这些状态都会根据输入路径的“未装饰部分”的长度来规定所添加的片段。

由于 $\text{T}_\text{E}_\text{X}$ 在数学计算方面的能力较弱，关于路径的计算精度并不很高。除了输入路径是水平或竖直的状态外，对装饰路径的计算精度都不高。

在 `<states>` 中的各个状态需要使用命令 `\state` 来定义。

`\state{<name>}[<options>]{<code>}`

`<name>` 是状态的名称。当装饰自动化读取状态 `<name>` 时，会产生以下动作：

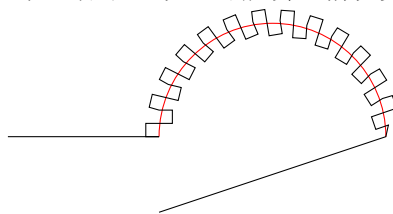
1. 解析 `<options>`。 `<options>` 中的选项可能会导致状态切换，即从当前状态切换到其它状态。若某个选项导致切换状态，则这个选项之后的内容不会被执行。`<options>` 中的选项按次序逐个被执行，即前一个选项的作用实现后，再执行下一个选项。
2. 代码 `<code>` 是一段绘图代码，即本状态规定的片段图形。作为绘图代码，`<code>` 有自己的坐标系（片段坐标系），这个坐标系的默认构建方式如下。

`<code>` 会被放入一个 $\text{T}_\text{E}_\text{X}$ 分组内执行，在这个分组内使用坐标变换，使得当前的参考点是其坐标系的原点；当读取第一个片段时，当前参考点当然就是输入路径的起点；前面已经提到，参考点 P_{n-1} 是创建片段 S_n 时的“当前参考点”，也就是说，参考点 P_{n-1} 是片段 S_n 的 `<code>` 坐标系的原点。

假设 S_i 与 S_j 是前后相连的两个片段，添加 S_i 后，输入路径上的“参考点”（见前文）变成 P_i ，输入路径曲线在 P_i 处有切向量 l_i ，那么 S_j 的 `<code>` 坐标系会被围绕它的原点（即当前参考点 P_i ）旋转，使其 x 轴的方向是切向量 l_i 的方向，而 y 轴方向在 x 轴方向的左侧，即 y 轴与 x 轴成右手系，这个坐标系类似曲线的“自然坐标系”的概念。

片段 S_i 的终点与 S_j 的起点之间的联系方式，决定于片段 S_j 的 `<code>` 的第一个路径命令，如果它的第一个路径命令是 `line-to` 操作，则用直线段连接两个点；如果它的第一个路径命令是 `move-to` 操作，则片段 S_i 与 S_j 之间有间断；如果它的第一个路径命令是 `curve-to` 操作，则用曲线连接两个点。

通常，你需要在 `<options>` 中使用选项 `width=<指定宽度>` 为片段规定一个“宽度”，即本片段所装饰的输入路径的长度。片段 `<code>` 作为一个图形有自己的固有宽度，其固有宽度不必与 `<指定宽度>` 一致，但如果二者不一致，可能会让情况变得复杂。比较下面两个图形：



```
\pgfdeclaredecoration{example}{initial}
```

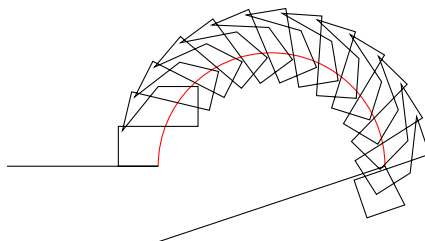
```
{
```

```
\state{initial}[width=10pt] % 选项 width 的作用是，指定片段的装饰长度，若输入路径的未装饰部分的长度小于 10pt，则切换到状态 final，本状态的片段的固有宽度正是 10pt.
```

```

{
  \pgfpathlineto{\pgfpoint{0pt}{5pt}}
  \pgfpathlineto{\pgfpoint{5pt}{5pt}}
  \pgfpathlineto{\pgfpoint{5pt}{-5pt}}
  \pgfpathlineto{\pgfpoint{10pt}{-5pt}}
  \pgfpathlineto{\pgfpoint{10pt}{0pt}}
}
\state{final} % 状态 final 的片段是连接到圆弧终点的线段
{
  \pgfpathlineto{\pgfpointdecoratedpathlast}
}
}
\tikz[decoration=example]
{
  \draw [red](2,0) arc(180:0:1.5cm);
  \draw (0,0) -- (2,0) decorate {arc(180:0:1.5cm)} -- (2,-1);
}

```



```


\pgfdeclaredecoration{example}{initial}
{
  \state{initial}[width=10pt] % 本状态的片段的固有宽度是 30pt，与选项 width 指
    定的宽度不一致。
  {
    \pgfpathlineto{\pgfpoint{0pt}{15pt}}
    \pgfpathlineto{\pgfpoint{15pt}{15pt}}
    \pgfpathlineto{\pgfpoint{15pt}{-15pt}}
    \pgfpathlineto{\pgfpoint{30pt}{-15pt}}
    \pgfpathlineto{\pgfpoint{30pt}{0pt}}
  }
  \state{final} % 状态 final 的片段是连接到圆弧终点的线段
  {
    \pgfpathlineto{\pgfpointdecoratedpathlast}
  }
}

```

```

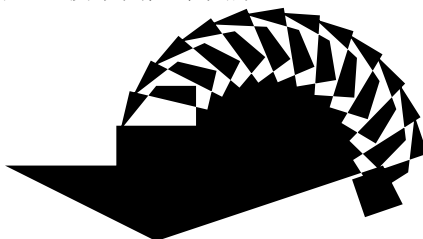
\tikz[decoration=example]
{
  \draw [red] (2,0) arc(180:0:1.5cm);
  \draw (0,0) -- (2,0) decorate {arc(180:0:1.5cm)} -- (2,-1);
}

```

上面例子中定义的装饰路径名称是 `example`，初始状态名称 `initial` 是个关键词。第 1 个图形中的初始状态定义的片段是：「」，输入路径是一个半圆弧。上面第 2 个图形中的初始片段的尺寸是第 1 个图形中的初始片段的尺寸的 3 倍，显然第 2 个图形中的初始片段已经严重走样（除了第一个片段），并且倒数第 2 个片段的终点还超出了输入路径的终点（最后一个片段，即 `final` 状态的片段是连接到圆弧终点的线段）。

注意，在装饰自动化过程中，前路径是始终被“记住”的，故添加到输入路径上的片段实际上是前路径的延伸。因此，如果在 `<code>` 中使用命令 `\pgfusepath{fill}`，那么被填充的路径就包括原来的前路径，以及延伸到当前的片段（装饰路径）。

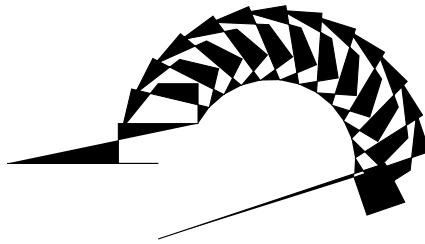
下面的图形是上面图形的修改，比较下面几个图形：



```

\pgfdeclaredecoration{example}{initial}
{
  \state{initial}[width=10pt]
  {
    \pgfpathlineto{\pgfpoint{0pt}{15pt}}
    \pgfpathlineto{\pgfpoint{15pt}{15pt}}
    \pgfpathlineto{\pgfpoint{15pt}{-15pt}}
    \pgfpathlineto{\pgfpoint{30pt}{-15pt}}
    \pgfpathlineto{\pgfpoint{30pt}{0pt}}
  }
  \state{final}
  {
    \pgfpathlineto{\pgfpointdecoratedpathlast}
  }
}
\tikz[decoration=example] % 使用命令 \fill, 看一下填充颜色的效果
{
  \fill (0,0) -- (2,0) decorate {arc(180:0:1.5cm)} -- (2,-1);
}

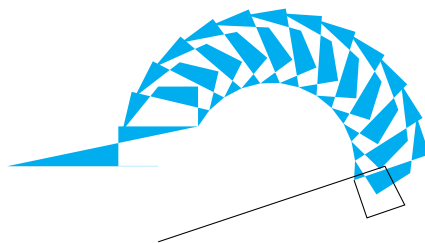
```



```

\pgfdeclaredecoration{example}{initial}
{
  \state{initial}[width=10pt]
  {
    \pgfpathlineto{\pgfpoint{0pt}{15pt}}
    \pgfpathlineto{\pgfpoint{15pt}{15pt}}
    \pgfpathlineto{\pgfpoint{15pt}{-15pt}}
    \pgfpathmoveto{\pgfpoint{15pt}{-15pt}} % 这里使用了 move-to 操作，将片段
      分离开来
    \pgfpathlineto{\pgfpoint{30pt}{-15pt}}
    \pgfpathlineto{\pgfpoint{30pt}{0pt}}
  }
  \state{final}
  {
    \pgfpathlineto{\pgfpointdecoratedpathlast}
  }
}
\tikz[decoration=example]
{
  \filldraw (0,0) -- (2,0) decorate {arc(180:0:1.5cm)} -- (2,-1);
}

```



```

\pgfdeclaredecoration{example}{initial}
{
  \state{initial}[width=10pt]
  {
    \pgfpathlineto{\pgfpoint{0pt}{15pt}}
    \pgfpathlineto{\pgfpoint{15pt}{15pt}}

```

```

\pgfpathlineto{\pgfpoint{15pt}{-15pt}}
\pgfsetfillcolor{cyan}
\pgfusepath{fill} % 结束一段路径
\pgfpathmoveto{\pgfpoint{15pt}{-15pt}} % 开启一段路径，与前一段路径是分离
的
\pgfpathlineto{\pgfpoint{30pt}{-15pt}}
\pgfpathlineto{\pgfpoint{30pt}{0pt}}
}
\state{final}
{
  \pgfpathlineto{\pgfpointdecoratedpathlast}
}
}
\tikz[decoration=example] % 画出路径
{
  \draw (0,0) -- (2,0) decorate {arc(180:0:1.5cm)} -- (2,-1);
}

```

下面的例子中，将 `node` 作为片段添加到路径上，而且 `node` 的边界线条颜色和填充颜色都是随机决定的：



```

\pgfdeclaredecoration{stars}{initial}{
  \state{initial}[width=15pt]
  {
    \pgfmathparse{round(rnd*100)}
    \pgfsetfillcolor{yellow!\pgfmathresult!orange}
    \pgfsetstrokecolor{yellow!\pgfmathresult!red}
    \pgfnode{star}{center}{-}{-}{\pgfusepath{stroke,fill}}
  }
}

```

```

}
\state{final}
{
  \pgfpathmoveto{\pgfpointdecoratedpathlast}
}
}
\tikz\path[decorate, decoration=stars, star point ratio=2, star points=5,
  inner sep=0, minimum size=rnd*10pt+2pt]
(0,0) .. controls (0,2) and (3,2) .. (3,0) .. controls (3,-3) and (0,0) ..
(0,-3) .. controls (0,-5) and (3,-5) .. (3,-3);

```

3. 当本状态的 `<code>` 执行完毕后（即本状态规定的片段添加完后），如果本状态的 `<options>` 中有 `next state` 选项，则程序会切换到下一个状态，解析下一个状态的选项和代码，否则程序会重复添加当前状态规定的片段，直到出现切换状态（或结束装饰过程）的条件。

命令 `\state` 的 `<options>` 中可以使用以下选项。

`/pgf/decoration automaton/switch if less than=<dimension> to <new state>`

本选项的作用是：当程序读取这个选项时，程序会检查输入路径的“未装饰部分”的长度，如果长度小于尺寸 `<dimension>`，就立即切换到状态 `<new state>`，不再解析本选项之后的选项或者代码（本状态规定的片段当然也不会被添加到输入路径上）；否则继续解析本选项之后的选项或者代码。



```

\pgfdeclaredecoration{illus}{initial}
{
  \state{initial}[switch if less than=10cm to final] % 若输入路径的未装饰部分
    的长度小于 10cm，则切换到状态 final
  {
    \pgfpathlineto{\pgfpoint{5pt}{5pt}}
    \pgfpathlineto{\pgfpoint{10pt}{0pt}}
  }
  \state{final}
  {
    \pgfpathcurveto
      {\pgfpointadd
        {\pgfpointscale{0.3333}{\pgfpointdecoratedpathlast}}
        {\pgfpoint{0pt}{20pt}}
      }
    {\pgfpointadd
      {\pgfpointscale{0.6666}{\pgfpointdecoratedpathlast}}

```

```

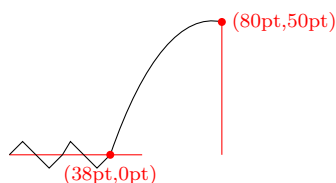
    {\pgfpoint{0pt}{20pt}}
  }
  {\pgfpointdecoratedpathlast}
}
}

\tikz [decoration=illus] \draw [decorate] (0,0)--(3,0); % 输入路径的长度是 3cm

```

`/pgf/decoration automaton/switch if input segment less than=<dimension> to <new state>`

当程序读取这个选项时，程序会计算当前的子输入路径的“未装饰部分”的长度，即参考点与当前子输入路径的终点之间的路径长度，若此长度小于 `<dimension>`，则切换到状态 `<new state>`。在装饰子输入路径时，这有利于避免装饰路径过分地突出子输入路径之外。



```

\pgfdeclaredecoration{illus}{initial}
{
  \state{initial}[width=18pt,switch if input segment less than=15pt to final
  ,]
  {
    \pgfpathlineto{\pgfpoint{5pt}{5pt}}
    \pgfpathlineto{\pgfpoint{15pt}{-5pt}}
    \pgfpathlineto{\pgfpoint{20pt}{0pt}}
  }
  \state{final}
  {
    \pgfpathcurveto
    {\pgfpointadd
     {\pgfpointscale{0.3333}{\pgfpointdecoratedpathlast}}
     {\pgfpoint{0pt}{20pt}}
    }
    {\pgfpointadd
     {\pgfpointscale{0.6666}{\pgfpointdecoratedpathlast}}
     {\pgfpoint{0pt}{20pt}}
    }
    {\pgfpointdecoratedpathlast}
  }
}

```



```

}
}

\tikz [decoration=illus]
{
  \draw [decorate] (0,0)--(50pt,0pt) (80pt,0pt)--(80pt,50pt);
  \draw [red] (0,0)--(50pt,0pt) (80pt,0pt)--(80pt,50pt);
  \fill [red] (38pt,0pt) circle(1.5pt) node [below] {\footnotesize(38pt,0pt)}
    (80pt,50pt) circle(1.5pt) node [right] {\footnotesize(80pt,50pt)};
}

```

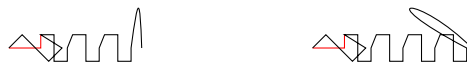
上面例子中，状态 `initial` 的片段的宽度规定为 `18pt`，片段的固有宽度是 `20pt`，第一个子输入路径是长度为 `50pt` 的线段。当在第一个子输入路径上添加两个初始状态的片段后，第二个片段的终点横标是 $18pt+20pt=38pt$ ，当前参考点的横标是 $18pt+18pt=36pt$ ，第一个子输入路径的“未装饰部分”长度是 $50pt-36pt=14pt$ ，小于选项 `switch if input segment less than` 规定的 `15pt`，因此切换到 `final` 状态，于是以第二个片段的终点 $(38pt,0pt)$ 为始点，以输入路径的终点 $(80pt,50pt)$ 为终点，构建一段控制曲线。

`/pgf/decoration automaton/width=<dimension>`

本选项指定的 `<dimension>` 规定了本状态的片段所占据的宽度，即片段所装饰的那一部分输入路径的长度。当程序读取这个选项时，程序计算输入路径的“未装饰部分”的长度，如果此长度大于本选项指定的 `<dimension>`，则添加本状态的片段；如果此长度小于本选项指定的 `<dimension>`，则直接切换到状态 `final`，因此本选项相当于

```
switch if less than=<dimension> to final
```

状态的 `<code>` 规定了片段，这个片段作为一个图形有自己的固有宽度，如果这个固有宽度与本选项指定的 `<dimension>` 不相同，就会让情况变得复杂一些。观察下面的例子。



```

\pgfdeclaredecoration{illus}{initial}
{
  \state{initial}[width=12pt,next state=second]
  {
    \pgfpathlineto{\pgfpoint{5pt}{5pt}}
    \pgfpathlineto{\pgfpoint{15pt}{-5pt}}
    \pgfpathlineto{\pgfpoint{20pt}{0pt}}
  }
  \state{second}[switch if less than=10pt to final]
  {
    \pgfpathlineto{\pgfpoint{0pt}{5pt}}
    \pgfpathlineto{\pgfpoint{5pt}{5pt}}
  }
}

```


```


\pgfpathlineto{\pgfpoint{5pt}{-5pt}}
\pgfpathlineto{\pgfpoint{10pt}{-5pt}}
\pgfpathlineto{\pgfpoint{10pt}{0pt}}
}
\state{final}
{
\pgfpathcurveto
{\pgfpointadd
{\pgfpointscale{0.3333}{\pgfpointdecoratedpathlast}}
{\pgfpoint{0pt}{20pt}}}
}
{\pgfpointadd
{\pgfpointscale{0.6666}{\pgfpointdecoratedpathlast}}
{\pgfpoint{0pt}{20pt}}}
}
{\pgfpointdecoratedpathlast}
}
}

\tikz [decoration=illus]
{
\draw [red] (0,0)--(12pt,0)---+(0,5pt);
\draw [decorate] (0,0)--(50pt,0pt);
}
\hspace{2cm}
\tikz [decoration=illus]
{
\draw [red] (0,0)--(12pt,0)---+(0,5pt);
\draw [decorate] (0,0)--(60pt,0pt);
}
}

```

在上面的例子中，自定义的装饰路径包括 3 个状态，initial, second, final.

状态 initial 的片段是“之字形”折线 .

原来所设想的状态 second 的片段是 .

状态 final 的片段是一段 curve-to 曲线。

状态 initial 的片段的固有宽度是 20pt，但是该状态的选项 width=12pt 规定这个片段只占据 12pt 的宽度，即所装饰的那一部分输入路径的长度只有 12pt，这导致状态 second 的片段与原来设想的不一样。

左右两个图的代码的差别仅仅是输入路径的长度不一样，一个是 50pt，一个是 60pt，但是二

者的最后一个片段差别很大。

`/pgf/decoration automaton/repeat state=<repetitions>` (无默认值, 初始值 0)

这个选项的参数 `<repetitions>` 是个整数。如果本状态的片段被添加, 那么添加片段后, 本选项使得片段再被重复添加 `<repetitions>` 次, 也就是说, 总共添加了 `<repetitions>+1` 次。在本选项之后应当使用选项 `next state` 指定下一个状态, 否则, 程序还是会继续添加当前状态的片段, 直到出现切换状态 (或结束装饰过程) 的条件。



```
\pgfdeclaredecoration{illus}{initial}
{
  \state{initial}[width=10pt,repeat state=2, next state=second]
  {
    \pgfpathlineto{\pgfpoint{5pt}{5pt}}
    \pgfpathlineto{\pgfpoint{15pt}{-5pt}}
    \pgfpathlineto{\pgfpoint{20pt}{0pt}}
  }
  \state{second}[width=5pt,repeat state=2]
  {
    \pgfpathlineto{\pgfpoint{0pt}{5pt}}
    \pgfpathlineto{\pgfpoint{5pt}{5pt}}
    \pgfpathlineto{\pgfpoint{5pt}{-5pt}}
    \pgfpathlineto{\pgfpoint{10pt}{-5pt}}
    \pgfpathlineto{\pgfpoint{10pt}{0pt}}
  }
  \state{final}
  {
    \pgfpathcurveto
      {\pgfpointadd
        {\pgfpointscale{0.3333}{\pgfpointdecoratedpathlast}}
        {\pgfpoint{0pt}{20pt}}
      }
    }
    {\pgfpointadd
      {\pgfpointscale{0.6666}{\pgfpointdecoratedpathlast}}
      {\pgfpoint{0pt}{20pt}}
    }
    {\pgfpointdecoratedpathlast}
  }
}
```

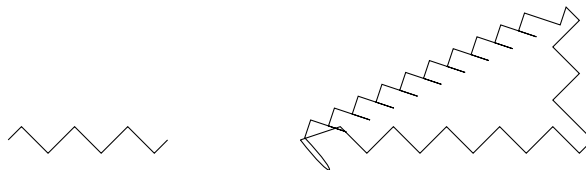
```
\tikz [decoration=illus]
{
  \draw [decorate] (0,0)--(100pt,0pt);
}
```

`/pgf/decoration automaton/next state=<new state>`

当程序读到这个选项时，会立即切换到状态 `<new state>`，原来状态中其它未读取的选项都被忽略。如果没有这个选项，程序会重复当前状态，直到出现切换状态（或结束装饰过程）的条件。

`/pgf/decoration automaton/if input segment is closepath=<options>`

当程序读到这个选项时，会检查当前的子输入路径是否为闭合路径，如果是，则针对“起到封闭作用的那段子路径”执行 `<options>`；如果不是，则没有动作，继续读取之后的选项。你可以利用这个选项的 `<options>` 对“起到封闭作用的那段子路径”做某种特殊操作，也可以切换到其它状态。



```
\pgfdeclaredecoration{illus}{initial}
{
  \state{initial}[width=20pt,if input segment is closepath={width=10pt},]
  {
    \pgfpathlineto{\pgfpoint{5pt}{5pt}}
    \pgfpathlineto{\pgfpoint{15pt}{-5pt}}
    \pgfpathlineto{\pgfpoint{20pt}{0pt}}
  }
  \state{final}
  {
    \pgfpathcurveto
    {\pgfpointadd
     {\pgfpointscale{0.3333}{\pgfpointdecoratedpathlast}}
     {\pgfpoint{0pt}{20pt}}
    }
    {\pgfpointadd
     {\pgfpointscale{0.6666}{\pgfpointdecoratedpathlast}}
     {\pgfpoint{0pt}{20pt}}
    }
  }
}
```

```

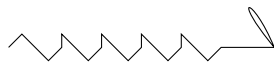
    {\pgfpointdecoratedpathlast}
  }
}

\tikz [decoration=illus]
{
  \draw [decorate] (-50pt,0pt)--(-100pt,0pt) (0,0)--(100pt,0pt)--(100pt,50pt)
    --cycle;
}

```

/pgf/decoration automaton/auto end on length=<dimension>

设计这个选项是为了便利，本选项的作用也可由其它选项实现。本选项的作用是：(1) 当程序读取本选项时，如果输入路径的“未装饰部分”的长度不大于 <dimension>，那么在当前坐标点与输入路径的终点之间画一个直线段并结束装饰路径；如果输入路径的结尾有闭合路径操作 (`closepath`)，则装饰路径自己也是闭合路径；(2) 如果输入路径的“未装饰部分”的长度大于 <dimension>，则检查“当前子输入路径是否由闭合路径操作生成 `closepath`”并且检查“当前子输入路径的未装饰部分的长度是否不大于 <dimension>”，如果二者都“是”，则在当前坐标点与子输入路径的终点之间画一个直线段并且把装饰路径作成闭合的，然后再继续处理之后的子输入路径，直到出现切换状态（或结束装饰过程）的条件。其它情况下，本选项无作用。



```

\begin{center}
\pgfdeclaredecoration{illus}{initial}
{
  \state{initial}[width=15pt,auto end on length=30pt]
  {
    \pgfpathlineto{\pgfpoint{5pt}{5pt}}
    \pgfpathlineto{\pgfpoint{15pt}{-5pt}}
    \pgfpathlineto{\pgfpoint{20pt}{0pt}}
  }
  \state{final}
  {
    \pgfpathcurveto
    {\pgfpointadd
      {\pgfpointscale{0.3333}{\pgfpointdecoratedpathlast}}
      {\pgfpoint{0pt}{20pt}}
    }
  }
  \pgfpointadd

```

```

        {\pgfpointscale{0.6666}{\pgfpointdecoratedpathlast}}
        {\pgfpoint{0pt}{20pt}}
    }
    {\pgfpointdecoratedpathlast}
}
}

\tikz [decoration=illus]
{
  \draw [decorate] (0,0)--(100pt,0pt);
}

```

/pgf/decoration automaton/auto corner on length=<dimension>

这个选项的作用是：首先，如果 T_EX-if 的条件判断宏 `\ifpgfdecoratepathhascorners` 的值是 `false`，则什么也不做；否则，检查当前子输入路径的“未装饰部分”的长度是否不大于 `<dimension>`。如果是，则用直线段将当前坐标点与当前子输入路径的终点连起来，然后在当前状态下继续装饰后面的子输入路径，直到出现切换状态（或结束装饰过程）的条件。

这个选项针对的是“子输入路径”，不过经测试未能实现。例如文件《`pgflibrarydecorations.pathmorphing.code`》对装饰样式 `bumps` 的定义中有选项 `auto end on length` 和 `auto corner on length`，后一个选项的作用没有显现。

/pgf/decoration automaton/persistent precomputation=<precode>

前面提到，如果当前状态的片段代码 `<code>` 要被执行，则会把 `<code>` 放入一个 T_EX 分组中来执行。使用本选项后，若 `<code>` 被执行，则本选项的 `<precode>` 会先被执行，然后再执行 `<code>`，并且 `<precode>` 会在容纳 `<code>` 的 T_EX 分组之外被执行。但是注意，若 `<precode>` 被执行，在容纳 `<code>` 的 T_EX 分组之内不会再使用变换矩阵。

/pgf/decoration automaton/persistent postcomputation=<postcode>

类似前一个选项。使用本选项后，当前状态的片段代码 `<code>` 被执行后，则本选项的 `<postcode>` 会被执行，并且 `<postcode>` 会在容纳 `<code>` 的 T_EX 分组之外被执行。

下面的宏保存的值可能很有用。

`\pgfdecoratedpathlength`

这个宏保存当前的输入路径的长度。如果输入路径包含数个子输入路径，则输入路径的长度是各个子输入路径的长度之和。

`\pgfdecoratedinputsegmentlength`

这个宏保存当前的子输入路径的长度。

`\pgfpointdecoratedpathlast`

这个宏保存当前的输入路径的终点。

`\pgfpointdecoratedinputsegmentlast`

这个宏保存当前的子输入路径的终点。

`\pgfdecoratedangle`

前面提到，如果当前状态的片段代码 `<code>` 要被执行，则会把 `<code>` 放入一个 `\TeX` 分组中来执行，并且在这个分组内进行平移和旋转变换，其中旋转变换的转角就是这个宏保存的值（使用角度制）。

`\pgfdecoratedremainingdistance`

这个宏保存的值是，当下的，输入路径的未装饰部分的长度。

`\pgfdecoratedcompleteddistance`

这个宏保存的值是，当下的，输入路径的已装饰部分的长度。

`\pgfdecoratedinputsegmentremainingdistance`

这个宏保存的值是，当下的，子输入路径的未装饰部分的长度。

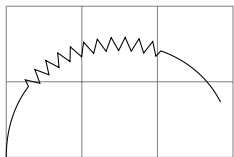
`\pgfdecoratedinputsegmentcompleteddistance`

这个宏保存的值是，当下的，子输入路径的已装饰部分的长度。

还有其它的选项和宏，参考 §48.

98.4 `{pgfdecoration}` 环境

当定义了装饰路径后，就可以使用它。下面介绍 `{pgfdecoration}` 环境，先看一个例子。



```
\begin{tikzpicture}[decoration={segment length=5pt}]
  \draw [help lines] grid (3,2);
  \begin{pgfdecoration}{curveto}{1cm},{zigzag}{2cm},{curveto}{1cm}}
    \pgfpathmoveto{\pgfpointorigin}
    \pgfpathcurveto{\pgfpoint{0cm}{2cm}} {\pgfpoint{3cm}{2cm}}
      {\pgfpoint{3cm}{0cm}}
  \end{pgfdecoration}
  \pgfusepath{stroke}
\end{tikzpicture}
```

`{pgfdecoration}` 环境用在 `{tikzpicture}` 环境或 `{pgfpicture}` 环境中，用于构建装饰路径，它有自己的结构。

`\begin{pgfdecoration}{<decoration list>}`

`<environment contents>`

`\end{pgfdecoration}`

`<decoration list>` 是用逗号分隔的装饰路径列表，列表项的格式是

`{<decoration>}{<length>}{<before code>}{<after code>}`

其中 `<decoration>` 是装饰路径的名称, `<length>` 是一个尺寸, 规定长度为 `<length>` 的一段输入路径用 `<decoration>` 来装饰。`<before code>` 与 `<after code>` 是可选的。`<before code>` 在装饰自动化过程之前执行, `<after code>` 在装饰自动化过程之后执行。

在前面的例子中, 装饰路径列表是

```
{\curveto}{1cm},{\zigzag}{2cm},{\curveto}{1cm}}
```

其中没有 `<before code>` 和 `<after code>`. 这个列表规定: 输入路径开始的长度为 1cm 的一段用 `curveto` 型路径装饰, 之后长度为 2cm 的一段用 `zigzag` 型路径装饰, 再后的长度为 1cm 的一段用 `curveto` 型路径装饰。

`<environment contents>` 是由路径构建命令组成, 这些命令构建“输入路径”。

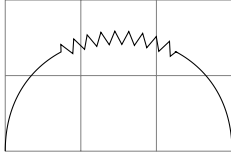
当程序处理这个环境时, 发生以下动作。

1. 保存前路径。
2. 执行 `<environment contents>` 中的路径命令, 得到输入路径, 保存之。然后分割输入路径为若干个输入路径, 分割的标准参照 `<decoration list>`. 假设 `<decoration list>` 中各个列表项的长度项依次是 `<length 1>`, `<length 2>`, `<length 3>`……`<length n>`, 程序会将输入路径分隔成长度依次是 `<length 1>`, `<length 2>`, `<length 3>`……`<length n>` 的子输入路径, 程序会在分割点处将路径“打断”。假如各长度之和 $L = \text{<length 1>} + \dots + \text{<length n>}$ 大于输入路径的长度, 那么超出输入路径的那一段装饰路径会被忽略。假如各长度之和 L 小于输入路径的长度, 那么最后一段输入路径就没有装饰。
3. 完成前路径。
4. 沿着输入路径构建装饰路径, 得到输出路径。

使用 `{pgfdecoration}` 环境时注意以下几点:

- 如果 `<environment contents>` 不以 `\pgfpathmoveto` 开头, 那么前路径的最后一个点就是输入路径的起点。
- 多余的 `\pgfpathmoveto` 命令会被忽略。
- `<environment contents>` 结尾处的 `\pgfpathmoveto` 命令会被忽略。
- `<environment contents>` 中的 `\pgfpathclose` 命令会被解释成一个直线段, 是输入路径的一部分。
- 在装饰自动化过程中, 程序不会自动使用 `move-to` 操作来造成分离的装饰路径, 程序只会根据状态规定以及计算的长度来构建装饰路径, 因此在需要间断的地方, 装饰路径可能没有间断。此时需要在输入路径中使用 `\pgfpathmoveto` 命令或其它办法 (如修改状态规定)。
- 如果在状态规定的片段 (装饰路径) 中或者在 `<after code>` 中使用 `\pgfusepath` 命令, 那么该命令使用的路径包括前路径和直到当前的装饰路径。

下面的例子中使用了 `\pgfdecoratedpathlength/3` 这样的长度算式:



```

\begin{tikzpicture}[decoration={segment length=5pt}]
\draw [help lines] grid (3,2);
\begin{pgfdecoration}{
  {curveto}{\pgfdecoratedpathlength/3},
  {zigzag}{\pgfdecoratedpathlength/3},
  {curveto}{\pgfdecoratedremainingdistance}
}
\pgfpathmoveto{\pgfpointorigin}
\pgfpathcurveto
  {\pgfpoint{0cm}{2cm}}{\pgfpoint{3cm}{2cm}}{\pgfpoint{3cm}{0cm}}
\end{pgfdecoration}
\pgfusepath{stroke}
\end{tikzpicture}

```

如果使用 <before code> 或者 <after code>, 下面的宏可能用的上。

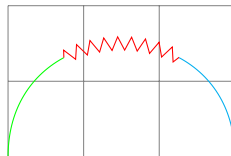
`\pgfpointdecoratedpathfirst`

这个宏保存当前输入路径的起点。

`\pgfpointdecoratedpathlast`

这个宏保存当前输入路径的终点。

下面的例子中, 使用 <before code> 或者 <after code> 达到某种效果:



```

\begin{tikzpicture}
\draw [help lines] grid (3,2);
\begin{pgfdecoration}{
  {curveto}{\pgfdecoratedpathlength/3}{\pgfsetstrokecolor{green}}{\pgfusepath{
    stroke}},
  {zigzag}{\pgfdecoratedpathlength/3}{\pgfpathmoveto{\pgfpointdecoratedpathfirst}
    \pgfdecorationsegmentlength=5pt}
  {\pgfsetstrokecolor{red}
    \pgfusepath{stroke}
    \pgfpathmoveto{\pgfpointdecoratedpathlast}
    \pgfsetstrokecolor{cyan}},
  {curveto}{\pgfdecoratedremainingdistance}
}
\pgfpathmoveto{\pgfpointorigin}
\pgfpathcurveto

```

```

    {\pgfpoint{0cm}{2cm}}{\pgfpoint{3cm}{2cm}}{\pgfpoint{3cm}{0cm}}
  \end{pgfdecoration}
  \pgfusepath{stroke}
\end{tikzpicture}

```

当 {pgfdecoration} 环境结束后，下面的宏可用：

\pgfdecorateexistingpath

这个宏保存 {pgfdecoration} 环境之前的前路径。

\pgfdecoratedpath

这个宏保存 {pgfdecoration} 环境之内的输入路径。

\pgfdecorationpath

这个宏保存 {pgfdecoration} 环境产生的输出路径。如果输出的装饰路径在 {pgfdecoration} 环境中被使用，即在状态规定的片段（装饰路径）中或者在 <after code> 中使用 \pgfusepath 命令，那么这个宏只保存最后一个未被使用的片段。

\pgfpointdecoratedpathlast

这个宏保存 {pgfdecoration} 环境内的输入路径的终点。

\pgfpointdecorationpathlast

这个宏保存 {pgfdecoration} 环境产生的输出路径的终点。

下面的样式是针对每个装饰路径（片段）的，你可以改变它的默认设置。

/pgf/every decoration (样式，初始值 empty)

如果要在 plain-TeX 中使用 {pgfdecoration} 环境，可以使用下面的组合：

```

\pgfdecoration{<name>}
  <environment contents>
\endpgfdecoration

```

还有一些比较简捷的命令能够实现 {pgfdecoration} 环境：

\pgfdecoratepath{<name>}{<path commands>}

<name> 是装饰路径的名称，<path commands> 构建输入路径。用 <name> 类型的装饰路径来装饰 <path commands>。等效于

```

\pgfdecorate{<name>}{\pgfdecoratedpathlength}
{\pgfdecoratebeforecode}{\pgfdecorateaftercode}}
  <the path commands>
\endpgfdecorate

```

\pgfdecoratecurrentpath{<name>}

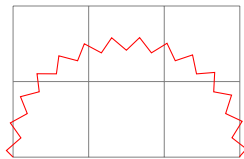
用 <name> 类型的装饰路径来装饰当前路径。

\pgfdecoratebeforecode{<code>}

相当于执行 <before code>.

`\pgfdecorateaftercode{<code>}`

相当于执行 <after code>.

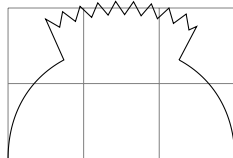


```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\pgfpathmoveto{\pgfpointorigin}
\pgfpathcurveto{\pgfpoint{0cm}{2cm}}{\pgfpoint{3cm}{2cm}}{\pgfpoint{3cm}{0cm}}
\pgfdecoratebeforecode{\pgfsetstrokecolor{red}}
\pgfdecoratecurrentpath{zigzag}
\pgfusepath{stroke}
\end{tikzpicture}
```

前面提到，片段（装饰路径）的代码会被放入一个 $\text{T}_\text{E}_\text{X}$ 分组内执行，并且在分组内会进行默认类型的坐标变换，得到一个用于绘制片段的“片段坐标系”。除了默认的坐标变换，还可以用下面的命令规定其它的变换（相对于“片段坐标系”）：

`\pgfsetdecorationsegmenttransformation{<code>}`

<code> 是坐标变换命令，这个命令用于对添加的片段（装饰路径）做某种变换（在本片段的坐标系内）。假设前后相继的两个片段，前一片段的终点是后一片段的起点，若对后一片段做变换，使得后一片段的起点偏离了前一片段的终点，那么就直接在这两点之间用直线段连起来，如下图所示：



```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\begin{pgfdecoration}{
\curveto{\pgfdecoratedpathlength/3},
\zigzag{\pgfdecoratedpathlength/3}
{
\pgfdecorationsegmentlength=5pt
\pgfsetdecorationsegmenttransformation{\pgftransformyshift{.5cm}}
},
\curveto{\pgfdecoratedremainingdistance}
}
```

```

\pgfpathmoveto{\pgfpointorigin}
\pgfpathcurveto
  {\pgfpoint{0cm}{2cm}}{\pgfpoint{3cm}{2cm}}{\pgfpoint{3cm}{0cm}}
\end{pgfdecoration}
\pgfusepath{stroke}
\end{tikzpicture}

```

98.5 Meta-Decorations

英语单词前缀 meta- [ˈmetə] 的意思如下。

《微软计算机词典》: prefix Literally, a prefix that describes a process or characteristic beyond the normal meaning of the word without the prefix. For example, metaphysics is “beyond physics.” In computing, meta- is usually attached to a word to indicate that the “metaterm” describes, defines, or acts upon objects or concepts of the same type as itself. So, for example, metadata is data about data and a metatool is a tool for working on tools.

