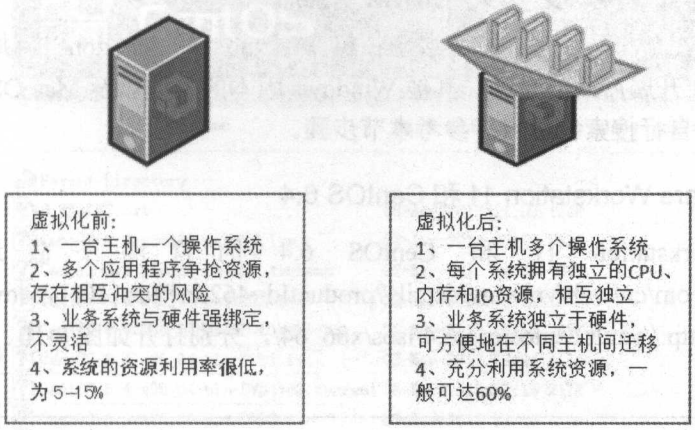

图 8-8　虚拟机实现一台计算机多系统

虚拟机可以模拟出其他种类的操作系统，比如我现在的系统是 win10 的，但是我需要 Linux 的系统，那我可以在虚拟机中安装 Linux 的系统。而且可以安装很多台，实现一台变多台。所以虚拟机也是我们后面章节学习多台分布式架构：副本集以及分片的好帮手。

虚拟机系统具有兼容性、隔离性、封装性、硬件独立性，它与主机系统相互独立运行，你完全可以把它们当成 2 台、3 台机子看待。具体有什么用处，举例来说，不开启文件共享，在虚拟机中尝试病毒之类的，不会影响到主机，如图 8-9 所示。

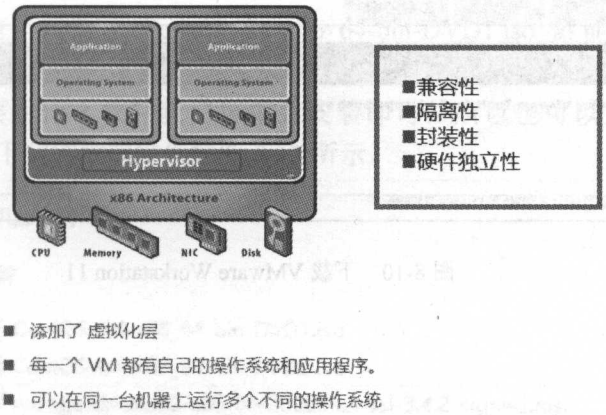

图 8-9　虚拟机的特点

流行的虚拟机软件有 VMware(VMWare ACE）、Virtual Box 和 Virtual PC，它们都能在 Windows 系统上虚拟出多个计算机。

Mac OS 系统中虚拟机也有免费的 VirtualBox 和收费的 VMware Fusion、ParallelsDesktop。

8.3.2 虚拟机安装以及安装 Linux 系统

我们本小节会在 Win 10 系统中安装虚拟机 VMware Workstation 11 并安装 Linux 系统 CentOS 6.4[14]，以便开展后面的学习。其他 Windows 版本的系统以及 Mac OS 安装虚拟机的步骤是类似的，可以自行搜索教程或者参考本节步骤。

1. 下载 VMware Workstation 11 和 CentOS 6.4

VMware Workstation 11 和 CentOS 6.4 的官网下载地址分别是 https://my.vmware.com/cn/web/vmware/details?productId=462&rPId=11036&downloadGroup=WKST-1110-WIN 和 http://vault.centos.org/6.4/isos/x86_64/，分别打开如图 8-10、图 8-11 所示的页面进行下载。

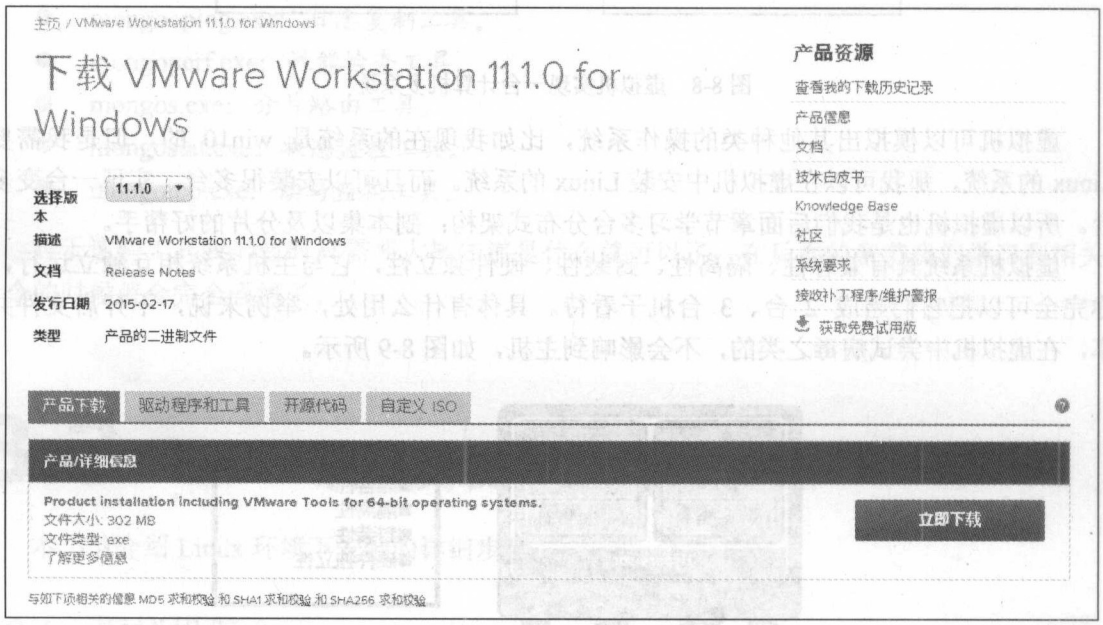

图 8-10 下载 VMware Workstation 11

[14] CentOS 是一个基于 Red Hat Linux 提供的可自由使用源代码的企业级 Linux 发行版本。每个版本的 CentOS 都会获得十年的支持（通过安全更新方式）。新版本的 CentOS 大约每两年发行一次，而每个版本的 CentOS 会定期（大概每六个月）更新一次，以便支持新的硬件。这样，建立一个安全、低维护、稳定、高预测性、高重复性的 Linux 环境。 CentOS 是 Community Enterprise Operating System 的缩写。CentOS 是 RHEL（Red Hat Enterprise Linux）源代码再编译的产物，而且在 RHEL 的基础上修正了不少已知的 Bug ，相对于其他 Linux 发行版，其稳定性值得信赖。

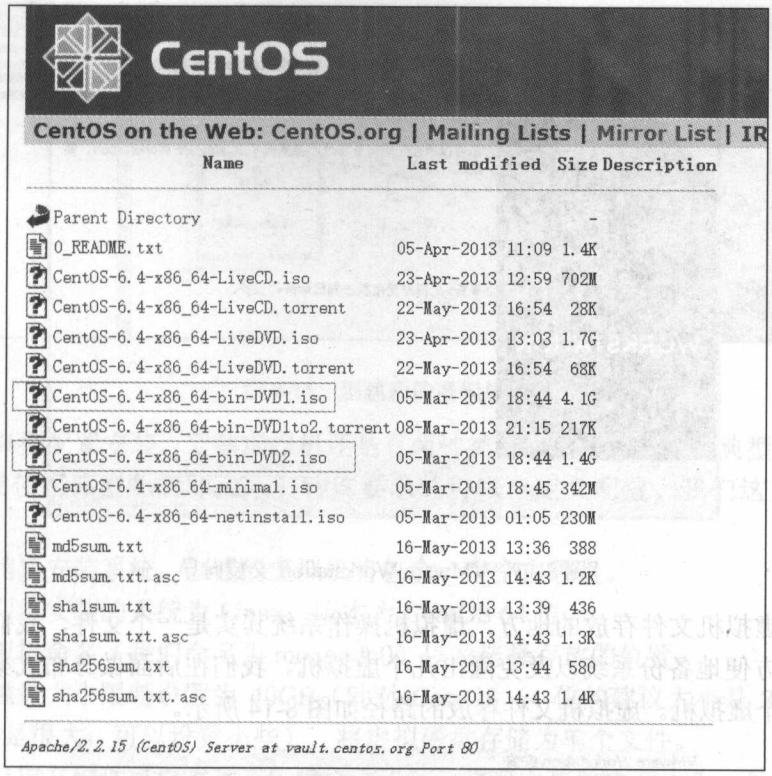

图 8-11　下载 CentOS 6.4

这里需要注意的是，只需要 CentOS-6.4-x86_64-bin-DVD1.iso 就能完成系统的安装。CentOS-6.4-x86_64-bin-DVD2.iso 是其他软件，DVD2 的作用是当你不能上网时，安装一些软件与依赖包。后面需要用什么软件我们再单独安装即可，所以也可以只下载 CentOS-6.4-x86_64-bin-DVD1.iso。下载完成的文件如图 8-12 所示。

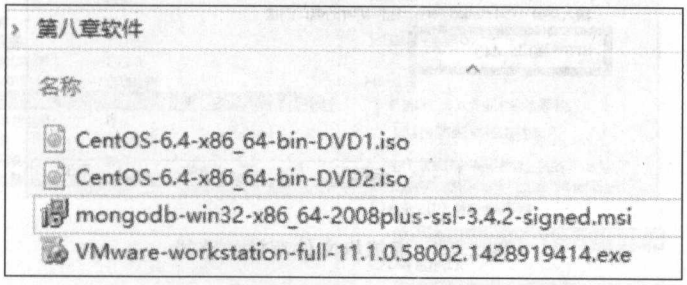

图 8-12　下载完成的文件

更多的版本选择请查看官网页面 http://www.vmware.com/cn.html 和 https://www.centos.org/download/ 。

2. 安装 VMware Workstation

下载完成后的 VMware Workstation 安装文件是 exe 执行文件，直接单击根据向导完成安装即可。VMware Workstation 安装向导如图 8-13 所示。

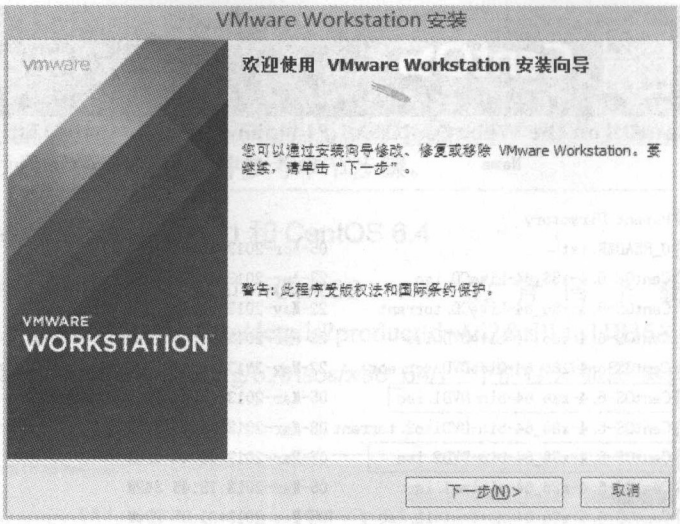

图 8-13　VMware Workstation 安装向导

需要记住虚拟机文件存放的地方，虚拟机操作系统其实是一个文件，我们只要使用这个文件就可以很方便地备份系统以及克隆出几个虚拟机。我们在后面做分布式架构学习时，就需要克隆出多个虚拟机。虚拟机文件存放的路径如图 8-14 所示。

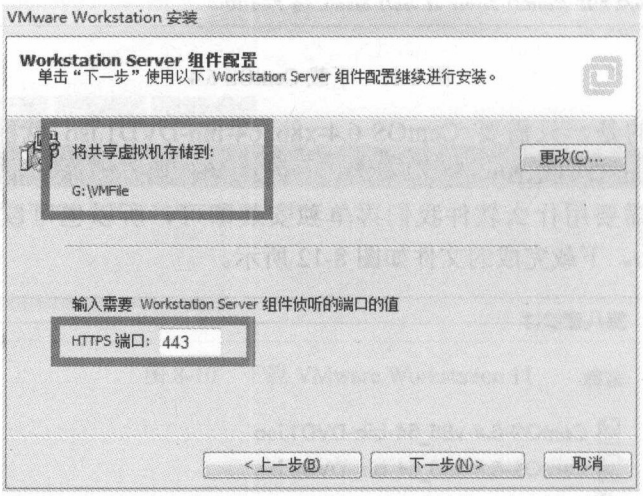

图 8-14　虚拟机文件存放的路径

3. 新建虚拟机

安装完成后就可以新建虚拟机了，打开 wmware workstation。单击创建新的虚拟机，如图 8-15 所示。具体选项的步骤如下：

图 8-15　创建新的虚拟机

（1）选择典型配置安装。新建虚拟机还是有两种类型的配置：一种是典型，一种是自定义。初学者建议使用典型配置即可，有特殊要求的可以自定义配置。我们这里选择典型配置。

（2）选择稍后安装系统，创建的虚拟机将包含一个空白硬盘。

（3）选择即将安装的系统为 Linux，版本为 CentOS 64 位。

（4）给虚拟机命名（我们命名为 mongodb0）以及选择存放的位置。

（5）配置磁盘大小限制设置为 40GB（针对 CentOS 64 位的建议大小是 20GB，如果读者的磁盘空间不是很大，可以设置小些），将虚拟磁盘存储为单个文件。

（6）单击自定义硬件可以添加或删除硬件设备，也可以更改内存大小、处理器核数以及网络等配置，我们这里配置内存为 1GB（读者可以根据自己计算机的情况设置），网络需要设置成桥接模式，才能有独立的内网 ip，如图 8-16 所示。

（7）单击完成后新建完成。

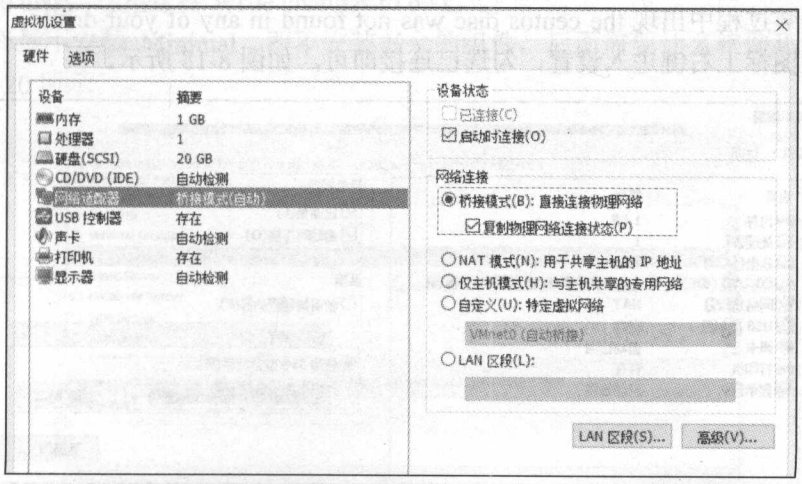

图 8-16　配置硬件和内存和网络情况

4. 虚拟机中安装 CentOS 系统

安装完的虚拟机是没有系统的，需要安装系统后才能正常使用。我们要在新建的虚拟机

中安装一个 CentOS 版本的 Linux 系统。

这里就需要我们之前下载的 CentOS 系统的 iso 安装文件，安装步骤说明如下。

单击 CD/DVD（IDE），如图 8-17 所示。

图 8-17　安装完的虚拟机界面

选择 iso 文件，浏览计算机，选择 CentOS-6.4-x86_64-bin-DVD1.iso。

勾选启动时连接，选择完后单击确定即可。

如果在安装过程中出现 the centos disc was not found in any of your drives，则在虚拟机右下角中的光盘图标上右键进入设置，勾选已连接即可，如图 8-18 所示。

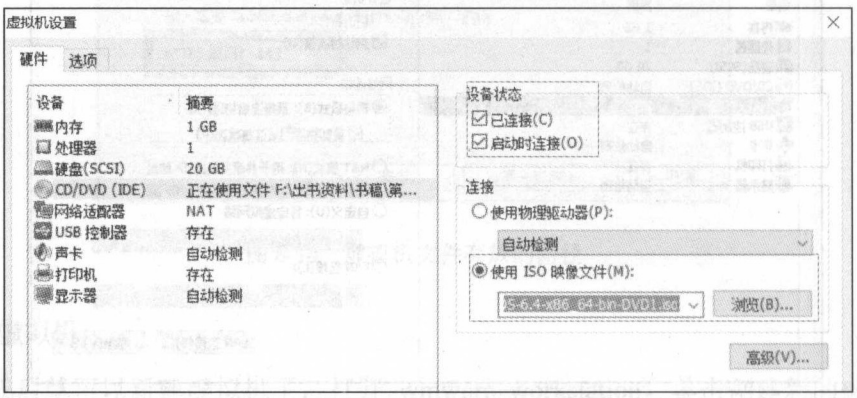

图 8-18　设置 DVD 的 IDE 为已连接

这时候看到首页 CD/DVD（IDE）已经显示正在使用文件了，我们启动虚拟机就会进入安装 CentOS 系统的界面，如图 8-19 所示，根据提示步骤安装即可。具体选项的步骤如下：

