**服务器运维**

**文档更新说明**

**编写人邮箱：jiayu\_zhou007@163.com**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编写人 | 行业 | 用途 | 更新内容 | 版本号 | 日期 |
| Jerry | 教育 | 服务器维护 | 初版 | V1.0 | 2021.7.7 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

本文档旨在为运维相关人员避免总去搜索获取信息，文档中内容均来源于网络

# Linux命令

## grep

强大的文本搜索工具

## 补充说明

**grep** （global search regular expression(RE) and print out the line，全面搜索正则表达式并把行打印出来）是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。用于过滤/搜索的特定字符。可使用正则表达式能多种命令配合使用，使用上十分灵活。

### 选项

-a --text # 不要忽略二进制数据。

-A <显示行数> --after-context=<显示行数> # 除了显示符合范本样式的那一行之外，并显示该行之后的内容。

-b --byte-offset # 在显示符合范本样式的那一行之外，并显示该行之前的内容。

-B<显示行数> --before-context=<显示行数> # 除了显示符合样式的那一行之外，并显示该行之前的内容。

-c --count # 计算符合范本样式的列数。

-C<显示行数> --context=<显示行数>或-<显示行数> # 除了显示符合范本样式的那一列之外，并显示该列之前后的内容。

-d<进行动作> --directories=<动作> # 当指定要查找的是目录而非文件时，必须使用这项参数，否则grep命令将回报信息并停止动作。

-e<范本样式> --regexp=<范本样式> # 指定字符串作为查找文件内容的范本样式。

-E --extended-regexp # 将范本样式为延伸的普通表示法来使用，意味着使用能使用扩展正则表达式。

-f<范本文件> --file=<规则文件> # 指定范本文件，其内容有一个或多个范本样式，让grep查找符合范本条件的文件内容，格式为每一列的范本样式。

-F --fixed-regexp # 将范本样式视为固定字符串的列表。

-G --basic-regexp # 将范本样式视为普通的表示法来使用。

-h --no-filename # 在显示符合范本样式的那一列之前，不标示该列所属的文件名称。

-H --with-filename # 在显示符合范本样式的那一列之前，标示该列的文件名称。

-i --ignore-case # 忽略字符大小写的差别。

-l --file-with-matches # 列出文件内容符合指定的范本样式的文件名称。

-L --files-without-match # 列出文件内容不符合指定的范本样式的文件名称。

-n --line-number # 在显示符合范本样式的那一列之前，标示出该列的编号。

-q --quiet或--silent # 不显示任何信息。

-R/-r --recursive # 此参数的效果和指定“-d recurse”参数相同。

-s --no-messages # 不显示错误信息。

-v --revert-match # 反转查找。

-V --version # 显示版本信息。

-w --word-regexp # 只显示全字符合的列。

-x --line-regexp # 只显示全列符合的列。

-y # 此参数效果跟“-i”相同。

-o # 只输出文件中匹配到的部分。

-m <num> --max-count=<num> # 找到num行结果后停止查找，用来限制匹配行数

### 规则表达式

^ # 锚定行的开始 如：'^grep'匹配所有以grep开头的行。

$ # 锚定行的结束 如：'grep$' 匹配所有以grep结尾的行。

. # 匹配一个非换行符的字符 如：'gr.p'匹配gr后接一个任意字符，然后是p。

\* # 匹配零个或多个先前字符 如：'\*grep'匹配所有一个或多个空格后紧跟grep的行。

.\* # 一起用代表任意字符。

[] # 匹配一个指定范围内的字符，如'[Gg]rep'匹配Grep和grep。

[^] # 匹配一个不在指定范围内的字符，如：'[^A-FH-Z]rep'匹配不包含A-R和T-Z的一个字母开头，紧跟rep的行。

\(..\) # 标记匹配字符，如'\(love\)'，love被标记为1。

\< # 锚定单词的开始，如:'\<grep'匹配包含以grep开头的单词的行。

\> # 锚定单词的结束，如'grep\>'匹配包含以grep结尾的单词的行。

x\{m\} # 重复字符x，m次，如：'0\{5\}'匹配包含5个o的行。

x\{m,\} # 重复字符x,至少m次，如：'o\{5,\}'匹配至少有5个o的行。

x\{m,n\} # 重复字符x，至少m次，不多于n次，如：'o\{5,10\}'匹配5--10个o的行。

\w # 匹配文字和数字字符，也就是[A-Za-z0-9]，如：'G\w\*p'匹配以G后跟零个或多个文字或数字字符，然后是p。

\W # \w的反置形式，匹配一个或多个非单词字符，如点号句号等。

\b # 单词锁定符，如: '\bgrep\b'只匹配grep。