《维基英语》: In computer science, a common prefix that means “about”. So, for example, metadata is data that describes other data (data about data). A metalanguage is a language used to describe other languages. A metafile is a file that contains other files.

Meta-Decorations 就是用“已有的装饰路径”来构建“新的装饰路径”。在 §98.3 中, 使用命令 `\pgfdeclaredcoration` 自定义装饰路径时, 状态的片段代码 `<code>` 中使用的是绘图命令。而用 Meta-Decorations 方式自定义装饰路径时, 则必须用已定义的装饰路径类型来构建状态的片段。

98.5.1 定义一个 Meta-Decorations

```
\pgfdeclaremetadecorate{<name>}{<initial state>}{<states>}
```

本命令的用法与 `\pgfdeclaredcoration` 类似。本命令定义一个 meta-decoration, 其名称为 `<name>`. `<initial state>` 是初始状态的名称。 `<states>` 是状态列表, 各个状态使用命令 `\state` 定义。

```
\state{<name>}[<options>]{<code>}
```

`<name>` 是状态的名称。meta-decoration 的状态片段代码 `<code>` 不会被放入 \TeX 分组内执行, 否则会有多余的计算。这里的 `<options>` 中可以使用以下选项。

```
/pgf/meta-decoration automaton/switch if less than=<dimension> to <new state>
```

当程序读取这个选项时, 会检查输入路径的“未装饰部分”的长度, 如果这个长度小于 `<dimension>` 则立即切换到状态 `<new state>`。

```
/pgf/meta-decoration automaton/width=<dimension>
```

本选项规定片段的“宽度”, 即片段所装饰的路径长度。如果输入路径的“未装饰部分”的长度小于 `<dimension>`, 则立即切换到终结状态 `final`。

```
/pgf/meta-decoration automaton/next state=<new state>
```

当本状态的片段添加完毕后，切换到这个选项指定的状态 `<new state>`，如果没有这个选项，就会一直添加本状态的片段，直到出现切换状态（或结束装饰过程）的条件。

在本命令的 `<code>` 中只能使用以下命令：

`\decoration{<name>}`

`<name>` 是已经定义的或已有的装饰类型的名称，指定当前状态的片段是 `<name>` 类型的。如果没有给出这个命令，就默认使用 `moveto` 装饰类型。

`\beforedecoration{<before code>}`

`<before code>` 是添加本状态的片段之前所需要执行的代码，例如，可以设置该片段中的线段长度、振幅等。可以没有这个命令。

`\afterdecoration{<after code>}`

`<after code>` 是添加本状态的片段之后所需要执行的代码。可以没有这个命令。

在定义状态时，下面的宏可能用得上。

`\pgfpointmetadecoratedpathfirst`

当执行 `<before code>` 时，这个宏保存当前子输入路径的起点，这个宏可以用在 `<before code>` 中。

`\pgfpointmetadecoratedpathlast`

当执行 `<after code>` 时，这个宏保存当前子输入路径的终点，这个宏可以用在 `<after code>` 中。

`\pgfmetadecoratedpathlength`

这个宏保存输入路径的长度。

`\pgfmetadecoratedcompleteddistance`

这个宏保存当下的输入路径的“已装饰部分”的长度。

`\pgfmetadecoratedremainingdistance`

这个宏保存当下的输入路径的“未装饰部分”的长度。

`\pgfmetadecoratedinputsegmentcompleteddistance`

这个宏保存当下的子输入路径的“已装饰部分”的长度。

`\pgfmetadecoratedinputsegmentremainingdistance`

这个宏保存当下的子输入路径的“未装饰部分”的长度。



```

\pgfdeclaremetadecoration{arrows}{initial}
{
  \state{initial}[width=0pt, next state=arrow]
  {
    \pgfmathdivide{100}{\pgfmetadecoratedpathlength}
    \let\factor\pgfmathresult
    \pgfsetlinewidth{1pt}
    \pgfset{/pgf/decoration/segment length=4pt}
  }
  \state{arrow}[
    switch if less than=\pgfmetadecorationsegmentlength to final,
    width=\pgfmetadecorationsegmentlength/3,
    next state=zigzag]
  {
    \decoration{curveto}
    \beforedecoration
    {
      \pgfmathparse{\pgfmetadecoratedcompleteddistance*\factor}
      \pgfsetcolor{red!\pgfmathresult!yellow}
      \pgfpathmoveto{\pgfpointmetadecoratedpathfirst}
    }
  }
  \state{zigzag}[width=\pgfmetadecorationsegmentlength/3, next state=end arrow]
  {
    \decoration{zigzag}
  }
  \state{end arrow}[width=\pgfmetadecorationsegmentlength/3, next state=move]
  {
    \decoration{curveto}
    \beforedecoration{\pgfpathmoveto{\pgfpointmetadecoratedpathfirst}}
    \afterdecoration
    {
      \pgfsetarrowsend{to}
      \pgfusepath{stroke}
    }
  }
  \state{move}[width=\pgfmetadecorationsegmentlength/2, next state=arrow]{}
  \state{final}{}
}

```

```

}
\tikz[rotate=90] \draw[decorate,decoration={arrows,meta-segment length=2cm}]
(0,0) .. controls (0,2) and (3,2) .. (3,0)
.. controls (3,-2) and (0,-2) .. (0,-4)
.. controls (0,-6) and (3,-6) .. (3,-8)
.. controls (3,-10) and (0,-10) .. (0,-8);

```

98.5.2 预定义的 Meta-decorations

目前没有预定义的 Meta-decorations.

98.5.3 `{pgfmetadecoration}` 环境

`{pgfmetadecoration}` 环境与 `{pgfdecoration}` 环境的用法是类似的。

```

\begin{pgfmetadecoration}{<name>}
  <environment contents>
\end{pgfmetadecoration}

```

这是 L^AT_EX 中 `{pgfmetadecoration}` 环境的用法。

```

\pgfmetadecoration{<name>}
  <environment contents>
\endpgfmetadecoration

```

这是 plain-T_EX 中 `{pgfmetadecoration}` 环境的用法。

99 使用路径

99.1 Overview

构建一个路径后就可以使用它，“使用”的意思是，例如，你可以画出它，填充颜色，用于剪切其它路径，等等。也有很多图形参数能影响路径的外观，例如线宽，线型，颜色等。设置图形参数的命令都以 `\pgfset` 开头。

注意，影响路径的外观的选项都是对整个路径有效的，例如，对于一个路径来说，你不能把该路径的前一段画成红色 (red)，而把后一段画成绿色 (green)。

路径的使用方式有以下几种：

1. 画出路径，`stroke` 或 `draw`.
2. 给路径添加箭头，`arrow tips`.
3. 填充颜色，`fill`.
4. 作为剪切路径来剪切之后的路径，`clip`.
5. 画阴影，`shade`.
6. 用作边界盒子，`use as bounding box`.

以上几种方式可以混合使用，当然同时使用 `fill` 和 `shade` 没有意义。

`\pgfusepath{<actions>}`

`<actions>` 是一个或数个使用路径的方式选项，相邻选项之间用逗号分开。下面是例子。

- `fill`，参考 §99.4.



```
\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{1cm}{1cm}}
  \pgfpathlineto{\pgfpoint{1cm}{0cm}}
  \pgfusepath{fill}
\end{pgfpicture}
```

- `stroke`，参考 §99.2.



```
\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{1cm}{1cm}}
  \pgfpathlineto{\pgfpoint{1cm}{0cm}}
  \pgfusepath{stroke}
\end{pgfpicture}
```

- `draw`，等于 `stroke`.
- `clip`，参考 §99.5.



```
\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{1cm}{1cm}}
  \pgfpathlineto{\pgfpoint{1cm}{0cm}}
  \pgfusepath{stroke,clip}
  \pgfpathcircle{\pgfpoint{1cm}{1cm}}{0.5cm}
  \pgfusepath{fill}
\end{pgfpicture}
```

- `discard`，丢弃当前路径，等效于 `\pgfusepath{}`.

`\pgfusepath{stroke,fill}` 等于 `\pgfusepath{fill,stroke}`，即这两个选项的次序可换。

使用命令 `\pgfshadepath` 给路径添加阴影效果，见 §109.

99.2 画出路径

用命令 `\pgfusepath{stroke}` 画出当前路径。有许多图形参数能影响画出的路径的外观，多数参数的有效范围都受到绘图环境的限制，有的参数，例如判断区域内部或外部的“奇偶规则”、“非零规则”，以及箭头选项，都受到 T_EX 分组的限制。不过在将来的版本中，可能会改变这一状况。

99.2.1 图形参数：线宽 Line Width

`\pgfsetlinewidth{<line width>}`

`<line width>` 是 T_EX 尺寸。本命令规定其后的 `\pgfusepath{stroke}` 命令画出的路径线条的线宽（限于当前绘图环境内）。



```
\begin{pgfpicture}
  \pgfsetlinewidth{1mm}
  \pgfpathmoveto{\pgfpoint{0mm}{0mm}}
  \pgfpathlineto{\pgfpoint{2cm}{0mm}}
  \pgfusepath{stroke}
  \pgfsetlinewidth{2\pgflinewidth} % 线宽加倍
  \pgfpathmoveto{\pgfpoint{0mm}{5mm}}
  \pgfpathlineto{\pgfpoint{2cm}{5mm}}
  \pgfusepath{stroke}
\end{pgfpicture}
```

`\pgflinewidth`

这个宏保存的是当前的线宽。你可以全局地设置这个宏的值，然后在某个绘图环境中局部地修改它。

0.4pt

```
\tikz;
\the\pgflinewidth
```

99.2.2 图形参数：线冠 Caps 与交接 Joins

`\pgfsetbuttcap`

规定线冠为 butt cap，见 §15.3.1.

`\pgfsetroundcap`

规定线冠为 round cap，见 §15.3.1.

`\pgfsetrectcap`

规定线冠为 square cap，见 §15.3.1.

`\pgfsetroundjoin`

规定交接为 round join，见 §15.3.1.

`\pgfsetbeveljoin`

规定交接为 bevel join，见 §15.3.1.

`\pgfsetmiterjoin`

规定交接为 miter join，见 §15.3.1.

`\pgfsetmiterlimit{<miter limit factor>}`

设置交接的极限因子，见 §15.3.1.

99.2.3 图形参数：线型 Dashing

`\pgfsetdash{<list of even length of dimensions>}{<phase>}`

这个命令规定线型，注意其中的花括号结构，举例来说：

```
\pgfsetdash{0.5cm}{0.5cm}{0.1cm}{0.2cm}{0.2cm}
```

这个命令规定的线型是，0.5cm 的实线，后接 0.5cm 的虚线，后接 0.1cm 的实线，后接 0.2cm 的虚线，这是一个“周期”，画线时不断重复这个周期。最后一个花括号规定“相位”。参考 §15.3.2.

`\pgfsetdash{}{0pt}`

这个命令得到实线。

99.2.4 图形参数：线条颜色

`\pgfsetstrokecolor{<color>}`

本命令设置之后画出的路径的颜色，<color> 是 L^AT_EX 的颜色格式 red 或 black!20!red. 也可以使用 L^AT_EX 的颜色命令 `\color{<color>}` 设置颜色，不过这个命令会修改所有颜色（包括线条和填充），并且其有效范围受到 T_EX 分组的限制。



```
\begin{pgfpicture}
  \pgfsetlinewidth{1pt}
  \color{red} % 设置各种颜色设为 red
  \pgfpathcircle{\pgfpoint{0cm}{0cm}}{3mm} \pgfusepath{fill,stroke}
  \pgfsetstrokecolor{black} % 设置线条颜色为 black
  \pgfpathcircle{\pgfpoint{1cm}{0cm}}{3mm} \pgfusepath{fill,stroke}
  \color{red} % 设置各种颜色设为 red
  \pgfpathcircle{\pgfpoint{2cm}{0cm}}{3mm} \pgfusepath{fill,stroke}
\end{pgfpicture}
```

注意上面图形中的第三个圆，其边界线条还是黑色，与手册描述不一样。

`\pgfsetcolor{<color>}`

这个命令同时设置线条颜色的填充颜色，其有限范围受到绘图环境的限制（而不是 T_EX 分组）。

99.2.5 线条透明度

使用命令 `\pgfsetstrokeopacity{<value>}` 设置线条透明度，见 §110.

99.2.6 双线的内线

“双线功能”参考 §15.3.4. 如果给双线加箭头，那么程序会先画出外线，再画内线，然后给内线加箭头，所以内线的线宽对箭头的尺寸有影响。

`\pgfsetinnerlinewidth{<dimension>}`

程序读取这个命令时，就默认对当前路径启用双线功能，本命令规定内线的线宽。在默认下这里的 <dimension> 是 0pt，在这个尺寸下根本不会有画内线的动作（注意区别“不画线”与“画出线宽为 0pt 的线”），此时如果加箭头就加在“外线”上。也就是说，<dimension> 是 0pt 的时候就会关闭双线功能。

这个命令的有限范围受到 \TeX 分组的限制。如果一个路径用作剪切路径，对该路径也不能使用双线功能（当然你可以手工作出双线效果），在将来的版本中可能会改变这一点。



```
\begin{pgfpicture}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{1cm}{1cm}}
  \pgfpathlineto{\pgfpoint{1cm}{0cm}}
  \pgfsetlinewidth{2pt}
  \pgfsetinnerlinewidth{1pt}
  \pgfusepath{stroke}
\end{pgfpicture}
```

`\pgfsetinnerstrokecolor{<color>}`

这个命令设置双线功能中内线的颜色，其有效范围受到 \TeX 分组的限制。

99.3 给路径加箭头

关于箭头的详细规定参考 §16.2. 注意不能给封闭路径或者剪切路径加箭头。

`\pgfsetarrowsstart{<start arrow tip specification>}`

本命令指定路径始端的箭头类型，其有效范围受到 \TeX 分组的限制。

`\pgfsetarrowsend{<end arrow tip specification>}`

本命令指定路径末端的箭头类型，其有效范围受到 \TeX 分组的限制。



```
\begin{pgfpicture}
  \pgfsetarrowsstart{Latex[length=10pt]}
  \pgfsetarrowsend{Computer Modern Rightarrow}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{1cm}{0cm}}
  \pgfusepath{stroke}
\end{pgfpicture}
```

`\pgfsetarrows{<argument>}`

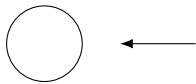
这个命令指定箭头的样式，<argument> 是指定箭头样式的那些句法格式，参考 §16.4.



```
\begin{pgfpicture}
  \pgfsetarrows{Latex[length=10pt]-{Stealth[] . Stealth[]}}
  \pgfpathmoveto{\pgfpointorigin}
  \pgfpathlineto{\pgfpoint{2cm}{0cm}}
  \pgfusepath{stroke}
\end{pgfpicture}
```

`\pgfsetshortenstart{<dimension>}`

这个命令可以在路径开头处将路径截去一段，也就是将路径起点沿着路径方向移动一段，使得路径缩短，这一段的长度就是 <dimension>. 如果对某个路径同时使用这个命令以及加箭头的命令，程序先将路径如此缩短，然后再加箭头，而添加箭头时程序还会自动将路径再缩短一段，这样本命令的效果相当于在箭头的尖端与箭头原来的目标点之间使用 `sep` 选项 (§16.4.2)。



```
\begin{pgfpicture}
  \pgfpathcircle{\pgfpointorigin}{5mm}
  \pgfusepath{stroke}
  \pgfsetarrows{Latex-}
  \pgfsetshortenstart{5mm}
  \pgfpathmoveto{\pgfpoint{5mm}{0cm}}
  \pgfpathlineto{\pgfpoint{2cm}{0cm}}
  \pgfusepath{stroke}
\end{pgfpicture}
```

`\pgfsetshortenend{<dimension>}`

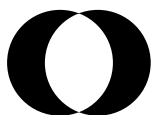
这个命令可以在路径结尾处将路径截去一段，也就是将路径终点沿着反路径方向移动一段，使得路径缩短，这一段的长度就是 <dimension>.

99.4 填充路径

填充路径的意思是给路径“内部”的点用某种颜色来着色。只有闭合路径才有“内部”与“外部”的区分。在填充路径时，程序会检查路径是否闭合，如果路径不是闭合的就把它当成是闭合的，即在路径的始点与终点之间使用闭合操作（但不画线）。对于一个闭合路径和某个点，需要某种规则来判断这个点是否属于该路径的“内部”。有两种规则，奇偶规则、非零规则，默认使用非零规则 (§15.5)。

99.4.1 图形参数：判断内部点的规则**`\pgfseteorule`**

这个命令指定奇偶规则，其有效范围受到 T_EX 分组的限制。



```
\begin{pgfpicture}
  \pgfseteorule
  \pgfpathcircle{\pgfpoint{0mm}{0cm}}{7mm}
  \pgfpathcircle{\pgfpoint{5mm}{0cm}}{7mm}
  \pgfusepath{fill}
\end{pgfpicture}
```

`\pgfsetnonzerorule`

这个命令指定非零规则，其有效范围受到 T_EX 分组的限制。



```
\begin{pgfpicture}
  \pgfsetnonzerorule
  \pgfpathcircle{\pgfpoint{0mm}{0cm}}{7mm}
  \pgfpathcircle{\pgfpoint{5mm}{0cm}}{7mm}
  \pgfusepath{fill}
\end{pgfpicture}
```

99.4.2 图形参数：填充色

`\pgfsetfillcolor{<color>}`

这个命令指定填充用的颜色。

99.4.3 图形参数：填充色的不透明度

用命令 `\pgfsetfillopacity{<value>}` 设置填充色的不透明度，见 §110.


99.5 剪切路径

当使用选项 `clip` 时，当前路径就成为剪切路径，对之后的路径起到剪切作用。剪切时，只保留路径内部的图形，所以这里仍然需要判断路径内部或外部点的规则。

剪切路径的剪切作用会一直持续到环境结束。参考 §15.9.

99.6 将路径用作边界盒子

使用 `use as bounding box` 选项后，当前路径就作为整个图形的边界盒子，之后的路径不计入边界盒子。命令 `\pgfresetboundingbox` 通常与 `\pgfusepath{use as bounding box}` 配合使用。

Left  right.

```

Left
\begin{pgfpicture}
  \pgfpathrectangle{\pgfpointorigin}{\pgfpoint{2ex}{1ex}}
  \pgfusepath{use as bounding box} % 将矩形用作边界盒子
  \pgfpathcircle{\pgfpointorigin}{2ex}
  \pgfusepath{stroke}
\end{pgfpicture}
right.

```

100 定义新的箭头

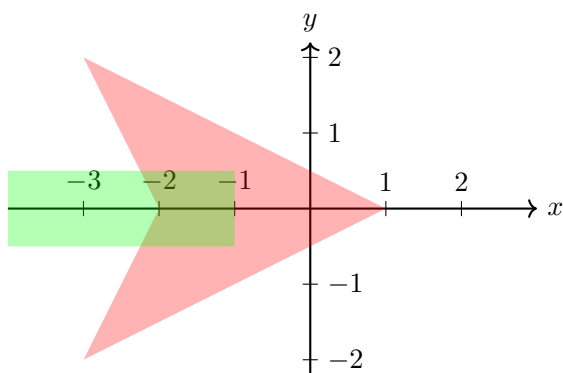
100.1 Overview

§16 介绍了 TikZ 中箭头的用法，不过 TikZ 并不对箭头做任何事情，处理箭头的是 PGF. 在 PGF 中，箭头是“meta-arrows”，就像在 $\text{T}_\text{E}\text{X}$ 中字体是“meta-fonts”. 一个 meta-font 并不是一种通常意义上的具体的字体，实际上它是个“蓝图” (blueprint)，或者说是一组程序，它能将实际输出的符号调整到指定的尺寸、形态。举例说，“Berlin”是正常尺寸 (10pt) 的单词，而“**Berlin**”是 tiny 尺寸 (5pt) 的单词放大 1 倍后的效果，显然，同一个单词从 tiny 尺寸变到正常尺寸，不是简单地放大 1 倍。

PGF 的 meta-arrows 的作用也是类似的。箭头的形态受到多个参数的影响，其中添加箭头的路径的线宽是个因素。5pt 线宽路径的箭头的尺寸，并不是 1pt 线宽路径的箭头尺寸的 5 倍，而是小于 5 倍。你可以想象实际的处理过程比较复杂。

100.2 有关术语

路径和路径上的箭头有几个特征点，参照下图：



上图的意思是给路径 $(-4, 0) \rightarrow (1, 0)$ 的端点加箭头。定义箭头时需要用绘图代码规定箭头的轮廓，这些绘图代码所在的坐标系叫做“箭头坐标系” (arrow tip’s coordinate system)。程序先开启箭头坐标系，按代码画出箭头，然后经过一些必要的变换，再添加到路径上，一般会使得箭头的尖端与路径原来的端点重合。注意绘制路径的代码所在的坐标系与“箭头坐标系”是不同的。上面图形中，把路径与箭头画在同一个坐标系内，只是为了方便而已。不过下面与箭头有关的“特征点”都是在“箭头坐标系”中描绘的。

- 点 (1,0) 叫作 tip end, 即箭头的尖端, 总是假定该点位于箭头坐标系的 x 轴上。
- 点 (-3,0) 叫作 back end, 总是假定该点位于箭头坐标系的 x 轴上。
- 点 (-1,0) 叫作 line end, 即路径端点, 总是假定该点位于箭头坐标系的 x 轴上。原来的端点是 (1,0), 添加箭头后路径的端点有所移动, 使得路径被裁去一段。如果不裁去一段, 因为该路径线宽是 10mm, 所以路径会突出箭头边界之外。
- 点 (-2,0) 叫作 visual back end, 总是假定该点位于箭头坐标系的 x 轴上。
- 与 visual back end 相对应, 也有一个 visual tip end, 总是假定该点位于箭头坐标系的 x 轴上。上图中 visual tip end 与 tip end 重合。
- 顶点, convex, 上图箭头的凸顶点有 3 个, (1,0), (-3,2), (-3,-2), 这三个点可以决定箭头的“尺寸”。一般情况下, PGF 会追踪画出的任何路径的边界盒子。但是 PGF 缓存 (cache) 箭头的有关代码时, 并不能确定箭头的尺寸。因此必须显式地提供箭头的凸顶点来确定箭头尺寸。

100.3 PGF 处理箭头的一般过程

1. 首先定义箭头。定义箭头的命令是 `\pgfdeclarearrow{<config>}`, 在 `<config>` 中需要设置关于箭头的 name, parameters, setup code, drawing code, 等等。这个命令只是保存箭头的定义, 并不画出箭头, 也不会处理定义代码。
2. 定义箭头后就可以使用箭头, 即给路径的端点添加箭头。继续使用前面的图形, 假设定义的箭头名称是 `name=foo`, 例如所添加的箭头是 `foo[length=5pt,open]`, 这里不仅有箭头名称, 也有箭头选项。箭头名称、箭头选项 (包括选项的值) 会被当作一个整体组合; 如果两个组合的箭头名称、箭头选项相同, 但选项的值不同, 也会被当作不同的组合。
3. 当一个组合, 例如, `foo[length=5pt,open]`, 第一次被使用时, PGF 会检查所保存的箭头定义并执行定义中的 setup code, 使得选项被处理为相应的值 (目前的这个例子中, 长度的值被处理为 5pt)。此时 setup code 有两个作用: 第一, 计算箭头的特征点; 第二, 为画出箭头做准备性的计算。setup code 并不画出箭头, 只是做一些准备性的工作。对每一个箭头名称、箭头选项的组合, setup code 只执行一次。
4. 处理 drawing code, 即绘制箭头的路径命令 (在目前的例子中, 箭头只有 4 个线段)。drawing code 会被放入一个“沙盒” (sandbox) 中来执行, 其中使用底层 (basic layer) 命令来绘图, 并将绘图结果“缓存” (cache)。一旦这个缓存创建成功, 当再次出现同一个箭头名称、箭头选项的组合时 (目前例子的组合是 `foo[length=5pt,open]`), 通常就不必重复计算了。不过在两种例外情况下 drawing code 会被重复计算: 第一, 箭头定义中使用了 `cachable=false` 选项, 当 drawing code 中含有底层驱动命令不能接受的内容时 (例如含有 `\pgftext`), 就需要使用这个选项; 第二, 箭头定义中使用了 `bend` 选项, 因为需要计算路径的曲率来实现 bend 效果, 而各个路径的曲率又是不固定的, 所以需要针对各个路径来分别执行 drawing code。

因为 drawing code 可能被执行多次, setup code 可能只执行一次, 而执行 drawing code 时可能需要用到 setup code 中计算出来某些个值 (为绘制箭头做准备是 setup code 的一个任务), 这就需要用命令 `\pgfarrowssave` 来保存 setup code 中计算出来某些个值, 以供执行 drawing code。每执行 drawing code 之前, 都会先刷新保存在命令 `\pgfarrowssave` 中的值。

100.4 自定义箭头

`\pgfdeclarearrow{<config>}`

这个命令有两个用处，新定义一个箭头，以及用已有的箭头来定义一个“shorthand”。<config> 中包括 name, parameters, setup code, drawing code 等选项。下面介绍可以在 <config> 中使用的内容。首先注意在这个命令内部不能有空行，并且这个命令的有效范围受到 T_EX 分组的限制。

新定义一种箭头。 <config> 是个键值列表，其中可以使用以下键 (key)。

- `name=<name>` 或者 `name=<start name>-<end name>`

这个选项规定箭头的名称。如果在命令内多次使用这个选项，那么后面的名称会覆盖前面的名称。还要注意，在当前的 T_EX 分组内，可以多次使用这个命令定义多个箭头，后定义的箭头名称不能重复使用已有的或前面已定义的箭头名称。在 PGF 中有一个惯例，以大写字母开头的名称表示一个完备的箭头，以小写字母开头的名称用于定义 shorthand 箭头。

名称里面可以使用连字符（上面第二个形式）。

- `parameters={<list of macros>}`

这个选项的值 <list of macros> 中列出 setup code 或 drawing code 涉及的各种参数，这里的参数表达为宏的形式（见 §100.5）。例如，假设定义的箭头可以具有长度选项 `length` 和宽度选项 `width`，与这两个选项对应的宏是 `\pgfarrowlength` 和 `\pgfarrowwidth`，这两个宏会被用在 setup code 或 drawing code 中，那么就要写下

```
parameters={\the\pgfarrowlength \the\pgfarrowwidth}
```

也可以写成带有逗号分隔的形式

```
parameters={\the\pgfarrowlength, \the\pgfarrowwidth}
```

命令 `\the\pgfarrowlength` 得到的是保存在宏 `\pgfarrowlength` 中的值。

<list of macros> 会被放入一个 `\csname-\endcsname` 组合中。

线宽参数宏 `\pgflinewidth` 和内线线宽参数宏 `\pgfinnerlinewidth` 通常不放在这里的 <list of macros> 中，这两个宏可以直接用在 setup code 或 drawing code 中。

- `setup code={<code>}`

当使用箭头时，保存在 parameters 中的各个参数宏会被展开，程序会检查这些参数宏的值是否遇到过。如果没有遇到过就执行这里的 <code>，并且仅执行一次。在 <code> 中需要明确指定箭头的特征点，并且可以为绘制箭头（即执行 drawing code）做准备性计算，这需要在 <code> 中使用下面的宏。

`\pgfarrowssettipend{<dimension>}`

这个宏指定箭头的尖端的位置。因为总是假定箭头的 tip end 位于箭头坐标系的 x 轴上，所以这里的尺寸 <dimension> 确定箭头坐标系的 x 轴上的一个点，该点作为箭头的尖端。对于目前的例子，应该写下

```
\pgfarrowssettipend{1cm}
```


注意这里的 `<dimension>` 并不用命令 `\pgfmathsetlength` 来解析,而是执行 `\pgf@x=<dimension>`. 这里的 `<dimension>` 可以是一个算式, 不过算式要以数字开头, 例如

```
\pgfarrowssettipend{1cm\advance\pgf@x by-.5\pgflinewidth}
```

如果 setup code 中没有明确给出这个宏, 就默认 `<dimension>` 等于 `0pt`, 即箭头的尖端在箭头坐标系的原点。

`\pgfarrowssetbackend{<dimension>}`

规定箭头的 back end, 对于目前的例子就是

```
\pgfarrowssettipend{-3cm}
```

如果 setup code 中没有明确给出这个宏, 就默认 `<dimension>` 等于 `0pt`, 即箭头的 back end 在箭头坐标系的原点。

`\pgfarrowssetlineend{<dimension>}`

设置 line end, 如果 setup code 中没有明确给出这个宏, 就默认 `<dimension>` 等于 `0pt`, 即路径的 line end 在箭头坐标系的原点。

对于目前的例子就是

```
\pgfarrowssettipend{-1cm}
```

`\pgfarrowssetvisualbackend{<dimension>}`

设置 visual back end, 如果 setup code 中没有明确给出这个宏, 就默认该点与 back end 重合。

对于目前的例子就是

```
\pgfarrowssetvisualbackend{-2cm}
```

`\pgfarrowssetvisualtipend{<dimension>}`

设置 visual tip end, 如果 setup code 中没有明确给出这个宏, 就默认该点与 tip end 重合。

`\pgfarrowshullpoint{<x dimension>}{<y dimension>}`

这个宏设置箭头的凸顶点。程序会执行 `\pgf@x=<x dimension>` 和 `\pgf@y=<y dimension>`, 其中 `<x dimension>` 和 `<y dimension>` 都可以是算式 (以数字开头)。

对于目前的例子就是

```
\pgfarrowshullpoint{1cm}{0pt}
\pgfarrowshullpoint{-3cm}{2cm}
\pgfarrowshullpoint{-3cm}{-2cm}
```

`\pgfarrowsupperhullpoint{<x dimension>}{<y dimension>}`

如果 `<y dimension>` 大于 0, 那么这个宏等效于

```
\pgfarrowshullpoint{<x dimension>}{<y dimension>}
\pgfarrowshullpoint{<x dimension>}{-<y dimension>}
```

也就是说, 这个宏能同时设置两个关于 x 轴对称的凸顶点。但如果箭头带有 `harpoon` 选项, 则没有第二个凸顶点。

如果 `<y dimension>` 不大于 0, 则这个宏等效于

```
\pgfarrowshullpoint{<x dimension>}{<y dimension>}
```

`\pgfarrowssave{<macro>}`

前面已经提到,setup code 中的某个参数宏可能用于 drawing code,这个参数宏应该另存在 <macro> 中。例如

```
\pgfarrowssave{\pgf@x}
\pgfarrowssave{\pgf@y}
```

这个形式不太直观,可以使用下面的宏。

`\pgfarrowssavethe{<register>}`

作用与 `\pgfarrowssave` 类似。这里 <register> 是寄存器(见 §100.5),寄存器的值 `\the<register>` 会被保存起来。例如

```
\pgfarrowssavethe{\pgfarrowlength}
\pgfarrowssavethe\pgfarrowwidth
```

使用这个宏后,要注意在 drawing code 中正确使用 <register>。

- **drawing code={<code>}**

<code> 是绘制箭头的路径命令,这些命令都应该是底层 (basic layer) 的命令,其中可以使用在 setup code 中引入的各个参数宏,以及线宽参数宏。

程序会开启箭头坐标系来绘制箭头。PGf 会对绘制的箭头做画布变换或者其它变换,并添加到路径端点处。

在 <code> 中需要注意:

- 在 <code> 中不要使用 `\pgfusepath` 命令,此命令会试图给箭头添加箭头,导致一种递推循环。你可以使用“quick”版的命令,参考 §112,例如
- ```
\pgfusepathqstroke
```
- 使用这种命令不会出现给箭头添加箭头的情况。
- 如果在 <code> 中使用 `\pgfusepathqstroke`,你可能需要先设置线型为实线,并设置线冠和线结合的样式。
  - 当 <code> 被执行时,<code> 会被放入一个底层的域 (scope) 中。
  - 当第一次执行 <code> 时,会使用高层的坐标变换矩阵。

- **cache=<true or false>**

这个选项的默认值是 true,此时,对于每个箭头、选项组合只执行 <code> 一次,并将执行结果缓存,重复使用。但是如果 drawing code 中含有不能缓存的代码,例如 `\pgftext`,应当设置 cache=false。

- **bending mode=<mode>**

当给路径添加箭头时,箭头可能带有 `bend` 选项,即要求箭头随着路径曲线的弯曲而弯曲。这个 key 决定箭头的弯曲方式。

如果箭头不需要弯曲,应当设置 `bending mode=none`,此时默认使用 `flex` 选项,见 §16.3.8。

如果箭头需要弯曲，那么至少有两种选择，orthogonal（正交的）或者 polar（极坐标的），见 §103.4.7. 这个选项的默认值是 `bending mode=orthogonal`.

- `defaults=<arrow keys>`

这个选项设置关于箭头的默认选项值，例如

```
defaults={length=4cm, width=2cm}
```

`<arrow keys>` 会在其它选项之前被执行。

对于前面的例子来说，可以用下面的代码来实现。

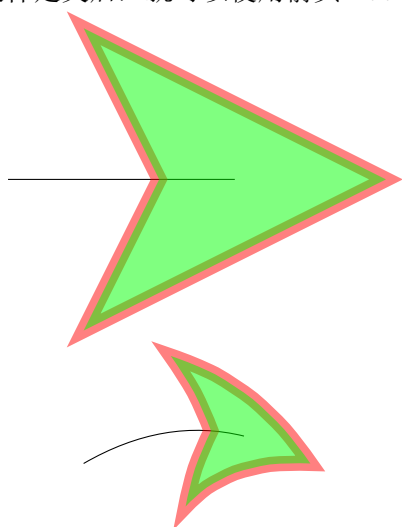
```
\pgfdeclarearrow{
 name = foo, % 箭头名称为 foo
 parameters = { \the\pgfarrowlength }, % 只有一个长度参数宏
 setup code = {
% 用长度宏指定特征点
 \pgfarrowssettipend{.25\pgfarrowlength}
 \pgfarrowssetlineend{-.25\pgfarrowlength}
 \pgfarrowssetvisualbackend{-.5\pgfarrowlength}
 \pgfarrowssetbackend{-.75\pgfarrowlength}
% 指定凸顶点
 \pgfarrowsshullpoint{.25\pgfarrowlength}{0pt}
 \pgfarrowsshullpoint{-.75\pgfarrowlength}{.5\pgfarrowlength}
 \pgfarrowsshullpoint{-.75\pgfarrowlength}{-.5\pgfarrowlength}
% 另存长度宏
 \pgfarrows.savethe\pgfarrowlength
 },
% 绘制箭头路径的底层命令
 drawing code = {
 \pgfpathmoveto{\pgfpoint{.25\pgfarrowlength}{0pt}} % 使用坐标命令 \pgfpoint
 \pgfpathlineto{\pgfpoint{-.75\pgfarrowlength}{.5\pgfarrowlength}}
 \pgfpathlineto{\pgfpoint{-.5\pgfarrowlength}{0pt}}
 \pgfpathlineto{\pgfpoint{-.75\pgfarrowlength}{-.5\pgfarrowlength}}
 \pgfpathclose
 \pgfsetstrokecolor{red} % 箭头边界路径颜色为红色
 \pgfsetlinewidth{2mm}
 \pgfsetstrokeopacity{.5}
 \pgfusepathqstroke % 画出箭头边界路径
 \pgfpathmoveto{\pgfqpoint{.25\pgfarrowlength}{0pt}} % 使用坐标命令 \pgfqpoint, 与 \
 pgfpoint 等效
 \pgfpathlineto{\pgfqpoint{-.75\pgfarrowlength}{.5\pgfarrowlength}}
```

```

\pgfpathlineto{\pgfqpoint{-.5\pgfarrowlength}{0pt}}
\pgfpathlineto{\pgfqpoint{-.75\pgfarrowlength}{-.5\pgfarrowlength}}
\pgfpathclose
\pgfsetfillcolor{green} % 箭头填充颜色为绿色
\pgfsetfillopacity{.5}
\pgfusepathqfill % 填充箭头
},
% 设置长度选项的默认值
defaults = { length = 4cm }
}

```

这样定义后，就可以使用箭头 `foo` 了，如下：



```

\tikz\draw[-{foo[color=blue]}] (0,0)--(5,0);
% 这里的选项 color=blue 没有作用

```

```

\tikz \draw [-{foo[length=2cm,bend]}]
(0,0) to [bend left] (3,0);

```

定义一个 **Shorthand** 箭头。可以用已有的箭头来定义一个 Shorthand 箭头,用到命令 `\pgfdeclarearrow`, 以及该命令的选项 `name` 和 `means`.

```

\pgfdeclarearrow{name=<shorthand arrow name>, means=<end arrow specification>}

```

选项 `name=<shorthand arrow name>` 指定所定义的 shorthand 箭头的名称。

选项 `means=<end arrow specification>` 规定 shorthand 箭头所代表的箭头样式。例如

```

\pgfdeclarearrow{ name=goo, means = bar[length=2cm+\mydimen] }

```

当给路径添加箭头 `goo` 时，实际添加的箭头是代码 `bar[length=2cm+\mydimen]` 规定的箭头。

`<end arrow specification>` 指定的是路径末端的箭头样式。如果 shorthand 箭头用于路径始端，那么所添加的箭头是末端的 `<end arrow specification>` 样式的翻转，具体参考 §16.4.4.