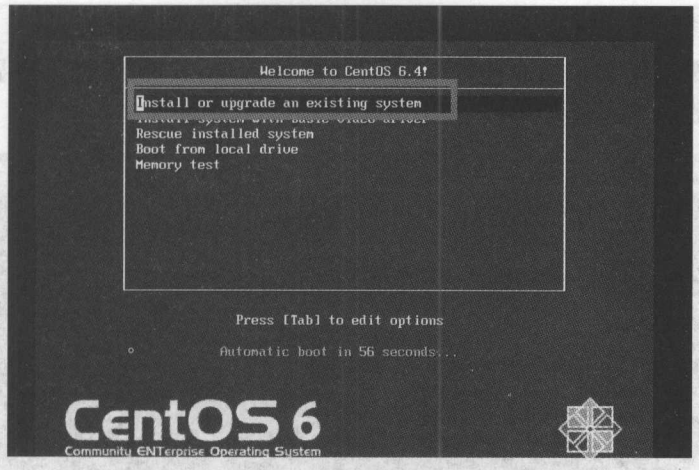

图 8-19　CentOS 安装引导界面

（1）首先选择 Install or upgrade an existing system，可能会弹出提示是否测试安装文件的完整性，单击 skip 跳过。

（2）选择 Basic Storage Devices，会提示警告 The storage device below may contain data。这是提示说磁盘内可能会有数据，询问我们是否清除，因为我们虚拟机的硬盘里没有数据，可以直接单击 yes,discard any data 就行，单击 yes 在安装过程中就会格式化虚拟机的硬盘（不影响到主机）。

（3）给虚拟机命名，我们命名为 localhost.mongodb0。

（4）然后很重要的一步，输入和确认 root 账户的密码，千万要记住这里的密码。

（5）选择替换安装 Replace Existing Linux System(s)，会提示警告 Writing storage configuration to disk，这里选择 Write changes to disk。

（6）选择最小安装 Minimal，即不安装多余的组件，后面需要什么软件我们可以再下载安装，如图 8-20 所示。

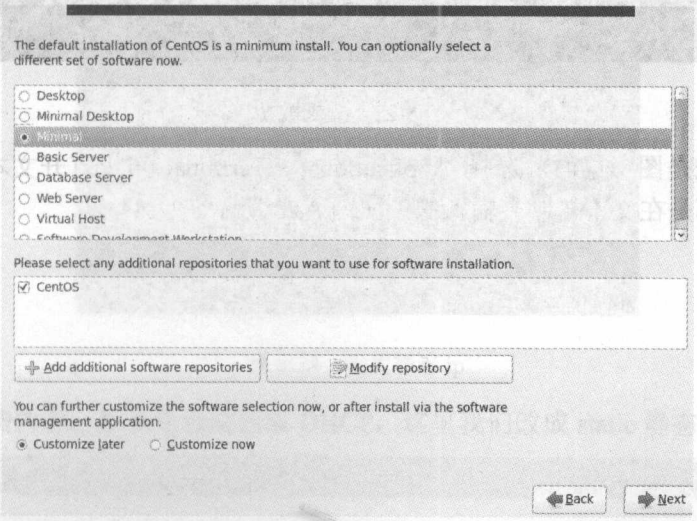

图 8-20　选择最小安装

（7）然后等待安装，过一会就安装完成了。单击重启 Reboot。

（8）重启后登录，登录时输入刚才的 root 密码，输入时屏幕上不会显示密码，输入完成后直接回车即可登录成功，如图 8-21、图 8-22 所示。

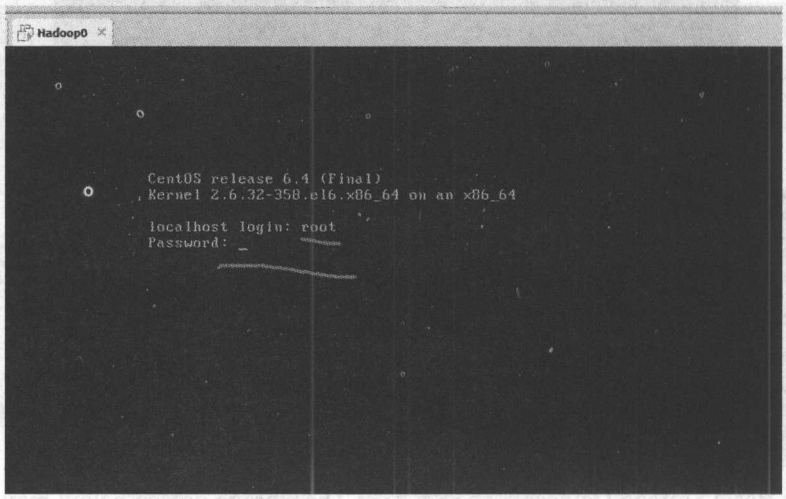

图 8-21　root 账户登录 CentOS 系统

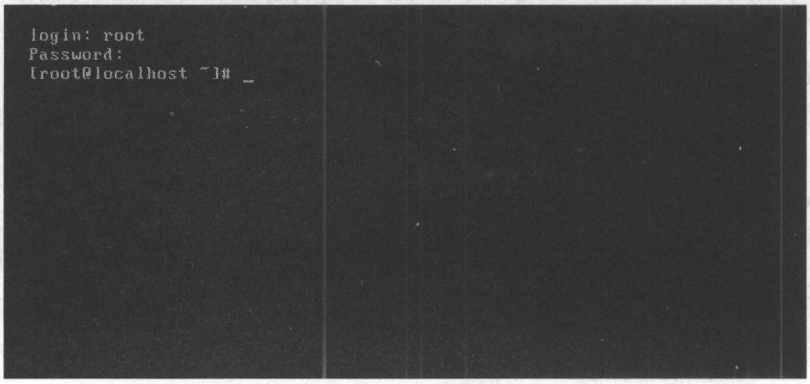

图 8-22　登录成功

如果进入系统是图形桌面，选择 Applications →Terminal 可以打开文本控制台。因为我们大多数的命令都是在文本控制台输入的，如图 8-23 所示。

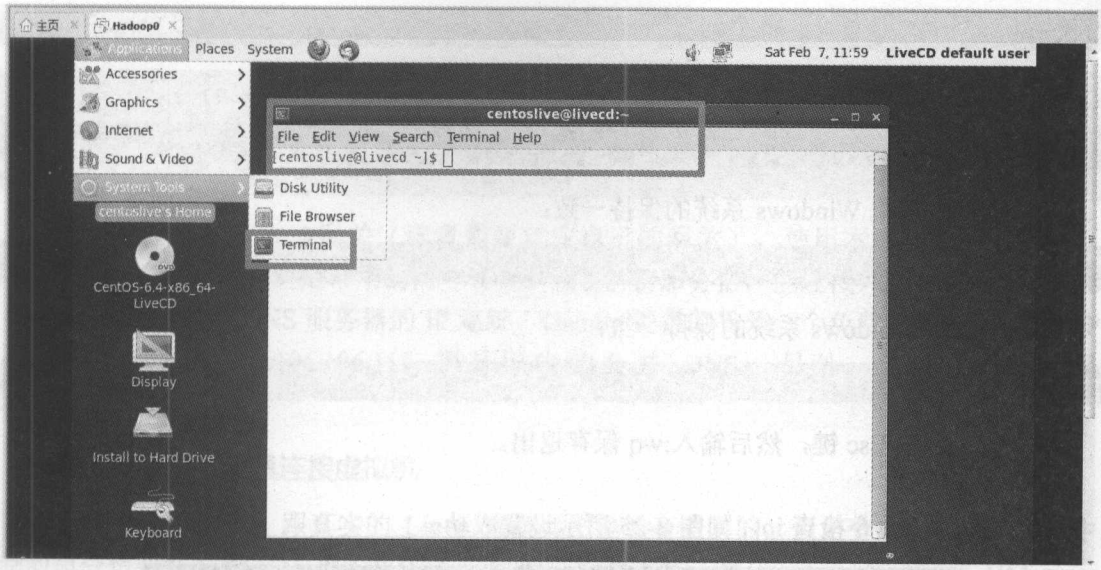

图 8-23　图形桌面打开文本控制台

5. 配置虚拟机局域网 ip

我们把虚拟机的网络模式设置为桥接模式，这个时候需要给它设置一个内网 ip，这个 ip 应该与主机的 ip 在同一个网段（前三位相同）。

例如，我这里主机 ip 是 192.168.199.217（可在 Windows 系统 cmd 窗口中使用 ipconfig 命令查看），则虚拟机可以设置为 192.168.199.X（X 表示小于等于 254 的非 217 的数值），我这里把它设置为 192.168.199.8。

使用 vi 命令编辑 Linux 系统网络配置文件，命令如下：

```
vi  /etc/sysconfig/network-scripts/ifcfg-eth0
```

修改后的内容如图 8-24 所示。

```
DEVICE=eth0
HWADDR=00:0C:29:12:73:DA
TYPE=Ethernet
UUID=af4a45e3-36eb-4bae-a2ec-5d35a988a4ae
ONBOOT=yes
NM_CONTROLLED=yes
BOOTPROTO=static
IPADDR=192.168.199.8
NETMASK=255.255.255.0
GATEWAY=192.168.199.1
```

图 8-24　设置静态 ip

需要改动一些内容。默认是自动获取 DHCP，这里我们改成 static 静态：

```
BOOTPROTO=static
```

设置网络配置重启生效需要增加的：

```
ONBOOT=yes
```

这个就是我们需要设置的 ip：

```
IPADDR=192.168.199.8
```

子网掩码与主机 Windows 系统的保持一致：

```
NETMASK=255.255.255.0
```

网关与主机 Windows 系统的保持一致：

```
GATEWAY=192.168.199.1
```

修改好之后按 Esc 键，然后输入:wq 保存退出。

重启虚拟机。

使用 ifconfig 命令检查 ip，如图 8-25 所示设置成功。

```
Kernel 2.6.32-358.el6.x86_64 on an x86_64

localhost login: root
Password:
Last login: Mon Mar 20 04:02:58 on tty1
[root@localhost ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:12:73:DA
          inet addr:192.168.199.8  Bcast:192.168.199.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe12:73da/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:162 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:12900 (12.5 KiB)  TX bytes:636 (636.0 b)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

图 8-25　设置静态 ip 成功

使用 ping 命令可以测试网络是否连通，例如百度首页的 ip 是 119.75.217.109，我们使用 ping 119.75.217.109 即可知道虚拟机是否能连通外网。

我们 Windows 系统主机的 ip 是 192.168.199.217，使用 ping 192.168.199.217 命令即可知道 Linux 系统是否能连通局域网。注意需要关闭 Windows 系统的防火墙或者将 Linux 系统的局域网 ip 设置为白名单。

6. 设置 DNS 服务器

我们在测试 Linux 系统是否能连接外网的时候发现，Linux 系统使用 ip 能够连通外网，例如 ping 119.75.217.109；但是使用域名来测试时，例如 ping www.baidu.com 会报错 ping: unknown host www.baidu.com。这是没有设置 DNS 域名解析服务器导致的。

需要编辑配置文件增加 DNS 域名服务器的配置，使用命令：

```
vi   /etc/resolv.conf
```

添加内容：

```
nameserver 192.168.199.1
nameserver 8.8.8.8
nameserver 202.106.196.115
```

192.168.199.1 对应本地网关（读者需要对应自己的网关），使用本地的 DNS 解析。8.8.8.8 和 202.106.196.115 是常用的两个 DNS 域名解析服务器，能连接外网时可用。8.8.8.8 是 Google 提供的免费 DNS 服务器的 IP 地址，Google 提供的另外一个免费 DNS 服务器的 IP 地址是：8.8.4.4 。202.106.196.115 则是国内的公共 DNS。另外一个公共 DNS 是 202.106.0.20。

7. 使用 PieTTY 工具连接虚拟机

VM 虚拟机的 Linux 跟真实的 Linux 系统几乎是没有差别的，但是有一个缺点：主机中复制的命令想要粘贴到 VM 虚拟机的 Linux 中，需要安装 VM ware Tools。安装 VM ware Tools 有些麻烦，我们这里还有另外的解决方法：使用远程登录软件操作 Linux 即可。

Pietty 是一种用于在 Windows 系统下登录到 Linux 虚拟机的远程登录软件。知道了虚拟机系统的 ip 之后，我们就可以使用 Pietty 来连接 Linux 系统。我们的 Linux 系统的 ip 是 192.168.199.8，默认的远程连接端口是 22。

连接如图 8-26 所示。

单击 open 后，首次连接会提示，单击是，然后输入账号 root 以及密码后登录成功，如图 8-27 所示。

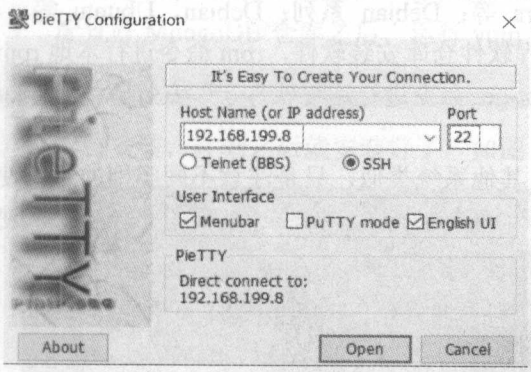

图 8-26　Pietty 登录

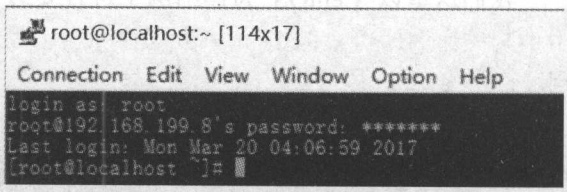

图 8-27　Pietty 登录成功

8.3.3　安装 MongoDB

MongoDB 数据库在 Linux 系统中的安装也非常简单，直接解压就可以用了。

1. 查看安装环境

Linux 系统中输入"uname -a "，可显示电脑以及操作系统的相关信息。输入"cat /proc/version"，查看正在运行的内核版本。输入"cat /etc/issue"，显示的是发行版本信息。输

入“lsb_release –a”，查看详细发行版本信息（适用于所有的 Linux，包括 Redhat、SuSE、Debian 等发行版，但是在 debian 下要安装 lsb），如图 8-28 所示。

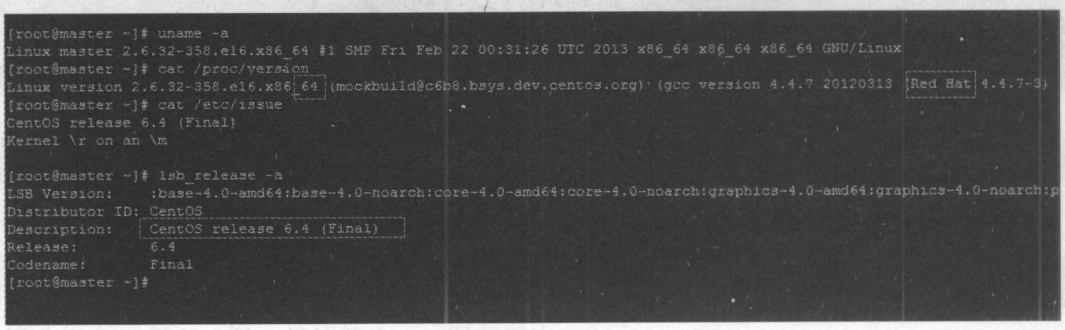

图 8-28　查看安装环境

从上图可以看到，我们的系统是 CentOS 6.4 ，64 位的系统，内核是 Red Hat。在下载 MongoDB 安装包的时候就可以下载相应版本的安装包。

2. Linux 软件仓库安装

Linux 提供了软件仓库自动安装软件服务。Linux 软件仓库是一个在线目录（所以需要能够联网才能使用仓库安装，如果计算机没有联网，则需要官方安装包来离线安装），其中包含许多软件。网络上有很多版本的 Linux 软件仓库，包括第三方 Linux 软件仓库，我们可以使用默认的软件仓库，也可以配置成第三方的软件仓库。

不同的 Linux 版本调用软件仓库安装的命令不同。一般来说著名的 Linux 系统基本上分两大类。RedHat 系列：RedHat、CentOS、Fedora 等；Debian 系列：Debian、Ubuntu 等。Red Hat 系列的 Linux 系统使用 yum 命令进行远程软件仓库安装软件，rpm 命令进行本地 rpm 安装包的安装。Debian 系列的 Linux 系统使用 apt-get 命令进行远程软件仓库安装软件，dpkg 命令进行本地 deb 安装包的安装。

我们这里以 CentOS 为详细例子进行安装，其他系统类似，只是注意不同类别的系统使用不同的命令即可，RedHat 系列的命令与 CentOS 的命令一样，Debian 系列的 Linux 系统使用 apt-get 命令。

同时我们也会给出 Debian 系列的安装步骤。更多详情可以查看官网：https://docs.mongodb.com/master/administration/install-on-linux/。

RedHat 系列使用 yum 命令之前需要先配置 MongoDB 官方仓库。

在/etc/yum.repos.d/目录下创建 mongodb.repo 文件，mongodb.repo 文件里记录 MongoDB 仓库的配置信息，步骤说明如下。

输入命令：

```
vi /etc/yum.repos.d/mongodb.repo
```

输入内容：

```
[mongodb]
name=MongoDB Repository
```

```
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-
org/3.4/x86_64/
enabled=1
gpgcheck=1
gpgkey=https://www.mongodb.org/static/pgp/server-3.4.asc
```

按 Esc 键，然后输入:wq，保存退出编辑。

name 是仓库软件的名称，名称是为了方便用户自己阅读，可以随便起。baseurl 就是我们要安装的 MongoDB 的仓库链接地址。enabled 表示这个 repo 中定义的源是启用的，0 为禁用。gpgcheck 表示这个 repo 中下载的 rpm 将进行 gpg 的校验，用来确定 rpm 包的来源是有效和安全的。gpgkey 就是用于校验的 gpg 密钥，这个要跟版本对应。当我们不启用 gpgcheck 时，就不需要 gpgkey。

我们这里安装的是 3.4 的版本。如果需要安装早期的版本，只需要把 baseurl 修改成对应的版本链接即可。当然如果启用 gpgcheck 的话，还需要对应的 gpgkey。例如我们要安装 2.6 版本的，不启用 gpgcheck，则配置文件的内容如下：

```
[mongodb]
name=MongoDB 2.6 Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/
gpgcheck=0
enabled=1
```

更多版本的 MongoDB 仓库链接可以在官网 MongoDB 仓库列表中寻找。MongoDB 仓库列表地址为 https://repo.mongodb.org/yum/redhat。

配置好 MongoDB 仓库之后，运行 yum 安装命令即可：

