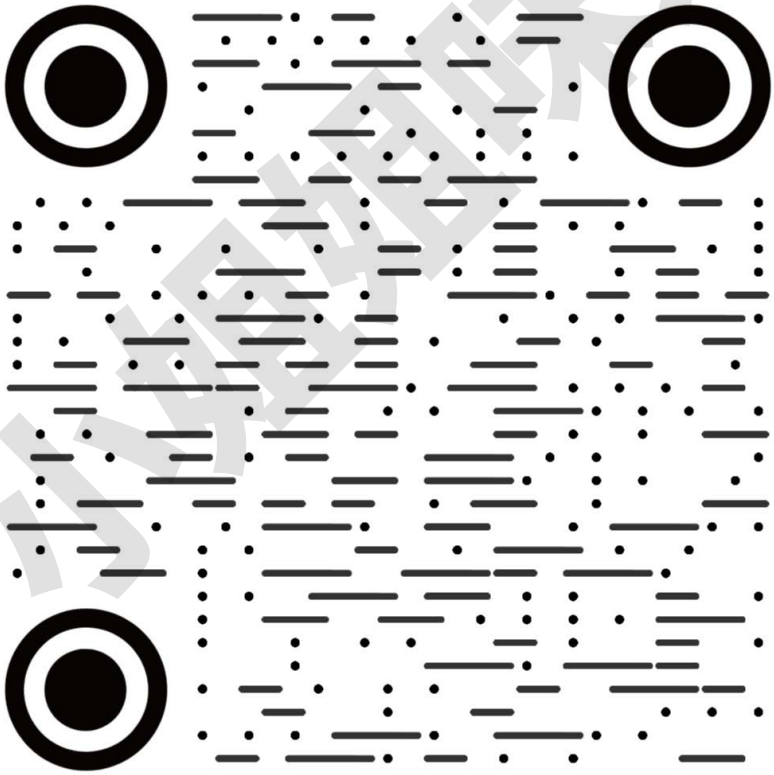
最有用Linux系列

2019年，微信公众号《小姐姐味道》输出了《最有用Linux系列》，获得极大反响和认可。其中，《Linux上，最常用的一批命令解析（10年精选）》这篇文章，在CSDN发布首日，即收获1000+赞。

这些文章，是作者十几年Linux使用经验的精华，这些内容经过多次培训、整理，最终更加易懂常用。

为了方便大家阅读，甚至离线或者大屏阅读，特制作此PDF。如果对你有所帮助，欢迎关注微信公众号《小姐姐味道》，ID：xjjdog。

主要关注Java领域的**基础架构**，和**Linux** 的深度使用。给你不一样的味道。本系列暂包含10篇文章。更多请关注。



Linux上，最常用的一批命令解析（10年精选）

> 原创：小姐姐味道（微信公众号ID：xjjdog），欢迎分享，转载请保留出处。

Linux这么多命令，通常会让初学者望而生畏。下面是我结合日常工作，以及在公司的内部培训中，针对对Linux不是很熟悉的同学，精选的一批必须要搞懂的命令集合。

任何一个命令其实都是可以深入的，比如 `tail -f` 和 `tail -F` 的区别。我们不去关心，只使用最常见的示例来说明。本文不会教你具体的用法，那是抢 `man` 命令的饭碗。这只是个引导篇，力求简洁。

学习方式：多敲多打，用条件反射替代大脑记忆--如果你将来或者现在要用它来吃饭的话。其中，也有一些难啃的骨头，关注小姐姐味道微信公众号，我们一起用锋利的牙齿，来把它嚼碎。

内容：
✔ 目录操作

✔ 文本处理

✔ 压缩

✔ 日常运维

✔ 系统状态概览

✔ 工作常用

目录操作

工作中，最常打交道的就是对目录和文件的操作。linux提供了相应的命令去操作他，并将这些命令抽象、缩写。

基本操作

可能是这些命令太常用了，多打一个字符都是罪过。所以它们都很短，不用阿拉伯数字，一个剪刀手就能数过来。

看命令。

mkdir 创建目录 make dir
cp 拷贝文件 copy
mv 移动文件 move
rm 删除文件 remove

例子：

```
# 创建目录和父目录a,b,c,d
mkdir -p a/b/c/d

# 拷贝文件夹a到/tmp目录
cp -rvf a/ /tmp/

# 移动文件a到/tmp目录，并重命名为b
mv -vf a /tmp/b

# 删除机器上的所有文件
rm -rvf /
```

漫游

linux上是黑漆漆的命令行，依然要面临人生三问：我是谁？我在哪？我要去何方？

`ls` 命令能够看到当前目录的所有内容。`ls -l` 能够看到更多信息，判断你是谁。

`pwd` 命令能够看到当前终端所在的目录。告诉你你在哪。

`cd` 假如你去错了地方，cd命令能够切换到对的目录。

`find` find命令通过筛选一些条件，能够找到已经被遗忘的文件。

至于要去何方，可能就是主宰者的意志了。

文本处理

这是是非常非常加分的技能。get到之后，也能节省更多时间来研究面向对象。小姐姐味道已经输出了“最常用的vim、sed、awk技巧系列”。下面附上链接。

《Linux生产环境上，最常用的一套“vim“技巧》

《Linux生产环境上，最常用的一套“Sed“技巧》

《Linux生产环境上，最常用的一套“AWK“技巧》

查看文件

cat

最常用的就是`cat`命令了，注意，如果文件很大的话，cat命令的输出结果会疯狂在终端上输出，可以多次按`ctrl+c`终止。

```
# 查看文件大小
du -h file

# 查看文件内容
cat file
```

less

既然cat有这个问题，针对比较大的文件，我们就可以使用`less`命令打开某个文件。
类似vim，less可以在输入`/`后进入查找模式，然后按`n`(N)向下(上)查找。

有许多操作，都和vim类似，你可以类比看下。

tail

大多数做服务端开发的同学，都了解这么命令。比如，查看nginx的滚动日志。

```
tail -f access.log
```

tail命令可以静态的查看某个文件的最后n行，与之对应的，head命令查看文件头n行。但head没有滚动功能，就像尾巴是往外长的，不会反着往里长。

```
tail -n100 access.log
head -n100 access.log
```

统计

sort和uniq经常配对使用。
sort可以使用`-t`指定分隔符，使用`-k`指定要排序的列。

下面这个命令输出nginx日志的ip和每个ip的pv，pv最高的前10

```
# 2019-06-26T10:01:57+08:00|nginx001.server.ops.pro.dc|100.116.222.80|10.31.150.232:41021|0.014|0.011|0.000|200|200|273|-|/visit|sign=91CD1988CE8B313B8A0454A4BBE930DF|-|-|http|POST|112.4.238.213

awk -F"|" '{print $3}' access.log | sort | uniq -c | sort -nk1 -r | head -n10
```

其他

grep

grep用来对内容进行过滤，带上`--color`参数，可以在支持的终端可以打印彩色，参数`n`则输出具体的行数，用来快速定位。

比如：查看nginx日志中的POST请求。

```
grep -rn --color POST access.log
```

推荐每次都使用这样的参数。

如果我想要看某个异常前后相关的内容，就可以使用ABC参数。它们是几个单词的缩写，经常被使用。
A after 内容后n行
B before 内容前n行
C count? 内容前后n行

就像是这样：

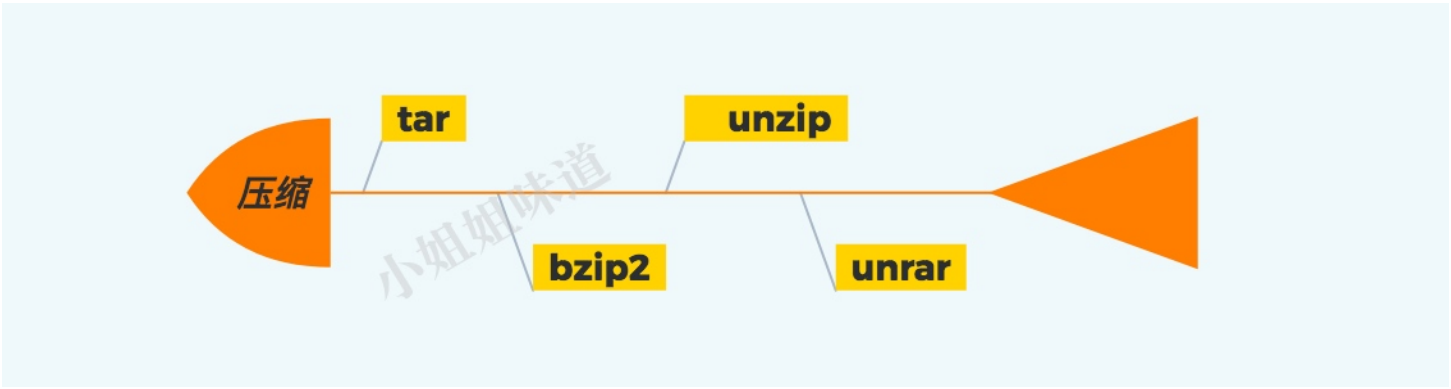
```
grep -rn --color Exception -A10 -B2   error.log
```

diff

diff命令用来比较两个文件是否的差异。当然，在ide中都提供了这个功能，diff只是命令行下的原始折衷。对了，diff和patch还是一些平台源码的打补丁方式，你要是不用，就pass吧。

压缩

为了减小传输文件的大小，一般都开启压缩。linux下常见的压缩文件有tar、bzip2、zip、rar等，7z这种用的相对较少。



.tar 使用tar命令压缩或解压

.bz2 使用bzip2命令操作

.gz 使用gzip命令操作

.zip 使用unzip命令解压

.rar 使用unrar命令解压

最常用的就是`.tar.gz`文件格式了。其实是经过了tar打包后，再使用gzip压缩。

创建压缩文件

```
tar cvfz  archive.tar.gz dir/
```

解压

```
tar xvfz. archive.tar.gz
```

快去弄清楚它们的关系吧。

日常运维

开机是按一下启动按钮，关机总不至于是长按启动按钮吧。对了，是shutdown命令，不过一般也没权限-.-!。passwd命令可以用来修改密码，这个权限还是可以有的。

mount

mount命令可以挂在一些外接设备，比如u盘，比如iso，比如刚申请的ssd。可以放心的看小电影了。

```
mount /dev/sdb1 /xiaodianying
```

chown

`chown` 用来改变文件的所属用户和所属组。

`chmod` 用来改变文件的访问权限。

这两个命令，都和linux的文件权限777有关。

示例：

```
# 毁灭性的命令
chmod 000 -R /

# 修改a目录的用户和组为 xjj
chown -R xjj:xjj a
```

```
# 给a.sh文件增加执行权限（这个太常用了）
chmod a+x a.sh
```

yum

假定你用的是centos，则包管理工具就是yum。如果你的系统没有wget命令，就可以使用如下命令进行安装。

```
yum install wget -y
```

systemctl

当然，centos管理后台服务也有一些套路。`service`命令就是。`systemctl`兼容了`service`命令，我们看一下怎么重启mysql服务。 推荐用下面这个。

```
service mysql restart
systemctl restart  mysqld
```

对于普通的进程，就要使用kill命令进行更加详细的控制了。kill命令有很多信号，如果你在用`kill -9`，你一定想要了解`kill -15`以及`kill -3`的区别和用途。

su

su用来切换用户。比如你现在是root，想要用xjj用户做一些勾当，就可以使用su切换。

```
su xjj
su - xjj
```

`-`可以让你干净纯洁的降临另一个账号，不出意外，推荐。

系统状态概览

登陆一台linux机器，有些命令能够帮助你快速找到问题。这些命令涵盖内存、cpu、网络、io、磁盘等。如需要更多了解，参考五件套。比较高阶一些。

Linux之《荒岛余生》（一）准备篇
Linux之《荒岛余生》（二）CPU篇
Linux之《荒岛余生》（三）内存篇
Linux之《荒岛余生》（四）I/O篇

uname

uname命令可以输出当前的内核信息，让你了解到用的是什么机器。

```
uname -a
```

ps

ps命令能够看到进程/线程状态。和top有些内容重叠，常用。

```
# 找到java进程
ps -ef|grep java
```

top

系统状态一览，主要查看。cpu load负载、cpu占用率。使用内存或者cpu最高的一些进程。下面这个命令可以查看某个进程中的线程状态。

```
top -H -p pid
```

free

top也能看内存，但不友好，free是专门用来查看内存的。包括物理内存和虚拟内存swap。

df

df命令用来查看系统中磁盘的使用量，用来查看磁盘是否已经到达上限。参数`h`可以以友好的方式进行展示。


```
df -h
```

ifconfig

查看ip地址，不啰嗦，替代品是`ip addr`命令。

ping

至于网络通不通，可以使用ping来探测。（不包括那些禁ping的网站）

netstat
虽然ss命令可以替代netstat了，但现实中netstat仍然用的更广泛一些。比如，查看当前的所有tcp连接。

```
netstat -ant
```

此命令，在找一些`本地起了什么端口`之类的问题上，作用很大。

工作常用

还有一些在工作中经常会用到的命令，它们的出现频率是非常高的 ， 都是些熟面孔。

export

很多安装了jdk的同学找不到java命令， `export` 就可以帮你办到它。export用来设定一些环境变量，env命令能看到当前系统中所有的环境变量。比如，下面设置的就是jdk的。

```
export PATH=$PATH:/home/xjj/jdk/bin
```

有时候，你想要知道所执行命令的具体路径。那么就可以使用whereis命令，我是假定了你装了多个版本的jdk。

crontab

这就是linux本地的job工具。不是分布式的，你要不是运维，就不要用了。比如，每10分钟提醒喝茶上厕所。

```
*/10 * * * * /home/xjj/wc10min
```

date

date命令用来输出当前的系统时间，可以使用-s参数指定输出格式。但设置时间涉及到设置硬件，所以有另外一个命令叫做 `hwclock` 。

xargs

xargs读取输入源，然后逐行处理。这个命令非常有用。举个栗子，删除目录中的所有class文件。

```
find . | grep .class$ | xargs rm -rvf

#把所有的rmvb文件拷贝到目录
ls *.rmvb | xargs -n1 -i cp {} /mount/xiaodianying
```

网络

linux是一个多作业的网络操作系统，所以网络命令有很多很多。工作中，最常和这些打交道。

ssh

这个，就不啰嗦了。你一定希望了解`ssh隧道`是什么。你要是想要详细的输出过程，记得加参数`-v`。

scp

scp用来进行文件传输。也可以用来传输目录。也有更高级的`sftp`命令。

```
scp a.txt 192.168.0.12:/tmp/a.txt
scp -r a_dir 192.168.0.12:/tmp/
```

wget

你想要在服务器上安装jdk，不会先在本地下载下来，然后使用scp传到服务器上吧（有时候不得不这样）。wget命令可以让你直接使用命令行下载文件，并支持断点续传。

```
wget -c http://oracle.fuck/jdk2019.bin
```

mysql

mysql应用广泛，并不是每个人都有条件用上`navicat`的。你需要了解mysql的连接方式和基本的操作，在异常情况下才能游刃有余。

```
mysql -u root -p -h 192.168.1.2
```

End

不要觉得复杂，命令是有限的，但激情无限；都会也不要骄傲，一个vim就够折腾一辈子。捷径就是总结，深入只有探索。白马过隙，终会行云流水，手到擒来。

物是人非，年华易老。唯有时光，不会辜负。

> 作者简介：**小姐姐味道** (xjjdog)，一个不允许程序员走弯路的公众号。聚焦基础架构和Linux。十年架构，日百亿流量，与你探讨高并发世界，给你不一样的味道。我的个人微信xjjdog0，欢迎添加好友，进一步交流。

近期热门文章

《必看！java后端，亮剑诛仙》

后端技术索引，中肯火爆

《Linux上，最常用的一批命令解析（10年精选）》
CSDN发布首日，1k赞。点赞率1/8。

《这次要是讲不明白Spring Cloud核心组件，那我就白编这故事了》
用故事讲解核心组件，包你满意

《Linux生产环境上，最常用的一套“Sed“技巧》
最常用系列Sed篇，简单易懂。Vim篇更加易懂。

Linux生产环境上，最常用的一套“vim“技巧

> 原创：小姐姐味道（微信公众号ID：xjjdog），欢迎分享，转载请保留出处。

引子

研发线上使用最多的编辑器，就是 `vi` 。无论是最快查看某个文件内容，还是快速编辑某个文件， `vi` 都能帮上忙。

软件世界貌似有一些非常长寿的东西， `vi` 算是一个。本篇文章聚焦的是研发线上最常用的一些功能。至于安装插件，写一些脚本，那一般是在开发机上玩的，生产环境没有条件、也没有时间忍受你做这些增强。希望看完本文，能够对这款神器有一个大体印象。当然，熟练的使用还需要日常有意识的培养。

> `vim` 是 `vi` 的增强版，一般现代 `linux` 都不缺那几兆空间，所以预装的都是增强版，本文默认使用 `vim` 。

养成习惯

`vim` 最大的贡献就是它的**按键系统**。这也是为什么**chrome**、**idea**、**atom**等编辑器都会提供一个 `vim mode` 。**笔者见过很多资深的程序员，包括架构师，习惯使用方向键去控制光标的移动。**这不能说不对，但这也抛弃了 `vim` 最大的精华所在，效率上低了一大截。坚持使用 `h` 、 `j` 、 `k` 、 `l` ，你会感谢你今天的纠正。大脑和手指真的是有记忆，当你用的足够多，这也就成了你约定俗成的设定。

`vim` 另外一个特点就是**带模式的**。一共四种模式，我们不需要记忆，只需要使用例子去理解即可。

不要添乱

不要使用 `vim` 打开大文件， `vim` 会一次性读取所有内容到内存，容易造成宿主机内存溢出。打开文件前，可以使用 `du –h` 命令查看文件大小。一般， `100MB` 以下为宜。

常用操作

以下操作在普通模式下执行，连续按键

漫游

> **j** 向下
> **30j** 向下移动30行
> **k** 向上
> **h** 向左
> **l** 向右
> **0** 到行首
> **^** 到行首第一个字符，如果前面有空格的话
> **$** 到行尾
> **gg** 快速到文件头
> **G** 快速到文件尾
> **100G** 跳转到第100行

不建议在插入模式下进行光标移动，这很低效

复制：y

> **yy** 复制一行
> **10yy** 向下复制10行
> **yw** 复制光标开始的一个单词
> **y$** 复制光标到行尾
> **yfB** 复制光标到第一个大写B中间的内容
> **y2fB** 复制光标到第二个大写B中间的内容

剪切: x

> **x** 向剪切一个一个字符，如果是在行尾，则为向前剪切
> **3x** 剪切三个
> **xp** 非行尾交换两个字符，如从`bs`变成`sb`

删除：d

删除的内容会放到剪贴板，按`p`即可粘贴到其他地方

> **dd** 删除一行
> **200dd** 删除200行
> **dw** 删除一个单词 （最喜欢啦）
> **df"** 删除到出现的第一个双引号

粘贴: p

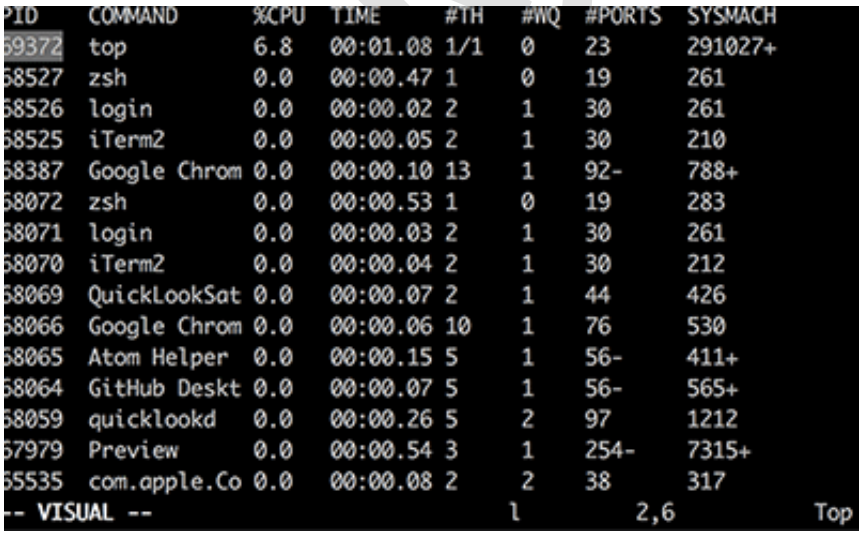
> **p** 粘贴复制或剪切的内容
> **3p** 将复制或剪切的内容粘贴三次

可视化模式

`v` 行模式，选择一些内容

可视化模式是非常有用的一种模式，在普通模式下按**v**即可进入。
使用`h`、`j`、`k`、`l`进行漫游，选中相应的内容。

例子，选中一部分想要的内容，并删除。



`ctrl+v` 块模式

> 演示：将文件中的每一行添加到`ArrayList`中：

1) 在命令模式下，执行 `%s/$/");/g` ，在行尾追加数据
2) 按 `ESC` 进入普通模式，并使用 `gg` 回到行首
3) 按 `ctrl+v` 进入可视化模式，然后按 `G` 到文件尾
4) 不要理会编辑器反应，按 `I` 进入插入模式，输入 `list.add("`
5) 按 `ESC` 回到普通模式，可以发现以上输入已经在每一行生效了

块模式还可以完成列的呼唤，貌似在 `UE` 里见过此神技。

命令模式

上面的例子里已经展示了命令模式的进入模式。**在普通模式下，输入 `:` 即可进入。**

> **%s/$/sth/** 在行尾追加sth
> **%s/^M//g** 替换掉dos换行符， `\^M` 使用 `ctrl+v + Enter` 即可输入
> **:g/^\s*$/d** 删除空行以及只有空格的行
> **%s/#.*//g** 删除 `#` 之后的字符

没错，命令模式用的是正则，这些经验是通用的

你已经发现了，这大概就是针对编辑器窗口的 `sed` 命令。

查找字符串

同样的，正则的知识也可以应用

在普通模式下，按下 `/` 直接进入查找，输入相应的字符串按确定即可。

> **n** 查找下一个匹配
> **N** 查找上一个匹配
> **2n** 查找下面第二个匹配

如果觉得跳来跳去晕头转向，可以在命令模式下输入 `set nu` 开启行号。

宏录制

这可以说是 `vim` 的一个杀手锏了。拿上面的例子来说。
将文件中的每一行添加到 `ArrayList` 中。

1) 按下 `gg` 到行首
2) 按下 `qa` 进行宏录制， `a` 是我们起的一个标记名称
3) 按 `I` 进入插入模式，输入 `list.add("`

4) 按 `ESC` 进入普通模式，然后按 `$` 跳到行尾
5) 按 `j` 进入下一行，然后按 `^` 回到行首
6) 再次按下 `q` 结束宏录制
7) 输入 `@a` 触发宏测试一下录制效果
8) 输入 `100@a` 重复宏100次，也就是影响下面的100行

可以录制不同的多个宏，方面的进行批量操作

其他

另外用的一些比较少的主要功能有

> **r** 替换字符
> **ggVG** 全选
> **u** 恢复更改
> **J** 合并下一行
> **gU** 光标处转大写
> **ggguG** 整篇文章大写转化为小写
> **%** 跳转到下一个匹配,如在 `<div>` 上按 `%`，则跳转到相应的 `</div>`
> **:e /tmp/a** 在同一个编辑器内打开 `/tmp/a` 文件。同一个编辑器的缓冲区是剪贴板是共享的，可以方便在多个文件中复制
> **bp** 跳转到上一个缓冲区
> **bn** 跳转到下一个缓冲区

退出编辑器

> **wq** 保存当前文件并退出
> **wqa** 保存所有文件并退出
> **q!** 不保存，直接退出
> **qa!** 有多个文件被打开，同时退出

本篇文章只聚焦常用功能，帮助读者快速处理线上文本。至于更多的，也装不下，只有你自己去探索喽。

`vim` 的入门门槛比较高，幸运的是，用多了，你就无法释手了。

> 作者简介：**小姐姐味道** (xjjdog)，一个不允许程序员走弯路的公众号。聚焦基础架构和Linux。十年架构，日百亿流量，与你探讨高并发世界，给你不一样的味道。我的个人微信xjjdog0，欢迎添加好友，进一步交流。

近期热门文章

《必看！java后端，亮剑诛仙》

后端技术索引，中肯火爆

《Linux上，最常用的一批命令解析（10年精选）》
CSDN发布首日，1k赞。点赞率1/8。

《这次要是讲不明白Spring Cloud核心组件，那我就白编这故事了》
用故事讲解核心组件，包你满意

《Linux生产环境上，最常用的一套“Sed“技巧》
最常用系列Sed篇，简单易懂。Vim篇更加易懂。

Linux生产环境上，最常用的一套“AWK“技巧

> 原创：小姐姐味道（微信公众号ID：xjjdog），欢迎分享，转载请保留出处。

hi，大家好，小姐姐味道最有用系列完结。记得多多转发，点赞哦。

最有用系列：

《Linux生产环境上，最常用的一套“vim“技巧》

《Linux生产环境上，最常用的一套“Sed“技巧》

《Linux生产环境上，最常用的一套“AWK“技巧》

《"Sed" 高级功能：我这小脑瓜都快绕晕了》

最有用系列：

《Linux生产环境上，最常用的一套“vim“技巧》
《Linux生产环境上，最常用的一套“Sed“技巧》
《"Sed" 高级功能：我这小脑瓜都快绕晕了》

敢用自己的名字做软件名字的，都有非常强大的自信。比如，垠语言什么的。

`awk`的命名得自于它的三个创始人姓别的首字母，都是`80来岁`的老爷爷了。当然也有四个人的组合：流行的GoF设计模式。但对于我这游戏爱好者来说，想到的竟然是三位一体，果然是不争气啊。

它长的很像C，为什么这么有名，除了它强大的功能，我们姑且认为`a`这个字母比较靠前吧。`awk`比`sed`简单，它更像一门编程语言。

打印某一列

下面，这几行代码的效果基本是相同的：打印文件中的第一列。

```
#Java
System.out.println(aStr.split(" ")[0]);
```



