第 1 章 Linux 文件常用操作命令

```
**查看当前系统的时间
date

**查看有谁在线（哪些人登陆到了服务器）
who   查看当前在线
last  查看最近的登陆历史记录


2、文件系统操作
**
ls /     查看根目录下的子节点（文件夹和文件）信息
ls -al   -a是显示隐藏文件    -l是以更详细的列表形式

**切换目录
cd  /home

**创建文件夹
mkdir aaa       这是相对路径的写法
mkdir -p aaa/bbb/ccc
mkdir  /data     这是绝对路径的写法
```

```
[root@118 wwy]# mkdir aaa
[root@118 wwy]# mkdir aaa/bbb/ccc
mkdir: 无法创建目录"aaa/bbb/ccc": 没有那个文件或目录
[root@118 wwy]# mkdir -p aaa/bbb/ccc
[root@118 wwy]# cd aaa/bbb/ccc
[root@118 ccc]# pwd
/home/wwy/aaa/bbb/ccc
[root@118 ccc]#
```

```
**创建文件夹
mkdir aaa       这是相对路径的写法
mkdir -p aaa/bbb/ccc
mkdir  /data     这是绝对路径的写法

**删除文件夹
rmdir    可以删除空目录
rm -r aaa    可以把aaa整个文件夹及其中的所有子节点全部删除
rm -rf aaa    强制删除aaa

**修改文件夹名称
mv aaa angelababy

**创建文件
touch  somefile.1    创建一个空文件
echo "i miss you,my baby" > somefile.2  利用重定向">"的功能，将一条指令的输出结果写入到一个文件中，会覆盖原文件内容
echo "huangxiaoming ,gun dan" >> somefile.2      将一条指令的输出结果追加到一个文件中，不会覆盖原文件内容
用vi文本编辑器来编辑生成文件
vi  somefile.4
1、首先会进入"一般模式"，此模式只接受各种快捷键，不能编辑文件内容
2、按i键，就会从一般模式进入编辑模式，此模式下，敲入的都是文件内容
3、编辑完成之后，按Esc键退出编辑模式，回到一般模式；
4、再按：，进入"底行命令模式"，输入wq命令，回车即可
```

利用重定向”>”的功能，将一条指令输出的结果写入到一个文件中。

```
[root@l18 wwy]# echo "i miss you,my baby"
i miss you,my baby
[root@l18 wwy]# echo "i miss you,my baby" > somefile.1
[root@l18 wwy]# cat somefile.1
i miss you,my baby
[root@l18 wwy]# ls
2018-07-21-02-30-14.058-VirtualBox-6859.log   somefile.1         VirtualBox VMs  公共  视频  文档  音乐
2018-08-31-08-54-46.072-VirtualBox-18478.log  thinclient_drives  wwy             模板  图片  下载  桌面
[root@l18 wwy]# touch somefile.2
[root@l18 wwy]# ls > somefile.2
[root@l18 wwy]# cat somefile.2
2018-07-21-02-30-14.058-VirtualBox-6859.log
2018-08-31-08-54-46.072-VirtualBox-18478.log
somefile.1
somefile.2
thinclient_drives
VirtualBox VMs
wwy
公共
模板
视频
图片
文档
下载
音乐
桌面
[root@l18 wwy]#
```

利用”>>”进行追加

```
[root@l18 wwy]# cat somefile.1
i miss you,my baby
[root@l18 wwy]# echo "i love you, my baby" >> somefile.1
[root@l18 wwy]# cat somefile.1
i miss you,my baby
i love you, my baby
[root@l18 wwy]#
```

vi 修改文件的常用快捷键

```
一些有用的快捷键（在一般模式下使用）：
a  在光标后一位开始插入
A   在该行的最后插入
I   在该行的最前面插入
gg   直接跳到文件的首行
G    直接跳到文件的末行
dd   删除行，如果  5dd   ，则一次性删除光标后的5行
yy  复制当前行，  复制多行，则  3yy，则复制当前行附近的3行
p   粘贴
v  进入字符选择模式，选择完成后，按y复制，按p粘贴
ctrl+v  进入块选择模式，选择完成后，按y复制，按p粘贴
shift+v  进入行选择模式，选择完成后，按y复制，按p粘贴

查找并替换（在底行命令模式中输入）
%s/sad/88888888888888     效果：查找文件中所有sad，替换为88888888888888
/you        效果：查找文件中出现的you，并定位到第一个找到的地方，按n可以定位到下一个匹配位置（按N定位到上一个）
```

文件权限的操作

```
****linux文件权限的描述格式解读
drwxr-xr-x       （也可以用二进制表示  111 101 101  -->  755）

d：标识节点类型（d：文件夹   -：文件   l:链接）
r：可读   w：可写    x：可执行
第一组rwx：   表示这个文件的拥有者对它的权限：可读可写可执行
第二组r-x：   表示这个文件的所属组对它的权限：可读，不可写，可执行
第三组r-x：   表示这个文件的其他用户（相对于上面两类用户）对它的权限：可读，不可写，可执行


****修改文件权限
chmod g-rw haha.dat    表示将haha.dat对所属组的rw权限取消
chmod o-rw haha.dat     表示将haha.dat对其他人的rw权限取消
chmod u+x haha.dat      表示将haha.dat对所属用户的权限增加x

也可以用数字的方式来修改权限
chmod 664 haha.dat
就会修改成   rw-rw-r--

如果要将一个文件夹的所有内容权限统一修改，则可以-R参数
chmod -R 770 aaa/
```

```
[hadoop@shizhan aaa]$ ll
总用量 8
drwxrwxr-x. 3 hadoop hadoop 4096 4月   2 23:22 bbb
-rw-rw-r--. 1 hadoop hadoop    5 4月   2 23:22 haha.dat
[hadoop@shizhan aaa]$ chmod g-rw haha.dat
[hadoop@shizhan aaa]$ ll
总用量 8
drwxrwxr-x. 3 hadoop hadoop 4096 4月   2 23:22 bbb
-rw----r--. 1 hadoop hadoop    5 4月   2 23:22 haha.dat
[hadoop@shizhan aaa]$ chmod o-rw haha.dat
[hadoop@shizhan aaa]$ ll
总用量 8
drwxrwxr-x. 3 hadoop hadoop 4096 4月   2 23:22 bbb
-rw-------. 1 hadoop hadoop    5 4月   2 23:22 haha.dat
[hadoop@shizhan aaa]$ chmod u+x haha.dat
[hadoop@shizhan aaa]$ ll
总用量 8
drwxrwxr-x. 3 hadoop hadoop 4096 4月   2 23:22 bbb
-rwx------. 1 hadoop hadoop    5 4月   2 23:22 haha.dat
[hadoop@shizhan aaa]$
```

基本的用户管理权限

```
4、基本的用户管理

*****添加用户
useradd  angela
要修改密码才能登陆
passwd angela  按提示输入密码即可


**为用户配置sudo权限
用root编辑 vi /etc/sudoers
在文件的如下位置，为hadoop添加一行即可
root    ALL=(ALL)       ALL
hadoop  ALL=(ALL)       ALL

然后，hadoop用户就可以用sudo来执行系统级别的指令
[hadoop@shizhan ~]$ sudo useradd huangxiaoming


5、系统管理操作
*****查看主机名
hostname
```

```
1.查看主机名
hostname

2.修改主机名(重启后无效)
hostname hadoop

3.修改主机名(重启后永久生效)
vi /ect/sysconfig/network

4.修改IP(重启后无效)
ifconfig eth0 192.168.12.22

5.修改IP(重启后永久生效)
vi /etc/sysconfig/network-scripts/ifcfg-eth0

6.查看系统信息
uname -a
uname -r

7.查看ID命令
id -u
id -g

8.日期
date
date +%Y-%m-%d
date +%T

date +%Y-%m-%d" "%T

9.日历
cal 2012

10.查看文件信息
file filename

11.挂载硬盘
mount
umount
加载windows共享
mount -t cifs //192.168.1.100/tools /mnt

12.查看文件大小
du -h
du -ah

13.查看分区
df -h

14.ssh

ssh hadoop@192.168.1.1

15.关机
shutdown -h now /init 0
shutdown -r now /reboot
```

查看文件大小 du -sh

```
[root@l18 wwy]# du -sh somefile.1
4.0K    somefile.1
[root@l18 wwy]#
```

查看磁盘空间

df -h

挂载命令 mount

```
mount ****  挂载外部存储设备到文件系统中
mkdir   /mnt/cdrom      创建一个目录，用来挂载
mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom/     将设备/dev/cdrom挂载到 挂载点 ：   /mnt/cdrom中

umount
umount /mnt/cdrom
```

```
[root@shizhan01 ~]# unmount /mnt/cdrom
-bash: unmount: command not found
[root@shizhan01 ~]# umount /mnt/cdrom
[root@shizhan01 ~]# ll /mnt/cdrom
总用量 0
[root@shizhan01 ~]# mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom/
[root@shizhan01 ~]# ll /mnt/cdrom
总用量 558
-r--r--r--. 2 root root     14 8月    5 2015 CentOS_BuildTag
dr-xr-xr-x. 3 root root   2048 8月    5 2015 EFI
-r--r--r--. 2 root root    212 11月 27 2013 EULA
-r--r--r--. 2 root root  18009 11月 27 2013 GPL
dr-xr-xr-x. 3 root root   2048 8月    5 2015 images
dr-xr-xr-x. 2 root root   2048 8月    5 2015 isolinux
dr-xr-xr-x. 2 root root 528384 8月    5 2015 Packages
-r--r--r--. 2 root root   1354 7月   25 2015 RELEASE-NOTES-en-US.html
dr-xr-xr-x. 2 root root   4096 8月    5 2015 repodata
-r--r--r--. 2 root root   1706 11月 27 2013 RPM-GPG-KEY-CentOS-6
-r--r--r--. 2 root root   1730 11月 27 2013 RPM-GPG-KEY-CentOS-Debug-6
-r--r--r--. 2 root root   1730 11月 27 2013 RPM-GPG-KEY-CentOS-Security-6
-r--r--r--. 2 root root   1734 11月 27 2013 RPM-GPG-KEY-CentOS-Testing-6
-r--r--r--. 1 root root   3380 8月    5 2015 TRANS.TBL
[root@shizhan01 ~]#
```

ssh 免密码登陆

```
******配置主机之间的免密ssh登陆
假如 A   要登陆   B
在A上操作：
%%首先生成密钥对
ssh-keygen    （提示时，直接回车即可）
%%再将A自己的公钥拷贝并追加到B的授权列表文件authorized_keys中
ssh-copy-id    B
```