程序读取选项 `means=<end arrow specification>` 时，`<end arrow specification>` 会被立即执行并缓存。



```

\pgfdeclarearrow{ name=goo, means = {Bar[length=0.5cm]} }
\tikz \draw [-{goo[bend,color=red]}]
(0,0) to [bend left] (2,0);

```

## 100.5 关于箭头的选项

给路径添加箭头时，箭头可能会带有选项，即 key，这些选项影响箭头的外观、特征尺寸。在定义箭头的命令 `\pgfdeclarearrow` 中，parameters, setup code, drawing code 都会涉及参数宏，这些参数宏与箭头选项之间有对应关联。当使用选项时，例如使用长度选项 `foo[length=1cm]`，选项的值 1cm 会被赋予宏 `\pgfarrowslength`，从而影响对 parameters, setup code, drawing code 的计算结果。

### 100.5.1 尺寸选项

有的箭头选项，如 length, width, 会设定某个 T<sub>E</sub>X 寄存器的值。例如，与 length 对应的 T<sub>E</sub>X 寄存器是 `\pgfarrowslength`，length 的值会变成 `\pgfarrowslength` 的值。

- 寄存器 `\pgfarrowslength` 对应选项 length 和 angle.
- 寄存器 `\pgfarrowswidth` 对应选项 width, width' 和 angle.
- 寄存器 `\pgfarrowsinset` 对应选项 inset 和 inset'.
- 寄存器 `\pgfarrowslinewidth` 对应选项 line width 和 line width'.

如果 setup code 或 drawing code 中用到了某个寄存器（宏），或者希望在箭头选项中使用相应的选项，就应当把相应的寄存器（宏）写入 parameters 的列表中。如前述，格式如

```
parameters={\the\pgfarrowlength \the\pgfarrowwidth} 或者
parameters={\the\pgfarrowlength, \the\pgfarrowwidth}
```

### 100.5.2 True-False 选项

有的箭头选项的值是 true 或 false，例如选项 reversed, 它们都设定某个对应的 T<sub>E</sub>X-if 条件判断宏的值。

- `\ifpgfarrowsreversed` 对应选项 reversed.
- `\ifpgfarrowswap` 对应选项 swap 和 right.
- `\ifpgfarrowsharpoon` 对应选项 harpoon, left 和 right.
- `\ifpgfarrowsroundcap` 的 true 值对应选项 line cap=round,round,其 false 值对应选项 line cap=butt, sharp.
- `\ifpgfarrowsroundjoin` 的 true 值对应选项 line join=round, round, 其 false 值对应选项 line join=miter, sharp.
- `\ifpgfarrowsopen` 的 true 值对应选项 fill=none, open, 其 false 值对应选项 fill=<color>, color.

如果 setup code 或 drawing code 中用到了某个 T<sub>E</sub>X-if 条件判断宏，或者希望在箭头选项中使用相应的选项，就应当把相应的宏写入 parameters 的列表中。格式是

```
parameters = { \the\pgfarrowlength,...,
 \ifpgfarrowharpoon h\fi\
 \ifpgfarrowroundjoin j\fi}
```

在上面的格式中 `\ifpgfarrowharpoon h\fi` 的作用是，若 `\ifpgfarrowharpoon` 的值为 true，则得到一个字符 h，或者说 `\ifpgfarrowharpoon` 对应字符 h。

注意在这个格式中，不同的  $\TeX$ -if 条件判断宏应该对应不同的字符。

选项 `reversed` 的作用是通过（在箭头坐标系中）将箭头做关于 y 轴对称的变换实现的；`swap` 的作用是通过（在箭头坐标系中）将箭头做关于 x 轴对称的变换实现的。

当选项 `reversed` 的作用实现后，原箭头的各种特征点的横坐标都变成相反数，原箭头的 back end 变成新箭头的 tip end，原箭头的 tip end 变成新箭头的 back end，原箭头的 visual back end 变成新箭头的 visual tip end，原箭头的 visual tip end 变成新箭头的 visual back end，但是注意，line end 也变成原值的相反数，这可能导致路径线宽突出箭头轮廓之外。因此在定义箭头时，要仔细考虑各种情况，合理设置各个特征点的位置，此时，你可能需要（在 setup code 中）用一个条件句，根据 `\ifpgfarrowreversed` 的不同值来分别设置 line end 的位置。

### 100.5.3 setup code 中不能引用的选项

有的选项影响箭头的外观，但不能在 setup code 中列出。

- `quick`, `flex`, `flex'`，这些选项会影响箭头的旋转，见 §16.3.8。
- `color`, `fill=<color>`.
- `sep`.

### 100.5.4 自定义箭头选项

预定义的箭头选项已经很多了，但有时你可能需要自己设计某个箭头选项来方便地实现某种效果。

假设你需要定义一个选项 `depth`，那么你应当引入一个寄存器或者宏来保存该选项的值，例如

```
\newdimen\pgfarrowdepth % 定义一个宏，该宏占用一个尺寸寄存器
```

而且你还需要用命令 `\pgfkeys` 定义一个 key:

```
/pgf/arrow keys/depth
```

为了使得选项 `depth` 能够修改宏 `\pgfarrowdepth` 的值，并且能让这个宏对箭头的位置、形态产生影响，需要仔细地设置一些代码。

前面提到了对于 setup code 和 drawing code 的缓存，而这里有“选项缓存”，即对于每一组“选项、选项值”，如 `foo[<options>]`，`<options>` 只被执行一次，并将执行结果缓存起来。PGF 的 key 操作的功能很强，但所需的代价也较高，所以应尽量减少 key 操作的次数。

为了使得某些代码能进入选项缓存，需要使用下面的命令：

```
\pgfarrowsaddtooptions{<code>}
```

例如对于前面需要定义的选项 `depth`，可以设置

```
\pgfkeys{/pgf/arrow keys/depth/.code=
```

```
\pgfarrowsaddtooptions{\pgfmathsetlength{\pgfarrowdepth}{#1}}
```

这样设置后，每当使用选项 `depth` 时，程序都会执行 `\pgfmathsetlength` 来为 `\pgfarrowdepth` 赋值（见 §89.1.1），这样选项 `depth` 与宏 `\pgfarrowdepth` 就对应起来了。

不过执行命令 `\pgfmathsetlength` 所需的代价较高，为了减少执行这个命令的次数，可以使用下面的代码：

```
\pgfkeys{/pgf/arrow keys/depth/.code=
 \pgfmathsetlength{\somedimen}{#1}
 \pgfarrowsaddtooptions{\pgfarrowdepth=\somedimen}
```

其中的 `\somedimen` 应该是某个寄存器。这样设置的想法是：只执行一次命令 `\pgfmathsetlength`，把两个寄存器宏 `\pgfarrowdepth` 和 `\somedimen` 联系起来，然后就可以绕过高代价命令 `\pgfmathsetlength`，直接在两个寄存器宏之间进行操作，运行起来就会快一些。

尽管这样的想法很好，但是这样的设置仍然不可用，因为在选项缓存后，宏 `\somedimen` 的值还可以变化。为此可以使用下面的代码：

```
\pgfkeys{/pgf/arrow keys/depth/.code=
 \pgfmathsetlength{\somedimen}{#1}
 \expandafter\pgfarrowsaddtooptions\expandafter{\expandafter\pgfarrowdepth\
 expandafter=\the\somedimen}
```

### `\pgfarrowsaddtolateoptions{<code>}`

这个命令类似上面的 `\pgfarrowsaddtooptions`，只是本命令的 `<code>` 具有 `late` 特征，即 `<code>` 会在其它（没有 `late` 特征的）选项（命令）都被处理过之后才被执行。有的箭头选项，例如 `width'`，该选项的使用格式是

```
width'=<dimension> <length factor> <line width factor>
```

其中的 `<length factor>` 是要用与箭头长度 `length` 相乘的，因此，如果存在这个 `<length factor>`，那么在程序处理箭头长度 `length` 的值之前（当然应当规定默认长度值），选项 `width'` 是不能被执行的。所以，选项 `width'` 就在其它（没有 `late` 特征的）选项（包括 `length`）被执行之后才被执行。

所以，对于没有 `late` 特征的选项最好指定其默认值，这样在定义具有 `late` 特征的选项时就可引用这些选项对应的宏。

### `\pgfarrowsaddtolengthscalelist{<dimension register>}`

这个命令会把寄存器 `<dimension register>` 与箭头选项 `scale length` 规定的数值相乘，即改变寄存器 `<dimension register>` 的值，然后再用该寄存器绘制箭头。

这个命令只能用于导言中。

### `\pgfarrowsaddtowidthscalelist{<dimension register>}`

这个命令与上面的 `\pgfarrowsaddtolengthscalelist` 类似，会把寄存器 `<dimension register>` 与箭头选项 `scale width` 规定的数值相乘，即改变寄存器 `<dimension register>` 的值，然后再用该寄存器绘制箭头。

这个命令只能用于导言中。

**`\pgfarrowsthreeparameters{<line-width dependent size specification>}`**

这里 `<line-width dependent size specification>` 是一个，或两个，或三个数值，数值之间用空格分隔。本命令会产生三个花括号，把这些数值依次分别放入花括号内，缺失的数值用 0 代替，然后将这个结果保存在宏 `\pgfarrowstheparameters` 中。

例如，执行命令 `\pgfarrowsthreeparameters{2pt 1}` 后，保存在 `\pgfarrowstheparameters` 中的值就是 `{2.0pt}{1}{0}`。

**`\pgfarrowslinewidthdependent{<dimension>}{<line width factor>}{<outer factor>}`**

这个命令用于定义类似箭头选项 `length` 那样的选项，`length` 的使用格式是

```
length=<dimension> <line width factor> <outer factor>
```

其意思是长度 `length` 的值依赖这里列出的三个参数（参考 §16.3.1）。程序参照这三个参数计算出一个尺寸，将这个尺寸作为 `length` 的值。

本命令也参照这三个参数计算出一个尺寸，并将这个尺寸保存在寄存器 `\pgf@x` 中。

选项 `length` 的定义是（见文件 `pgflibraryarrows.meta.code`）

```
\pgfkeys{
 /pgf/arrow keys/.cd,
 length/.code={%
 \pgfarrowsthreeparameters{#1}%
 \expandafter\pgfarrowstheparameters\expandafter{\expandafter\
 pgfarrowslinewidthdependent\pgfarrowstheparameters\pgfarrowlength\pgf@x}%
 },

}
```

仿照这个定义，可以定义依赖那三个参数的选项 `depth`，代码如下：

```
\pgfkeys{/pgf/arrow keys/depth/.code={%
 \pgfarrowsthreeparameters{#1}%
 \expandafter
 \pgfarrowstheparameters % 使用有 late 特征的命令
 \expandafter{%
 \expandafter
 \pgfarrowslinewidthdependent % 将下一行的宏作为参数，计算结果保存在 \pgf@x 中
 \pgfarrowstheparameters % 引用前面命令的结果，即三个数值
 \pgfarrowdepth\pgf@x % 将 \pgf@x 的值赋予 \pgfarrowdepth
 }%
}
```

**`\pgfarrowslengthdependent{<dimension>}{<length factor>}{<dummy>}`**

这个命令用于定义类似箭头选项 `width` 那样的选项，本命令也参照这三个参数计算出一个尺寸，并将



这个尺寸保存在寄存器 `\pgf@x` 中。

例如可以定义选项 `depth'`，使得这个选项的值依赖这三个参数：

```
\pgfkeys{/pgf/arrow keys/depth' /.code={%
 \pgfarrowsthreeparameters{#1}%
 \expandafter
 \pgfarrowsaddtolateoptions
 \expandafter{%
 \expandafter
 \pgfarrowslengthdependent
 \pgfarrowstheparameters%
 \pgfarrowdepth\pgf@x%
 }%
}
```

## 101 Nodes and Shapes

本节介绍 `shapes` 模块。

```
\usepgfmodule{shapes} % LATEX and plain TEX and pure pgf
```

这个模块包含的命令可以创建 `nodes` 和 `shapes`。PGF 会自动加载这个模块，但如果你只是使用内核 `pgfcore`，你也可以手工加载这个模块。

### 101.1 Overview

PGF 有创建 `node` 和 `shape` 的完善的机制。一个 `node` 通常就是一个 `shape` 加 `text`，所谓 `shape` 一般是个路径，例如 `rectangle` 或 `circle`。

#### 101.1.1 创建与索引 `node`

你可以用命令 `\pgfnode` 或 `\pgfmultipartnode` 来创建 `node`，并且在当前的 `{pgfpicture}` 环境中，程序会记住所创建的 `node`，因此在当前绘图环境中，创建一个 `node` 后你可以引用它。创建 `node` 的命令也支持坐标变换。

#### 101.1.2 锚 `anchors`

`node` 和 `shape` 都有“锚”（`anchors`），即属于 `node` 或 `shape` 上的点，或者说是“位置”、“部位”。锚位置 `text` 是 `node` 的文字的左下角。当指定锚位置后，这个锚位置会处于 `node` 的指向点上。

锚位置接受坐标变换。

### 101.1.3 shape 的“层” Layers

最简单的 shape 是 coordinate，它只有一个锚位置，即 `center`，它的文字内容通常是空的。一个 shape 通常有好几个“层”，例如，当填充一个 node 时，填充色位于文字之下，文字不会被填充色遮挡，因为填充色与文字处于不同的层上。

一个 shape 通常有 7 个层：

1. behind the background layer，画出或者填充 shape 路径的命令通常处于这一层上。
2. background path layer
3. before the background path layer
4. label layer，盛放 node 的文字的盒子 (box) 处于这一层上。
5. behind the foreground layer
6. foreground path layer，当画出文字后，再画出这一层的内容。
7. before the foreground layer

### 101.1.4 Node Parts

一个 shape 或 node 可以带有文字，文字会被放入一个盒子内。多数 shape 或 node 只有一个文字盒子，文字的左下角会处于的锚位置 `text` 上。有的 shape 或 node 有多个文字盒子，这样它的文字就分成了数个部分，或者说它分成了数个部分，这些部分叫作“node parts”，有数个 node parts 的就称为 multipart shape 或 multipart node。

定义一个 multipart shape 时，每个 node part 都定义自己的名称，每个部分的文字都会被放入一个盒子中，每个 node part 都应该定义一个锚位置来放置文字盒子。node part 的名称与盒子的名称是对应的，例如，某个 node part 的名称是 `XYZ`，那么该部分中盛放文字的盒子名称就应该是 `\pgfnodepartXYZbox`，这个盒子会参照该部分的某个指定的锚位置来放置。

## 101.2 创建 node

### 101.2.1 创建简单 node

```
\pgfnode{<shape>}{<anchor>}{<label text>}{<name>}{<path usage command>}
```

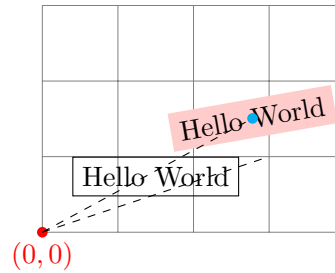
这个命令创建只有一个 node part 的 node。

其中 `<shape>` 是已经定义的某个 shape 的名称，`<anchor>` 是 `<shape>` 的一个锚位置。本命令会把锚位置 `<anchor>` 放在原点上，如果你想把锚位置 `<anchor>` 放在其它点上，就需要在本命令之前使用坐标变换命令。

`<label text>` 是 node 的文字，文字会被放入 T<sub>E</sub>X 盒子 `\pgfnodeparttextbox` 中。

`<name>` 是所创建的 node 的名称，用于之后索引该 node。如果没有 `<name>`，那么在画出该 node 之后，程序就会“忘记”这个 node。

`<path usage command>` 是使用路径的命令，对 background path 或者 foreground path 进行操作，例如，画出路径、填充路径等。

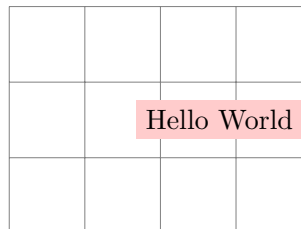


```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (4,3);
\fill [red] circle (2pt) node [below] {$(0,0)$};
{
\pgftransformshift{\pgfpoint{1.5cm}{1cm}}
\pgfnode{rectangle}{north}{Hello World}{hellonode}{\pgfusepath{stroke}}
}
{
\color{red!20}
\pgftransformrotate{10}
\pgftransformshift{\pgfpoint{3cm}{1cm}}
\pgfnode{rectangle}{center}
{\color{black}Hello World}{hellonode}{\pgfusepath{fill}}
}
\fill [cyan] (hellonode.center) circle (2pt);
\draw [dashed] (0,0)--(3,1);
\draw [dashed,rotate=10] (0,0)--(3,1);
\end{tikzpicture}

```

从上面的例子看出,坐标变换对 shape 和文字都有效。node 的指向点是原点,当用坐标变换改变 node 的指向点的位置时,node 的位置也会随之变化。如果还有旋转变换,node 在整体上也会被旋转。如果不希望旋转变换对 node 起作用,就需要在命令 `\pgfnode` 之前使用命令 `\pgftransformresetnontranslations`。



```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (4,3);
{
\color{red!20}

```

```

\pgftransformrotate{10}
\pgftransformshift{\pgfpoint{3cm}{1cm}}
\pgftransformresetnontranslations
\pgfnode{rectangle}{center}
 {\color{black}Hello World}{hellonode}{\pgfusepath{fill}}
}
\end{tikzpicture}

```

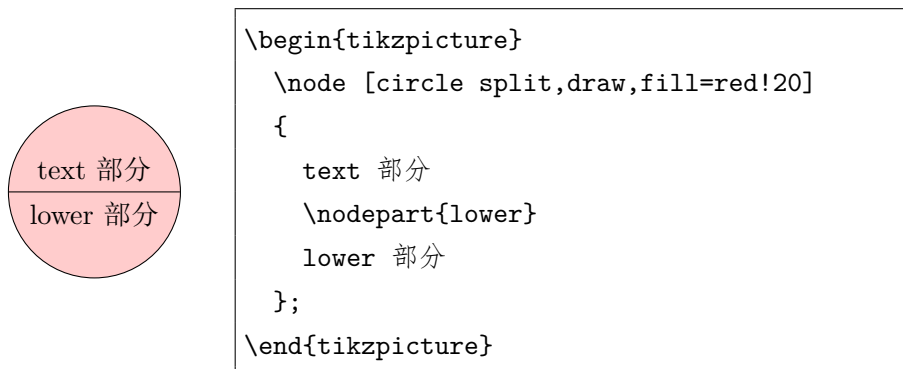
### 101.2.2 创建 Multi-Part Nodes

`\pgfmultipartnode{<shape>}{<anchor>}{<name>}{<path usage command>}`

这个命令是 `\pgfnode` 的推广，可以创建 multipart node。

创建 multipart node 时，首先你要使用具有多个部分的 shape，参考 S67.6。因为每一部分文字都会被放入一个  $\text{T}_\text{E}\text{X}$  盒子中，所以在使用本命令之前，你需要先定义数个盒子，用来盛放各个部分的文字。

以预定义的 circle split（见 §67.6）为例，这个 shape 是圆形的，一个水平的直径将这个圆分为上下两个 node part，即 text 部分和 lower 部分，如下所示：



注意这两个 node part 的名称分别是 `text` 和 `lower`，因此与这两个名称相对应，预定义两个盒子：

```

\pgfnodeparttextbox
\pgfnodepartlowerbox

```

第一个盒子放在 `text` 部分，第二个盒子放在 `lower` 部分。这两个盒子内可以盛放  $\text{T}_\text{E}\text{X}$  盒子能接受的任何内容，例如文字，表格环境，数学公式等等。

circle split 的上下两个部分分别定义了锚位置 `text` 和 `lower`，参考 §67.6，上部文字盒子的左下角放在锚位置 `text` 上，下部文字盒子的左下角放在锚位置 `lower` 上。



```

\setbox\pgfnodeparttextbox=\hbox{q1}
\setbox\pgfnodepartlowerbox=\hbox{01}
\begin{pgfpicture}
 \pgfmultipartnode{circle split}{center}{my state}{\pgfusepath{stroke}}
\end{pgfpicture}

```

注意：如上面的例子所示，你可以在 `{pgfpicture}` 环境之前使用命令 `\setbox` 来定义盒子。如果你在 `{pgfpicture}` 环境内使用命令 `\setbox`，那么你要用 `\pgfinterruptpath` 和 `\endpgfinterruptpath` 将命令 `\setbox` 包裹起来。

注意：T<sub>E</sub>X 的盒子寄存器的个数很少，注意节约使用。

`\pgfcoordinate{<name>}{<coordinate>}`

这个命令在坐标点 `<coordinate>` 处，创建一个形状为 `coordinate` 的、名称为 `<name>` 的 node。

`\pgfnodealias{<new name>}{<existing node>}`

`<existing node>` 是某个已创建的 node 的名称，本命令为该 node 再设置一个名称 `<new name>`，也就是说，该 node 有两个名称，任何一个都可以用来索引该 node。

`\pgfnoderename{<new name>}{<existing node>}`

`<existing node>` 是某个已创建的 node 的名称，本命令为该 node 重命名，即修改其名称为 `<new name>`，废弃原来的名称 `<existing node>`。

下面的选项影响 node 的尺寸。

`/pgf/minimum width=<dimension>` (无默认值，初始值 1pt)

`/tikz/minimum width`

这个选项设置 node 的最小宽度，即宽度可以大于但不能小于 `<dimension>`。

注意这个选项的初始值是 1pt，并且这个选项值只是个“推荐值”，在某些情况下这个选项可能会被忽略。



```
\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,1);
 \pgfset{minimum width=3cm}
 \pgfnode{rectangle}{south west}{\color{cyan} Hello World}{}
 {\pgfsetstrokecolor{red} \pgfusepath{stroke}}
\end{tikzpicture}
```

`/pgf/minimum height=<dimension>` (无默认值，初始值 1pt)

`/tikz/minimum height`

这个选项设置 node 的最小高度。这个选项值只是个“推荐值”。

`/pgf/minimum size=<dimension>` (无默认值)

`/tikz/minimum size`

本选项同时设置 `/pgf/minimum width` 和 `/pgf/minimum height` 的值为 `<dimension>`。

`/pgf/inner xsep=<dimension>` (无默认值，初始值 0.3333em)

`/tikz/inner xsep`

这个选项在水平方向上，设置 node 的背景形状路径与文字的间距为 `<dimension>`，这个选项值只是个“推荐值”，在某些情况下这个选项可能会被忽略。

注意，这里的间距指的是路径线条的中心与文字的间距，而不是路径线条的外缘与文字的间距。

```
/pgf/inner ysep=<dimension> (无默认值, 初始值 0.3333em)
```

```
/tikz/inner ysep
```

这个选项在垂直方向上，设置 node 的边界形状路径与文字的间距为 <dimension>，这个选项值只是个“推荐值”。

```
/pgf/inner sep=<dimension> (无默认值)
```

```
/tikz/inner sep
```

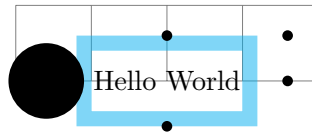
本选项同时设置 /pgf/inner xsep 和 /pgf/inner ysep 的值为 <dimension>。

```
/pgf/outer xsep=<dimension> (无默认值, 初始值 0.5\pgflinewidth)
```

```
/tikz/outer xsep
```

这个选项在水平方向上，设置 node 的背景边界形状路径与“外部锚位置”的间距。例如，如果 <dimension> 是 1cm，那么锚位置 east 与背景形状路径的边界的距离就是 1cm。这个选项值只是个“推荐值”。

注意，这里的间距指的是路径线条的中心与“外部锚位置”的间距，而不是路径线条的外缘与“外部锚位置”的间距。本选项的初始值是  $0.5\text{\pgflinewidth}$ ，这恰好使得“外部锚位置”处于路径线条的外缘上。



```
\begin{tikzpicture}
 \draw[help lines] (-2,0) grid (2,1);
 \tikzset{line width=2mm} % 这个选项是 tikz 的选项, 修改 \pgflinewidth 的值,
 \pgfset{minimum height=1cm, outer xsep=.5cm}
 \pgfnode{rectangle}{center}{Hello World}{x}
 {
 \pgfsetlinewidth{2mm} % 规定 node 形状路径线条的线宽为 2mm, 否则其线宽为默认值 0.4pt
 \pgfsetstrokecolor{cyan}
 \pgfsetstrokeopacity{0.5}
 \pgfusepath{stroke}
 }
 \pgfpathcircle{\pgfpointanchor{x}{north}}{2pt}
 \pgfpathcircle{\pgfpointanchor{x}{south}}{2pt}
 \pgfpathcircle{\pgfpointanchor{x}{east}}{2pt}
 \pgfpathcircle{\pgfpointanchor{x}{west}}{.5cm}
 \pgfpathcircle{\pgfpointanchor{x}{north east}}{2pt}
 \pgfusepath{fill}
\end{tikzpicture}
```

```
\end{tikzpicture}
```

将上面例子中的 `\pgfsetlinewidth{2mm}` 注释掉，得到下面的图形，注意比较 node 的形状路径线条的线宽：



```
\begin{tikzpicture}
 \draw[help lines] (-2,0) grid (2,1);
 \tikzset{line width=2mm} % 这个选项是 tikz 的选项，修改 \pgflinewidth 的值，
 \pgfset{minimum height=1cm, outer xsep=.5cm}
 \pgfnode{rectangle}{center}{Hello World}{x}
 {
% \pgfsetlinewidth{2mm} % 注释掉这一命令，此时 node 形状路径线条的线宽为默认值 0.4pt
 \pgfsetstrokecolor{cyan}
 \pgfsetstrokeopacity{0.5}
 \pgfusepath{stroke}
 }
 \pgfpathcircle{\pgfpointanchor{x}{north}}{2pt}
 \pgfpathcircle{\pgfpointanchor{x}{south}}{2pt}
 \pgfpathcircle{\pgfpointanchor{x}{east}}{2pt}
 \pgfpathcircle{\pgfpointanchor{x}{west}}{.5cm}
 \pgfpathcircle{\pgfpointanchor{x}{north east}}{2pt}
 \pgfusepath{fill}
\end{tikzpicture}
```

在上面两个例子中，只是设置 `outer xsep` 的值，而 `outer ysep` 的值仍然是初始值 `0.5\pgflinewidth`，通过对比可知，在确定“外部锚位置”时使用的线宽是 `\pgflinewidth=2mm`。也就是说，设置命令 `\tikzset{line width=2mm}`，对后面画圆点的命令 `\pgfpathcircle` 中的锚位置有效。但在画出 node 的形状路径线条时，用到的线宽是由 node 本身的设置决定的，与 node 之外的设置无关。

`/pgf/outer ysep=<dimension>` (无默认值，初始值 `0.5\pgflinewidth`)

`/tikz/outer ysep`

这个选项在垂直方向上，设置 node 的背景边界形状路径与“外部锚位置”的间距。这个选项值只是个“推荐值”。

注意，这里的间距指的是路径线条的中心与“外部锚位置”的间距，而不是路径线条的外缘与“外部锚位置”的间距。

`/pgf/outer sep=<dimension>` (无默认值)

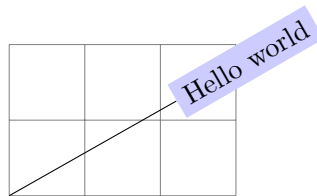
`/tikz/outer sep`

本选项同时设置 `/pgf/outer xsep` 和 `/pgf/outer ysep` 的值为 `<dimension>`。

### 101.2.3 另一种添加 node 的方法

用前面介绍的命令 `\pgfnode` 创建 node 后, node 就立即被添加到当前图形中的某个位置上, 之后就难以再改变这个 node 的位置。使用下面介绍的命令 `\pgfpositionnodelater` 可以设计并保存 node, 然后再用命令 `\pgfpositionnodenow` 在需要的位置放置保存的 node, 这样就把设计 node 的步骤与放置 node 的步骤分开了。

先看一个例子。



```

\newbox\mybox % 指定一个盒子寄存器
\def\mysaver{ % 定义宏 \mysaver
 \global\setbox\mybox=\box\pgfpositionnodelaterbox % 定义盒子 \mybox 为全局盒子
 \global\let\myname=\pgfpositionnodelatername % 定义全局命令 \myname
 \global\let\myminx=\pgfpositionnodelaterminx
 \global\let\myminy=\pgfpositionnodelaterminy
 \global\let\mymaxx=\pgfpositionnodelatermaxx
 \global\let\mymaxy=\pgfpositionnodelatermaxy
}

\begin{tikzpicture}
 { % 开启一个分组
 \pgfpositionnodelater{\mysaver} % 使用命令 \pgfpositionnodelater
 \node [fill=blue!20,below,rotate=30] (hi) at (1,0) {Hello world}; % 定义一个 node
 } % 关闭分组
 \draw [help lines] (0,0) grid (3,2);
 \let\pgfpositionnodelatername=\myname
 \let\pgfpositionnodelaterminx=\myminx
 \let\pgfpositionnodelaterminy=\myminy
 \let\pgfpositionnodelatermaxx=\mymaxx
 \let\pgfpositionnodelatermaxy=\mymaxy
 \setbox\pgfpositionnodelaterbox=\box\mybox
 \pgfpositionnodenow{\pgfqpoint{2cm}{2cm}}
 \draw (hi) -- (0,0);
\end{tikzpicture}

```

`\pgfpositionnodelater{<macro name>}`



这个命令的有效范围受到“域”（scope）的限制。在域内，本命令之后定义的所有 node 都受到本命令的作用，其作用是：不管所定义的 node 的指向点是哪个位置，都被看作是原点；node 并不被立即添加到图形中，而是被保存在盒子 `\pgfpositionnodelaterbox` 中；node 也不直接与图形的边界盒子相关联，程序仍然计算盛放 node 的边界盒子，这个盒子的上、下、左、右边界值会被保存在宏 `\pgfpositionnodelaterminx`, `\pgfpositionnodelaterminy`, `\pgfpositionnodelatermaxx`, `\pgfpositionnodelatermaxy` 之内，注意这个计算的前提是设定 node 的指向点为原点。

宏 `<macro name>` 是需要在本命令之前定义的，这个宏的定义要包含下面的宏。在定义宏 `<macro name>` 时，你需要把下面宏的内容转移到其它自定义宏中，如同前面的例子所示。

#### `\pgfpositionnodelaterbox`

目前，这个盒子寄存器的编号是 0，用来保存所定义的 node。在定义宏 `<macro name>` 时，你需要把盒子 `\pgfpositionnodelaterbox` 的内容转移到另一个你自定义的盒子中。

#### `\pgfpositionnodelatername`

定义的 node 的名称加上前缀 `not yet positionedPGFINTERNAL` 就是这个宏的内容，这个内容会被临时指定为 node 的名称。当使用命令 `\pgfpositionnodenow` 时，node 的名称会被改回原来名称。因为在使用命令 `\pgfpositionnodenow` 之前可能会引用 node 的名称，临时修改它的名称能避免混乱。

#### `\pgfpositionnodelaterminx`

node 的边界盒子的左侧边界值保存在这个宏中。这个宏的值是以 pt 为单位的尺寸。注意这个计算的前提是设定 node 的指向点为原点。

#### `\pgfpositionnodelaterminy`

#### `\pgfpositionnodelatermaxx`

#### `\pgfpositionnodelatermaxy`

注意命令 `\pgfpositionnodelater` 的有效范围受到域（绘图环境或者 TeX 分组）的限制，该命令的有效范围也是以上几个宏的存在范围，超出这个范围就没有以上几个宏了。要把 node 添加到图形中就应当使用以上几个宏，因为它们保存了 node 的相关信息。为了能在命令 `\pgfpositionnodelater` 的有效范围之外使用以上几个宏，你必须如上面的例子那样，自定义数个宏来转存以上几个宏的值。

#### `\pgfpositionnodelaterpath`

这个宏保存 node 的背景路径。参考 §115。

#### `\pgfpositionnodenow{<coordinate>}`

本命令把命令 `\pgfpositionnodelater` 保存的 node 添加到图形中，并且会按照 `<coordinate>` 来平移 node；如果 node 的定义中有类似 `at=<position>` 的位置选项，还会有参照 `<position>` 的平移，如前面的例子所示。

在定义宏 `<macro name>` 时，转存了盒子、尺寸值。在本命令之前，你需要恢复之前转存的宏及其值，因为本命令用到这几个宏及其值。

如果设置 `<macro name>=\relax`，就会取消整个机制。当用命令 `\pgfpositionnodenow` 把 node 添加

到图形上后，程序会自动设置 `<macro name>=\relax`.