```
sudo yum install -y mongodb-org
```

等待安装，片刻之后就安装好了，如图 8-29 所示。

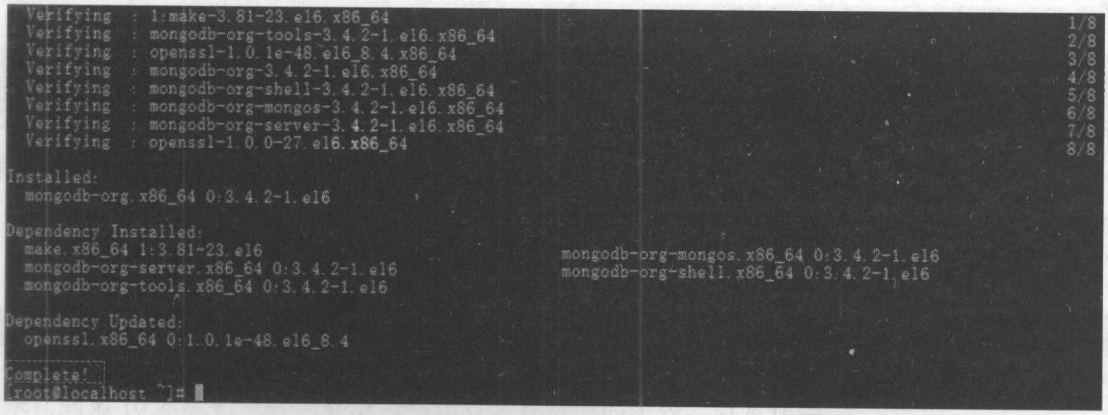

图 8-29　yum 命令安装成功

Debian 系列使用 apt-get 命令。使用 apt-get 命令安装 MongoDB 之前也需要配置安装包来源，还要增加 MongoDB 的公钥 Key 才能正确安装软件，步骤如下。

（1）添加 MongoDB 软件源，输入命令：

```
vi  /etc/apt/sources.list.d/mongodb.list。
```

输入内容需要根据 Ubuntu 的版本来决定。Ubuntu 版本的命名规则是根据正式版发行的年月命名，Ubuntu 16.04 也就意味着 2016 年 04 月发行。除此之外，每个版本的 Ubuntu 还有一个用 2 个英文单词组成的开发代号，都是动物名称组成。安装软件源的版本则需要对应一下。

Ubuntu 12.04 版本的代号是穿山甲 Precise Pangolin，则输入内容是：

```
deb [ arch=amd64 ] http://repo.mongodb.org/apt/ubuntu precise/mongodb-org/3.4
multiverse
```

Ubuntu 14.04 版本的代号是塔尔羊 Trusty Tahr，则输入内容是：

```
deb [ arch=amd64 ] http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.4
multiverse
```

Ubuntu 16.04 版本的代号是地松鼠 Xenial Xerus，则输入内容是：

```
deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-
org/3.4 multiverse
```

PS:知道了配置软件源的原理之后，这里提供了一种快速配置的方法，不需要 vi 输入，echo 和 tee 命令可以把内容输入到文件中。lsb_release -sc 则可以自动提取出版本动物名，所以以下命令可以替代上面的 vi 输入内容的方法。

直接输入命令：

```
echo "deb http://repo.mongodb.org/apt/ubuntu "$(lsb_release -sc)"/mongodb-
org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb.list
```

（2）增加 MongoDB 的公钥 Key：

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
0C49F3730359A14518585931BC711F9BA15703C6
```

MongoDB 的公钥 Key 会更新，最新的公钥只能在官网 Ubuntu 系统的安装页面中查看 https://docs.mongodb.com/master/tutorial/install-mongodb-on-ubuntu/。

（3）告诉 apt-get 扫描新增加的仓库 MongoDB 软件源。使用命令：

```
sudo apt-get update
```

安装：

```
sudo apt-get install -y mongodb-org
```

3. 官网安装包安装

仓库安装法几条命令即可完成安装，但是在 Linux 系统没有网络的情况下，或者镜像与计算机之间网络情况太差，老是安装失败的情况下，我们可以在其他有网络的 Windows 系统中下载好官网安装包之后使用 ssh 工具（例如 SSH Secure File Transfer Client）上传到 Linux 系统中（一般情况下仍然是使用 ip 和端口 22 即可连接）。安装的方法很简单，解压安装包就可以使用了。首先得根据版本选择安装包，如图 8-30 所示。

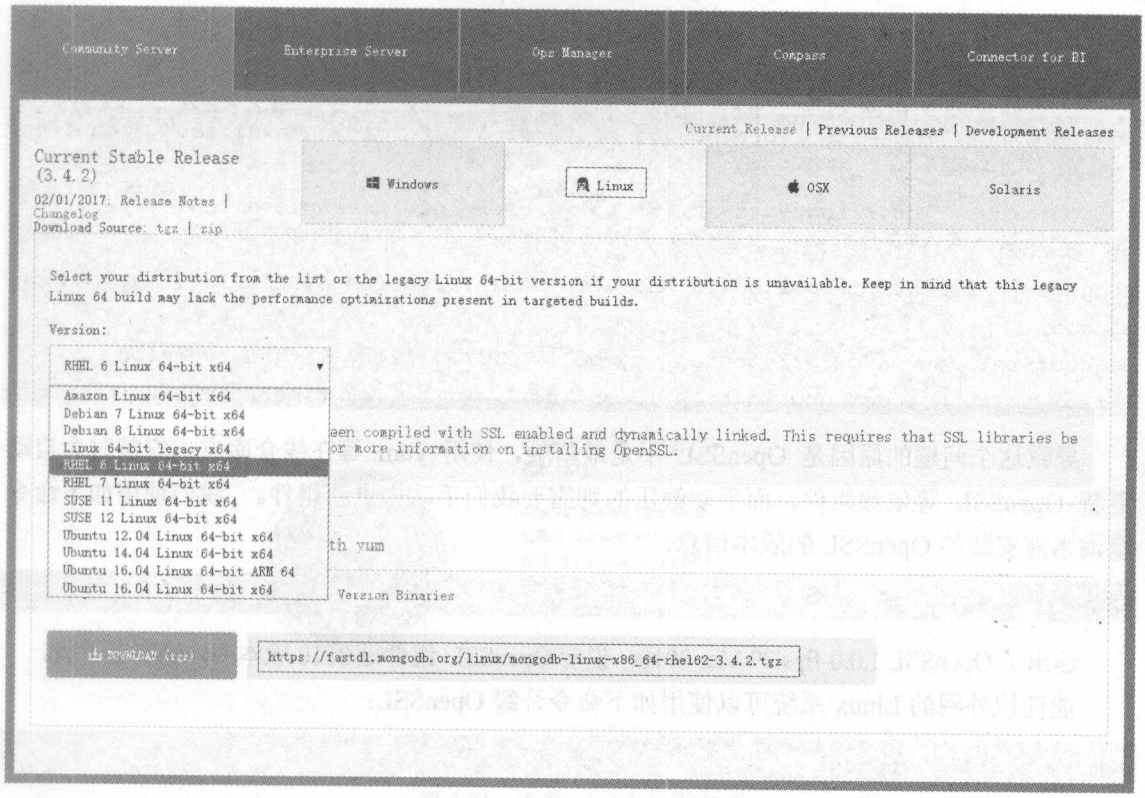

图 8-30　Linux 系统官网安装包的选择

读者可能会发现 MongoDB 没有针对 CentOS 系统的版本，如果要在 CentOS 使用官网安装包安装怎么办？我们在上面的章节已经知道 CentOS 的内核是 RedHat 的，所以 CentOS 可以用 RedHat 的，完全兼容。

例如我们下载好了 mongodb-linux-x86_64-rhel62-3.4.2.tgz 安装包。

使用命令 tar　xzf　mongodb-linux-x86_64-rhel62-3.4.2.tgz。

该命令把安装包的所有内容解压到新目录 mongodb-linux-x86_64-xxxx-yy-zz 中，该目录位于你的当前目录中。该目录将包含许多子目录和文件，包含可执行文件的目录被称作 bin 目录。到第 10 章我们就会介绍这些可执行文件对应的命令，以及作用。

解压完毕后安装完成，不再需要任何额外的操作，所以说手动安装 MongoDB 并不会花费更多的时间。不过，手动安装 MongoDB 也确实有自身的局限，因为使用 yum 等命令进行在线仓库安装会把 MongoDB 注册为服务，把可执行文件目录添加到环境变量，这样可以在

任何路径目录使用 MongoDB 服务。而手动解压的 MongoDB 只能在 bin 目录下才能执行 MongoDB 的可执行文件。当然，我们在第 9 章也会讲到怎么把 MongoDB 的 bin 目录添加到环境变量中。

另外，有网络的情况下，也可以使用官网安装包安装，可以使用 curl 命令进行下载安装包，例如选择好版本链接之后，使用如下命令下载：

```
curl -O https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel62-3.4.2.tgz。
```

然后使用 tar 命令 tar xzf mongodb-linux-x86_64-rhel62-3.4.2.tgz 解压即可。

解压官网安装包方式安装的 MongoDB 在启动时可能遇到报错：

```
mongod: /usr/lib64/libcrypto.so.10: no version information available (required
by mongod)
mongod: /usr/lib64/libcrypto.so.10: no version information available (required
by mongod)
mongod: /usr/lib64/libssl.so.10: no version information available (required by
mongod)
mongod: relocation error: mongod: symbol TLSv1_2_client_method, version
libssl.so.10 not defined in file libssl.so.10 with link time reference
```

导致这个问题的原因是 OpenSSL 不是最新版，使用 yum 等在线仓库方式安装时会自动更新 OpenSSL 等依赖组件，而手动解压的则需要我们手动去更新组件。我们使用如下命令查询当前安装的 OpenSSL 的版本信息：

```
openssl version
```

输出了 OpenSSL 1.0.0-fips 29 Mar 2010，说明 OpenSSL 组件是 2010 版本的，确实老了些。

能连接外网的 Linux 系统可以使用如下命令升级 OpenSSL：

```
yum -y install openssl
```

不能连接外网的可以去 http://mirrors.163.com/centos/6/os/x86_64/Packages/下载最新版的 OpenSSL 更新包，可以在页面上 Ctrl+F 输入 OpenSSL 查找，找到最新的包为 openssl-1.0.1e-48.el6.i686.rpm。下载好最新的.rpm 包后使用 ssh 工具上传到 Linux 系统中，进入该路径，我们可以使用如下命令升级当前的 OpenSSL：

```
rpm -Uvh  openssl-1.0.1e-48.el6.i686.rpm
```

升级完成后，再次查看当前版本信息，已经升级好了，就可以正常启动了。

如果遇到 libc.so.6 is needed by openssl-1.0.1e-48.el6.i686 错误，说明缺少相关依赖包。有外网的情况下使用如下命令可解决：

```
yum install glibc
```

如果没有外网，只能一个个手动去下载安装包，再上传到 Linux 中使用 rpm 方式安装缺少的依赖包。由此可见 yum 在线安装方式的方便，yum 命令会自动去更新下载需要的依赖

包，也体现出解压安装官网安装包版本选择的重要性，比如老的版本 MongoDB 2.0 等在 CentOS 6 系统中解压安装是能直接使用的。

8.4 Mac OSX 系统安装 MongoDB

8.4.1 查看安装环境

从 MongoDB 3.0 版本开始，MongoDB 只支持 OS X 10.7(Lion) 64 位以上的系统。

所以我们在安装 MongoDB 之前先查看一下安装环境以便选择相应的 MongoDB 版本。单击左上角的苹果菜单“关于本机”，在出现的“关于本机”窗口中，可以看到处理器型号及内存大小，单击中间的灰色字，可以查看到当前系统的版本、系统代码和本机序列号，如图 8-31 所示。

图 8-31　查看 macOS 的版本

8.4.2 官网安装包安装

Mac OSX 中使用官网安装包安装 MongoDB 也很方便，跟 Linux 系统一样只需要把官网安装包解压即可。首先在官网选择安装包，如图 8-32 所示。

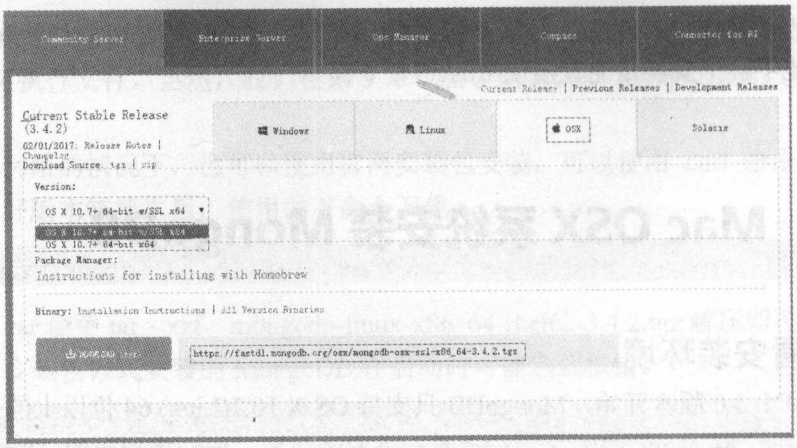

图 8-32　Mac OSX 系统官网安装包的选择

或者使用 curl 命令下载，命令如下：

```
curl -O https://fastdl.mongodb.org/osx/mongodb-osx-ssl-x86_64-3.4.2.tgz
```

下载完成后使用 tar 命令进行解压：

```
tar -zxvf mongodb-osx-ssl-x86_64-3.4.2.tgz
```

等待解压完毕后安装完成。

8.4.3　Mac 软件仓库安装

在 Mac 上也有 Homebrew 和 MacPorts 这些软件在线安装仓库，但是 MacPorts 安装 MongoDB 时，编译 MongoDB 需要的 Boost 库可能要花费数小时，这个是需要注意的地方。所以我们这里使用 Homebrew 来进行 MongoDB 软件仓库的安装。步骤如下：

更新 Homebrew 的软件仓库，在 Mac 的终端中输入命令：

```
brew update
```

然后耐心等待，过程中屏幕上没有任何显示，估计要几分钟，取决于网络的速度。然后就列出了一大堆东西，就可以进行后续步骤了。

开始安装 MongoDB，输入命令：

```
brew install mongodb
```

然后继续等待 MongoDB 下载完成。这次是有下载进度百分比的，下载完成后会自动安装，等待安装成功的提示即可。同时 Homebrew 的安装最后会自动把 MongoDB 执行文件加入到环境变量中。

更多详情可查看官网关于 Mac OSX 安装 MongoDB：

```
http://docs.mongodb.org/manual/tutorial/install-mongodb-on-os-x/
```

第 9 章

◂ 启动和停止MongoDB ▸

9.1 命令行方式启动和参数

9.1.1 Windows 系统命令行启动 MongoDB

第一步：新建一个目录用来存放 MongoDB 的数据库文件，目录路径可以自己选择，这里新建目录 F:\mongodb\data 。

第二步：按 Win+R 键，在打开的运行对话框中输入 cmd 后，回车打开 cmd 窗口，键入如下命令（每条命令输入完毕后回车执行）：

```
cd  C:\Program Files\MongoDB\Server\3.4\bin
mongod  -dbpath  “F:\mongodb\data”
```

如图 9-1 所示。

图 9-1　启动 MongoDB

命令解析：第一条命令是进入安装目录下的 bin 目录，因为 cmd 窗口一打开默认就是在 C 盘，所以如果安装目录是在 C 盘直接用 cd 即可。

如果安装在其他盘，需要 cd 过后再键入盘符加冒号才能进入。

例如安装目录是 E 盘的 MongoDB\Server\3.4.2 则启动命令如下：

```
cd  E:\MongoDB\Server\3.4.2\bin
E:
mongod  -dbpath  “F:\mongodb\data”
```

最后一行命令中的-dbpath 参数值就是我们第一步新建的文件夹。这个文件夹一定要在开启服务之前事先建立好，否则会报错，MongoDB 不会自己创建。

成功启动后 cmd 的界面如图 9-2 所示，注意这个 cmd 窗口就是我们的 MongoDB 数据库服务端，不能关闭 cmd 窗口，否则服务端也会关闭。在 Linux 服务端可以使用--fork 参数后台运行，后台运行可以关闭当前服务器窗口，MongoDB 数据库服务仍在运行。但是 Windows 系统中没有这个参数配置，所以在 Windows 系统中关闭当前服务器窗口，MongoDB 数据库服务也就停止了。

```
C:\Program Files\MongoDB\Server\3.4\bin>mongod  -dbpath  "F:\mongodb\data"
2017-03-19T16:41:19.818+0800 I CONTROL  [initandlisten] MongoDB starting : pid=6
312 port=27017 dbpath=F:\mongodb\data 64-bit host=joe
2017-03-19T16:41:19.819+0800 I CONTROL  [initandlisten] targetMinOS: Windows 7/W
indows Server 2008 R2
2017-03-19T16:41:19.819+0800 I CONTROL  [initandlisten] db version v3.4.2
2017-03-19T16:41:19.820+0800 I CONTROL  [initandlisten] git version: 3f76e40c105
fc223b3e5aac3e20dcd026b83b38b
2017-03-19T16:41:19.820+0800 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
 1.0.1u-fips  22 Sep 2016
2017-03-19T16:41:19.821+0800 I CONTROL  [initandlisten] allocator: tcmalloc
2017-03-19T16:41:19.822+0800 I CONTROL  [initandlisten] modules: none
2017-03-19T16:41:19.823+0800 I CONTROL  [initandlisten] build environment:
2017-03-19T16:41:19.823+0800 I CONTROL  [initandlisten]     distmod: 2008plus-ss
l
2017-03-19T16:41:19.824+0800 I CONTROL  [initandlisten]     distarch: x86_64
2017-03-19T16:41:19.825+0800 I CONTROL  [initandlisten]     target_arch: x86_64
2017-03-19T16:41:19.825+0800 I CONTROL  [initandlisten] options: { storage: { db
Path: "F:\mongodb\data" } }
2017-03-19T16:41:19.840+0800 I -        [initandlisten] Detected data files in F
:\mongodb\data created by the 'wiredTiger' storage engine, so setting the active
 storage engine to 'wiredTiger'.
2017-03-19T16:41:19.840+0800 I STORAGE  [initandlisten] wiredtiger_open config:
create,cache_size=5577M,session_max=20000,eviction=(threads_max=4),config_base=f
alse,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=sn
appy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),st
atistics_log=(wait=0),
2017-03-19T16:41:23.848+0800 I CONTROL  [initandlisten]
2017-03-19T16:41:23.849+0800 I CONTROL  [initandlisten] ** WARNING: Access contr
ol is not enabled for the database.
2017-03-19T16:41:23.849+0800 I CONTROL  [initandlisten] **          Read and wri
te access to data and configuration is unrestricted.
```