查看文件

```
*****查看文件内容
cat    somefile     一次性将文件内容全部输出（控制台）
more   somefile     可以翻页查看，下翻一页(空格)    上翻一页（b）    退出（q）
less   somefile     可以翻页查看,下翻一页(空格)    上翻一页（b），上翻一行(↑)  下翻一行（↓）   可以搜索关键字（/keyword）

tail -10  install.log   查看文件尾部的10行
tail -f install.log    小f跟踪文件的唯一inode号，就算文件改名后，还是跟踪原来这个inode表示的文件
tail -F install.log    大F按照文件名来跟踪

head -10  install.log   查看文件头部的10行
```

后台服务管理

```
******后台服务管理
service network status    查看指定服务的状态
service network stop      停止指定服务
service network start     启动指定服务
service network restart   重启指定服务
service --status-all   查看系统中所有的后台服务


chkconfg




*****系统启动级别管理
vi  /etc/inittab

# Default runlevel. The runlevels used are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:
```

```
设置后台服务的自启配置
chkconfig    查看所有服务器自启配置
chkconfig iptables off    关掉指定服务的自动启动
chkconfig iptables on   开启指定服务的自动启动
```

```
软件安装


1、如何上传安装包到服务器
**可以使用图形化工具，如：  filezilla
**可以使用sftp工具：   alt+p 调出后，用put命令上传
上传（如果不cd指定目录，则上传到当前用户的主目录）：
sftp> cd /home/
sftp> put C:\Users\Administrator\Desktop\day02\soft\jdk-7u45-linux-x64.tar.gz
下载（lcd指定下载到本地的目标路径）
sftp> lcd d:/
sftp> get /home/jdk-7u45-linux-x64.tar.gz

**lrzsz


2、安装jdk
**压缩解压缩的相关命令****
%%压缩解压缩%%
root@mini1 ~]# gzip access.log
[root@mini1 ~]# ll
总用量 134892
-rw-r--r--. 1 root root        68 4月    3 17:37 access.log.gz

解压gz文件：   gzip -d access.log.gz
```

```
%%一次性完成打包&&压缩的操作%%
产生压缩包：
[root@mini1 ~]# tar -zcvf my.tar.gz aaa/
aaa/
aaa/2.txt
aaa/3.txt
aaa/1.txt

解压缩包：
[root@mini1 ~]# tar -zxvf my.tar.gz
aaa/
aaa/2.txt
aaa/3.txt
aaa/1.txt
```

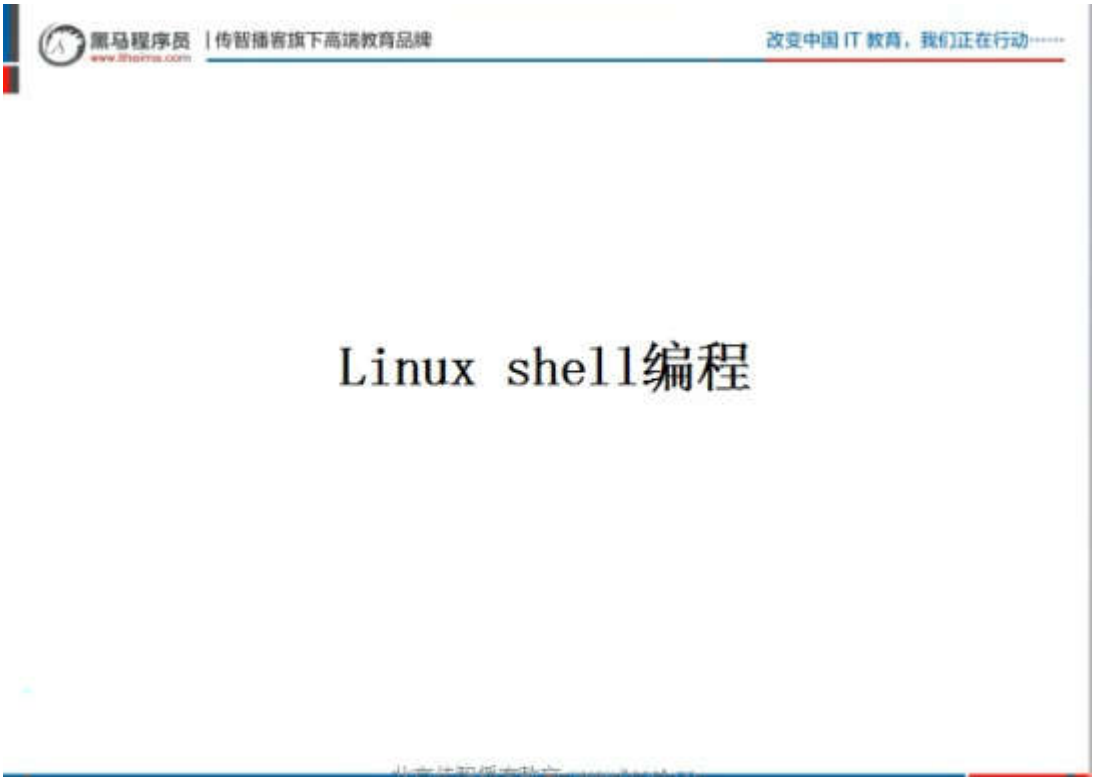
iptables

3. iptables 原理简介

3.1. iptables 的结构

在 iptables 中有四张表，分别是 filter、nat、mangle 和 raw 每一个表中都包含了各自不同的链，最常用的是 filter 表。

第 2 章 Shell 编程



什么是Shell

- Shell是用户与内核进行交互操作的一种接口，目前最流行的Shell称为bash Shell
- Shell也是一门编程语言<解释型的编程语言>，即shell脚本
- 一个系统可以存在多个shell，可以通过cat /etc/shells命令查看系统中安装的shell，不同的shell可能支持的命令语法是不相同的

Shell脚本的执行方式

- 第一种：输入脚本的绝对路径或相对路径

首先要赋予+x权限

/root/helloWorld.sh

./helloWorld.sh

或者，不用赋予+x权限，而用解释器解释执行

sh helloworld.sh

- 第二种：bash或sh +脚本

sh /root/helloWorld.sh

sh helloWorld.sh

- 第三种：在脚本的路径前再加"."

  ./root/helloWorld.sh

  ../helloWorld.sh

- 区别：第一种和第二种会新开一个bash，不同bash中的变量无法共享

Shell中的变量

- Linux Shell中的变量分为“系统变量”和“用户自定义变量”，可以通过set命令查看那系统变量
- 系统变量：$HOME、$PWD、$SHELL、$USER等等

显示当前shell中所有变量： set

定义变量

- 变量=值 （例如STR=abc）
- 等号两侧不能有空格
- 变量名称一般习惯为大写
- 双引号和单引号有区别，双引号仅将空格脱意，单引号会将所有特殊字符脱意

脱意是指将这些特殊的符号也作为普通字符处理。

```
[root@118 wwy]# x=33
[root@118 wwy]# echo $x
33
[root@118 wwy]# x=hello world
bash: world: 未找到命令...
[root@118 wwy]# x="hello world"
[root@118 wwy]# echo $x
hello world
[root@118 wwy]# y=33
[root@118 wwy]# echo $y
33
[root@118 wwy]# x="hello world $y"
[root@118 wwy]# echo $x
hello world 33
[root@118 wwy]# x='hello world $y'
[root@118 wwy]# echo $x
hello world $y
[root@118 wwy]#
```

STR="hello world"

A=9

unset A 撤销变量 A

readonly B=2 声明静态的变量 B=2，不能 unset

export 变量名 可把变量提升为全局环境变量，可供其他shell程序使用

source /etc/profile source 命令即是执行/etc/profile 文件中的 export 命令

将命令的返回值赋给变量

- A=`ls -la` 反引号，运行里面的命令，并把结果返回给变量A
- A=$(ls -la) 等价于反引号

反引号为键盘上数字 1 左边

Shell中的特殊变量

- $? 表示上一个命令退出的状态
- $$ 表示当前进程编号
- $0 表示当前脚本名称
- $n 表示n位置的输入参数（n代表数字，n>=1）
- $# 表示参数的个数，常用于循环
- $*和$@ 都表示参数列表

```
[root@118 wwy]# lls
bash: lls: 未找到命令...
相似命令是：  'ls'
[root@118 wwy]# echo $?
127
[root@118 wwy]# ls
2018-07-21-02-30-14.058-VirtualBox-6859.log    thinclient
2018-08-31-08-54-46.072-VirtualBox-18478.log   VirtualBox
[root@118 wwy]# echo $?
0
[root@118 wwy]#
```

$*与$@区别

- $* 和 $@ 都表示传递给函数或脚本的所有参数，不被双引号" "包含时，都以$1 $2 … $n 的形式输出所有参数
- 当它们被双引号" "包含时，"$*" 会将所有的参数作为一个整体，以"$1 $2 … $n"的形式输出所有参数；"$@" 会将各个参数分开，以"$1" "$2" … "$n" 的形式输出所有参数