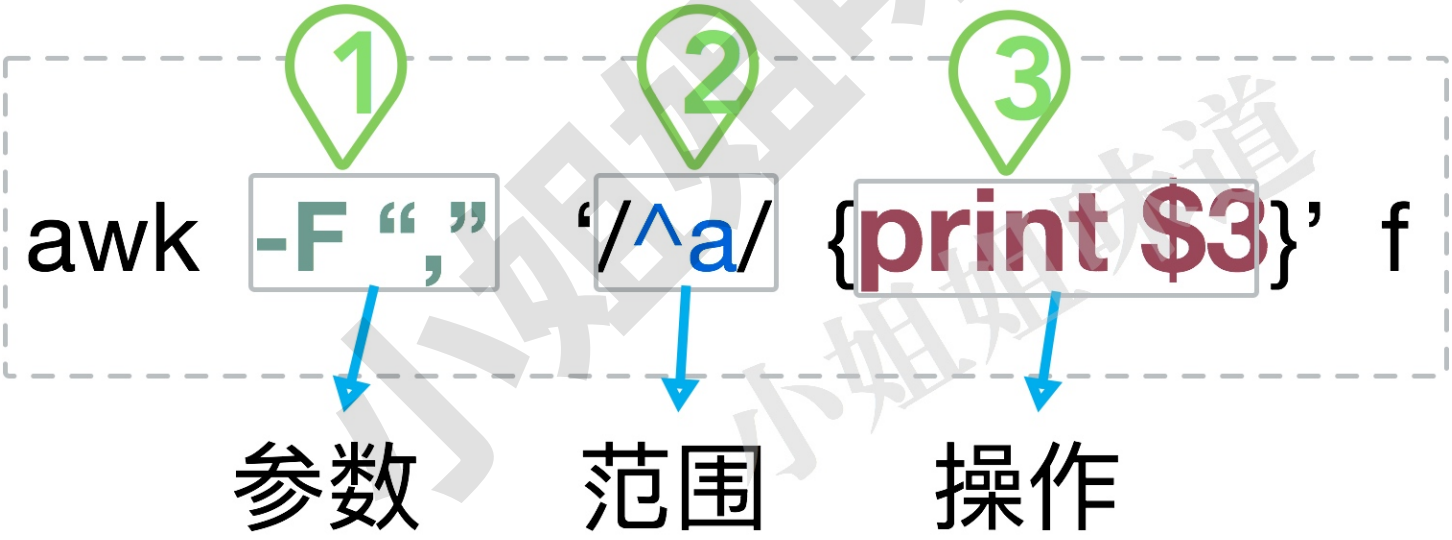
```
#Python
print(aString.split(" ")[0])

#cut 命令
cut -d " " -f1   file

#awk命令
awk '{print $1}' file
```

这可能是awk最常用的功能了：**打印文件中的某一列**。它智能的去切分你的数据，不管是`空格`，还是`TAB`，大概率是你想要的。

对于csv这种文件来说，分隔的字符是`,`。AWK使用`-F`参数去指定。以下代码打印csv文件中的第1和第2列。

```
awk -F ","  '{print $1,$2}' file
```

由此，我们可以看出一个基本的awk命令的组成部分。



> 一般的开发语言，数组下标是以0开始的，但awk的列`$`是以`1`开始的，而`0`指的是原始字符串。

网络状态统计

本小节，采用awk统计netstat命令的一些网络状态，来看一下awk语言的基本要素。netstat的输出类似于：


```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:9999            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:39735           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:35704           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:10050           0.0.0.0:*               LISTEN
tcp        0      0 10.66.204.156:9092      0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:9000            0.0.0.0:*               LISTEN
tcp        0      0 10.66.204.156:60756     10.80.86.57:9092        TIME_WAIT
tcp        0      0 10.66.204.156:9092      10.81.28.181:53454      ESTABLISHED
tcp        0      0 127.0.0.1:37642         127.0.0.1:8000          TIME_WAIT
tcp        0      0 10.66.204.156:45568     10.66.204.156:9092      TIME_WAIT
tcp        0      0 10.66.204.156:9092      10.81.28.181:53464      ESTABLISHED
tcp        0      0 10.66.204.156:37926     10.80.55.181:2181       ESTABLISHED
tcp        0      0 127.0.0.1:37870         127.0.0.1:8000          TIME_WAIT
```

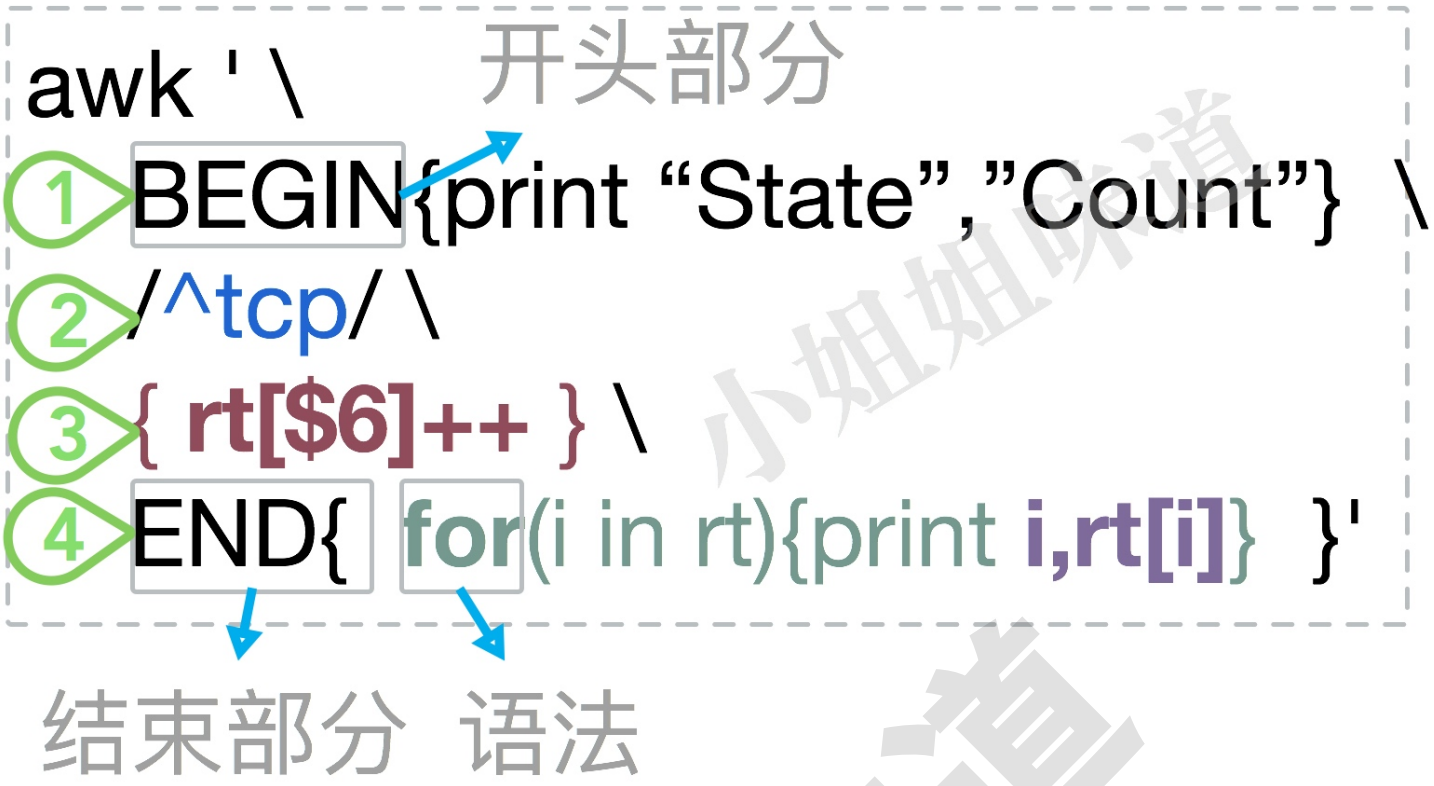
其中，第6列，标明了网络连接所处于的网络状态。我们先给出awk命令，看一下统计结果。

```
netstat  -ant |
awk ' \
    BEGIN{print  "State","Count" }  \
    /^tcp/ \
    { rt[$6]++ } \
    END{  for(i in rt){print i,rt[i]}  }'
```

输出结果为：

```
State Count
LAST_ACK 1
LISTEN 64
CLOSE_WAIT 43
ESTABLISHED 719
SYN_SENT 5
TIME_WAIT 146
```

下面这张图会配合以上命令详细说明，希望你能了解awk的精髓。



乍一看，好吓人的命令，但是很简单。awk和我们通常的程序不太一样，它分为四个部分。

1、**BEGIN 开头部分**，可选的。用来设置一些参数，输出一些表头，定义一些变量等。上面的命令仅打印了一行信息而已。

2、**END 结尾部分**，可选的。用来计算一些汇总逻辑，或者输出这些内容。上面的命令，使用简单的for循环，输出了数组rt中的内容。

3、**Pattern 匹配部分**，依然可选。用来匹配一些需要处理的行。上面的命令，只匹配tcp开头的行，其他的不进入处理。

4、**Action 模块**。主要逻辑体，按行处理，统计打印，都可以。

> ### 注意点
>
> 1、awk的主程序部分使用单引号‘包围，而不能是双引号
> 2、awk的列开始的index是0，而不是1

例子

我们从几个简单的例子，来看下awk的作用。

1、输出Recv-Q不为0的记录


```
netstat -ant | awk '$2 > 0 {print}'
```

2、外网连接数，根据ip分组

```
netstat -ant | awk '/^tcp/{print $4}' | awk -F: '!/^:/{print $1}' | sort | uniq -c
```

3、打印RSS物理内存占用

```
top -b -n 1 | awk 'NR>7{rss+=$6}END{print rss}
```

4、过滤（去掉）空白行

```
awk 'NF' file
```

5、打印奇数行

```
awk 'a=!a' file
```

6、输出行数

```
awk 'END{print NR}' file
```

这些命令，是需要了解awk的一些内部变量的，接下来我们来介绍。

内置变量

FS

下面的两个命令是等价的 。

```
awk -F ':'  '{print $3}' file
awk 'BEGIN{FS=":"}{print $3}' file
```

BEGIN块中的`FS`，就是内部变量，可以直接指定或者输出。如果你的文件既有用`,`分隔的，也有用`:`分割的，FS甚至可以指定多个分隔符同时起作用。

```
FS="[,:|]"
```

其他

OFS 指定输出内容的分割符，列数非常多的时候，简化操作。相似命令：

```
awk -F ':' '{print $1,"-",$2,"-",$4}' file
awk 'BEGIN{FS=":";OFS="-"}{print $1,$2,$4}' file
```

NF 列数。非常有用，比如，过滤一些列数不满足条件的内容。

```
awk -F, '{if(NF==3){print}}' file
```

NR 行号，例如，下面两个命令是等价的。

```
cat -n file
awk '{print NR,$0}' file
```

RS 记录分隔标志
ORS 指定记录输出的分隔标志

FILENAME 当前处理的文件名称，在一次性处理多个文件时非常有用

编程语言特性

数学运算

从上面的代码可以看出，awk可以做一些简单的运算。它的语言简洁，不需要显示的定义变量的类型。

比如上面的`rt[$6]++`，就已经默认定义了一个叫做rt的hash(array?)，里面的key是网络状态，而value是可以进行运算的(+-*/%)。

包含一些内置的数学运算（有限)

```
int
log
sqrt
```

```
exp
sin
cos
atan2
rand
srand
```

字符串操作

类似其他语言，awk也内置了很多字符串操作函数。它本来就是处理字符串的，所以必须强大。

```
length(str) #获取字符串长度
split(input-string,output-array,separator)
substr(input-string, location, length)
```

语言特性

awk是个小型的编程语言，看它的基本语法，如果你需要复杂一点的逻辑，请自行深入了解，包括一些时间处理函数：

```
# logic
if(x=a){}
if(x=a){}else{}
while(x=a){break;continue;}
do{}while(x=a)
for(;;){}

# array
arr[key] = value
for(key in arr){arr[key]}
delete arr[key]

asort(arr) #简单排序
```

据说，awk可以胜任所有的文本操作。因为它本身就是一门语言啊。

End

曾经使用awk编写过复杂的日志处理和统计程序。虽然比写 `sed` 舒畅了很多，但还是备受煎熬。更加上现在有各种nawk,gawk版本之间的区别，所以业务复杂度一增长，就习惯性的转向更加简

洁、工具更全的python。

awk处理一些简单的文本还是极其方便的，最常用的还是打印某一列之类的，包括一些格式化输出。对于awk，要简单的滚瓜烂熟，复杂的耳熟能详，毕竟有些`大牛`，就喜欢写这种脚本呢。

> 作者简介：**小姐姐味道** (xjjdog)，一个不允许程序员走弯路的公众号。聚焦基础架构和Linux。十年架构，日百亿流量，与你探讨高并发世界，给你不一样的味道。我的个人微信xjjdog0，欢迎添加好友，进一步交流。

近期热门文章

《必看！java后端，亮剑诛仙》

后端技术索引，中肯火爆

《Linux上，最常用的一批命令解析（10年精选）》
CSDN发布首日，1k赞。点赞率1/8。

《这次要是讲不明白Spring Cloud核心组件，那我就白编这故事了》
用故事讲解核心组件，包你满意

《Linux生产环境上，最常用的一套“Sed“技巧》
最常用系列Sed篇，简单易懂。Vim篇更加易懂。

Linux生产环境上，最常用的一套“Sed“技巧

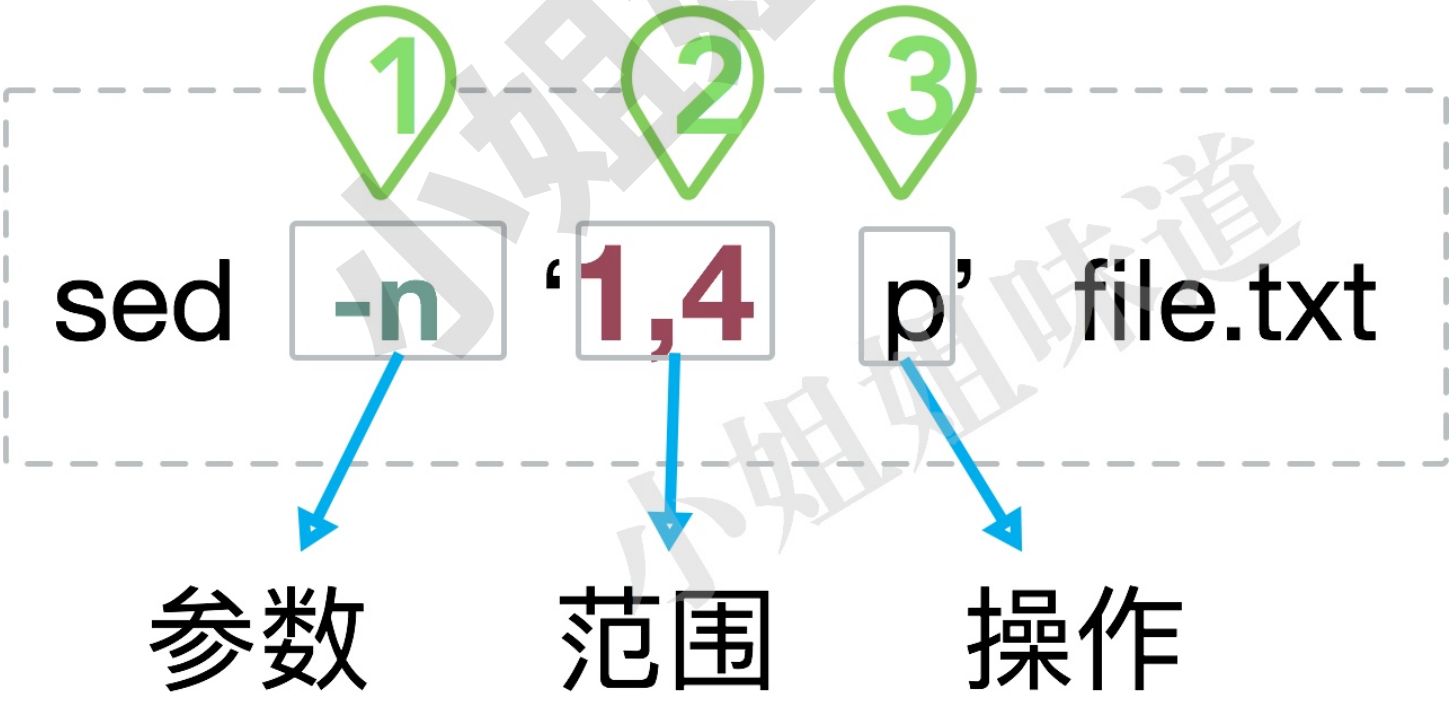
> 原创：小姐姐味道（微信公众号ID：xjjdog），欢迎分享，转载请保留出处。

`sed`命令应用广泛，使用简单，是快速文本处理的利器。它其实没多少技巧，背诵、使用是最合适的学习渠道，属于硬技能。但它又很复杂，因为高级功能太多。本篇不去关注sed的高级功能，仅对常用的一些操作，进行说明。

随着使用，你会发现它和`vim`的一些理念是想通的，正则表达式的语法也基本上一样，并没有多少学习成本。从个人视野和工作效率上来看，sed命令都是程序员必须掌握的一个重要工具。

> 那些说可以现场google用法的，大多习惯将文本拷贝到excel里，慢慢磨洋工，遇到大批量文件更是手忙脚乱。不是一家人不进一家门，本文不是为你写的。

一个简单的入门



如图，一个简单的sed命令包含三个主要部分：`参数`、`范围`、`操作`。要操作的文件，可以直接挂在命令行的最后。除了命令行，sed也可以通过-f参数指定一个sed脚本，这个属于高级用法，不做过多描述。

有些示例命令我会重复多次，聪明如你一定能发现其中规律，有时连解释都用不着。

参数

-n 这个参数是`--quiet`或者`--silent`的意思。表明忽略执行过程的输出，只输出我们的结果即可。

我们常用的还有另外一个参数：`-i`。

> 使用此参数后，所有改动将在原文件上执行。你的输出将覆盖原文件。**非常危险**，一定要注意。

范围

1,4 表示找到文件中1,2,3,4行的内容。
这个范围的指定很有灵性，请看以下示例（请自行替换图中的范围部分）。

> **5** 选择第5行。
> **2,5** 选择2到5行，共4行。
> **1~2** 选择奇数行。
> **2~2** 选择偶数行。
> **2,+3** 和`2,5`的效果是一样的，共4行。
> **2,$** 从第二行到文件结尾。

范围的选择还可以使用正则匹配。请看下面示例。

> **/sys/,+3** 选择出现sys字样的行，以及后面的三行。
> **/^sys/,/mem/** 选择以sys开头的行，和出现mem字样行之间的数据。

为了直观，下面的命令一一对应上面的介绍，范围和操作之间是可以有空格的。

```
sed -n '5p' file
sed -n '2,5 p' file
sed -n '1~2 p' file
sed -n '2~2 p' file
sed -n '2,+3p' file
sed -n '2,$ p' file

sed -n '/sys/,+3 p' file
sed -n '/^sys/,/mem/p' file
```

操作

最常用的操作就是`p`，意思就是打印。比如，以下两个命令就是等同的：

```
cat file
sed -n 'p' file
```

除了打印，还有以下操作，我们来说常用的。

> **p** 对匹配内容进行打印。
> **d** 对匹配内容进行删除。这个时候就要去掉`-n`参数了，想想为什么。
> **w** 将匹配内容写入到其他地方。

`a`，`i`，`c`等操作虽基本但使用少，不做介绍。我们依然拿一些命令来说明。

```
sed -n '2,5 p' file
sed    '2,5 d' file
sed -n '2,5 w output.txt' file
```

我们来看一下sed命令都能干些啥，上点命令体验一下。

删除所有#开头的行和空行。

```
sed -e 's/#.*//' -e '/^$/ d' file
```

最常用的，比如下面这个。

```
sed -n '2p' /etc/group
```

表示打印group文件中的第二行。

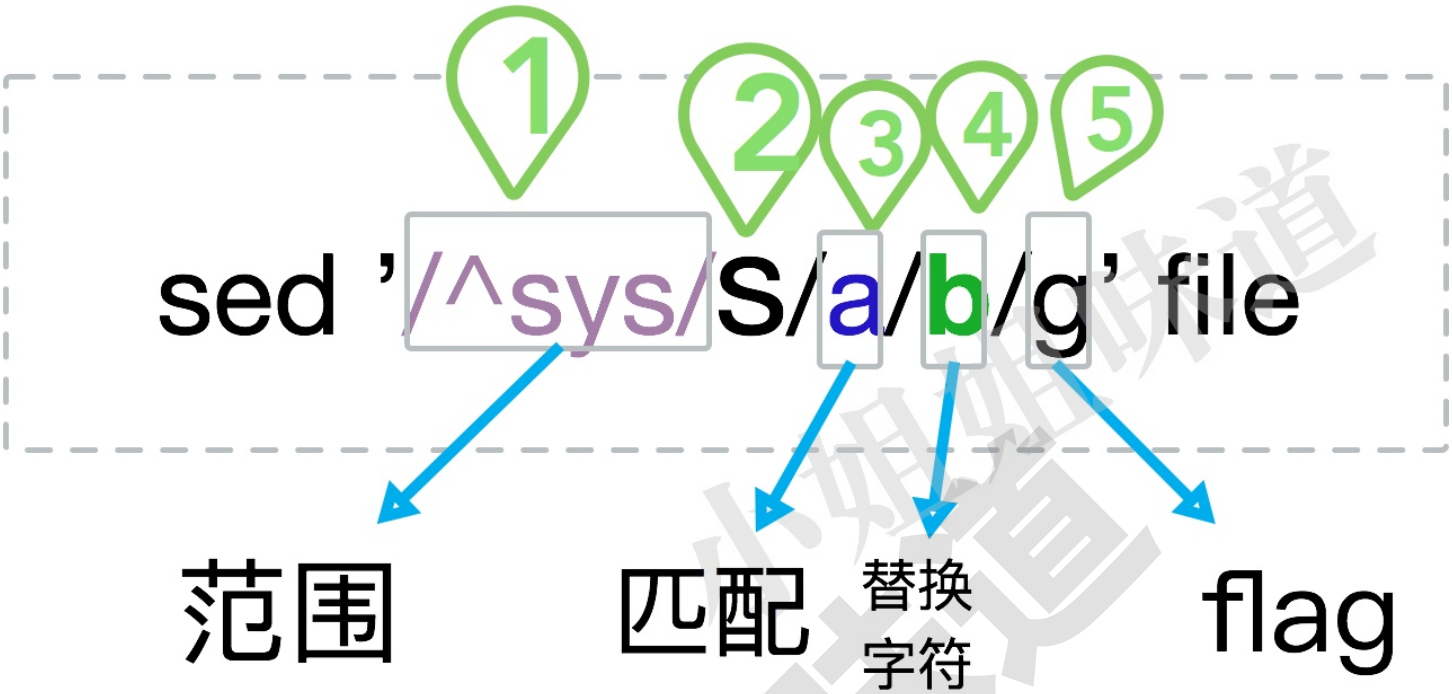
```
1、参数部分 比如 -n
2、模式部分 比如'2p'
3、文件，比如/etc/group
```

那么我想一次执行多个命令，还不想写sed脚本文件怎么办？那就需要加-e参数。

sed的操作单元是`行`。

替换模式

以上是 `sed` 命令的常用匹配模式，但它还有一个强大的替换模式，意思就是查找替换其中的某些值，并输出结果。使用替换模式很少使用 `-n` 参数。



替换模式的参数有点多，但第一部分和第五部分都是可以省略的。替换后会将**整个文本**输出出来。

前半部分用来匹配一些范围，而后半部分执行替换的动作。

范围

这个范围和上面的范围语法类似。看下面的例子。

> **/sys/,+3** 选择出现sys字样的行，以及后面的三行。
> **/^sys/,/mem/** 选择以sys开头的行，和出现mem字样行之间的数据。

具体命令为：

```
sed '/sys/,+3 s/a/b/g' file
sed '/^sys/,/mem/s/a/b/g' file
```

命令

这里的命令是指s。也就是substitute的意思。

查找匹配

查找部分会找到要被替换的字符串。这部分可以接受纯粹的字符串，也可以接受正则表达式。看下面的例子。

> **a** 查找范围行中的字符串 `a` 。
> **[a,b,c]** 从范围行里查找字符串a或者b或者c。

命令类似：

```
sed 's/a/b/g' file
sed 's/[a,b,c]/<&>/g' file
#这个命令我们下面解释
```

替换

是时候把找出的字符串给替换掉了。本部分的内容将替换查找匹配部分找到的内容。

可惜的是，这部分不能使用正则。常用的就是精确替换。比如把a替换成b。

但也有高级功能。和java或者python的正则api类似，sed的替换同样有 `Matched Pattern` 的含义，同样可以得到Group，不深究。常用的替位符，就是 `&` 。

`&` 号，再重复一遍。当它用在替换字符串中的时候，代表的是原始的查找匹配数据。

> **[&]** 表明将查找到的数据使用[]包围起来。
> **"&"** 表明将查找的数据使用""包围起来。

下面这条命令，将会把文件中的每一行，使用引号包围起来。

```
sed 's/.*/"&"/' file
```

flag 参数

这些参数可以单个使用，也可以使用多个，仅介绍最常用的。

> **g** 默认只匹配行中第一次出现的内容，加上g，就可以全文替换了。常用。
> **p** 当使用了 `-n` 参数， `p` 将仅输出匹配行内容。
> **w** 和上面的w模式类似，但是它仅仅输出有变换的行。

> **i** 这个参数比较重要，表示忽略大小写。
> **e** 表示将输出的每一行，执行一个命令。不建议使用，可以使用xargs配合完成这种功能。

看两个命令的语法：

```
sed -n 's/a/b/gipw output.txt' file
sed 's/^/ls -la/e' file
```

好玩

由于正则的关系，很多字符需要转义。你会在脚本里做些很多`\\`，`\*`之类的处理。你可以使用`|^@!`四个字符来替换`\`。
比如，下面五个命令是一样的。

```
sed '/aaa/s/\/etc/\/usr/g' file
sed '/aaa/s@/etc@/usr@g' file
sed '/aaa/s^/etc^/usr^g' file
sed '/aaa/s|/etc|/usr|g' file
sed '/aaa/s!/etc!/usr!g' file
```

注意：前半部分的范围是不能使用这种方式的。我习惯使用符号`@`。

其他

正则表达式

可以看到，正则表达式在命令行中无处不在。以下，紧做简要说明。

> **^** 行首
> **$** 行尾
> **.** 单个字符
> ***** 0个或者多个匹配
> **+** 1个或者多个匹配
> **?** 0个或者1个匹配
> **{m}** 前面的匹配重复m次
> **{m,n}** 前面的匹配重复m到n次
> ** 转义字符

> **[0-9]** 匹配括号中的任何一个字符,or的作用
> | or，或者
> **\b** 匹配一个单词。比如 `\blucky\b` 只匹配单词lucky