图 9-2　启动成功的 MongoDB 服务窗口

9.1.2　Linux 系统命令行启动 MongoDB

首先第一步仍然是新建 MongoDB 数据库文件存放的路径，这里新建/mongodb/data，使用命令如下：

```
mkdir  -p  /mongodb/data
```

-p 参数表示需要时创建上层目录，如目录早已存在则不当作错误。

然后查看 MongoDB 的启动可执行文件 mongod 在哪一个路径，使用命令：

```
whereis mongod
```

我这里是通过在线仓库 yum 安装的，找到的可执行文件的路径是/usr/bin，如果是官网安装包解压安装的话，解压路径就是安装路径。因为我们的 mongod 可执行文件的路径是/usr/bin/mongod，所以启动的命令为：

```
/usr/bin/mongod  --dbpath=/mongodb/data
```

--dbpath 参数值就是我们第一步新建的用来指定存放数据库文件的文件夹。

Linux 中想要在任何路径下不带 MongoDB 安装路径启动 MongoDB，需要把 MongoDB 的安装路径中的 bin 目录添加到环境变量。如果是通过 yum 命令等在线安装的方式安装的 MongoDB，已经自动把 MongoDB 的安装路径中的 bin 目录添加到环境变量中，所以可以在任何路径下通过以下命令启动 MongoDB：

```
mongod  --dbpath=/mongodb/data
```

如果是解压官方安装包方式安装的 MongoDB，则需要手动去把 MongoDB 的安装路径中的 bin 目录添加到环境变量中，否则会提示-bash: mongod: command not found。例如我这里解压的路径是: /usr/local/mongodb/mongodb-linux-x86_64-rhel62-3.4.2。

查看当前环境变量使用命令：

```
echo $PATH
```

Linux 设置环境变量有下面三种方式：

1. 临时设置

使用如下命令可以临时让 MongoDB 的可执行文件在任何路径下能执行，关闭 shell 终端或者新开的 shell 中无效，因为是临时设置环境变量。命令如下（需要注意命令中的:号必须是英文半角，否则报错 not a valid identifier）：

```
export  PATH=$PATH:$HOME/bin:<MongoDB 安装目录的 bin 路径>
```

我们这里使用的命令为：

```
export  PATH=$PATH:$HOME/bin:/usr/local/mongodb/mongodb-linux-x86_64-rhel62-
3.4.2/bin
```

2. 针对当前用户永久设置

例如我现在登录的用户是 joe，希望以后 joe 用户能在任何路径下不带安装目录运行 MongoDB 的执行文件。使用 vi 命令编辑 joe 用户下的.bash_profile 文件：

```
vi  /home/joe/.bash_profile
```

添加如下内容：

```
export  PATH=$PATH: $HOME/bin: <MongoDB 安装目录的 bin 路径>
```

根据我的安装目录我添加的内容是：

```
export  PATH=$PATH:$HOME/bin: /usr/local/mongodb/mongodb-linux-x86_64-rhel62-3.4.2/bin
```

使用 source 命令让配置文件马上生效：

```
source ~/.bash_profile
```

~符号表示当前用户目录，也就是~等于/home/joe。

3. 针对所有用户永久设置

我们修改当前用户目录下的.bash_profile 环境变量配置只能对当前用户生效，如果想要全局生效，则需要修改 /etc/profile 文件。使用命令：

```
vi   /etc/profile
```

添加如下内容：

```
export  PATH=$PATH: $HOME/bin: <MongoDB 安装目录的 bin 路径>
```

根据我的安装目录我添加的内容是：

```
export .PATH=$PATH:$HOME/bin: /usr/local/mongodb/mongodb-linux-x86_64-rhel62-3.4.2/bin
```

使用 source 命令让配置文件马上生效：

```
source  /etc/profile
```

添加好环境变量之后解压官网安装包方式安装的 MongoDB 也能在任何路径不带安装目录启动了。使用如下命令即可启动：

```
mongod  --dbpath=/mongodb/data
```

需要注意的是直接使用 mongod --dbpath=/mongodb/data 这样的命令在 shell 终端启动 MongoDB 服务，也会遇到跟 Windows 系统中一样的问题：当启动 MongoDB 进程的 session 窗口不小心被关闭时，MongoDB 服务也就随之停止，这毫无疑问是非常不安全的。MongoDB 在 Linux 系统中提供了后台 Daemon[15]方式启动的参数，只需要加上--fork 参数即可。但如果要使用--fork 参数，就必须启用--logpath 参数来指定 MongoDB 服务的运行日志文件。

首先新建 log 的存放路径：

```
mkdir -p /mongodb/log
```

Daemon 方式启动 MongoDB 服务命令如下：

---

[15] Daemon 程序是一直运行的服务端程序，又称为守护进程。通常在系统后台运行，没有控制终端不与前台交互，Daemon 程序一般作为系统服务使用，在系统关闭时才结束。

```
mongod  --dbpath=/mongodb/data  --logpath=/mongodb/log  --fork
```

使用 yum 方式安装的 MongoDB 已经自动注册为了 service，所以还多了一种启动方式：

```
service mongod start
```

service 启动 MongoDB 需要配置文件，yum 方式安装的配置文件在/etc/mongod.conf。关于配置文件的启动方式，我们会在 9.3 配置文件方式启动中讲解。

如果要把解压官网安装包方式安装的也注册成 service，可参考 9.6.2 小节“ Linux 系统设置 MongoDB 开机启动”的方式一。

9.1.3　Mac OS 系统命令行启动 MongoDB

第一步：在终端使用 mkdir 命令创建数据存放的目录，例如我们 MongoDB 的数据库文件保存在/data/db 中，则使用命令：

```
mkdir  -p  /data/db
```

-p 参数表示需要时创建上层目录，如果目录早已存在不当作错误。

使用 which 命令找到 MongoDB 的启动可执行文件 mongod，命令如下：

```
which mongod
```

我这里输出的是/usr/local/bin/mongod，说明可执行文件在/usr/local/bin 路径下。Homebrew 方式安装的一般都是这个路径，但这并不是 MongoDB 真正的安装路径，这是 Homebrew 在安装 MongoDB 时自动把 MongoDB 的可执行文件加入环境变量里了。

要知道 Homebrew 安装 MongoDB 在哪一个真实的路径，可以右击 Dock 中的 Finder 选中前往文件夹，输入/usr/local/bin 找到 mongod 可执行文件，右击 mongod 可执行文件，选中显示简介，可以看到 MongoDB 真实的安装路径，例如：/usr/local/Cellar/mongodb/3.4.2/bin/mongod。解压官网安装包方式安装 MongoDB 的话 MongoDB 的安装目录则是解压的路径。

我们这里 mongod 启动执行文件的路径是 /usr/local/bin/mongod 或者 /usr/local/Cellar/mongodb/3.4.2/bin/mongod，所以启动命令是：

```
/usr/local/bin/mongod  --dbpath  /data/db
```

或者

```
/usr/local/Cellar/mongodb/3.4.2/bin/mongod --dbpath  /data/db
```

--dbpath 参数值就是我们第一步新建的文件夹，指定 MongoDB 以后数据文件存放的地方。

有读者觉得，每次都带目录启动太麻烦了。Homebrew 方式安装 MongoDB 时环境变量中已经有启动文件，所以可以在任何路径下直接 mongod 命令启动，命令如下：

```
mongod  --dbpath  /data/db
```

解压官网安装包方式安装的话，也有一个方法可以不带目录启动，把 MongoDB 安装目录下的 bin 目录输出到 PATH 环境中即可。使用命令：

```
export PATH=<MongoDB 安装目录>/bin:$PATH
```

例如我这里 MongoDB 安装目录是：/usr/local/Cellar/mongodb/3.4.2，所以使用命令：

```
export PATH=/usr/local/Cellar/mongodb/3.4.2/bin:$PATH。
```

然后可以在任何路径使用：

```
mongod  --dbpath  /data/db
```

启动 MongoDB 数据库服务。

9.2 启动参数

使用 mongod 命令启动 MongoDB 数据库服务时可以设置非常多的参数。默认情况下，最重要的参数就是数据库文件存放的路径/data/db，只要/data/db 路径有访问权限就能启动 MongoDB 服务。其他参数可适当选用。

mongod 的启动参数说明如下。

1. 基本配置

- --dbpath 参数：指定数据库文件保存路径，默认为/data/db。
- --port 参数：指定服务端口号，默认端口 27017。
- --bind_ip 参数：限制监听 ip，指定多个 ip 时用逗号隔开。
- --logpath 参数：指定 MongoDB 日志文件，注意是指定文件不是目录。
- --logappend：使用追加的方式写日志。
- --fork：以守护进程的方式运行 MongoDB，创建服务器进程。
- --auth：启用验证。
- --journal：启用日志选项，MongoDB 的数据操作将会写入到 journal 文件夹的文件里。
- --journalOptions 参数：启用日志诊断选项。
- --repair：修复所有数据库。
- --repairpath 参数：修复数据库时使用的处理目录，默认是在 dbpath 路径下的_tmp 目录。
- --quiet：屏蔽部分 MongoDB 服务信息输出，只打印重要信息。
- --pidfilepath 参数：PID File 的完整路径，如果没有设置，则没有 PID 文件。
- --keyFile 参数：集群的私钥的完整路径，只在副本集架构中生效。
- --unixSocketPrefix 参数：UNIX 域套接字替代目录，（默认为/tmp）。
- --cpu：定期显示 CPU 的 CPU 利用率和 iowait。

- --diaglog 参数：创建一个非常详细的故障排除和各种错误的诊断日志记录。参数默认为 0 表示关闭；1 表示记录写操作；2 表示记录读操作；3 表示记录读写操作；7 表示记录写和一些读操作。
- --directoryperdb：设置每个数据库将被保存在一个单独的目录。
- --ipv6：启用 IPv6 选项。
- --jsonp：允许 JSONP 形式通过 HTTP 访问（有安全影响）。
- --maxConns 参数：最大同时连接数，默认 2000。
- --noauth：不启用验证，默认不启用。
- --nohttpinterface：关闭 http 接口，默认关闭 27018 端口访问。
- --noprealloc：禁用数据文件预分配（往往影响性能）。
- --noscripting：禁用脚本引擎。
- --notablescan：不允许表扫描。
- --nounixsocket：禁用 UNIX 套接字监听。
- --nssize 参数(=16)：设置数据库.ns 文件大小（MB）。
- --objcheck：在收到客户数据时，检查的有效性。
- --profile 参数：数据库分析等级设置。记录一些操作性能到标准输出或者指定的 logpath 的日志文件中，默认 0 表示关闭；1 表示开，仅包括慢操作；2 表示开，包括所有操作。
- --quota：限制每个数据库的文件数。
- --quotaFiles 参数：限制每个数据库的文件数设置，默认为 8，需要--quota 配合使用。
- --rest：开启简单的 rest API，默认关闭。
- --slowms 参数(=100)：value of slow for profile and console log。
- --smallfiles：使用较小的默认文件。
- --syncdelay 参数(=60)：数据写入磁盘的时间秒数（0=never，不推荐）。
- --sysinfo：打印一些诊断系统信息。
- --upgrade：如果需要，自动升级数据库。

2. 复制参数

- --fastsync：从一个 dbpath 里启用从库复制服务，该 dbpath 的数据库是主库的快照，可用于快速启用同步。
- --autoresync：如果从库与主库同步数据差得多，自动重新同步。
- --oplogSize 参数：设置 oplog 的大小(MB)。

3. 主从参数

- --master：主库模式。
- --slave：从库模式。
- --source 参数：从库端口号。

- --only 参数：指定单一的数据库复制。
- --slavedelay 参数：设置从库同步主库的延迟时间。

4. 副本集参数

- --replSet 参数：设置副本集名称。
- 分片参数。
- --configsvr：声明这是一个集群的 config 服务，默认端口 27019，默认目录 /data/configdb。
- --shardsvr：声明这是一个集群的分片，默认端口 27018。
- --noMoveParanoia：关闭 paranoid 模式。

上述参数都可以写入 mongod.conf 配置文档里，基本配置中的前几项比较常用，其他的参数就要用得少一些。在后面副本集和分片章节中我们会用到复制相关的参数。

9.3 配置文件方式启动

我们已经学习了如何用命令行启动 MongoDB 服务，在这个过程中需要配置我们的参数，例如数据库文件存放路径，以及日志 log 文件存放路径等。但是如果是几个同事共同管理 MongoDB，当有参数变化时，需要随时相互交流沟通，并更新启动 MongoDB 的命令行，而且命令行容易输入错误，给 MongoDB 的管理和维护造成了不便。针对这种情况，MongoDB 提供了读取启动配置文件的方式来启动数据库。在生产环境中我们也建议使用这种方式来管理启动 MongoDB。

yum 命令等在线安装方式安装的 MongoDB 会自动生成一个/etc/mongod.conf 配置文件，使用 cat 命令查看配置文件：

```
cat  /etc/mongod.conf
```

配置文件内容为：

```
# mongod.conf
# for documentation of all options, see:
# http://docs.mongodb.org/manual/reference/configuration-options/
# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log
# Where and how to store data.
storage:
  dbPath: /var/lib/mongo
```

```
  journal:
    enabled: true
#  engine:
#  mmapv1:
#  wiredTiger:
# how the process runs
processManagement:
  fork: true  # fork and run in background
  pidFilePath: /var/run/mongodb/mongod.pid  # location of pidfile
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1  # Listen to local interface only, comment to listen on all
interfaces.
#security:
#operationProfiling:
#replication:
#sharding:
## Enterprise-Only Options
#auditLog:
#snmp:
```

#表示是注释内容，用于说明配置文件的参数作用。更多的配置参数可查看官网文档 http://docs.mongodb.org/manual/reference/configuration-options/。基本日志的参数分为几大块，分别是日志配置 systemLog、存储路径配置 storage、网络配置 net，以及进程管理 processManagement。下属的详细参数可参考 9.2 节“启动参数”。

使用配置文件启动 MongoDB 的命令为：

```
mongod  --config /etc/mongod.conf
```

或者

```
mongod  -f  /etc/mongod.conf
```

这个命令在 Windows 系统、Linux 系统以及 Mac OS 系统中通用。

如果没有把 MongoDB 安装目录下的 bin 目录写入环境变量，则 mongod 命令前需要加上 MongoDB 安装目录下 bin 目录的路径。

需要注意的是，解压官网安装包方式安装的 MongoDB 没有自动生成/etc/mongod.conf 文件，需要自己生成一个。使用命令：

```
vi  /etc/mongod.conf
```

把需要的参数输入保存即可。

9.4 启动 MongoDB 客户端

我们前面已经了解到 mongo.exe 就是客户端的启动程序，只要找到 mongo 的可执行文件运行，即可进入 MongoDB 的客户端。一般来说，mongo 可执行文件与 mongod 服务端启动可执行文件在同一个目录下。我们以 Windows 系统为例，新建一个 cmd 窗口，进入安装目录下的 bin 目录中，执行 mongo 命令即可（切记需要保持服务端开启的状态下）。

```
cd  C:\Program Files\MongoDB\Server\3.4\bin
mongo
```

如图 9-3 所示，成功进入 mongodb 数据库的 shell 操作界面。

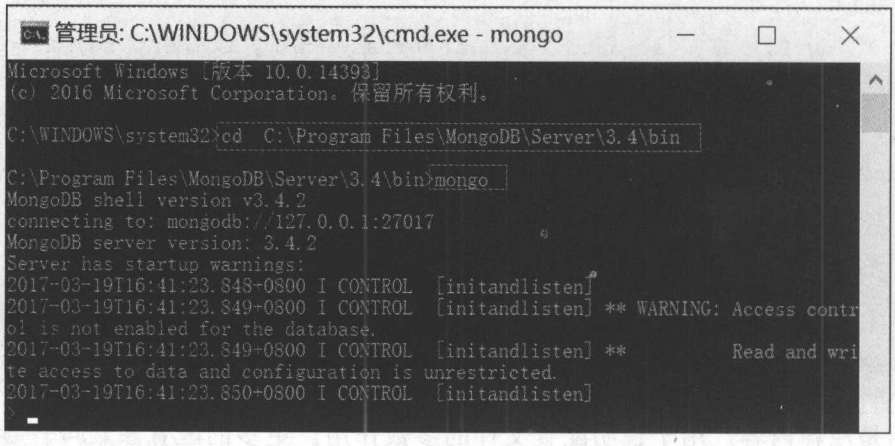

图 9-3　进入 MongoDB 的 shell 操作界面

Linux 以及其他系统也类似，找到 mongo 客户端可执行文件所在目录使用命令 mongo 即可启动。

9.5 关闭 MongoDB

9.5.1 Windows 系统设置 MongoDB 关闭

Windows 系统中 MongoDB 关闭有以下三种方式。

1. 方式一：关闭当前服务端窗口（不推荐）

我们在 9.1.1 小节已经说过 Windows 版本的 MongoDB 不支持使用--fork 参数后台运行，所以当前服务端 cmd 窗口关闭 MongoDB 就关闭，但这种方式容易导致未正常关闭 MongoDB 下次启动不起来，或是无法连接。修复方法查看 9.7 节。