`\pgfnodepostsetupcode{<node name>}{<code>}`

在命令 `\pgfpositionnodelater` 的有效范围内使用这个命令，`<code>` 会被保存，当名称为 `<node name>` 的 node 被命令 `\pgfpositionnodenow` 添加到图形上后，`<code>` 被执行。如果多次使用这个命令，那么各次的 `<code>` 会被累计。

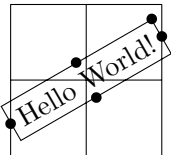
### 101.3 使用锚位置 Anchors

每个形状 (shape) 都有各种锚位置 (anchors)，通常，锚位置 `center` 会被放在指向点上。

#### 101.3.1 在一个图形中引用锚位置

`\pgfpointanchor{<node>}{<anchor>}`

这个命令指定一个坐标点，即名称为 `<node>` 的 node 的锚位置 `<anchor>`。

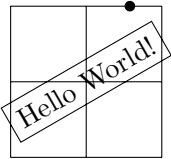


```
\begin{pgfpicture}
 \pgfpathgrid{\pgfpoint{-1cm}{-1cm}}{\pgfpoint{1cm}{1cm}}
 \pgftransformrotate{30}
 \pgfnode{rectangle}{center}{Hello World!}{x}{\pgfusepath{stroke}}

 \pgfpathcircle{\pgfpointanchor{x}{north}}{2pt}
 \pgfpathcircle{\pgfpointanchor{x}{south}}{2pt}
 \pgfpathcircle{\pgfpointanchor{x}{east}}{2pt}
 \pgfpathcircle{\pgfpointanchor{x}{west}}{2pt}
 \pgfpathcircle{\pgfpointanchor{x}{north east}}{2pt}
 \pgfusepath{fill}
\end{pgfpicture}
```

你可能觉得上面的例子有点奇怪，看上去旋转变换应当对命令 `\pgfnode` 和 `\pgfpathcircle` 都有效，首先 `\pgfnode` 接受旋转矩阵，使得 node 被旋转，从而其锚位置被旋转；之后 `\pgfpathcircle` 接受旋转矩阵，使得锚位置再次被旋转，因此那些锚位置应当脱离 node 的背景路径。但实际上那些锚位置并没有脱离 node 的背景路径，这是因为命令 `\pgfpointanchor` 会引入旋转矩阵的逆矩阵，将作用于 `\pgfpathcircle` 的旋转取消了。

下面的例子中，使用命令 `\pgftransformreset` 将命令 `\pgfpointanchor` 引入的旋转矩阵的逆矩阵变成单位矩阵，从而使得锚位置被旋转两次，脱离 node 的背景路径：



```

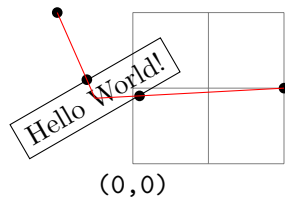
\begin{pgfpicture}
 \pgfpathgrid{\pgfpoint{-1cm}{-1cm}}{\pgfpoint{1cm}{1cm}}
 \pgftransformrotate{30}
 \pgfnode{rectangle}{center}{Hello World!}{x}{\pgfusepath{stroke}}
 {
 \pgftransformreset
 \pgfpointanchor{x}{east}
 \makeatletter
 \xdef\mycoordinate{\noexpand\pgfpoint{\the\pgf@x}{\the\pgf@y}}
 \makeatother
 }
 \pgfpathcircle{\mycoordinate}{2pt}
 \pgfusepath{fill}
\end{pgfpicture}

```

### `\pgfpointshapeborder{<node>}{<point>}`

这个命令确定一个坐标点。以名称为 `<node>` 的 node 的锚位置 `center` 为始点，做经过坐标点 `<point>` 的射线，射线与 `<node>` 的边界形状路径相交，交点就是本命令确定的点。如果 `<node>` 的边界形状路径很复杂，本命令会把这个复杂路径退化为一个相对简单的路径来计算，此时本命令确定的点可能偏离 `<node>` 的边界形状路径。

注意这个命令与 `\pgfpointanchor` 不同，这个命令不会引入变换矩阵的逆矩阵。



```

\begin{tikzpicture}
 \draw [help lines] (0,0) grid (2,2);
 \begin{pgfscope} % 用 pgfscope 环境限制旋转变换
 \pgftransformrotate{30}
 \pgftransformshift{\pgfpoint{0cm}{1cm}}
 \pgfnode{rectangle}{center}{Hello World!}{x}{\pgfusepath{stroke}}
 \end{pgfscope}
 \pgfpathcircle{\pgfpointshapeborder{x}{\pgfpoint{2cm}{1cm}}}{2pt}
 \pgfpathcircle{\pgfpoint{2cm}{1cm}}{2pt}
 \pgfpathcircle{\pgfpointshapeborder{x}{\pgfpoint{-1cm}{2cm}}}{2pt}
 \pgfpathcircle{\pgfpoint{-1cm}{2cm}}{2pt}
 \pgfusepath{fill}

```

```

\draw [red](x.center)--(2,1);
\draw [red](x.center)--(-1,2);
\node [below] {\tt(0,0)};
\end{tikzpicture}

```

### 101.3.2 跨图引用 node 的锚位置

通常一个 `{tikzpicture}` 环境或者 `{pgfpicture}` 环境创建一个图形。在一个图形中, 创建、引用 node 都是很直接地, 当一个图形创建完毕、添加到页面上后, PGF 就会忘记这个图形在页面上的位置, 之后你不能再引用这个图形中的坐标点。假设有前后两个图形, 在前一个图形中的 (3,2) 处有名称为 `<node>` 的 node, 当在后一个图形中引用前图中的 `<node>` 时, 如果程序会认为前图中的 `<node>` 位于后图中的位置 (3,2) 处, 这样就会出现错误。

如果要跨图引用 node, 即在后面的图形中引用前面图形中 node, 就需要让 PGF 记住前后两个图形在页面上的位置, 这是让两个图形在页面上联系起来的必要条件。让 PGF 记住图形在页面上的位置, 需要注意以下几点:

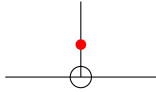
1. 使用的驱动应当支持 PGF 记住图形在页面上的位置, `pdflatex` 支持这一点, 目前 `dvips` 不支持这一点。
2. 在绘图环境中, 或者在绘图环境外的某个适当位置, 写下命令 `\pgfrememberpicturepositiononpage-true`, 这个命令将 `\ifpgfrememberpicturepositiononpage` 的值设为 `true`, 参考 §95.3.1. 如果在绘图环境中使用这个命令, PGF 会记住这个环境创建的图形在页面上的位置。如果在绘图环境外使用这个命令, PGF 会记住此命令之后创建的图形在页面上的位置。  
注意这个命令的有效范围受到 `TEX` 分组的限制。
3. 让 `TEX` 编译文件两次。第一次编译会把关于图形位置的数据写入外部文件, 而且页面可能变得很奇怪, 不过不用担心, 第二次编译会让页面变得正常。
4. 注意图形的边界盒子, 当后面的图形引用前面图形中 node 的锚位置时, 后面图形的边界盒子会包含前面图形中 node 的锚位置, 这样就使得后面图形的边界盒子很大, 两个图形几乎成为一个图形。因此在后面图形中, 在引用前面图形中 node 的锚位置之前, 需要在的适当位置使用命令 `\pgfusepath{use as bounding box}`。

### 101.4 特殊 node

下面是预定义的 node.

#### current bounding box

这个 node 的形状是 `rectangle`, 它是当前绘图环境创建的图形的边界盒子。在当前环境内, 每添加一个路径, 这个盒子就会被计算一次, 因此它的上、下、左、右界限可能不断改变。

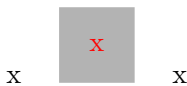


```
\tikz {
 \draw(0,0)--(2,0);
 \draw(current bounding box.center) circle (4pt)--(1,1);
 \fill [red] (current bounding box.center) circle (2pt);}
```

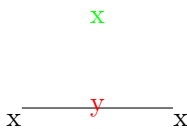
### current path bounding box

这个 node 的形状是 `rectangle`，它是当前路径的边界盒子。关于这个预定义 node 要注意两点：

1. 计算当前路径的边界盒子时不考虑线宽，只是把路径当作无宽度的曲线来计算其边界盒子。
2. 计算当前路径的边界盒子时不考虑添加到路径上的 node。



```
x\begin{tikzpicture}
 \draw [line width=10mm,draw opacity=0.3] (0,0)--(1,0)
 node[at=(current path bounding box.north)]{\color{red} x};
\end{tikzpicture}x
```



```
x\begin{tikzpicture}
 \draw (0,0)--(1,0)node[above=1cm]{\color{green}x}--(2,0)
 node[at=(current path bounding box.north)]{\color{red}y};
\end{tikzpicture}x
```

### current subpath start

这个 node 的形状是 `coordinate`，它是当前“子路径”的起点。

### current page

将当前页面假想为一个图形——一个 node，就是 `current page`，并且这个图形是“被记住”的，因此你可以在任何绘图环境中引用这个 node，只要这个绘图环境也是“被记住”的。

下面的例子中，将一段文字放在当前页面的左下角，注意要使用适当的驱动来支持“记住图形的功能”，否则文字会被插入到当前位置。

```
\pgfrememberpicturepositiononpagetrue
\begin{pgfpicture}
 \pgfusepath{use as bounding box}
 \pgftransformshift{\pgfpointanchor{current page}{south west}}
 \pgftransformshift{\pgfpoint{1cm}{1cm}}
 \pgftext[left,base]{
 \textcolor{red}{
 Text absolutely positioned in
 the lower left corner.}
```

Text absolutely positioned in the lower left corner.

```

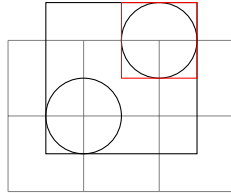
}
\end{pgfpicture}

```

```
/pgf/local bounding box=<node name>
```

```
/tikz/local bounding box
```

这个选项用作子环境 `{scope}` 的环境选项，将该子环境创建的图形的边界盒子作成一个 node，其名称为 `<node name>`。这个选项会让程序跟踪子环境的边界盒子，当需要计算的边界盒子太多时，可能会让  $\TeX$  的处理速度变慢。



```

\begin{tikzpicture}
 \draw [help lines] (0,0) grid (3,2);
 { [local bounding box=outer box] % 这个分组开启一个 scope 环境
 \draw (1,1) circle (.5) [local bounding box=inner box] (2,2) circle (.5);
 }
 \draw (outer box.south west) rectangle (outer box.north east);
 \draw[red] (inner box.south west) rectangle (inner box.north east);
\end{tikzpicture}

```

## 101.5 定义新的 shape

预定义的 shape 只有三个，即 `rectangle`, `circle`, `coordinate`，见 §67.2。另外，程序库 `shapes.symbols`, `shapes.geometric`, `shapes.callouts`, `shapes.misc`, `shapes.arrows`, `shapes.multipart` 等还定义了大量的 shape。

本节介绍自定义 shape 的方法，这个方法显然比较复杂，因为一个 shape 可以用作 node 的形状，其尺寸要有弹性以容纳各种文字，还要有各种锚位置。PGF 应当有能力处理包含数百个 node 的图形，也应当有能力处理包含数千个 node 的文档。但是，不能让 PGF 计算并记住所有 node 的各种锚位置。

### 101.5.1 一个 shape 具备的要素

定义 shape 时可能要给出以下要素：

- 名称。
- 计算 saved anchors 和 saved dimensions 的代码。
- 用 saved anchors 计算锚位置的代码。
- 画出 background path 和 foreground path 的代码，可以没有这个内容。
- 在画出 background path 和 foreground path 之前（之后）需要执行的代码，可以没有这个内容。
- 规定 node parts，可以没有这个内容。

在定义箭头时，需要指定箭头的特征点、箭头路径，这些点和路径都需要在一个坐标系——箭头坐标系——内指定。当在路径上添加箭头时，程序首先在箭头坐标系内生成箭头，然后通过变换将箭头放到路径的端点处。

情况对于 shape 的也是类似的。定义 shape 时，需要指定它的 background path, foreground path, 各个锚位置，这些路径和点都需要在一个坐标系——shape 坐标系——内指定。当用命令 `\pgfnode` 或 `\pgf-multipartnode` 调用 shape 时，程序首先在 shape 坐标系内生成它，然后通过变换添加到图形中。

### 101.5.2 Normal Anchors 与 Saved Anchors

我们将锚位置 (anchors) 分为 Normal Anchors 与 Saved Anchors 两类。例如，对于形状 `rectangle` 来说，其右上角位置是 `northeast`，这是个 normal anchor，通常用在绘图命令中；而在 `rectangle` 的定义中有宏 `\northeast`，这是个 saved anchor，当读取 `rectangle` 的定义时，这个锚位置会被计算并保存。

通常 saved anchors 会被计算并保存，而对于 normal anchors 来说，只有在用到它们时才会按照代码将其计算出来，这样会节省存储空间。在计算 normal anchors 时，可以使用各种坐标点命令，也可以利用已保存的 saved anchors，例如，形状 `rectangle` 只有两个 saved anchors，即右上角 `\northeast` 和左下角 `\southwest`，利用这两个点可以计算出矩形上的其它点。坐标点 `\southwest` 的 x 分量与 `\northeast` 的 y 分量可以组合成锚位置 `north west`，这是个 normal anchor。目前在 `rectangle` 上定义了 13 个 normal anchor，都是利用两个 saved anchors 定义的。

所有的锚位置(包括 saved anchors 和 normal anchors)都在“局部形状坐标空间”(local shape coordinate space)中指定，命令 `\pgfnode` 会自动把坐标变换应用于这个空间中的坐标点。

### 101.5.3 定义新 shape 的命令

`\pgfdeclareshape{<shape name>}{<shape specification>}`

本命令定义一个名称为 `<shape name>` 的 shape，定义后，`<shape name>` 可以用于 `\pgfnode` 中。

`<shape specification>` 是一些 T<sub>E</sub>X 代码，其中包含一些特殊的命令来定义 `<shape name>`。

下面是形状 `coordinate` 的定义：

```
\pgfdeclareshape{coordinate}
{
 \savedanchor\centerpoint{%
 \pgf@x=.5\wd\pgfnodeparttextbox%
 \pgf@y=.5\ht\pgfnodeparttextbox%
 \advance\pgf@y by -.5\dp\pgfnodeparttextbox%
 }
 \anchor{center}{\centerpoint}
 \anchorborder{\centerpoint}
}
```

下面介绍用在 `<shape specification>` 中的命令。

`\nodeparts{<list of node parts>}`

本命令规定 `<shape name>` 的 node parts 的个数和名称, `<list of node parts>` 是各个 node parts 的名称列表, 有几个名称就有几个 node part. 在默认下, 一个 shape 只有一个 node part, 其名称为 `text`. 有的形状, 如 `circle split` 有两个 node parts, 其名称分别是 `text` 和 `lower`, 因此在 `circle split` 的定义中应该有:

```
\nodeparts{text,lower}
```

当向各个 node parts 中添加文字时要用到 node part 的名称, 应当按照 `<list of node parts>` 中列出的名称次序, 依次向各个 node parts 中添加文字. 每个 node part 中都有一个盒子来盛放文字. 你还需要在各个部分中规定一个锚位置来放置文字盒子. 例如某个 node part 的名称是 `XYZ`, 这个部分中盛放文字的盒子就被规定为 `\pgfnodepartXYZbox`, 而放置这个文字盒子的锚位置名称也会被规定为 `XYZ`. 盒子 `\pgfnodepartXYZbox` 的左下角会被放在锚位置 `XYZ` 上.

### `\savedanchor{<command>}{<code>}`

这个命令定义一个 saved anchor, 其中 `<command>` 是个 T<sub>E</sub>X 宏, 例如 `\centerpoint`.

`<code>` 是对 `<command>` 的定义. 每当用命令 `\pgfnode` 或 `\pgfmultipartnode` 调用形状 `<shape name>` 时, 定义 saved anchor 的 `<code>` 都会被执行, 从而计算并保存 `<command>`, 并且, node parts 中的文字盒子也会包含文字. 当然, 文字盒子的内容可能是空的. 例如, 当执行 `<code>` 时, 如果只有一个 node part, 名称为 `text`, 那么程序会创建文字盒子 `\pgfnodeparttextbox` 并把文字放入盒子中.

在 `<code>` 中需要为 T<sub>E</sub>X 尺寸 `\pgf@x` 和 `\pgf@y` 赋值, 而 `<command>` 会自动被定义为点

```
\pgfpoint{\pgf@x}{\pgf@y}
```

当在 `<code>` 中引入 shape 坐标系中的点时, 这个点会自动引入 `\pgf@x` 和 `\pgf@y`, 因为这两个值构成坐标点的两个坐标分量.

你也可以在 `<code>` 中直接为 `\pgf@x` 和 `\pgf@y` 赋值, 赋值表达式中可以使用文字盒子的宽度、高度、深度来计算 saved anchor (即 `<command>`), 也可以使用 `\pgfshapeminwidth` 或 `\pgfshapeinnerxsep` 等宏, 也可以包含由命令 `\pgfnode` 引入的关于形状的其他变量. 这样就可以把文字盒子的尺寸与形状的锚位置 `<command>` 联系起来, 使得 `<command>` 的位置具有能适应文字盒子尺寸的弹性.

例如:

```
\savedanchor{\upperrightcorner}{
 \pgf@y=.5\ht\pgfnodeparttextbox % 文字盒子的高度
 \pgf@x=.5\wd\pgfnodeparttextbox % 文字盒子的宽度
}
```

如果要在 `<code>` 中使用宏 `\pgfshapeminwidth` 或者 `\pgfshapeinnerxsep`, 需要注意这两个宏是“纯文本宏”, 而不是尺寸宏. 表面上宏 `\pgfshapeminwidth` 的值可能是“2cm”, 但是这个“2cm”是纯文本, 不是尺寸, 所以“`0.5\pgfshapeminwidth`”是“0.52cm”, 而不是“1cm”, 所以需要使用以下句式:

```
\savedanchor{\upperrightcorner}{
 \pgf@y=.5\ht\pgfnodeparttextbox % 文字盒子的高度
 \pgf@x=.5\wd\pgfnodeparttextbox % 文字盒子的宽度
 \setlength{\pgf@xa}{\pgfshapeminwidth}
```



```

\ifdim\pgf@x<.5\pgf@xa
 \pgf@x=.5\pgf@xa
\fi
}

```

在 `<code>` 中计算出来的 `\pgf@x` 和 `\pgf@y` 的值会被命令 `\pgfnode` 保存起来。注意 `<command>` 是局部定义、局部计算、局部使用地，因此即使名称 `<command>` 由很简短的字符构成（如 `\center` 或 `\a`），也不太可能引起名称冲突。

### `\saveddimen{<command>}{<code>}`

本命令类似 `\savedanchor`。 `<command>` 是 T<sub>E</sub>X 宏，在 `<code>` 中你需要为 `\pgf@x` 赋值，这个值会被赋予 `<command>`。

```

\saveddimen{\depth}{
 \pgf@x=\dp\pgfnodeparttextbox
}

```

### `\savedmacro{<command>}{<code>}`

`<command>` 是 T<sub>E</sub>X 宏，在 `<code>` 中你需要直接定义 `<command>`，`<command>` 的值通常是数值或文字。

### `\anchor{<name>}{<code>}`

这个命令定义一个名称为 `<name>` 的 normal anchor（名称不以反斜线开头），`<code>` 是对这个锚位置的规定，在 `<code>` 中可以使用已定义的 saved anchor 来规定 `<name>` 的位置。与 saved anchor 不同，只有在用到 `<name>` 时才会执行 `<code>`。由于名称 `<name>` 不会被传递到系统层（system level），所以其构成可以比较随意，例如可以包含字母，数字，冒号。

`<code>` 应当能够设置 `\pgf@x` 和 `\pgf@y` 的值，`<name>` 所对应的位置会被自动设定为点

```

\pgfpoint{\pgf@x}{\pgf@y}

```

当在 `<code>` 中引入 shape 坐标系中的点时，这个点会自动引入 `\pgf@x` 和 `\pgf@y`，因为这两个值构成坐标点的两个坐标分量。

当在 `<code>` 中使用已定义的 saved anchor 时，会自动将 saved anchor 对应的 `\pgf@x` 和 `\pgf@y` 引入到 `<code>` 中，因为 saved anchor 就是用这两个值定义的。

在绘图环境中，绘图命令中使用的是 normal anchor，而不是 saved anchor。一个 saved anchor 不能自动转换为一个 normal anchor，如果要转换，可以使用本命令。如

```

\anchor{north east}{\upperrightcorner} % 这里 \upperrightcorner 是个 saved anchor

```

锚位置 north west 可以如下定义：

```

\anchor{north west}{
 \upperrightcorner % 这里 \upperrightcorner 是个 saved anchor
 \pgf@x=-\pgf@x % 变成原值的相反数
}

```

在 `<code>` 中可以使用各种坐标点命令来规定 `<name>` 对应的位置，例如

```
\anchor{center}{\pgfpointorigin} % shape 坐标系的原点
```

在默认下，一个 shape 只有一个 node part，这个 node part 的名称默认为 `text`，其中放置文字盒子的锚位置的名称也默认为 `text`，文字盒子的左下角会处于锚位置 `text` 上。锚位置 `text` 的默认位置是 shape 坐标系的原点，你也可以自己指定一个位置，例如：

```
\savedanchor{\upperrightcorner}{
 \pgf@y=.5\ht\pgfnodeparttextbox
 \pgf@x=.5\wd\pgfnodeparttextbox
}
\anchor{text}{
 \upperrightcorner
 \pgf@x=-\pgf@x
 \pgf@y=-\pgf@y
}
```

按上面代码规定锚位置 `text` 后，文字盒子 `\pgfnodeparttextbox` 的中心就位于 shape 坐标系的原点了。注意下面的代码：

```
\anchor{text}{\pgfpoint{-.5\wd\pgfnodeparttextbox}{-.5\ht\pgfnodeparttextbox}}
```

乍看之下与前面的代码是一样的，但是却可能导致意外的结果，这是因为这行代码中没有引入 `\pgf@x` 和 `\pgf@y`。这行代码规定的是名称为 `text` 的 node part 的锚位置，当执行这行代码时，`\pgfnodeparttextbox` 很可能还是之前的 shape 中文字盒子，或者是前一个 node part 的文字盒子。

当一个 shape 有多个 node parts 时，你需要用

```
\anchor{<node part name>}{<code>}
```

为每个 node part 定义放置文字盒子的锚位置，当然这个锚位置的名称与所在的 node part 的名称是一样的。

```
\deferredanchor{<name>}{<code>}
```

这个命令类似 `\anchor`。<name> 是个锚位置名称，<code> 是对这个锚位置的规定。

以下两个命令

```
\anchor{<name>}{<code>}
\deferredanchor{<name>}{<code>}
```

中的 <name> 都是（不以反斜线开头的）字符串（其中可以有空格）。如果某个宏的展开值是字符串（不含特殊字符），那么这个宏就可以用于 <name> 中。

当程序读取命令 `\pgfdeclareshape` 的内容时，如果 `\anchor{<name>}{<code>}` 的 <name> 中含有宏（展开值是字符串），那么这个宏会被立即展开；如果 `\deferredanchor{<name>}{<code>}` 的 <name> 中含有宏（展开值是字符串），那么这个宏不会被立即展开，而是等到在绘图命令中使用所定义的 shape 以及锚位置 <name> 时，才会把 <name> 中的宏展开。

观察下面的例子：

•  
anchor bar anchor

•  
anchor foo anchor

```

\makeatletter
\def\foo{foo} % 宏 \foo 的值是字母串 foo
\pgfdeclareshape{simple shape}{%
 \savedanchor{\center}{\pgfpointorigin}
 \anchor{center}{\center}
 \savedanchor{\anchorfoo}{%
\pgf@x=1cm
\pgf@y=0cm}
\deferredanchor{anchor \foo}{\anchorfoo}} % 用宏 \foo 构成锚位置名称
\makeatother

\begin{tikzpicture}
 \node[simple shape] (Test1) at (0,0) {};
 \fill (Test1.anchor foo) circle (2pt) node[below] {anchor foo anchor}; % 引用锚位置

 \def\foo{bar} % 重定义宏 \foo 的值是字母串 bar
 \node[simple shape] (Test2) at (2,1) {};
 \fill (Test2.anchor bar) circle (2pt) node[below] {anchor bar anchor}; % 引用锚位置
\end{tikzpicture}

```

### `\anchorborder{<code>}`

这个命令确定 shape 的边界路径上的一个点。当你使用命令 `\pgfpointshapeborder` 时，本命令的 `<code>` 会被执行。`<code>` 的执行结果是 shape 的边界路径上的一个点，这个点对应命令 `\pgfpointshapeborder` 确定的点。

命令 `\pgfpointshapeborder` 的句法是

```
\pgfpointshapeborder{<node name>}{<point>}
```

注意这里的 `<point>` 是绘图环境的坐标系中的点，假设将 `<point>` 变换到 shape 坐标系中得到的是点  $p$ ，另外我们假设在 shape 坐标系中 shape 的中心锚位置是点  $c$ 。

当执行 `<code>` 时，尺寸宏 `\pgf@x` 和 `\pgf@y` 的值会被设定到点  $p$  处（即成为该点的坐标分量，在 shape 坐标系下）。通过执行 `<code>` 中的代码，变化 `\pgf@x` 和 `\pgf@y` 的值，使之能够作为 shape 的边界路径上某个点的坐标分量，这个点应当是射线  $cp$  与 shape 的边界路径的交点。再将这个交点变换到绘图环境的坐标系中，就得到命令 `\pgfpointshapeborder` 确定的点。

在 `<code>` 中你可以通过算式直接为 `\pgf@x` 和 `\pgf@y` 赋值，也可以使用某些命令确定一个点，程序会自动把 `\pgf@x` 和 `\pgf@y` 作成这个点的坐标分量。当然，在编写 `<code>` 时，你需要假设前面提到的点  $p$  是已知的，并且  $p$  的坐标分量就是 `\pgf@x` 和 `\pgf@y` 的值（二者的初始值）。

当 shape 的边界路径形状比较复杂时，<code> 中的代码可能会比较复杂，需要大量计算。如果你不太相信 T<sub>E</sub>X 的计算能力，你可以用某个简单的路径来计算“交点”，当然这个交点很可能不在原本复杂的边界路径上。

举个简单的例子。假设我们定义一个简单的矩形 shape，它的中心在 shape 坐标系的原点，它的右上角是 saved anchor 锚位置 `\upperrightcorner`，可以如下定义：

```
\anchorborder{%
 \@tempdima=\pgf@x % 转存 \pgf@x 的值
 \@tempdimb=\pgf@y
% 下面调用命令 \pgfpointborderrectangle 做计算
 \pgfpointborderrectangle{\pgfpoint{\@tempdima}{\@tempdimb}}{\upperrightcorner}
}
```

### `\backgroundpath{<code>}`

这个命令定义 shape 的 background path，注意在 <code> 中定义的这个路径是没有被“使用”的，“使用”的意思是，如，用某个线宽的线条画出路径，填充颜色，渐变颜色等等。当然你在调用 shape 时可以“使用”它的 background path，或者直接弃之不用。

当 <code> 被执行时，所有的 saved anchor 都是有效的可用的，<code> 中应当包含创建路径的命令。例如

```
\backgroundpath{
 \pgfpathrectanglecorners
 {\upperrightcorner}
 {\pgfpointscale{-1}{\upperrightcorner}}
}
```

background path 所在的“层”处于文字层之下，即先画出 background path，然后添加文字。

<code> 中可以直接使用 tikz 的绘图命令（不带环境，但命令结束处要加分号）。

### `\foregroundpath{<code>}`

这个命令定义 shape 的 foreground path，在添加文字后再画出 foreground path，例如，§67.4，程序库 shapes.symbols 中定义的形状 `correct forbidden sign`：



```
\begin{tikzpicture}
 \node [correct forbidden sign,line width=1ex,draw=red,fill=white]
 {Smoking};
\end{tikzpicture}
```

文字被 foreground path 遮挡。

<code> 中可以直接使用 tikz 的绘图命令（不带环境，但命令结束处要加分号）。

### `\behindbackgroundpath{<code>}`

这个命令定义 shape 的 behind background path, 这里 `<code>` 中不仅创建路径, 而且可以“使用”路径。Behind background path 会在 background path 之前画出。

`<code>` 中可以直接使用 tikz 的绘图命令 (不带环境, 但命令结束处要加分号)。

`\beforebackgroundpath{<code>}`

这个命令定义 shape 的 before background path, 这个路径在 background path 画出之后, 添加文字之前画出。

`\behindforegroundpath{<code>}`

这个命令定义 shape 的 behind foreground path, 这个路径在添加文字之后, 画出 foreground path 之前画出。

`\beforeforegroundpath{<code>}`

这个命令定义 shape 的 before foreground path, 这个路径在画出 foreground path 之后画出。

`\inheritsavedanchors[from={<another shape name>}]`

这里 `<another shape name>` 是某个已定义的 shape 的名称。在 `<another shape name>` 的定义中, 有关于它的所有 saved anchors 或 saved dimensions 的定义代码, 本命令直接将这些代码全部拿过来, 用做新定义的 shape 的 saved anchors 或 saved dimensions 的定义代码。

显然, 当新定义的 shape 与 `<another shape name>` 很相像时, 本命令是个快捷的办法。

你可以多次使用这个命令, 将多个已定义 shape 的全部 saved anchors 或 saved dimensions 的定义代码搬过来。

`\inheritbehindbackgroundpath[from={<another shape name>}]`

这里 `<another shape name>` 是某个已定义的 shape 的名称。本命令将 `<another shape name>` 的定义中, 定义 behind background path 的代码全部搬过来。

`\inheritbackgroundpath[from={<another shape name>}]`

这里 `<another shape name>` 是某个已定义的 shape 的名称。本命令将 `<another shape name>` 的定义中, 定义 background path 的代码全部搬过来。

`\inheritbeforebackgroundpath[from={<another shape name>}]`

本命令将 `<another shape name>` 的定义中, 定义 before background path 的代码全部搬过来。

`\inheritbehindforegroundpath[from={<another shape name>}]`

本命令将 `<another shape name>` 的定义中, 定义 behind foreground path 的代码全部搬过来。

`\inheritforegroundpath[from={<another shape name>}]`

本命令将 `<another shape name>` 的定义中, 定义 foreground path 的代码全部搬过来。

`\inheritbeforeforegroundpath[from={<another shape name>}]`

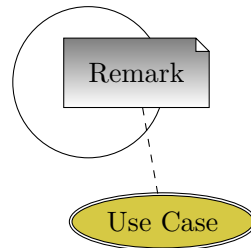
本命令将 `<another shape name>` 的定义中, 定义 before foreground path 的代码全部搬过来。

`\inheritanchor[from={<another shape name>}]{<name>}`

这里 `<another shape name>` 是某个已定义的 shape 的名称, `<name>` 是 `<another shape name>` 的某个 normal anchor 的名称 (一串字符)。本命令将 `<another shape name>` 的定义中, 定义 `<name>` 的代码全部搬过来。

`\inheritanchorborder[from={<another shape name>}]`

本命令将 `<another shape name>` 的定义中, 所有的 normal anchors 的定义代码都搬过来。



```

\makeatletter
\pgfdeclareshape{document}{
 \inheritsavedanchors[from=rectangle] % 使用 rectangle 的 saved anchors
 \inheritanchorborder[from=rectangle]
 \inheritanchor[from=rectangle]{center}
 \inheritanchor[from=rectangle]{north}
 \inheritanchor[from=rectangle]{south}
 \inheritanchor[from=rectangle]{west}
 \inheritanchor[from=rectangle]{east}
 \behindbackgroundpath{\draw (0,0)circle(1cm);}
 \backgroundpath{
 \southwest % 引入锚位置 \southwest 的两个坐标分量 \pgf@x 和 \pgf@y
 \pgf@xa=\pgf@x \pgf@ya=\pgf@y % 为 \pgf@xa 和 \pgf@ya 赋值, 它们是左下角点的坐标分量
 \northeast % 引入锚位置 \northeast 的两个坐标分量 \pgf@x 和 \pgf@y
 \pgf@xb=\pgf@x \pgf@yb=\pgf@y % 为 \pgf@xb 和 \pgf@yb 赋值, 它们是右上角点的坐标分量
 \pgf@xc=\pgf@xb \advance\pgf@xc by-5pt % 为 \pgf@xc 赋值, 即从 \pgf@xb 处左移 5pt
 \pgf@yc=\pgf@yb \advance\pgf@yc by-5pt % 为 \pgf@yc 赋值, 即从 \pgf@yb 处下移 5pt
 \pgfpathmoveto{\pgfpoint{\pgf@xa}{\pgf@ya}} % 以左下角点为始点构建路径
 \pgfpathlineto{\pgfpoint{\pgf@xa}{\pgf@yb}} % 连线到左上角点
 \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yb}} % 连线到右上角点左侧 5pt 点处
 \pgfpathlineto{\pgfpoint{\pgf@xb}{\pgf@yc}} % 连线到右上角点下部 5pt 点处
 \pgfpathlineto{\pgfpoint{\pgf@xb}{\pgf@ya}} % 连线到右下角点
 \pgfpathclose
 \pgfpathmoveto{\pgfpoint{\pgf@xc}{\pgf@yb}} % 以右上角点左侧 5pt 点为始点构建路径
 \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}} % 从前一个点向下连线
 \pgfpathlineto{\pgfpoint{\pgf@xb}{\pgf@yc}} % 连线到右上角点下部 5pt 点处
 }
}

```

```

}
\makeatother

\begin{tikzpicture}
 \node[shade,draw,shape=document,inner sep=2ex] (x) {Remark};
 \node[fill=yellow!80!black,draw,ellipse,double] at ([shift=(-80:2cm)]x) (y) {Use Case};
 \draw[dashed] (x) -- (y);
\end{tikzpicture}

```

## 102 矩阵

首先调用模块 `matrix`.

```
\usepgfmodule{matrix} % LaTeX and plain TeX and pure pgf
```

### 102.1 Overview

参考 §20.

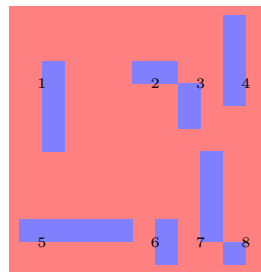
命令 `\pgfmatrix` 创建矩阵。矩阵被创建为一个 `node`，矩阵的元素称为“cell”，矩阵的元素或者留空，或者是个“图形”（cell pictures）。

### 102.2 矩阵元素的对齐方式

参考 §20.3.1.

在一行之内，相邻两个元素之间用命令 `\pgfmatrixnextcell` 隔开。在每一行的末尾（包括最后一行）使用命令 `\pgfmatrixendrow` 结束，也可以使用两个反斜线符号“`\\`”结束一行。

`\pgfmatrixnextcell`，`\pgfmatrixendrow` 以及 `\\` 都可以带有选项。



```

\begin{tikzpicture}[x=3mm,y=3mm,fill=blue!50] % 选项 fill=blue!50 对元素图形的 \fill 命令
 无效,
 \def\atorig#1{\node[black] at (0,0) {\tiny #1};}
 \def\pgfmatrixbegincode{\pgfsetfillcolor{blue!50}} % 这个设置对元素图形的 \fill 命令有效
 \pgfmatrix{rectangle}{center}{mymatrix}{\pgfsetfillcolor{red!50}\pgfusepath{fill}}
 {\pgfpointorigin}{}

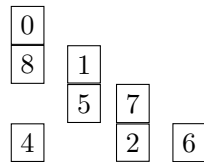
```

```

{
 \fill (0,-3) rectangle (1,1); \atorig1 \pgfmatrixnextcell
 \fill (-1,0) rectangle (1,1); \atorig2 \pgfmatrixnextcell
 \fill (-1,-2) rectangle (0,0); \atorig3 \pgfmatrixnextcell
 \fill (-1,-1) rectangle (0,3); \atorig4 \\
 \fill (-1,0) rectangle (4,1); \atorig5 \pgfmatrixnextcell
 \fill (0,-1) rectangle (1,1); \atorig6 \pgfmatrixnextcell
 \fill (0,0) rectangle (1,4); \atorig7 \pgfmatrixnextcell
 \fill (-1,-1) rectangle (0,0); \atorig8 \\
}
\end{tikzpicture}

```

矩阵元素的行列排布类似表格，但是要比表格的要求宽松一些，矩阵某两行的元素个数可以不相等，某两列的元素个数也可以不相等，观察下面的例子：



```

\begin{tikzpicture}[every node/.style=draw]
 \pgfsetmatrixcolumnsep{1mm}
 \pgfmatrix{rectangle}{center}{mymatrix}{\pgfusepath{}}{\pgfpointorigin}{\let\&=\pgfmatrixnextcell}
 {
 \node {0}; \\
 \node {8}; \&[2mm] \node {1}; \\
 \& \node {5}; \&[1mm] \node {7}; \\
 \node {4}; \& \node {2}; \&[2mm] \node {6}; \\
 }
\end{tikzpicture}

```

### 102.3 矩阵命令

`\pgfmatrix{<shape>}{<anchor>}{<name>}{<usage>}{<shift>}{<pre-code>}{<matrix cells>}`

这个命令创建一个矩阵，矩阵实为一个 node，名称为 <name>，形状为 <shape>。在默认下，矩阵的锚位置 <anchor> 处于（绘图环境坐标系的）原点上。向量 <shift> 会使得矩阵被平移，不过平移向量是 <shift> 的负向量。可以这样理解：先平移矩阵使得锚位置 <anchor> 处于原点上，再按 <shift> 的负向量平移矩阵，确定矩阵的位置。

<matrix cells> 规定矩阵的各个元素图形。元素图形的代码是直接的绘图命令，不必放入绘图环境，程序会跟踪元素图形的边界盒子，并将元素按规则对齐。



`<usage>` 是关于使用路径的命令，使用的是矩阵的路径（background path），不是元素图形中的路径。

**分隔元素的命令与换行符号** 前文已经提到 `\pgfmatrixnextcell`，`\pgfmatrixendrow` 以及 `\\`。

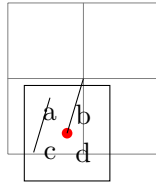
注意，在这里不能使用 `&` 作为分隔左右两个元素的符号，除非将 `&` 定义为 `\pgfmatrixnextcell`。

因为 `{minipage}` 环境会重定义 `\\`，所以在 `{minipage}` 环境中使用矩阵时，最好不用 `\\` 换行。

**矩阵的锚位置与指向点** 矩阵是只有一个 node part 的 node（参考 §101），这个 node part 的名称默认为 `text`，其中盛放文字的盒子用来放置矩阵的元素。盒子的左下角位于矩阵的锚位置 `text` 上。

我们假设矩阵的元素都是 node，每个元素 node 都有自己的各种锚位置。假设有个元素 node 的名称是 inner node，有锚位置 `inner node.north`。前面提到，向量 `<shift>` 会使得矩阵按 `<shift>` 的负向量平移。当读取 `<shift>` 时，各个元素 node 的各种锚位置都是可以引用的。如果 `<shift>` 是 `\pgfpointanchor{inner node}{north}`，矩阵又会怎样平移呢？

先看一下 `\pgfpointanchor{inner node}{north}` 是怎样确定的。以矩阵的锚位置 `text` 为原点建立一个坐标系，从这个原点到点 `inner node.north` 的向量就是 `\pgfpointanchor{inner node}{north}`。因此，如果在矩阵命令中，令 `<anchor>` 是 `text`，`<shift>` 是 `\pgfpointanchor{inner node}{north}`，那么点 `inner node.north` 就会处于图形的原点——绘图环境坐标系的原点。



```
\begin{tikzpicture}
\draw [help lines] (-1,-1) grid (1,1);
\pgfmatrix{rectangle}{center}{mymatrix}{\pgfusepath{draw}}{\pgfpointanchor{a}{north}}{
{
\node (a){a}; \pgfmatrixnextcell \node {b}; \pgfmatrixendrow
\node {c}; \pgfmatrixnextcell \node {d}; \pgfmatrixendrow
}
\fill [red] (mymatrix.center) circle(2pt);
\draw (mymatrix.center)--(0,0);
\draw (mymatrix.text)--(a.north); % 两条线段平行且长度相等
\end{tikzpicture}
```

**旋转与放缩** 如果矩阵命令之前有旋转、放缩、平移命令，那么这些命令对矩阵无效，将来不打算改变这一点。如果要对矩阵做变换，你只能自己规定画布变换。

在矩阵元素图形的命令中使用变换选项，可以被变换元素图形。

**调用命令** 宏 `\pgfmatrixbegincode`，`\pgfmatrixendcode`，`\pgfmatrixemptycode` 分别保存一组代码。见 §102.5。

当执行任何一个元素图形代码时，会先执行宏 `\pgfmatrixbegincode` 保存的代码。

在执行任何一个元素图形代码之后，会执行 `\pgfmatrixendcode` 保存的代码。

宏 `\pgfmatrixemptycode` 保存的代码是针对空元素的。

**pre-code** 矩阵的代码会被放入一个  $\text{T}_{\text{E}}\text{X}$  分组内执行，矩阵命令中的 `<pre-code>`（一组代码）也会被放入这个  $\text{T}_{\text{E}}\text{X}$  分组内，但在矩阵代码之前被执行，并且 `<pre-code>` 不属于矩阵代码。

你可以在 `<pre-code>` 中定义

```
\let\&=\pgfmatrixnextcell
```

也可以在 `<pre-code>` 中使用命令 `\aftergroup`，待  $\text{T}_{\text{E}}\text{X}$  分组结束后执行某些操作。

**元素对齐过程中的宏展开** 矩阵元素的对齐过程（行方向与列方向）实际是个构造列表的过程，即把元素图形排布成一个矩形列表，其中使用了命令 `\halign`，在 §17.4.3 中，选项 `node halign header` 也利用了这个命令。这个命令可能会做一些奇怪的事情，例如，它会把元素图形代码的第一个宏展开为通常的形式。因此你需要注意：

- 如果某个元素的代码中有一个宏，它展开后包含 `\pgfmatrixnextcell` 或 `\pgfmatrixendrow`，你需要注意不要再重复使用 `\pgfmatrixnextcell` 或 `\pgfmatrixendrow`。
- 你可以定义一个宏，它展开后包含 `\pgfmatrixnextcell` 或 `\pgfmatrixendrow`，这样就可添加矩阵的列或行。

## 102.4 行间距与列间距

`\pgfsetmatrixcolumnsep{<sep list>}`

这个宏设置默认的列间距为尺寸 `<sep list>`，这个间距会用在任意相邻两列之间。

`<sep list>` 可以是一个尺寸，也可以是用逗号分隔的数个尺寸，例如

```
\pgfsetmatrixcolumnsep{1mm,2mm,3mm}
```

此时本命令指定的列间距就是  $1\text{mm}+2\text{mm}+3\text{mm}=6\text{mm}$ 。

`<sep list>` 中可以使用选项 `between borders` 或者 `between origins`（见 §20.3.2），默认使用 `between borders`，这里的此种选项设置是针对任意相邻两列的。

`\pgfmatrixnextcell[<additional sep list>]`

这个命令有两个作用。一是分隔左右两个元素，二是使用选项 `<additional sep list>` 调整列间距，这里 `<additional sep list>` 是个尺寸。这个尺寸选项是可选的，如果不给出这个可选尺寸，那么列间距就是由宏 `\pgfsetmatrixcolumnsep{<sep list>}` 指定的默认尺寸 `<sep list>`；如果给出这个可选尺寸，那么列间距就是默认尺寸加上这个可选尺寸。也就是说，本命令指定的是个“附加间距”。

当这个命令带有尺寸选项时，一般只能用在第一行中。假设第一行有  $n$  个元素，而之后的某一行有  $n+1$  个元素，那么在该行的第  $n$  个元素与第  $n+1$  个元素之间可以使用这个命令并带上尺寸选项。在前面的例子中已经显示了这一点。

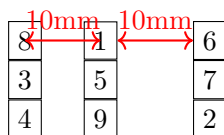
`<additional sep list>` 可以是一个尺寸，也可以是用逗号分隔的数个尺寸，例如

```
\pgfmatrixnextcell[1mm,2mm,3mm]
```

此时本命令指定的附加列间距就是  $1\text{mm}+2\text{mm}+3\text{mm}=6\text{mm}$ .