运算符

- 格式 :expr m + n 或$((m+n)) 注意expr运算符间要有空格
- 例如计算（2 +3 ）×4 的值

1 .分步计算

```
S=`expr 2 + 3`
expr $S \* 4
```

2.一步完成计算

```
expr `expr 2 + 3 ` \* 4
echo `expr \`expr 2 + 3\` \* 4`
```

或

```
$(((2+3)*4))
```

for 循环

for循环

- 第一种：

```
for N in 1 2 3
do
    echo $N
done
```

或

```
for N in 1 2 3; do echo $N; done
```

或

```
for N in {1..3}; do echo $N; done
```

- 第二种：

```
for ((i = 0; i <= 5; i++))
do
    echo "welcome $i times"
done
或
for ((i = 0; i <= 5; i++)); do echo "welcome $i times";
done
```

while 循环

while循环

- 第一种

```
while expression
do
command
...
done
```

- 第二种

```
int=1
while ((int<=3))
do
  echo $int
  let int++
done
```

- 格式

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

Read 命令

read命令

- read -p(提示语句)-n(字符个数) -t(等待时间)

```
read -p "please input your name: " NAME
```

使用示例：

```
[root@localhost ~]# read -p "please input your name: " c
please input your name: name
[root@localhost ~]# echo $c
name
[root@localhost ~]#
```

if判断

- 语法

```
if condition
then
   statements
[elif condition
   then statements. ..]
[else
   statements ]
fi
```

判断语句

- [condition] (注意condition前后要有空格)

  #非空返回true，可使用$?验证（0为true，>1为false）

  [itcast]

  #空返回false

  [ ]

- [condition] && echo OK || echo notok

  条件满足，执行后面的语句

常用判断条件

= 字符串比较
-lt 小于
-le 小于等于
-eq 等于
-gt 大于
-ge 大于等于
-ne 不等于

-r 有读的权限
-w 有写的权限
-x 有执行的权限
-f 文件存在并且是一个常规的文件
-s 文件存在且不为空
-d 文件存在并是一个目录
-b文件存在并且是一个块设备
-L 文件存在并且是一个链接

Shell自定义函数

- 语法

```
[ function ] funname [()]
{
    action;
    [return int;]
}
```

function start() / function start / start()

Shell自定义函数

- 注意

1.必须在调用函数地方之前，先声明函数，shell脚本是逐行运行。不会像其它语言一样先预编译

2.函数返回值，只能通过$? 系统变量获得，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值。 return后跟数值n(0-255)

脚本调试

- sh -vx helloWorld.sh
- 或者在脚本中增加set -x

第 3 章高级文本处理命令

sed命令

sed全称是：Stream EDitor即流编辑器，是一个很好的文本处理工具，本身是一个管道命令，处理时，把当前处理的行存储在临时缓冲区中，接着用sed命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。接着处理下一行。它是以行为单位进行处理，可以将数据行进行替换、删除、新增、选取等特定工作。

sed function

- a：新增， a 的后面可以接字串，而这些字串会在新的一行出现(目前的下一行)
- d：删除，因为是删除啊，所以 d 后面通常不接任何内容
- i：插入， i 的后面可以接字串，而这些字串会在新的一行出现(目前的上一行)
- p：列印，亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运行
- s：取代，可以直接进行取代的工作！通常这个 s 的动作可以搭配正规表示法！例如 1,20s/old/new/g

awk命令

AWK是一种优良的文本处理工具。其名称得自于它的创始人 Alfred Aho 、Peter Weinberger 和 Brian Kernighan 姓氏的首个字母，AWK 提供了极其强大的功能：可以进行样式装入、流控制、数学运算符、进程控制语句甚至于内置的变量和函数。它具备了一个完整的语言所应具有的几乎所有精美特性。实际上 AWK 的确拥有自己的语言：AWK 程序设计语言， 三位创建者已将它正式定义为“样式扫描和处理语言”。它允许您创建简短的程序，这些程序读取输入文件、为数据排序、处理数据、对输入执行计算以及生成报表，还有无数其他的功能。

wc –c 程序名称 //统计程序内代码的行数

cut 语法

```
cut语法

[root@www ~]# cut -d'分隔字符' -f fields <==用于有特定分隔字符
[root@www ~]# cut -c 字符区间             <==用于排列整齐的信息
选项与参数:
-d  : 后面接分隔字符。与 -f 一起使用;
-f  : 依据 -d 的分隔字符将一段信息分割成为数段，用 -f 取出第几段的意思;
-c  : 以字符 (characters) 的单位取出固定字符区间;


PATH 变量如下

[root@www ~]# echo $PATH
/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin:/usr/X11R6/bin:/usr/games
# 1 | 2       | 3   | 4       | 5             | 6            | 7


将 PATH 变量取出，我要找出第五个路径。

#echo $PATH | cut -d ':' -f 5
/usr/local/bin


将 PATH 变量取出，我要找出第三和第五个路径。

#echo $PATH | cut -d ':' -f 3,5
/sbin:/usr/local/bin
```

Sort 语法

```
sort
sort 命令对 File 参数指定的文件中的行排序，并将结果写到标准输出。如果 File 参数指定多个文件，那么 sort
命令将这些文件连接起来，并当作一个文件进行排序。

sort语法


[root@www ~]# sort [-fbMnrtuk] [file or stdin]
选项与参数:
-f  : 忽略大小写的差异，例如 A 与 a 视为编码相同;
-b  : 忽略最前面的空格符部分;
-M  : 以月份的名字来排序，例如 JAN，DEC 等等的排序方法;
-n  : 使用『纯数字』进行排序(默认是以文字型态来排序的);
-r  : 反向排序;
-u  : 就是 uniq ，相同的数据中，仅出现一行代表;
-t  : 分隔符，默认是用 [tab] 键来分隔;
-k  : 以那个区间 (field) 来进行排序的意思


对/etc/passwd 的账号进行排序
[root@www ~]# cat /etc/passwd | sort
adm:x:3:4:adm:/var/adm:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
sort 是默认以第一个数据来排序，而且默认是以字符串形式来排序,所以由字母 a 开始升序排序。
```

/etc/passwd 内容是以 ： 来分隔的，我想以第三栏来排序，该如何

```
[root@www ~]# cat /etc/passwd | sort -t ':' -k 3
root:x:0:0:root:/root:/bin/bash
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
```

默认是以字符串来排序的，如果想要使用数字排序：

```
cat /etc/passwd | sort -t ':' -k 3n
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
```

默认是升序排序，如果要倒序排序，如下

```
cat /etc/passwd | sort -t ':' -k 3nr
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
ntp:x:106:113::/home/ntp:/bin/false
messagebus:x:105:109::/var/run/dbus:/bin/false
sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
```

如果要对/etc/passwd,先以第六个域的第2个字符到第4个字符进行正向排序，再基于第一个域进行反向排序。

```
cat /etc/passwd |  sort -t':' -k 6.2,6.4 -k 1r
sync:x:4:65534:sync:/bin:/bin/sync
proxy:x:13:13:proxy:/bin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
```

查看/etc/passwd有多少个shell:对/etc/passwd的第七个域进行排序，然后去重：

```
cat /etc/passwd |  sort -t':' -k 7 -u
root:x:0:0:root:/root:/bin/bash
syslog:x:101:102::/home/syslog:/bin/false
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
```

Uniq 语法

```
uniq
 uniq命令可以去除排序过的文件中的重复行，因此uniq经常和sort合用。也就是说，为了使uniq起作用，所有的重复行必须是相邻的。

uniq语法

[root@www ~]# uniq [-icu]
选项与参数:
-i  ： 忽略大小写字符的不同;
-c  ： 进行计数
-u  ： 只显示唯一的行


testfile的内容如下


cat testfile
hello
world
friend
hello
world
hello
```

直接删除未经排序的文件，将会发现没有任何行被删除

```
#uniq testfile
hello
world
friend
hello
world
hello
```

排序文件，默认是去重

```
#cat testfile | sort |uniq
friend
hello
world
```

排序之后删除了重复行，同时在行首位置输出该行重复的次数

```
#sort testfile | uniq -c
1 friend
3 hello
2 world
```

wc 语法

```
[root@www ~]# wc [-lwm]
选项与参数:
-l  : 仅列出行;
-w  : 仅列出多少字(英文单字);
-m  : 多少字符;
```

默认使用wc统计/etc/passwd

```
#wc /etc/passwd
40   45 1719 /etc/passwd
```

40是行数，45是单词数，1719是字节数

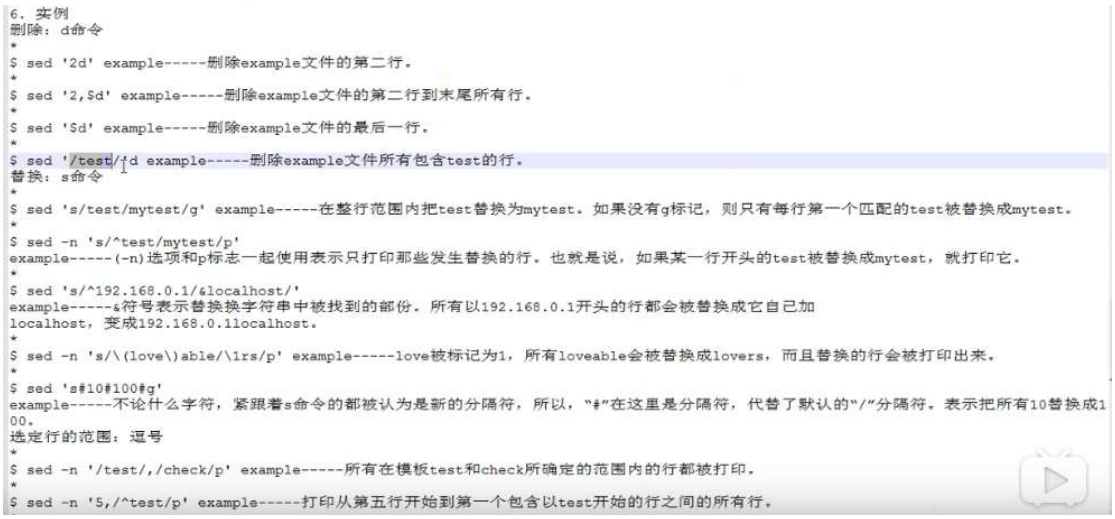
wc的命令比较简单使用，每个参数使用如下：

```
#wc -l /etc/passwd    #统计行数，在对记录数时，很常用
40 /etc/passwd        #表示系统有40个账户

#wc -w /etc/passwd   #统计单词出现次数
45 /etc/passwd

#wc -m /etc/passwd   #统计文件的字符数
1719
```