参数i

上面已经简单介绍了参数i，它的作用是让操作在原文件执行。无论你执行了啥，原始文件都将会被覆盖。这是非常危险的。
通过加入一个参数，可以将原文件做个备份。

```
sed -i.bak 's/a/b/' file
```

以上命令会对原file文件生效，并生成一个file.bak文件。强烈建议使用i参数同时指定bak文件。

表演一下

我们通过两个命令，来稍微看下sed和其他命令组合起来的威力。

输出长度不小于50个字符的行

```
sed -n '/^.{50}/p'
```

统计文件中有每个单词出现了多少次

```
sed 's/ /\n/g' file | sort | uniq -c
```

查找目录中的py文件，删掉所有行级注释

```
find ./ -name "*.py" | xargs sed  -i.bak '/^[ ]*#/d'
```

查看第5-7行和10-13行

```
sed -n -e '5,7p' -e '10,13p' file
```

仅输出ip地址

```
ip route show | sed -n '/src/p' | sed -e 's/  */ /g' | cut -d' ' -f9
```

End

本文配合《Linux生产环境上，最常用的一套“vim“技巧》
一文，查看更佳，你会发现很多相似的东西，这和`KISS`原则是密不可分的。
sed还有一个精华就是`x`(Exchange)命令，但也属于高级功能。有些你可能在很多Makefile里见过了。sed甚至还可以写推箱子游戏，虽然代码很酷，但，脑回路完全不够用啊。

```
https://github.com/aureliojargas/sokoban.sed
```

> 作者简介：**小姐姐味道** (xjjdog)，一个不允许程序员走弯路的公众号。聚焦基础架构和Linux。十年架构，日百亿流量，与你探讨高并发世界，给你不一样的味道。我的个人微信xjjdog0，欢迎添加好友，进一步交流。

近期热门文章

《必看！java后端，亮剑诛仙》

后端技术索引，中肯火爆

《Linux上，最常用的一批命令解析（10年精选）》
CSDN发布首日，1k赞。点赞率1/8。

《这次要是讲不明白Spring Cloud核心组件，那我就白编这故事了》
用故事讲解核心组件，包你满意

《Linux生产环境上，最常用的一套“Sed“技巧》
最常用系列Sed篇，简单易懂。Vim篇更加易懂。

"Sed" 高级功能：你需要一个人肉状态机

> 原创：小姐姐味道（微信公众号ID：xjjdog），欢迎分享，转载请保留出处。

sed命令有两个空间，一个叫pattern space，一个叫hold space。这两个空间能够证明人类的脑瓜容量是非常小的，需要经过大量的训练和烧脑的理解，才能适应一些非常简单的操作。

不信看下面简单的命令，作用是，删除文件中最后两行。

```
sed 'N; $!P;$!D;$d' file
```

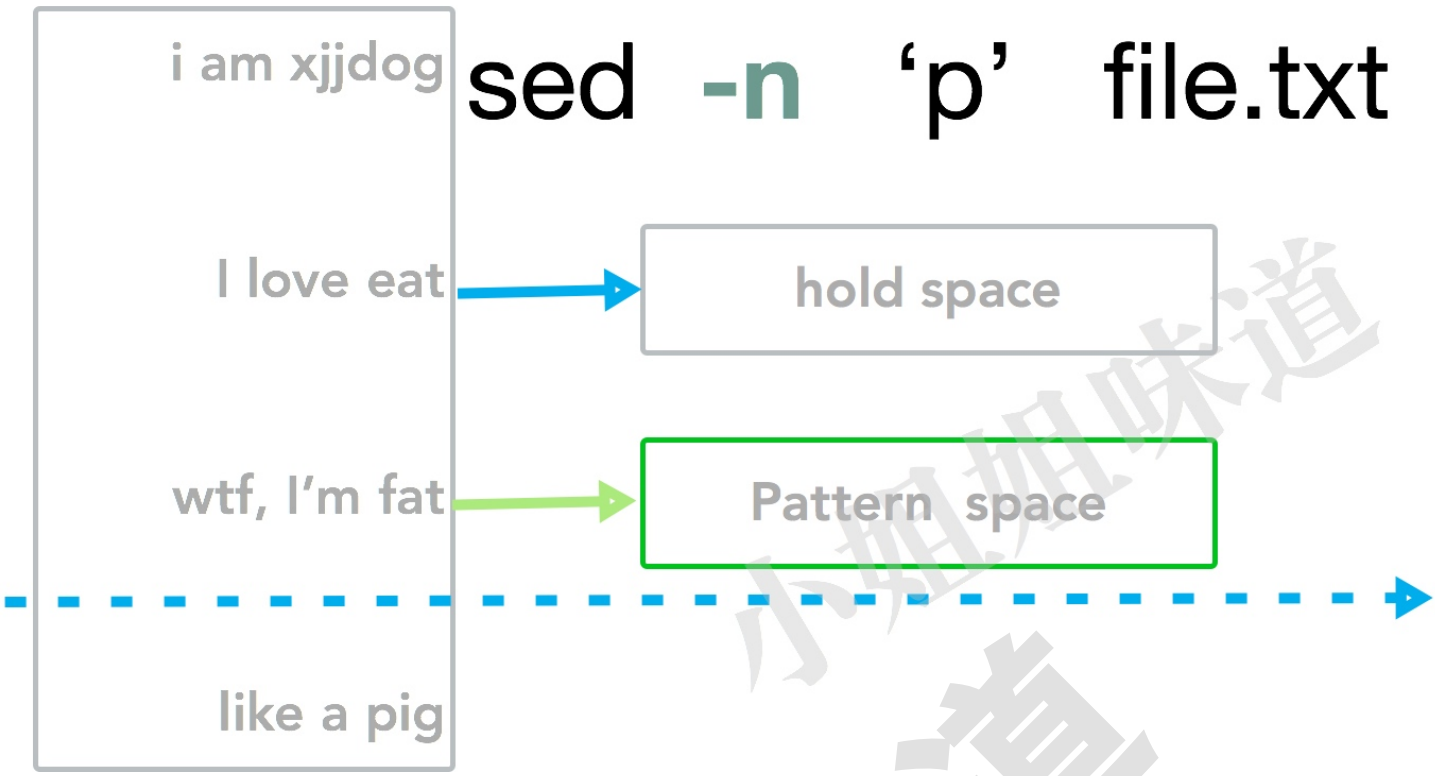
在《Linux生产环境上，最常用的一套“Sed“技巧》一文中，我们介绍了常用的sed命令和操作，而且使用了两张图来作为辅助。但可惜的是，这两张图，严格来说是不准确的 (比如s命令，只是其中的一个子集)，即使它能够帮助初学者快速入门。

本篇属于sed的中级用法，常见在一些sed脚本中，在日常中应用并不多，但往往能够获得意想不到的效果。

原理

工作模式

这要从sed的工作模式来说起。



按照我们读取一个文件的尿性，一般是持续循环读取。比如下面的python代码，print代表p命令。

```
with open('file', 'r') as f:
    for line in f.readlines():
        print(line)
```

sed命令在这之上，还缓冲了另外一个东西。那就是“上一行的内容” ,叫做hold space。而当前行，叫做patter space。用python简单表现一下：

```
hold_space = ""
with open('file', 'r') as f:
    for pattern_space in f.readlines():
        print(hold_space,pattern_space)
        hold_space = pattern_space
```

具体过程，大体与上面的代码类似，以上面的图为例。在这个例子中，hold space不参与运算，是全程无感的：

1、 读取当前行 `wtf..` 到 Pattern space

2、 执行命令 `p` ，这会打印出当前行

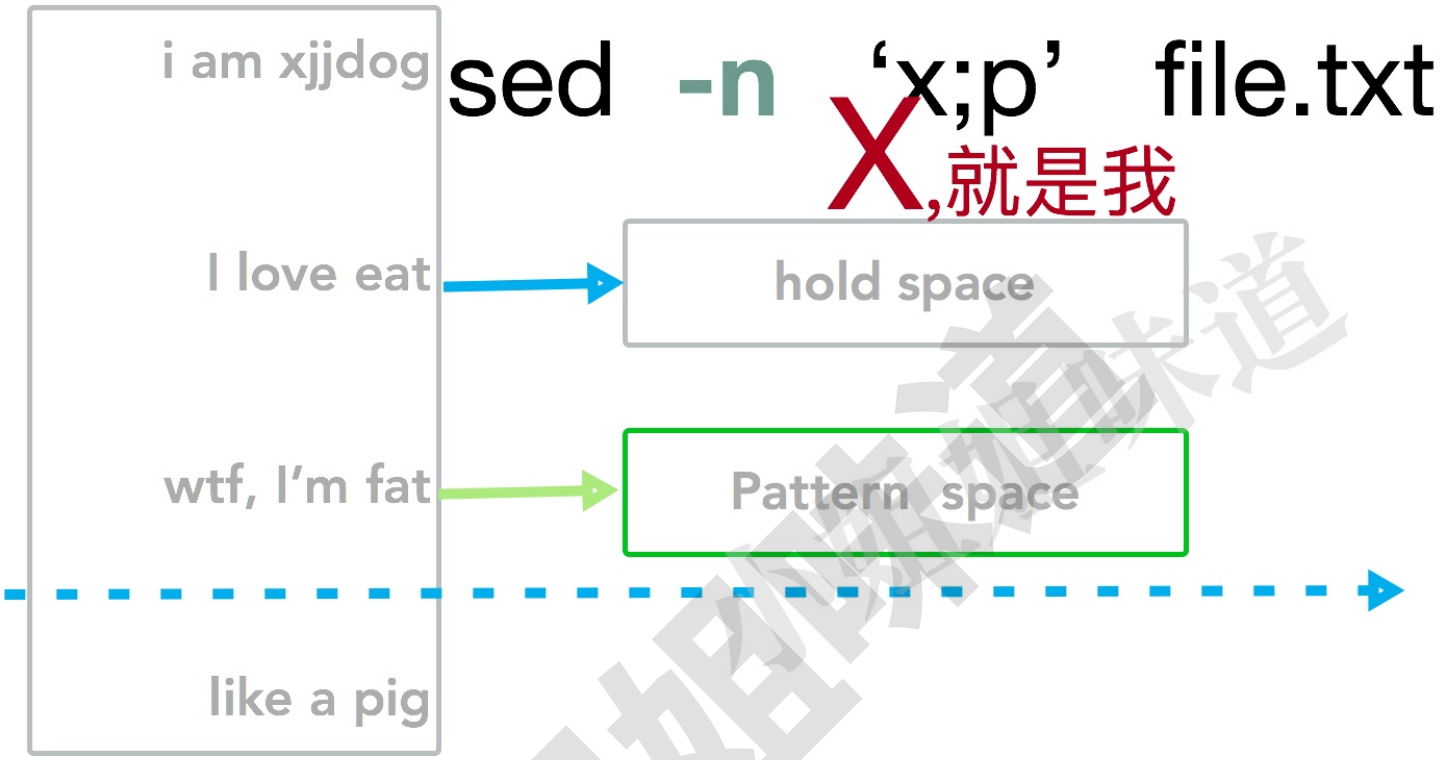
3、 把Pattern space的内容，赋值给Hold space

4、 继续下一行的处理，循环这个过程

一个例子:x

但我想稍微操作一下这两个缓冲区。这个操作就是swap，使用x表示，这也是一些文本编辑器的一贯尿性。

也就是，在p之前，我们加上了个x。表示先将这两个缓冲区进行置换，然后再往下走。




```
hold_space = ""
with open('file', 'r') as f:
    for pattern_space in f.readlines():
        swap(hold_space,pattern_space)
        print(hold_space,pattern_space)
        hold_space = pattern_space
```

让我我们来想象一下这个过程。

> 1、 刚开始，hold_space的内容是空。然鹅，还没被填充，它就被使用了，和当前行进行了置换
>
> 2、 p命令用在了置换后的缓冲区上，第一次打印出了空行，fuck
>
> 3、 继续嘟嘟嘟，现在到了最后一行，马上进行了置换，没机会打印就到了hold_space中了
>
> 4、 当前行，存放的是倒数第二行的数据，最后一行见光死，就永远没有机会面世了

我们当然有办法把它搞出来，比如，我执行偶数次的交换x。


```
sed -n 'x;x;x;x;p' file
```

有木有一股骑着羊驼走天下的的感觉?

应用

举个栗子

你可能会想，我对这两个缓冲区交换，有什么用？接下来看这个文件。

```
小姐姐味道公众号
CEO
加菲猫经理
IT Manager
系统毁灭师
Sysadmin
小哥哥味道
Developer
爱卖东西的经理
Sales Manager
风清扬
Dog
```

文件奇数行是名称，偶数行是职位。我们想要输出所有`Manager`的名字。就可以使用下面的命令。

```
sed -n -e 'x;n' -e '/Manager/{x;p}' file
```

命令分为两个部分。

`x;n` 表示将偶数行保存在pattern space，那么奇数行就保存在hold space中。

`/Manager/{x;p}` 命令将在pattern space上执行对Manager关键字的查找。如果符合条件，则再次交换p和h缓冲区，输出奇数行对应的名字。

上面的`x`和`n`，就是针对这两个缓冲区的命令。这样的命令有很多。

命令

这些命令，如果多了，可以使用{}包围起来，就像上面的命令一样。这些命令的位置与我们上一篇所说的，在同一个地方。

常用的：

> **x** 请容许我用英文装个b：Exchange the contents of the hold and pattern spaces.
>
> **d** 清空当前的pattern space，然后进入下一个循环
>
> **D** 删除pattern space的第一行（multiline pattern）
>
> **h** 复制pattern space到hold space
>
> **H** 追加pattern spaced到hold space
>
> **g** 复制hold space到pattern space
>
> **G** 追加hold space到pattern space
>
> **n** 读取下一个输入行到pattern space
>
> **N** 追加下一个输入行到pattern space，同时将两行看做一行，但是两行之间依然含有\n换行符
>
> **p** 打印当前的pattern space
>
> **P** 打印当前的pattern space中的第一行

不常用的

上次提到的推箱子游戏，就用了很多这种东西。为了使使用者在书写sed脚本的时候真正的"自由"，sed还允许在脚本中用":"设置记号。标签，有种类似编程语言的特性了。

> **q** 退出sed，可以增加执行速度
>
> **l** 列出当前行，包含不可打印字符
>
> **l width** 列出当前行，使用一个`width characters`结尾
>
> **b label** 跳到相应的标签，分之命令。
>
> **t label** if分支，从最后一行开始，条件一旦满足或者T，t命令，将导致分支到带有标号的命令处，或者到脚本的末尾。测试命令。
>
> **T label** 错误分支，从最后一行开始，一旦发生错误或者T，t命令，将导致分支到带有标号

> 的命令处，或者到脚本的末尾。

当然还有其他更不常用的，可以使用man命令查看

```
man sed
```

一些命令：开启训练模式

看着一行行进行处理，好像很简单是不是？不可能的，看下面几个简单的命令，训练一下生锈的脑子。

一个流水线一样的命令

```
sed -n '2{h;n;x;H;x};p' file
```

> 交换第2行和第3行的内容

输出最后一行

```
sed 'N;D' file
```

输出文件中最后两行

```
sed '$!N; $!D' file
```

删除文件中最后两行

```
sed 'N; $!P;$!D;$d' file
```

打印偶数行的另一种写法

```
sed -n 'n;p' file
```

每隔5行加入一个空行。

```
sed 'n;n;n;n;G' file
```

输出含AAA和BBB和CCC（任意顺序）的段落

```
sed -e '/./{H; $!d;}' -e 'x;/AAA/!d; /BBB/!d; /CCC/!d' file
```

颠倒行序（使末行变首行，首行变末行）

```
sed -n '1!G; h; $p' file
```

> 这个命令有点绕，首先，`1!G`对除了第一行的其他行进行了G操作，然后反向复制回去，到了最后一行，就直接打印Pattern Space。`$`表示到了最后一行执行下面的命令，也可以是`${p}`

一个带标签的用法

```
sed -e :a -e '$q;N;11,$D;ba'
```

打印最后10行。`a`是标签。语法就是单独的行，使用`:`分隔。

End

为了提高你在公司的竞争力，你是可以弄一堆sed脚本唬人（埋雷）的。和某些perl脚本一样，即使是有相关经验的开发着，理解起来也要下点功夫，就不要说其他人了。

这就是sed，简约但不简单的命令，本文算是一个中级入门。中级入门也有点烧脑，因为你的脑瓜里，需要一直维护着这两个缓冲区。又是置换，又是清空，相当于人肉状态机。当然，怎么把这个过程讲的尽量简单一点，还是浪费了作者不少脑细胞的。哪怕你点个赞，也是延缓小姐姐走向老年痴呆时间的一个途径。有了你的支持，小姐姐也可以想点技术之外的事情，比如喷喷bat什么的。

> 作者简介：**小姐姐味道** (xjjdog)，一个不允许程序员走弯路的公众号。聚焦基础架构和Linux。十年架构，日百亿流量，与你探讨高并发世界，给你不一样的味道。我的个人微信xjjdog0，欢迎添加好友，进一步交流。

近期热门文章

《必看！java后端，亮剑诛仙》

后端技术索引，中肯火爆

《Linux上，最常用的一批命令解析（10年精选）》
CSDN发布首日，1k赞。点赞率1/8。

《这次要是讲不明白Spring Cloud核心组件，那我就白编这故事了》

用故事讲解核心组件，包你满意

《Linux生产环境上，最常用的一套“Sed“技巧》

最常用系列Sed篇，简单易懂。Vim篇更加易懂。

Linux之《荒岛余生》（一）准备篇

> 原创：小姐姐味道（微信公众号ID：xjjdog），欢迎分享，转载请保留出处。

xin片之争，已经暴露了中国xin的问题，我等码农束手无策；而在操作系统方面，成果也是乏善可陈；现如今酷炫的Web监控工具，让很多研发丧失了真正处理问题的能力。

越接近底层，就越接近真相，在计算机的世界，同样适用。

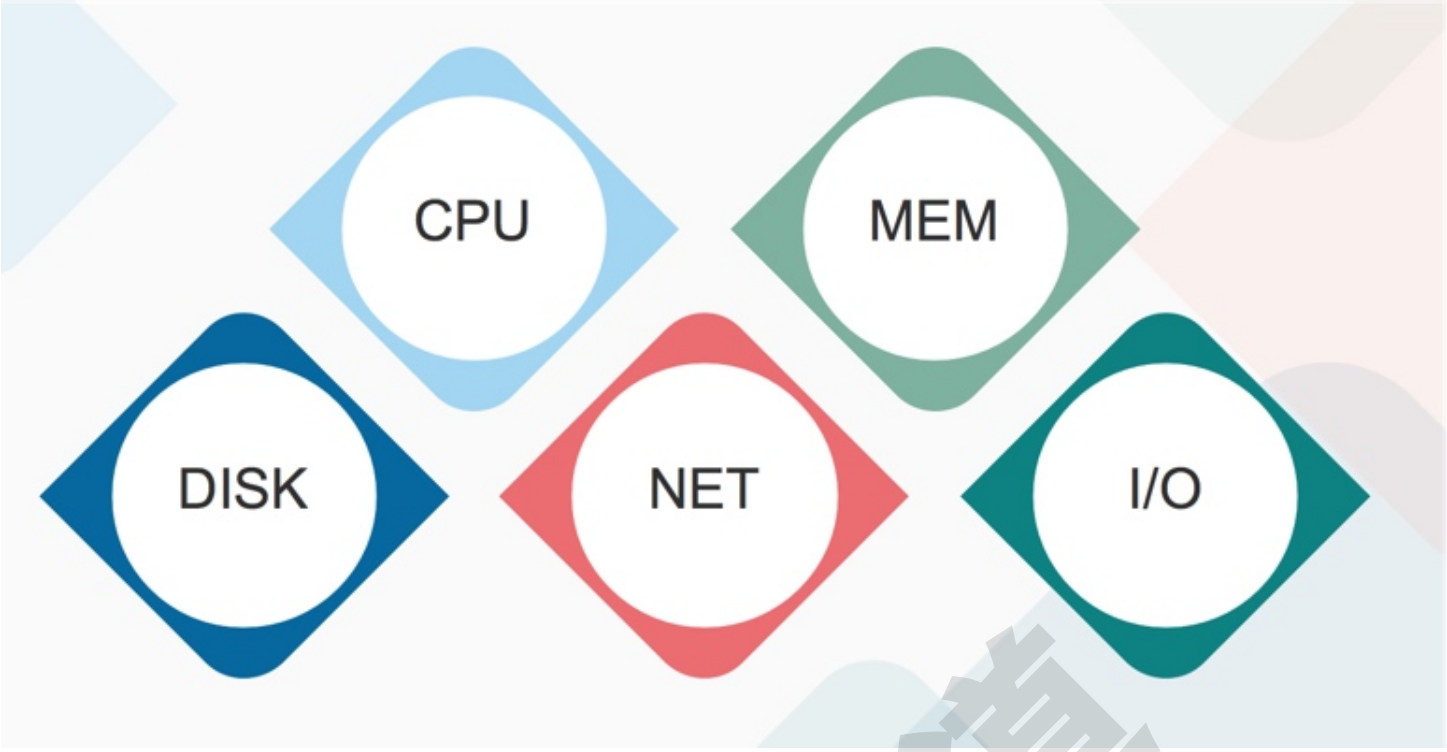
我们的目的，就像是《荒岛余生》一样：找到一个信念，在最残酷的环境中，生存下去。说的比较隐晦，其实就是：你换公司了，而你的新公司比较推崇devops，你要自己面对问题。

吹的那么高大上，一副拯救世界的感觉，但本系列的文章知识并不深，很多已经在大学里的操作系统见过了，虽然照读课本的叫兽并不能让你勾起丝毫兴趣。

如果本系列能够勾起你的些许兴趣，就算目的达到了。本来是想要聊仔细点，但由于时间有限，又不是写书，原理性的东西就不多说了。

内容

文章将会尝试单纯的Cpu、Mem、Net、Disk、IO问题排查，然后组合各种元素，解决一些棘手问题，就是一些常用命令的组合。当然我们是java系的，所以会多一些java方面的讨论。如果你不了解行文风格，可以先读读：《Java堆外内存排查小结--小姐姐味道》



为什么Linux系统会出现这样那样的问题呢？主要的原因就是计算机的各个部件的速度不均衡。Cpu在等cache line，cache 在等内存，内存在等设备。就像在连续17公里高速下坡路口设个收费站一样，一不小心就车毁人亡。

设备五花八门，通常我们接触的设备，就是硬盘和网卡。整个业务系统和操作系统充斥着各种各样的缓冲区，CPU要通过中断负责他们之间的协调。这样，会有很多地方会发生bottleneck。

监控值

排查问题也是有过程的。通常，关注一个硬件资源，比如CPU，我们关注以下基本要素：
1) 利用率 一般是瞬时值，属于采样范围，用来判断有没有峰值。比如cpu utilization
2) 饱和度 一般指资源已完全使用，新请求在特定queue里排队。比如cpu load过高
3) 错误信息 硬件或者驱动错误，比如 `dmesg` 命令显示的OOM
4) 联想信息 对引起的原因进行猜测，并用更多的工具验证猜想。比如系统响应慢猜测大量用到了swap

原因

监控值只是一种表象，具体引起的原因才是重点。我们通常希望纯粹的资源限制所引起的故障，这种问题都比较好定位。大多数情况下都没那么幸运，所以广度上的信息共享能帮助很多。过程如下：

1) 信息收集 问题起始时间，上下文

2) 改动集合 问题发生前所有变更列表
3) 问题抽象 将描述抽象成具体的资源问题
4) 问题排查 将信息整理完毕，就可以进行真正的荒野之旅了

测试

本测试用来决定你是不是本文目标受众，如果无法回答以下问题，建议先看一点基本的Linux知识，这将会节省你的时间，因为文章不会对此提太多。

io wait 是什么意思?

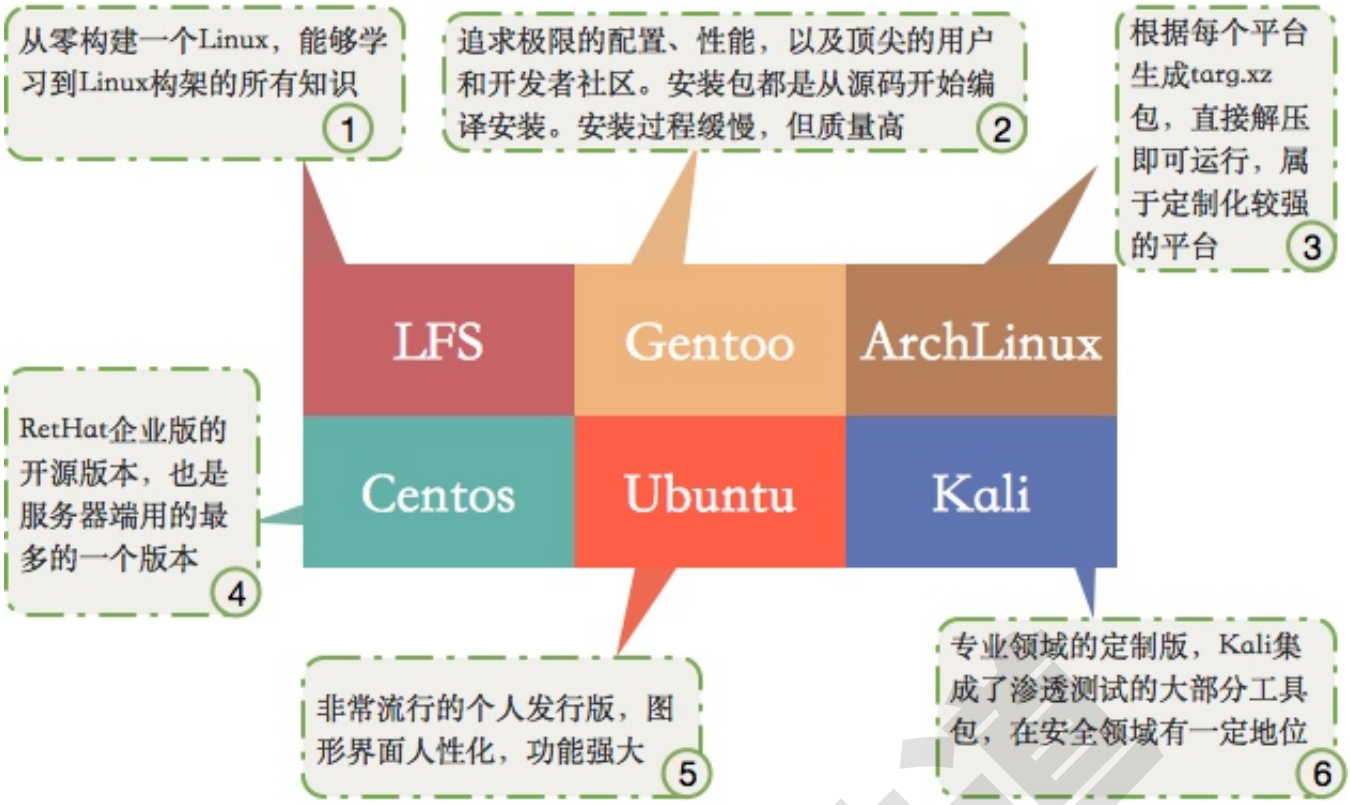
swap是什么分区，怎么关闭?