`<additional sep list>` 中可以使用选项 `between borders` 或者 `between origins` (见 §20.3.2), 默认使用 `between borders`, 这里的此种选项设置只是针对当前的相邻两列, 其优先性高于宏 `\pgfsetmatrixcolumnsep` 中的此种选项设置。

注意, 如果 `<additional sep list>` 中使用选项 `between borders` 或者 `between origins`, 那么本命令只能用在第一行中。



```
\begin{tikzpicture}[every node/.style=draw]
 \pgfsetmatrixcolumnsep{1cm,between origins}
 \pgfmatrix{rectangle}{center}{mymatrix}{\pgfusepath{}}{\pgfpointorigin}
 {\let\&=\pgfmatrixnextcell}
 {
 \node (a) {8}; \& \node (b) {1}; \&[between borders] \node (c) {6}; \\
 \node {3}; \& \node {5}; \& \node {7}; \\
 \node {4}; \& \node {9}; \& \node {2}; \\
 }
 \begin{scope}[every node/.style=]
 \draw [<->,red,thick] (a.center) -- (b.center) node [above,midway] {10mm};
 \draw [<->,red,thick] (b.east) -- (c.west) node [above,midway]{10mm};
 \end{scope}
\end{tikzpicture}
```

`\pgfsetmatrixrowsep{<sep list>}`

类似宏 `\pgfsetmatrixcolumnsep`, 只是针对行间距。

`\pgfmatrixendrow[<additional sep list>]`

类似命令 `\pgfmatrixnextcell`, 只是针对行间距。

`<additional sep list>` 中可以使用选项 `between borders` 或者 `between origins`.

注意符号 `\&` 与本命令的作用是一样的。

在最后一行的末尾也要使用本命令, 但它的选项无效。

## 102.5 调用命令

`\pgfmatrixemptycode`

你需要用定义命令, 如 `\def` 为这个宏赋值

```
\def\pgfmatrixemptycode{<code>}
```

这个定义中的 `<code>` 是针对空元素的。如果在命令 `\pgfmatrixnextcell`（或其等效形式）之前或之后的元素是“空的”，那么这个元素就是“空元素”。程序会在空元素的位置上执行 `<code>`。

```
a empty b
empty c d empty
```

```
\begin{tikzpicture}
 \def\pgfmatrixemptycode{\node{empty};}
 \pgfmatrix{rectangle}{center}{mymatrix}{\pgfusepath{}}{\pgfpointorigin}
 {\let\&=\pgfmatrixnextcell}
 {
 \node {a}; \& \& \node {b}; \& \&
 \& \node{c}; \& \node {d}; \& \&
 }
\end{tikzpicture}
```

### `\pgfmatrixbegincode`

你需要用定义命令，如 `\def` 为这个宏赋值

```
\def\pgfmatrixbegincode{<code>}
```

这个定义中的 `<code>` 会加在所有“非空元素”的开头，即执行 `<code>` 后再执行非空元素的绘图命令——针对任何非空元素。

### `\pgfmatrixendcode`

你需要用定义命令，如 `\def` 为这个宏赋值

```
\def\pgfmatrixendcode{<code>}
```

这个定义中的 `<code>` 会加在所有“非空元素”的结尾，即执行非空元素的绘图命令后再执行 `<code>`——针对任何非空元素。

在宏 `\pgfmatrixbegincode` 与 `\pgfmatrixendcode` 之间，并非只有非空元素的绘图命令，其间 PGF 还会插入某些不可见的命令，包括 `\let` 或者 `\gdef`。如果定义 `\pgfmatrixbegincode` 的代码以 `\csname` 结束，并且定义 `\pgfmatrixendcode` 的代码以 `\endcsname` 结束，那么可能会导致错误——这不是个好主意。

下面例子中，所列出的矩阵元素都是字母，不是绘图命令，但是通过对宏 `\pgfmatrixbegincode` 和 `\pgfmatrixendcode` 的定义，将 `\node [draw] \bgroup` 放在字母开头，将 `\egroup;` 放在字母结尾，从而做成一个 `\node` 命令。

|   |   |   |
|---|---|---|
| a | b | c |
| d |   | e |

```
\begin{tikzpicture}
 \def\pgfmatrixbegincode{\node[draw]\bgroup}
 \def\pgfmatrixendcode{\egroup;} % 会把元素转成 node
```

```

\pgfmatrix{rectangle}{center}{mymatrix}{\pgfusepath{}}{\pgfpointorigin}
{\let\&=\pgfmatrixnextcell}
{
 a \& b \& c \\
 d \& \& e \\
}
\end{tikzpicture}

```

`\pgfmatrixcurrentrow`

这个宏是个计数器, 它的值是当前行的行号, 不要随意改动它的值。

`\pgfmatrixcurrentcolumn`

这个宏是个计数器, 它的值是当前列的列号, 不要随意改动它的值。

## 103 坐标变换, 画布变换, 非线性变换

### 103.1 Overview

坐标变换只针对坐标点, 不针对线宽、文字。

非线性变换处理起来比较慢, 不太好用, 所以专门有一个模块 `nonlineartransformations` 来处理非线性变换。默认下是不会自动载入这个模块的, 如果要用它, 你需要手工载入这个模块。

### 103.2 坐标变换

#### 103.2.1 坐标变换矩阵

一般来说, 变换矩阵对坐标点  $\begin{pmatrix} x \\ y \end{pmatrix}$  的变换如下:

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} a & b & s \\ c & d & t \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} ax + by + s \\ cx + dy + t \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} ax + by + s \\ cx + dy + t \end{pmatrix},$$

注意, 上面的式子中将坐标点  $\begin{pmatrix} x \\ y \end{pmatrix}$  扩展为齐次坐标做计算。

当使用多个坐标变换矩阵时, 相应的变换矩阵会被依次相乘。

坐标变换矩阵对其后的路径有作用, 其作用范围受到  $\text{T}_\text{E}_\text{X}$  分组的限制, 而画布变换受到 `{pgfscope}` 环境的限制。

#### 103.2.2 坐标变换命令

`\pgftransformshift{<point>}`

按向量 `<point>` 做平移。



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (2,1) -- (1,0);
\pgftransformshift{\pgfpoint{1cm}{1cm}}
\draw[red] (0,0) -- (2,1) -- (1,0);
\end{tikzpicture}
```

### `\pgftransformxshift{<dimensions>}`

在 x 轴方向做平移, 只需一个尺寸 <dimensions>, 这个尺寸可正可负。



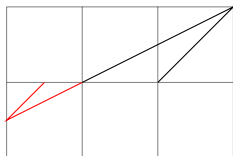
```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (2,1) -- (1,0);
\pgftransformxshift{.5cm}
\draw[red] (0,0) -- (2,1) -- (1,0);
\end{tikzpicture}
```

### `\pgftransformyshift{<dimensions>}`

在 y 轴方向做平移, 只需一个尺寸 <dimensions>, 这个尺寸可正可负。

### `\pgftransformscale{<factor>}`

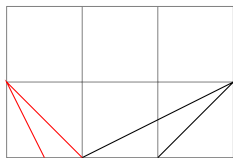
以原点为中心做位似变换, <factor> 是放缩比例。如果 <factor> 是负值, 则先以原点为中心做中心对称, 再做位似变换。也就是将 <factor> 与点向量相乘。



```
\begin{tikzpicture}
\draw[help lines] (-1,-1) grid (2,1);
\draw (0,0) -- (2,1) -- (1,0);
\pgftransformscale{-.5}
\draw[red] (0,0) -- (2,1) -- (1,0);
\end{tikzpicture}
```

### `\pgftransformxscale{<factor>}`

保持点的纵坐标不变, 将点的横坐标乘上 <factor>。



```
\begin{tikzpicture}
\draw[help lines] (-1,0) grid (2,2);
\draw (0,0) -- (2,1) -- (1,0);
\pgftransformxscale{-.5}
\draw[red] (0,0) -- (2,1) -- (1,0);
\end{tikzpicture}
```

**\pgftransformyscale{<factor>}**

保持点的横坐标不变, 将点的纵坐标乘上 <factor>.

**\pgftransformxslant{<factor>}**

这个变换是  $(x, y) \rightarrow (x + y \cdot \langle \text{factor} \rangle, y)$ .



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (2,1) -- (1,0);
\pgftransformxslant{.5}
\draw[red] (0,0) -- (2,1) -- (1,0);
\end{tikzpicture}
```

**\pgftransformyslant{<factor>}**

这个变换是  $(x, y) \rightarrow (x, y + x \cdot \langle \text{factor} \rangle)$ .

**\pgftransformrotate{<angles>}**

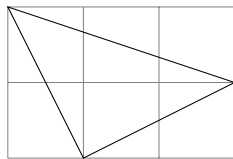
将点围绕原点旋转 <angles> 角度。



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (2,1) -- (1,0);
\pgftransformrotate{30}
\draw[red] (0,0) -- (2,1) -- (1,0);
\end{tikzpicture}
```

**\pgftransformtriangle{<a>}{<b>}{<c>}**

这个变换是针对坐标系的, 即它变换“标架”,  $(0,0) \rightarrow \langle a \rangle$ ,  $(1\text{pt}, 0\text{pt}) \rightarrow \langle b \rangle$ ,  $(0\text{pt}, 1\text{pt}) \rightarrow \langle c \rangle$ , 也就是说, 新的标架以  $\langle a \rangle$  为原点, 以  $\langle b \rangle$  为“横轴”单位向量, 以  $\langle c \rangle$  为“纵轴”单位向量。



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\pgftransformtriangle
{\pgfpoint{1cm}{0cm}}
{\pgfpoint{0cm}{2cm}}
{\pgfpoint{3cm}{1cm}}
\draw (0,0) -- (1pt,0pt) -- (0pt,1pt) -- cycle;
\end{tikzpicture}
```

**\pgftransformcm{<a>}{<b>}{<c>}{<d>}{<point>}**

等效于

```
/tikz/cm={<a>,,<c>,<d>,<coordinate>}
```

### `\pgfpointtransformed{<point>}`

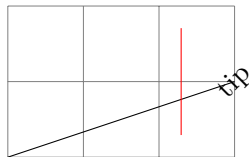
本命令将当前的线性变换矩阵用于点 `<point>`. 本命令经常被内部命令调用。

### `\pgftransformarrow{<start>}{<end>}`

点 `<start>` 到点 `<end>` 构成一个向量, 记这个向量的倾角是  $a$ , 本命令将标架原点平移到点 `<end>` 处, 并将标架旋转角度  $a$ , 得到一个新的标架, 之后的路径都在这个新标架内画出。

本命令对路径做平移、旋转。

注意预定义的路径 `rectangle`, `circle` 以其中心为起点。



```
\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,2);
 \draw (0,0) -- (3,1);
 \pgftransformarrow{\pgfpoint{1cm}{-1cm}}
 {\pgfpoint{3cm}{1cm}}
 \pgftext{tip}
 \draw [red](-1,0) -- (0,1);
\end{tikzpicture}
```

### `\pgftransformlineattime{<time>}{<start>}{<end>}`

点 `<start>` 到点 `<end>` 构成一个有向线段  $d$ , 本命令会调用命令 `\pgfpointlineattime` (参考 §96.5.2) 来确定一个点  $p$ , 将标架原点平移到点  $p$  处, 之后各路径都在新标架内画出。如果 `\ifpgfslopedattime` 的值是 `true`, 本命令还会将标架围绕点  $p$  旋转, 旋转角度是线段  $d$  的倾角。

本命令对路径做平移、旋转。



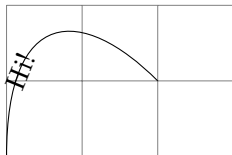
```
\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,2);
 \draw (0,0) -- (2,1);
 \pgfslopedattimetrue
 \pgftransformlineattime{.25}
 {\pgfpointorigin}{\pgfpoint{2cm}{1cm}}
 \pgftext{Hi!}
\end{tikzpicture}
```

### `\pgftransformcurveattime{<time>}{<start>}{<first support>}{<second support>}{<end>}`

点 `<start>` 作为始点, 点 `<first support>` 作为第一控制点, 点 `<second support>` 作为第二控制点, 点 `<end>` 作为终点, 确定一段控制曲线, 本命令调用命令 `\pgfpointcurveattime` (参考 §96.5.2) 来确定一个点  $p$ , 将标架原点平移到点  $p$  处, 之后各路径都在新标架内画出。如果 `\ifpgfslopedattime` 的值是 `true`, 本命令还会将标架围绕点  $p$  旋转, 旋转角度是控制曲线在点  $p$  处的切线的倾角。



本命令对路径做平移、旋转。

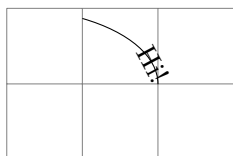


```
\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,2);
 \draw (0,0) .. controls (0,2) and (1,2) .. (2,1);
 \pgfslopedattimetrue
 \pgftransformcurveattime{.25}{\pgfpointorigin}
 {\pgfpoint{0cm}{2cm}}{\pgfpoint{1cm}{2cm}}
 {\pgfpoint{2cm}{1cm}}
 \pgftext{Hi!}
\end{tikzpicture}
```

`\pgftransformarccaxesattime{<time t>}{<center>}{<0-degree axis>}{<90-degree axis>}{<start angle>}{<end angle>}`

本命令调用命令 `\pgfpointarccaxesattime` 来确定一个点  $p$ , 将标架原点平移到点  $p$  处, 之后各路径都在新标架内画出。如果 `\ifpgfslopedattime` 的值是 `true`, 本命令还会将标架围绕点  $p$  旋转, 旋转角度是椭圆弧在点  $p$  处的切线的倾角。

本命令对路径做平移、旋转。



```
\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,2);
 \pgfpathmoveto{\pgfpoint{2cm}{1cm}}
 \pgfpatharccaxes{0}{60}{\pgfpoint{2cm}{0cm}}
 {\pgfpoint{0cm}{1cm}}
 \pgfusepath{stroke}
 \pgfslopedattimetrue
 \pgftransformarccaxesattime{.25}{\pgfpoint{0cm}{1cm}}
 {\pgfpoint{2cm}{0cm}}{\pgfpoint{0cm}{1cm}}{0}{60}
 \pgftext{Hi!}
\end{tikzpicture}
```

`\ifpgfslopedattime`

这个 T<sub>E</sub>X-if 针对前面提到的带有“attime”的变换命令, 其作用已经在前面解释过了。它的默认值是 `false`, 用命令 `\pgfslopedattimetrue` 将它的值设为 `true`, 用命令 `\pgfslopedattimefalse` 将它的值设为 `false`。

`\ifpgfallowupsidedowattime`

这个 T<sub>E</sub>X-if 针对前面提到的带有“attime”的变换命令。

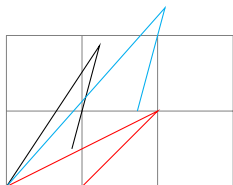
`\ifpgfresetnontranslationsattime`

这个 T<sub>E</sub>X-if 针对前面提到的带有“attime”的变换命令。

## 103.2.3 其它变换

`\pgftransformreset`

本命令将之前出现的坐标变换矩阵变成单位矩阵, 即取消变换, 其有效范围受到 `TEX` 分组的限制。



```
\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,2);
 \pgftransformrotate{30}
 \draw (0,0) -- (2,1) -- (1,0);
 {
 \pgftransformreset
 \draw[red] (0,0) -- (2,1) -- (1,0);
 }
 \draw[cyan] (0,0) -- (3,1) -- (2,0);
\end{tikzpicture}
```

`\pgftransformresetnontranslations`

本命令将之前出现的坐标变换矩阵变成平移矩阵  $\begin{pmatrix} 1 & 0 & s \\ 0 & 1 & t \\ 0 & 0 & 1 \end{pmatrix}$ , 也就是说, 将之前的变换中的旋转、反射、缩放成分去掉, 只保留平移作用。

本命令的有效范围受到 `TEX` 分组的限制。

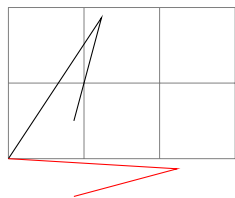


```
\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,2);
 \pgftransformscale{2}
 \pgftransformrotate{30}
 \pgftransformxshift{1cm}
 \pgftext{\color{red}rotated}
 \pgftransformresetnontranslations
 \pgftext{shifted only}
\end{tikzpicture}
```

`\pgftransforminvert`

本命令将当前的变换矩阵换成其逆矩阵, 作用于之后的路径。如果当前变换矩阵的条件数太小, 即当前变换矩阵接近奇异矩阵, 本命令的作用可能会有明显的误差。

本命令的有效范围受到 `TEX` 分组的限制。



```
\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,2);
 \pgftransformrotate{30}
 \draw (0,0) -- (2,1) -- (1,0);
 \pgftransforminvert
 \draw[red] (0,0) -- (2,1) -- (1,0);
\end{tikzpicture}
```

### 103.2.4 保存或使用某个变换矩阵

#### `\pgfgettransform{<macro>}`

本命令定义宏 `<macro>`, 并把当前的变换矩阵保存在宏 `<macro>` 中, 之后可以用命令 `\pgfsettransform` 引用这个变换矩阵。

#### `\pgfsettransform{<macro>}`

`<macro>` 是命令 `\pgfgettransform` 定义的宏, 本命令调用保存在宏 `<macro>` 中变换矩阵。

#### `\pgfgettransformentries{<macro for a>}{<macro for b>}{<macro for c>}{<macro for d>}{<macro for shift x>}{<macro for shift y>}`

本命令定义 6 个宏。设当前变换矩阵是  $\begin{pmatrix} a & b & s \\ c & d & t \\ 0 & 0 & 1 \end{pmatrix}$ , 本命令将  $a, b, c, d, s, t$  分别保存在这 6 个宏中。

#### `\pgfsettransformentries{<a>}{<b>}{<c>}{<d>}{<shiftx>}{<shifty>}`

本命令指定变换矩阵的 6 个元素, 从而重定义变换矩阵, 作用于之后的路径, 实际上等效于

```
\pgftransformreset
\pgftransformcm{<a>}{}{<c>}{<d>}{\pgfpoint{<shiftx>}{<shifty>}}
```

### 103.2.5 坐标变换中的调整

#### `\pgftransformationadjustments`

这个命令针对各种带有“scale”, “slant”的放缩、拉伸变换, 但是最好只针对带有“scale”的放缩变换。当你使用了放缩变换后, 你却可能希望路径命令中的某个特殊点不接受放缩变换, 而你还要路径中的其它点仍然接受放缩变换, 此时你就需要自己计算那个特殊点的坐标, 让它表现出“不接受放缩变换”的样子。

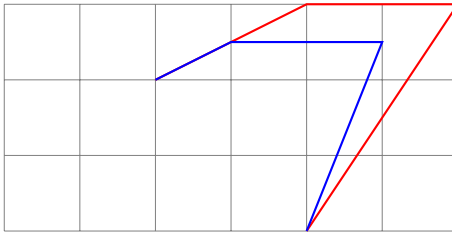
如果你不想自己计算那个特殊点的坐标, 那就使用本命令。本命令需要配合下面两个宏使用。

#### `\pgfhorizontaltransformationadjustment`

假设当前的放缩变换将  $(1, 0)$  变成  $(a, b)$ , 那么这个宏的值就是  $\frac{1}{\|(a,b)\|}$ 。

#### `\pgfverticaltransformationadjustment`

假设当前的放缩变换将  $(0, 1)$  变成  $(a, b)$ , 那么这个宏的值就是  $\frac{1}{\|(a,b)\|}$ 。



```

\begin{tikzpicture}
 \draw [help lines] (0,0) grid (6,3);
 \begin{scope}[scale=2,thick]
 \draw [red] (1,1) -- ++(1,.5) -- ++(1,0) -- (2,0);
 \pgftransformationadjustments
 \draw [blue] (1,1) --
 ++(\pgfhorizontaltransformationadjustment,.5*\pgfverticaltransformationadjustment)
 -- ++(1,0) -- (2,0);
 \end{scope}
\end{tikzpicture}

```

### 103.3 画布变换

画布变换并非完全由 PGF 自己处理, 而是主要由 PDF 或 PostScript 来处理。PGF 只是使用底层命令 `\pgfsys@` 来改变画布变换矩阵。

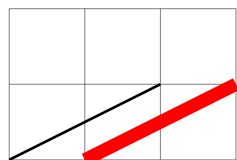
画布变换的有效范围受到 `{pgfscope}` 环境的限制, 这是因为画布变换主要由后台驱动来实现, 而不是由 TeX 或 PGF 来实现。

目前, PGF 不能跟踪画布变换, 当使用画布变换时, PGF 不能再正确计算 `shape` 或 `node` 的各种锚位置及其边界盒子。

PGF 提供数个命令, 可以把坐标变换矩阵转换成画布变换矩阵。

`\pgflowlevelsyncm`

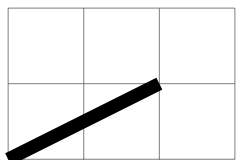
本命令将当前的坐标变换矩阵转成画布变换矩阵, 同时取消坐标变换矩阵, 并将画布变换矩阵作用于之后的路径。



```
\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,2);
 \pgfsetlinewidth{1pt}
 \pgftransformscale{5}
 \draw (0,0) -- (0.4,.2);
 \pgftransformxshift{0.2cm}
 \pgflevelsyncm
 \draw[red] (0,0) -- (0.4,.2);
\end{tikzpicture}
```

### `\pgflevel{<transformation code>}`

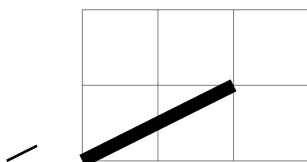
<transformation code> 是坐标变换命令, 本命令将 <transformation code> 确定的变换矩阵转成画布变换矩阵。



```
\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,2);
 \pgfsetlinewidth{1pt}
 \pgflevel{\pgftransformscale{5}}
 \draw (0,0) -- (0.4,.2);
\end{tikzpicture}
```

### `\pgflevelobj{<transformation code>}{<code>}`

<transformation code> 是坐标变换命令, <code> 是绘图命令。本命令创建一个 `{pgfscope}` 环境, 在这个环境中, 先调用命令 `\pgflevel` 来处理 <transformation code>, 然后执行 <code>。



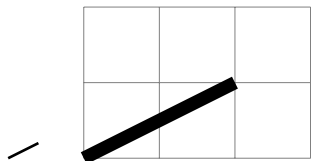
```
\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,2);
 \pgfsetlinewidth{1pt}
 \pgflevelobj{\pgftransformscale{5}}
 {\draw (0,0) -- (0.4,.2);}
 \pgflevelobj{\pgftransformxshift{-1cm}}
 {\draw (0,0) -- (0.4,.2);}
\end{tikzpicture}
```

### `\begin{pgflevelscope}{<transformation code>}`

<environment contents>

### `\end{pgflevelscope}`

这个环境创建一个 `{pgfscope}` 环境, 将 <environment contents> 放入该环境中, 先调用命令 `\pgflevel` 来处理 <transformation code>, 然后执行 <environment contents>。



```

\begin{tikzpicture}
 \draw[help lines] (0,0) grid (3,2);
 \pgfsetlinewidth{1pt}
 \begin{pgfflowlevelscope}{\pgftransformscale{5}}
 \draw (0,0) -- (0.4,.2);
 \end{pgfflowlevelscope}
 \begin{pgfflowlevelscope}{\pgftransformxshift{-1cm}}
 \draw (0,0) -- (0.4,.2);
 \end{pgfflowlevelscope}
\end{tikzpicture}

```

```
\pgfflowlevelscope{<transformation code>}
```

```
<environment contents>
```

```
\endpgfflowlevelscope
```

这是 Plain TeX 中的用法。

```
\startpgfflowlevelscope{<transformation code>}
```

```
<environment contents>
```

```
\stoppgfflowlevelscope
```

这是 ConTeXt 中的用法。

## 103.4 非线性变换

首先载入模块

```
\usepgfmodule{nonlineartransformations} % LATEX and plain TEX and pure pgf
```

```
\usepgfmodule[nonlineartransformations] % ConTeXt and pure pgf
```

### 103.4.1 导引

非线性变换只能作用于坐标点, 对线宽、文字、颜色渐变等无作用。

变换  $(r, d) \rightarrow (d \cos r, d \sin r)$  是个非线性变换, 将直角坐标系中的  $(r, d)$  变成极坐标系中的  $(r, d)$ , 对应直角坐标系中的  $(d \cos r, d \sin r)$ , 这里规定: 在极坐标系中, 点  $(r, d)$  的分量分别代表角度、半径。这个规定可能跟教科书上不同, 但是《手册》中的例子就是这样的。在这个变换下, 直角坐标系中有如下变换关系:

水平线段  $(0, d_0) \text{--} (1, d_0) \rightarrow$  圆弧  $(d_0, 0) \text{ arc } (0:1:d_0)$

竖直线段  $(r_0, 0) \text{--} (r_0, 1) \rightarrow$  线段  $(0, 0) \text{--} (\{\cos(r_0)\}, \{\sin(r_0)\})$

下面以这个变换为例介绍相关命令。

为了理解下面的非线性变换命令, 引入“非线性坐标系”的概念。直角坐标系是一种“线性坐标系”, 极坐标系是一种“非线性坐标系”。因为在极坐标系中, 当角度参数  $\theta$  成线性变化时, 所得到的坐标点不是线性变化的, 即映射

$$(\theta, d) \rightarrow (d \cos \theta, d \sin \theta)$$

不是线性的, 其中的圆括号表示二维点。

非线性变换将直角坐标系变成“非线性坐标系”, 然后在非线性坐标系中画出路径, 从而表现出非线性变换对路径的作用。用下面的代码解释一下:

<非线性变换命令>

```
\draw (0,1)--(1,1);
```

< 非线性变换命令 > 将直角坐标系变成非线性坐标系, 线段  $(0,1)--(1,1)$  看作是非线性坐标系内的“线段”(不是直角坐标系内的), 并在非线性坐标系内画出这个“线段”, 因此这个“线段”可能是弯曲的。例如在极坐标系内画出的“线段”  $(0,2)--(1,2)$  实际上是一段圆弧, 半径是 2, 圆弧的角度范围是  $[0,1]$ 。

这种解释只是为了便于理解非线性变换的作用效果, 不代表真实的程序处理过程。

### 103.4.2 定义并载入一个非线性变换

`\pgftransformnonlinear{<transformation code>}`

代码 <transformation code> 定义一个变换, 本命令引入这个变换, 对之后的路径起作用。本命令所做的定义的有效范围受到 T<sub>E</sub>X 分组的限制, 注意一个 T<sub>E</sub>X 分组内只能使用本命令一次。

当使用本命令后, 路径上的点  $p$  会被传递给 <transformation code>, 执行 <transformation code> 后将  $p$  变成  $q$ 。注意坐标点是由两个分量值 `\pgf@x` 和 `\pgf@y` 确定的, 当这两个值变化时, 相应的坐标点就变化。也就是说, 把点  $p$  的分量值 `\pgf@x` 和 `\pgf@y` 传递给 <transformation code>, 然后改变 `\pgf@x` 和 `\pgf@y` 的值, 使之成为点  $q$  的分量值。因此在编写 <transformation code> 时, 你需要假设点  $p$  是已知的, 即假设给出了 `\pgf@x` 和 `\pgf@y` 的初始值, 然后改变这两个值。这个情况类似命令 `\anchorborder{<code>}` 的 <code> 的编写过程, 你可以用代码运算确定点  $q$ , 也可以直接为 `\pgf@x` 和 `\pgf@y` 赋值。

下面的代码将变换  $(r, d) \rightarrow (d \cos r, d \sin r)$  保存到宏 `\polartransformation` 中:

```
\makeatletter
\def\polartransformation{%
 \pgfmathsincos@{\pgf@sys@tonumber\pgf@x}%
 \pgf@x=\pgfmathresultx\pgf@y % 数值 \pgfmathresultx 与尺寸 \pgf@y 相乘
 \pgf@y=\pgfmathresulty\pgf@y%
}
\makeatother
```

为了理解上面的代码, 看一下文件“pgfmathfunctions.trigonometric.code”对 `\pgfmathsincos` 的定义:

```
\def\pgfmathsincos#1{%
 \pgfmathparse{#1}%
 \expandafter\pgfmathsincos@\expandafter{\pgfmathresult}}
\def\pgfmathsincos@#1{%
 \edef\pgfmath@temparg{#1}%
 \pgfmathsin@{\pgfmath@temparg}\edef\pgfmathresulty{\pgfmathresult}%
 \pgfmathcos@{\pgfmath@temparg}\edef\pgfmathresultx{\pgfmathresult}%
}
```

我们来猜一下这个定义的意思, `\pgfmathsincos@` 是一元函数, 自变量是 `#1`, 此函数的结果是两个值  $\cos(\#1)$  和  $\sin(\#1)$ , 这两个值分别保存在 `\pgfmathresultx` 和 `\pgfmathresulty` 中, 而函数 `\pgfmathsincos` 由 `\pgfmathsincos@` 定义. 我们用这个函数做个计算,

```
0.86603
0.5
```

```
\pgfmathsincos{30}
\pgfmathresultx \
\pgfmathresulty
```

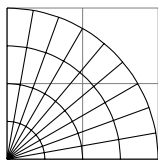
计算结果正如我们的猜测。

文件 “pgfsys.code” 对 `\pgf@sys@tonumber` 的定义是:

```
% The following conversion functions are used to convert from TeX
% dimensions to postscript/pdf points.
%
{\catcode`\p=12\catcode`\t=12\gdef\Pgf@geT#1pt{#1}}

\def\pgf@sys@tonumber#1{\expandafter\Pgf@geT\the#1}
```

下面利用定义的宏 `\polartransformation` 将直角坐标系的网格变成极坐标系的网格。



```
\begin{tikzpicture}
 \draw [help lines] (0,0) grid (2,2);
 \pgftransformnonlinear{\polartransformation}
 \draw (0pt,0mm) grid [xstep=10pt, ystep=5mm] (90pt, 20mm);
\end{tikzpicture}
```

我们另外定义一个非线性变换并保存在 `\xpingfang` 中:

```
\makeatletter
\def\xpingfang{
 \edef\pgfmath@temparg{\pgf@sys@tonumber\pgf@x}
 \pgfmathsign{\pgfmath@temparg}
 \edef\fuhaoy{\pgfmathresult}
 \pgfmathpow{\pgfmath@temparg}{2} % 即 x^2
 \pgfmathmultiply{\pgfmathresult}{0.035146} % 乘以长度单位转换因子
 \edef\pingfang{\pgfmathresult}
 \ifthenelse{\fuhaoy>0}
 {\pgf@y=\pingfang pt}
 {\pgf@y=-\pingfang pt}
}
\makeatother
```



上面的代码相当于变换

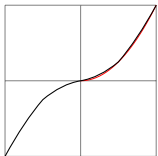
$$(x, y) \rightarrow (x, f(x)), \quad f(x) = \begin{cases} x^2, & x > 0; \\ -x^2, & x \leq 0. \end{cases}$$

上面代码中, 长度单位转换因子 0.035146 是  $\frac{1}{28.45274}$  的近似值。在绘图时, 通常以 cm 为长度单位, 但数学函数在计算时会把单位转换为 pt, 而  $1\text{cm} = 28.45274\text{pt}$ ,

$$1\text{cm} \rightarrow 28.45274\text{pt} \xrightarrow{\text{数值平方}} 809.5584135076\text{pt}$$

因此, `\pgfmathpow{1cm}{2}\pgfmathresult` 得到的是 809.55861, 不是 1。所以为了实现 (cm 下的) 平方运算, 这里必须乘上长度单位转换因子。

然后用 `\xpingfang` 作用于线段  $(-1,0)-(1,0)$ , 如下:



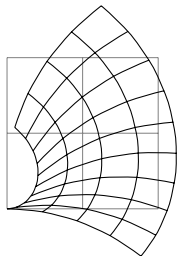
```
\begin{tikzpicture}
\draw [help lines] (-1,-1) grid (1,1);
\draw [red,line width=0.1pt] plot[domain=0:1] (\x,{\x*\x});
\pgftransformnonlinear{\xpingfang}
\draw (-1,0) -- (1,0);
\end{tikzpicture}
```