## grep命令常见用法

在文件中搜索一个单词，命令会返回一个包含 **“match\_pattern”** 的文本行：

grep match\_pattern file\_name

grep "match\_pattern" file\_name

在多个文件中查找：

grep "match\_pattern" file\_1 file\_2 file\_3 ...

输出除之外的所有行 **-v** 选项：

grep -v "match\_pattern" file\_name

标记匹配颜色 **--color=auto** 选项：

grep "match\_pattern" file\_name --color=auto

使用正则表达式 **-E** 选项：

grep -E "[1-9]+"

# 或

egrep "[1-9]+"

只输出文件中匹配到的部分 **-o** 选项：

echo this is a test line. | grep -o -E "[a-z]+\."

line.

echo this is a test line. | egrep -o "[a-z]+\."

line.

统计文件或者文本中包含匹配字符串的行数 **-c** 选项：

grep -c "text" file\_name

输出包含匹配字符串的行数 **-n** 选项：

grep "text" -n file\_name

# 或

cat file\_name | grep "text" -n

#多个文件

grep "text" -n file\_1 file\_2

打印样式匹配所位于的字符或字节偏移：

echo gun is not unix | grep -b -o "not"

7:not

#一行中字符串的字符便宜是从该行的第一个字符开始计算，起始值为0。选项 \*\*-b -o\*\* 一般总是配合使用。

搜索多个文件并查找匹配文本在哪些文件中：

grep -l "text" file1 file2 file3...

### grep递归搜索文件

在多级目录中对文本进行递归搜索：

grep "text" . -r -n

# .表示当前目录。

忽略匹配样式中的字符大小写：

echo "hello world" | grep -i "HELLO"

# hello

选项 **-e** 制动多个匹配样式：

echo this is a text line | grep -e "is" -e "line" -o

is

line

#也可以使用 \*\*-f\*\* 选项来匹配多个样式，在样式文件中逐行写出需要匹配的字符。

cat patfile

aaa

bbb

echo aaa bbb ccc ddd eee | grep -f patfile -o

在grep搜索结果中包括或者排除指定文件：

# 只在目录中所有的.php和.html文件中递归搜索字符"main()"

grep "main()" . -r --include \*.{php,html}

# 在搜索结果中排除所有README文件

grep "main()" . -r --exclude "README"

# 在搜索结果中排除filelist文件列表里的文件

grep "main()" . -r --exclude-from filelist

使用0值字节后缀的grep与xargs：

# 测试文件：

echo "aaa" > file1

echo "bbb" > file2

echo "aaa" > file3

grep "aaa" file\* -lZ | xargs -0 rm

# 执行后会删除file1和file3，grep输出用-Z选项来指定以0值字节作为终结符文件名（\0），xargs -0 读取输入并用0值字节终结符分隔文件名，然后删除匹配文件，-Z通常和-l结合使用。

grep静默输出：

grep -q "test" filename

# 不会输出任何信息，如果命令运行成功返回0，失败则返回非0值。一般用于条件测试。

打印出匹配文本之前或者之后的行：

# 显示匹配某个结果之后的3行，使用 -A 选项：

seq 10 | grep "5" -A 3

5

6

7

8

# 显示匹配某个结果之前的3行，使用 -B 选项：

seq 10 | grep "5" -B 3

2

3

4

5

# 显示匹配某个结果的前三行和后三行，使用 -C 选项：

seq 10 | grep "5" -C 3

2

3

4

5

6

7

8

# 如果匹配结果有多个，会用“--”作为各匹配结果之间的分隔符：

echo -e "a\nb\nc\na\nb\nc" | grep a -A 1

a

b

--

a

b

## awk

文本和数据进行处理的编程语言

补充说明

**awk** 是一种编程语言，用于在linux/unix下对文本和数据进行处理。数据可以来自标准输入(stdin)、一个或多个文件，或其它命令的输出。它支持用户自定义函数和动态正则表达式等先进功能，是linux/unix下的一个强大编程工具。它在命令行中使用，但更多是作为脚本来使用。awk有很多内建的功能，比如数组、函数等，这是它和C语言的相同之处，灵活性是awk最大的优势。

awk命令格式和选项

**语法形式**

awk [options] 'script' var=value file(s)

awk [options] -f scriptfile var=value file(s)