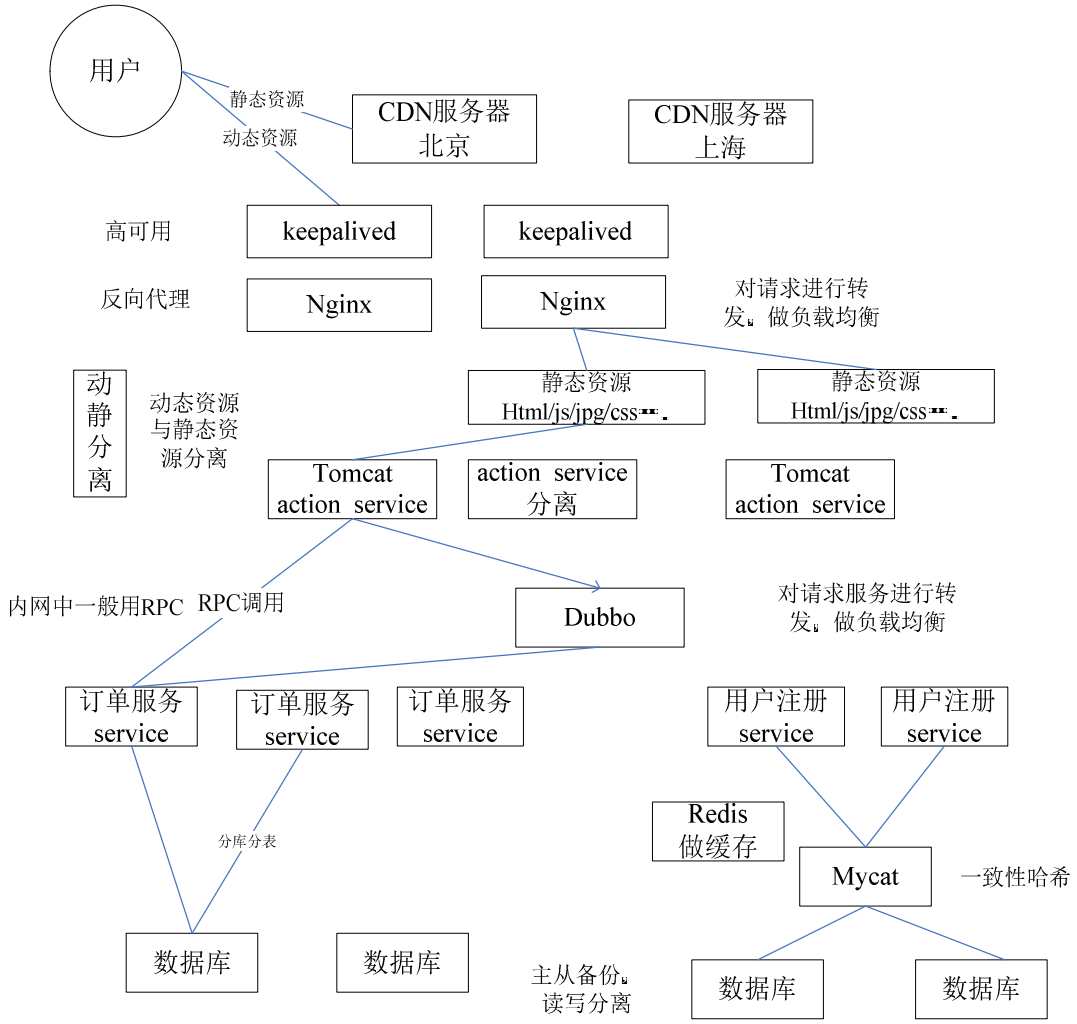
sed ‘2d’ testfile　//删除 testfile 的第二行，并非真的删除，而是显示的时候没有显示

```
[root@118 wwy]# cat testfile
hello
world
friend
hello
world
hello

[root@118 wwy]# sed '2d' testfile
hello
friend
hello
world
hello

[root@118 wwy]# cat testfile
hello
world
friend
hello
world
hello

[root@118 wwy]#
```

sed –i ‘2d’ testfile //删除 testfile 第二行，真的删除



第 4 章高并发网站架构



安装 Nginx

3.1. 下载 nginx

官网：http://nginx.org/

3.2. 上传并解压 nginx

tar -zxvf nginx-1.8.1.tar.gz -C /usr/local/src

3.3. 编译 nginx

#进入到 nginx 源码目录
cd /usr/local/src nginx-1.8.1

#检查安装环境
./configure --prefix=/usr/local/nginx

#缺包报错 ./configure: error: C compiler cc is not found

#使用 YUM 安装缺少的包
yum -y install gcc pcre-devel openssl openssl-devel

#编译安装
make && make install

验证安装是否成功

4.1. 配置反向代理

1.修改 nginx 配置文件

```
server {
    listen          80;
    server_name     nginx-01.itcast.cn;    #nginx所在服务器的主机名
    #172.16.203.101/hello.html
    location / {
        root html;
        proxy_pass http://192.168.0.21:8080;
    }                                      #代理走向的目标服务器：tomcat
}
```

2.启动 tomcat-01 上的 tomcat

3.启动 nginx-01 上的 nginx

./nginx

4.2. 动静分离

#动态资源 index.jsp

```
location ~ .*\.(jsp|do|action)$ {
    proxy_pass http://tomcat-01.itcast.cn:8080;
}
```

#静态资源

```
location ~ .*\.(html|js|css|gif|jpg|jpeg|png)$ {
    expires 3d;
```

4.3. 负载均衡

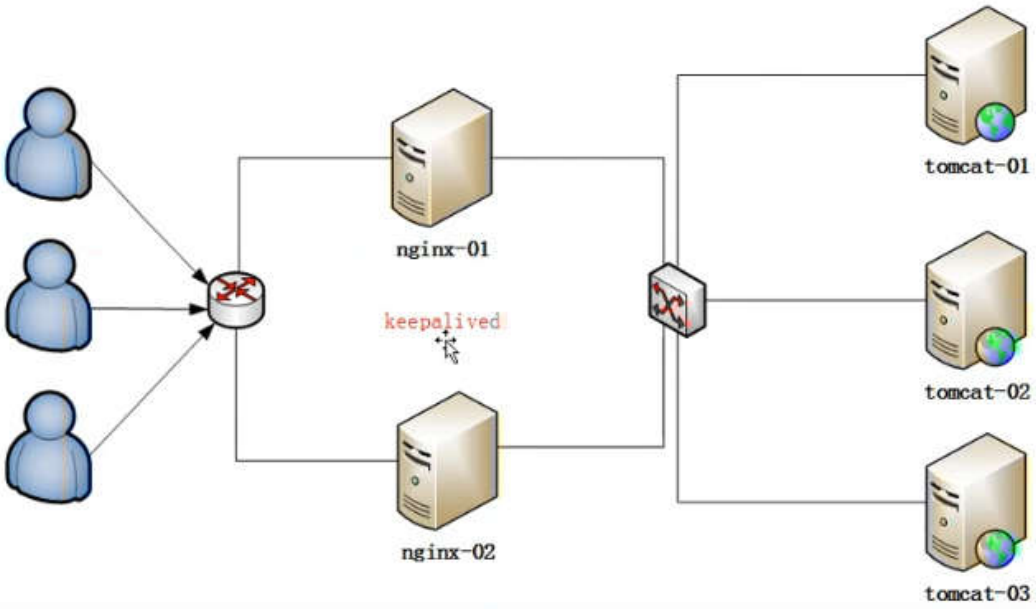
在 http 这个节下面配置一个叫 upstream 的，后面的名字可以随意取，但是要和 location 下的 proxy_pass http://后的保持一致。

```
http {
    upstream tomcats {
        server 172.16.203.20:8080 weight=1;
        server tomcat-02.itcast.cn:8080 weight=1;
        server tomcat-02.itcast.cn:8080 weight=1;
    }
    location ~ .*\.(jsp|do|action) {
        proxy_pass http://tomcats;
    }
}
```

5. 利用 keepalived 实现高可靠（HA）

5.1. 高可靠概念

HA(High Available)，高可用性集群，是保证业务连续性的有效解决方案，一般有两个或两个以上的节点，且分为活动节点及备用节点。



5.2. 高可靠软件 keepalived

keepalive 是一款可以实现高可靠的软件，通常部署在 2 台服务器上，分为一主一备。Keepalived 可以对本机上的进程进行检测，一旦 Master 检测出某个进程出现问题，将自己切换成 Backup 状态，然后通知另外一个节点切换成 Master 状态。

5.3. keepalived 安装

下载 keepalived 官网:http://keepalived.org

将 keepalived 解压到/usr/local/src 目录下
tar -zxvf keepalived-1.2.19.tar.gz -C /usr/local/src

进入到/usr/local/src/keepalived-1.2.19 目录
cd /usr/local/src/keepalived-1.2.19

开始 configure
./configure --prefix=/usr/local/keepalived

#编译并安装
make && make install

5.4. 将 keepalived 添加到系统服务中

拷贝执行文件
cp /usr/local/keepalived/sbin/keepalived /usr/sbin/
将 init.d 文件拷贝到 etc 下,加入开机启动项
cp /usr/local/keepalived/etc/rc.d/init.d/keepalived /etc/init.d/keepalived
将 keepalived 文件拷贝到 etc 下
cp /usr/local/keepalived/etc/sysconfig/keepalived /etc/sysconfig/
创建 keepalived 文件夹
mkdir -p /etc/keepalived
将 keepalived 配置文件拷贝到 etc 下
cp /usr/local/keepalived/etc/keepalived/keepalived.conf /etc/keepalived/keepalived.conf
添加可执行权限
chmod +x /etc/init.d/keepalived