显然, 与 `plot` 算子画出的抛物线相比, `\xpingfang` 的作用不够光滑。

### 103.4.3 将非线性变换用于一个点

#### `\pgfpointtransformednonlinear{<point>}`

本命令先把当前的线性变换矩阵用于点 `<point>`, 得到 `p`, 然后再将当前的非线性变换用于点 `p`, 得到的是变换后的点 `q`。与命令 `\pgfpointtransformed` 一样, 本命令经常被内部命令调用。



```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\pgftransformrotate{20}
\pgftransformnonlinear{\polartransformation}
\draw (0pt,0mm) grid [xstep=10pt, ystep=5mm] (90pt, 20mm);
\end{tikzpicture}
```

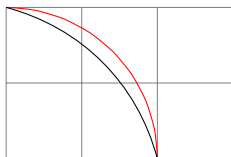
上面例子中, 网格线先被做线性变换——旋转 20 度——网格线不再是横平竖直的了, 然后再做 (前面定义的) 非线性变换 `\polartransformation`, 所以结果是扭曲的。

### 103.4.4 将非线性变换用于一个路径

PGF 会先把线性变换作用于路径上的点, 然后再做非线性变换。

对于一个线段, 无论是用 `\pgfpathlineto` 还是 `\pgfpathclose` 得到的线段, PGF 会把它看作是“退化的”控制曲线, 把它的两个三等分点当作是控制点, 对它的起点、终点、三等分点做变换, 然后用变换后的点构建一段控制曲线。

对于一段控制曲线, PGF 直接对它的起点、终点、控制点做变换, 然后用变换后的点构建一段控制曲线。为了方便, 把这种变换方式称为“变换模式”。对于很多控制曲线来说, 这样的变换过于粗糙, 例如下面的例子:



```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
{
\pgftransformnonlinear{\polartransformation}
\draw [red] (0,20mm) -- (90pt,20mm);
}
\draw (0:20mm) .. controls (30pt:20mm) and (60pt:20mm) .. (90pt:20mm);
\end{tikzpicture}
```

上面例子中, 对水平线段  $(0,20\text{mm})--(90\text{pt},20\text{mm})$  做变换 `\polartransformation`, 按这个变换的本意, 这个水平线段应当变成一段圆弧。但是, 作为一个退化的控制曲线, 这个线段四个构成点是

$$(0,20\text{mm}), (30\text{pt},20\text{mm}), (60\text{pt},20\text{mm}), (90\text{pt},20\text{mm}),$$

在变换 `\polartransformation` 之下, 它们变成

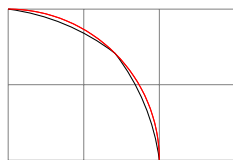
$$(0:20\text{mm}), (30\text{pt}:20\text{mm}), (60\text{pt}:20\text{mm}), (90\text{pt}:20\text{mm}),$$

这 4 个点构建图中黑色的控制曲线, 因此如果直接按“变换模式”对线段做变换就偏离变换本意过多。但实际上水平线段没有变成黑色的控制曲线, 而是变成了红色的圆弧, 这是因为下一命令的作用。

`\pgfsettransformnonlinearflatness{<dimension>} (initially 5pt)`

记控制曲线为  $C(t)$ ,  $t \in [0,1]$ , 可以把  $t$  看作是“时间”。记  $C(t)$  的 4 个控制点是  $P_0, P_1, P_2, P_3$ . PGF 会检查线段  $P_0P_1, P_1P_2, P_2P_3$  的长度, 如果这 3 个线段中的某一个的长度大于本命令指定的尺寸 `<dimension>`, 那么就在  $t = 0.5$  处将  $C(t)$  拆分为两段:  $C_{11}(t)$  和  $C_{12}(t)$ ,  $t \in [0,1]$ ; 如果这 3 个线段的长度都不大于本命令指定的尺寸 `<dimension>`, 那么就将“变换模式”用于  $C(t)$ . 也就是说, PGF 会对  $C(t)$  执行“检查——拆分——变换”的操作, 为了方便, 简称这种操作为“检拆变模式”。然后 PGF 会对  $C_{11}(t)$  和  $C_{12}(t)$  分别使用“检拆变模式”……如此继续下去, 直到完成变换。

这个命令有初始值 `5pt`, 也就是说, 即使你不明确写出本命令, PGF 也会按初始值 `5pt` 来使用本命令, 所以前面的例子中, 线段按“检拆变模式”变成了红色的圆弧, 而不是按“变换模式”变成黑色的控制曲线。



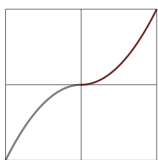
```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
```

```

\draw[red] (0:20mm) arc [start angle=0, end angle=90, radius=2cm];
{
 \pgftransformnonlinear{\polartransformation}
 \pgfsettransformnonlinearflatness{30pt}
 \draw (0,20mm) -- (90pt,20mm);
 \pgfsettransformnonlinearflatness{2pt}
 \draw [red] (0,20mm) -- (90pt,20mm);
}
\end{tikzpicture}

```

我们使用本命令重画前面的关于变换 `\xpingfang` 的例子并修改一下线宽和透明度:



```

\begin{tikzpicture}
 \draw [help lines] (-1,-1) grid (1,1);
 \draw [red,line width=0.2pt] plot[domain=0:1] (\x,{\x*\x});
 \pgftransformnonlinear{\xpingfang}
 \pgfsettransformnonlinearflatness{1pt}
 \draw [draw opacity=0.5,thick](-1,0) -- (1,0);
\end{tikzpicture}

```

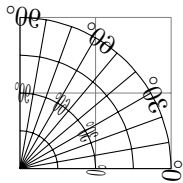
可见 `plot` 算子画出的抛物线与 `\xpingfang` 的作用已经很相近了。

#### 103.4.5 将非线性变换用于文字

前面提到, 非线性变换对文字无作用。不过在非线性变换下使用命令 `\pgftext` 或 `\pgfnode` 时, PGF 会“尽量正确”地显示“文字”。“尽量正确”的意思是让文字尽量体现出非线性变换所规定的位置变化、旋转、镜像翻转、放缩、拉伸、扭曲等特点, 但是 PGF 并非真地将非线性变换作用于文字, 而是用某个近似于非线性变换的线性变换来作用于文字, 所以“扭曲”这个特点就无法实现, 因为线性变换只能把直线变成直线。

另外需要注意, 当用线性变换来近似非线性变换时, 应当明确是在哪个点处做近似。这是因为在非线性坐标系中, 通常只能在局部的小范围内实现“线性近似”。不同点处的扭曲状态、拉伸状态等特点可能很不同, 因此不同点处的线性近似就可能差别很大。

PGF 会根据 `\pgftext` 或 `\pgfnode` 的指向点来确定线性近似。



```

\begin{tikzpicture}
 \draw [help lines] (0,0) grid (2,2);
 \pgftransformnonlinear{\polartransformation}
 \draw (0pt,0mm) grid [xstep=10pt, ystep=5mm] (90pt, 20mm);
 \foreach \angle in {0,30,60,90}
 \foreach \dist in {1,2}
 {
 \pgftransformshift{\pgfpoint{\angle pt}{\dist cm}}
 \pgftext{\angle$^\circ$}
 }
\end{tikzpicture}

```

注意在上面的图形中, 添加的文字被“镜像翻转”了。

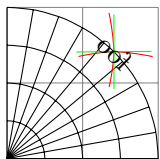
#### 103.4.6 用线性变换近似非线性变换

前面已经提到了用线性变换近似非线性变换, 线性近似的效果只是局部地。下面介绍近似的命令。

##### `\pgfapproximatelineartransformation`

这个命令有两个作用。第一, 清除当前  $\text{T}_\text{E}_\text{X}$  分组内的非线性变换, 只保留线性变换。第二, 在清除非线性变换之前, 本命令会调整 (修改) 线性变换矩阵, 使之能够近似非线性变换在 origin 处的作用效果。也就是说, 本命令得到的线性变换是在 origin 处对非线性变换的近似。

在非线性变换之下, 命令 `\pgftext` 或 `\pgfnode` 会调用本命令来变换文字。



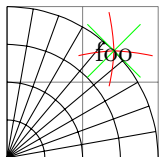
```

\begin{tikzpicture}
 \draw [help lines] (0,0) grid (2,2);
 \pgftransformnonlinear{\polartransformation}
 \draw (0pt,0mm) grid [xstep=10pt, ystep=5mm] (90pt, 20mm);
 \begin{scope}[shift={(45pt,20mm)}]
 \draw [red] (-10pt,-10pt) -- (10pt,10pt);
 \draw [red] (10pt,-10pt) -- (-10pt,10pt);
 \pgfapproximatelineartransformation % 做近似
 \draw [green] (-10pt,-10pt) -- (10pt,10pt);
 \draw [green] (10pt,-10pt) -- (-10pt,10pt);
 \pgftext{foo};
 \end{scope}
\end{tikzpicture}

```

##### `\pgfapproximatelineartranslation`

这个命令得到的线性变换也是在 origin 处对非线性变换的近似, 只不过本命令只保留平移成分, 其它的旋转、镜像、拉伸、放缩等变换成分都去掉。



```

\begin{tikzpicture}
 \draw [help lines] (0,0) grid (2,2);
 \pgftransformnonlinear{\polartransformation}
 \draw (0pt,0mm) grid [xstep=10pt, ystep=5mm] (90pt, 20mm);
 \begin{scope}[shift={(45pt,20mm)}]
 \draw [red] (-10pt,-10pt) -- (10pt,10pt);
 \draw [red] (10pt,-10pt) -- (-10pt,10pt);
 \pgfapproximateonlineartranslation % 做近似
 \draw [green] (-10pt,-10pt) -- (10pt,10pt);
 \draw [green] (10pt,-10pt) -- (-10pt,10pt);
 \pgftext{foo};
 \end{scope}
\end{tikzpicture}

```

#### 103.4.7 非线性变换程序库

```
\usepgflibrary{curvilinear} % LATEX and plain TEX and pure pgf
```

```
\usetikzlibrary{curvilinear} % LATEX and plain TEX when using Tik Z
```

这个程序库利用 Bézier 曲线来定义非线性坐标系, 本程序库也用于实现箭头的弯曲效果, 即当箭头带有 `bend` 选项时, 箭头会随着路径的弯曲而弯曲。

```
\pgfsetcurvilinearbeziercurve{<start>}{<first support>}{<second support>}{<end>}
```

本命令的有效范围受到  $\text{T}_{\text{E}}\text{X}$  分组的限制, 并且在一个  $\text{T}_{\text{E}}\text{X}$  分组内至多能使用本命令一次。为了使用本程序库的其它命令, 你需要先使用本命令。

本命令的 4 个参数是 4 个点, 用于构建一段控制曲线, 记该控制曲线是  $C(t)$ 。本命令还会创建一个数据表, 这个数据表反应的是曲线长度与参数  $t$  的对应:

$$\int_0^t d(C(t)) \rightarrow t,$$

其中  $d(C(t))$  是  $C(t)$  的弧长元素。下面的命令 `\pgfcurvilineardistancetotime` 利用这个数据表做插值计算。

```

\pgfsetcurvilinearbeziercurve
 {\pgfpointorigin}
 {\pgfpoint{1cm}{1cm}}
 {\pgfpoint{2cm}{1cm}}
 {\pgfpoint{3cm}{0cm}}

```

```
\pgfcurvilineardistancetotime{<distance>}
```

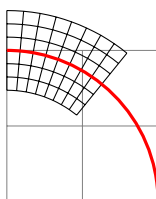
本命令利用上面命令提供的数据表, 计算当控制曲线的长度为 `<distance>` 时所对应的参数  $t$  的值, 并将这个值保存在 `\pgf@x` 中。注意本命令会在运算速度与精度之间作出权衡。本命令在曲线开端处的精度最好, 如果控制曲线的退化程度越高, 则精度越差。

`\pgfpointcurvilinearbezierorthogonal{<distance>}{<offset>}`

这个命令定义一个非线性变换。

先设想这样一个“曲线坐标系”：曲线坐标系的“横轴”就是前面的命令构建的控制曲线  $C(t)$ ，曲线的起点就“横轴”的 0 点；设点  $p$  是横轴上的一个点，在  $p$  处有对应  $p$  的“纵轴”—— $C(t)$  在  $p$  处的法线。在这样一个曲线坐标系中，坐标为  $(x, y)$  的点就可以这样确定：从控制曲线  $C(t)$  的起点开始沿着它运动，运动的长度是  $x$ ，到达点  $p$ ，然后旋转  $90^\circ$ ，直线移动长度  $y$ （可以是负值尺寸），到达点  $q$ ，点  $q$  的坐标就是  $(x, y)$ 。

本命令的两个参数 `<distance>` 和 `<offset>` 是两个尺寸，这两个尺寸代表这个“曲线坐标系”内的点的坐标： $(\begin{smallmatrix} \text{<distance>} \\ \text{<offset>} \end{smallmatrix})$ 。在 `<distance>` 和 `<offset>` 中可以使用 `\pgf@x` 和 `\pgf@y` 这两个值，这两个值代表路径中的点的坐标分量。



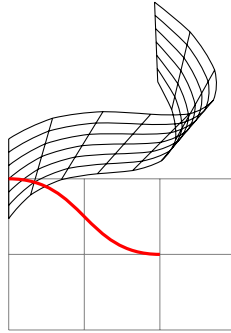
```
\begin{tikzpicture}
 \draw [help lines] (0,0) grid (2,2);
 {
 \pgfsetcurvilinearbeziercurve
 {\pgfpoint{0mm}{20mm}}
 {\pgfpoint{11mm}{20mm}}
 {\pgfpoint{20mm}{11mm}}
 {\pgfpoint{20mm}{0mm}}
 \makeatletter
 \pgftransformnonlinear{
 \pgfpointcurvilinearbezierorthogonal{0.5\pgf@x}{-0.5\pgf@y}}%
 \makeatother
 \draw (0,-30pt) grid [step=10pt] (80pt,30pt);
 }
 \draw[red, very thick] (0mm,20mm) .. controls (11mm,20mm) and (20mm,11mm) .. (20mm,0mm);
\end{tikzpicture}
```

在上面代码中有如下的变换：

$$S: \begin{pmatrix} \text{\pgf@x} \\ \text{\pgf@y} \end{pmatrix} \rightarrow \begin{pmatrix} 0.5\text{\pgf@x} \\ -0.5\text{\pgf@y} \end{pmatrix}.$$

假设被变换路径——网格 `(0,-30pt) grid [step=10pt] (80pt,30pt)` 是点集  $A$ ，在上述变换  $S$  下，点集  $A$  变成点集  $B$ ，然后把点集  $B$  看作是“曲线坐标系”内的点集，在曲线坐标系内画出这个点集即可得到上面的图形。

再看下面的例子:

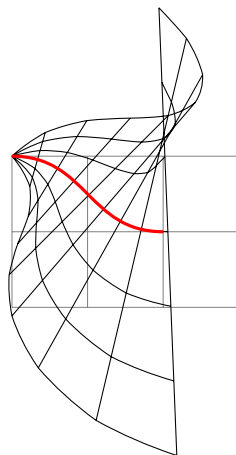


```
\begin{tikzpicture}
 \draw [help lines] (0,0) grid (3,2);
 {
 \pgfsetcurvilinearbeziercurve
 {\pgfpoint{0mm}{20mm}}
 {\pgfpoint{10mm}{20mm}}
 {\pgfpoint{10mm}{10mm}}
 {\pgfpoint{20mm}{10mm}}
 \makeatletter
 \pgftransformnonlinear{
 \pgfpointcurvilinearbezierorthogonal{\pgf@x}{\pgf@x-0.5\pgf@y}}%
 \makeatother
 \draw (0,-30pt) grid [step=10pt] (80pt,30pt);
 }
 \draw[red, very thick](0mm,20mm) .. controls (10mm,20mm) and (10mm,10mm) .. (20mm
 ,10mm);
\end{tikzpicture}
```

在上面代码中有如下的变换:

$$T: \begin{pmatrix} \pgf@x \\ \pgf@y \end{pmatrix} \rightarrow \begin{pmatrix} \pgf@x \\ \pgf@x-0.5\pgf@y \end{pmatrix}.$$

假设被变换路径——网格  $(0,-30\text{pt})$  `grid [step=10pt] (80pt,30pt)` 是点集  $A$ , 在上述变换  $T$  下, 点集  $A$  变成点集  $B$ , 然后把点集  $B$  看作是“曲线坐标系”内的点集, 在曲线坐标系内画出这个点集即可得到上面的图形。



```

\begin{tikzpicture}
 \draw [help lines] (0,0) grid (3,2);
 {
 \pgfsetcurvilinearbeziercurve
 {\pgfpoint{0mm}{20mm}}
 {\pgfpoint{10mm}{20mm}}
 {\pgfpoint{10mm}{10mm}}
 {\pgfpoint{20mm}{10mm}}
 \makeatletter
 \pgftransformnonlinear{
 \pgfpointcurvilinearbezierorthogonal{\pgf@x}{0.035146\pgf@x*\pgf@y}}%
 \makeatother
 \draw (0,-30pt) grid [step=10pt] (80pt,30pt);
 }
 \draw[red, very thick](0mm,20mm) .. controls (10mm,20mm) and (10mm,10mm) .. (20mm,10mm);
\end{tikzpicture}

```

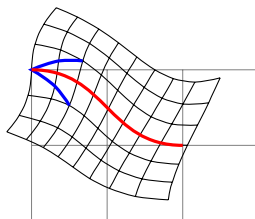
### `\pgfpointcurvilinearbezierpolar{<x>}{<y>}`

本命令定义一个非线性变换。

本命令确定一个“曲线坐标系”，该曲线坐标系的“横轴”就是前面的命令构建的控制曲线  $C(t)$ ，曲线的起点就“横轴”的 0 点  $O$ 。假设点  $Q = (x, y)$  是这个曲线坐标系内的一个点，这个点在曲线坐标系内的位置这样确定：从曲线  $C(t)$  的起点  $O$  开始沿着它运动，运动长度是  $\sqrt{x^2 + y^2}$ ，到达点  $P$ ，然后将  $P$  围绕起点  $O$  旋转，旋转角度是  $\arcsin \frac{(1,0) \times (x,y)}{\|(x,y)\|}$ ，这样就把  $P$  变成了点  $Q = (x, y)$ 。

本命令的参数尺寸 `<x>` 和 `<y>` 代表这个曲线坐标系内的一个点。这个变换依赖欧氏距离  $\sqrt{x^2 + y^2}$  和转角  $\arcsin \frac{(1,0) \times (x,y)}{\|(x,y)\|}$ ，不太直观，下面例子中使用本命令变换网格，大致可以看出变换效果，路径起点处的箭头是“Straight Barb”。





```

\begin{tikzpicture}
 \draw [help lines] (0,0) grid (3,2);
 {
 \pgfsetcurvilinearbeziercurve
 {\pgfpoint{0mm}{20mm}}
 {\pgfpoint{10mm}{20mm}}
 {\pgfpoint{10mm}{10mm}}
 {\pgfpoint{20mm}{10mm}}
 \makeatletter
 \pgftransformnonlinear{\pgfpointcurvilinearbezierpolar\pgf@x\pgf@y}
 \makeatother
 \draw (0,-30pt) grid [step=10pt] (80pt,30pt);
 \draw [blue, very thick] (20pt,10pt) -- (0,0) -- (20pt,-10pt);
 }
 \draw[red, very thick] (0mm,20mm) .. controls (10mm,20mm) and (10mm,10mm) .. (20mm
 ,10mm);
\end{tikzpicture}

```

## 104 图样 Patterns

### 104.1 Overview

参考 §15.5.1, §60.

就像用瓷砖铺地面一样，可以用“图样砖”（tile）铺成图样（tiling pattern）。图样砖本身是个图形，需要在“图样砖坐标系”内画出它，然后在水平方向和垂直方向铺放从而得到图样。你可以假设图样是“足够大”的，当用图样填充路径时，路径会剪切图样。

图样分为两种：inherently colored patterns 和 form-only patterns，前者有固定颜色，其颜色不可变；后者只有固定的轮廓线条，没有固定颜色，其颜色可变。

PGF 会直接把图样映射到图形语言（PostScript, pdf, svg）的图样机制中。

图样本身可能缺少“可修改性”，例如，对于多数图样来说，一旦定义（声明）它后，就不能再改变它的形状、线宽，也不能对它做比例变换。图样的“可修改性”主要与 PGF 本身有关（而非绘图语言），这取决于如何定义图样。有的图样具有某些“可修改性”，当然处理这样的图样会困难一些。

当用图样填充路径时，先确定第一个图样砖的位置，然后在水平方向和垂直方向不断重复铺放图样砖得到图样，然后再用路径剪切图样。所以，第一个图样砖的位置——它的“图样坐标系”原点的位置，对图样

的填充外观有一定的影响。被填充路径有自己的边界盒子，SVG 规定第一个图样砖的“原点”位于该盒子的左上角。而 PostScript 和 pdf 则在整个图形中选定了个固定点，将第一个图样砖的“原点”放在这个固定点上，也就是说，在整个图形中，图样的位置不会因为被填充路径的变化而变化。

## 104.2 声明一个图样

先声明一个图样，然后使用它。

```
\pgfdeclarepatternformonly[<variables>]{<name>}{<bottom left>}{<top right>}{<tile size>}{<code>}
```

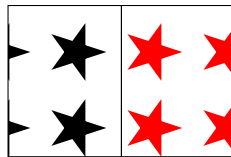
本命令的声明是全局有效的。

本命令声明一个 form-only pattern，其名称是 <name>，实际上本命令定义一个“图样砖”。

当使用这个命令声明图样时，你需要假设有一个“图样砖坐标系”，本命令的参数中涉及的坐标点、绘图代码都是在“图样砖坐标系”内描绘图样砖的。

坐标点 <bottom left> 和 <top right> 构建一个边界盒子，这个边界盒子用来盛放图样砖中的图形。图样砖图形本身是个矩形盒子，图样砖图形盒子的左下角默认在图样砖坐标系的原点，坐标点 <tile size> 是图样砖图形盒子的右上角。注意，尽管图样砖图形本身是个矩形盒子，但它的边界盒子却由前面的两个坐标点指定。边界盒子与图样砖图形盒子可以相差很大，前者“剪切”后者。

<code> 是能够被“protocolled”的 PGF 绘图命令，其中不能涉及颜色。

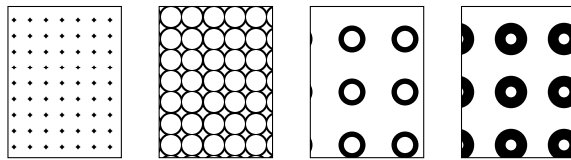


```
\pgfdeclarepatternformonly{stars}{\pgfpointorigin}{\pgfpoint{1cm}{1cm}}{\pgfpoint{1cm}{1cm}}
{
 \pgftransformshift{\pgfpoint{.5cm}{.5cm}}
 \pgfpathmoveto{\pgfpointpolar{0}{4mm}}
 \pgfpathlineto{\pgfpointpolar{144}{4mm}}
 \pgfpathlineto{\pgfpointpolar{288}{4mm}}
 \pgfpathlineto{\pgfpointpolar{72}{4mm}}
 \pgfpathlineto{\pgfpointpolar{216}{4mm}}
 \pgfpathclose%
 \pgfusepath{fill}
}
\begin{tikzpicture}
 \filldraw[pattern=stars] (0,0) rectangle (1.5,2);
 \filldraw[pattern=stars,pattern color=red] (1.5,0) rectangle (3,2);
\end{tikzpicture}
```

本命令的可选项 `<variables>` 是一个用逗号分隔的列表,列表项可以是宏 (macro), 寄存器 (registers), 键 (key)。如果要列出某个 key, 则需要写出它的完整路径。注意所列出的宏和键应当是“简单”的, 即它们仅仅用来保存一个值 (values), 而不是展开为复杂的命令组合或运算。

`<variables>` 中列出的各个项目用在定义图样的 `<code>` 中, 其中的绘图命令将这些项目作为变量, 使得所定义的图样具有“可修改性”。

当 `<variables>` 非空时, 本命令实际上并不创建图样, 只是保存各个变量, 因此 `<variables>` 中的各个项目不必提前定义。当使用图样填充路径时, 必须创建图样, 此时 `<code>` 中的各个变量应当具有自己的值, 因此在使用图样做填充之前, 需要为各个变量赋值。PGF 能够自动区分宏、寄存器、键。

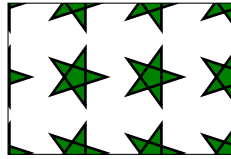


```
\pgfdeclarepatternformonly[/tikz/radius,\thickness,\size]{rings}
{\pgfpoint{-0.5*\size}{-0.5*\size}}
{\pgfpoint{0.5*\size}{0.5*\size}}
{\pgfpoint{\size}{\size}}
{
 \pgfsetlinewidth{\thickness}
 \pgfpathcircle\pgfpointorigin{\pgfkeysvalueof{/tikz/radius}}
 \pgfusepath{stroke}
}
\newdimen\thickness
\tikzset{
 radius/.initial=4pt,
 size/.store in=\size, size=20pt,
 thickness/.code={\thickness=#1},
 thickness=0.75pt
}
\begin{tikzpicture}[rings/.style={pattern=rings}]
 \filldraw [rings, radius=2pt, size=6pt] (0,0) rectangle +(1.5,2);
 \filldraw [rings, radius=2pt, size=8pt] (2,0) rectangle +(1.5,2);
 \filldraw [rings, radius=6pt, thickness=2pt] (4,0) rectangle +(1.5,2);
 \filldraw [rings, radius=8pt, thickness=4pt] (6,0) rectangle +(1.5,2);
\end{tikzpicture}
```

```
\pgfdeclarepatterninherentlycolored[<variables>] {<name>} {<lower left>} {<upper right>}
 {<tile
size>} {<code>}
```

本命令的声明是全局有效的。

本命令声明一个 inherently colored pattern，各个参数的意思与上一个命令类似，只是需要在 `<code>` 中使用 PGF 的颜色命令来设置图样砖图形的颜色。注意不能使用命令 `\color`，此命令不能被“protocollled”。



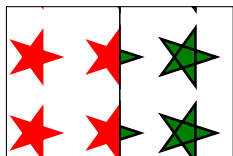
```
\pgfdeclarepatterninherentlycolored{green stars}
{\pgfpointorigin}{\pgfpoint{1cm}{1cm}}
{\pgfpoint{1cm}{1cm}}
{
 \pgfsetfillcolor{green!50!black}
 \pgftransformshift{\pgfpoint{.5cm}{.5cm}}
 \pgfpathmoveto{\pgfpointpolar{0}{4mm}}
 \pgfpathlineto{\pgfpointpolar{144}{4mm}}
 \pgfpathlineto{\pgfpointpolar{288}{4mm}}
 \pgfpathlineto{\pgfpointpolar{72}{4mm}}
 \pgfpathlineto{\pgfpointpolar{216}{4mm}}
 \pgfpathclose%
 \pgfusepath{stroke,fill}
}
\begin{tikzpicture}
 \filldraw[pattern=green stars] (0,0) rectangle (3,2);
\end{tikzpicture}
```

### 104.3 使用图样

声明一个图样后，就可以使用它。

`\pgfsetfillpattern{<name>}{<color>}`

本命令指定图样 `<name>`，提供给之后的填充路径的命令使用。如果 `<name>` 是 inherently colored pattern，则忽略 `<color>`；如果 `<name>` 是 form-only pattern，则用颜色 `<color>` 填充图样 `<name>`，然后再用于填充路径。



```
\begin{tikzpicture}
 \pgfsetfillpattern{stars}{red}
 \filldraw (0,0) rectangle (1.5,2);
 \pgfsetfillpattern{green stars}{red}
 \filldraw (1.5,0) rectangle (3,2);
\end{tikzpicture}
```

## 105 声明、使用外部图形

### 105.1 Overview

L<sup>A</sup>T<sub>E</sub>X 宏包 `graphicx` 提供的命令 `\includegraphics` 可以插入外部图形，这个命令的设计要比 PGF 的插图机制好。不过在某些情况下你可能需要使用 PGF 的插图机制，例如，plain T<sub>E</sub>X 中没有 `graphicx` 宏包。PGF 的插图机制需要后台驱动 `pdftex` 的支持，对于其它驱动，目前来说，PGF 会调用 `graphicx` 宏包来插入图形。

用 PGF 插入外部图形通常需要两个步骤，首先用命令 `\pgfdeclareimage` 声明需要插入的外部图形，然后用命令 `\pgfuseimage` 插入这个图形。另外，命令 `\pgfimage` 可以把“声明图形”和“插入图形”合并为一个步骤。

`\usepackage[draft]{pgf}`

这个命令启用 PGF 的草稿模式，插入的外部图形都会被方框代替。这个模式会检查外部图形是否存在，但不会读入图形。如果没有明确指定插入图形的宽度和高度，就默认宽度和高度为 1cm。

### 105.2 声明外部图形

`\pgfdeclareimage[<options>]{<image name>}{<filename>}`

`<filename>` 是外部图形文件的名称，可以不带扩展名。如果 `<filename>` 不带扩展名，PDF 会自动尝试扩展名 `.pdf`，`.jpg`，`.png`；PostScript 会自动尝试扩展名 `.eps`，`.epsi`，`.ps`。

在 `<options>` 中可以使用以下选项。

`height=<dimension>`

本选项指定插入图形的高度。如果不同时指定宽度，则保持原图的宽高比例。

`width=<dimension>`

本选项指定插入图形的宽度。如果不同时指定高度，则保持原图的宽高比例。

`page=<page number>`

如果文件 `<filename>` 有多个页面，每个页面都是一个图形，那么本选项指定需要插入的图形是第 `<page number>` 页的图形。实际上，使用本选项后，PGF 会先查找名称为

`<filename>.page<page number>.<extension>`

的图形文件，如果找到了就插入这个图形；如果没有找到，再将文件 `<filename>.<extension>` 中的图形插入。

`interpolate=<true or false>`

本选项决定，当插入图形被放大或缩小时，是否让图形“光滑”。默认 false。

`mask=<mask name>`

本选项给插入的图形“带上面具” `<mask name>`，`<mask name>` 是一个具有某种透明度的颜色，需要提前用命令 `\pgfdeclaremask` 声明（见后文）。本选项只对 pdf 有效，有的阅读器不支持这种特性。

`\pgfaliasimage{<new image name>}{<existing image name>}`

`<existing image name>` 是原来的图形文件名称，本命令给这个图形文件起一个新名称 `<new image name>`，这两个名称都指向同一图形文件。例如：

```
\pgfaliasimage{image.!30!white}{image.!25!white}
```

如果你觉得这种图形名称很奇怪，请参考 §86。

### 105.3 使用外部图形

`\pgfuseimage{<image name>}`

声明一个外部图形后，可以用本命令将其插入到文档中。如果想把这个图形插入到 `{pgfpicture}` 环境中，你需要在命令 `\pgftext` 的参数中使用本命令。



```
\pgfdeclareimage[interpolate=true,width=1cm,height=1cm] {image1}{brave-gnu-world-logo}
}
\pgfdeclareimage[interpolate=true,width=1cm] {image2}{brave-gnu-world-logo}
\begin{pgfpicture}
 \pgftext[at=\pgfpoint{5cm}{1cm},left,base]{\pgfuseimage{image1}}
 \pgftext[at=\pgfpoint{3cm}{1cm},left,base]{\pgfuseimage{image2}}
 \pgfpathrectangle{\pgfpoint{5cm}{1cm}}{\pgfpoint{1cm}{1cm}}
 \pgfpathrectangle{\pgfpoint{3cm}{1cm}}{\pgfpoint{1cm}{1cm}}
 \pgfusepath{stroke}
\end{pgfpicture}
```

为了顺利理解下面的内容，请先阅读 §86，不过下面介绍的查找并插入图形的机制未能实现。

假设宏 `\pgfalternateextension`（见下文）的展开值是 `!25!white`，那么在使用本命令插入图形时，PGF 会先查找并插入名称为 `<image name>.!25!white` 的图形文件；如果找不到这个名称的图形文件，PGF 就查找并插入名称为 `<image>.25!white` 的图形文件；如果找不到这个名称的图形文件，PGF 就查找并插入名称为 `<image>.white` 的图形文件；如果找不到这个名称的图形文件，PGF 就查找并插入名称为 `<image>` 的图形文件。

`\pgfalternateextension`

**\pgfimage[<options>]{<filename>}**

本命令的 <options> 中的选项可以使用命令 `\pgfdeclareimage` 的选项。

本命令的作用有二：一是将文件 <filename> 中的图形的名称声明为 `pgflastimage`（这是本命令默认的名称），而且 <options> 中的选项也一并保存在名称 `pgflastimage` 之下；二是把图形 `pgflastimage` 插入到文档中。可见，名称 `pgflastimage` 所指向的图形文件是可变的。使用本命令后，你可以使用命令 `\pgfaliasimage` 来给 `pgflastimage` 另外起一个新名称。



```
\begin{pgfpicture}
 \pgftext[at=\pgfpoint{5cm}{1cm},left,base]
 {\pgfimage[interpolate=true,width=1cm,height=1cm]{brave-gnu-world-logo}}
 \pgfaliasimage{xinmingcheng}{pgflastimage}
 \pgftext[at=\pgfpoint{3cm}{1cm},left,base]
 {\pgfuseimage{xinmingcheng}}
 \pgfpathrectangle{\pgfpoint{5cm}{1cm}}{\pgfpoint{1cm}{1cm}}
 \pgfpathrectangle{\pgfpoint{3cm}{1cm}}{\pgfpoint{1cm}{1cm}}
 \pgfusepath{stroke}
\end{pgfpicture}
```

**105.4 给图形“带面具”****\pgfdeclaremask[<options>]{<mask name>}{<filename>}**

本命令声明一个名称为 <mask name> 的“透明面具”（transparency mask），在 PDF 手册中称之为 `soft mask`。声明 <mask name> 后，就可以在命令 `\pgfdeclareimage` 和 `\pgfimage` 的选项中使用选项 `mask=<mask name>` 了。

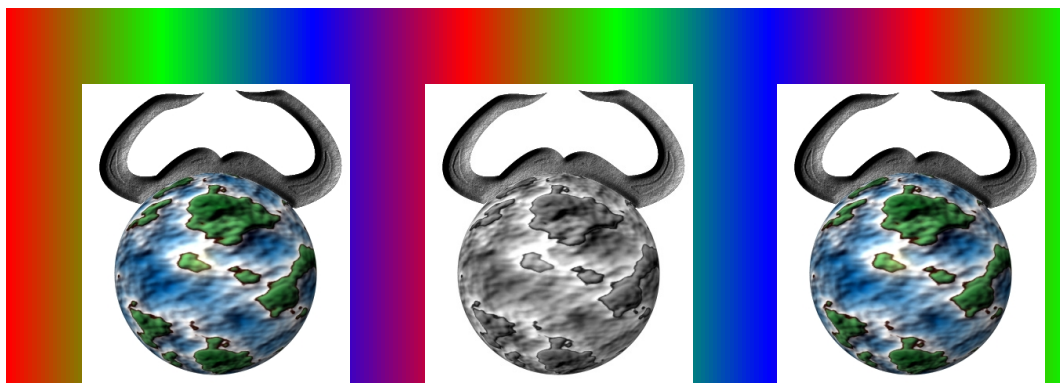
插入到文档中的图形通常是没有透明度的，面具的作用就是让插入图形的各个像素点具有自己的透明度，从而使图形呈现某种视觉效果。使用面具这一“技术”时要注意四点：第一，面具的尺寸应当与插入图形的尺寸一致；第二，插入的图形必须是“像素图”（如 `jpg`，`png` 格式的图），不能是“矢量图”（如 `pdf` 格式的图）；第三，面具必须是真正的“灰度图”（只有单个颜色通道），而不是用 `RGB` 等颜色模式通过混色得到的“黑白灰”图；第四，有的阅读器不支持面具技术。

本命令从文件 <filename> 中读取面具——一个灰度图。灰度图由诸多像素点构成，每个像素点都有自己的灰度，程序会把像素点的灰度转变成透明度，白色像素点是透明的，黑色像素点是不透明的，即灰度与透明度负相关——灰度值越高，透明度越低。这样就把灰度图变成了“透明度图”。然后想象一下——把面具覆盖到图形上后，面具的像素点与图形的像素点实现一一对应，此时面具的像素点的透明度就会变成图形上对应像素点的透明度，然后把面具拿走，但把透明度保留在图形上。这就是面具的作用。

在 <options> 中可以使用下一个选项：

**matte={<color components>}**

<color components> 用于指定一种颜色。



```
\pgfdeclarehorizontalshading{colorful}{5cm}
 {color(0cm)=(red);color(2cm)=(green); color(4cm)=(blue); color(6cm)=(red);
 color(8cm)=(green); color(10cm)=(blue); color(12cm)=(red);color(14cm)=(green)}
\hbox{
 \pgfuses shading{colorful}\hskip-14cm\hskip1cm
 \pgfimage[height=4cm]{brave-gnu-world-logo}\hskip1cm
 \pgfimage[height=4cm]{brave-gnu-world-logo-mask}\hskip1cm
 \pgfdeclaremask{mymask}{brave-gnu-world-logo-mask}
 \pgfimage[mask=mymask,height=4cm,interpolate=true]{brave-gnu-world-logo}}
```