**常用命令选项**

* **-F fs** fs指定输入分隔符，fs可以是字符串或正则表达式，如-F:
* **-v var=value** 赋值一个用户定义变量，将外部变量传递给awk
* **-f scripfile** 从脚本文件中读取awk命令
* **-m[fr] val** 对val值设置内在限制，-mf选项限制分配给val的最大块数目；-mr选项限制记录的最大数目。这两个功能是Bell实验室版awk的扩展功能，在标准awk中不适用。

awk模式和操作

awk脚本是由模式和操作组成的。

**模式**

模式可以是以下任意一个：

* /正则表达式/：使用通配符的扩展集。
* 关系表达式：使用运算符进行操作，可以是字符串或数字的比较测试。
* 模式匹配表达式：用运算符~（匹配）和!~（不匹配）。
* BEGIN语句块、pattern语句块、END语句块：参见awk的工作原理

**操作**

操作由一个或多个命令、函数、表达式组成，之间由换行符或分号隔开，并位于大括号内，主要部分是：

* 变量或数组赋值
* 输出命令
* 内置函数
* 控制流语句

awk脚本基本结构

awk 'BEGIN{ print "start" } pattern{ commands } END{ print "end" }' file

一个awk脚本通常由：BEGIN语句块、能够使用模式匹配的通用语句块、END语句块3部分组成，这三个部分是可选的。任意一个部分都可以不出现在脚本中，脚本通常是被 **单引号** 或 **双引号** 中，例如：

awk 'BEGIN{ i=0 } { i++ } END{ print i }' filename

awk "BEGIN{ i=0 } { i++ } END{ print i }" filename

**awk的工作原理**

awk 'BEGIN{ commands } pattern{ commands } END{ commands }'

* 第一步：执行BEGIN{ commands }语句块中的语句；
* 第二步：从文件或标准输入(stdin)读取一行，然后执行pattern{ commands }语句块，它逐行扫描文件，从第一行到最后一行重复这个过程，直到文件全部被读取完毕。
* 第三步：当读至输入流末尾时，执行END{ commands }语句块。

**BEGIN语句块** 在awk开始从输入流中读取行 **之前** 被执行，这是一个可选的语句块，比如变量初始化、打印输出表格的表头等语句通常可以写在BEGIN语句块中。

**END语句块** 在awk从输入流中读取完所有的行 **之后** 即被执行，比如打印所有行的分析结果这类信息汇总都是在END语句块中完成，它也是一个可选语句块。

**pattern语句块** 中的通用命令是最重要的部分，它也是可选的。如果没有提供pattern语句块，则默认执行{ print }，即打印每一个读取到的行，awk读取的每一行都会执行该语句块。

**示例**

echo -e "A line 1nA line 2" | awk 'BEGIN{ print "Start" } { print } END{ print "End" }'

Start

A line 1

A line 2

End

当使用不带参数的print时，它就打印当前行，当print的参数是以逗号进行分隔时，打印时则以空格作为定界符。在awk的print语句块中双引号是被当作拼接符使用，例如：

echo | awk '{ var1="v1"; var2="v2"; var3="v3"; print var1,var2,var3; }'

v1 v2 v3

双引号拼接使用：

echo | awk '{ var1="v1"; var2="v2"; var3="v3"; print var1"="var2"="var3; }'

v1=v2=v3

{ }类似一个循环体，会对文件中的每一行进行迭代，通常变量初始化语句（如：i=0）以及打印文件头部的语句放入BEGIN语句块中，将打印的结果等语句放在END语句块中。