##以上所有命令一次性执行：

```
cp /usr/local/keepalived/sbin/keepalived /usr/sbin/
cp /usr/local/keepalived/etc/rc.d/init.d/keepalived /etc/init.d/keepalived
cp /usr/local/keepalived/etc/sysconfig/keepalived /etc/sysconfig/
```

```
mkdir -p /etc/keepalived
cp /usr/local/keepalived/etc/keepalived/keepalived.conf /etc/keepalived/keepalived.conf
chmod +x /etc/init.d/keepalived
chkconfig --add keepalived
chkconfig keepalived on
```

添加 keepalived 到开机启动

chkconfig --add keepalived

chkconfig keepalived on

5.5. 配置 keepalived 虚拟 IP

修改配置文件： /etc/keepalived/keepalived.conf

#MASTER 节点

```
global_defs {
}
vrrp_instance VI_1 {
    state MASTER    #指定 A 节点为主节点 备用节点上设置为 BACKUP 即可
    interface eth0    #绑定虚拟 IP 的网络接口
    virtual_router_id 51    #VRRP 组名，两个节点的设置必须一样，以指明各个节点属于同一 VRRP 组
    priority 100    #主节点的优先级（1-254 之间），备用节点必须比主节点优先级低
    advert_int 1    #组播信息发送间隔，两个节点设置必须一样
    authentication {    #设置验证信息，两个节点必须一致
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {    #指定虚拟 IP, 两个节点设置必须一样
        192.168.33.60/24    #如果两个 nginx 的 ip 分别是 192.168.33.61,..62，则此处的虚拟 ip 跟它俩同一个网段即可
    }
}
```

#BACKUP 节点

```
global_defs {
}
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 51
    priority 99
```

```
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.33.60/24
    }
}
```

#分别启动两台机器上的 keepalived
service keepalived start

测试：
杀掉 master 上的 keepalived 进程，你会发现，在 slave 机器上的 eth0 网卡多了一个 ip 地址
查看 ip 地址的命令： ip addr

5.6. 配置 keepalived 心跳检查

原理：
Keepalived 并不跟 nginx 耦合，它俩完全不是一家人
但是 keepalived 提供一个机制：让用户自定义一个 shell 脚本去检测用户自己的程序，返回状态给 keepalived 就可以了

```
#MASTER 节点
global_defs {
}

vrrp_script chk_health {
    script "[[ `ps -ef | grep nginx | grep -v grep | wc -l` -ge 2 ]] && exit 0 || exit 1"
    interval 1      #每隔 1 秒执行上述的脚本，去检查用户的程序 ngnix
    weight -2
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 1
    priority 100
    advert_int 2
    authentication {
        auth_type PASS
        auth_pass 1111
    }

    track_script {
        chk_health
    }

    virtual_ipaddress {
        10.0.0.10/24
    }

    notify_master "/usr/local/keepalived/sbin/notify.sh master"
    notify_backup "/usr/local/keepalived/sbin/notify.sh backup"
    notify_fault "/usr/local/keepalived/sbin/notify.sh fault"
}
```

#添加切换通知脚本

vi /usr/local/keepalived/sbin/notify.sh

```
#!/bin/bash

case "$1" in
        master)
                /usr/local/nginx/sbin/nginx
                exit 0
        ;;
        backup)
                /usr/local/nginx/sbin/nginx -s stop
                /usr/local/nginx/sbin/nginx
                exit 0
        ;;
        fault)
                /usr/local/nginx/sbin/nginx -s stop
        fault)
                /usr/local/nginx/sbin/nginx -s stop
                exit 0
        ;;
        *)
                echo 'Usage: notify.sh {master|backup|fault}'
                exit 1
        ;;
esac
```

#添加执行权限

chmod +x /usr/local/keepalived/sbin/notify.sh

```
global_defs {
```

第 5 章本地 yum 仓库的安装配置

光驱属于文件存储设备，需要给挂载到文件系统中的一个目录下，然后通过这个目录让问里面的文件。

mount –t iso9660 -o ro /dev/cdrom /mnt/cdrom

iso9660 需要挂载的外部存储的类型

-o ro 只读

/dev/cdrom 需要挂载的外部设备

/mnt/cdrom 挂载到的目录

启动 HTTP

service httpd start

HTTP 根目录/var/www/

把挂载的光盘软连接到/var/www/html 下面

ln -s /mnt/cdrom/ ./centos

```
[root@yyb html]# ln -s /mnt/cdrom/ ./centos
[root@yyb html]# ls
centos  hello.html
[root@yyb html]#
```

查看 yum 源

cd /etc/yum.repos.d/

把 yum 源 baseurl 修改为本地,并把 enabled 设置为 1，启用该源

```
# CentOS-Media.repo
#
#  This repo can be used with mounted DVD media, verify the mount point for
#  CentOS-7.  You can use this repo and yum to install items directly off the
#  DVD ISO that we release.
#
# To use this repo, put in your DVD and use it with the other repos too:
#  yum --enablerepo=c7-media [command]
#
# or for ONLY the media repo, do this:
#
#  yum --disablerepo=\* --enablerepo=c7-media [command]

[c7-media]
name=CentOS-$releasever - Media
baseurl=http://127.0.0.1/centos
gpgcheck=0
enabled=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
```

查看 yum 是否生效

yum repolist

```
[root@yyb yum.repos.d]# yum repolist
已加载插件：fastestmirror, langpacks
Loading mirror speeds from cached hostfile
源标识                                          源名称                                状态
c7-media                                        CentOS-yyb                            3,971
repolist: 3,971
[root@yyb yum.repos.d]#
```

Linux 删除软链接

首先我们先来创建一个文件

#mkdir test_chk

#touch test_chk/test.txt

#vim test_chk/test.txt (这一步随便在这个 test.txt 里写点东东即可)

下面我们来创建 test_chk 目录 的软链接

#ln-s test_chk　　test_chk_ln

软链接创建好了，我们来看看怎么删除它

正确的删除方式（删除软链接，但不删除实际数据）

rm -rf　./test_chk_ln

错误的删除方式

rm -rf ./test_chk_ln/ (这样就会把原来 test_chk 下的内容删除)

第 6 章自动化部署脚本

boot.sh

#!/bin/bash

```
SERVERS="mini1 mini2 mini3"
PASSWORD=hadoop
BASE_SERVER=mini4

auto_ssh_copy_id() {
    expect -c "set timeout -1;
        spawn ssh-copy-id $1;
        expect {
            *(yes/no)* {send -- yes\r;exp_continue;}
            *assword:* {send -- $2\r;exp_continue;}
            eof        {exit 0;}
        }";
}

ssh_copy_id_to_all() {
    for SERVER in $SERVERS
    do
        auto_ssh_copy_id $SERVER $PASSWORD
    done
}

ssh_copy_id_to_all


for SERVER in $SERVERS
do
    scp install.sh root@$SERVER:/root
    ssh root@$SERVER /root/install.sh
done
[root@mini4 ~]#
```

Install.sh

```
#!/bin/bash

BASE_SERVER=mini4
yum install -y wget
wget $BASE_SERVER/soft/jdk-7u45-linux-x64.tar.gz
tar -zxvf jdk-7u45-linux-x64.tar.gz -C /usr/local
cat >> /etc/profile << EOF
export JAVA_HOME=/usr/local/jdk1.7.0_45
export PATH=\$PATH:\$JAVA_HOME/bin
EOF
~
~
~
```