## 107 创建 Plots

本节介绍 plot 模块。

```
\usepgfmodule{plot} % LATEX and plain TEX and pure pgf
\usepgfmodule[plot] % ConTEXt and pure pgf
```

PGF 会自动加载这个模块。如果只是在内核 pgfcore 下使用本模块，那就需要手工调用这个模块。

### 107.1 Overview

大体上讲，PGF 按两个步骤创建 plot：首先生成一个图流（a plot stream），这个流由一些坐标点构成；然后将某个图柄（a plot handler）作用于图流。PGF 预定义了多个图柄，例如 `\pgfplotohandlerlineto`，程序库 `plotohandlers` 也定义了多个图柄，见 §62。图柄对图流的作用是，例如图柄 `\pgfplotohandlerlineto` 对图流的作用是，将 `line-to` 操作作用于图流中的坐标点。

### 107.2 创建图流

注意创建的图流是“全局地”，不受  $\text{T}_\text{E}\text{X}$  分组的限制，你可以在绘图环境之外创建图流。例如可以在绘图环境外写出下面的代码：



```

\pgfplotstreamrecord{\mystream}
\pgfplotstreamstart
\pgfplotstreampoint{\pgfpoint{1cm}{0cm}}
\pgfplotstreampoint{\pgfpoint{2cm}{1cm}}
\pgfplotstreampoint{\pgfpoint{3cm}{1cm}}
\pgfplotstreampoint{\pgfpoint{1cm}{2cm}}
\pgfplotstreamend

```

上面的代码将图流保存在宏 `\mystream` 中，然后在绘图环境内可以用宏 `\mystream` 来引用图流。

### 107.2.1 图流的基本结构

用以下命令构建图流：

- `\pgfplotstreamstart`
- `\pgfplotstreampoint`
- `\pgfplotstreampointoutlier`
- `\pgfplotstreampointundefined`
- `\pgfplotstreamnewdataset`
- `\pgfplotstreamspecial`
- `\pgfplotstreamend`

上面命令的名称大致表明了它们各自的用处。在以上任意两个命令之间可以插入任何代码。下面是个创建图流的例子：

```

\pgfplotstreamstart % 开启一个图流
\pgfplotstreampoint{\pgfpoint{1cm}{1cm}} % 将点 (1cm,1cm) 添加到图流中
\newdimen\mydim % 定义一个新尺寸
\mydim=2cm % 为新尺寸赋值
\pgfplotstreampoint{\pgfpoint{\mydim}{2cm}} % 将点 (2cm,2cm) 添加到图流中
\advance \mydim by 3cm % 将尺寸 \mydim 变成 5cm
\pgfplotstreampoint{\pgfpoint{\mydim}{2cm}} % 将点 (5cm,2cm) 添加到图流中
\pgfplotstreamend % 结束图流

```

#### `\pgfplotstreamstart`

本命令开启一个图流，它会调用内部命令 `\pgf@plotstreamstart`，当前的图柄可能会修改这个内部命令，使之对图流的始点做某些适当调整。

#### `\pgfplotstreampoint{<point>}`

本命令将点 `<point>` 添加到当前的图流中。本命令调用内部命令 `\pgf@plotstreampoint`，当前的图柄可能会修改这个内部命令的作用。

#### `\pgfplotstreampointoutlier{<point>}`

本命令将点 `<point>` 设置为 outlier，即“考虑范围之外的点”，例如，一个 outlier 点可能代表无穷远点，或者曲线的间断点。

对 outlier 点的处理方式由以下选项决定：

`/pgf/handle outlier points in plots=<how>` (无默认值, 初始值 `jump`)  
或者 `/tikz/handle outlier points in plots=<how>`

这个选项中的 `<how>` 可以取以下值：

- `plot`, 这个值导致“画出 outlier 点”，相当于 `\pgfplotstreampoint<outlier 点 >`。
- `ignore`, 这个值导致“忽略 outlier 点”。
- `jump`, 这个值导致内部宏 `\pgf@plotstreamjump` 被调用，会在图流中制造一个“缺口”。举例来说，如果下面线段：

```
(0,0)--(1,0)--(1.2,0)--(1.4,0)--(2,0)
```

中的点 `(1.2,0)` 是被“`jump`”的点，那么这个线段就被分成了两端：

```
(0,0)--(1,0) (1.2,0) (1.4,0)--(2,0)
```

### `\pgfplotstreampointundefined`

这个命令生成一个“未定义点”(`undefined`)，这个点是无法画出的，因此不需要指定它的坐标。“未定义点”的作用是引起某种动作，这个动作由以下选项规定：

`/pgf/handle undefined points in plots=<how>` (无默认值, 初始值 `jump`)  
或者 `/tikz/handle undefined points in plots=<how>`

这个选项规定对 `undefined` 点的处理方式，`<how>` 可以取以下值：

- `ignore`, 这个值导致“忽略 `undefined` 点”。
- `jump`, 这个值导致内部宏 `\pgf@plotstreamjump` 被调用，会在图流中制造一个“缺口”。

### `\pgfplotstreamnewdataset`

本命令暂时中断图流的创建，开启一个新的数据点集。本命令会引起一个动作，这个动作由以下选项规定：

`/pgf/handle new data sets in plots=<how>` (无默认值, 初始值 `jump`)  
或者 `/tikz/handle new data sets in plots=<how>`

这个选项规定对命令 `\pgfplotstreamnewdataset` 的处理方式，`<how>` 可以取以下值：

- `ignore`, 忽略这个命令。
- `jump`, 这个值导致内部宏 `\pgf@plotstreamjump` 被调用，会在图流中制造一个“缺口”。

### `\pgfplotstreamspecial{<text>}`

这个命令导致内部宏 `\pgf@plotstreamspecial` 被调用，`<text>` 是这个内部宏的参数，这个宏可以向图柄传递某些信息。所有“正常的”图柄都会忽略这个命令。

### `\pgfplotstreamend`

本命令结束一个图流。这个命令调用内部宏 `\pgf@plotstreamend`，执行某些必要的收尾清理工作。

注意，图流不会被缓冲(`buffered`)，图流中的坐标点会在读取时被立即处理。使用“记录图柄”(recording handler)可以记录一个图流。

## 107.2.2 生成图流的命令

```
\pgfplotxyfile{<filename>}
```

```
\pgfplotxyzfile{<filename>}
```

第一个命令用于创建二维绘图流，第二个命令用于创建三维图流。<filename> 是某个外部文件的名称，通常这个文件是个文本文件。本命令读取 <filename> 中的数据，将其中的数据变成图流。当本命令读取文件 <filename> 时，首先会引入命令 `\pgfplotstreamstart` 开启一个图流，然后逐行读取 <filename> 中的数据，读完后再添上命令 `\pgfplotstreamend` 结束图流。

文件 <filename> 的格式及其转换如下所示：

<pre>% Some comments 0 Nan u 1 1 some text 3 9 4 16 o 5 25 oo</pre>	<pre>\pgfplotstreamstart \pgfplotstreamnewdataset \pgfplotstreampointundefined \pgfplotstreampoint{\pgfpointxy{1}{1}} \pgfplotstreampoint{\pgfpointxy{3}{9}} \pgfplotstreamnewdataset \pgfplotstreampointoutlier{\pgfpointxy{4}{16}} \pgfplotstreampoint{\pgfpointxy{5}{25}} \pgfplotstreamend</pre>
<pre># Some comments 2 -5 1 first entry 2 -.2 2 o 2 -5 2 third entry</pre>	<pre>\pgfplotstreamstart \pgfplotstreamnewdataset \pgfplotstreampoint{\pgfpointxyz{2}{-5}{1}} \pgfplotstreampointoutlier{\pgfpointxyz{2}{-.2}{2}} \pgfplotstreampoint{\pgfpointxyz{2}{-5}{2}} \pgfplotstreamend</pre>

其中以 % 或 # 开头的行是注释行，注释行会被当作是空行，空行都被转为命令 `\pgfplotstreamnewdataset`。非空行必须是以下 6 种形式之一：

```
数字 □ 数字
数字 □ 数字 □ 文字符号
数字 □ 数字 □ 数字
数字 □ 数字 □ 数字 □ 文字符号
符号 □ 符号 □ u
符号 □ 符号 □ 符号 □ u
```

解释一下上面 6 种形式：

- 如果某行有 2 个数字，则这 2 个数字代表一个二维点，会被转到 xy 坐标系中；

- 如果某行有 3 个数字，则这 3 个数字代表一个三维点，会被转到 xyz 坐标系中；
- 如果某一行以 “\u” 结尾，那么该行会被转成命令 `\pgfplotstreampointundefined`，即被转成 undefined 点，因此该行可以不严格地包含数字；
- 如果某一行的“文字符号”是单个字母“o”或以“\u”结尾，例如

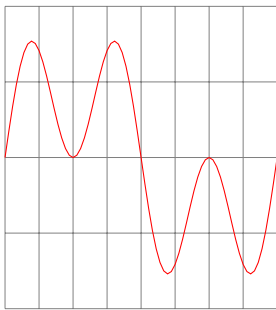
```
5 25 foo o
```

那么该行会被转成 `\pgfplotstreampointoutlier{\pgfpointxy{4}{16}}`，即被转成 outlier 点；

- 其它形式的“文字符号”会被忽略。

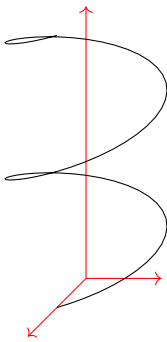
`\pgfplotfunction{<variable>}{<sample list>}{<function point>}`

本命令利用函数创建图流，先看一个例子：



```
\begin{tikzpicture}[x=3.6cm/360]
 \draw[xstep=4.5mm,help lines](0,-2) grid (360,2);
 \pgfplothandlerlineto
 \pgfplotfunction{\t}{0,5,...,360}
 {\pgfpointxy{\t}{sin(\t)+sin(3*\t)}}
 \pgfsetstrokecolor{red}
 \pgfusepath{stroke}
\end{tikzpicture}
```

本命令中的 `<variable>` 和 `<sample list>` 相当于 `\foreach<variables> in <list> ...` 中的变量和变量取值列表，其格式要符合 `\foreach` 语句的限制，不过本命令只能使用一个变量。变量用于 `<function point>` 的数学表达式中参与运算。



```
\begin{tikzpicture}[y=3.6cm/360]
 \foreach \m in {(1,0,0),(0,360,0),(0,0,2)}
 \draw [->,red](0,0,0)--\m;
 \pgfplothandlerlineto
 \pgfplotfunction{\y}{0,5,...,360}
 {\pgfpointxyz{sin(2*\y)}{\y}{cos(2*\y)}}
 \pgfusepath{stroke}
\end{tikzpicture}
```

若 `<function point>` 中的数学表达式很复杂，则可能导致处理过程变慢。

`\pgfplotgnuplot[<prefix>]{<function>}`

本命令调用外部程序 gnuplot 来创建函数 `<function>` 的图流。

### 107.3 图柄

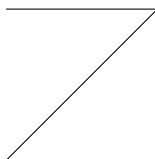
图柄 (plot handler) 决定用什么方式将图流中的点联系起来。你必须在开启图流之前写出所用的图柄。图柄通过设置或者重置 `\pgf@plotstreamstart`, `\pgf@plotstreampoint`, `\pgf@plotstreamend` 来起作用。

注意，图流是全局定义的，因此图柄的作用不受  $\text{T}_{\text{E}}\text{X}$  分组或者子环境 `{scope}` 的限制。一个图柄对之后的各个图流都会有作用，直到更换图柄。

下面是 PGF 预定义的几个图柄，另外程序库 `pgflibraryplohandlers` 也定义了大量图柄，见 S62.

### `\pgfplotshandlerlineto`

这个图柄将命令 `\pgfpathlineto` 用于图流中的点，当然图流的第一个点除外。



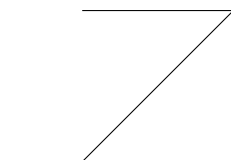
```
\begin{pgfpicture}
 \pgfpathmoveto{\pgfpointorigin}
 \pgfplotshandlerlineto
 \pgfplotstreamstart
 \pgfplotstreampoint{\pgfpoint{1cm}{0cm}}
 \pgfplotstreampoint{\pgfpoint{2cm}{1cm}}
 \pgfplotstreampoint{\pgfpoint{3cm}{2cm}}
 \pgfplotstreampoint{\pgfpoint{1cm}{2cm}}
 \pgfplotstreamend
 \pgfusepath{stroke}
\end{pgfpicture}
```

### `\pgfsetmovetofirstplotpoint`

本命令将 `move-to` 操作作用于图流的第一个点，因此本命令或者把图流创建为当前路径的子路径，或者把图流创建为一个新路径。在默认下，画线的图柄，例如 `\pgfplotshandlerlineto` 会调用本命令。

### `\pgfsetlinetofirstplotpoint`

本命令将 `line-to` 操作作用于图流的第一个点，因此本命令之前的坐标点与图流的第一个点会被直线段连接起来。也就是说，本命令将图流添加到当前路径中，成为当前路径的连续延伸。



```
\begin{pgfpicture}
 \pgfpathmoveto{\pgfpointorigin}
 \pgfsetlinetofirstplotpoint
 \pgfplotshandlerlineto
 \pgfplotstreamstart
 \pgfplotstreampoint{\pgfpoint{1cm}{0cm}}
 \pgfplotstreampoint{\pgfpoint{2cm}{1cm}}
 \pgfplotstreampoint{\pgfpoint{3cm}{2cm}}
 \pgfplotstreampoint{\pgfpoint{1cm}{2cm}}
 \pgfplotstreamend
 \pgfusepath{stroke}
\end{pgfpicture}
```

### `\pgfplotshandlerpolygon`

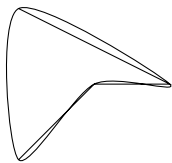
这个图柄类似 `\pgfplotshandlerlineto`，只是本命令会在图流的最后一个点之后加上命令 `\pgfpathclose`，将图流作成一个封闭的多边形。

### `\pgfplotshandlerdiscard`

这个图柄直接把后面的图流“扔掉”。

### `\pgfplotshandlerrecord{<macro>}`

`<macro>` 是个宏，不需要提前定义。写出本命令后，本命令之后的图流会保存在宏 `<macro>` 中，然后用宏 `<macro>` 调用这个图流。注意本命令只能保存一个图流，如果本命令之后有数个图流，那么本命令只保存最后一个图流。



```
\begin{pgfpicture}
 \pgfplotshandlerrecord{\mystream}
 \pgfplotstreamstart
 \pgfplotstreampoint{\pgfpoint{1cm}{0cm}}
 \pgfplotstreampoint{\pgfpoint{2cm}{1cm}}
 \pgfplotstreampoint{\pgfpoint{3cm}{1cm}}
 \pgfplotstreampoint{\pgfpoint{1cm}{2cm}}
 \pgfplotstreamend
 \pgfplotshandlerlineto
 \mystream
 \pgfplotshandlerclosedcurve
 \mystream
 \pgfusepath{stroke}
\end{pgfpicture}
```

## 107.4 定义新图柄

### `\pgfdeclareplotshandler{<macro>}{<arguments>}{<configuration>}`

本命令定义一个图柄，并将图柄保存在宏 `<macro>` 中。`<arguments>` 是 `#1, #1#2#3` 这种形式的变量列举，对应宏 `<macro>` 的变量。`<configuration>` 是个“键值对”列表，具体规定图柄的作用，其中可以使用 `<arguments>` 列出的变量。

我们定义一个图柄 `\myhandler`：

```
\pgfdeclareplotshandler{\myhandler}{#1}{...}
...
\myhandler{foo}
\pgfplotstreamstart
...
\pgfplotstreamend
```

下面以图柄 `\myhandler` 为例介绍有关选项。

用在 <configuration> 中的选项如下。

**start=<code>**

当使用图柄 `\myhandler` 处理图流时, 若图柄遇到 `\pgfplotstreamstart` 则执行 <code>, <code> 中可以使用 <arguments> 列出的变量。

Hi 某.Bye 某.  
Hi 某某.Bye 某某.

```
\pgfdeclareplotheadhandler{\myhandler}{#1}{
 start = Hi #1.,
 end = Bye #1.,
}
\myhandler{某}
\pgfplotstreamstart
\pgfplotstreamend \\
\myhandler{某某}
\pgfplotstreamstart
\pgfplotstreamend
```

**end=<code>**

类似选项 `start`, 图柄会在遇到 `\pgfplotstreamend` 时执行 <code>.

**point=<code>**

当使用图柄 `\myhandler` 处理图流时, 若图柄遇到 `\pgfplotstreampoint` 或 `\pgfplotstreampointoutlier`, 则执行 <code>. <code> 中可以使用 <arguments> 列出的变量。如果 <code> 中使用变量 `##1`, 则这个变量符号会引用 `\pgfplotstreampoint{<point>}` (或 `\pgfplotstreampointoutlier{<point>}`) 中的坐标 <point>.

Hi 某.  
**nice** 天气  
Bye 某.  
**nice** 天气

```
\pgfdeclareplotheadhandler{\myhandler}{#1#2}{
 start = Hi #1.,
 end = Bye #1.,
 point = nice #2
}
\myhandler{某}{天气}
\pgfplotstreamstart \\
\color{red}
\pgfplotstreampoint{} \\
\color{cyan}
\pgfplotstreamend \\
\pgfplotstreampoint{}
```

**jump=<code>**

命令 `\pgfplotstreampointoutlier,\pgfplotstreampointundefined,\pgfplotstreamnewdataset,`

都可能引起“jump”选项发生作用，本选项的 `<code>` 规定“jump”选项的动作。

**special=<code>**

当使用图柄 `\myhandler` 处理图流时，若图柄遇到 `\pgfplotstreamspecial{<something>}`，则执行 `<code>`。如果 `<code>` 中使用变量 `##1`，则这个变量符号会引用 `\pgfplotstreamspecial{<something>}` 中的 `<something>`。

**point macro=<some macro>**

`<some macro>` 是个宏，使用这个选项后，图流命令 `\pgfplotstreampoint` 会调用宏 `<some macro>`，而内部命令 `\pgf@plotstreampoint` 会等同于宏 `<some macro>`。在 `<some macro>` 中至多能使用一个变量 `#1`，这个变量引用 `\pgfplotstreampoint{<point>}` 中的 `<point>`。

**special macro=<some macro>**

`<some macro>` 是个宏，使用这个选项后，图流命令 `\pgfplotstreamspecial` 会调用宏 `<some macro>`。在 `<some macro>` 中至多能使用一个变量 `#1`，这个变量引用 `\pgfplotstreamspecial{<something>}` 中的 `<something>`。

**start macro=<some macro>**

`<some macro>` 是个宏，使用这个选项后，图流命令 `\pgfplotstreamstart` 会调用宏 `<some macro>`。在 `<some macro>` 中不能使用变量。

**end macro=<some macro>**

`<some macro>` 是个宏，使用这个选项后，图流命令 `\pgfplotstreamend` 会调用宏 `<some macro>`。在 `<some macro>` 中不能使用变量。

**jump macro=<some macro>**

`<some macro>` 是个宏，使用这个选项后，选项“jump”会调用宏 `<some macro>`。在 `<some macro>` 中不能使用变量。

## 108 图层

### 108.1 Overview

PGF 提供分层绘图机制。设想有两块玻璃板，每一块玻璃板上都画有图形，把两块玻璃板上下叠放，那么上层玻璃板的图形会（全部或部分地）遮挡下层玻璃板的图形。下面的玻璃板可以看作是“背景图层”（background layer），上面的玻璃板可以看作是“前端图层”（foreground layer）。在 PGF 中，你可以声明多个图层并规定它们的上下叠放次序，每个图层上都可以画出属于“本图层”的图形，按图层的叠放次序，这些图形有其“遮挡次序”。

### 108.2 声明图层

PGF 的图层都有自己的名称，预定义的图层是 `main`，如果不声明其它图层，所有绘图环境都在 `main` 层上表现出来。



**`\pgfdeclarelayer{<name>}`**

这个命令声明一个图层，其名称为 `<name>`。你可以多次使用本命令声明多个图层。

这个命令的声明是全局有效的。

**`\pgfsetlayers{<layer list>}`**

`<layer list>` 是由已经声明的图层名称构成的列表，名称之间用逗号分隔，其中必须包含图层 `main`。本命令按照这个列表次序，规定各个图层（自下而上）的叠放次序。例如：

```
\pgfdeclarelayer{background layer}
\pgfdeclarelayer{foreground layer}
\pgfsetlayers{background layer,main,foreground layer}
```

这个命令可以用在绘图环境之外，也可以用在绘图环境内。如果本命令用在绘图环境内，那么要确保本命令与 `\end{pgfpicture}` 之间没有 `TEX` 分组的结束标志，也不能有其它以 `\end` 开头的控制语句。当然你可以把本命令放在 `\end{pgfpicture}` 的前面，这也是有效的。

### 108.3 在图层上绘图

在绘图环境中，如果不指明一个绘图命令属于哪个图层，那么这个绘图命令就属于 `main` 层。

**`\begin{pgfonlayer}{<layer name>}`**

`<environment contents>`

**`\end{pgfonlayer}`**

本环境用在绘图环境 `{pgfpicture}` 内或者子绘图环境 `{pgfscope}` 内。

`<layer name>` 是已经被“排序”的图层名称，`<environment contents>` 是绘图命令。本环境指定 `<environment contents>` 属于图层 `<layer name>`。

如果在绘图环境内针对某个图层多次使用这个环境，那么 `<environment contents>` 会在该图层上累计。

注意不能针对 `main` 使用本环境。

**`\pgfonlayer{<layer name>}`**

`<environment contents>`

**`\endpgfonlayer`**

这是 plain `TEX` 中的用法。



```
\pgfdeclarelayer{background layer}
\pgfdeclarelayer{foreground layer}
\pgfsetlayers{background layer,main,foreground layer}
\begin{tikzpicture}
```

```


\fill[blue] (0,0) circle (1cm); % 在 main 层上
\begin{pgfonlayer}{background layer} % 在 background layer 层上
 \fill[yellow] (-1,-1) rectangle (1,1);
\end{pgfonlayer}
\begin{pgfonlayer}{foreground layer} % 在 foreground layer 层上
 \node[white] {foreground};
\end{pgfonlayer}
\begin{pgfonlayer}{background layer} % 在 background layer 层上
 \fill[black] (-.8,-.8) rectangle (.8,.8);
\end{pgfonlayer}
\fill[blue!50] (-.5,-1) rectangle (.5,1); % 在 main 层上
\end{tikzpicture}

```

## 109 颜色渐变

### 109.1 Overview

颜色渐变就是在某个区域中，一种或数种颜色逐渐变化、平滑过渡。有多种渐变方式：横向渐变，纵向渐变，辐射渐变，函数渐变。PGF 会把渐变区域作成一个小盒子，这个小盒子可以直接放在文档中，不必放在绘图环境中。先用声明命令——如声明横向渐变的命令 `\pgfdeclarehorizontalshading`——声明一个渐变样式，然后用命令 `\pgfuseshading` 使用这个渐变样式，即在一个盒子中实现这个样式并将这个盒子插入文档中。例如：



```

\pgfdeclarehorizontalshading{myshadingA}{1cm}
 {rgb(0cm)=(1,0,0); color(2cm)=(green); color(4cm)=(blue)}
\pgfuseshading{myshadingA}

```

上面例子中声明一个横向渐变。设想有一个“渐变坐标系”，参数 `rgb(0cm)=(1,0,0)` 指定渐变坐标系横轴的 0cm 点处的颜色是 rgb 颜色模式下的红色，其颜色值是 (1,0,0)；参数 `color(2cm)=(green)` 指定横轴的 2cm 点处是绿色。参数 `color(4cm)=(blue)` 指定横轴的 4cm 点处是蓝色。这样渐变盒子的宽度就是 4cm，高度被指定为 1cm。这个渐变盒子表现为一个“颜色条”，呈现“红、绿、蓝”三色渐变。**注意在指定各个位置的颜色时所用的标点符号，位置和颜色要用圆括号括起来，各位置颜色之间用分号分隔。**

如果想在环境 `{pgfpicture}` 中画出渐变盒子，就需要把渐变盒子用作命令 `\pgftext` 的参数。在绘图环境中，你可以旋转一个渐变盒子，也可以用某个路径剪切它。

命令 `\pgfshadepath` 必须用在绘图环境中，它为路径作出颜色渐变效果，即用某个渐变样式填充路径。目前，PGF 的颜色渐变功能支持 rgb 颜色模式和 gray 颜色模式。

### 109.2 声明渐变样式

#### 109.2.1 横向渐变与纵向渐变

```
\pgfdeclarehorizontalshading[<color list>]{<shading name>}{<shading height>}{<color specification>}
```

这个命令声明一个横向渐变，将这个渐变定义在一个矩形之内，即定义一个横向变化的“颜色条”。

<shading name> 是渐变的名称；<shading height> 是颜色条的高度；<color specification> 是沿着横轴的颜色设置，如前例所示，其中起点与终点的距离决定颜色条的宽度。在指定 <color specification> 时应当设想有一个“渐变坐标系”，参照这个坐标系指定渐变样式。

可选项 <color list> 是个颜色列表，其中的颜色可以是尚未定义的颜色，其作用见下面的例子：

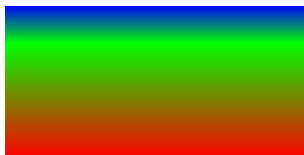


```
\pgfdeclarehorizontalshading[mycolorA,mycolorB]{myshadingA}
 {1cm}{rgb(0cm)=(1,0,0); color(2cm)=(mycolorA); color(4cm)=(mycolorB)}
\definecolor{mycolorA}{rgb}{0.5,0.1,0.3}
\colorlet{mycolorB}{cyan!10!yellow}
\pgfuses shading{myshadingA}
\colorlet{mycolorB}{cyan!80!orange}
\pgfuses shading{myshadingA}
```

在声明一个颜色渐变时，PGF 实际上只是保存定义，并不计算颜色值，因此 <color list> 中的颜色名称可以是“没有颜色的”。或者说，如果 <color list> 中的颜色名称是尚未被定义的，那么本命令就把这个名称声明为“颜色名称”，只不过颜色值为“空”。只有在使用命令 \pgfuses shading 时，PGF 才会检查需要的颜色值并计算出渐变颜色值，因此在使用命令 \pgfuses shading 之前要确保需要的颜色都有颜色值。可见使用可选项 <color list> 的方便之处在于，可以随时更改渐变中的颜色。

```
\pgfdeclareverticalshading[<color list>]{<shading name>}{<shading width>}{<color specification>}
```

这个命令声明一个纵向渐变，确定一个纵向变化的颜色条，<shading name> 是渐变的名称，<shading width> 是颜色条的宽度，<color specification> 沿着纵轴指定颜色。可选项 <color list> 的用处与前一个命令相同。



```
\pgfdeclareverticalshading{myshadingC}{4cm}
 {rgb(0cm)=(1,0,0); rgb(1.5cm)=(0,1,0); rgb(2cm)=(0,0,1)}
\pgfuses shading{myshadingC}
```

## 109.2.2 辐射渐变

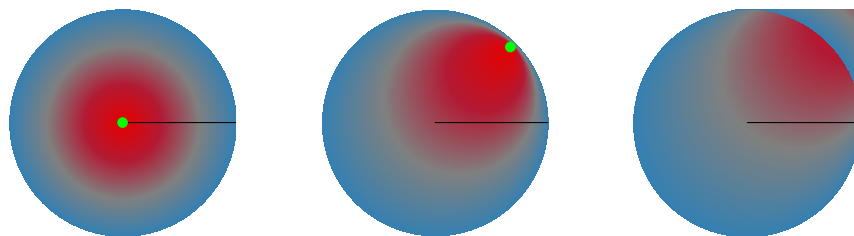
辐射渐变的意思是，颜色以某个点为中心，向外围渐变。

```
\pgfdeclareradialshading[<color list>]{<shading name>}{<center point>}{<color specification>}
```

这个命令声明一个辐射渐变。<shading name> 是所声明的辐射渐变的名称。<color list> 的作用与前面的命令类似。

<color specification> 是沿着横轴给出的“点—颜色”列表，它指定一个线段以及该线段上某些点处的颜色，这个线段就作为半径构造一个“辐射圆”，用于呈现辐射渐变。

点 <center point> 是辐射渐变的中心，如果这个点不是“辐射圆”的圆心，那么可能出现意外效果。



```
\pgfkeys{
 fushejianbian/.code={
 \pgfdeclareradialshading{sphere}{\pgfpoint{#1cm}{#1cm}}%
 {rgb(0cm)=(0.9,0,0);
 rgb(0.5cm)=(0.7,0.1,0.2);
 rgb(1cm)=(0.5,0.5,0.5);
 rgb(1.5cm)=(0.2,0.5,0.7)}
 \tikz{\pgftext{\pgfuseshading{sphere}}}
 \draw (0,0)--(1.5,0);
 \fill [green](#1,#1)circle(2pt);}
 }
}
\pgfkeys{fushejianbian={0}} \hspace{.5cm}
\pgfkeys{fushejianbian={1}} \hspace{.5cm}
\pgfkeys{fushejianbian={2}}
```

### 109.2.3 函数渐变

```
\pgfdeclarefunctionalshading[<color list>]{<shading name>}{<upper right corner>}{<init code>}{<type 4 function>}
```

注意，这种渐变可能给渲染器带来很大负担，也可能无法正确显示。

这个命令创建一个函数渐变。<shading name> 是渐变的名称。点 <lower left corner> 和点 <upper right corner> 确定一个矩形，本命令计算这个矩形内各点的颜色。

可选选项 <color list> 的用处与前面的命令类似。

<init code> 通常与 <color list> 配合使用，见后文命令的例子。

这里的函数 <type 4 function> 指的是《PDF Reference》(version 1.7) 的 §3 中所说的 type 4 类型的函数, 这类函数是 PostScript 语言中的函数, 是用一些基本的“算子”构造的。PostScript 语言中有很多算子, 这里只能使用某些限定的算子来构造 type 4 类型的函数, 在后文对此有说明。程序通过执行函数 <type 4 function> 来计算坐标点所对应的颜色, 执行函数计算的是 PDF 渲染器, 而不是 PGF 或 T<sub>E</sub>X, 所以这里使用 type 4 类型的函数, 被执行的函数不能过于复杂。

这里输入给 type 4 类型函数的值都必须是不带单位的实数值, 函数的输出也是不带单位的数值。函数逐个点进行计算。函数计算一个坐标点的颜色时, 程序先把两个坐标分量中的长度单位转换成 bp, 然后去掉长度单位作成两个实数, 然后再输入给函数; 之后经过函数计算, 输出的是 rgb 模式下的 3 个颜色参数, 对应该点的颜色。

下面是个例子:



```
\pgfdeclarefunctionalshading{twospots}
 {\pgfpointorigin}{\pgfpoint{4cm}{4cm}}{}
 {
 2 copy
 45 sub dup mul exch
 40 sub dup mul 0.5 mul add sqrt
 dup mul neg 1.0005 exch exp 1.0 exch sub
 3 1 roll
 70 sub dup mul .5 mul exch
 70 sub dup mul add sqrt
 dup mul neg 1.002 exch exp 1.0 exch sub
 1.0 3 1 roll
 }
\pgfusesshading{twospots}
```

看一下上面例子中的 type 4 函数计算的是什么。上面例子中, 点 `\pgfpointorigin` 和 `\pgfpoint-{4cm} {4cm}` 确定一个矩形, type 4 函数逐个计算这个矩形内点的颜色。假设矩形内任意一个点的坐标是  $(x, y)$ , 将坐标分量作成数值堆栈 `x y` 提供给函数, 然后由函数中的算子按次序进行处理。函数对 `x y` 的处理就是:

```
x y 2 copy
 45 sub dup mul exch
 40 sub dup mul 0.5 mul add sqrt
 dup mul neg 1.0005 exch exp 1.0 exch sub
 3 1 roll
 70 sub dup mul .5 mul exch
 70 sub dup mul add sqrt
 dup mul neg 1.002 exch exp 1.0 exch sub
 1.0 3 1 roll
```

下面我们分析前几个算子的运算过程，为了方便其中使用了数学公式。

```

x┌y┌2┌copy → x┌y┌x┌y
45┌sub → x┌y┌x┌y-45
dup → x┌y┌x┌y-45┌y-45
mul → x┌y┌x┌(y-45)2
exch → x┌y┌(y-45)2┌x
40┌sub → x┌y┌(y-45)2┌x-40
dup → x┌y┌(y-45)2┌x-40┌x-40
mul → x┌y┌(y-45)2┌(x-40)2
0.5┌mul → x┌y┌(y-45)2┌0.5(x-40)2
add → x┌y┌(y-45)2 + 0.5(x-40)2
sqrt → x┌y┌√((y-45)2 + 0.5(x-40)2)
dup mul → x┌y┌(y-45)2 + 0.5(x-40)2
neg → x┌y┌-(y-45)2 - 0.5(x-40)2
1.0005┌exch → x┌y┌1.0005┌-(y-45)2 - 0.5(x-40)2
exp → x┌y┌1.0005-(y-45)2-0.5(x-40)2
1.0┌exch → x┌y┌1.0┌1.0005-(y-45)2-0.5(x-40)2
sub → x┌y┌1.0 - 1.0005-(y-45)2-0.5(x-40)2
3┌1┌roll → 1.0 - 1.0005-(y-45)2-0.5(x-40)2┌x┌y

```

最后的结果就是：

$$1.0┌1.0 - 1.0005<sup>-(y-45)^2-0.5(x-40)^2</sup>┌1.0 - 1.002<sup>-0.5(y-70)^2-(x-70)^2</sup>$$

这三个数值在 rgb 颜色模式下决定一个颜色，可见这种 <type 4 function> 函数的运算并不直观。

在 <type 4 function> 中不能直接使用颜色名称，必须从颜色名称对应的颜色中提取三个 (rgb 模式的) 颜色参数并保存在某个宏中，然后再把这个宏用于 <type 4 function> 中，这相当于向数值堆栈中引入了 3 个数值。使用下面的命令把颜色参数保存在某个宏中：

```
\pgfshadecolorortorgb{<color name>}{<macro>}
```

这个命令将颜色名称 <color name> 对应颜色的三个 rgb 模式下的颜色参数保存在宏 <macro> 中，这里 <color name> 必须是某个“已经被声明”的颜色名称。保存在宏 <macro> 中的三个颜色参数是 0 到 1 之间的数值，之间由空格分隔。本命令会定义宏 <macro>，如果 <macro> 已经存在，则本命令会重定义它，为它赋值。

```
0.0 1.0 1.0
```

```
\pgfshadecolorortorgb{cyan}{\RGBcyan}
\RGBcyan
```

下面的例子展示了可选项 <color list> 与 <init code> 的配合，二者通过本命令联系起来。



```

\pgfdeclarefunctionalshading[mycol]{sweep}{\pgfpoint{-1cm}{-1cm}}
{\pgfpoint{1cm}{1cm}}{\pgfshadecolorortorgb{mycol}{\myrgb}} % 将 mycol 的颜色值保存在 \
myrgb 中
{
 2 copy
 2 copy abs exch abs add 0.0001 ge { atan } { pop } ifelse
 3 1 roll
 dup mul exch
 dup mul add sqrt
 30 mul
 add
 sin
 1 add 2 div
 dup
 \myrgb % 引入名称 mycol 对应的颜色值
 5 4 roll
 mul
 3 1 roll
 3 index
 mul
 3 1 roll
 4 3 roll
 mul
 3 1 roll
}
\colorlet{mycol}{white} % 为颜色 mycol 赋值
\pgfuses shading{sweep}%
\colorlet{mycol}{red} % 为颜色 mycol 重新赋值
\pgfuses shading{sweep}

```

当使用本命令后，PGF 还会自动定义 3 个宏来分别保存三个颜色值。举例说，假如定义

```
\pgfshadecolorortorgb{orange}{\mycol}
```

PGF 会自动定义 `\mycolred`、`\mycolgreen`、`\mycolblue`，它们的名称是由宏 `\mycol` 的名称与 `red`、`green`、`blue` 结合而成的，它们分别保存颜色 `orange` 的（`rgb` 模式下的）三个颜色参数。

```
1.0 0.5 0.0
1.0
0.5
0.0
```

```
\pgfshadecolor{orange}{\mycol}
\mycol\
\mycolred\
\mycolgreen\
\mycolblue
```