awk内置变量（预定义变量）

说明：[A][N][P][G]表示第一个支持变量的工具，[A]=awk、[N]=nawk、[P]=POSIXawk、[G]=gawk

\*\*$n\*\* 当前记录的第n个字段，比如n为1表示第一个字段，n为2表示第二个字段。

\*\*$0\*\* 这个变量包含执行过程中当前行的文本内容。

[N] \*\*ARGC\*\* 命令行参数的数目。

[G] \*\*ARGIND\*\* 命令行中当前文件的位置（从0开始算）。

[N] \*\*ARGV\*\* 包含命令行参数的数组。

[G] \*\*CONVFMT\*\* 数字转换格式（默认值为%.6g）。

[P] \*\*ENVIRON\*\* 环境变量关联数组。

[N] \*\*ERRNO\*\* 最后一个系统错误的描述。

[G] \*\*FIELDWIDTHS\*\* 字段宽度列表（用空格键分隔）。

[A] \*\*FILENAME\*\* 当前输入文件的名。

[P] \*\*FNR\*\* 同NR，但相对于当前文件。

[A] \*\*FS\*\* 字段分隔符（默认是任何空格）。

[G] \*\*IGNORECASE\*\* 如果为真，则进行忽略大小写的匹配。

[A] \*\*NF\*\* 表示字段数，在执行过程中对应于当前的字段数。

[A] \*\*NR\*\* 表示记录数，在执行过程中对应于当前的行号。

[A] \*\*OFMT\*\* 数字的输出格式（默认值是%.6g）。

[A] \*\*OFS\*\* 输出字段分隔符（默认值是一个空格）。

[A] \*\*ORS\*\* 输出记录分隔符（默认值是一个换行符）。

[A] \*\*RS\*\* 记录分隔符（默认是一个换行符）。

[N] \*\*RSTART\*\* 由match函数所匹配的字符串的第一个位置。

[N] \*\*RLENGTH\*\* 由match函数所匹配的字符串的长度。

[N] \*\*SUBSEP\*\* 数组下标分隔符（默认值是34）。

**示例**

echo -e "line1 f2 f3\nline2 f4 f5\nline3 f6 f7" | awk '{print "Line No:"NR", No of fields:"NF, "$0="$0, "$1="$1, "$2="$2, "$3="$3}'

Line No:1, No of fields:3 $0=line1 f2 f3 $1=line1 $2=f2 $3=f3

Line No:2, No of fields:3 $0=line2 f4 f5 $1=line2 $2=f4 $3=f5

Line No:3, No of fields:3 $0=line3 f6 f7 $1=line3 $2=f6 $3=f7

使用print $NF可以打印出一行中的最后一个字段，使用$(NF-1)则是打印倒数第二个字段，其他以此类推：

echo -e "line1 f2 f3n line2 f4 f5" | awk '{print $NF}'

f3

f5

echo -e "line1 f2 f3n line2 f4 f5" | awk '{print $(NF-1)}'

f2

f4

打印每一行的第二和第三个字段：

awk '{ print $2,$3 }' filename

统计文件中的行数：

awk 'END{ print NR }' filename

以上命令只使用了END语句块，在读入每一行的时，awk会将NR更新为对应的行号，当到达最后一行NR的值就是最后一行的行号，所以END语句块中的NR就是文件的行数。

一个每一行中第一个字段值累加的例子：

seq 5 | awk 'BEGIN{ sum=0; print "总和：" } { print $1"+"; sum+=$1 } END{ print "等于"; print sum }'

总和：

1+

2+

3+

4+

5+

等于

15

将外部变量值传递给awk

借助 **-v选项** ，可以将外部值（并非来自stdin）传递给awk：

VAR=10000

echo | awk -v VARIABLE=$VAR '{ print VARIABLE }'

另一种传递外部变量方法：

var1="aaa"

var2="bbb"

echo | awk '{ print v1,v2 }' v1=$var1 v2=$var2

当输入来自于文件时使用：

awk '{ print v1,v2 }' v1=$var1 v2=$var2 filename

以上方法中，变量之间用空格分隔作为awk的命令行参数跟随在BEGIN、{}和END语句块之后。

查找进程pid

netstat -antup | grep 7770 | awk '{ print $NF NR}' | awk '{ print $1}'

awk运算与判断

作为一种程序设计语言所应具有的特点之一，awk支持多种运算，这些运算与C语言提供的基本相同。awk还提供了一系列内置的运算函数（如log、sqr、cos、sin等）和一些用于对字符串进行操作（运算）的函数（如length、substr等等）。这些函数的引用大大的提高了awk的运算功能。作为对条件转移指令的一部分，关系判断是每种程序设计语言都具备的功能，awk也不例外，awk中允许进行多种测试，作为样式匹配，还提供了模式匹配表达式~（匹配）和!~（不匹配）。作为对测试的一种扩充，awk也支持用逻辑运算符。