/tmp目录有什么特殊性?

管道是什么东东?

Linux发行版

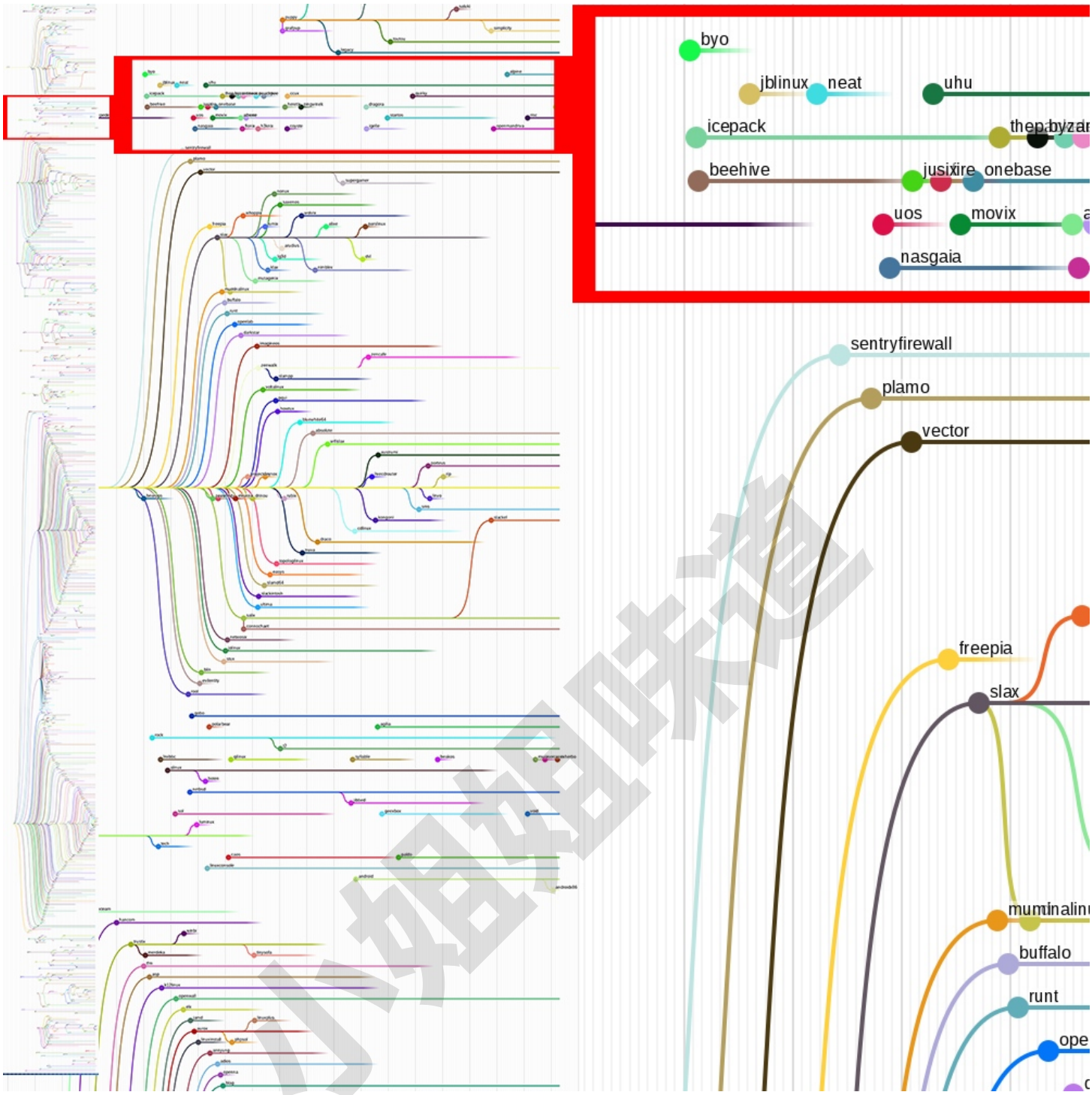
接下来热热身，瞧瞧Linux有什么发行版。



我这里挑选了6个代表性的版本，版本聚焦的功能向专业化和个性化发展。其中，Centos作为最常见的服务器版本，占据了大量的市场份额；Ubuntu在GUI和易用性上赢得了桌面用户；Kali代表了向专业化发展的一个分支。

个人使用时间最长的是archlinux，尤其喜欢它的滚动升级功能。但由于Centos在服务器端的市场份额实在太大，我们以下的讨论都基于Centos。

据不完全统计，已经有上千个linux版本，见下图（高清大图见 https://distrowatch.com/images/other/distro-family-tree.png ）。你来告诉我，红旗、麒麟的位置在哪里。



将常用脚本加入到PATH中

有些命令组合不好记，频繁输入也觉得烦，可以将这些过程整理成脚本，扔到path中。

还记得第一次安装jdk，添加的环境变量么？Linux和它类似，不过它有多种`shell`。

通常我们用的叫`bash`，平常说的shell脚本就是bash脚本。但也有很多其他好用的shell，比如csh、ksh、zsh等。

查看/etc/shells文件看一下你安装过的shell


```
[root@localhost ~]$ cat /etc/shells
/bin/sh
/bin/bash
/bin/zsh
/sbin/nologin
/bin/dash
```

在个人领域，zsh配合oh-my-zsh（推荐）达到最佳，但服务器一般不会去改你的shell，通过一个环境变量，能够看到你当前所使用的shell终端。

```
[root@localhost ~]$ echo $SHELL
/bin/bash
```

针对于bash，我们的配置就在用户目录下的`.bashrc`文件中。

在用户目录下创建`.bin`目录

```
mkdir ~/.bin
```

将目录加入到环境变量PATH中

```
echo "export PATH=\$PATH:~/.bin/" >> ~/.bashrc
```

在.bin创建一个文件xjj，内容为
echo "pleasant taste"

```
cat > ~/.bin/xjj <<EOF
echo "pleasant taste"
EOF
```

给xjj增加可执行权限

```
chmod a+x ~/.bin/xjj
```

这样，使用你的用户，在任何地方，都可以执行xjj了

```
[root@localhost ~]$ xjj
pleasant taste
```

真是令人愉悦的味道~

> 作者简介：**小姐姐味道** (xjjdog)，一个不允许程序员走弯路的公众号。聚焦基础架构和Linux。十年架构，日百亿流量，与你探讨高并发世界，给你不一样的味道。我的个人微信xjjdog0，欢迎添加好友，进一步交流。

近期热门文章

《必看！java后端，亮剑诛仙》

后端技术索引，中肯火爆

《Linux上，最常用的一批命令解析（10年精选）》
CSDN发布首日，1k赞。点赞率1/8。

《这次要是讲不明白Spring Cloud核心组件，那我就白编这故事了》
用故事讲解核心组件，包你满意

《Linux生产环境上，最常用的一套“Sed“技巧》
最常用系列Sed篇，简单易懂。Vim篇更加易懂。

Linux之《荒岛余生》（二）CPU篇

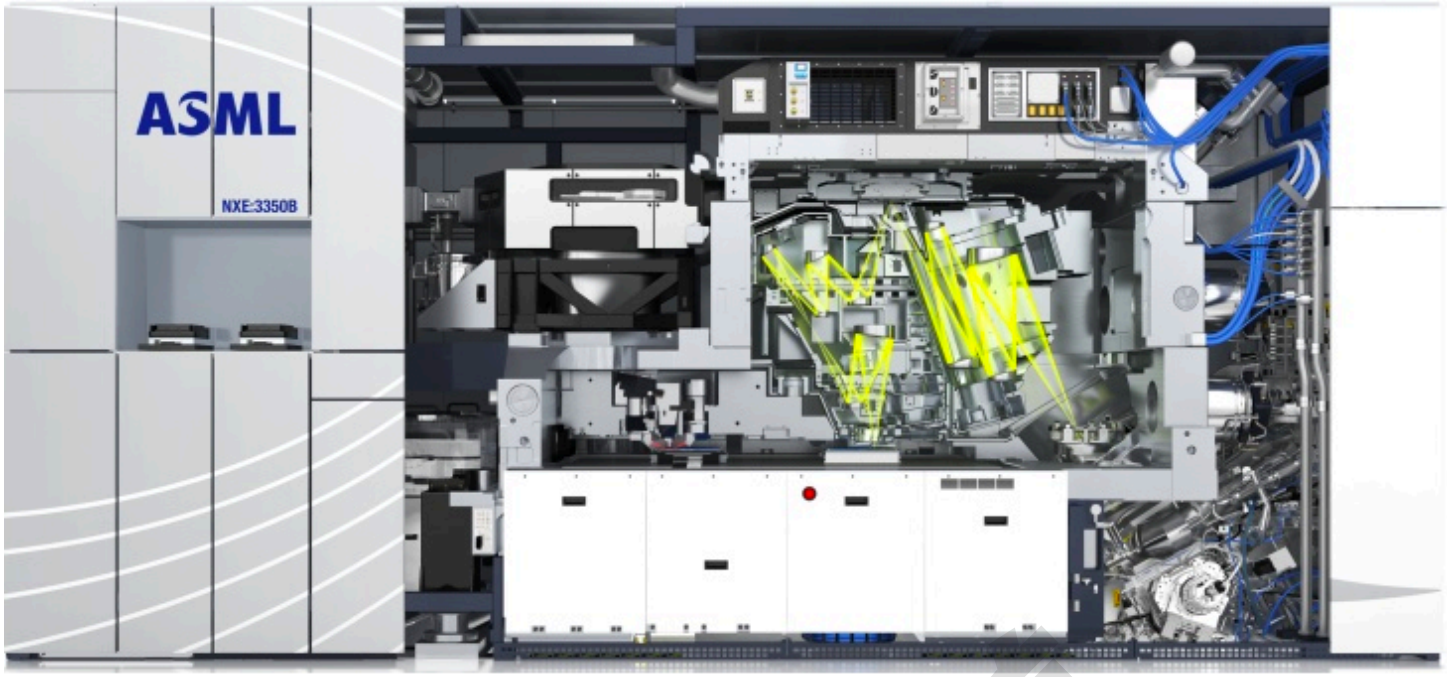
> 原创：小姐姐味道（微信公众号ID：xjjdog），欢迎分享，转载请保留出处。

温馨提示，动图已压缩，流量党放心查看。CPU方面内容不多，我们顺便学点命令。本篇是《荒岛余生》系列第二篇，垂直观测CPU。其余参见：

如何做一个CPU

cpu是芯片的一种，我们以汉芯为例，看一下制作七步曲。
❶ 提纯精度11个9的硅片（99.999999999%）
❷ 生成晶圆
❸ 使用光刻机加工晶圆
❹ 使用刻蚀机沟槽
❺ 完成P型半导体制作
❻ 使用200号的粗砂纸抹掉原标志
❼ 涂上新标志
bingo，完工!

虽然CPU很小，但生产它的设备可不简单。如下图，就是一台重十几吨，占地上百平米，全世界都当宝贝的光刻机!



你我就这样饱受科技的恩泽，有时间探讨在中央处理器上发生的故事了。

找到占用CPU最高的线程

接下来看一个实际的例子。公司有点穷，所以机器上混合部署了多个java应用，突然有一天，CPU炸了，我们要找到是谁引起的。这个谁不是进程，而是线程，离真相最近的那个。

传统做法

通常的做法是：

❶ 在命令行输入 `top`，然后 `shift+p` 查看占用CPU最高的进程，记下进程号

❷ 在命令行输入 `top -Hp 进程号`，查看占用CPU最高的线程

❸ 使用 `printf 0x%x 线程号`，得到其16进制线程号

❹ 使用 `jstack 进程号` 得到java执行栈，然后 `grep` 16进制找到相应的信息

录个屏先

```
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x00000007866f0890> (a org.apache.tomcat.util.net.JIoEndpoint$Worker)
        at java.lang.Object.wait(Object.java:503)
        at org.apache.tomcat.util.net.JIoEndpoint$Worker.await(JIoEndpoint.java:472)
        - locked <0x00000007866f0890> (a org.apache.tomcat.util.net.JIoEndpoint$Worker)
        at org.apache.tomcat.util.net.JIoEndpoint$Worker.run(JIoEndpoint.java:498)
        at java.lang.Thread.run(Thread.java:745)

"http-8082-31" daemon prio=10 tid=0x00007f9cf4054800 nid=0x1032 in Object.wait() [
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x00000007866f0cb8> (a org.apache.tomcat.util.net.JIoEndpoi
        at java.lang.Object.wait(Object.java:503)
        at org.apache.tomcat.util.net.JIoEndpoint$Worker.await(JIoEndpoint.java:47
        - locked <0x00000007866f0cb8> (a org.apache.tomcat.util.net.JIoEndpoint$Wo
        at org.apache.tomcat.util.net.JIoEndpoint$Worker.run(JIoEndpoint.java:498)
        at java.lang.Thread.run(Thread.java:745)
```

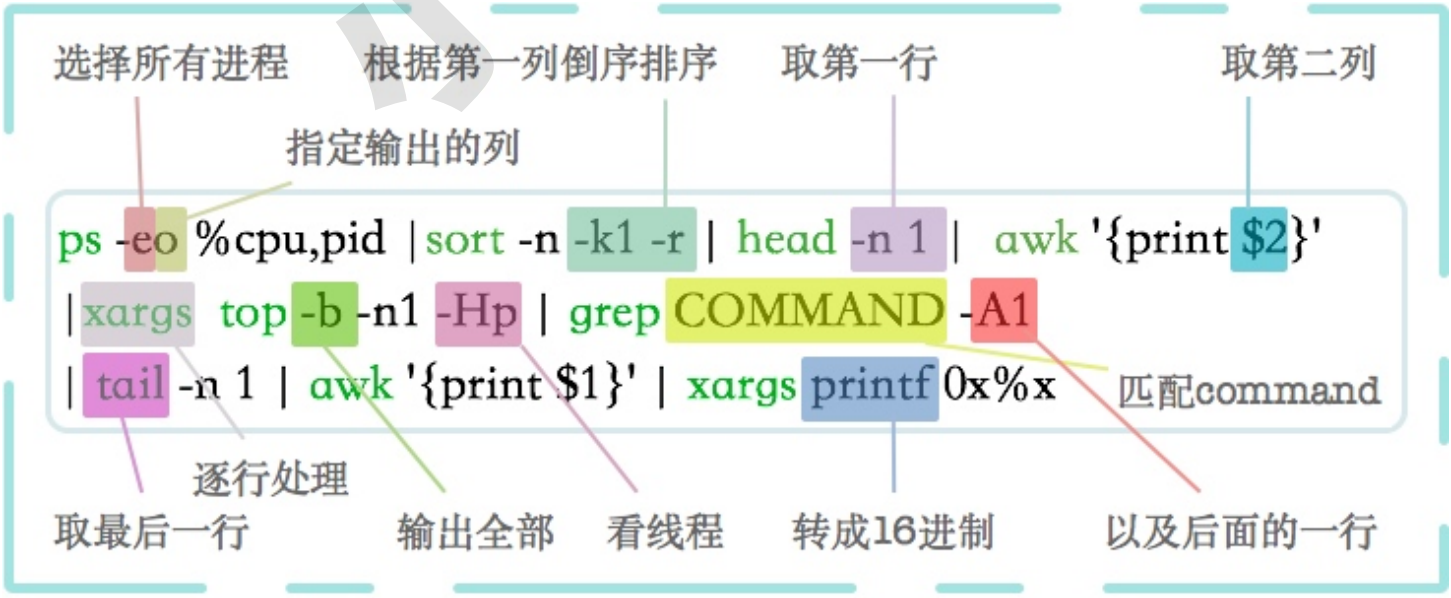
小姐姐味道

拔萝卜带泥

但我想通过另外一种方式来实现这个功能（最多样化），顺便学几个其他常用的命令。

```
ps -eo %cpu,pid |sort -n -k1 -r | head -n 1 |  awk '{print $2}' |xargs  top -b -n1 -Hp | grep COMMAND -A1 | tail -n 1 | awk '{print $1}' | xargs printf 0x%x
```

这一行Shell的意思是，找到使用CPU最高的进程之使用CPU最高的线程的16进制号。
这么长的命令，是不是晕了？别怕，我们一点点来。通常情况下，练习熟练了命令中出现的这几个，就能够应对50%的常用工作了。Come on，上图。



命令拆解 小姐姐味道

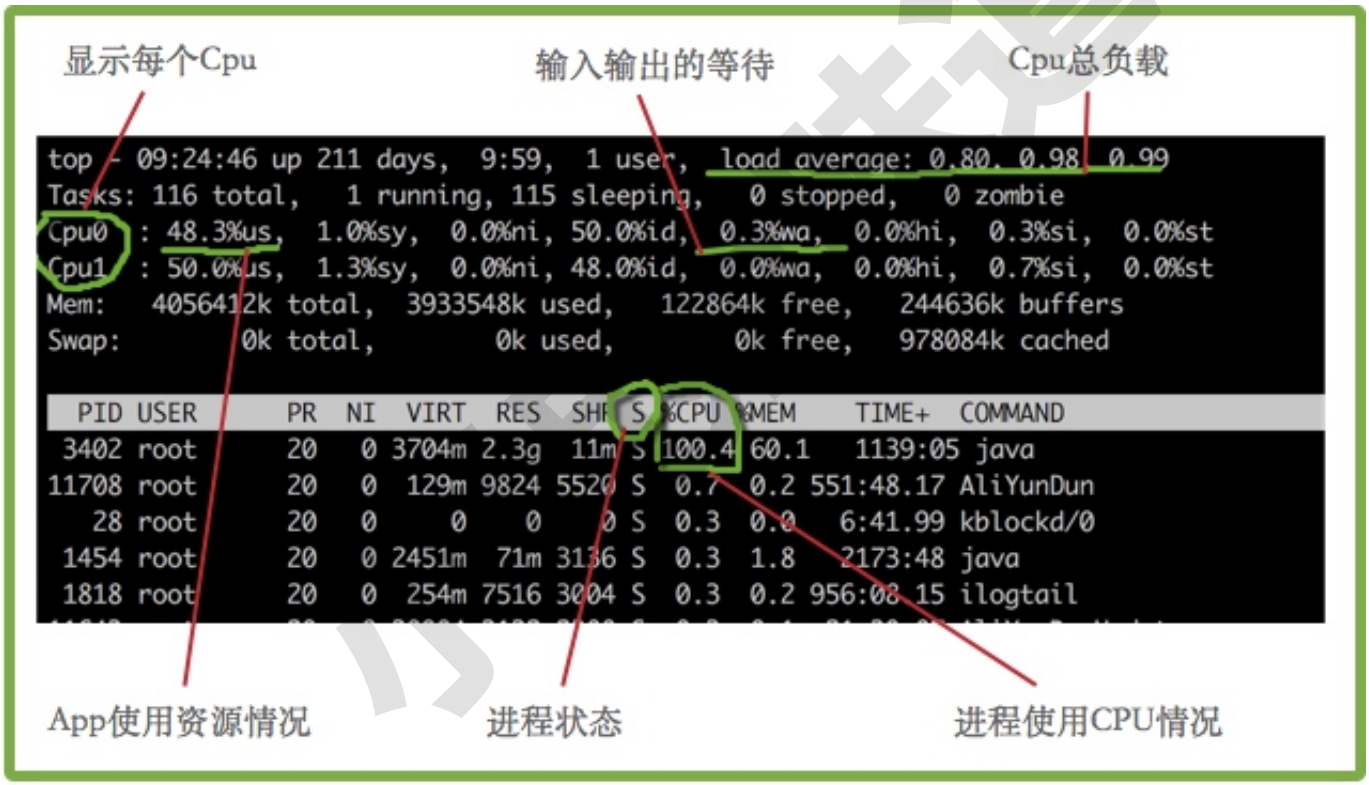
接下来，试着写一下脚本吧。

有哪些查看CPU的命令

top

其实从上面命令就可以瞧出来，top和ps的命令是互通的，只不过表现形式不同，我们直接拿top来说。按数字1就可以显示每核CPU的使用情况。基本上都是些单词的缩写，看几遍就忘不掉了。比如：

```
us ==> user CPU time
```

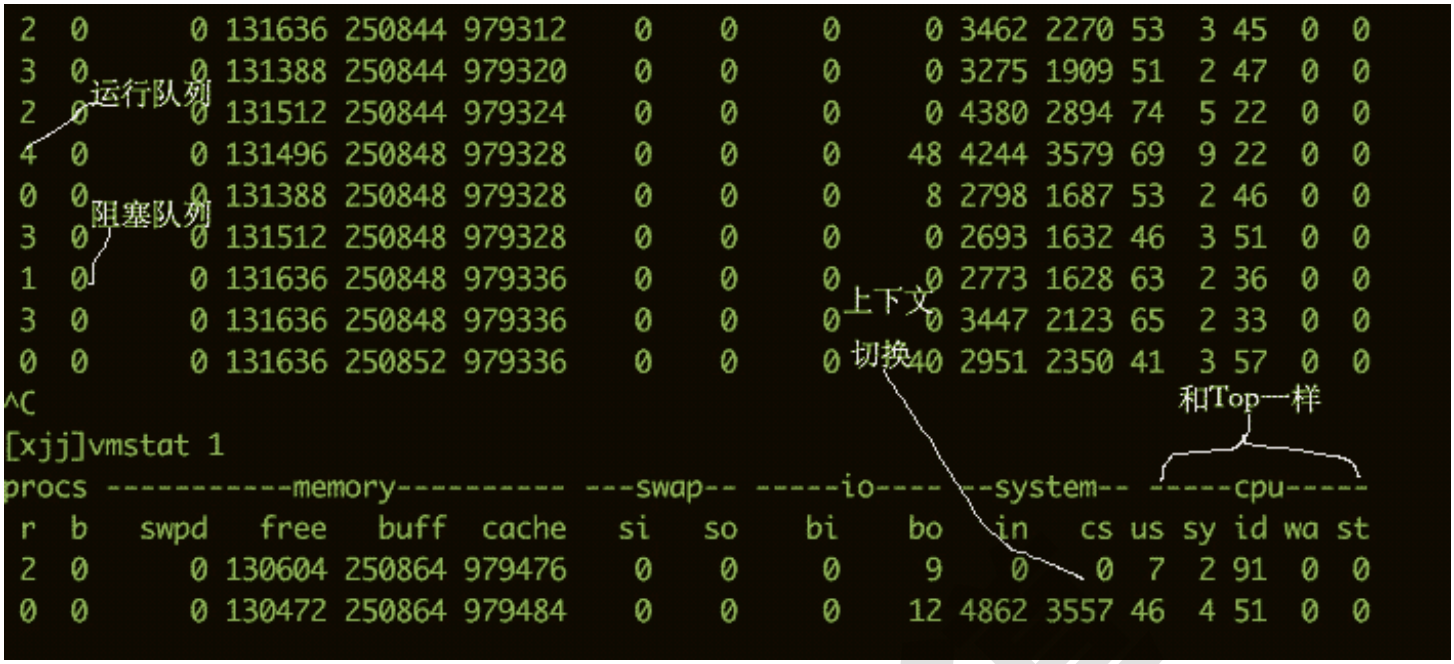


Top命令中的CPU 小姐姐味道

先记住这些判断准则，我们在示例中再聊：
❶ 如果load超过了cpu核数，则负载过高
❷ 如果wa过高，可初步判断I/O有问题
❸ sy,si,hi,st，任何一个超过5%，都有问题
❹ 进程状态长时处于D、Z、T状态，提高注意度
❺ cpu不均衡，判断亲和性和优先级问题

vmstat

vmstat 以另一种形式来展示一些信息。如图：



除了关注类似top的一些指标，还有：

❶ `b` 置于等待队列（等待资源、等待输入/输出）的内核线程数目。数字过大则cpu太忙。

❷ `cs` 如果频繁的进行上下文切换，则考虑是否是线程数开的过多

❸ `si` / `so` 显示了交换分区的现状，有时候会造成cpu问题，一并关注

sar

是目前Linux上最为全面的系统性能分析工具之一，但可能没有预装。在centos上使用以下命令即可安装。

```
yum install sysstat -y
```

sar主要的好处是可以看到历史，显示友好，可以对结果进行二次处理。sar还有图形化工具，执行 `sar -A` 即可获得所有数据。

```
https://github.com/vlsi/ksar
```

针对于CPU方面，我们关注：

❶ sar -u 默认

❷ sar -P ALL 每颗cpu的使用状态信息

❸ sar -q cpu队列的长度，runq-sz>cpu count就表明有瓶颈了

❹ sar -w 每秒上下文交换

可以瞧见，关注的也就那几个点而已。

mpstat

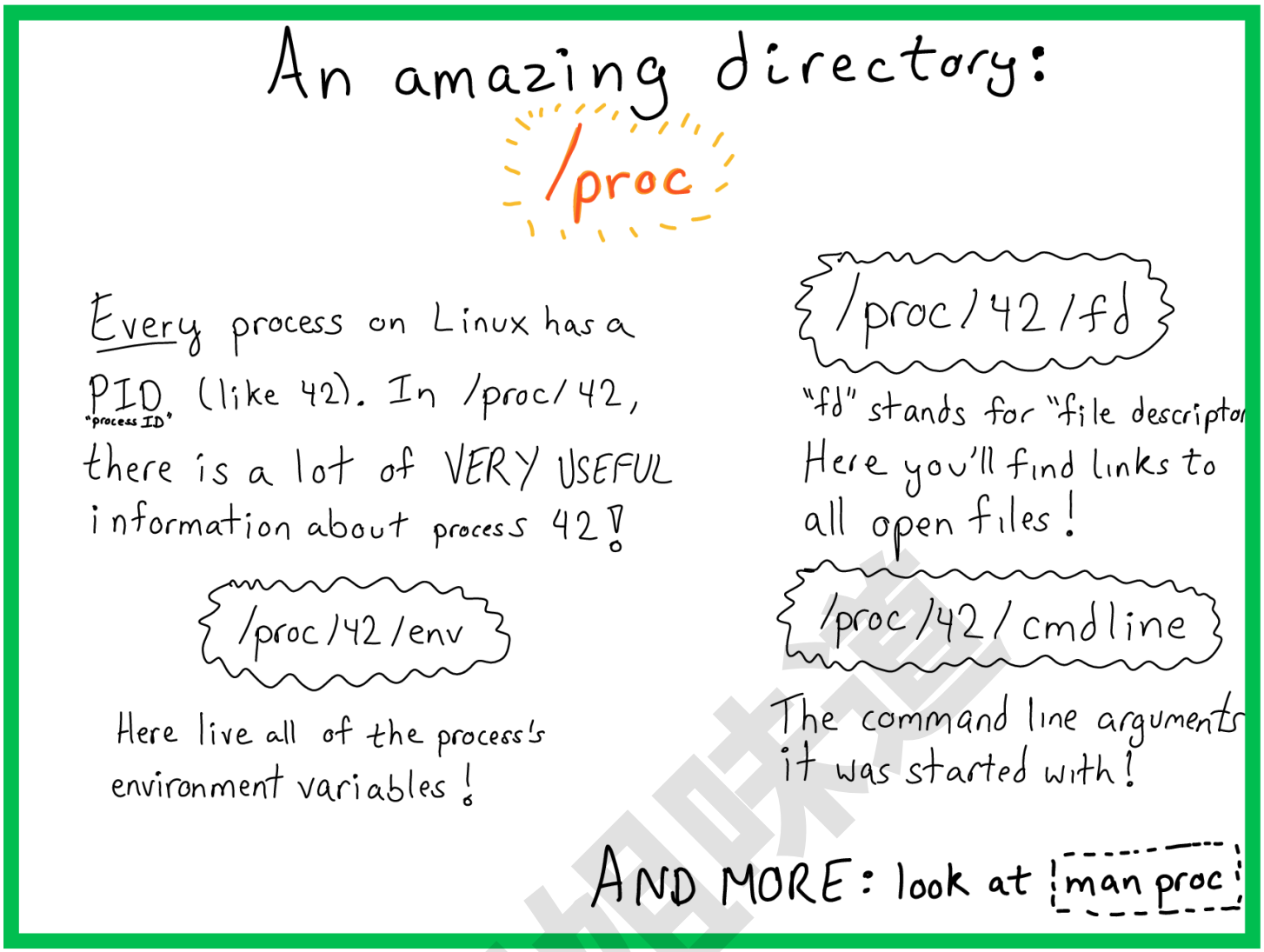
还有 `pidstat`，包括彩色的 `dstat`，功能都差不多， 用熟一个就ok了。

数据从何而来

那么数据从何而来?
`/proc` 目录是一个虚拟目录，存储的是当前内核的一系列特殊文件，你不仅能查看一些状态，甚至能修改一些值来改变系统的行为。

比如top的load （使用uptime命令得到同样的结果）。读取的就是
`/proc/loadavg` 文件
而每核cpu的信息，读取
`/proc/stat` 文件

这些命令，是对/proc目录中一系列信息的解析和友好的展示，这些值，Linux内核都算好了躺在那呢。



(图片来源网络)
创建这个目录的人真是天才!