### 109.3 使用颜色渐变

#### `\pgfuseshading{<shading name>}`

这个命令在一个盒子中实现渐变 `<shading name>`，并将盒子插入到文档中。本命令用在 `\pgftext` 之内，可以用于 `{pgfpicture}` 环境中。



```
\begin{tikzpicture}
 \pgfdeclareverticalshading{myshadingD}
 {20pt}{color(0pt)=(red); color(20pt)=(blue)}
 \pgftext[at=\pgfpoint{1cm}{0cm}] {\pgfuseshading{myshadingD}}
 \pgftext[at=\pgfpoint{2cm}{0.5cm}] {\pgfuseshading{myshadingD}}
\end{tikzpicture}
```

#### `\pgfshadepath{<shading name>}{<angle>}`

这个命令必须用在 `{pgfpicture}` 环境中。当创建一个路径后，可以使用本命令来填充当前路径，填充内容就是名称 `<shading name>` 对应的颜色渐变。

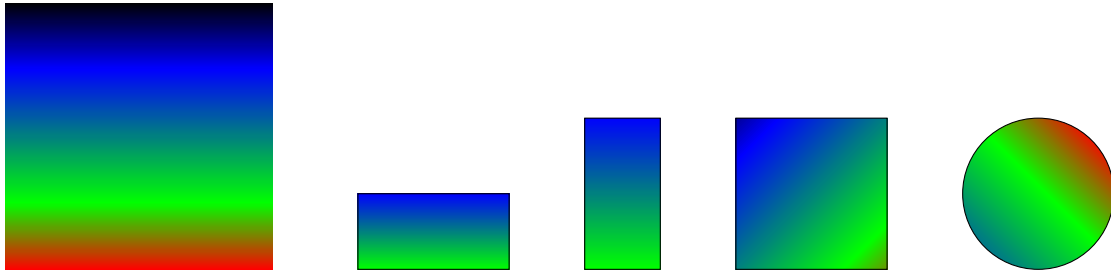
下面介绍一下这个命令的工作过程。用在这个命令中的渐变 `<shading name>` 的盒子宽度和高度都应当是 100bp。在用渐变填充路径时，PGF 会创建一个局部域（a local scope），在这个域中使用路径来剪切渐变盒子得到填充效果。PGF 会计算路径的边界盒子和渐变盒子的尺寸，当然二者的尺寸可能相差很大。为了能让渐变填满路径的边界盒子，需要对渐变盒子做变换。

在这个局部域内，PGF 调整底层的变换矩阵，使得渐变盒子的中心与路径边界盒子的中心重合，这是个平移变换。然后参照盒子中心点对渐变盒子做伸缩变换，PGF 自动确定伸缩变换在水平方向和竖直方向上的伸缩系数，使得渐变盒子的宽度和高度分别是路径边界盒子的 2 倍。注意此时路径边界盒子只能覆盖渐变盒子面积的  $\frac{1}{4}$ 。之后，如果本命令的参数 `<angle>` 非空，则将渐变盒子绕中心点旋转 `<angle>` 角度。之后，如果本命令前面还使用了宏 `\pgfsetadditionalshadettransform`，则针对渐变盒子执行这个宏所保存的变换。再之后，用路径剪切渐变盒子，得到填充效果。

#### `\pgfsetadditionalshadettransform{<transformation>}`

这个命令用在 `\pgfshadepath` 之前，这个命令的参数 `<transformation>` 是某些变换命令，是针对渐变盒子的。



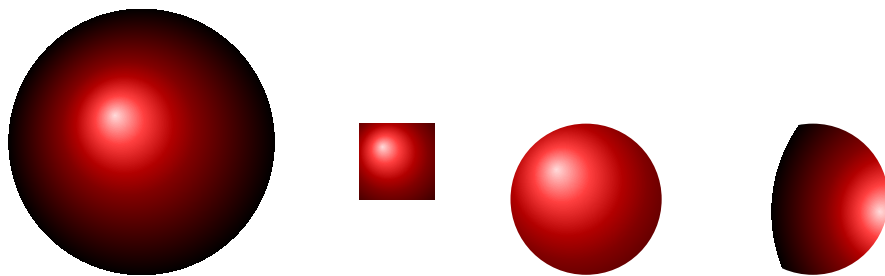


```

\pgfdeclareverticalshading{myshadingE}{100bp}
 {color(0bp)=(red); color(25bp)=(green); color(75bp)=(blue); color(100bp)=(black)}
\pgfuses shading{myshadingE}
\hskip 1cm
\begin{pgfpicture}
 \pgfpathrectangle{\pgfpointorigin}{\pgfpoint{2cm}{1cm}}
 \pgfshadepath{myshadingE}{0}
 \pgfusepath{stroke}
 \pgfpathrectangle{\pgfpoint{3cm}{0cm}}{\pgfpoint{1cm}{2cm}}
 \pgfshadepath{myshadingE}{0}
 \pgfusepath{stroke}
 \pgfpathrectangle{\pgfpoint{5cm}{0cm}}{\pgfpoint{2cm}{2cm}}
 \pgfshadepath{myshadingE}{45}
 \pgfusepath{stroke}
 \pgfpathcircle{\pgfpoint{9cm}{1cm}}{1cm}
 \pgfsetadditionalshadetransform{\pgftransformrotate{90}\pgftransformmyshift{0.8cm}}
 \pgfshadepath{myshadingE}{45}
 \pgfusepath{stroke}
\end{pgfpicture}

```

下面是辐射渐变的例子。



```

\pgfdeclare radialshading{ballshading}{\pgfpoint{-10bp}{10bp}}{
 color(0bp)=(red!15!white); color(9bp)=(red!75!white);
 color(18bp)=(red!70!black); color(25bp)=(red!50!black); color(50bp)=(black)}
\pgfuses shading{ballshading}
\hskip 1cm

```

```

\begin{pgfpicture}
 \pgfpathrectangle{\pgfpointorigin}{\pgfpoint{1cm}{1cm}}
 \pgfshadepath{ballshading}{0}
 \pgfusepath{}
 \pgfpathcircle{\pgfpoint{3cm}{0cm}}{1cm}
 \pgfshadepath{ballshading}{0}
 \pgfusepath{}
 \pgfpathcircle{\pgfpoint{6cm}{0cm}}{1cm}
 \pgfsetadditionalshadetransform{\pgftransformshift{\pgfpoint{0.8cm}{-1cm}}}
 \pgfshadepath{ballshading}{45}
 \pgfusepath{}
\end{pgfpicture}

```



```

\pgfdeclareverticalshading{rainbow}{100bp}{
 color(0bp)=(red); color(25bp)=(red); color(35bp)=(yellow);
 color(45bp)=(green); color(55bp)=(cyan); color(65bp)=(blue);
 color(75bp)=(violet); color(100bp)=(violet)}
\begin{tikzpicture}[shading=rainbow]
 \shade (0,0) rectangle node[white] {\textsc{pride}} (2,1);
 \shade[shading angle=90] (3,0) rectangle +(1,2);
\end{tikzpicture}

```

## 109.4 关于 type 4 函数的补充

以下内容来自《PDF Reference》(version 1.7) 的 §3.

PDF 不是程序语言，一个 PDF 文件也不是一个程序。PDF 提供多种类型的“函数对象”（function objects），它们是参数化的函数，函数的构造涉及数学表达式，样本点等。在 PDF 中函数有多种用途，例如提供光栅信息用于高质量输出（halftone spot functions 和 transfer functions），在某个颜色空间中进行颜色变换，用于创建在颜色渐变。

PDF 中的函数是数值变换。例如加法函数以两个数值为输入，一个数值为输出：

$$f(x_0, x_1) = x_0 + x_1$$

两数的算术平均和几何平均函数是：

$$f(x_0, x_1) = \frac{x_0 + x_1}{2}, \sqrt{x_0 x_1}$$

一般，一个函数可以有  $m$  个输入数值，产生  $n$  个输出值：

$$f(x_0, \dots, x_{m-1}) = y_0, \dots, y_{n-1}$$

PDF 函数的输入和输出都是数值，函数本身只是实现数值变换，没有其它作用。

每个函数都有自己的定义域，有的函数还有值域。如果输入给函数的数值不在其定义域内，函数就会从定义域中选择一个最接近输入值的数来代替它，然后执行计算。如果函数计算出来的输出值不在其值域内，函数就从其值域内选择一个最接近输出值的数来代替它，然后输出替换值。例如，假设函数

$$f(x) = x + 2,$$

它的定义域是  $[-1, 1]$ ，如果调用此函数，且输入值是 6，那么这个输入值就被替换为 1，然后计算  $f(1)$ ，得到输出值 3。

假设函数

$$f(x_0, x_1) = 3x_0 + x_1,$$

的值域是  $[0, 100]$ ，输入  $-6$  和  $4$ （假设在定义域内），计算结果是  $-14$ ，结果不在值域内，于是输出 0。

可用的函数有 4 种类型：

**type 0** 样本函数 (sampled function)，是个表格，用于插值计算。

**type 2** 指数插值函数 (exponential interpolation function)。

**type 3** 缝合函数 (stitching function)。

**type 4** 某些 PostScript 计算函数，用的是 PostScript 语言。

type 4 类函数的表达式中只涉及整数、实数、布尔值，没有字符串、数组，也没有其它指令、变量、名称。能够用于构造这类函数的算子如下表所示：

type 4 类函数中的算子

算子类型	算子				
计算算子	abs	cvi	floor	mod	sin
	add	cvr	idiv	mul	sqrt
	atan	div	ln	neg	sub
	ceiling	exp	log	round	truncate
	cos				
	真值算子	and	false	le	not
bitshift		ge	lt	or	xor
eq		gt	ne		
条件算子	if	ifelse			
堆栈算子	copy	exch	pop		
	dup	index	roll		

下面解释一下这些算子，参考《PostScript LANGUAGE REFERENCE》(third edition) 的第 8 章。

设想有一个数值堆栈 (stack)，将某些数值按次序存放到这个堆栈中，位置靠前的数值处于“下部”，位置靠后的数值处于“上部”，第一个数值处于“底部”，最后一个数值处于“顶端” (topmost)。将一个算子放在数值堆栈之后，算子针对堆栈中的某些数值做运算，也就是说，算子都是“后置”算子 (数学中的算子多数是“前置算子”)。以加法算子 `add` 为例，这个算子是二元算子，即它需要两个输入值 (运算对象 operand)，假设写出一串用空格分隔的数值，`1 2 3 4`，然后写上 `add`：

```
1 2 3 4 add
```

执行这一行代码，算子 `add` 将前面的 3 4 求和，得到的是数值列表：

```
1 2 7
```

下面用例子说明这些算子的作用。

**abs** 绝对值运算，输出值的类型与输入值相同；若输入是最小的整数，则输出是实数。

```
0 2 -1.0 abs 输出 0 2 1.0
```

**add** 求两数和，若两个输入值都是整数则输出整数，否则输出实数。

```
1 2 3.0 add 输出 1 5.0
```

**and** 逻辑与运算，输入可以是布尔值：

```
true false and 输出 false
```

输入也可以是整数，此时是“按位运算”：

```
2 99 1 and 输出 2 1，这里只对 99 和 1 做运算。
```

**atan** 变异的反正切函数，输入值是两个数，输出值是 0 到 360 之间的实数，代表角度。

```
13 -1 1 atan 输出 13 135.0，输出值是从向量 (1,0) 沿着逆时针方向转到向量 (-1,1) 所转过
的角度。
```

注意：

```
1 0 atan 输出 90.0
```

```
-100 0 atan 输出 270.0
```

**bitshift** 移位算子，以两个整数为输入值，第一个输入整数最好是正整数，本算子输出一个整数。

```
0 7 3 bitshift 输出 0 56，将 7 的二进制表示向左移动 3 位
```

```
142 -3 bitshift 输出 17，将 142 的二进制表示向右移动 3 位
```

**ceiling** 向上取整，例如

```
11 3.2 ceiling 输出 11 4.0
```

```
12 -4.8 ceiling 输出 12 -4.0
```

**copy** 复制算子，本算子从数值堆栈中选取靠近“顶部”的某些数值，复制这些数值并将它们添加到原来的堆栈中，即用复制的方法增加堆栈中的数值数量，增加堆栈长度（高度）。本算子选取数值的“标准”由堆栈的“顶端”数值决定，如下面的例子所示。

```
(a) (b) (c) 2 copy 输出 (a) (b) (c) (b) (c)
```

```
(a) (b) (c) 0 copy 输出 (a) (b) (c)
```

**cos** 角度的余弦值，本算子需要一个输入，输出 -1 到 1 之间的实数。

```
-1 60 cos 输出 -1 0.5
```

**cvi** 向零取整，是 `convert to integer` 的缩写。

**cvr** 将输入值转化为实数类型的数，是 `convert to real` 的缩写。

**div** 两数除法，需要两个输入值，计算前数除以后数的商并输出，输出值总是实数，其符号与第一个输入值相同。

```
3 2 div 输出 1.5
4 2 div 输出 2.0
```

**dup** 复制顶端值，本算子的输入只是堆栈顶端的数值，复制顶端值并添加到堆栈中。

```
1 2 dup 输出 1 2 2
```

**eq** 判断两数是否相等，需要两个输入值。

```
0 2 2 eq 输出 0 true
```

**exch** 换位，本算子需要两个输入值，交换它们在堆栈中的位置。

```
1 2 3 4 exch 输出 1 2 4 3
```

**exp** 幂运算，

```
1 2 3 exp 输出 1 8，这里计算 2^3
```

**false** 布尔值 false，本算子没有输入，只有输出值 false.

```
1 2 false 输出 1 2 false
```

**floor** 向下取整.

**ge** 判断是否不小于，需要两个输入值，判断前数是否不小于后数。

```
1 2 3 ge 输出 1 false
```

**gt** 判断是否大于，需要两个输入值，判断前数是否大于后数。

```
1 2 3 gt 输出 1 false
```

**idiv** 取整除法，需要两个整数作为输入，计算前数除以后数的商，并输出商的整数部分，输出值的符号与第一个输入值相同。

```
3 2 idiv 输出 1
4 2 idiv 输出 2
-5 2 idiv 输出 -2
```

**if** 条件算子，它只需要两个输入，本身没有输出。

```
<bool> <pro> if 如果 <bool> 为 true 则执行 <pro>，否则继续后面的处理。
```

**ifelse** 条件算子，它只需要三个输入，本身没有输出。

```
<bool> <pro1> <pro2> ifelse 如果 <bool> 为 true 则执行 <pro1>，否则执行 <pro2>。
```

**index** 倒序索引，从中堆栈中选出某个数值，复制它并添加到堆栈中，使之处于“顶端”位置。本算子选择数值的标准由原堆栈的“顶端”数值决定。

```
<any> ... <any0> n index 输出 <any> ... <any0> <any>
(a) (b) (c) (d) 0 index 输出 (a) (b) (c) (d) (d)
(a) (b) (c) (d) 3 index 输出 (a) (b) (c) (d) (a)
```

**le** 判断是否不大于，需要两个输入值，判断前数是否不大于后数。

```
1 2 le 输出 true
```

**ln** 计算自然对数值，需要一个输入值。

```
10 ln 输出 2.30259
```

```
100 ln 输出 4.60517
```

**log** 计算常用对数值（以 10 为底），

```
10 log 输出 1.0
```

```
100 log 输出 2.0
```

**lt** 判断是否小于，需要两个输入值，判断前数是否小于后数。

```
1 2 lt 输出 true
```

**mod** 余数运算，需要两个整数作为输入值，计算前数除以后数的余数并输出之，余数的符号与第一个输入值相同。

```
5 3 mod 输出 2
```

```
-5 3 mod 输出 -2
```

**mul** 计算两数乘积，需要两个输入值。

**ne** 判断是否不等于，需要两个输入值。

**neg** 取相反数，需要一个输入值。

**not** 逻辑非，需要一个输入值。

**or** 逻辑或，需要两个输入值，输入值可以是布尔值，也可以是整数（此时是按位运算）。

**pop** 删除顶端值，这个算子只有一个输入，没有输出。它删除堆栈中的顶端数值。

```
1 2 3 pop 输出 1 2
```

```
1 2 3 pop pop 输出 1
```

**roll** 轮换，它的前面应当是两个正整数，本算子把这两个正整数与堆栈中的其它数值区别开来，参照这两个正整数，将堆栈中的其余数值做轮换。

$$any_{n-1} \cdots any_0 \ n \ j \ \text{roll} \ \text{输出} \ any_{(j-1) \bmod n} \cdots any_0 \ any_{n-1} \cdots any_j \bmod n$$

```
(a) (b) (c) 3 1 roll 输出 (c) (a) (b)
```

**round** 四舍五入，需要一个输入值。

**sin** 计算角度的正弦值，需要一个输入值，输出为实数。

**sqrt** 计算非负数的平方根。

**sub** 计算两数的差，需要两个输入值，计算前数减去后数的差。

**true** 逻辑值 true.

**truncate** 去掉输入值的小数部分，保留整数部分并输出之，输出数值类型与输入值相同。

**xor** 异或运算，需要两个输入值，输入值可以是布尔值，也可以是整数（此时是按位运算）。

## 110 透明度

先阅读 §23.

### 110.1 指定不透明度

可以为画出的线条（stroke）、填充的颜色（fill）指定不透明度。

`\pgfsetstrokeopacity{<value>}`

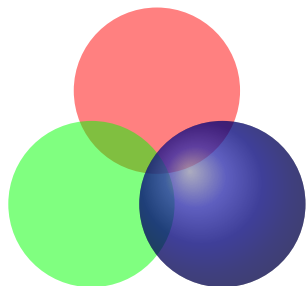
本命令为画线操作指定不透明度，<value> 是 0 到 1 之间的数值，1 代表“完全不透明”，0 代表“完全透明”。



```
\begin{pgfpicture}
 \pgfsetlinewidth{5mm}
 \color{red}
 \pgfpathcircle{\pgfpoint{0cm}{0cm}}{8mm} \pgfusepath{stroke}
 \color{black}
 \pgfsetstrokeopacity{0.5}
 \pgfpathcircle{\pgfpoint{1cm}{0cm}}{8mm} \pgfusepath{stroke}
\end{pgfpicture}
```

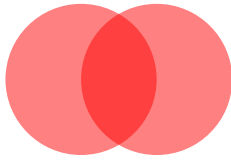
`\pgfsetfillopacity{<value>}`

本命令为填充颜色操作指定不透明度，<value> 是 0 到 1 之间的数值。这个命令指定的不透明度不仅对填充色有效，对路径上的文字、插入的外部图形、颜色渐变都有效。



```
\begin{tikzpicture}
 \pgfsetfillopacity{0.5}
 \fill[red] (90:1cm) circle (11mm);
 \fill[green] (210:1cm) circle (11mm);
 \fill[shading=ball] (-30:1cm) circle (11mm);
\end{tikzpicture}
```

注意，在默认下，同一区域内的不透明度是叠加。也就是说，如果两个绘图命令都带有不透明度设置，并且两个命令画的图有重合部分，那么重合部分的不透明度是两个命令的不透明度的叠加。如果不希望叠加不透明度，可以通过“透明度组”来设置，见后文。

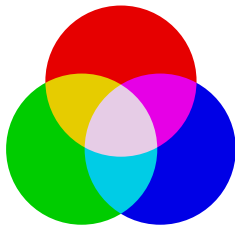


```
\begin{tikzpicture}
 \pgfsetfillopacity{0.5}
 \fill[red] (0,0) circle (1);
 \fill[red] (1,0) circle (1);
\end{tikzpicture}
```

## 110.2 指定混色模式

`\pgfsetblendmode{<mode>}`

混色模式见 §23.3. 混色模式是 PDF 的高级特性，未必总能得到正确显示。



```
\tikz [transparency group] {
 \pgfsetblendmode{screen}
 \fill[red!90!black] (90:.6) circle (1);
 \fill[green!80!black] (210:.6) circle (1);
 \fill[blue!90!black] (330:.6) circle (1);
}
```

## 110.3 “褪色” (fading) 效果

Fading 效果涉及“亮度” (luminosity) 这个概念，这个概念可以简单理解为：一个像素点上，一秒钟内发出的可见光能量有多少。假设在一个白色背景上有个红色点，红色点的亮度越高，该点的红色就越鲜艳、越刺眼，但不会变成“暗红色”，因为红色与暗红色不是同一种颜色。红色点的亮度越低，该点的红色可见光就越少，白色背景就会显露出来，可见亮度与透明度有一定关系。如果一个点处没有出现任何光线，那么在理论上这个点没有颜色，而在视觉上这个点是黑色的。因此没有颜色的点和黑色像素点的亮度都规定为 0，白色像素点的亮度是 1。

假设在一个  $\text{T}_\text{E}_\text{X}$  盒子里写下单词 TikZ，并且把文字颜色设为白色，此时盒子里的点只有两种：一种是构成文字的白色点，亮度为 1，另一种是没有颜色的点，亮度为 0。然后把这盒子想象成一个特别的“印章”，把另外某个盒子看作是“纸”，把“印章”盖到“纸”上，同时把这些亮度值转换成不透明度值印在“纸”上，这样“纸”上的点就有了自己的不透明度值。如果这张“纸”原本是红色的，被盖章后，被白色文字 TikZ 覆盖的部分的不透明度是 1，其余部分的不透明度都是 0，于是原本的红色“纸”就变了，我们得到红色的文字 TikZ。也就是说，“印章”是一种“映射”，或者说赋值技术。

`\pgfdeclarefading{<name>}{<contents>}`

这个命令声明一个名称为 <name> 的 fading 样式，以供之后引用。本命令的声明全局有效。

<contents> 是  $\text{T}_\text{E}_\text{X}$  内容，会被放入一个  $\text{T}_\text{E}_\text{X}$  盒子里。<contents> 可以是文字，表格环境，数学公式，`{\pgfpicture}` 环境等，用于制作“印章”。注意 <contents> 中的颜色最好只涉及黑、白、灰 3 种颜色，灰色用 `black!20` 之类的颜色表达式表示。如果 <contents> 中涉及彩色可能不太容易控制。

为了方便，称盛放“印章”的盒子为“fading 盒子”。





```
\pgfdeclarefading{fading1}{\color{white}Ti\emph{k}Z}
\begin{tikzpicture}
 \fill [black!20] (0,0) rectangle (2,2);
 \fill [black!30] (0,0) arc (180:0:1);
 \pgfsetfading{fading1}{\pgftransformshift{\pgfpoint{1cm}{1cm}}}
 \fill [red] (0,0) rectangle (2,2);
\end{tikzpicture}
```

上面例子中，白色印章 fading1 印在红色纸上，得到红色文字 TikZ。

### pgftransparent

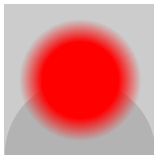
这个预定义颜色就是黑色，在 fading 中对应“完全透明”。pgftransparent!20 表示一个灰色，pgftransparent!0 表示白色。

下面用一个颜色渐变图形制作一个“fading 盒子”。



```
\pgfdeclarefading{fading2}{
 \tikz \shade[left color=pgftransparent!0,
 right color=pgftransparent!100] (0,0) rectangle (2,2);}
\begin{tikzpicture}
 \fill [black!20] (0,0) rectangle (2,2);
 \fill [black!30] (0,0) arc (180:0:1);
 \pgfsetfading{fading2}{\pgftransformshift{\pgfpoint{1cm}{1cm}}}
 \fill [red] (0,0) rectangle (2,2);
\end{tikzpicture}
```

下面先声明一个辐射渐变，然后将这个辐射渐变用于 fading 的声明中。



```
\pgfdeclareradialshading{myshading}{\pgfpointorigin}
{
 color(0mm)=(pgftransparent!0);
 color(5mm)=(pgftransparent!0);
 color(8mm)=(pgftransparent!100);
 color(15mm)=(pgftransparent!100)
}
\pgfdeclarefading{fading3}{\pgfuses shading{myshading}}
\begin{tikzpicture}
 \fill [black!20] (0,0) rectangle (2,2);
 \fill [black!30] (0,0) arc (180:0:1);
 \pgfsetfading{fading3}{\pgftransformshift{\pgfpoint{1cm}{1cm}}}
 \fill [red] (0,0) rectangle (2,2);
\end{tikzpicture}
```

**`\pgfsetfading{<name>}{<transformations>}`**

声明一个 fading 后，就可以用本命令使用来使用它。在前文已经有本命令的例子。

本命令对之后的路径起作用，为之后的路径设置“状态参数”，本命令的有效范围一直延续到绘图环境（不是 TeX 分组）结束，或者遇到下一个 `\pgfsetfading`。

在声明 fading 时，“fading 盒子”是个 TeX 盒子。在使用 fading 盒子时，这个盒子的中心会被放在绘图环境坐标系的原点上，并且程序不会自动调整这个盒子的尺寸。

命令参数 `<transformations>` 是 PGF 的变换命令，用于对 fading 盒子做某些变换，例如平移、旋转、放缩等。对 fading 盒子做完变换后，本命令之后的路径会剪切 fading 盒子。此时可能有多种情况，例如，路径内部包含 fading 盒子但不能被 fading 盒子填满，或者，路径内部不包含 fading 盒子但与之有交集，或者，路径内部与 fading 盒子无交集。无论何种情况，对于路径内部的点，只要未被 fading 盒子“盖公章”，其不透明度就是 0，因而完全透明。

**`\pgfsetfadingforcurrentpath{<name>}{<transformations>}`**

这个命令类似 `\pgfsetfading`，只是作用稍复杂。使用本命令后会有以下效果：

1. 如果当前路径是空，则与 `\pgfsetfading` 相同。
2. 如果当前路径非空，则类似 `\pgfshadepath`，本命令变换 fading 盒子，使其中心与当前路径的边界盒子的中心重合，并变换 fading 盒子的宽度和高度，使之分别成为当前路径边界盒子的宽度和高度的 2 倍。
3. 执行参数 `<transformations>`，这是某些 PGF 变换命令，进一步变换 fading 盒子。

**`\pgfsetfadingforcurrentpathstroked{<name>i}{<transformations>}`**

本命令类似 `\pgfsetfadingforcurrentpath`，只是会在水平方向和垂直方向上分别扩大当前路径边界盒子的尺寸，扩大的增量是路径线宽。在计算当前路径的边界盒子时并不考虑路径线宽（见 §101.4），如果路径线宽值较大并且需要画出路径来显示路径的 fading 效果，那么就可能需要扩大当前路径边界盒子的尺寸来容纳路径线宽。

比较下面两个图形，其中用了 `fadings` 程序库提供的 `east` 褪色样式：



```
\begin{tikzpicture}
 \pgfsetlinewidth{4mm}
 \pgfpathmoveto{\pgfpointorigin}
 \pgfpathlineto{\pgfpoint{2cm}{0cm}}
 \pgfsetfadingforcurrentpathstroked{east}{}
 \pgfusepath{stroke}
\end{tikzpicture}
```

```

\begin{tikzpicture}
 \pgfsetlinewidth{4mm}
 \pgfpathmoveto{\pgfpointorigin}
 \pgfpathlineto{\pgfpoint{2cm}{0cm}}
 \pgfsetfadingforcurrentpath{east}{}
 \pgfusepath{stroke}
\end{tikzpicture}

```

## 110.4 透明度组

下面的环境声明一个透明度组。

```

\begin{pgftransparencygroup}[<options>]
 <environment contents>
\end{pgftransparencygroup}

```

这个环境只能用在 `{pgfpicture}` 环境中。

参考 §23.5. 在默认下，本环境外的透明度设置不能影响本环境内的内容；本环境的透明度设置也不会影响到环境外的内容。

环境选项 `<options>` 中可以使用的选项如下，其作用参考 §23.5:

`knockout=<true or false>` (默认值 `true`, 初始值 `false`)

注意这个选项的默认值是 `true`，初始值是 `false`，有的渲染器不支持这个选项的作用。

`isolated=<true or false>` (默认值 `true`, 初始值 `true`)

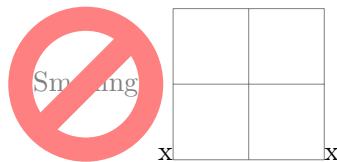
```

\pgftransparencygroup
 <environment contents>
\endpgftransparencygroup

```

这是 Plain TEX 中的 `{pgftransparencygroup}` 环境。

透明度组是被放入 PDF 的 XForm 中的，因此 PGF 依赖驱动获得透明度组的边界盒子的尺寸。



```

x\begin{tikzpicture}
 \draw [help lines] (0,0) grid (2,2);
 \node [left,overlay] at (0,1) {
 \begin{tikzpicture}
 \pgfsetfillopacity{0.5}
 \pgftransparencygroup

```

```

 \node at (2,0) [forbidden sign,line width=2ex,
 draw=red,fill=white]{Smoking};
 \endpgftransparencygroup
\end{tikzpicture}
};
\end{tikzpicture}x

```

上面的 node 选项 overlay 见 §17.13.1.

## 111 临时寄存器

如果你有全面的 T<sub>E</sub>X 编程知识，并且熟悉 PGF 的基本层，那你可以为 PGF 编写新的程序库，此时你可能用到 T<sub>E</sub>X 的临时寄存器（temporary registers），这里介绍分配给 PGF 的几个临时寄存器。

`\pgf@x`

`\pgf@y`

这是 PGF 内部的两个尺寸寄存器，用于处理点的坐标，它们用作点的两个坐标分量，见 §96.7.

这两个寄存器是被“全局地”设置的，PGF 的其它寄存器则是“临时地”。这两个寄存器的值是频繁变化的，所以最好不要用它们设计你的计算过程，除非你知道它们的值是怎样变化的。在设计计算过程时，推荐使用临时寄存器。

`\pgf@xa`

`\pgf@xb`

`\pgf@xc`

`\pgf@ya`

`\pgf@yb`

`\pgf@yc`

这是 6 个临时尺寸寄存器，你可以在一个 T<sub>E</sub>X 分组内随意修改它们的值。

注意：PGF 使用这些寄存器来执行路径操作，为了提高效率，路径命令并不监视这些寄存器，它们的值被重定义时，PGF 并不中断处理。在文件《pgfcorepoints.code》中对 `\pgfpoint`，`\pgfpointadd` 和 `\pgfpointlineattime` 的定义分别是：

```

\def\pgfpoint#1#2{%
 \pgfmathsetlength\pgf@x{#1}%
 \pgfmathsetlength\pgf@y{#2}\ignorespaces}

\def\pgfpointadd#1#2{%
 \pgf@process{#1}%
 \pgf@xa=\pgf@x%
 \pgf@ya=\pgf@y%
 \pgf@process{#2}%

```

```

\advance\pgf@x by\pgf@xa%
\advance\pgf@y by\pgf@ya}

\def\pgfpointlineattime#1#2#3{%
 \pgf@process{#3}%
 \pgf@xa\pgf@x%
 \pgf@ya\pgf@y%
 \pgf@xc\pgf@x%
 \pgf@yc\pgf@y%
 \pgf@process{#2}%
 \pgf@xb\pgf@x%
 \pgf@yb\pgf@y%
 \pgfmathsetmacro\pgf@temp{#1}%
 \advance\pgf@xa by-\pgf@x%
 \advance\pgf@ya by-\pgf@y%
 \advance\pgf@x by\pgf@temp\pgf@xa%
 \advance\pgf@y by\pgf@temp\pgf@ya%
}

```

在 `\pgfpointlineattime` 的定义中用到了 `\pgf@xb` 与 `\pgf@yb`，分析下面的代码：

<pre> \pgf@x=13.0pt \pgf@y=14.0pt \pgf@xa=2.0pt \pgf@ya=2.0pt \pgf@xb=11.0pt \pgf@yb=12.0pt </pre>	<pre> \makeatletter \pgfmathsetlength{\pgf@xa}{1pt} \pgfmathsetlength{\pgf@xb}{2pt} \pgfmathsetlength{\pgf@ya}{3pt} \pgfmathsetlength{\pgf@yb}{4pt} \pgfpointlineattime{0.5}{\pgfpoint{11pt}{12pt}}   {\pgfpoint{13pt}{14pt}} \pgfpointadd{\pgfpoint{\pgf@xa}{\pgf@ya}}   {\pgfpoint{\pgf@xb}{\pgf@yb}} \string\pgf@x=\the\pgf@x\ \string\pgf@y=\the\pgf@y\ \ \string\pgf@xa=\the\pgf@xa\ \string\pgf@ya=\the\pgf@ya\ \ \string\pgf@xb=\the\pgf@xb\ \string\pgf@yb=\the\pgf@yb \makeatother </pre>
----------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

其中一开始设置了 `\pgf@xb` 与 `\pgf@yb` 的值，但是随后这两个值又被命令 `\pgfpointlineattime` 改造了，所以最后命令 `\pgfpointadd` 计算的并不是

$$(1pt, 2pt) + (3pt, 4pt) = (4pt, 6pt)$$

而是

$$(2pt, 2pt) + (11pt, 12pt) = (13pt, 14pt)$$

显然这个算式并不直观。可见在使用内部的临时寄存器做计算时应当谨慎一些，因为内部命令会使用这些寄存器，可能改变它们的值。

`\pgfutil@tempdima`

`\pgfutil@tempdimb`

这是两个临时尺寸寄存器。

`\c@pgf@counta`

`\c@pgf@countb`

`\c@pgf@countc`

`\c@pgf@countd`

这是 4 个计数器，用于执行关于整数的计算，用在  $\TeX$  分组内。

`\w@pgf@writea`

`\r@pgf@reada`

`\pgfutil@tempboxa`

在  $\TeX$  分组内使用的临时盒子。

## 112 快速命令

快速命令，即 “quick” commands，是普通命令的 “q” 版，运行起来比普通命令要快，但也牺牲了某些功能，因此最好在适当的情况下使用，例如需要执行的命令数量太多时，另外在 §100.4 中介绍 `drawing code` 时也提到过，其中要使用快速箭头。

### 112.1 快速坐标命令

`\pgfqpoint{<x>}{<y>}`

类似 `\pgfpoint`，不过 `<x>` 和 `<y>` 都应该是简单的尺寸，例如 `1pt` 或 `1cm`，但不可以是 `2ex` 或 `1cm+1pt`。

`\pgfqpointxy{<sx>}{<sy>}`

类似 `\pgfpointxy`，`<sx>` 和 `<sy>` 应当是简单的数值，如 `1.234`，不可以是表达式或者长度单位。

`\pgfqpointxyz{<sx>}{<sy>}{<sz>}`

类似 `\pgfpointxy`，只是用于三维坐标点，`<sx>`，`<sy>`，`<sz>` 都应当是简单的数值。

`\pgfpointscale{<factor>}{<coordinate>}`

类似 `\pgfpointscale`，`<factor>` 应当是简单的数值。

### 112.2 快速创建路径的命令

快速创建路径的命令有以下特点：

- 不跟踪边界盒子。

- 不能使用圆角。
- 不接受坐标变换。

快速创建路径的命令都以 `\pgfpathq` 开头。

`\pgfpathqmoveto{<x dimension>}{<y dimension>}`

类似 `\pgfpathmoveto`，可以开启一个路径或者以 `move-to` 方式延伸路径。

`\pgfpathqlineto{<x dimension>}{<y dimension>}`

类似 `\pgfpathlineto`。

`\pgfpathqcurveto{<s1x>}{<s1y>}{<s2x>}{<s2y>}{<tx>}{<ty>}`

类似 `\pgfpathcurveto`。

`\pgfpathqcircle{<radius>}`

类似 `\pgfpathcircle`，只不过构建的圆以原点为圆心，以 `<radius>` 为半径。

### 112.3 快速使用路径的命令

快速使用路径的命令都以 `\pgfusepathq` 开头，它们有以下特点：

- 不能用来添加箭头。
- 不会调整路径。
- 不能截去路径的末端，对比命令 `\pgfsetshortenend`（见 §99.3）。
- 不能使用圆角。

`\pgfusepathqstroke`

只画出路径，没有其它动作，例如不添加箭头，不使用圆角。

`\pgfusepathqfill`

只填充路径，没有其它动作。

`\pgfusepathqfillstroke`

只填充、画出路径，没有其它动作。

`\pgfusepathqclip`

用当前路径剪切本命令之后的各个路径，但是并不处理当前路径。

### 112.4 快速文字盒子命令

`\pgfqbox{<box number>}`

`<box number>` 是某个  $\text{T}_{\text{E}}\text{X}$  盒子的编号，这个盒子内可以包含任何允许的内容，如文字、表格环境、数学公式、`{\pgfpicture}` 环境等。本命令用在 `{\pgfpicture}` 环境内，将盒子 `<box number>` 插入到环境坐标系的原点处，盒子中心与坐标系原点重合。

`\pgfqboxsynced{<box number>}`

类似 `\pgfqbox`，只不过本命令在插入盒子 `<box number>` 之前，会先把当前的坐标变换矩阵转换为画布变换矩阵，等效于下面两个命令：

```
\pgflowlevelsyncm
\pgfqbox{<box number>}
```