2. 方式二：在当前服务端窗口使用 Ctrl+C（推荐）

Ctrl+C 命令在 cmd 命令中是终端里结束操作的意思。在当前服务端窗口使用 Ctrl+C，MongoDB 将会自己做清理退出，把没有写好的数据写完成，并最终关闭 MongoDB 服务，如图 9-4 所示。

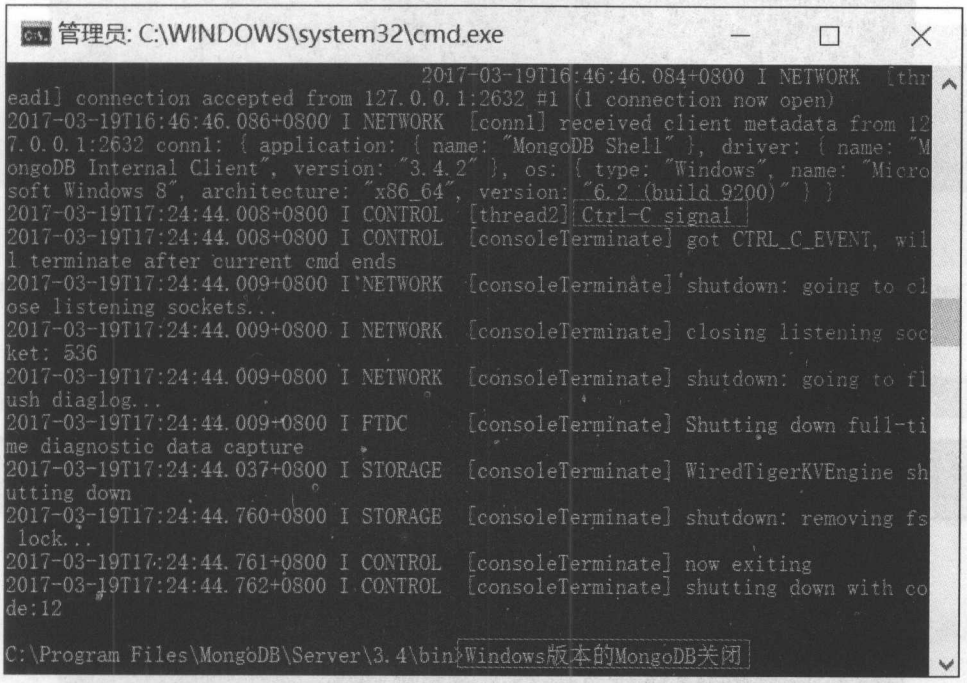

图 9-4　MongoDB 服务端使用 ctrl+c 关闭 mongodb 服务

3. 方式三：进入 mongo 客户端运行命令停止（推荐）

重新打开一个 cmd 窗口，进入 mongodb 安装目录下的 bin 目录，运行客户端。然后使用 admin 用户，使用命令 shutdownServer 关闭，如图 9-5、图 9-6 所示。完整代码如下：

```
cd  C:\Program Files\MongoDB\Server\3.4\bin
mongo
use admin
db.shutdownServer();
```

```
C:\Program Files\MongoDB\Server\3.4\bin>mongo
MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-19T17:31:00.288+0800 I CONTROL  [initandlisten]
2017-03-19T17:31:00.288+0800 I CONTROL  [initandlisten] ** WARNING: Access contr
ol is not enabled for the database.
2017-03-19T17:31:00.289+0800 I CONTROL  [initandlisten] **          Read and wri
te access to data and configuration is unrestricted.
2017-03-19T17:31:00.290+0800 I CONTROL  [initandlisten]
> use admin
switched to db admin
> db.shutdownServer();
2017-03-19T17:32:25.740+0800 I NETWORK  [thread1] Socket recv() 远程主机强迫关闭
了一个现有的连接。 127.0.0.1:27017
2017-03-19T17:32:25.740+0800 I NETWORK  [thread1] SocketException: remote: (NONE
):0 error: 9001 socket exception [RECV_ERROR] server [127.0.0.1:27017]
server should be down...
2017-03-19T17:32:25.743+0800 I NETWORK  [thread1] trying reconnect to 127.0.0.1:
27017 (127.0.0.1) failed
2017-03-19T17:32:30.742+0800 W NETWORK  [thread1] Failed to connect to 127.0.0.1
:27017 after 5000ms milliseconds, giving up.
2017-03-19T17:32:30.743+0800 I NETWORK  [thread1] reconnect 127.0.0.1:27017 (127
.0.0.1) failed failed
```

图 9-5　MongoDB 客户端关闭 MongoDB 服务

```
2017-03-19T17:31:52.869+0800 I -        [conn1] end connection 127.0.0.1:3784 (1
 connection now open)
2017-03-19T17:32:10.570+0800 I NETWORK  [thread1] connection accepted from 127.0
.0.1:3794 #2 (1 connection now open)
2017-03-19T17:32:10.572+0800 I NETWORK  [conn2] received client metadata from 12
7.0.0.1:3794 conn2: { application: { name: "MongoDB Shell" }, driver: { name: "M
ongoDB Internal Client", version: "3.4.2" }, os: { type: "Windows", name: "Micro
soft Windows 8", architecture: "x86_64", version: "6.2 (build 9200)" } }
2017-03-19T17:32:24.000+0800 I COMMAND  [conn2] terminating, shutdown command re
ceived
2017-03-19T17:32:24.000+0800 I NETWORK  [conn2] shutdown: going to close listeni
ng sockets...
2017-03-19T17:32:24.001+0800 I NETWORK  [conn2] closing listening socket: 484
2017-03-19T17:32:24.002+0800 I NETWORK  [conn2] shutdown: going to flush diaglog
...
2017-03-19T17:32:24.002+0800 I FTDC     [conn2] Shutting down full-time diagnost
ic data capture
2017-03-19T17:32:25.002+0800 I STORAGE  [conn2] WiredTigerKVEngine shutting down

2017-03-19T17:32:25.722+0800 I STORAGE  [conn2] shutdown: removing fs lock...
2017-03-19T17:32:25.722+0800 I CONTROL  [conn2] now exiting
2017-03-19T17:32:25.723+0800 I CONTROL  [conn2] shutting down with code:0
2017-03-19T17:32:25.723+0800 I CONTROL  [initandlisten] shutting down with code:
0

C:\Program Files\MongoDB\Server\3.4\bin>
```

图 9-6　MongoDB 服务端服务被客户端关闭

9.5.2　Linux 系统设置 MongoDB 关闭

如果是在 Shell 中启动的 MongoDB 服务并且没有带--fork 后台运行，则可以使用下面介绍的方式一关闭 MongoDB 服务。

1. 方式一：在当前服务端窗口使用 Ctrl+C

Ctrl+C 命令在 Shell 界面里也是结束操作的意思。在当前 MongoDB 服务端 Shell 窗口使用 Ctrl+C，MongoDB 将会自己做清理退出，把没有写好的数据写完成，并最终关闭

MongoDB 服务。

2. 方式二：进入 mongo 客户端运行命令停止（推荐）

打开一个 Shell 窗口，进入 mongo 执行文件所在的 bin 目录，运行客户端。然后使用 admin 用户，使用命令 shutdownServer 关闭。完整代码如下：

```
cd  /usr/bin/
mongo
use admin
db.shutdownServer();
```

3. 方式三：kill 命令停止进程

使用 kill 命令停止 MongoDB 数据库实例，必须现找到实例的进程，使用命令（如图 9-7 所示）：

```
ps aux|grep mongod
```

```
[root@localhost ~]# ps aux|grep mongod
mongod    1796 16.2  3.4 319780 35112 ?        Sl   22:45   0:00 /usr/bin/mongod -f /etc/mongod.conf
root      1816  0.0  0.0 103236   856 pts/1    S+   22:45   0:00 grep mongod
[root@localhost ~]#
```

图 9-7　查看 MongoDB 实例进程

我们看到执行启动 mongod 命令的进程号 PID 是 1796，这个就是 MongoDB 数据库实例的进程，然后可以使用 kill -2 PID 或者 kill -15 PID。我这里使用命令：

```
kill -2 1796
```

kill 命令带数字参数，数字表示信号声明，详情可用 kill –l 查看。-2 和-15 都会等 MongoDB 处理完事情释放相应资源后再停止。kill -9 则是马上停止进程，不要用 kill -9 来杀死 MongoDB 进程，否则很可能导致 MongoDB 的数据损坏。

4. 方式四：如果 MongoDB 已经自动注册为 service，会多一种关闭方式

```
service mongod stop
```

9.5.3　Mac OS 系统设置 MongoDB 关闭

进入 mongo 客户端，运行命令停止。

打开一个终端窗口。进入 mongo 执行文件所在的 bin 目录，运行客户端。然后使用 admin 用户，使用命令 shutdownServer 关闭。完整代码如下：

```
mongo
use admin
db.shutdownServer();
```

9.6 设置 MongoDB 开机启动

9.6.1 Windows 系统设置 MongoDB 开机启动

Windows 系统设置 MongoDB 开机启动只要把 MongoDB 服务写入 Windows 服务中，设置为自动启动即可。

9.1.1 小节已经说过 Windows 系统的 MongoDB 不支持使用--fork 参数后台运行，所以当前服务端 cmd 窗口关闭 MongoDB 就关闭。但是服务端 cmd 窗口很容易不小心单击到了关闭。那怎么才能让 Windows 系统的 MongoDB 也能在后台运行呢。有一个方法就是把 MongoDB 服务写入 Windows 服务，而且写入 Windows 服务之后，MongoDB 就能实现自动启动了，也就是开机启动。

操作步骤如下：

步骤 01 以管理员身份打开 cmd 窗口，这一步非常重要，否则会报错。

步骤 02 进入 MongoDB 安装目录下的 bin 目录。

```
cd  C:\Program Files\MongoDB\Server\3.4\bin
```

输入：

```
mongod -logpath  "F:\mongodb\log\mongodb.log" -logappend  -dbpath
"F:\mongodb\data"   --serviceName  "mongodbService"   --install
```

步骤 03 在 cmd 窗口中输入 services.msc 回车，打开服务窗口，就可以在服务窗口中找到【mongodbService】服务

注意写入 Windows 服务时，必须使用--serviceName 参数给该服务起个名字，我们这里起名为 mongodbService。

同时必须使用-logpath 参数配置日志文件，日志文件上级的目录必须提前创建好。我这里在 F 盘创建了 mongodb 目录和在 mongodb 目录下创建了 log 目录，如图 9-8 所示。

```
C:\Program Files\MongoDB\Server\3.4\bin>mongod -logpath  "F:\mongodb\log\mongodb
.log" -logappend  -dbpath   "F:\mongodb\data"   --serviceName  "mongodbService"
  --install

C:\Program Files\MongoDB\Server\3.4\bin>services.msc

C:\Program Files\MongoDB\Server\3.4\bin>_
```

图 9-8 把 MongoDB 服务写入 windows 服务

把 MongoDB 服务写入 Windows 服务之后，把它的启动类型设置为自动，MongoDB 就能实现开机自启动了，如图 9-9 所示。

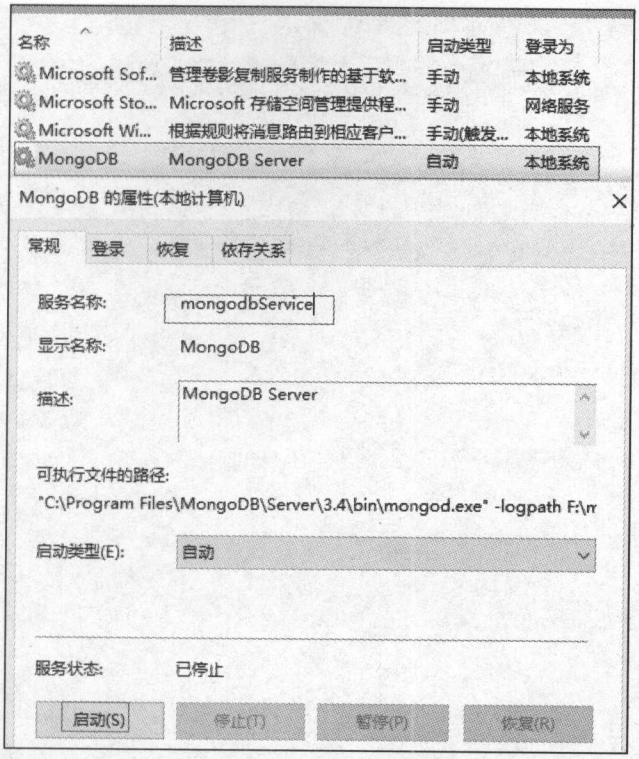

图 9-9　Windows 服务中的 MongoDB 服务

写入 Windows 服务的 MongoDB 服务启动和停止的方式：在 Windows 服务中找到它右击，选择属性，选择启动和停止。

如果要从系统服务中卸载 MongoDB 服务，以管理员身份进入 cmd 窗口 MongoDB 的 bin 目录，输入命令：mongod.exe --remove --serviceName "mongodbService"，出现“Service successfully removed.”提示，表示移除服务成功。或者以管理员身份进入 cmd 窗口输入：sc delete "服务名称"。注意：服务名称要写在英文状态的双引号中。

9.6.2　Linux 系统设置 MongoDB 开机启动

1. 方式一：在/etc/init.d 目录下创建脚本（推荐）

使用 yum 方式安装的 MongoDB 已经自动注册为 service，service 会开机启动。如果是解压官网安装包安装的 MongoDB，则需要在/etc/init.d 目录下创建 service 脚本才能实现注册为 service，进而实现开机启动。

init.d 目录是一个很重要的目录，包含许多系统各种服务的启动和停止脚本。使用 init.d 目录下的脚本，你需要有 root 权限或 sudo 权限。使用如下命令创建 MongoDB 的 service 脚本。

```
vi  /etc/init.d/mongodb
```

输入内容：

```
# cp ssh mongodb
# vim mongodb
# cat mongodb
#!/bin/bash
#
# mongod        Start up the MongoDB server daemon
#
# source function library
. /etc/rc.d/init.d/functions
#定义命令
CMD=/usr/local/mongodb/bin/mongod
#定义数据目录
DBPATH=/usr/local/mongodb/data
#定义日志目录
LOGPATH=/usr/local/mongodb/log/mongo.log
start()
{
    #fork 表示后台运行
    $CMD --dbpath=$DBPATH --logpath=$LOGPATH --fork
    echo "MongoDB is running background..."
}
stop()
{
    pkill mongod
    echo "MongoDB is stopped."
}
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    *)
        echo $"Usage: $0 {start|stop}"
esac
```

这里的 CMD 和 DBPATH 以及 LOGPATH 分别对应 mongod 可执行文件的目录、数据库文件存放的路径和日志文件存放的路径。

也可以使用配置文件的方式注册为 service。配置文件如下：

```
vi  /etc/mongod.conf
```

输入内容：

```
#代表端口号，如果不指定则默认为27017
#port=27027
#MongoDB 数据文件目录
dbpath=/usr/local/mongodb/data
#MongoDB 日志文件目录
logpath=/usr/local/mongodb/log/mongo.log
#日志文件自动累加
logappend=true
则/etc/init.d/mongodb 的内容为：# vim mongodb
# cat mongodb
#!/bin/bash
#
# mongod        Start up the MongoDB server daemon
#
# source function library
. /etc/rc.d/init.d/functions
#定义命令
CMD=/usr/local/mongodb/bin/mongod
#定义配置文件路径
INITFILE=/etc/mongod.conf
start()
{
#&表示后台启动，也可以使用--fork 参数
#或者在配置文件中使用 fork=true
    $CMD -f $INITFILE &
    echo "MongoDB is running background..."
}

stop()
{
    pkill mongod
    echo "MongoDB is stopped."
}
```

```
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    *)
        echo $"Usage: $0 {start|stop}"
esac
```

按 Esc 键，输入:wq，保存/etc/init.d/mongodb 后，需要给/etc/init.d/mongodb 添加执行权限。使用命令：

```
chmod +x /etc/init.d/mongodb
```

现在，就可以使用 service 命令来控制 MongoDB 了。

启动 MongoDB 使用命令：

```
service mongodb start
```

或者使用命令：

```
/etc/init.d/mongodb start
```

停止 MongoDB 使用命令：

```
service mongodb  stop
```

或者使用命令：

```
/etc/init.d/mongodb  stop
```

到此我们已经把 MongoDB 注册为 service 了。实现开机启动使用如下代码：

```
update  -rc.d  mongodb  defaults
```

删除开机启动使用命令：

```
update  -rc.d  -f  mongodb  remove
```

2. 方式二：写入/etc/rc.local 配置

rc.local 也是 Linux 系统中经常使用的一个脚本。Linux 系统开机后会启动里面的程序，因此可以安全地在里面添加你想在系统启动之后执行的脚本。使用命令：

```
vi /etc/rc.local
```

输入内容：

```
/usr/local/mongodb/bin/mongod --dbpath=/usr/local/mongodb/data --
logpath=/usr/local/mongodb/log/mongo.log --logappend --port=27017 --fork
```

或者配置文件启动方式：

```
/usr/local/mongodb/bin/mongod  --config  /etc/mongod.conf
```

注意上面的 dbpath 文件夹需要创建，--logpath 的参数值是文件不是文件夹，不需要手动创建日志文件，但需要创建上层目录。读者在尝试时需要与自己的环境路径相对应。

9.6.3　Mac OS 系统设置 MongoDB 开机启动

Mac OS 系统可以通过 Plist 文件设置 MongoDB 开机启动。Plist 文件是以.plist 为结尾的文件的总称。众所周知，Plist 在 Mac OS X 系统中起着举足轻重的作用，就如同 Windows 系统里面的注册表一样，系统和程序使用 Plist 文件来存储自己的安装、配置、属性等信息。