几个例子

此文归属微信公众号「小姐姐味道」，转载注明出处。CPU过高是表象。除了系统确实负载已经到了极限，其他的，都是由其他原因引起的，比如I/O；比如设备。这些我们放在其他章节进行讨论。

GC引起的CPU过高

接着我们最开始的例子来。通过查看jstack找到相应的16进制进程，结果发现是GC线程。

```
"VM Thread" prio=10 tid=0x00007f06d8089000 nid=0x58c7 runnable

"GC task thread#0 (ParallelGC)" prio=10 tid=0x00007f06d801b800 nid=0x58d7 run
nable
```


这种情况，一般都是JVM内存不够用了，疯狂GC，可能是socket/线程忘了关闭了，也可能是大对象没有回收。这种情况只能通过重启来解决了，记得重启之前，使用jmap dump一下堆栈哦。当然，你可能会得到jdk版本的问题。

st%占比过高

st过高一般是物理CPU资源不足所致，也就是只发生在虚拟机上。
如果你买的虚拟机st一直很高，那你的服务提供商可能在超卖，挤占你的资源。不信双11的时候看下你的虚拟机?

网卡导致单cpu过高

业务方几台kafka，cpu使用处于正常水平，才10%左右，但有一核cpu，负载特别的高，si奇高。

mpstat -I SUM -P ALL 查看cpu使用情况，cpu0的中断确实比较多。

```
20:15:18  CPU    intr/s
20:15:23  all  34234.20
20:15:23    0   9566.20
20:15:23    1      0.00
```

网卡需要cpu服务时，都会抛出一个中断，中断告诉cpu发生了什么事情，cpu就要停止目前的工作来处理这个中断。其实，默认所有的中断处理都集中在cpu0 上，导致服务器负载过高。cpu0成了瓶颈，而其他cpu却还闲着。
❶ 解决方式1：使用CPU亲和性功能，kafka略过网卡所使用的CPU
❷ 解决方式2: 更换网卡
❸ 通常修改的方式还是有些复杂了，比如，修改/proc/irq/{seq}/smp_affinity
我们可以直接安装 `irqbalance` ，然后执行就可以了。

```
yum install irqbalance -y
service irqbalance start
```

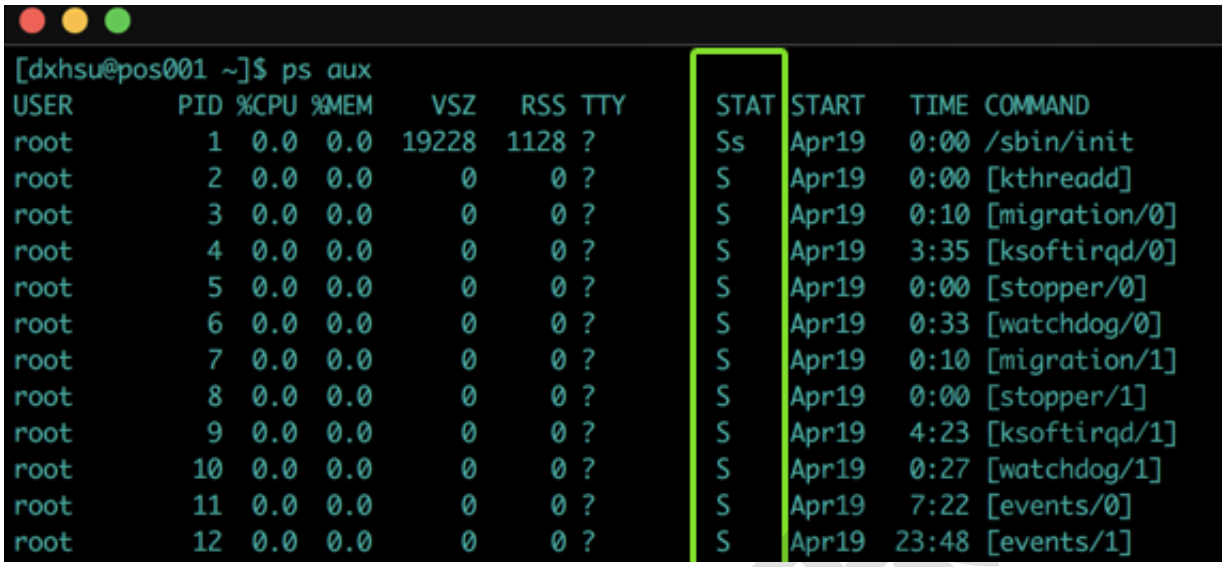
cpu使用率低，但负载高

cpu id%高，也就是空闲，比如90%。但 load average非常高，比如4核达到10。

分析：load average高，说明其任务已经排队，许多任务正在等待。出现此种情况，可能存在大量不可中断的进程。

使用top或者ps可以看到进程相应的状态。

```
ps aux
```



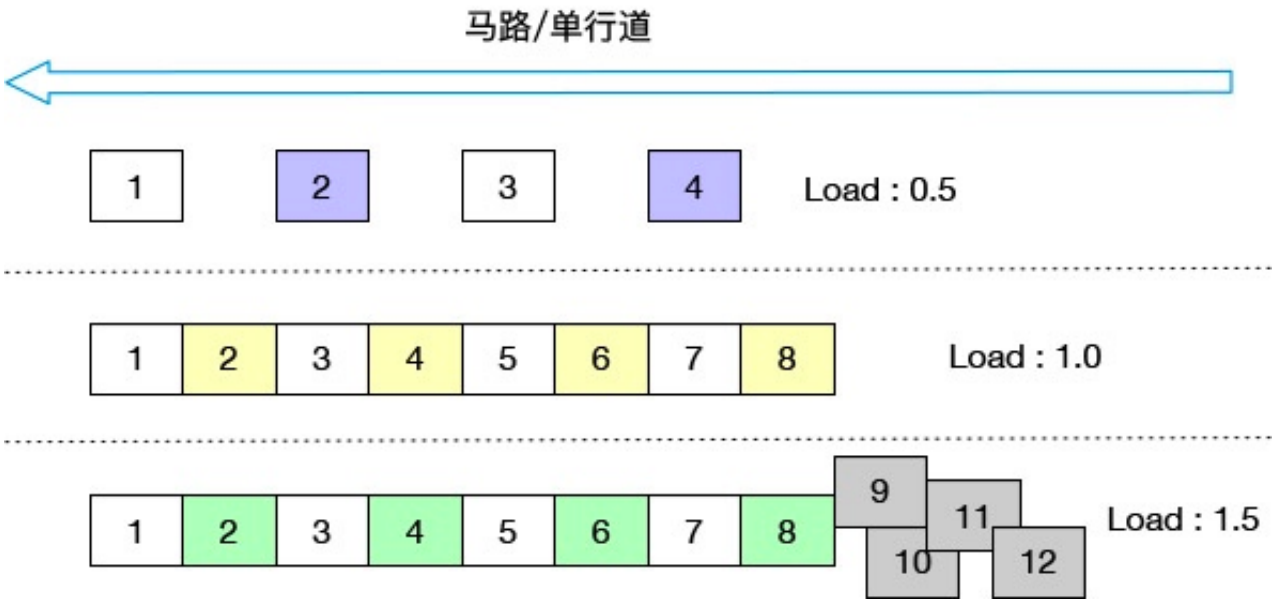
```
[dxhsu@pos001 ~]$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  19228  1128 ?        Ss   Apr19   0:00 /sbin/init
root         2  0.0  0.0      0     0 ?        S    Apr19   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Apr19   0:10 [migration/0]
root         4  0.0  0.0      0     0 ?        S    Apr19   3:35 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S    Apr19   0:00 [stopper/0]
root         6  0.0  0.0      0     0 ?        S    Apr19   0:33 [watchdog/0]
root         7  0.0  0.0      0     0 ?        S    Apr19   0:10 [migration/1]
root         8  0.0  0.0      0     0 ?        S    Apr19   0:00 [stopper/1]
root         9  0.0  0.0      0     0 ?        S    Apr19   4:23 [ksoftirqd/1]
root        10  0.0  0.0      0     0 ?        S    Apr19   0:27 [watchdog/1]
root        11  0.0  0.0      0     0 ?        S    Apr19   7:22 [events/0]
root        12  0.0  0.0      0     0 ?        S    Apr19  23:48 [events/1]
```

一种情况就是有大量进程处于D的状态，也就是不可中断的睡眠状态，所以很可能是硬件问题。详见《Linux进程状态(ps stat)之R、S、D、T、Z、X》

高频问题：load

load代表的是啥

说句白话，load代表的就是你目前系统进程的排队情况。



如图，以单核为例，将CPU资源抽象成一条单行马路。则会发生三种情况：
❶ 马路上的车只有4辆，车辆畅通无阻，load大约是0.5
❷ 马路上的车有8辆，正好能首尾相接安全通过，此时load大约为1
❸ 马路上的车有12辆，除了在马路上的8辆车，还有4辆交集的等在外面，也就是超出容量了，需要排队。此时load大约为1.5

load为1代表的是啥

针对这个问题，误解还是比较多的。很多同学认为，load达到1，系统就到了瓶颈，这不完全正确。
load的值和cpu核数息息相关：
❶ 单核的cpu达到100%，load约1
❷ 双核的cpu都达到100%，load约2
❸ 四核的cpu都达到100%，load约为4

所以，对于一个load到了10，却是16核的机器，你的系统还远没有达到负载极限。

结尾

本篇实际的排查过程较少，因为cpu问题一般都伴随着其他问题。但文中出现的这些命令可不简单，尤其是它们丰富的参数。这些参数，执行一下man，就可以一睹芳容了。比如：

```
man top
```

当然，也可以这样～

```
[dxhsu@pos001 ~]$ which woman love man
/usr/bin/which: no woman in (/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/us
in)
/usr/bin/which: no love in (/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr
n)
/usr/bin/man
[dxhsu@pos001 ~]$
```

no woman、 no love，果然是一个只有男人的世界！

> 作者简介：**小姐姐味道** (xjjdog)，一个不允许程序员走弯路的公众号。聚焦基础架构和Linux。十年架构，日百亿流量，与你探讨高并发世界，给你不一样的味道。我的个人微信xjjdog0，欢迎添加好友，进一步交流。

近期热门文章

《必看！java后端，亮剑诛仙》

后端技术索引，中肯火爆

《Linux上，最常用的一批命令解析（10年精选）》
CSDN发布首日，1k赞。点赞率1/8。

《这次要是讲不明白Spring Cloud核心组件，那我就白编这故事了》
用故事讲解核心组件，包你满意

《Linux生产环境上，最常用的一套“Sed“技巧》
最常用系列Sed篇，简单易懂。Vim篇更加易懂。

Linux之《荒岛余生》（三）内存篇

> 原创：小姐姐味道（微信公众号ID：xjjdog），欢迎分享，转载请保留出处。

内存问题，脑瓜疼脑瓜疼。脑瓜疼的意思，就是脑袋运算空间太小，撑的疼。本篇是《荒岛余生》系列第三篇，让人脑瓜疼的内存篇。其余参见：

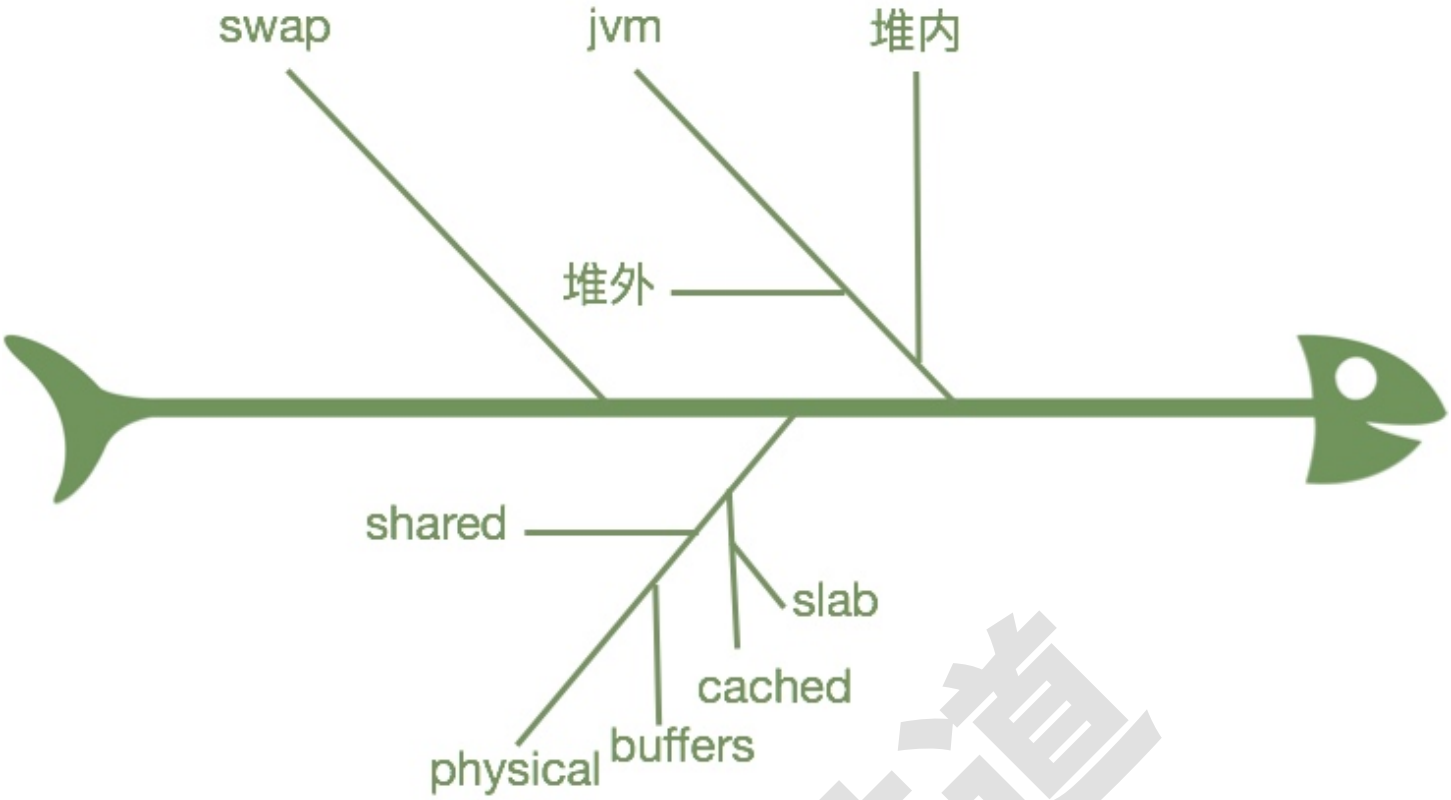
小公司请求量小，但喜欢滥用内存，开一堆线程，大把大把往jvm塞对象，最终问题是内存溢出。

大公司并发大，但喜欢强调HA，所以通常保留swap，最终问题是服务卡顿。

而喜欢用全局集合变量的某些同仁，把java代码当c写，对象塞进去但忘了销毁，最终问题是内存泄漏。

如何避免?
合理参数、优雅代码、禁用swap，三管齐下，
trouble shooter。

从一个故事开始



老王的疑问

一个阳光明媚的下午，一条报警短信弹了出来。老王微微一笑，是cpu问题，idle瞬时值，大概是某批请求比较大引起的峰值问题。老王每天都会收到这样的短信，这样的一个小峰值，在数千台服务器中，不过是沧海一栗，继续喝茶就是了。

但，这次不一样。几分钟之后，几百个服务的超时报警铺天盖地到来。事后老王算了一下，大概千分之零点几的服务超时了，不过这已经很恐怖了。
事态升级，恐怕没时间喝茶了。

大面积报警，应该是全局问题，是网络卡顿？还是数据库抽风？老王挑了一台最近报警的服务器，轮流监控了各种状态，总结如下：

- cpu偶尔有瞬时峰值，但load非常正常
- 内存虽然free不多了，但cached还有不少
- 网络各种ping，基本正常
- 磁盘I/O一般，毕竟是服务计算节点
- 数据库连接池稳定，应该不是db抽风
- swap用了不少，但好像每台机器都用了，没啥大不了

全局性的东西不太多，网关、LVS、注册中心、DB、MQ，好像都没问题。老王开始脑瓜疼了。

让老王休息一下，我们把镜头转向小王。

小王的操作

小王不是老王的儿子，他是老王的徒弟。徒弟一思考，导师就发笑。这次小王用的是vim，想查找一个Exception，他打开了一个8GB的日志文件，然后乐呵呵的在那等着加载。然后，服务器就死了。

答案

这里直接给出答案，原因等读完本文自然会了解。

老王的问题最终定位到是由于某个运维工程师使用ansible批量执行了一句命令

```
find / | grep "x"
```

他是想找一个叫做x的文件，看看在哪台服务器上。结果，这些老服务器由于文件太多，扫描后这些文件信息都缓存到了slab区。而服务器开了swap，操作系统发现物理内存占满后，并没有立即释放cache，导致每次GC，都和硬盘打一次交道。然后，所有服务不间歇卡顿了...

最终，只能先关闭swap分区，然后强制内核释放cache，然后再开启swap。当然这个过程也不会顺利，因为开、关swap，同样会引起大量I/O交换，所以不能批量去执行。这几千台机器，是要忙活一阵喽。

小王的问题就简单多了。他使用vim打开大文件，所有文件的内容都会先加载到内存。结果，内存占满、接着swap也满了，然后oom-killer杀死了服务进程，给一头雾水的小王留下了个莫名其妙。

排查内存的一些命令

内存分两部分，物理内存和swap。物理内存问题主要是内存泄漏，而swap的问题主要是用了swap～，我们先上一点命令。

(#1) 物理内存


```
#根据使用量排序查看RES
top -> shift + m
#查看进程使用的物理内存
ps -p 75 -o rss,vsz
#显示内存的使用情况
free -h
#使用sar查看内存信息
sar -r
#显示内存每个区的详情
cat /proc/meminfo
#查看slab区使用情况
slabtop
```

通常，通过查看物理内存的占用，你发现不了多少问题，顶多发现那个进程占用内存高（比如vim等旁路应用）。meminfo和slabtop对系统的全局判断帮助很大，但掌握这两点坡度陡峭。

(#2) swap

```
#查看si,so是否异常
vmstat 1
#使用sar查看swap
sar -W
#禁用swap
swapoff
#查询swap优先级
sysctl -q vm.swappiness
#设置swap优先级
sysctl vm.swappiness=10
```

建议关注非0 swap的所有问题，即使你用了ssd。swap用的多，通常伴随着I/O升高，服务卡顿。swap一点都不好玩，不信搜一下《swap罪与罚》这篇文章看下，千万不要更晕哦。

(#3) jvm

```
# 查看系统级别的故障和问题
dmesg
# 统计实例最多的类前十位
jmap -histo pid | sort -n -r -k 2 | head -10
# 统计容量前十的类
jmap -histo pid | sort -n -r -k 3 | head -10
```


以上命令是看堆内的，能够找到一些滥用集合的问题。堆外内存，依然推荐《Java堆外内存排查小结》

(#4) 其他

```
# 释放内存
echo 3 > /proc/sys/vm/drop_caches
#查看进程物理内存分布
pmap -x 75  | sort -n -k3
#dump内存内容
gdb --batch --pid 75 -ex "dump memory a.dump 0x7f2bceda1000 0x7f2bcef2b000"
```

内存模型

二王的问题表象都是CPU问题，CPU都间歇性的增高，那是因为Linux的内存管理机制引起的。你去监控Linux的内存使用率，大概率是没什么用的。因为经过一段时间，剩余的内存都会被各种缓存迅速占满。一个比较典型的例子是ElasticSearch，分一半内存给JVM，剩下的一半会迅速被Lucene索引占满。

如果你的App进程启动后，**经过两层缓冲后还不能落地，迎接它的，将会是oom killer**。

接下来的知识有些烧脑，但有些名词，可能是你已经听过多次的了。

操作系统视角

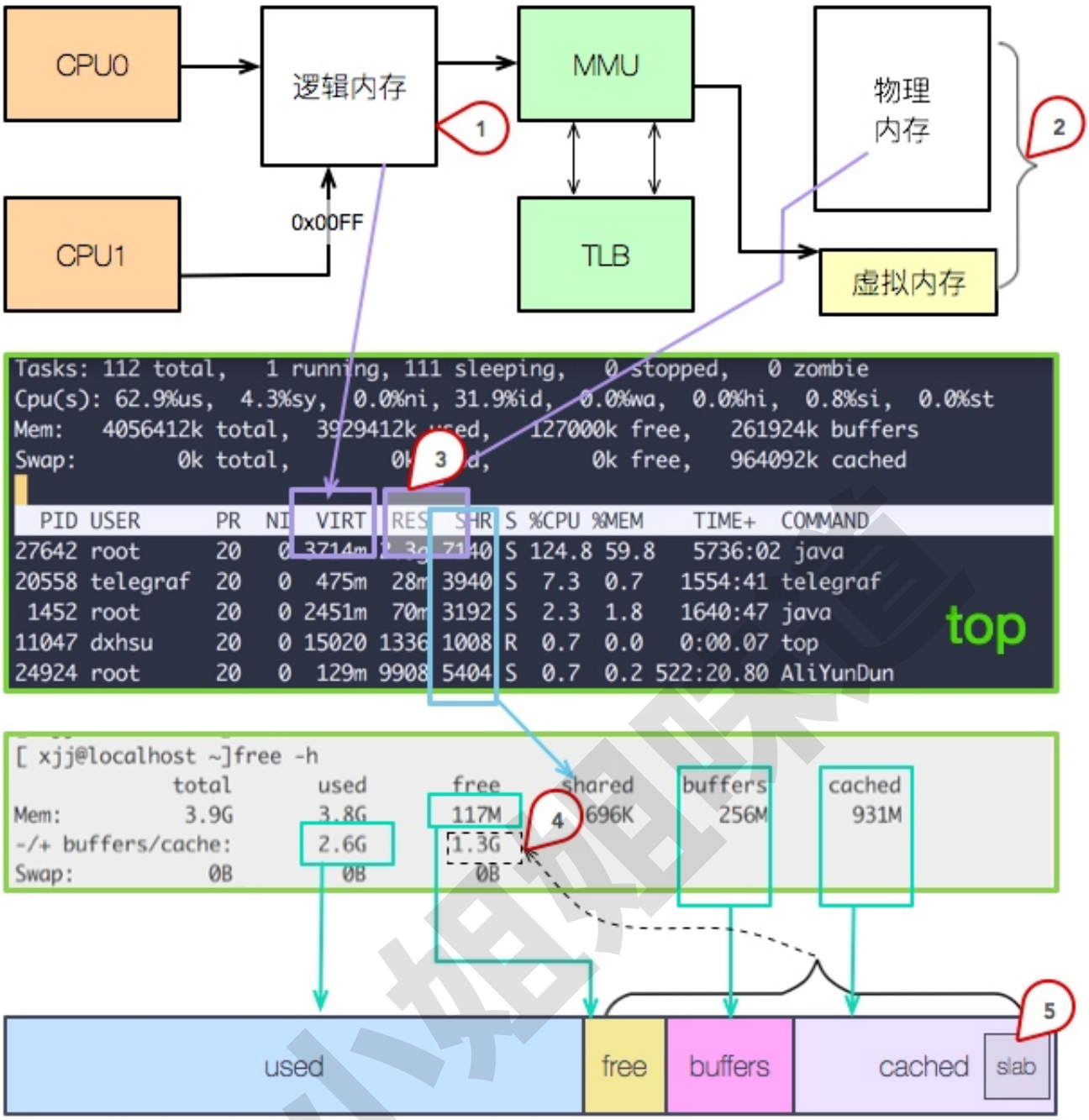


图2 操作系统视图 小姐姐味道《荒岛余生》

我们来解释一下上图，第一部分是逻辑内存和物理内存的关系；第二部分是top命令展示的一个结果，详细的列出了每一个进程的内存使用情况；第三部分是free命令展示的结果，它的关系比较乱，所以给加上了箭头来作说明。