第七章 Zookeeper

1. Zookeeper 概念简介：

Zookeeper 是一个分布式协调服务，就是为用户的分布式应用程序提供协调服务

A、zookeeper 是为别的分布式程序服务的

B、Zookeeper 本身就是一个分布式程序（只要有半数以上节点存活，zk 就能正常服务）

C、Zookeeper 所提供的服务涵盖：主从协调、服务器节点动态上下线、统一配置管理、分布式共享锁、统一名称服务……

D、虽然说可以提供各种服务，但是 zookeeper 在底层其实只提供了两个功能：

管理(存储，读取)用户程序提交的数据；

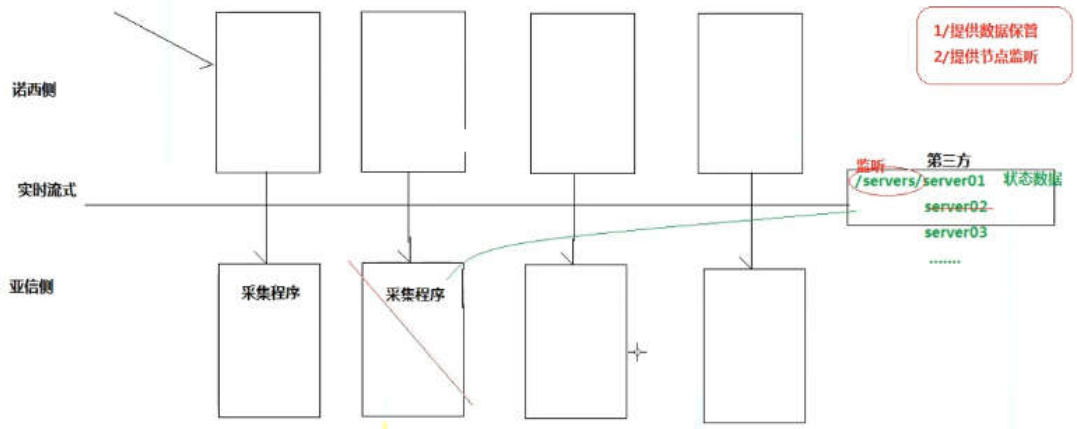
并为用户程序提供数据节点监听服务；

Zookeeper 常用应用场景：

《见图》

Zookeeper 集群的角色： Leader 和 follower （Observer）

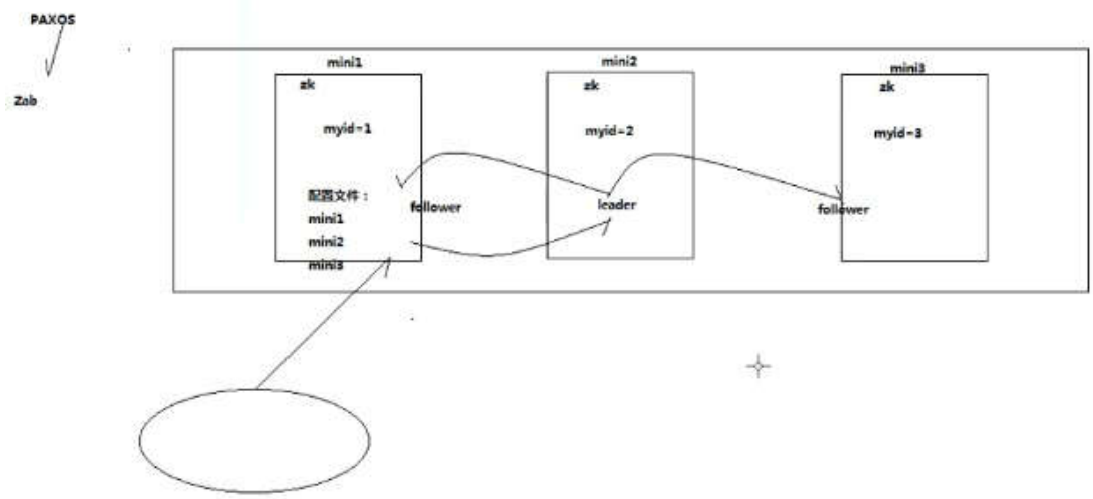
只要集群中有半数以上节点存活，集群就能提供服务



2. zookeeper 集群机制

半数机制，集群中半数以上机器存活，集群可用。

zookeeper 适合装在奇数台机器上!!!



leader 选举机制

leader 选举机制主要是运用了简化的 PAXOS 算法，首先 mini1、mini2、mini3 上面都有一个配置文件 mini1、mini2、mini3 即该集群中的所有主机名，另外还有个每个主机都有一个 myid 编号，选举投票的时候，数字大的优先，票数大于集群中主机数量一半时即可当选为 leader，其他的便为 follower。

当 mini1 启动的时候，他会以广播的形式进行投票，由于集群中就它自己，故最终投票结果为 mini1 一票，少于集群中票数的一半，故选举失败。

但 mini1 依然以广播的方式进行投票，当 mini2 启动的时候，mini1 会给 mini2 一票，自己一票，mini2 也以广播的形式进行投票，自己一票，mini1 一票，最终结果 mini1、mini2 各自两票，不能选举出 leader。继续进行投票，这时会以 myid 编号大优先的原则进行投票，mini1 投票时，考虑到 mini2 的 myid 比自己大，所以只给 mini2 投一票，自己不投自己，mini2 只给自己投一票，最终结果 mini2 两票，mini1 零票，则 mini2 变为 leader，mini1 为 follower。

当 mini3 启动的时候，集群中已有 leader，则不需要投票，自己直接变为 follower，所以三个主机的集群中，myid 最小的主机永远不是 leader。

当集群中有 leader 后，客户端进行数据更新的时候，leader 先更新，然后再把这个信息发给其他的所有 follower 进行更新，当集群非常大的时候，一个 leader 要让所有的 follower 都进行数据更新的时候，需要的时间就非常久，这时候就可能出现延迟的问题。

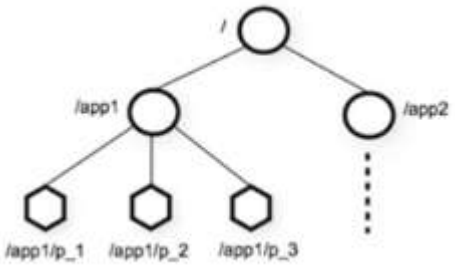
Zookeeper 客户端创建连接的时候需要三个参数：

1、 服务器端主机名及端口号
2、 客户端连接服务器 session 会话的超时时间
3、 Watcher 接口的一个实例，Watcher 接口实例负责接收 Zookeeper 数据变化时产生的事件回调。

4.2. zookeeper 数据结构

1、层次化的目录结构，命名符合常规文件系统规范(见下图)
2、每个节点在 zookeeper 中叫做 znode,并且其有一个唯一的路径标识
3、节点 Znode 可以包含数据和子节点（但是 EPHEMERAL 类型的节点不能有子节点，下一页详细讲解）
4、客户端应用可以在节点上设置监视器（后续详细讲解）

4.3. 数据结构的图



4.4. 节点类型

1、Znode 有两种类型：
短暂（ephemeral）（断开连接自己删除）
持久（persistent）（断开连接不删除）
2、Znode 有四种形式的目录节点（默认是 persistent ）
PERSISTENT
PERSISTENT_SEQUENTIAL（持久序列/test0000000019 ）
EPHEMERAL
EPHEMERAL_SEQUENTIAL
3、创建 znode 时设置顺序标识，znode 名称后会附加一个值，顺序号是一个单调递增的计数器，由父节点维护

短暂型数据节点不能拥有子节点
Zookeeper 的功能无非就两种：
（1） 往里面写数据读数据
（2） 提供监听
Zookeeper 命令

```
[zk: localhost:2181(CONNECTED) 9] help
ZooKeeper -server host:port cmd args
        stat path [watch]
        set path data [version]
        ls path [watch]
        delquota [-n|-b] path
        ls2 path [watch]
        setAcl path acl
        setquota -n|-b val path
        history
        redo cmdno
        printwatches on|off
        delete path [version]
        sync path
        listquota path
        rmr path
        get path [watch]
        create [-s] [-e] path data acl
        addauth scheme auth
        quit
        getAcl path
        close
        connect host:port
```

（1）创建命令 create

create [-s] [-e] path data acl

[-s][-e]可选项，表示创建节点类型，-e 表示短暂节点，-s 表示是否带序号，可以不写

path 文件路径 data 写入的数据 acl 权限，客户端的权限一般都是开放的，不用管

```
[zk: localhost:2181(CONNECTED) 10] create /test "thisistest"
Created /test
[zk: localhost:2181(CONNECTED) 11] ls /
[zookeeper, test, hbase]
[zk: localhost:2181(CONNECTED) 12] get /test
"thisistest"
cZxid = 0x1600000008
ctime = Fri Jan 25 02:13:19 UTC 2019
mZxid = 0x1600000008
mtime = Fri Jan 25 02:13:19 UTC 2019
pZxid = 0x1600000008
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 12
numChildren = 0
```

create -e /test-emphemeral 88888 创建短暂的数据类型节点 内容为 88888

(2)获取文件命令 get

get path [watch] watch 监听功能

```
[zk: localhost:2181(CONNECTED) 16] get /test/app
"172.28.4.1"                    文件内容
cZxid = 0x1600000009            创建数据的节点编号
ctime = Fri Jan 25 02:15:56 UTC 2019    创建数据的时间
mZxid = 0x1600000009            修改数据节点的编号
mtime = Fri Jan 25 02:15:56 UTC 2019    修改数据节点的时间
pZxid = 0x1600000009            保存事务的节点
cversion = 0                    创建数据的版本号
dataVersion = 0                 数据版本号
aclVersion = 0                  权限版本号
ephemeralOwner = 0x0
dataLength = 12                 数据长度
numChildren = 0                 子节点个数
```

（3） 修改文件命令 set

set path data [version]

```
1、使用 ls 命令来查看当前 ZooKeeper 中所包含的内容：
[zk: 202.115.36.251:2181(CONNECTED) 1] ls /
2、创建一个新的 znode，使用 create /zk myData。这个命令创建了一个新的 znode 节点“ zk ”
以及与它关联的字符串：
[zk: 202.115.36.251:2181(CONNECTED) 2] create /zk "myData"
3、我们运行 get 命令来确认 znode 是否包含我们所创建的字符串：
[zk: 202.115.36.251:2181(CONNECTED) 3] get /zk
4、下面我们通过 set 命令来对 zk 所关联的字符串进行设置：
[zk: 202.115.36.251:2181(CONNECTED) 4] set /zk "zsl"
5、下面我们将刚才创建的 znode 删除：
[zk: 202.115.36.251:2181(CONNECTED) 5] delete /zk
6、删除节点：rmr
[zk: 202.115.36.251:2181(CONNECTED) 5] rmr /zk
```

export A=1 定义的变量，会对自己所在的 shell 进程及其子进程生效

B=1 定义的变量，只对自己所在的 shell 进程生效

在 script.sh 中定义的变量，在当前登陆的 shell 进程中 source script.sh 时，脚本中定义的变量也会进入当前登陆的进程

Zookeeper 启动脚本 zkStart.sh

（1）三个主机名为 mini1，mini2，mini3

```
#!/bin/sh
echo "start zkServer..."
for i in 1 2 3
do
ssh mini$i "source /etc/profile;/root/apps/zookeeper-3.4.5/bin/zkServer.sh start"
done
~
~
~
```

（2）三个主机名为 hadoop-master，hadoop-slave1，hadoop-slave2

```
#!/bin/sh
echo "start zkServer..."
ssh hadoop-master "source /etc/profile;/usr/local/zookeeper/bin/zkServer.sh start"
ssh hadoop-slave1 "source /etc/profile;/usr/local/zookeeper/bin/zkServer.sh start"
ssh hadoop-slave2 "source /etc/profile;/usr/local/zookeeper/bin/zkServer.sh start"
```