Homebrew 方式安装的 MongoDB 会产生一个启动项配置 Plist 文件，一般位于 mongod 可执行文件的上一级 bin 文件夹所在的目录中，名称为 homebrew.mxcl.mongodb.plist，我们可以直接使用这个 plist 文件，也可以新建一个。总之需要把 plist 文件中的参数修改成跟我们期望的配置一致。

在修改 Plist 文件之前需要准备好使用的启动配置文件 mongod.conf。

编辑启动配置文件 mongod.conf 之前先找到 mongod 可执行文件所在的目录。使用命令：

```
which mongod
```

我这里输出的是/usr/local/bin/mongod，说明可执行文件在/usr/local/bin 路径下，Homebrew 方式安装的一般都是这个路径，但这并不是 MongoDB 真正的安装路径，这是 Homebrew 在安装 MongoDB 时自动把 MongoDB 的可执行文件加入环境变量里了。

要知道 Homebrew 安装 MongoDB 在哪一个真实的路径，可以右击 Dock 中的 Finder，选中前往文件夹，输入/usr/local/bin，找到 mongod 可执行文件，右击 mongod 可执行文件，选中显示简介，可以看到 MongoDB 真实的安装路径，例如我这里是：

```
/usr/local/Cellar/mongodb/3.4.2/bin/mongod。
```

解压官网安装包方式安装 MongoDB 的话，MongoDB 的安装目录则是解压的路径。

创建数据库文件存放的路径为/data/db，使用命令：

```
mkdir -p /data/db
```

创建日志文件存放的路径为/data/log，使用命令：

```
mkdir -p /data/log
```

设置文件夹权限，并输入用户密码：

```
sudo chmod -R 777 /data
```

编辑新建 mongod.conf 文件，使用命令：

```
vi  /data/mongod.conf
```

输入内容：

```
dbpath=/data/db  #数据库路径
logpath=/data/log/mongodb.log   #日志输出文件路径
logappend=true   #日志输出方式
```

按 Esc 键，输入:wq，保存退出。

新建 mongodb.plist 文件，使用命令：

```
vi  /data/mongodb.plist
```

输入内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
 <key>Label</key>
 <string> mongodb </string>
 <key>ProgramArguments</key>
 <array>
  <string>/usr/local/Cellar/mongodb/3.4.2/bin/mongod </string>
  <string>run</string>
  <string>--config</string>
  <string>/data/mongod.conf </string>
 </array>
 <key>RunAtLoad</key>
 <true/>
 <key>KeepAlive</key>
 <true/>
 <key>WorkingDirectory</key>
 <string>/usr/local/Cellar/mongodb </string>
 <key>StandardErrorPath</key>
 <string>/data /output.log</string>
 <key>StandardOutPath</key>
 <string>/data /output.log</string>
</dict>
</plist>
```

其中的关键字解释：

（1）Label（必选）

该项服务的名称。

（2）Program（ProgramArgument 是必选的，在没有 ProgramArgument 的情况下，必须要包含 Program 关键字）

指定可执行文件的路径和名称。

（3）RunAtLoad（可选）

标识 launchd 在加载完该项服务之后立即启动路径指定的可执行文件。默认值为 false。设置为 true 即可实现开机运行脚本文件。

（4）WorkingDirectory（可选）

运行可执行文件之前，指定当前工作目录的路径。

（5）KeepAlive（可选）

这个关键字可以用来控制是否让可执行文件持续运行，默认值为 false。当设置值为 ture 时，表明无条件地开启可执行文件，并使之保持在整个系统运行周期内，一旦 Service 异常崩溃，系统会自动重启服务。

（6）StartCalendarInterval（可选）

该关键字可以用来设置定时执行可执行程序，可使用 Month、Day、Hour、Minute 等子关键字，它可以指定脚本在多少月、天、小时、分钟、星期几等时间上执行，若缺少某个关键字则表示任意该时间点，类似于 UNIX 的 Crontab 计划任务的设置方式，比如在该例子中设置为每天 11 点钟执行脚本文件。

所有 key 关键字详细使用说明，可以在 Mac OS X 终端下使用命令 man launchd.plist 查询。

（7）StandardErrorPath

Service 错误日志文件路径。

（8）StandardOutPath

Service 日志文件路径。

然后把 mongodb.plist 文件复制到系统启动目录下，注意目录~/Library/LaunchDaemons/和/Library/LaunchDaemons/的区别（LaunchAgents 目录同理），~/Library/LaunchDaemons/是用户目录下的启动目录，/Library/LaunchDaemons/是根目录下的启动目录，我们这里把启动 plist 文件放到根目录下。

开机启动分为两种：

（1）在用户登录前启动。 plist 文件放置在目录：/Library/LaunchDaemons。

（2）在用户登录后启动。plist 文件放置在目录：/Library/LaunchAgents 。

我们这里选择在用户登录前启动 MongoDB，所以语句为：

```
cp  /data/mongodb.plist  /Library/LaunchDaemons/
```

最后使用 root 权限启动服务：

```
sudo launchctl  load -w /Library/LaunchDaemons/mongodb.plist
```

这样就实现了开机启动。

如果要关闭服务，使用命令：

```
sudo launchctl unload -w  /Library/LaunchDaemons/mongodb.plist
```

若发现以下错误：

```
Path had bad permissions
```

是因为文件的权限不够，将权限修改为 root，执行以下命令，再执行启动服务：

```
sudo chown root /data/mongodb.plist
```

9.7 修复未正常关闭的 MongoDB

MongoDB 如果未正常关闭，会导致无法启动。

1. 现象

MongoDB 服务无法启动，弹出框报错：Windows 无法启动 MongoDB 服务 错误 1067：进程意外终止。在事件查看器中可以看到该错误：MongoDB 服务因无法创建另一个系统信号灯，服务特定错误而停止。

2. 解决方法

进入 MongoDB 安装目录/data/，将此文件夹下的 mongod.lock 删除，使用该方法 MongoDB 服务可以启动起来。Linux 系统可以使用 find / -name 'mongod.lock'查找路径，数据方面不会受到影响。mongod.lock 文件是 MongoDB 服务端启动后在硬盘中创建的一个锁文件，如果你正常退出 MongoDB 服务，该文件即使还存在，也不会影响下一次启动 MongoDB 服务。

这个文件还会记录 MongoDB 在运行过程中的一些状态，以便在正常重新启动服务时能够获取异常信息提示。

删除 lock 文件之后，如果有损坏文档，需要使用 mongod --repair 命令修复一次，再正常启动。修复数据库的实际过程很简单：将所有的文档导出后马上导入，忽略无效的文档，完成后会重建索引。因为所有数据都要验证，所有索引都要重建，数据量大的话，会花费较多时间。数据量大的情况下，临时修复目录所在磁盘空间一定要大，否则会出现磁盘容量不够的提示，无法修复。

第 10 章

基本命令

学会启动 MongoDB 数据库之后，我们就可以开始尝试使用 MongoDB 了。MongoDB 提供了 Shell 界面客户端让我们可以输入命令，在 mongo 可执行文件的目录下使用 mongo 命令进入客户端。

10.1 数据库常用命令

（1）查看命令提示

```
db.help();
```

（2）切换/创建数据库

```
use mydb;
```

切换到名称为 mydb 的数据库，如果该数据库不存在，则会自动新建。

MongoDB 中默认的数据库为 test，如果你没有创建新的数据库就执行集合或者文档操作，数据将存放在 test 数据库中。

还有一种切换数据库的方法是，进入 mongo 客户端时就指定数据库名，例如：

```
mongo mydb
```

（3）查询所有数据库

```
show dbs;
```

（4）删除当前使用数据库

```
db.dropDatabase();
```

（5）从指定主机上克隆数据库

```
db.cloneDatabase("192.168.199.9");
```

将指定机器上的同名数据库的数据克隆到当前数据库，例如当前是 test 数据库，命令会将 192.168.199.9 服务器中的 test 数据库克隆到本机的 test 数据库中。

（6）从指定的机器上复制指定数据库数据到某个数据库

```
db.copyDatabase("mydb", "temp", "192.168.199.9");
```

将 192.168.199.9 的 mydb 的数据复制到本机 temp 数据库中。

（7）修复当前数据库

```
db.repairDatabase();
```

该命令不仅能整理碎片还可以回收磁盘空间，但是需要注意的是 repairDatabase 期间会产生锁，建议关闭应用后再进行此操作。repairDatabase 所需要的磁盘剩余空间需求很大，所以一般生产环境比较少使用这个命令。

（8）查看当前使用的数据库

```
db.getName();
```

或者

```
db;
```

db 和 db.getName()方法是一样的效果，都可以查询当前使用的数据库。

（9）显示当前 db 状态

```
db.stats();
```

该命令显示数据库的统计信息，包括集合数量、平均文档大小、数据大小、索引数量和大小等。

（10）当前 db 版本

```
db.version();
```

（11）查看当前 db 的链接机器地址

```
db.getMongo();
```

（12）查询之前的错误信息

```
db.getPrevError();
```

（13）清除错误记录

```
db.resetError();
```

10.2 集合

（1）创建一个集合

```
db.createCollection("mycoll");
```

或者带参数创建固定集合：

```
db.createCollection("log", {size: 20, capped:true, max: 100});
```

在执行文档写入操作时，如果集合不存在，则会自动创建集合，所以一般比较少使用创建集合命令，除非是创建固定集合。

- capped：是否启用集合限制，有两种选择：true 或 false，为 true 时启用限制创建为固定集合；如果开启需要制定一个限制条件，默认为 false，不启用固定集合。
- size：限制集合使用空间的大小，默认为没有限制。
- max：集合中最大条数限制，默认为没有限制。
- autoIndexId：是否使用_id 作为索引，有两种选择：true 或 false，默认为 true，使用_id 作为索引。

注意，size 的优先级比 max 要高。

（2）显示当前数据库中的集合

```
show collections;
```

或者

```
db.getCollectionNames();
```

（3）使用集合

```
db.mycoll
```

或者

```
db.getCollection("mycoll")
```

注意当集合名称为全数字时不能使用 db.mycoll 集合的方式，例如 db.123 会报错，可以使用 db.getCollection("123")。

（4）查看集合命令帮助文档

```
db.mycoll.help();
```

（5）查询当前集合的数据条数

```
db.mycoll.count();
```

（6）查看集合数据大小

```
db.mycoll.dataSize();
```

显示出的数字的单位是字节，因此如果需要转换单位为 KB 需要除以 1024。

（7）查看集合索引大小

```
db.mycoll.totalIndexSize();
```

（8）为集合分配的空间大小，包括未使用的空间

```
db.mycoll.storageSize();
```

（9）显示集合总大小，包括索引和数据的大小和分配空间的大小

```
db.mycoll.totalSize();
```

（10）显示当前集合所在的 db

```
db.mycoll.getDB();
```

（11）显示当前集合的状态

```
db.mycoll.stats();
```

（12）集合的分片版本信息

```
db.mycoll.getShardVersion();
```

（13）集合重命名

```
db.mycoll.renameCollection("users");
```

或者

```
db.getCollection("mycoll").renameCollection("users");
```

将 mycoll 重命名为 users，集合名为纯数字时，只能使用 db.getCollection("mycoll").renameCollection 这种方式。

（14）显示当前 db 所有集合的状态信息

```
db.printCollectionStats();
```

（15）删除当前集合

```
db.mycoll.drop();
```

10.3 文档

（1）写入文档

```
db.user.insert({"name":"joe"});
```

或者

```
db.user.save({"name":"joe"});
```

保存文档{"name":"joe"}到集合 user。如果 user 集合不存在则会自动新建。文档应该满足 BSON 格式。

save 与 insert 的区别在于不仅有写入数据功能还具有更新数据功能。

使用 save 时，如果数据库中已经有这条数据，则会更新它；如果没有，则写入。

使用 insert 时，如果数据库中已经有这条数据，则会报错 E11000 duplicate key error collection；如果没有，则写入。

数据库是否已经有这条数据是通过_id 字段去判断的。

也就是说 save 保存文档，如果文档带有_id 字段时，会找到集合有这个_id 的数据，更新该数据成新的文档。save 和 insert 的区别如图 10-1 所示。

```
> db.user.save({"_id" : ObjectId("579036a9de4344710224234d"), "myName" : "jay" })
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("579036a9de4344710224234d")
})
> db.user.insert({"_id" : ObjectId("579036a9de4344710224234d"), "myName" : "jay" })
WriteResult({
        "nInserted" : 0,
        "writeError" : {
                "code" : 11000,
                "errmsg" : "E11000 duplicate key error collection: user.user index: _id_ dup key: { : ObjectId('579036a9de4
344710224234d') }"
        }
})
> db.user.save({"_id" : ObjectId("579036a9de4344710224234d"), "myName" : "joe" })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.user.find();
{ "_id" : ObjectId("579036a9de4344710224234d"), "myName" : "joe" }
>
```

图 10-1　save 和 insert 的区别

（2）查看文档

```
db.user.find();
```

更多查询方式请查看 10.5 节“基本查询”、10.6 节“条件查询”、10.7 节“特定类型查询”和 10.8 节“高级查询$where”。

（3）更新文档

MongoDB 使用 save() 和 update()方法来更新集合中的文档。

save()通过传入的文档来替换已有文档，根据_id 对应已有文档。

```
db.user.save({"_id" : ObjectId("579036a9de4344710224234d"), "myName" : "joe",
"age" : 20})
```

update() 方法用于更新已存在的文档，语法格式如下：

```
db.collection.update(
   查询条件,
   整个文档或者修改器,
    upsert: boolean,
    multi: boolean 或者 multi 文档,
    writeConcern: 异常信息等级
  )
```

参数说明：

查询条件是传入文档的部分信息让我们定位到需要修改的文档，查询条件语句与 find 中查询方式一致。

第二个参数是整个文档或者修改器。当参数为整个文档时，传入的文档替换已有文档。当参数是修改器时，会根据修改器的种类只做相应的改动。更多修改器参考 10.10 节“修改器”。

upsert 参数可选，这个参数的意思是，如果不存在 update 的记录，是否写入新文档，true 为写入，默认是 false，不写入。

multi 参数可选，mongodb 默认是 false，只更新找到的第一条记录，如果这个参数为 true，就把按条件查出来的多条记录全部更新。

writeConcern 参数可选，抛出异常的级别。作用是保障更新操作的可靠性。

WriteConcern 的几种抛出异常的级别参数如下：

- WriteConcern.NONE：没有异常抛出。
- WriteConcern.NORMAL：仅抛出网络错误异常，没有服务器错误异常（默认）。
- WriteConcern.SAFE：抛出网络错误异常、服务器错误异常；并等待服务器完成写操作。
- WriteConcern.MAJORITY：抛出网络错误异常、服务器错误异常；并等待一个主服务器完成写操作。
- WriteConcern.FSYNC_SAFE：抛出网络错误异常、服务器错误异常；写操作等待服务器将数据刷新到磁盘。
- WriteConcern.JOURNAL_SAFE：抛出网络错误异常、服务器错误异常；写操作等待服务器提交到磁盘的日志文件。
- WriteConcern.REPLICAS_SAFE：抛出网络错误异常、服务器错误异常；等待至少 2 台服务器完成写操作。

update()使用示例：

```
db.user.update({"myName" : "joe"},{$set:{"age" : 20,"company":
"google"}},true,{multi:true},WriteConcern.SAFE);
db.user.update({"myName" : "joe"},{$set:{"age" : 20,"company":
"google"}},true,true,WriteConcern.SAFE);
db.user.update({"myName" : "joe"},{$set:{"age" : 20,"company":
"google"}},true,WriteConcern.SAFE);
db.user.update({"myName" : "joe"},{$set:{"age" : 20,"company":
"google"}},{multi:1});
```

查询 myName 是 joe 的文档，把它的 age 字段修改为 20，company 字段修改为 google。第一个 true 是 upsert 的值，表示不存在该文档时写入新文档。

{multi:true}表示如果找到多条记录，更新多条记录，也可以使用{multi:1}或者直接使用 true。multi 直接使用 true 时需要注意必须在 upsert 参数之后。

WriteConcern.SAFE 表示抛出网络错误异常、服务器错误异常；并等待服务器完成写操作。

（4）删除文档

MongoDB remove()函数是用来移除集合中的文档，必须带查询条件。

remove() 方法的基本语法格式如下：

```
db.collection.remove(
   查询条件,
   justOne: boolean
)
```

如果你的 MongoDB 是 2.6 版本以后的，语法格式如下：

```
db.collection.remove(
   查询条件,
    justOne: boolean,
    writeConcern: 异常信息等级
)
```

参数说明：

查询条件是传入文档的部分信息，让我们定位到需要删除的文档，查询条件语句与 find 中查询方式一致。

justOne 参数（可选），默认为 false，如果设为 true 或 1，则查询到多个文档时只删除一个文档。

writeConcern 参数可选，抛出异常的级别。

使用示例：

```
db.user.remove({"myName" : "joe"},1);
db.user.remove({"myName" : "joe"});
```

（5）更新文档并返回文档

```
db.user.findAndModify({
    query: {age: {$gte: 25}},
    sort: {age: -1},
    update: {$set: {name: 'a2'}, $inc: {age: 2}}
});
```

或者

```
db.runCommand({ findandmodify : "user",
    query: {age: {$gte: 25}},
    sort: {age: -1},
   update: {$set: {name: 'a2'}, $inc: {age: 2}}
});
```

（6）删除文档并返回文档

```
db.user.findAndModify({
    query: {age: {$gte: 25}},
    sort: {age: -1},
    remove: true
});
```

或者

```
db.runCommand({ findandmodify : "user",
    query: {age: {$gte: 25}},
    sort: {age: -1},
    remove: true
});
```

（7）查询满足条件的文档数量