- 学过计算机组成结构的都知道，程序编译后的地址是逻辑内存，需要经过翻译才能映射到物理内存。这个管翻译的硬件，就叫`MMU`；`TLB`就是存放这些映射的小缓存。内存特别大的时候，会涉及到`hugepage`，**在某些时候，是进行性能优化的杀手锏，比如优化redis (THP，注意理解透彻前不要妄动)**

- 物理内存的可用空间是有限的，所以逻辑内存映射一部分地址到硬盘上，以便获取更大的物

理内存地址，这就是`swap`分区。**swap是很多性能场景的万恶之源，建议禁用**

- 像`top`展示的字段，`RES`才是真正的物理内存占用（不包括swap，ps命令里叫`RSS`）。在java中，代表了堆内+堆外内存的总和。而VIRT、SHR等，几乎没有判断价值(某些场景除外)
- 系统的可用内存，包括：`free` + `buffers` + `cached`，因为后两者可以自动释放。但不要迷信，有很大一部分，你是释放不了的
- slab区，是内核的缓存文件句柄等信息等的特殊区域，slabtop命令可以看到具体使用

更详细的，从`/proc/meminfo`文件中可以看到具体的逻辑内存块的大小。有多达40项的内存信息，这些信息都可以通过/proc一些文件的遍历获取，本文只挑重点说明。

```
[xjj@localhost ~]$ cat /proc/meminfo
MemTotal:        3881692 kB
MemFree:          249248 kB
MemAvailable:    1510048 kB
Buffers:           92384 kB
Cached:          1340716 kB
40+ more ...
```

oom-killer

以下问题已经不止一个小伙伴问了：**我的java进程没了，什么都没留下，就像个屁一样蒸发不见了**

why？是因为对象太多了么？

执行`dmesg`命令，大概率会看到你的进程崩溃信息躺尸在那里。

```
[Thu Sep  6 18:34:39 2018] [20572]     0 20572    28848       99      14        0             0 bash
[Thu Sep  6 18:34:39 2018] [20954]     0 20954   877124   137223     348        0             0 java
[Thu Sep  6 18:34:39 2018] Out of memory: Kill process 27218 (java) score 345 or sacrifice child
[Thu Sep  6 18:34:39 2018] Killed process 27218 (java) total-vm:3651512kB, anon-rss:1381060kB, file-rss:0kB, shmem
-rss:0kB
[Thu Sep  6 21:57:05 2018] AliYunDun invoked oom-killer: gfp_mask=0x201da, order=0, oom_score_adj=0
[Thu Sep  6 21:57:05 2018] AliYunDun cpuset=/ mems_allowed=0
[Thu Sep  6 21:57:05 2018] CPU: 0 PID: 24654 Comm: AliYunDun Not tainted 3.10.0-693.2.2.el7.x86_64 #1
[Thu Sep  6 21:57:05 2018] Hardware name: Alibaba Cloud Alibaba Cloud ECS, BIOS rel-1.7.5-0-ge51488c-20140602_1646
12-nilsson.home.kraxel.org 04/01/2014
[Thu Sep  6 21:57:05 2018]  ffff8800baea1fa0 00000000efd6cb79 ffff880136afb8f8 ffffffff816a3db1
[Thu Sep  6 21:57:05 2018]  ffff880136afb988 ffffffff8169f1a6 ffffffff810e939c ffff8800bae36bb0
[Thu Sep  6 21:57:05 2018]  0000000000000001 ffff880136afb930 0000000000000202 ffff880136afb978
[Thu Sep  6 21:57:05 2018] Call Trace:
[Thu Sep  6 21:57:05 2018]  [<ffffffff816a3db1>] dump_stack+0x19/0x1b
[Thu Sep  6 21:57:05 2018]  [<ffffffff8169f1a6>] dump_header+0x90/0x229
[Thu Sep  6 21:57:05 2018]  [<ffffffff810e939c>] ? ktime_get_ts64+0x4c/0xf0
[Thu Sep  6 21:57:05 2018]  [<ffffffff8113d36f>] ? delayacct_end+0x8f/0xb0
```

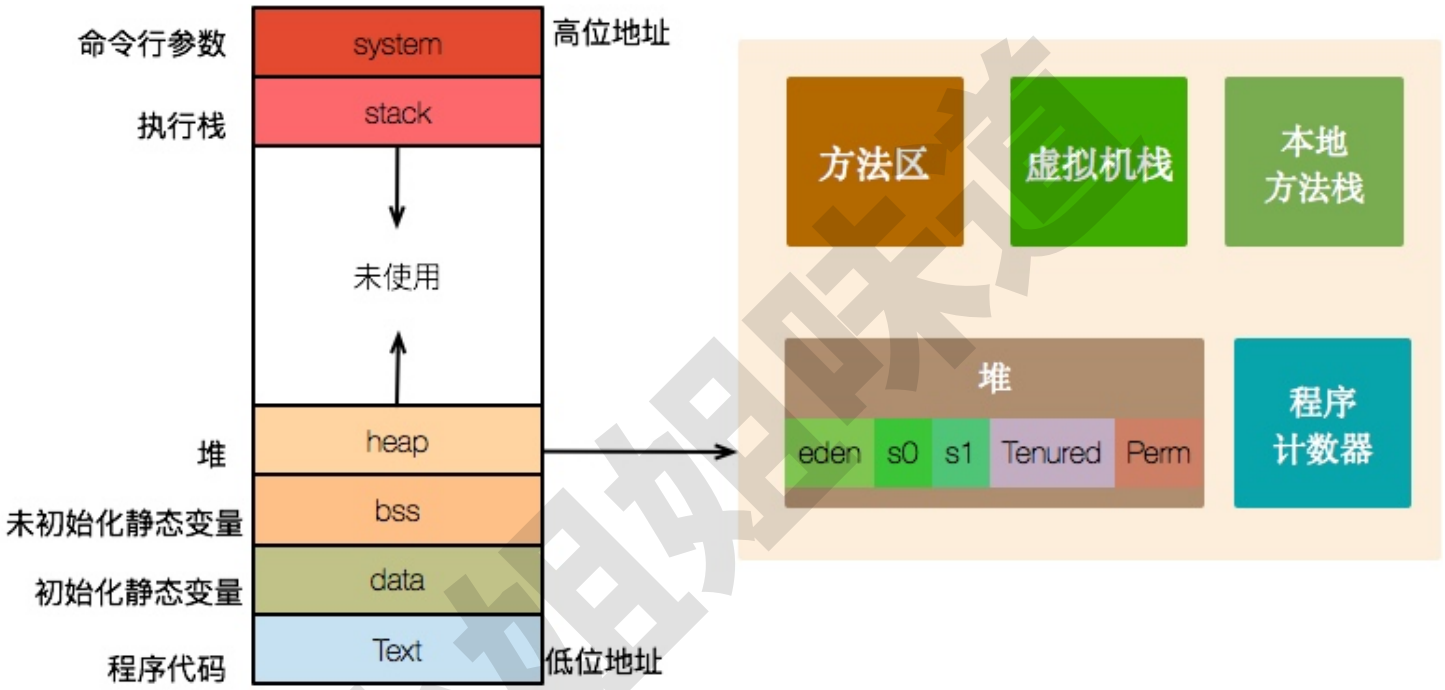
为了能看到发生的时间，我们习惯性加上参数T

```
dmesg -T
```

由于linux系统采用的是虚拟内存，进程的`代码`，`库`，`堆`和`栈`的使用都会消耗内存，但是申请出来的内存，只要没真正access过，是不算的，因为没有真正为之分配物理页面。

第一层防护墙就是swap；当swap也用的差不多了，会尝试释放cache；当这两者资源都耗尽，杀手就出现了。oom killer会在系统内存耗尽的情况下跳出来，选择性的干掉一些进程以求释放一点内存。2.4内核杀新进程；2.6杀用的最多的那个。所以，买内存吧。

这个oom和jvm的oom可不是一个概念。顺便，瞧一下我们的JVM堆在什么位置。



例子

jvm内存溢出排查

应用程序发布后，jvm持续增长。使用jstat命令，可以看到old区一直在增长。

```
jstat  -gcutil 28266 1000
```

在jvm参数中，加入`-XX:+HeapDumpOnOutOfMemoryError`，在jvm oom的时候，生成hprof快照。然后，使用Jprofile、VisualVM、Mat等打开dump文件进行分析。

你要是个急性子，可以使用jmap立马dump一份


```
jmap -heap:format=b pid
```

最终发现，有一个全局的Cache对象，不是guava的，也不是commons包的，是一个简单的ConcurrentHashMap，结果越积累越多，最终导致溢出。

溢出的情况也有多种区别，这里总结如下：

关键字	原因
Java.lang.OutOfMemoryError: Java heap space	堆内存不够了，或者存在内存溢出
java.lang.OutOfMemoryError: PermGen space	Perm区不够了，可能使用了大量动态加载的类，比如cglib
java.lang.OutOfMemoryError: Direct buffer memory	堆外内存、操作系统没内存了，比较严重的情况
java.lang.StackOverflowError	调用或者递归层次太深，修正即可
java.lang.OutOfMemoryError: unable to create new native thread	无法创建线程，操作系统内存没有了，一定要预留一部分给
java.lang.OutOfMemoryError: Out of swap space	同样没有内存资源了，swap都用光了

jvm程序内存问题，除了真正的内存泄漏，大多数都是由于太贪心引起的。一个4GB的内存，有同学就把jvm设置成了3840M，只给操作系统256M，不死才怪。

另外一个问题就是swap了，**当你的应用真正的高并发了，swap绝对能让你体验到它魔鬼性的一面：进程倒是死不了了，但GC时间长的无法忍受。**

我的ES性能低

业务方的ES集群宿主机是32GB的内存，随着数据量和访问量增加，决定对其进行扩容=>内存改成了64GB。

内存升级后，发现ES的性能没什么变化，某些时候，反而更低了。

通过查看配置，发现有两个问题引起。
一、64GB的机器分配给jvm的有60G，预留给文件缓存的只有4GB，造成了文件缓存和硬盘的频

繁交换，比较低效。
二、JVM大小超过了32GB，内存对象的指针无法启用压缩，造成了大量的内存浪费。由于ES的对象特别多，所以留给真正缓存对象内容的内存反而减少了。

解决方式：给jvm的内存30GB即可。

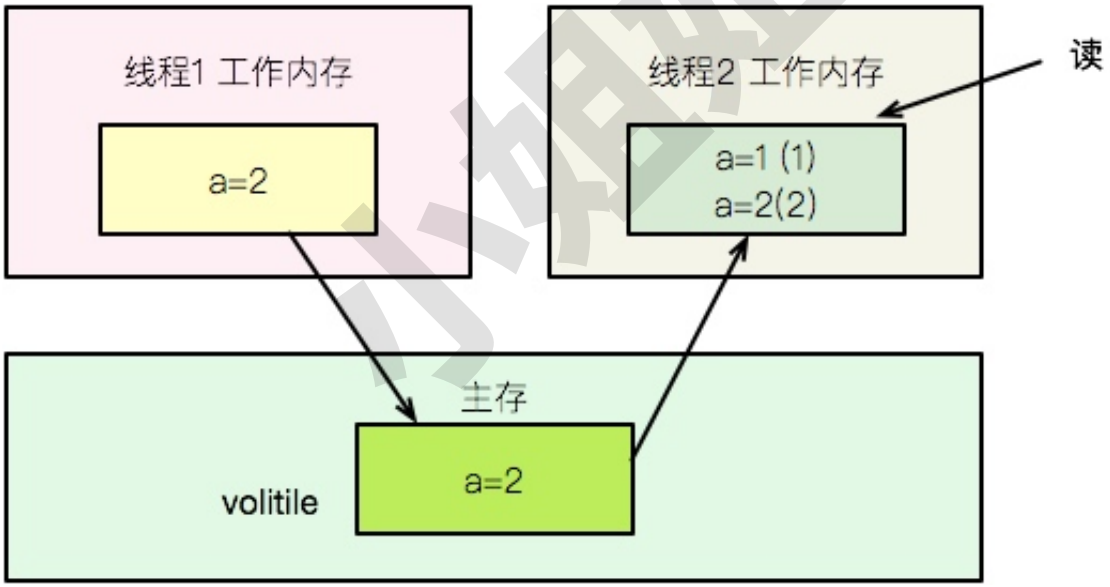
其他

基本上了解了内存模型，上手几次内存溢出排查，内存问题就算掌握了。但还有更多，这条知识系统可以深挖下去。

JMM

还是拿java来说。java中有一个经典的内存模型，一般面试到volitile关键字的时候，都会问到。其根本原因，就是由于线程引起的。

当两个线程同时访问一个变量的时候，就需要加所谓的锁了。由于锁有读写，所以java的同步方式非常多样。wait,notify、lock、cas、volitile、synchronized等，我们仅放上volitile的**读可见**性图作下示例。

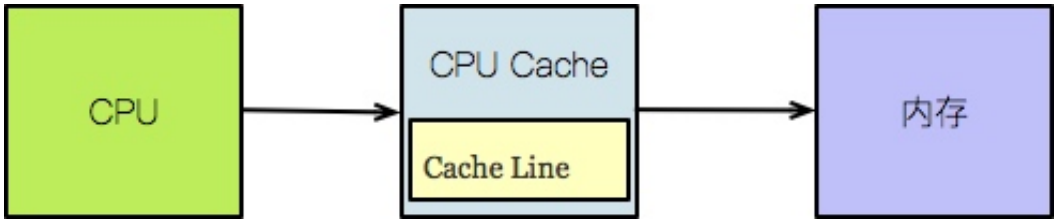


线程对共享变量会拷贝一份到工作区。线程1修改了变量以后，其他线程读这个变量的时候，都从主存里刷新一份，此所谓读可见。

JMM问题是纯粹的内存问题，也是高级java必备的知识点。

CacheLine & False Sharing

是的，内存的工艺制造还是跟不上CPU的速度，于是聪明的硬件工程师们，就又给加了一个缓存（哦不，是多个）。而Cache Line为CPU Cache中的最小缓存单位。

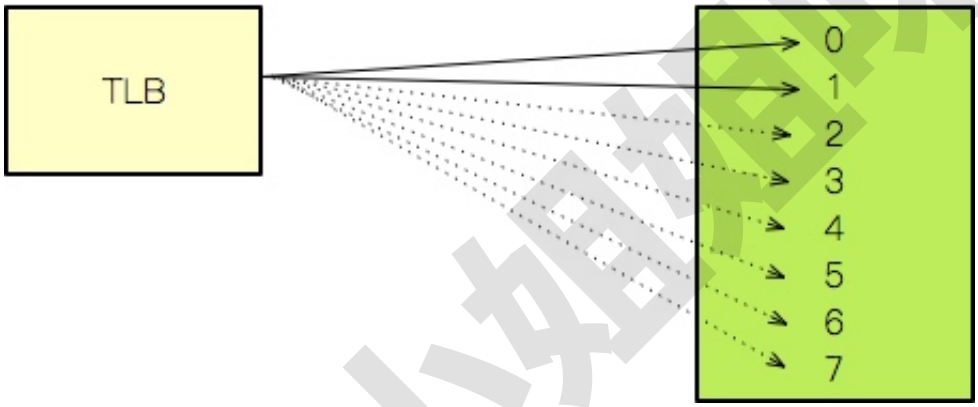


这个缓存是每个核的，而且大小固定。如果存在这样的场景，有多个线程操作不同的成员变量，但是相同的缓存行，这个时候会发生什么？。没错，伪共享（False Sharing）问题就发生了！

伪共享也是高级java的必备技能（虽然几乎用不到），赶紧去探索吧。

HugePage

回头看我们最长的那副图，上面有一个TLB，这个东西速度虽然高，但容量也是有限的。当访问频繁的时候，它会成为瓶颈。
TLB是存放Virtual Address和Physical Address的映射的。如图，把映射阔上一些，甚至阔上几百上千倍，TLB就能容纳更多地址了。像这种将Page Size加大的技术就是Huge Page。



HugePage有一些副作用，比如竞争加剧（比如redis： https://redis.io/topics/latency ）。但在大内存的现代，开启后会一定程度上增加性能（比如oracle： https://docs.oracle.com/cd/E11882_01/server.112/e10839/appi_vlm.htm)。

Numa

本来想将Numa放在cpu篇，结果发现numa改的其实是内存控制器。这个东西，将内存分段，分别"绑定"在不同的CPU上。也就是说，你的某核CPU，访问一部分内存速度贼快，但访问另外一些内存，就慢一些。

所以，Linux识别到NUMA架构后，默认的内存分配方案就是：优先尝试在请求线程当前所处的CPU的内存上分配空间。如果绑定的内存不足，先去释放绑定的内存。

以下命令可以看到当前是否是NUMA架构的硬件。

```
numactl --hardware
```

NUMA也是由于内存速度跟不上给加的折衷方案。Swap一些难搞的问题，大多是由于NUMA引起的。

总结

本文的其他，是给想更深入理解内存结构的同学准备的提纲。Linux内存牵扯的东西实在太多，各种缓冲区就是魔术。如果你遇到了难以理解的现象，费了九牛二虎之力才找到原因，不要感到奇怪。对发生的这一切，我深表同情，并深切的渴望通用量子计算机的到来。

那么问题来了，内存尚且如此，磁盘呢?

> 作者简介：**小姐姐味道** (xjjdog)，一个不允许程序员走弯路的公众号。聚焦基础架构和Linux。十年架构，日百亿流量，与你探讨高并发世界，给你不一样的味道。我的个人微信xjjdog0，欢迎添加好友，进一步交流。

近期热门文章

《必看！java后端，亮剑诛仙》

后端技术索引，中肯火爆

《Linux上，最常用的一批命令解析（10年精选）》
CSDN发布首日，1k赞。点赞率1/8。

《这次要是讲不明白Spring Cloud核心组件，那我就白编这故事了》
用故事讲解核心组件，包你满意

《Linux生产环境上，最常用的一套“Sed“技巧》
最常用系列Sed篇，简单易懂。Vim篇更加易懂。

Linux之《荒岛余生》（四）I/O篇

> 原创：小姐姐味道（微信公众号ID：xjjdog），欢迎分享，转载请保留出处。

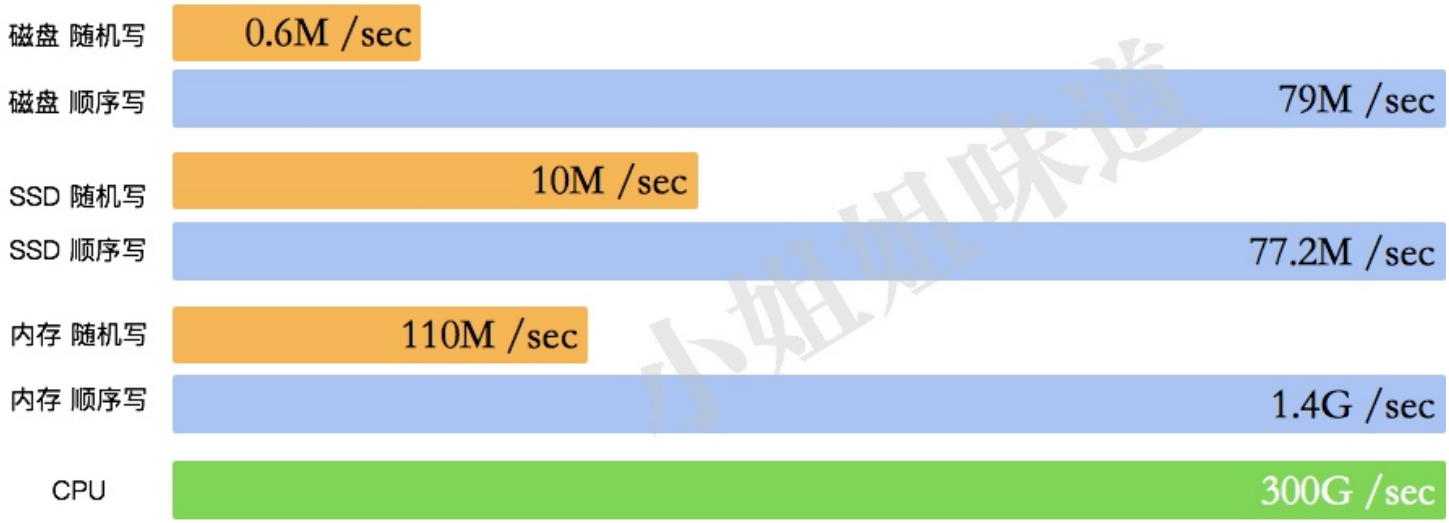
我们在cpu篇就提到，iowait高一般代表硬盘到瓶颈了。wait的意思，就是等，就像等正在化妆的女朋友，总是带着一丝焦躁。本篇是《荒岛余生》系列第四篇，I/O篇，计算机中最慢的那一环。其余参见：

一点背景

速度差异

`I/O` 不仅仅是硬盘，还包括外围的所有设备，比如键盘鼠标，比如 `1.44M` 的 `3.5` 英寸软盘（还有人记得么）。但服务器环境，泛指硬盘。

硬盘有多慢呢？我们不去探究不同设备的实现细节，直接看它的写入速度（数据有出入，仅作参考）：



可以看到普通磁盘的随机写和顺序写相差是非常大的。而 `随机写` 完全和cpu内存不在一个数量

级。缓冲区依然是解决速度差异的唯一工具，所以在极端情况比如断电等，就产生了太多的不确定性。这些缓冲区，都容易丢。

我们举例看一下为了消除这些性能差异，软件方面都做了哪些权衡。

- 数据库设计，采用`BTree`结构组织数据，通过减少对磁盘的访问和随机读取，来提高性能
- `Postgres`通过顺序写`WAL`日志、`ES`通过写`translog`等，通过预写，避免断电后数据丢失问题
- `Kafka`通过`顺序写`来增加性能，但在topic非常多的情况下性能弱化为随机写
- `Kafka`通过`零拷贝`技术，利用DMA绕过内存直接发送数据
- `Redis`使用内存模拟存储，它流行的主要原因就是和硬盘打交道的传统DB速度太慢
- 回忆一下内存篇的`buffer区`，是用来缓冲写入硬盘的数据的。linux的`sync`命令可以将buffer的数据刷到硬盘上，突然断电的话，就不好说了

做一个内存盘

如果你的内存够大，那么可以做一个内存盘。跑游戏，做文件交换什么的不要太爽。

```
mkdir /memdisk
mount  -t tmpfs -o size=1024m  tmpfs /memdisk/
```

以上命令划出`1GB`内存，挂载到`/memdisk`目录，然后就可以像使用普通文件夹一样使用它了。只是，速度不可同日而语。

使用dd命令测试写入速度

```
[root@xjj memdisk]# time dd if=/dev/zero of=test.file bs=4k count=200000
200000+0 records in
200000+0 records out
819200000 bytes (819 MB) copied, 0.533173 s, 1.5 GB/s

real    0m0.534s
user    0m0.020s
sys 0m0.510s
```

你见过这么快的硬盘么?