给启动脚本执行权限 chmod +x zkStart.sh

4.6. zookeeper-api 应用

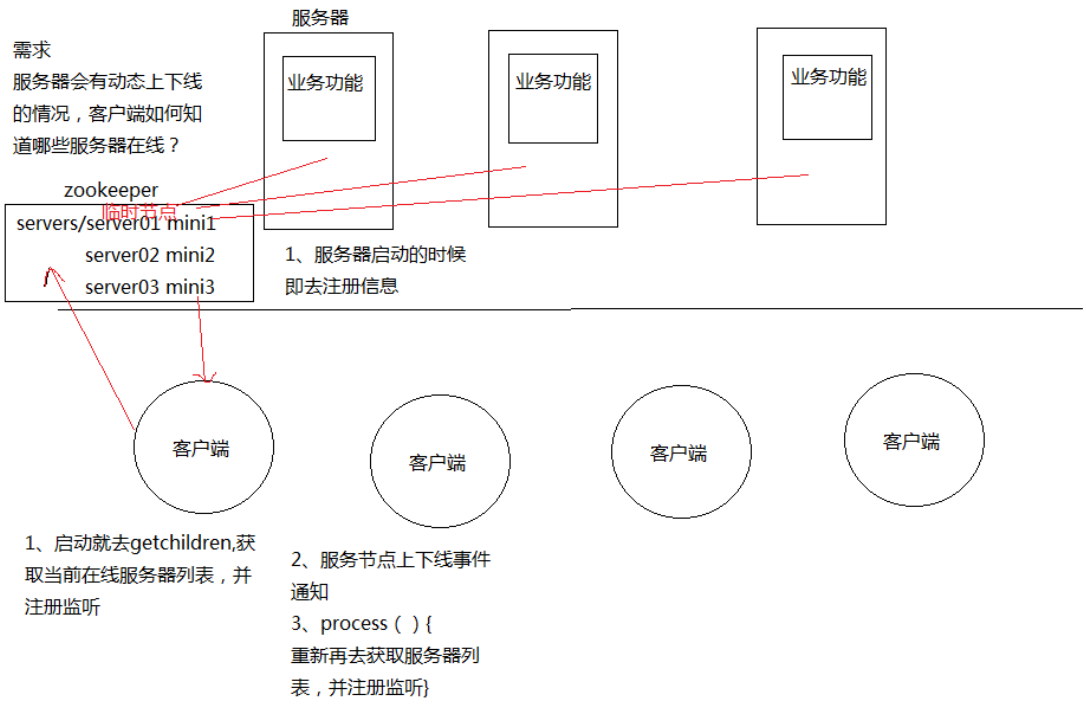
4.6.1. 基本使用

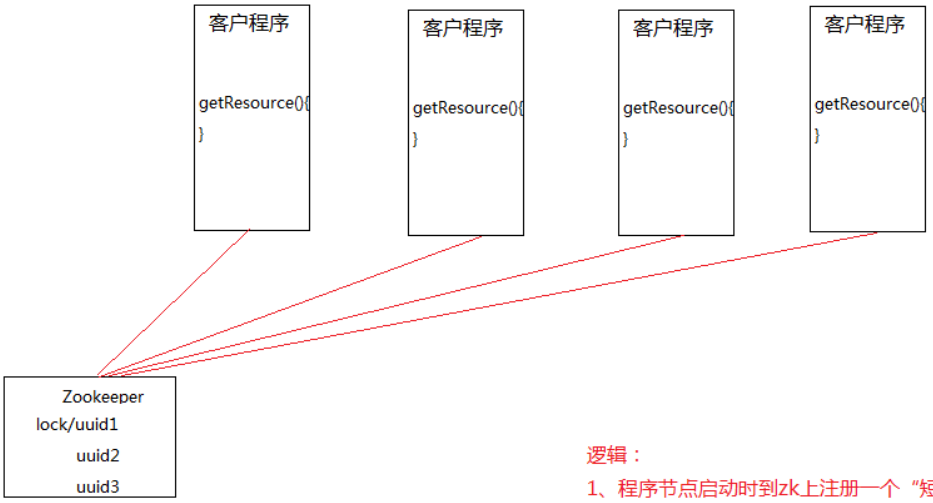
org.apache.zookeeper.Zookeeper 是客户端入口主类，负责建立与 server 的会话
它提供了表 1 所示几类主要方法 ：

功能	描述
create	在本地目录树中创建一个节点
delete	删除一个节点
exists	测试本地是否存在目标节点
get/set data	从目标节点上读取 / 写数据
get/set ACL	获取 / 设置目标节点访问控制列表信息
get children	检索一个子节点上的列表
sync	等待要被传送的数据

表 1 ： ZooKeeper API 描述

idea 右键无法新建 Java Class
https://www.cnblogs.com/zjfjava/p/9219237.html
服务器动态上下线监测





逻辑：

1、程序节点启动时到zk上注册一个“短暂+序号”的znode，并且监听父节点

2、获取父节点下的所有程序子节点，比较序号的大小

3、序号最小的获取到“锁”，去访问资源，访问资源后，删除自己的节点，相当于释放资源，并且重新注册一个新的子节点

4、其他程序节点会收到事件通知，则可以去zk上获取锁

//getConnection

//registerLock(/lock/app.emphemeral_sequential)

//getLock(){获取字节点：比较自己的序号是否是最小的，如果是，则返回锁获取成功}

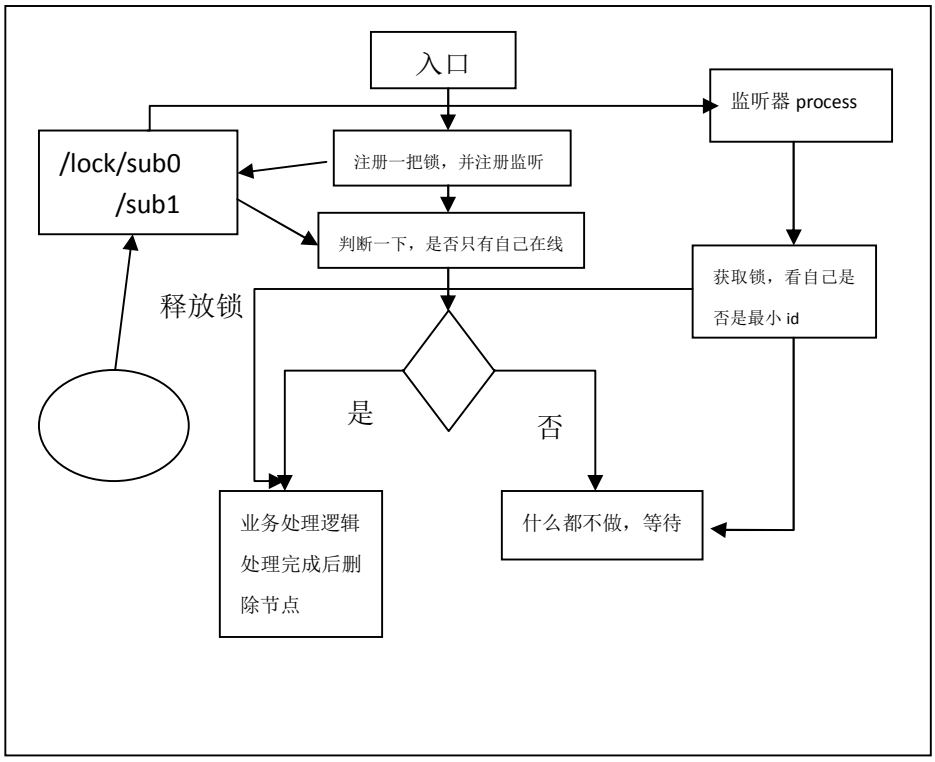
//访问资源

//releaseLock(){删除自己的子节点，并创建一个新的子节点}

//监听器

process(){getLock}

分布式共享锁的实现逻辑



分布式共享锁实现流程

第八章 Java 多线程增强

1.Java 多线程基本知识

%.1.1. 进程介绍

不管是我们开发的应用程序，还是我们运行的其他的应用程序，都需要先把程序安装在本地的硬盘上。然后找到这个程序的启动文件，启动程序的时候，其实是电脑把当前的这个程序加载到内存中，在内存中需要给当前的程序分配一段独立的运行空间。这片空间就专门负责当前这个程序的运行。

不同的应用程序运行的过程中都需要在内存中分配自己独立的运行空间，彼此之间不会相互的影响。我们把每个独立应用程序在内存的独立空间称为当前应用程序运行的一个进程。

进程：它是内存中的一段独立的空间，可以负责当前应用程序的运行。当前这个进程负责调度当前程序中的所有运行细节。

%.1.2. 线程介绍

启动的 QQ 聊天软件，需要和多个人进行聊天。这时多个人之间是不能相互影响，但是它们都位于当前 QQ 这个软件运行时所分配的内存的独立空间中。

在一个进程中，每个独立的功能都需要独立的去运行，这时又需要把当前这个进程划分成多个运行区域，每个独立的小区域（小单元）称为一个线程。

线程：它是位于进程中，负责当前进程中的某个具备独立运行资格的空间。

进程是负责整个程序的运行，而线程是程序中具体的某个独立功能的运行。一个进程中至少应该有一个线程。

%.1.3. 多线程介绍

现在的操作系统基本都是多用户，多任务的操作系统。每个任务就是一个进程。而在这个进程中就会有线程。

真正可以完成程序运行和功能的实现靠的是进程中的线程。

多线程：在一个进程中，我们同时开启多个线程，让多个线程同时去完成某些任务（功能）。(比如后台服务系统，就可以用多个线程同时响应多个客户的请求)

多线程的目的：提高程序的运行效率。

%.1.4. 多线程运行的原理

cpu 在线程中做时间片的切换。

其实真正电脑中的程序的运行不是同时在运行的。CPU 负责程序的运行，而 CPU 在运行程序的过程中某个时刻点上，它其实只能运行一个程序。而不是多个程序。而 CPU 它可以在多个程序之间进行高速的切换。而切换频率和速度太快，导致人的肉眼看不到。每个程序就是进程， 而每个进程中会有多个线程，而 CPU 是在这些线程之间进行切换。了解了 CPU 对一个任务的执行过程，我们就必须知道，多线程可以提高程序的运行效率，但不能无限制的开线程。