```
db.user.count({$or: [{age: 14}, {age: 28}]});
```

10.4 索引

（1）创建索引

```
db.user.ensureIndex({age: 1});
db.user.ensureIndex({myName: 1, age: -1});
```

一个集合可以创建单个索引，也可以创建复合索引，1 表示该字段索引升序排序，-1 表示降序排序。

复合索引例如{myName: 1, age: -1}，表示 myName 先按升序排序，myName 同组的文档按 age 降序排序。

例如我们有数据：

```
{"myName":"joe",age:14}
{"myName":"ad",age:14}
{"myName":"ad",age:38}
{"myName":"ad",age:24}
{"myName":"ab",age:14}
```

使用 db.user.ensureIndex({myName: 1, age: -1})建立索引后，索引中的数据组织为：

```
{"myName":"ab",age:14}
{"myName":"ad",age:38}
{"myName":"ad",age:24}
{"myName":"ad",age:14}
{"myName":"joe",age:14}
```

对某个键创建索引会加速对该键的查询，然而，对于其他查询可能没有帮助。所以创建索引是需要构思的，需要与自己的业务相结合。

创建索引还可以搭配一些参数：

```
db.test.ensureIndex({"username":1},{"background":true})
```

{"background":true}表示在后台模式下创建索引，这样不会阻塞数据库的其他操作。

```
db.test.ensureIndex({"userid":1},{"unique":true})
```

{"unique":true}表示创建唯一索引，这时数据库中的 userid 是唯一的，没有重复值。如果插入 userid 重复的数据会报错。

如果在创建唯一索引时已经存在了重复项，我们可以通过 dropDups 参数帮助我们在创建唯一索引时消除重复文档，仅保留发现的第一个文档，如：

```
db.test.ensureIndex({"userid":1},{"unique":true,"dropDups":true})
```

已经设置了唯一索引的字段 userid 是不能重复的，即可是空值，也只能有一条 userid 为空的数据，如果我的业务需要存放很多 userid 为空的数据呢？我们可以使用 sparse 参数来设置稀疏索引：

```
db.test.ensureIndex({"userid":1},{"unique":true,"sparse":true})
```

{"sparse":true }表示 userid 为空值或者 userid 不存在时该文档都不进入索引。

（2）查询集合所有索引

```
db.user.getIndexes();
```

（3）查看集合总索引记录大小

```
db.user.totalIndexSize();
```

（4）读取当前集合的所有 index 信息

```
db.user.reIndex();
```

（5）删除指定索引

```
db.user.dropIndex("myName");
```

（6）删除集合所有索引

```
db.user.dropIndexes();
```

10.5 基本查询

下面讲述 MongoDB 中的最基本的查询。

10.5.1 find 简介

MongoDB 使用 find()来进行文档的查询，然后以非结构化的方式来显示返回的文档。例如：

```
db.user.find();
```

如果需要结构化显示返回的文档可以加上 pretty()方法，如下所示：

```
db.user.find().pretty();
```

结构化显示查询文档如图 10-2 所示。

```
> db.user.find();
{ "_id" : ObjectId("58d9b6afa159504ca6c572e0"), "myName" : "joe", "age" : 28, "company" : "google", "name" : "a2" }
{ "_id" : ObjectId("58d9ba2d6097167df6313438"), "myName" : "joe", "age" : 14 }
{ "_id" : ObjectId("58d9ba2d6097167df6313439"), "myName" : "ad", "age" : 14 }
{ "_id" : ObjectId("58d9ba2d6097167df631343a"), "myName" : "ad", "age" : 38 }
{ "_id" : ObjectId("58d9ba2d6097167df631343b"), "myName" : "ad", "age" : 24 }
{ "_id" : ObjectId("58d9ba2e6097167df631343c"), "myName" : "ab", "age" : 14 }
> db.user.find().pretty();
{
        "_id" : ObjectId("58d9b6afa159504ca6c572e0"),
        "myName" : "joe",
        "age" : 28,
        "company" : "google",
        "name" : "a2"
}
{
        "_id" : ObjectId("58d9ba2d6097167df6313438"),
        "myName" : "joe",
        "age" : 14
}
{
        "_id" : ObjectId("58d9ba2d6097167df6313439"),
        "myName" : "ad",
        "age" : 14
```

图 10-2　结构化显示查询文档

find()括号中可以设置两个参数，以逗号分隔识别。两个参数都必须是文档，第一个参数是查询条件，第二个参数则指定返回的字段，_id 默认返回。

例如：

```
db.user.find({"myName":"joe"},{"age":1})
```

查询 user 集合中 myName 为 joe 的文档，且只返回 age 字段。

括号外可以跟查询辅助方法，对查询的数量和跳过多少条数据进行设置。

更多查询条件的用法和查询辅助方法我们将在下面的章节讲解。

10.5.2　游标

游标是一种容器，可以用来存放 find 执行结果。而放入游标中的数据无论是单条还是包括多条数据结果集，每次都只能提取一条数据。

游标一般用于遍历数据集。通过 hasNext()判断是否有下一条数据，next()获取下一条数据。

例如：

```
var cursor= db.user.find();
while(cursor.hasNext()){
var temp=cursor.next()
print(temp.myName);
}
```

游标还实现了迭代器接口，所以可以使用 forEach。

```
var cursor= db.user.find();
cursor.forEach(function(temp){
print(temp.myName);
});
```

游标遍历如图 10-3 所示。

```
> var cursor= db.user.find();
>
> while(cursor.hasNext()){
... var temp=cursor.next()
... print(temp.myName);
... }
joe
joe
ad
ad
ad
ab
> var cursor= db.user.find();
>
> cursor.forEach(function(temp){
... print(temp.myName);
... });
joe
joe
ad
ad
ad
ab
>
```

图 10-3　游标遍历

10.6 条件查询

find()的第一个参数是查询条件文档，查询条件文档需要满足 BSON 格式，MongoDB 提供了很多种查询方式。

10.6.1 与操作

```
db.user.find({"myName":"joe","age":16})
```

查询出同时满足 myName 为 joe 和 age 为 16 的文档。

10.6.2 或操作$or

```
db.user.find({$or: [{age: 14}, {age: 28}]})
```

查询出 age 为 14 或者 age 为 28 的文档，满足其中一个条件即可。

10.6.3 大于$gt

```
db.user.find({age: {$gt: 20}})
```

查询 age 大于 20 的文档。

10.6.4 小于$lt

```
db.user.find({age: {$lt: 20}})
```

查询 age 小于 20 的文档。

10.6.5 大于等于$gte

```
db.user.find({age: {$gte: 20}})
```

查询 age 大于等于 20 的文档。

10.6.6 小于等于$lte

```
db.user.find({age: {$lte: 20}})
```

查询 age 小于等于 20 的文档。

10.6.7 类型查询$type

$type 操作符用来查询文档中字段与指定类型匹配的数据，并返回结果。

指定类型表格如表 10-1 所示。

表 10-1　数据类型与$type 参数对应表

Type	Number	Alias	Notes
Double	1	“double”	
String	2	“string”	
Object	3	“object”	
Array	4	“array”	
Binary data	5	“binData”	
Undefined	6	“undefined”	Deprecated.
ObjectId	7	“objectId”	
Boolean	8	“bool”	
Date	9	“date”	
Null	10	“null”	
Regular Expression	11	“regex”	
DBPointer	12	“dbPointer”	Deprecated.
JavaScript	13	“javascript”	
Symbol	14	“symbol”	Deprecated.
JavaScript (with scope)	15	“javascriptWithScope”	
32-bit integer	16	“int”	
Timestamp	17	“timestamp”	
64-bit integer	18	“long”	
Decimal128	19	“decimal”	New in version 3.4.
Min key	-1	“minKey”	
Max key	127	“maxKey”	

使用方式如下：

```
db.user.find( { "myName" : { $type : 2 } } );
db.user.find( { "myName" : { $type : "string" } } );
```

查询 myName 字段为 String 类型的文档。

10.6.8　是否存在$exists

```
db.user.find({"age": {$exists: true}})
```

查询 age 字段存在的文档。

10.6.9　取模$mod

```
db.user.find({"age": {$mod : [10, 0]}});
```

查询 age 取模 10 等于 0 的数据。

10.6.10　不等于$ne

```
db.user.find({ "age" : { "$ne" : 23}})
```

查询 age 不等于 23 的数据。

10.6.11 包含$in

```
db.user.find({ "myName" : { "$in" : [ "joe" , "ab"]}})
```

查询 myName 的值被包含在["joe" , "ab"]数组中的文档。

10.6.12 不包含$nin

```
db.user.find({ "myName" : { "$nin" : [ "joe" , "ab"]}})
```

查询 myName 的值不被包含在["joe" , "ab"]数组中的文档。

10.6.13 $not: 反匹配

以上所有字段查询操作都能取非，比如：

```
db.user.find({ "myName" : { "$in" : [ "joe" , "ab"]}})
db.user.find({ "myName" : {$not:{ "$in" : [ "joe" , "ab"]}}})
```

10.7 特定类型查询

我们在条件查询中学习的操作都是针对普通数据类型的字段进行的查询。本小节学习一些特定类型的查询。

10.7.1 null

null 在 MongoDB 中表示为空，也就是没有这个字段或者该字段为 null 类型时。相关查询如下：

```
db.user.find({"company":null})
```

查询 company 字段为 null 的文档。

```
db.user.find({"company":{$nin:[null]} })
```

查询 company 字段不为空的文档。

10.7.2 正则查询（模糊查询）

之前学习的条件查询都是精确查询或者范围查询，如果我们对要查的字段的值记不清楚了，可以使用模糊查询。MongoDB 中可以使用正则查询来达到模糊查询的效果。

1. 什么是正则表达式

正则表达式，又称规则表达式，英语为 Regular Expression，在代码中常简写为 regex、regexp 或 RE，计算机科学的一个概念。正则表通常被用来检索、替换那些符合某个模式（规

则）的文本，也就是符合规则的文本会被匹配到。

2. 正则表达式的基本符号和例子

正则表达式的用法博大精深，我们这里只能简单举几个例子，感兴趣的读者可以自己深入学习正则表达式。

常用的元字符的代码及说明：

- .: 匹配除换行符以外的任意字符。
- \w: 匹配字母、数字、下划线、汉字。
- \s: 匹配任意的空白符。
- \d: 匹配数字。
- \b: 匹配单词的开始或结束。
- ^: 匹配字符串的开始。
- $: 匹配字符串的结束。

常用的限定符的代码/语法及说明如下：

- *: 重复零次或更多次。
- +: 重复一次或更多次。
- ?: 重复零次或一次。
- {n}: 重复 n 次。
- {n,}: 重复 n 次或更多次。
- {n,m}: 重复 n 到 m 次。

常用的反义代码/语法及说明：

- \W: 匹配任意不是字母，数字，下划线，汉字的字符。
- \S: 匹配任意不是空白符的字符。
- \D: 匹配任意非数字的字符。
- \B: 匹配不是单词开头或结束的位置。
- [^x]: 匹配除了 x 以外的任意字符。
- [^aeiou]: 匹配除了 aeiou 这几个字母以外的任意字符。

例子：

- \S+匹配不包含空白符的字符串。
- <a[^>]+>匹配用尖括号括起来的以 a 开头的字符串。

例子：

- "^J": 表示以"J"开始的字符串，比如"Joe"，"Jay"等。
- "of despair$": 表示以"of despair"结尾的字符串。
- "^abc$": 表示开始和结尾都是"abc"的字符串，也就是"abc"自己了。

- "notice": 表示任何包含"notice"的字符串。

方括号表示某些字符允许在一个字符串中的某一特定位置出现：

- "[ab]": 表示一个字符串有一个"a"或"b"（相当于"a|b"）。
- "[a-d]": 表示一个字符串包含小写的'a'到'd'中的一个（相当于"a|b|c|d"或者"[abcd]"）。
- "^[a-zA-Z]": 表示一个以字母开头的字符串。
- "[0-9]%": 表示一个百分号前有一位的数字。
- ",[a-zA-Z0-9]$": 表示一个字符串以一个逗号后面跟着一个字母或数字结束。

3. MongoDB 中的正则用法

MongoDB 使用//表示启用正则表达式，如下：

```
db.user.find({"name":/^j/})
```

查询 name 字段以 j 开头的文档。

10.7.3 嵌套文档

BSON 格式的文档是可以相互嵌套的，例如如下文档 phone 字段的值就是一个子文档：

```
{
"name" : "huangz",
"phone" : { "home" : 123321,
        "mobile" : 15820123123}
}
```

1. 精确匹配查询

指定完整的文档，查询出子文档完全匹配指定文档的文档。

```
db.user.find({"phone":{"home" : 123321,"mobile" : 15820123123}})
```

查询出子文档为{"phone":{"home" : 123321,"mobile" : 15820123123}}的文档。

2. 点查询

如果我们不知道子文档的完整文档，只知道子文档中一个字段的值，可以通过点查询。

```
db.user.find({"phone.home":123321})
```

查询 phone 子文档的 home 值为 123321 的文档。

10.7.4 数组

文档中某个字段的值可能是数组，以下是数组的查询方式。

1. 数组单元素查询

```
db.user.find({favorite_number:6});
```

查询 favorite_number 数组包含数值 6 的文档。

2. $all 数组多元素查询

$all 操作符表示字段的值完全包含指定数组。

```
db.user.find({favorite_number : {$all : [6, 8]}});
```

查询 favorite_number 字段完全包含数组[6, 8]的文档。

{name: 'David', age: 26, favorite_number: [6, 8, 9] } 完全包含数组[6, 8]，可以查询到。

{name: 'David', age: 26, favorite_number: [6, 7, 9] } 不包含[6, 8]，则不符合查询条件。

3. $size 数组长度查询

```
db.user.find({favorite_number: {$size: 3}});
```

查询 favorite_number 字段的值数组长度为 3 的文档。

4. $slice 返回数组子集

通过$slice 作为限定参数可以只返回数组的部分数据。

```
db.user.find({},{favorite_number: {$slice: 2}});
db.user.find({},{favorite_number: {$slice: -2}});
```

{favorite_number: {$slice: -2}}需要放在第二个参数作为限定参数，而不是放在第一个参数。

$slice:2 表示只返回 favorite_number 数组的前两个元素。

$slice:-2 表示只返回 favorite_number 数组的后两个元素。

5. 精确匹配查询

指定完整的数组，查询出完全匹配指定数组的文档。

```
db.user.find({favorite_number :[6, 8]});
```

查询出 favorite_number 的数组等于[6, 8]的文档。

6. 点查询

点查询用于查询更复杂的数组，例如数组中包含的是子文档的情况：

```
{
 "name" : "joe",
 "phone" :[ { "home" : 123321,
              "mobile" : 1854046352},
```

```
        { "home" : 123652,
            "mobile" : 15820123123} ,
        { "home" : 123456,
            "mobile" : 13820123123}
    ]
}
```

需要查询 phone 数组中子文档的 home 值为 123456 的文档，使用命令：

```
db.user.find({"phone.home":123456});
```

7. 索引查询

数组都有索引，例如[6,8]，6 是第 0 个元素，8 是第 1 个元素（数组索引以 0 开头）。要查找某个元素指定值的文档可以使用点和索引值：

```
db.user.find({"favorite_number.0":6});
```

查询 favorite_number 数组第 0 个元素是 6 的文档。

注意 favorite_number.0 需要加双引号，否则会报错。

点查询中只要数组的子文档里有一个 home 值满足查询值就会返回文档。如果我们要精确到第几个元素也可以用索引查询。

```
db.user.find({"phone.2.home":123456});
```

查询第 2 个元素的 home 值是 123456 的文档。

8. 元素查询$elemMatch

数组的子文档如果有多个字段，查询出子文档同时满足两个条件的文档有两种方式：

```
db.user.find({"phone.home":123456,"phone.mobile":13820123123});
```

或者

```
db.user.find( {
        phone: {
          $elemMatch: {
              home :123456,
            mobile: 13820123123
}
            }
        } )
```

10.8 高级查询$where

细心的读者会发现我们在上面小节所使用的查询条件都是键值对的 BSON 格式，如果是很复杂的查询条件，需要构造的 BSON 就会很复杂，而且有些查询条件由于 BSON 格式的显示是无法表达的。MongoDB 提供了一个$where 操作器，利用这个$where 可以执行任意的 JavaScript 作为查询条件的一部分，这样 MongoDB 就几乎能完成所有的查询需求。

10.8.1 JavaScript 语言简介

JavaScript 一种直译式脚本语言，是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为 JavaScript 引擎，是浏览器的一部分，广泛用于客户端的脚本语言，最早是在 HTML 网页上使用，用来给 HTML 网页增加动态功能。JavaScript 脚本语言同其他语言一样，有它自身的基本数据类型、表达式和算术运算符及程序的基本程序框架。JavaScript 提供了 4 种基本的数据类型和 2 种特殊数据类型用来处理数据和文字，而变量提供存放信息的地方，表达式则可以完成较复杂的信息处理。JavaScript 代码不进行预编译就可以运行。

10.8.2 JavaScript 编程简单例子

```
db.user.find().forEach(function(item){
    if(item.age>18){
    item.tag="adult";
   }
  db.user.save(item);
})
```

这段 JavaScript 脚本可以在 mongo 中直接运行，意思是遍历 user 集合中的文档，如果文档中 age 字段数值大于 18，则给文档增加一个 tag 字段，tag 值设置为 adult。

10.8.3 JavaScript 与$where 结合使用

查询 age > 18 的记录，以下 4 条命令作用都一样。

```
db.user.find({age: {$gt: 18}});
```

或者

```
db.user.find({$where: "this.age > 18"});
```

或者

```
db.user.find("this.age > 18");
```

或者

```
f = function() {return this.age > 18}; db.user.find(f);
```

10.9 查询辅助

查询辅助主要是跟在 find()方法之后，对数据的查询给出一些限定条件。

10.9.1 条数限制 limit

```
db.user.find().limit(2);
```

只查出 user 集合前 2 条数据。

10.9.2 起始位置 skip

```
db.user.find().skip(3).limit(5);
```

跳过前 3 条数据，从第 4 条记录开始，返回 5 条记录（limit(5)）。

10.9.3 排序 sort