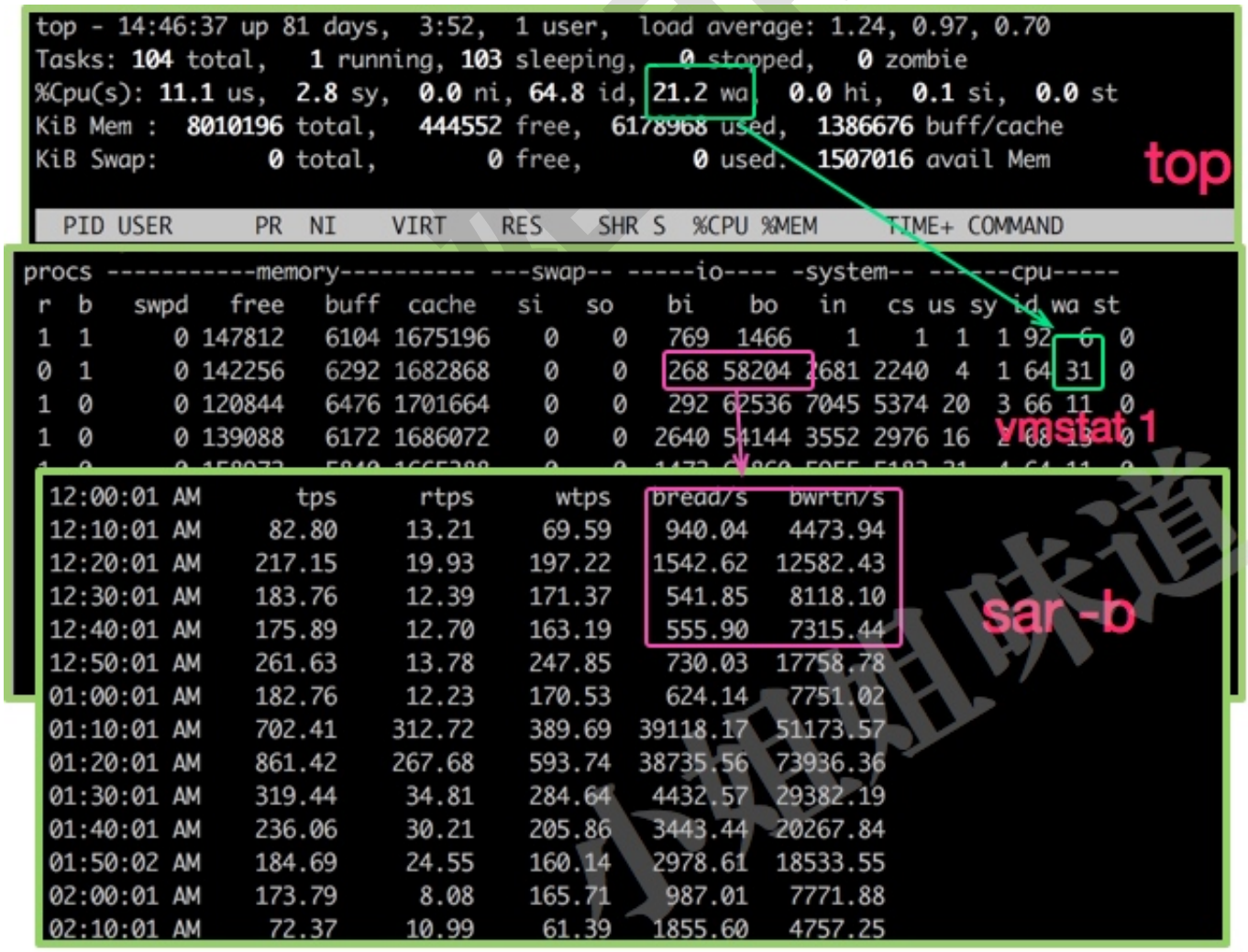
排查I/O问题的一般思路

判断 `I/O` 问题的命令其实并不多，大体有下面几个。

```
#查看wa
top
#查看wa和io(bi、bo)
vmstat 1
#查看性能相关i/o详情
sar -b 1 2
# 查看问题相关i/o详情
iostat -x 1
# 查看使用i/o最多的进程
iotop
```

惊鸿一瞥

首先是我们的老面孔。top、vmstat、sar命令，可以初步判断io情况。

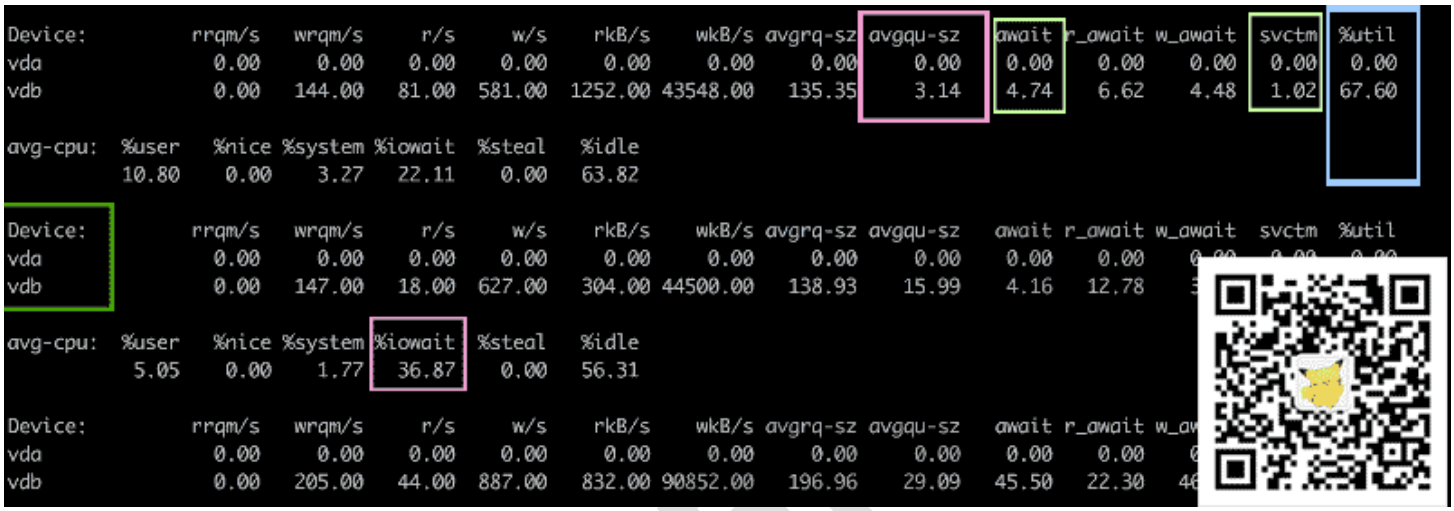


bi、bo等在你了解磁盘的类型后才有判断价值。我们有更专业的判断工具，所以这些信息一瞥即可。

在本例中，`wa`已经达到`30%`，证明cpu耗费在上面的时间太多。

定位问题

如何判断还需要结合`iostat`的帮助。有时候你是无可奈何的，比如这台MySQL的宿主机。你可能会更换更牛X的磁盘，或者整治耗I/O的慢SQL，再或者去改参数。



你瞧瞧，其实一个`iostat`命令就够了！我们对一些重要结果进行说明：

- **%util** 最重要的判断参数。一般地，如果该参数是100%表示设备已经接近满负荷运行了
- **Device** 表示发生在哪块硬盘。如果你有多快，则会显示多行
- **avgqu-sz** 还记得准备篇里提到的么？这个值是请求队列的饱和度，也就是平均请求队列的长度。毫无疑问，队列长度越短越好
- **await** 响应时间应该低于5ms，如果大于10ms就比较大了。这个时间包括了队列时间和服务时间
- **svctm** 表示平均每次设备`I/O`操作的服务时间。如果`svctm`的值与`await`很接近，表示几乎没有`I/O`等待，磁盘性能很好，如果`await`的值远高于`svctm`的值，则表示`I/O`队列等待太长，系统上运行的应用程序将变慢

总体来说，`%util`代表了硬盘的繁忙程度，是你进行扩容增加配置的指标。
而`await`、`avgqu-sz`、`svctm`等是硬盘的性能指标，如果`%util`正常的情况下反应异常则代表你的磁盘可能存在问题。

> iostat打印出的第1个报告，数值是基于最后一次系统启动的时间统计的；基于这个原因，在

> 大部份情况下，iostat打印出的第1个报告应该被忽略。

另外一种方式就是通过ps命令或者top命令得到状态为D的进程。比如下面命令，循环10次进行状态抓取。

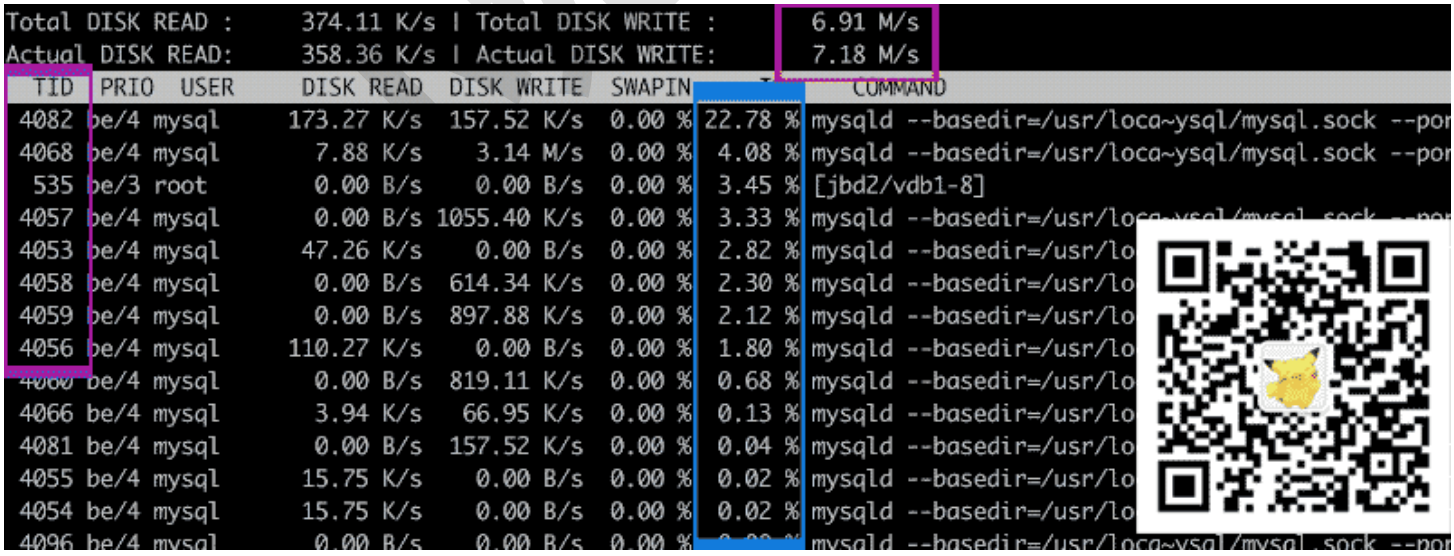
```
for x in `seq 1 1 10`; do ps -eo state,pid,cmd | grep "^D"; echo "----"$x; sl
eep 5; done
```

找到I/O大户

`iostat` 查看的是硬盘整体的状况，但我们想知道到底是哪个应用引起的。top系列有一个 `iotop` ，能够像top一样，看到占用 `I/O` 最多的应用。 `iotop` 的本质是一个python脚本，从 `proc` 目录中获取thread的IO信息，进行汇总。比如

```
[root@xjj ~]# cat  /proc/5178/io
rchar: 628
wchar: 461
syscr: 2
syscw: 8
read_bytes: 0
write_bytes: 0
cancelled_write_bytes: 0
```

如图，显示了当前系统硬盘的读写速度和应用的I/O使用占比。



那么怎么看应用所关联的所有文件信息呢？可以使用lsof命令，列出了所有的引用句柄。

```
[root@xjj ~]# lsof -p 4050
COMMAND   PID   USER    FD    TYPE             DEVICE      SIZE/OFF       NODE NAME
```



```
mysqld  4050 mysql  314u  IPv6          115230644          0t0       TCP iZ2z
eeaoqoxksuhnqbgfjjZ:mysql->10.30.134.8:54943 (ESTABLISHED)
mysqld  4050 mysql  320u   REG             253,17         2048  44829072 /dat
a/mysql/mysql/user.MYI
mysqld  4050 mysql  321u   REG             253,17         3108  44829073 /dat
a/mysql/mysql/user.MYD
...
```

更深层的信息，可以通过类似`Percona-Toolkit`这种工具去深入排查，比如`pt-ioprofile`，在此不做详解。

几个特殊进程说明*

kswapd0

这依然是由于swap虚拟内存引起的，证明虚拟内存正在大量使用

jbd2

全称是journaling block driver。这个进程实现的是文件系统的日志功能，磁盘使用日志功能来保证数据的完整性。

可以通过以下方法将其关掉，但一定要权衡

```
dumpe2fs /dev/sda1
tune2fs -o journal_data_writeback /dev/sda1
tune2fs -O "^has_journal" /dev/sda1
e2fsck -f /dev/sda1
```

同时在fstab下重新设定一下，在defaults之后增加

```
defaults,data=writeback,noatime,nodiratime
```

你可能会有一个数量级的性能提升。

其他

硬盘快满了

使用`df`命令可以看到磁盘的使用情况。一般，使用达到90%就需要重点关注，然后人工介入删除文件了。

```
[root@xjj ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        40G  8.3G   29G  23% /
/dev/vdb1      1008G   22G  935G   3% /data
tmpfs           1.0G  782M  243M  77% /memdisk
```

使用`du`命令可以查看某个文件的大小。

```
[root@xjj ~]# du -h test.file
782M    test.file
```

如果想把一个文件置空，千万不要直接rm。其他应用可能保持着它的引用，经常发上文件删除但空间不释放的问题。比如tomcat的calatina.out文件，如果你想清空里面的内容，不要rm，可以执行下面的命令进行文件内容清空

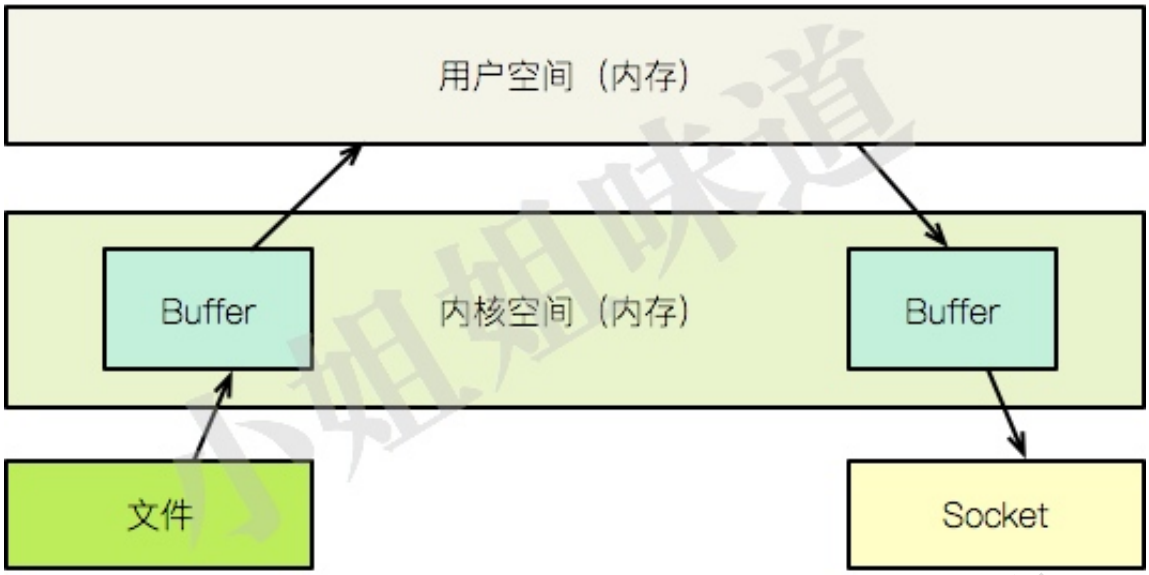
```
cat /dev/null > calatina.out
```

这非常安全。

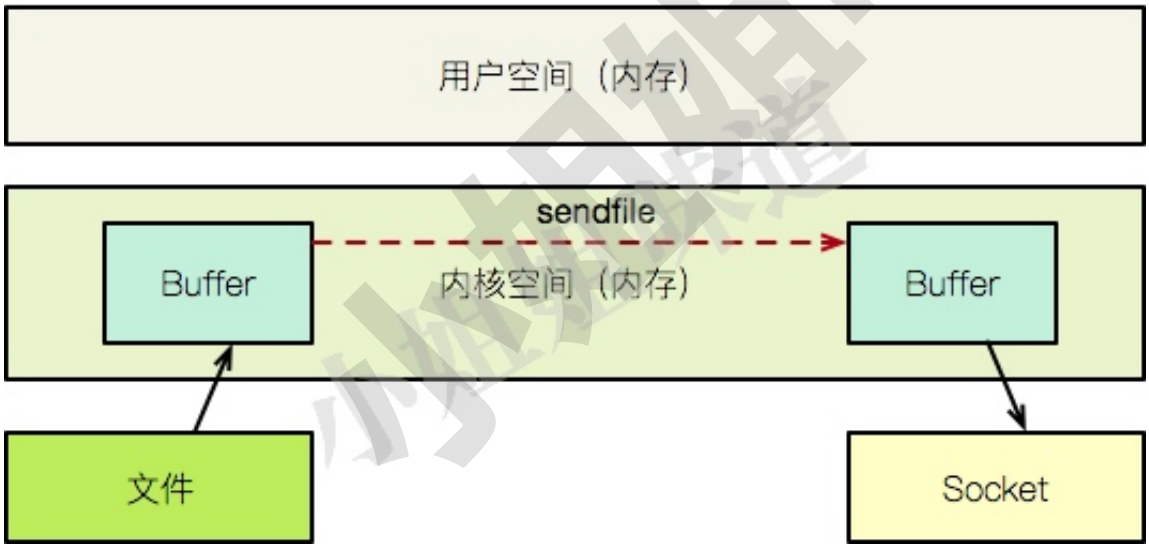
zero copy

kafka比较快的一个原因就是使用了zero copy。所谓的Zero copy，就是在操作数据时，不需要将数据buffer从一个内存区域拷贝到另一个内存区域。因为少了一次内存的拷贝，CPU的效率就得到提升。

我们来看一下它们之间的区别：



要想将一个文件的内容通过socket发送出去，传统的方式需要经过以下步骤：
=>将文件内容拷贝到内核空间
=>将内核空间的内容拷贝到用户空间内存，比如java应用
=>用户空间将内容写入到内核空间的缓存中
=>socket读取内核缓存中的内容，发送出去



如上图，zero copy在内核的支持下，少了一个步骤，那就是内核缓存向用户空间的拷贝。即节省了内存，也节省了cpu的调度时间，效率很高。

值得注意的是，java中的zero copy，指的其实是DirectBuffer；而Netty的zero copy是在用户空间中进行的优化。两者并不是一个概念。

Linux通用I/O模型

面向接口编程？linux从诞生开始就有了。在linux下，一切都是文件，比如设备、脚本、可执行文

件、目录等。操作它们，都有公用的接口。所以，编写一个设备驱动，就是实现这些接口而已。

- fd = open(pathname, flags, mode)
- rlen = read(fd, buf, count)
- wlen = write(fd, buf, count)
- status = close(fd)

使用`stat`命令可以看到文件的一些状态。

```
[root@xjj ~]# stat test.file
  File: ‘test.file’
  Size: 819200000   Blocks: 1600000    IO Block: 4096   regular file
Device: 26h/38d Inode: 3805851     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2018-11-29 12:56:34.801412100 +0800
Modify: 2018-11-29 12:56:35.334415131 +0800
Change: 2018-11-29 12:56:35.334415131 +0800
```

而使用`file`命令，能得到文件的类型信息

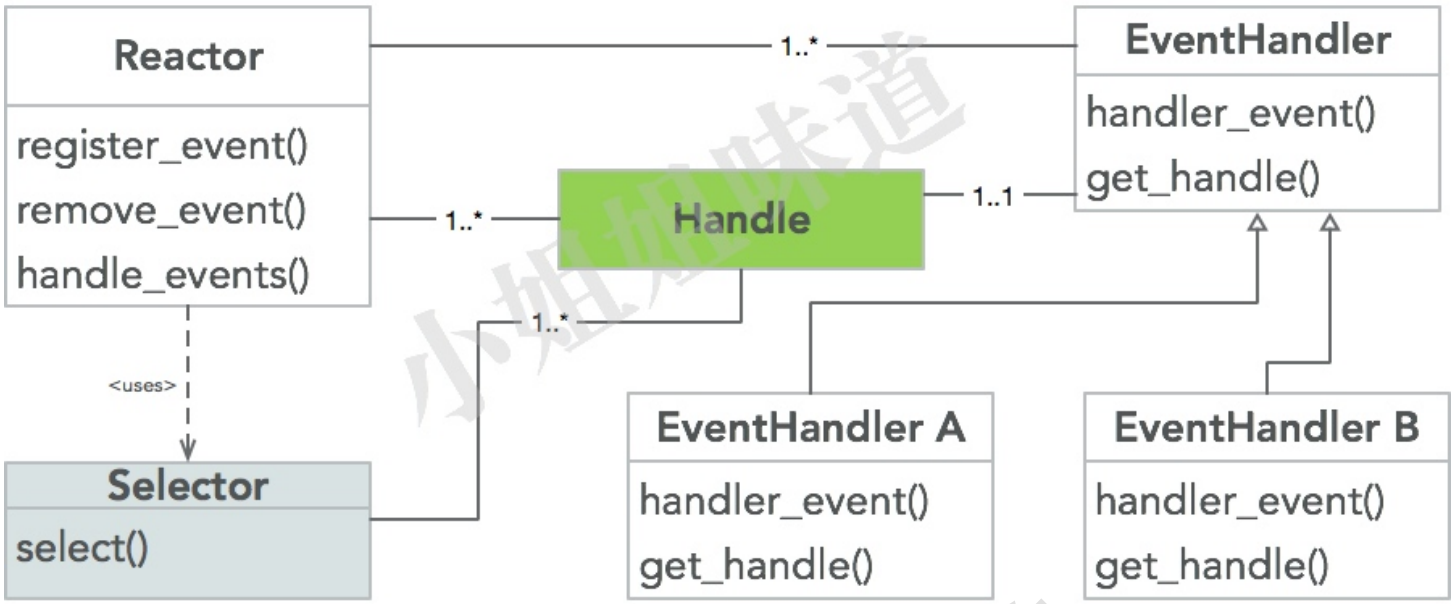
```
[root@xjj ~]# file /bin/bash
/bin/bash: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically l
inked (uses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=ab347e897f002d8
e3836479e2430d75305fe6a94, stripped
```

I/O模型

谈完了`I/O`问题的定位方法，就不得不提一下Linux下的`5`种`I/O`模型。等等，这其实是一个网络问题。

- **同步阻塞IO（Blocking IO）** 传统的IO模型
- **同步非阻塞IO（Non-blocking IO）** 非阻塞IO要求socket被设置为NONBLOCK
- **IO多路复用（IO Multiplexing）** 即经典的Reactor设计模式
- **异步IO（Asynchronous IO）** 即经典的Proactor设计模式

java中`nio`使用的就是多路复用功能，也就是使用的Linux的`epoll`库。一般手撸nio的比较少了，大都是直接使用`netty`进行开发。它们用到的，就是经典的reactor模式。



我们能得到什么

除了能够帮助我们评价I/O瓶颈，一个非常重要的点就是：业务研发要合理输出日志，日志文件不仅仅是影响磁盘那么简单，它还会耗占大量的CPU。

对于我们平常的优化思路，也有章可循。像mysql、es、postgresql等，在写真正的数据库文件之前，会有很多层缓冲。如果你对数据可靠性要求并不是那么严重，调整这些缓冲参数的阈值和执行间隔，通常会得到较大的性能提升。

当然，了解I/O还能帮助我们更好的理解一些软件的设计理念。比如leveldb是如何通过LSM来组织数据；ES为什么会存在那么多的段合并；甚至Redis为何存在。

当然，你可能再也无法忍受单机硬盘的这些特性，转而寻求像ceph这样的解决方案。但无论如何，我们都该向所有的数据库研发工作者致敬，在很长一段时间里，我们依然需要和缓慢的I/O共行。

> 作者简介：**小姐姐味道** (xjjdog)，一个不允许程序员走弯路的公众号。聚焦基础架构和Linux。十年架构，日百亿流量，与你探讨高并发世界，给你不一样的味道。我的个人微信xjjdog0，欢迎添加好友，进一步交流。

近期热门文章

《必看！java后端，亮剑诛仙》

后端技术索引，中肯火爆

《Linux上，最常用的一批命令解析（10年精选）》
CSDN发布首日，1k赞。点赞率1/8。

《这次要是讲不明白Spring Cloud核心组件，那我就白编这故事了》
用故事讲解核心组件，包你满意

《Linux生产环境上，最常用的一套“Sed“技巧》
最常用系列Sed篇，简单易懂。Vim篇更加易懂。

Linux之《荒岛余生》（五）网络篇

> 原创：小姐姐味道（微信公众号ID：xjjdog），欢迎分享，转载请保留出处。

你想通过执行ping google.com来判断网络连通性么？我想你这是在侮辱方教授。本篇是《荒岛余生》系列第五篇，网络篇，但不会教你fq。其余参见：

Linux之《荒岛余生》（一）准备篇
Linux之《荒岛余生》（二）CPU篇
Linux之《荒岛余生》（三）内存篇
Linux之《荒岛余生》（四）I/O篇
Linux之《荒岛余生》（五）网络篇

看着kali linux上百个网络命令，我陷入了沉思。专业的网络命令实在是太多了，如果要罗列，上千个也是有的。个人不是渗透测试工作者，大部分功能只知皮毛。所以本文是非常浅显的技术总结，仅聚焦工作中常用到的一些Linux命令。

由于 `nio` 的普及， `ck10k` 的问题已经成为过去式。现在随便一台服务器，就可以支持数十万级别的连接了。那么我们来算一下，100万的连接需要多少资源。

首先，每一个连接都是文件句柄，所以需要文件描述符数量支持才行，每一个socket内存占用15k-20k之间，这样，仅维护相应socket，就需要 `20G` 内存；而广播一个1KB的消息需要占用的带宽为 `1000M` ！

查看当前系统的连接

如何看当前系统有多少连接呢？可以使用 `netstat` 结合 `awk` 进行统计。如下脚本，统计了每一种状态的tcp连接数量

```
# netstat -antp | awk '{a[$6]++}END{ for(x in a)print x,a[x]}'
LISTEN 41
CLOSE_WAIT 24
ESTABLISHED 150
Foreign 1
TIME_WAIT 92
```

但如果你在一台有上万连接的服务器上执行这个命令，你可能会等上很长时间。所以，我们有了第二代网络状态统计工具： `netstat` => `ss` （可别和那个越狱工具搞混了）。


```
# ss -s
Total: 191 (kernel 220)
TCP:   5056 (estab 42, closed 5000, orphaned 3, synrecv 0, timewait 5000/0),
ports 3469
...
```

`netstat`属于`net-tools`工具集，而`ss`属于`iproute`。其命令对应如下，是时候和net-tools说Bye了。

用途	net-tools	iproute
统计	ifconfig	ss
地址	netstat	ip addr
路由	route	ip route
邻居	arp	ip neigh
VPN	iptunnel	ip tunnel
VLAN	vconfig	ip link
组播	ipmaddr	ip maddr

ss命令

基本使用

我们按照使用场景来看下ss的用法。

查看系统正在监听的tcp连接

```
ss -atr
ss -atn #仅ip
```

查看系统中所有连接

```
ss -alt
```

查看监听444端口的进程pid

```
ss -ltp | grep 444
```

查看进程555占用了哪些端口

```
ss -ltp | grep 555
```

显示所有udp连接

```
ss -u -a
```

查看TCP sockets，使用-ta选项
查看UDP sockets，使用-ua选项
查看RAW sockets，使用-wa选项
查看UNIX sockets，使用-xa选项

和某个ip的所有连接

```
ss dst 10.66.224.130
ss dst 10.66.224.130:http
ss dst 10.66.224.130:smtp
ss dst 10.66.224.130:443
```