%.1.5. 实现线程的两种方式

1、继承 Thread 的方式
见代码 MyThreadWithExtends
2、声明实现 Runnable 接口的方式
见代码 MyThreadWithImpliment

如果调用 thread 的 run 方法，则只是一个普通的方法调用，不会开启新的线程
thread.start()是开启一个新的线程

%.2.java 同步关键词解释

%.2.1. synchronized

加同步格式，

```
synchronized( 需要一个任意的对象（锁） ){
    代码块中放操作共享数据的代码。
}
```

见代码 MySynchronized

- synchronized 的缺陷

synchronized 是 Java 中的一个关键字，也就是说是 Java 语言内置的特性。

如果一个代码块被 synchronized 修饰了，当一个线程获取了对应的锁，并执行该代码块时，其他线程便只能一直等待，等待获取锁的线程释放锁，而这里获取锁的线程释放锁只会有两种情况：

1）获取锁的线程执行完了该代码块，然后线程释放对锁的占有；
2）线程执行发生异常，此时 JVM 会让线程自动释放锁。

例子 1：
如果这个获取锁的线程由于要等待 IO 或者其他原因（比如调用 sleep 方法）被阻塞了，但是又没有释放锁，其他线程便只能干巴巴地等待，试想一下，这多么影响程序执行效率。
因此就需要有一种机制可以不让等待的线程一直无期限地等待下去（比如只等待一定的时间或者能够响应中断），通过 Lock 就可以办到。
例子 2：
当有多个线程读写文件时，读操作和写操作会发生冲突现象，写操作和写操作会发生冲突现象，但是读操作和读操作不会发生冲突现象。
但是采用 synchronized 关键字来实现同步的话，就会导致一个问题：
如果多个线程都只是进行读操作，当一个线程在进行读操作时，其他线程只能等待无法进行读操作。

因此就需要一种机制来使得多个线程都只是进行读操作时，线程之间不会发生冲突，通过 Lock 就可以办到。
另外，通过 Lock 可以知道线程有没有成功获取到锁。这个是 synchronized 无法办到的。
总的来说，也就是说 Lock 提供了比 synchronized 更多的功能。

2.2 lock

- lock 和 synchronized 的区别

1）Lock 不是 Java 语言内置的，synchronized 是 Java 语言的关键字，因此是内置特性。Lock 是一个类，通过这个类可以实现同步访问；
2）Lock 和 synchronized 有一点非常大的不同，采用 synchronized 不需要用户去手动释放锁，当 synchronized 方法或者 synchronized 代码块执行完之后，系统会自动让线程释放对锁的占用；而 Lock 则必须要用户去手动释放锁，如果没有主动释放锁，就有可能导致出现死锁现象。

- java.util.concurrent.locks 包下常用的类

◇ Lock
首先要说明的就是 Lock，通过查看 Lock 的源码可知，Lock 是一个接口：

```
public interface Lock {
    void lock();
    void lockInterruptibly() throws InterruptedException;
    boolean tryLock();
    boolean tryLock(long time, TimeUnit unit) throws InterruptedException;
    void unlock();
    }
```

Lock 接口中每个方法的使用：
lock()、tryLock()、tryLock(long time, TimeUnit unit)、lockInterruptibly()是用来获取锁的。

unLock()方法是用来释放锁的。

四个获取锁方法的区别：

lock()方法是平常使用得最多的一个方法，就是用来获取锁。如果锁已被其他线程获取，则进行等待。

由于在前面讲到如果采用 Lock，必须主动去释放锁，并且在发生异常时，不会自动释放锁。因此一般来说，使用 Lock 必须在 try{}catch{}块中进行，并且将释放锁的操作放在 finally 块中进行，以保证锁一定被被释放，防止死锁的发生。

tryLock()方法是有返回值的，它表示用来尝试获取锁，如果获取成功，则返回 true，如果获取失败（即锁已被其他线程获取），则返回 false，也就说这个方法无论如何都会立即返回。在拿不到锁时不会一直在那等待。

tryLock(long time, TimeUnit unit)方法和 tryLock()方法是类似的，只不过区别在于这个方法在拿不到锁时会等待一定的时间，在时间期限之内如果还拿不到锁，就返回 false。如果如果一开始拿到锁或者在等待期间内拿到了锁，则返回 true。

lockInterruptibly()方法比较特殊，当通过这个方法去获取锁时，如果线程正在等待获取锁，则这个线程能够响应中断，即中断线程的等待状态。也就使说，当两个线程同时通过 lock.lockInterruptibly()想获取某个锁时，假若此时线程 A 获取到了锁，而线程 B 只有等待，那么对线程 B 调用 threadB.interrupt()方法能够中断线程 B 的等待过程。

注意，当一个线程获取了锁之后，是不会被 interrupt()方法中断的。

因此当通过 lockInterruptibly()方法获取某个锁时，如果不能获取到，只有进行等待的情况下，是可以响应中断的。

而用 synchronized 修饰的话，当一个线程处于等待某个锁的状态，是无法被中断的，只有一直等待下去。

- ReentrantLock

  直接使用 lock 接口的话，我们需要实现很多方法，不太方便，ReentrantLock 是唯一实现了 Lock 接口的类，并且 ReentrantLock 提供了更多的方法，ReentrantLock，意思是“可重入锁”。

以下是 ReentrantLock 的使用案例：

例子 1，lock()的正确使用方法
见代码 MyLockTest

例子 2，tryLock()的使用方法
见代码 MyTryLock

例子 3，lockInterruptibly()响应中断的使用方法：
见代码 MyInterruptibly

✧ ReadWriteLock

ReadWriteLock 也是一个接口，在它里面只定义了两个方法：

```
public interface ReadWriteLock {
    /**
     * Returns the lock used for reading.
     *
     * @return the lock used for reading.
     */
    Lock readLock();

    /**
     * Returns the lock used for writing.
     *
     * @return the lock used for writing.
     */
    Lock writeLock();
}
```

一个用来获取读锁，一个用来获取写锁。也就是说将文件的读写操作分开，分成 2 个锁来分配给线程，从而使得多个线程可以同时进行读操作。下面的 ReentrantReadWriteLock 实现了 ReadWriteLock 接口。

✧ ReentrantReadWriteLock

ReentrantReadWriteLock 里面提供了很多丰富的方法，不过最主要的有两个方法，readLock()和 writeLock()用来获取读锁和写锁。

下面通过几个例子来看一下 ReentrantReadWriteLock 具体用法。

例子 1：假如有多个线程要同时进行读操作的话，先看一下 synchronized 达到的效果

见代码 MySynchronizedReadWrite

例子 2：改成用读写锁的话：

见代码 MyReentrantReadWriteLock

注意：

不过要注意的是，如果有一个线程已经占用了读锁，则此时其他线程如果要申请写锁，则申请写锁的线程会一直等待释放读锁。

如果有一个线程已经占用了写锁，则此时其他线程如果申请写锁或者读锁，则申请的线程会一直等待释放写锁。

✧ Lock 和 synchronized 的选择

1）Lock 是一个接口，而 synchronized 是 Java 中的关键字，synchronized 是内置的语言实现；

2）synchronized 在发生异常时，会自动释放线程占有的锁，因此不会导致死锁现象发生；而 Lock 在发生异常时，如果没有主动通过 unLock()去释放锁，则很可能造成死锁现象，因此使用 Lock 时需要在 finally 块中释放锁；

3）Lock 可以让等待锁的线程响应中断，而 synchronized 却不行，使用 synchronized 时，等待的线程会一直等待下去，不能够响应中断；

4）通过 Lock 可以知道有没有成功获取锁，而 synchronized 却无法办到。

5）Lock 可以提高多个线程进行读操作的效率。

在性能上来说，如果竞争资源不激烈，两者的性能是差不多的，而当竞争资源非常激烈时（即有大量线程同时竞争），此时 Lock 的性能要远远优于 synchronized。所以说，在具体使用时要根据适当情况选择。

第九章 Java 并发包

➢ java 并发包

%.1.java 并发包介绍

JDK5.0 以后的版本都引入了高级并发特性，大多数的特性在 java.util.concurrent 包中，是专门用于多线程发编程的，充分利用了现代多处理器和多核心系统的功能以编写大规模并发应用程序。主要包含原子量、并发集合、同步器、可重入锁，并对线程池的构造提供了强力的支持。

线程池

✧ 线程池的 5 中创建方式：

1、 Single Thread Executor：只有一个线程的线程池，因此所有提交的任务是顺序执行，
代码： Executors.newSingleThreadExecutor()

2、 Cached Thread Pool：线程池里有很多线程需要同时执行，老的可用线程将被新的任务触发重新执行，如果线程超过 60 秒内没执行，那么将被终止并从池中删除，
代码：Executors.newCachedThreadPool()

3、 Fixed Thread Pool：拥有固定线程数的线程池，如果没有任务执行，那么线程会一直等待，
代码： Executors.newFixedThreadPool(4)
在构造函数中的参数 4 是线程池的大小，你可以随意设置，也可以和 cpu 的数量保持一致，获取 cpu 的数量 int cpuNums = Runtime.getRuntime().availableProcessors();

4、 Scheduled Thread Pool：用来调度即将执行的任务的线程池，
代码：Executors.newScheduledThreadPool()

5、 Single Thread Scheduled Pool：只有一个线程，用来调度任务在指定时间执行，代码：Executors.newSingleThreadScheduledExecutor()

✧ 线程池的使用

提交 Runnable ，任务完成后 Future 对象返回 null

见代码：ThreadPoolWithRunable

提交 Callable，该方法返回一个 Future 实例表示任务的状态
见代码：ThreadPoolWithcallable