```
db.user.find().sort({age: 1});
```

查询文档并按 age 升序返回数据。

```
db.user.find().sort({age: -1});
```

查询文档并按 age 降序返回数据。

10.10 修改器

update()更新文档时，第二个参数可以是完整的文档，也可以是修改器。MongoDB 提供了多种修改器给我们使用。

10.10.1 $set

$set 用于指定修改的字段和值。

```
db.user.update({"name":"joe"},{$set:{"age":18,"company":"google"}});
```

把 name 为 joe 的文档 age 字段值修改为 18，company 字段的值修改为 google。

10.10.2　$unset

$unset 用于取消字段，也就是去掉文档中的某个字段。

```
db.user.update({"name":"joe"},{$unset:{"company":1}});
```

把 name 为 joe 的文档 company 字段去掉。

10.10.3　$inc

$inc 用于增加或减少数值。

```
db.user.update({"name":"joe"},{$inc: {age: 50}});
```

把 name 为 joe 的文档 age 增加 50。

10.10.4　$push

$push 用于把元素追加到数组字段中，如果字段不存在，会新增一个数组类型的字段进去。

```
db.user.update({"name":"joe"},{$push: {phone:
{"home":456789,"mobile":"13562352412"}}});
```

name 为 joe 的文档下的数组 phone 追加元素{"home":456789,"mobile":"13562352412"}。

10.10.5　$pushAll

$pushAll 作用与$push 类似，$push 追加的是单个元素，$pushAll 则是给数组追加多个元素，以数组的形式作为参数。

```
db.user.update({"name":"joe"},{$pushAll: {phone:
[{"home":456789,"mobile":"13562352412"},{"home":123456,"mobile":"13562352412"}
]}});
```

name 为 joe 的文档下的数组 phone 追加元素{"home":456789,"mobile":"13562352412"}和{"home":123456,"mobile":"13562352412"}。

10.10.6　$pull

$pull 删除数组中满足条件的元素，数组中如有多个元素满足，则都会被删除。

```
db.user.update({"name":"joe"},{$pull: {phone:{"home": 456789} }});
```

name 为 joe 的文档，phone 数组里元素的 home 值为 456789 的元素被删除。

10.10.7 $addToSet

$addToSet 增加一个值到数组内，类似于$push，但是只有当这个值不在数组内才增加，避免重复添加。

```
db.user.update({"name":"joe"},{$addToSet: {phone:
{"home":456789,"mobile":"13562352412"}}});
```

name 为 joe 的文档下的数组 phone 追加元素{"home":456789,"mobile":"13562352412"}，如果已经有该元素，则不添加。

10.10.8 $pop

$pop 删除数组的第一个或最后一个元素，根据参数来决定，1 表示删除最后一个元素，-1 表示删除第一个元素。

```
db.user.update({"name":"joe"},{$pop: {phone:1}});
```

name 为 joe 的文档删除 phone 数组中的最后一个元素。

10.10.9 $rename

$rename 修改字段名称：

```
db.user.update({"name":"joe"},{$rename: {"phone":"call"}});
```

name 为 joe 的文档把字段名 phone 修改为 call。

10.10.10 $bit

$bit 位操作，主要是将数值转换为二进制。进行对比得到结果值。

MongoDB 提供了三种位操作：and、or 和 xor。xor 是 MongoDB 2.6 版本新增加的支持。

and 运算通常用于二进制的取位操作，例如一个数 and 1 的结果就是取二进制的最末位。这可以用来判断一个整数的奇偶，二进制的最末位为 0 表示该数为偶数，最末位为 1 表示该数为奇数。

or 运算通常用于二进制特定位上的无条件赋值，例如一个数 or 1 的结果就是把二进制最末位强行变成 1。如果需要把二进制最末位变成 0，对这个数 or 1 之后再减一就可以了，其实际意义就是把这个数强行变成最接近的偶数。

异或（xor）的符号是^。按位异或运算，对等长二进制模式或二进制数的每一位执行逻辑按位异或操作，操作的结果是如果某位不同则该位为 1，否则该位为 0。

xor 运算的逆运算是它本身，也就是说两次异或同一个数最后结果不变，即（a xor b) xor b = a。xor 运算可以用于简单的加密，比如我想对我女朋友说 1314520，但怕别人知道，于是双方约定拿我的生日 19900503 作为密钥。1314520 xor 19900503 = 20686479，我就把 20686479 告诉女朋友。她对这个数字进行解密，再次计算 20686479 xor 19900503 的值，得到 1314520，于是她就明白了我要说的话。

因为位操作只能针对整型数值和 Long 型数值，所以我们这里先增加数据再进行运算。

```
db.bit.save({ _id: 1, expdata: NumberInt(13) });
db.bit.update({_id: 1},{ $bit: { "expdata": { and: NumberInt(5) } } });
db.bit.update({_id: 1},{ $bit: { "expdata": { or: NumberInt(5) } } });
db.bit.update({_id: 1},{ $bit: { "expdata": { xor: NumberInt(5) } } });
```

10.11 原生聚合运算

有数据的地方就离不开统计。尤其是需要对 Web 应用的数据进行统计分析时就需要对数据进行聚合。

聚合，指分散地聚集到一起，在网络用语中指对互联网各种信息的集合。MongoDB 的原生聚合运算有：count、distinct、group 以及 mapreduce。

10.11.1 数量查询 count

count 是最简单的聚合运算，返回文档的数量。

我们在之前的命令中已经学习过它的运用，count()中可以带查询条件文档。

```
db.user.count({age: {$gte: 18}})
```

查询年龄大于 18 的文档数量。

需要注意的是 count 的原理很简单，它只是去统计满足条件的文档的数量，没有充分考虑分片的情况。

所以在分片集群中 count 统计的数量是不准确的，会略大于真实数量值。

因为在迁移块时，同一个文档可能存在于不同的分片上，所以统计时会计算多了。

MongoDB 2.2 版本之后增加了新的聚合管道 aggregate，aggregate 考虑到了 shard 的环境，所以官方文档是推荐使用 aggregate 来进行 shard 环境下的 count。

aggregate 解决了分片集群计数的问题，我们在下一小节学习 aggregate。

更多关于聚合运算 count 的信息可以查看官网链接：

```
https://docs.mongodb.com/manual/reference/command/count/
```

10.11.2 不同值 distinct

distinct 用来找出指定字段的所有不同的值，使用时必须指定集合和字段。

我们有数据如下：

```
> db.user.find()
{ "_id" : ObjectId("58d9b6afa159504ca6c572e0"), "myName" : "joe", "age" : 28,
"company" : "google", "name" : "a2" }
```

```
{ "_id" : ObjectId("58d9ba2d6097167df6313438"), "myName" : "joe", "age" : 14 }
{ "_id" : ObjectId("58d9ba2d6097167df6313439"), "myName" : "ad", "age" : 14 }
{ "_id" : ObjectId("58d9ba2d6097167df631343a"), "myName" : "ad", "age" : 38 }
{ "_id" : ObjectId("58d9ba2d6097167df631343b"), "myName" : "ad", "age" : 24 }
{ "_id" : ObjectId("58d9ba2e6097167df631343c"), "myName" : "ab", "age" : 14 }
```

使用命令：

```
db.runCommand({"distinct":"user", "key":"age"})
```

找出 user 集合 age 字段的所有值。输出如下：

```
> db.runCommand({"distinct":"user", "key":"age"})
{ "values" : [ 28, 14, 38, 24 ], "ok" : 1 }
```

更多关于聚合运算 distinct 的信息可以查看官网链接：

```
https://docs.mongodb.com/manual/reference/command/distinct/
```

10.11.3 分组 group

group 根据我们设定的字段将文档分为不同的组，然后把每个组的文档中的数据进行聚合再返回一个最终的结果文档。

group 命令的语法：

```
db.collection.group({
    key:{field:1},
    initial:{count:0},
    cond:{},
    reduce: function ( curr, result ) { },
   keyf: function(doc){},
   finalize:function(result) {}
})
```

key 指定按什么字段进行分组。

keyf 有时候指定字段不能满足我们需要的分组情况，可以把文档中的字段经过重组后作为指定分组字段，这时候就需要使用 keyf，keyf 把函数的返回值作为指定字段使用。doc 参数表示当前的文档，我们可以对 doc 中的字段进行重组后作为分组的指定字段。

key 和 keyf 必须二选一。

initial，进行分组前变量初始化，该处声明的变量可以在以下回调函数中作为 result 的属性使用。

reduce，处理业务的方法，我们在这里进行数据统计。使用 key 做的分组会依次传入这个方法中，curr 参数代表当前分组中此刻遍历到的文档，result 参数就代表当前分组。

cond，可选配置，对哪些数据进行统计的查询条件。

finalize，可选配置，汇总分组结果的方法，result 参数也就是 reduce 的 result，都是代表当前分组，这个函数是在走完当前分组结束后回调。

除了分组的 key 字段外，结果文档中只包含 finalize 函数中操作过的属性字段。

做 group 之前首先要明确自己的思路，要进行什么样的统计操作。

我们有数据如下：

```
> db.user.find()
{ "_id" : ObjectId("58d9b6afa159504ca6c572e0"), "myName" : "joe", "age" : 28,
"company" : "google", "name" : "a2" }
{ "_id" : ObjectId("58d9ba2d6097167df6313438"), "myName" : "joe", "age" : 14 }
{ "_id" : ObjectId("58d9ba2d6097167df6313439"), "myName" : "ad", "age" : 14 }
{ "_id" : ObjectId("58d9ba2d6097167df631343a"), "myName" : "ad", "age" : 38 }
{ "_id" : ObjectId("58d9ba2d6097167df631343b"), "myName" : "ad", "age" : 24 }
{ "_id" : ObjectId("58d9ba2e6097167df631343c"), "myName" : "ab", "age" : 14 }
```

我们这里可以做一个统计：每个年龄有多少人，则使用 age 字段作为 key，在 reduce 时进行同组的数量计数，finalize 中组织结果文档格式。

使用代码如下：

```
db.user.group({
   key:{age:1},
   initial:{count:0},
   cond:{"age":{$gt:13}},
   reduce: function(curr,result) {
    result.count += 1;
   },
  finalize:function(result) {
    result.年龄=result.age;
    result.人数=result.count;
   }
});
```

代码表示使用 age 作为分组，初始化 count 变量值为 0，只选出 age 大于 13 的数据进行统计，每遍历一个当前分组中的文档，当前分组的数量加 1，最后增加结果字段年龄和人数。

输入结果为：

```
> db.user.group({
...     key:{age:1},
...     initial:{count:0},
...     cond:{"age":{$gt:13}},
...     reduce: function(curr,result) {
```

```
...      result.count += 1;
...    },
...    finalize:function(result) {
...      result.年龄=result.age;
...      result.人数=result.count;
...    }
... });
[
        {
                "age" : 28,
                "count" : 1,
                "年龄" : 28,
                "人数" : 1
        },
        {
                "age" : 14,
                "count" : 3,
                "年龄" : 14,
                "人数" : 3
        },
        {
                "age" : 38,
                "count" : 1,
                "年龄" : 38,
                "人数" : 1
        },
        {
                "age" : 24,
                "count" : 1,
                "年龄" : 24,
                "人数" : 1
        }
]
```

原生的聚合运算 group 在 MongoDB 3.4 版本中 group 命令已经不推荐使用，MongoDB 3.4 及以后的版本实现 group 请使用 aggregate 聚合管道中的 group 策略或者使用 mapreduce 来实现 group。

更多关于聚合运算 group 的信息可以查看官网链接：

```
https://docs.mongodb.com/manual/reference/command/group/
```

10.11.4　灵活统计 MapReduce

MongoDB 提供的原生聚合运算 count、distinct 和 group 并没能满足所有的统计需求，更多的高级聚合函数，比如 sum、average、max、min、variance（方差）和 standard deviation（标准差）等需要通过 MapReduce 来实现。

MapReduce 执行流程如图 10-4 所示。

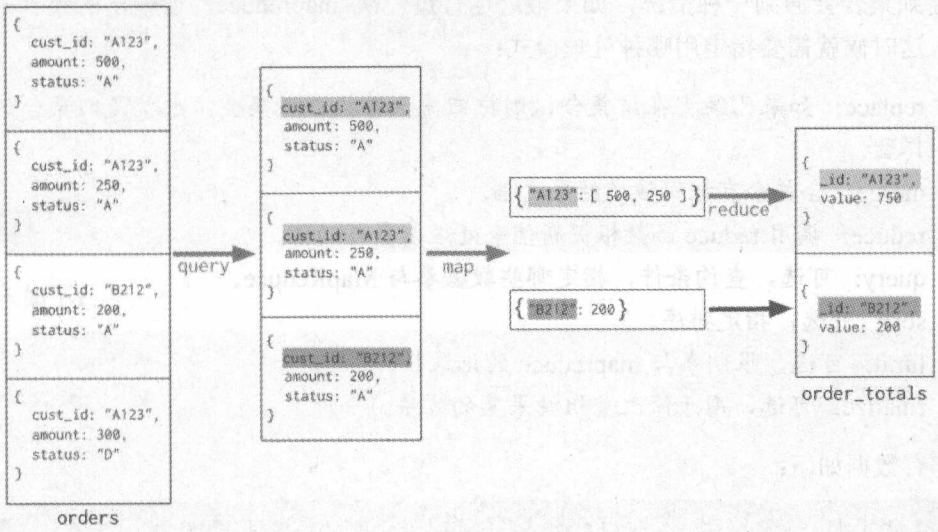

图 10-4　MapReduce 执行流程

MapReduce 命令的语法：

```
db.collection.mapReduce(
function() { emit(key,value);},
function(key,values){},
{out: collection 或者{},
query:{},
sort: {},
limit: number,
finalize: function (key, reduced) {}
})
```

第一个 function 是 map 函数，提交两个参数 key 和 value，数据会根据 key 的值进行分组，把同组的 value 的值放入 values 中。key 和 values 作为 reduce 函数的参数。

第二个 function 是 reduce 函数，key 参数是分组的字段，values 参数是同组的值。我们在 reduce 函数中做业务处理，比如说计数等。

out 指定结果集生成在什么地方，可以是一个集合名，也可以使用如下文档进行配置：

```
{ inline : 1}
{ replace : "collectionName" }
```

```
{ merge : "collectionName"}
{ reduce : "collectionName" }
```

{ inline : 1}将结果集放在内存中，返回内容给客户端，仅适用于结果集符合 16MB 返回限制的情况。

其他配置都是保存到集合中。

保存到集合会遇到一种情况，如果最近运行过一次 mapreduce，要输出的集合中已经有了数据，这时候就需要指定用哪种处理模式：

- replace: 如果已经存在该集合，则把旧集合替换成新集合。也就是旧集合的数据不保留。
- merge: 合并含有相同键的结果文档。
- reduce: 调用 reduce 函数根据新值来处理旧集合的值。
- query: 可选，查询条件，指定哪些数据参与 MapReduce。
- sort: 可选，指定排序。
- limit: 可选，限制参与 mapreduce 的记录数。
- finalize: 可选，用于修改重组结果集的方法。

我们有数据如下：

```
> db.user.find()
{ "_id" : ObjectId("58d9b6afa159504ca6c572e0"), "myName" : "joe", "age" : 28,
"company" : "google", "name" : "a2" }
{ "_id" : ObjectId("58d9ba2d6097167df6313438"), "myName" : "joe", "age" : 14 }
{ "_id" : ObjectId("58d9ba2d6097167df6313439"), "myName" : "ad", "age" : 14 }
{ "_id" : ObjectId("58d9ba2d6097167df631343a"), "myName" : "ad", "age" : 38 }
{ "_id" : ObjectId("58d9ba2d6097167df631343b"), "myName" : "ad", "age" : 24 }
{ "_id" : ObjectId("58d9ba2e6097167df631343c"), "myName" : "ab", "age" : 14 }
```

我们使用 MapReduce 做一个统计：每个年龄有多少人，this 表示当前文档，使用 this.age 的值作为 key。在 reduce 时进行同组的数量计数，finalize 中组织结果文档格式。

代码如下：

```
db.user.mapReduce(
    function () {
        emit(
            this.age,
            {age: this.age, count: 1}
        );
    },
    function (key, values) {
        var count = 0;
```

```
        values.forEach(function(val) {
            count += val.count;
        });
        return {age: key, count: count};
    },
    {
        out: { inline : 1 },
        finalize: function (key, reduced) {
            return {"年龄": reduced.age, "人数": reduced.count};
        }
    }
)
```

输入如下：

```
> db.user.mapReduce(
... function () {
... emit(
... this.age,
... {age: this.age, count: 1}
... );
... },
... function (key, values) {
... var count = 0;
... values.forEach(function(val) {
... count += val.count;
... });
... return {age: key, count: count};
... },
... {
... out: { inline : 1 },
... finalize: function (key, reduced) {
... return {"年龄": reduced.age, "人数": reduced.count};
... }
... }
... )
{
        "results" : [
                {
                        "_id" : 14,
```

```
                "value" : {
                        "年龄" : 14,
                        "人数" : 3
                }
        },
        {
                "_id" : 24,
                "value" : {
                        "年龄" : 24,
                        "人数" : 1
                }
        },
        {
                "_id" : 28,
                "value" : {
                        "年龄" : 28,
                        "人数" : 1
                }
        },
        {
                "_id" : 38,
                "value" : {
                        "年龄" : 38,
                        "人数" : 1
                }
        }
    ],
    "timeMillis" : 37,
    "counts" : {
            "input" : 6,
            "emit" : 6,
            "reduce" : 1,
            "output" : 4
    },
    "ok" : 1
}
```

我们这里实现计数 sum 的汇总，如果要实现其他 average、max、min、variance（方差）和 standard deviation（标准差）等更多的功能，只需要修改 reduce 方法体中的业务代码即可。