显示所有的http连接

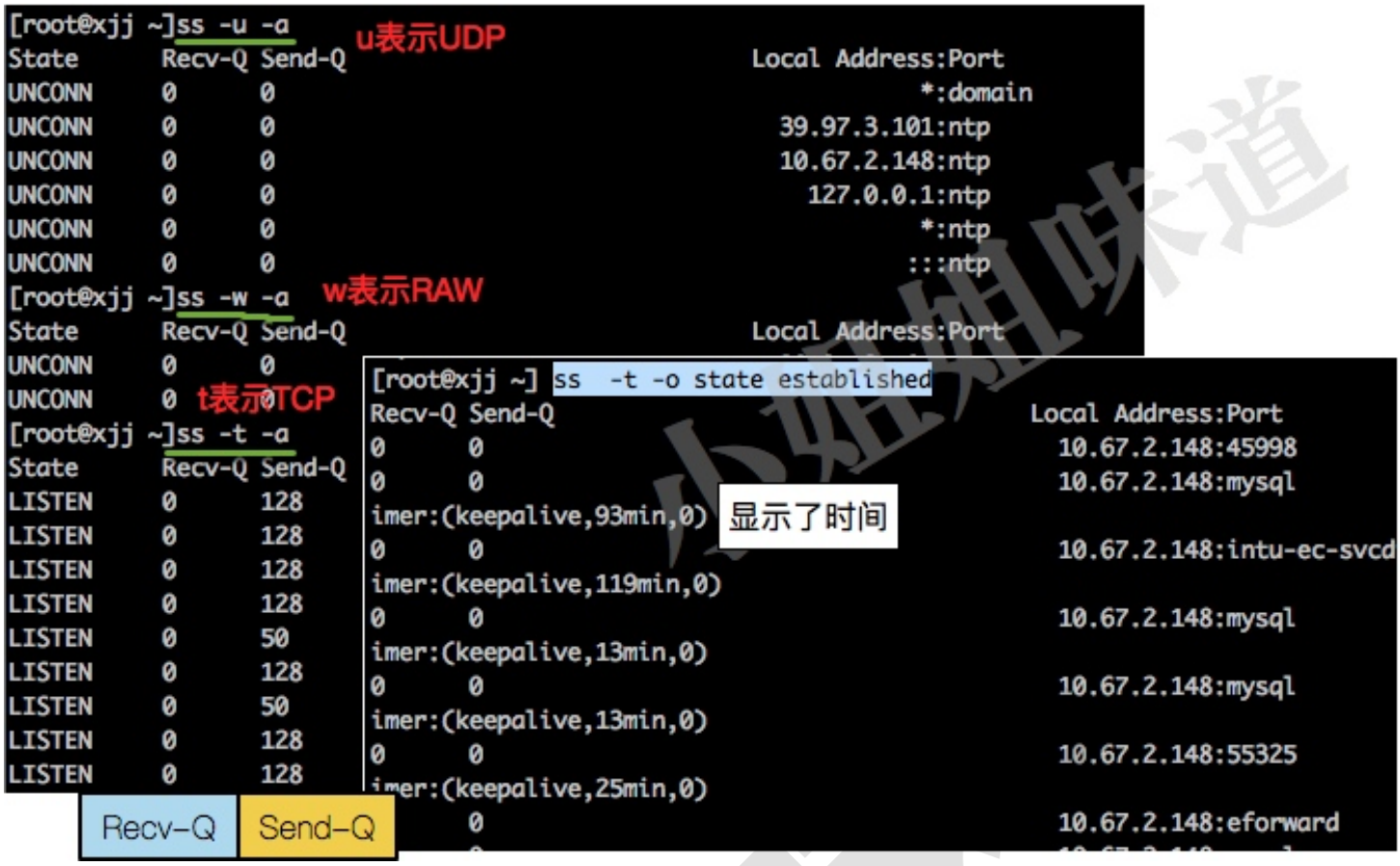
```
ss  dport = :http
```

查看连接本机最多的前10个ip地址

```
netstat -antp | awk '{print $4}' | cut -d ':' -f1 | sort | uniq -c  | sort -n
 -k1 -r | head -n 10
```

Recv-Q和Send-Q

注意ss的执行结果，我们说明一下Recv-Q和Send-Q。



这两个值，在 LISTEN 和 ESTAB 状态分别代表不同意义。一般，正常的应用程序这两个值都应该为0（backlog除外）。数值越大，说明问题越严重。

LISTEN状态

- Recv-Q：代表建立的连接还有多少没有被accept，比如Nginx接受新连接变的很慢
- Send-Q：代表listen backlog值

ESTAB状态

- Recv-Q：内核中的数据还有多少(bytes)没有被应用程序读取，发生了一定程度的阻塞
- Send-Q：代表内核中发送队列里还有多少(bytes)数据没有收到ack，对端的接收处理能力不强

查看网络流量

查看流量

有很多工具可以看网络流量，但我最喜欢sar。sar是linux上功能最全的监控软件。如图，使

用 `sar -n DEV 1` 即可每秒刷新一次网络流量。

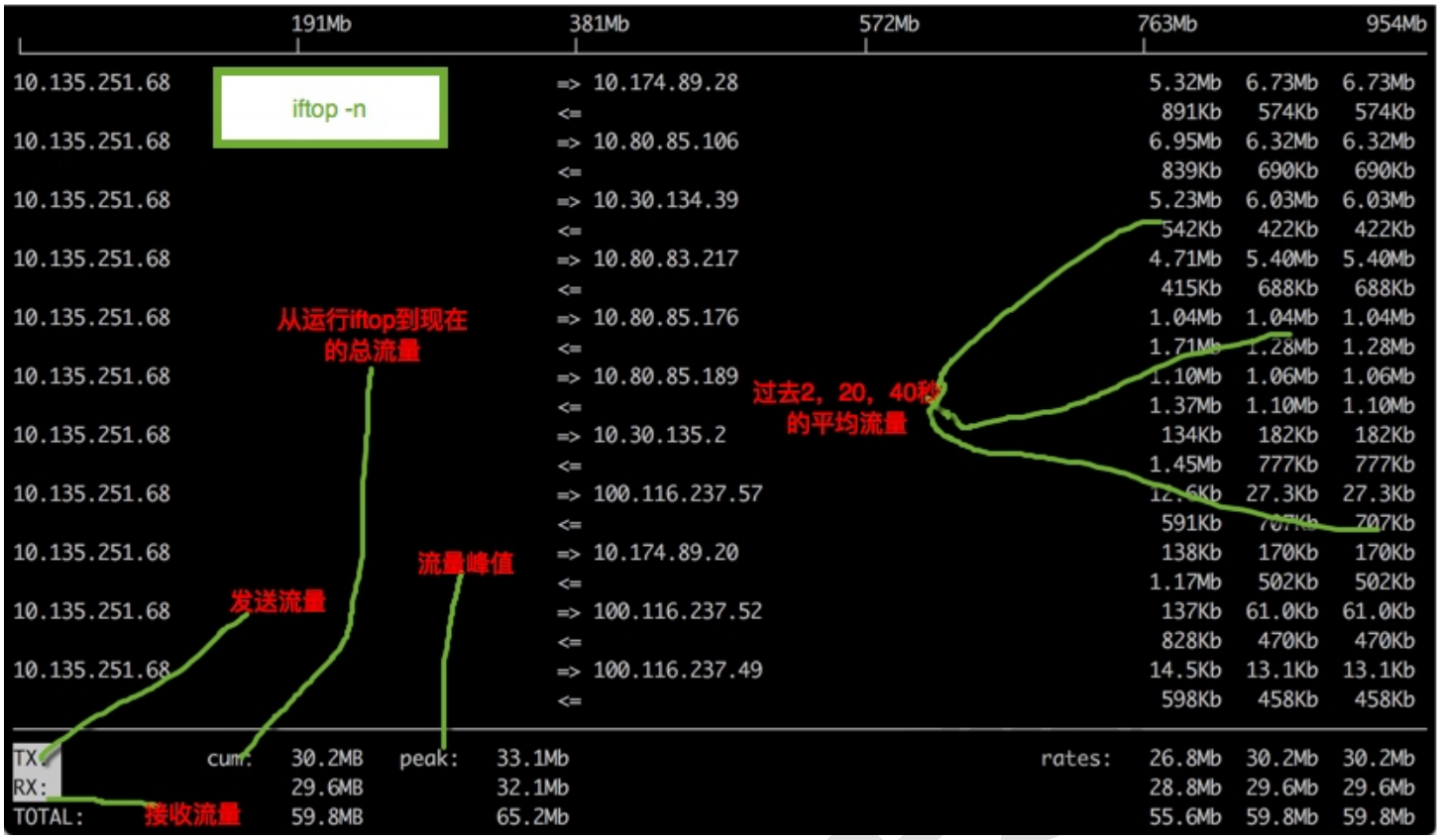
```
11:13:52 AM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
11:13:53 AM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
11:13:53 AM      eth0   4867.71   4808.33   3452.16   3512.70      0.00      0.00      0.00

11:13:53 AM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
11:13:54 AM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
11:13:54 AM      eth0   4961.86   4901.03   3380.09   3393.61      0.00      0.00      0.00

11:13:54 AM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
11:13:55 AM        lo      3.03      3.03      0.18      0.18      0.00      0.00      0.00
11:13:55 AM      eth0   5928.28   5739.39   3242.26   3294.73      0.00      0.00      0.00

11:13:55 AM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
11:13:56 AM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
11:13:56 AM      eth0   6151.04   6092.71   3955.68   3927.48      0.00      0.00      0.00

11:13:56 AM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
11:13:57 AM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
11:13:57 AM      eth0   7194.85   6932.99   3913.72   3889.58      0.00      0.00      0.00
```

当然，你也可以使用ifstat、nload、iptraf等命令查看。然而数据来源，还是来自我们的/proc目录

```
watch cat /proc/net/dev
```

查看占流量最大的IP

有时候我们发现网络带宽占用非常高，但我们无法判断到底流量来自哪里。这时候，iftop就可以帮上忙了。如图，可以很容易的找出流量来自哪台主机。



当你不确定内网的流量来源，比如有人在压测，api调用不合理等，都可以通过这种方法找到他。

抓包

tcpdump

当我们需要判断是否有流量，或者调试一个难缠的netty应用问题，则可以通过抓包的方式去进行进一步的判断。在Linux上，可以通过 `tcpdump` 命令抓取数据，然后使用 `Wireshark` 进行分析。

```
tcpdump -i eth0 -nn -s0 -v port 80
```

- `-i` 指定网卡进行抓包
- `-n` 和ss一样，表示不解析域名
- `-nn` 两个n表示端口也是数字，否则解析成服务名
- `-s` 设置抓包长度，0表示不限制
- `-v` 抓包时显示详细输出，-vv、-vvv依次更加详细

1）加入 `-A` 选项将打印ascii， `-X` 打印hex码。


```
tcpdump -A -s0 port 80
```

2）抓取特定ip的相关包

```
tcpdump -i eth0 host 10.10.1.1
tcpdump -i eth0 dst 10.10.1.20
```

3） `-w` 参数将抓取的包写入到某个文件中

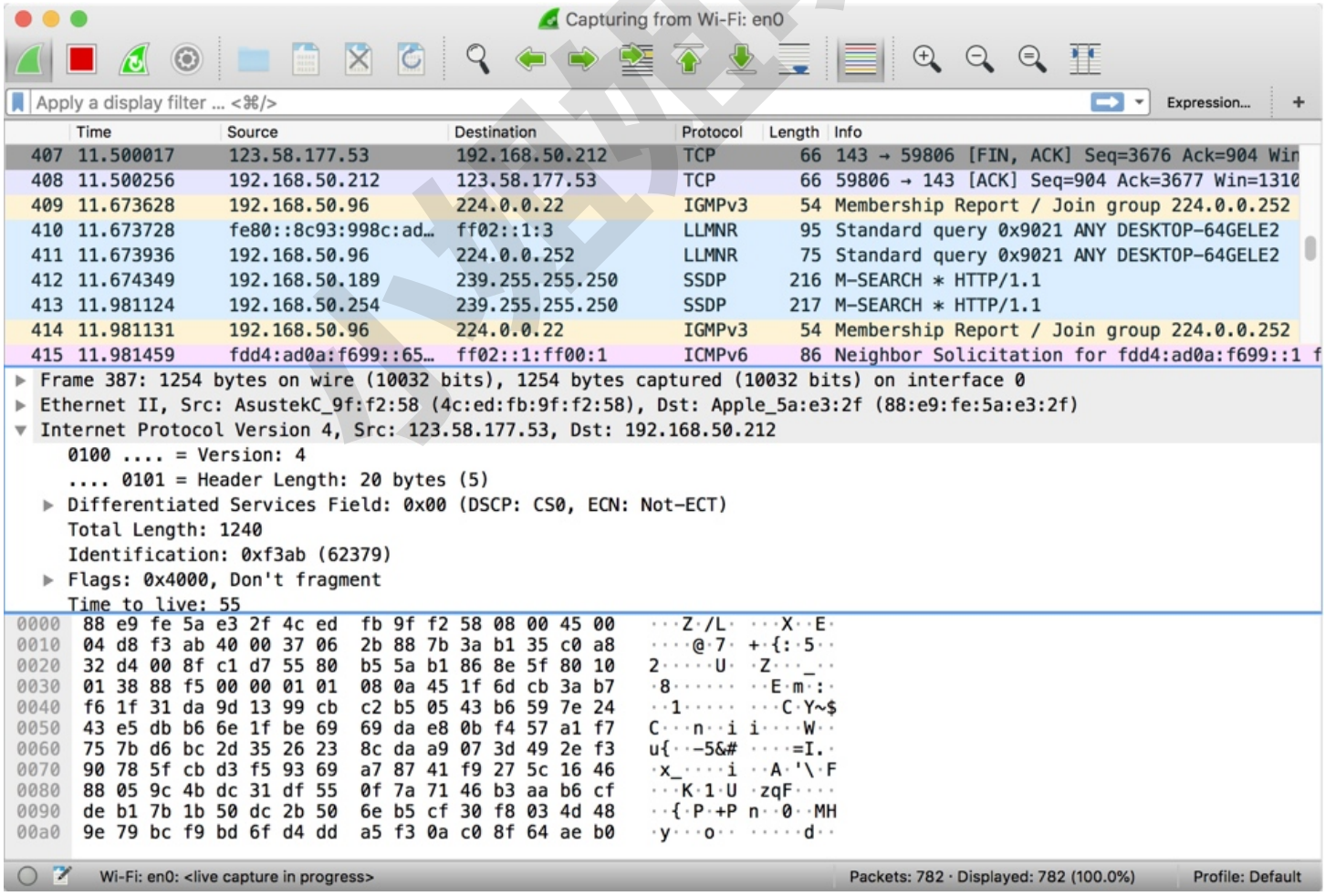
```
tcpdump -i eth0 -s0 -w test.pcap
```

4）tcpdump支持表达式，还有更加复杂的例子，比如抓取系统中的get,post请求（非https)

```
tcpdump -s 0 -v -n -l | egrep -i "POST /|GET /|Host:"
```

更多参见
https://hackertarget.com/tcpdump-examples/

抓取的数据，使用wireshark查看即可。



http抓包

抓包工具将自身当作代理，能够抓取你的浏览器到服务器之间的通讯，并提供修改、重放、批量执行的功能。是发现问题，分析协议，攻击站点的利器。常用的有以下三款：

- Burpsuite （跨平台)
- Fiddle2 (Win)
- Charles (Mac)

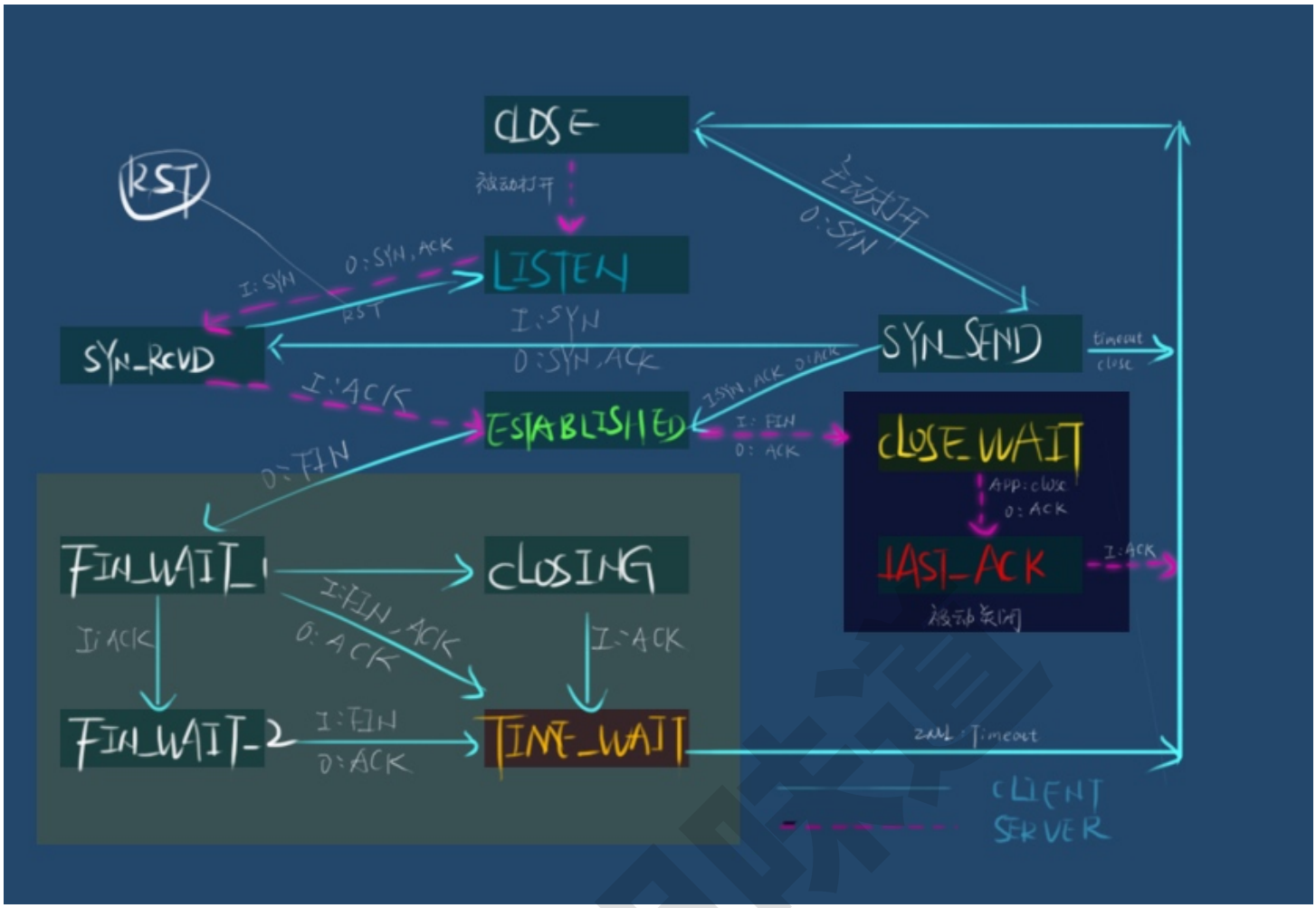
坏事要偷偷的干哦。

流量复制

你可能需要使你的生产环境HTTP真实流量在开发环境或者预演环境重现，这样就用到了流量复制功能。
有三个工具可供选择，个人倾向于Gor。

- Gor
- TCPReplay
- TCPCopy

连接数过多问题



根据TCP/IP介绍，socket大概包含10个连接状态。我们平常工作中遇到的，除了针对SYN的拒绝服务攻击，如果有异常，大概率是TIME_WAIT和CLOSE_WAIT的问题。
TIME_WAIT一般通过优化内核参数能够解决；CLOSE_WAIT一般是由于程序编写不合理造成的，更应该引起开发者注意。

TIME_WAIT

TIME_WAIT是主动关闭连接的一方保持的状态，像nginx、爬虫服务器，经常发生大量处于time_wait状态的连接。TCP一般在主动关闭连接后，会等待 `2MS` ，然后彻底关闭连接。由于HTTP使用了TCP协议，所以在这些频繁开关连接的服务器上，就积压了非常多的TIME_WAIT状态连接。

某些系统通过dmesg可以看到以下信息。

```
__ratelimit: 2170 callbacks suppressed
TCP: time wait bucket table overflow
TCP: time wait bucket table overflow
TCP: time wait bucket table overflow
TCP: time wait bucket table overflow
```


通过ss -s命令查看，可以看到timewait已经有2w个了。

```
ss -s
Total: 174 (kernel 199)
TCP:   20047 (estab 32, closed 20000, orphaned 4, synrecv 0, timewait 20000/0
), ports 10785
```

sysctl命令可以设置这些参数，如果想要重启生效的话，加入/etc/sysctl.conf文件中。

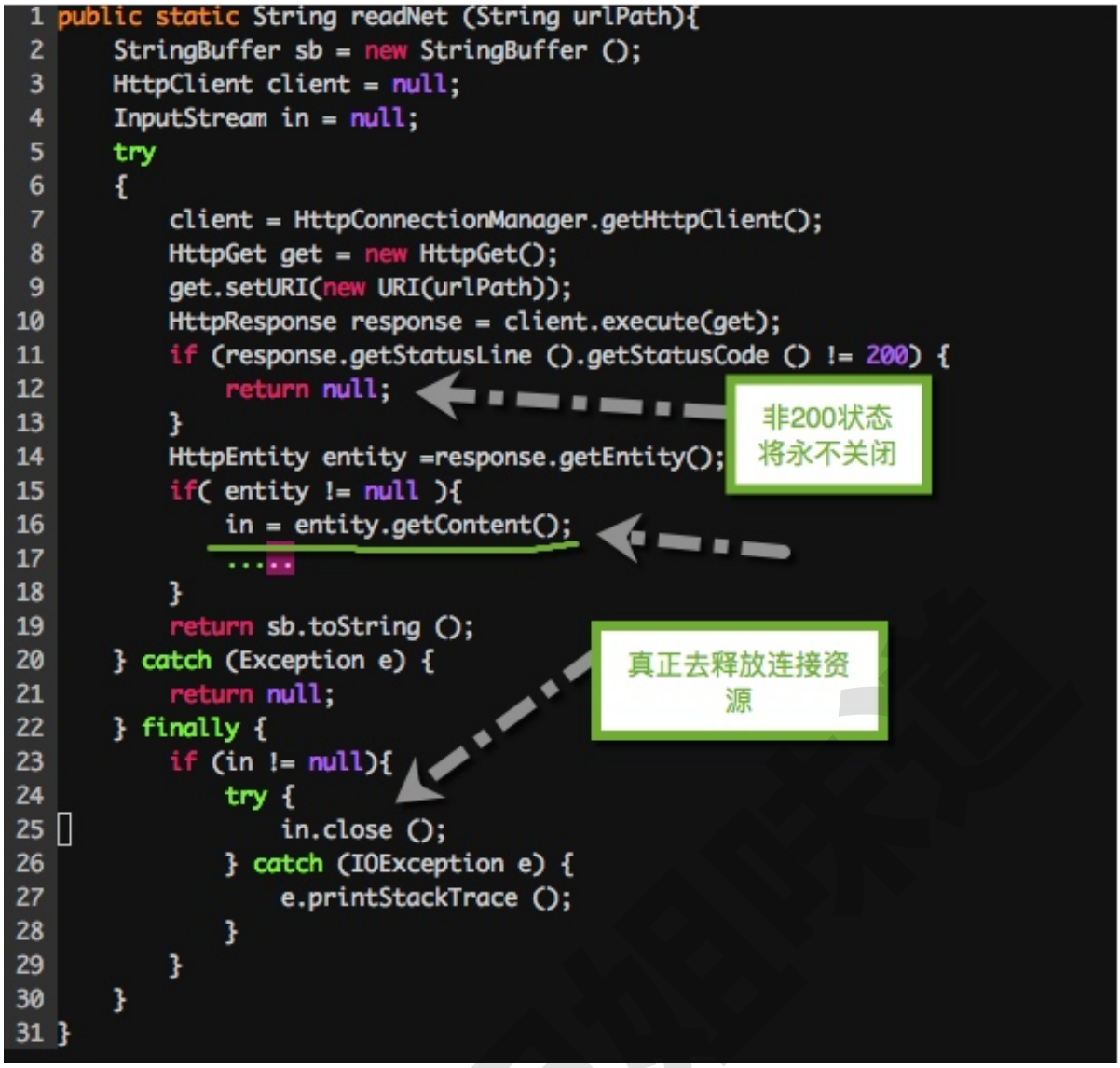
```
# 修改阈值
net.ipv4.tcp_max_tw_buckets = 50000
# 表示开启TCP连接中TIME-WAIT sockets的快速回收
net.ipv4.tcp_tw_reuse = 1
#启用timewait 快速回收。这个一定要开启，默认是关闭的。
net.ipv4.tcp_tw_recycle= 1
# 修改系统默认的TIMEOUT时间,默认是60s
net.ipv4.tcp_fin_timeout = 10
```

测试参数的话，可以使用 sysctl -w net.ipv4.tcp_tw_reuse = 1 这样的命令。如果是写入进文件的，则使用sysctl -p生效。

CLOSE_WAIT

CLOSE_WAIT一般是由于对端主动关闭，而我方没有正确处理的原因引起的。说白了，就是程序写的有问题，属于危害比较大的一种。

我们拿"csdn 谐音太郎"遇到的一个典型案例来说明。



代码是使用HttpClient的一个使用片段。在这段代码里，通过调用in.close()来进行连接资源的清理。但可惜的是，代码中有一个判断：非200状态的连接直接返回null。在这种情况下，`in`连赋值的机会都没有，当然也就无法关闭，然后就发生了连接泄漏。

所以，HttpClient的正确关闭方式是使用其api：`abort()`。

其他常用命令

应用软件

```
# 断点续传下载文件
wget -c $url
# 下载整站
wget -r -p -np -k $url
# 发送网络连接（常用）
```



```
curl -XGET $url
# 传输文件
scp
sftp
# 数据镜像备份
rsync
```

检测工具

```
# 连通性检测
ping google.com
# 到对端路由检测
tracepath google.com
# 域名检测
dig google.com
nslookup google.com
# 网络扫描工具
nmap
# 压力测试
iperf
# 全方位监控工具（好东西）
nmon
```

配置工具

```
# 停止某个网卡
ifdown
# 开启某个网卡
ifup
# 多功能管理工具
ethtool
```

压力测试

```
wrk
ab
webbench
http_load
```

多功能工具

```
# 远程登录
telnet
ssh
nc
# 防火墙
iptables -L
```

结尾

除了基本的工具，本文提到的很多网络命令，都不是预装的，需要使用yum自行安装。网络编程方面的学习，我觉得，读一下《TCP/IP详解 卷1：协议》这本书，然后写几个Netty应用就可以了。我这里找到了一本在线的，不用花钱买了。
http://www.52im.net/topic-tcpipvol1.html?mobile=no

如果想要更加深入，可以选择：

- 《TCP/IP详解 卷1：协议》
- 《UNIX网络编程》
- 《Netty权威指南》

NIO我们已经在I/O篇提起了，在此不再做详细介绍。等你碰到所谓的拆包粘包问题，遇到心跳和限流问题，甚至遇到了流量整形问题，那么证明你离一个专业的网络编程程序员越来越近了。

> 作者简介：**小姐姐味道** (xjjdog)，一个不允许程序员走弯路的公众号。聚焦基础架构和Linux。十年架构，日百亿流量，与你探讨高并发世界，给你不一样的味道。我的个人微信xjjdog0，欢迎添加好友，进一步交流。

近期热门文章

《必看！java后端，亮剑诛仙》

后端技术索引，中肯火爆

《Linux上，最常用的一批命令解析（10年精选）》
CSDN发布首日，1k赞。点赞率1/8。

《这次要是讲不明白Spring Cloud核心组件，那我就白编这故事了》

用故事讲解核心组件，包你满意

《Linux生产环境上，最常用的一套“Sed“技巧》
最常用系列Sed篇，简单易懂。Vim篇更加易懂。