**算术运算符**

| **运算符** | **描述** |
| --- | --- |
| + - | 加，减 |
| \* / & | 乘，除与求余 |
| + - ! | 一元加，减和逻辑非 |
| ^ \*\*\* | 求幂 |
| ++ -- | 增加或减少，作为前缀或后缀 |

例：

awk 'BEGIN{a="b";print a++,++a;}'

0 2

注意：所有用作算术运算符进行操作，操作数自动转为数值，所有非数值都变为0

**赋值运算符**

| **运算符** | **描述** |
| --- | --- |
| = += -= \*= /= %= ^= \*\*= | 赋值语句 |

例：

a+=5; 等价于：a=a+5; 其它同类

**逻辑运算符**

| **运算符** | **描述** |
| --- | --- |
| || | 逻辑或 |
| && | 逻辑与 |

例：

awk 'BEGIN{a=1;b=2;print (a>5 && b<=2),(a>5 || b<=2);}'

0 1

**正则运算符**

| **运算符** | **描述** |
| --- | --- |
| ~ !~ | 匹配正则表达式和不匹配正则表达式 |

例：

awk 'BEGIN{a="100testa";if(a ~ /^100\*/){print "ok";}}'

ok

**关系运算符**

| **运算符** | **描述** |
| --- | --- |
| < <= > >= != == | 关系运算符 |

例：

awk 'BEGIN{a=11;if(a >= 9){print "ok";}}'

ok

注意：> < 可以作为字符串比较，也可以用作数值比较，关键看操作数如果是字符串就会转换为字符串比较。两个都为数字才转为数值比较。字符串比较：按照ASCII码顺序比较。

**其它运算符**

| **运算符** | **描述** |
| --- | --- |
| $ | 字段引用 |
| 空格 | 字符串连接符 |
| ?: | C条件表达式 |
| in | 数组中是否存在某键值 |

例：

awk 'BEGIN{a="b";print a=="b"?"ok":"err";}'

ok

awk 'BEGIN{a="b";arr[0]="b";arr[1]="c";print (a in arr);}'

0

awk 'BEGIN{a="b";arr[0]="b";arr["b"]="c";print (a in arr);}'

1

**运算级优先级表**

!级别越高越优先  
级别越高越优先

awk高级输入输出

**读取下一条记录**

awk中next语句使用：在循环逐行匹配，如果遇到next，就会跳过当前行，直接忽略下面语句。而进行下一行匹配。next语句一般用于多行合并：

cat text.txt

a

b

c

d

e

awk 'NR%2==1{next}{print NR,$0;}' text.txt

2 b

4 d

当记录行号除以2余1，就跳过当前行。下面的print NR,$0也不会执行。下一行开始，程序有开始判断NR%2值。这个时候记录行号是：2 ，就会执行下面语句块：'print NR,$0'

分析发现需要将包含有“web”行进行跳过，然后需要将内容与下面行合并为一行：

cat text.txt

web01[192.168.2.100]

httpd ok

tomcat ok

sendmail ok

web02[192.168.2.101]

httpd ok

postfix ok

web03[192.168.2.102]

mysqld ok

httpd ok

0

awk '/^web/{T=$0;next;}{print T":t"$0;}' test.txt

web01[192.168.2.100]: httpd ok

web01[192.168.2.100]: tomcat ok

web01[192.168.2.100]: sendmail ok

web02[192.168.2.101]: httpd ok

web02[192.168.2.101]: postfix ok

web03[192.168.2.102]: mysqld ok

web03[192.168.2.102]: httpd ok

**简单地读取一条记录**

awk getline用法：输出重定向需用到getline函数。getline从标准输入、管道或者当前正在处理的文件之外的其他输入文件获得输入。它负责从输入获得下一行的内容，并给NF,NR和FNR等内建变量赋值。如果得到一条记录，getline函数返回1，如果到达文件的末尾就返回0，如果出现错误，例如打开文件失败，就返回-1。

getline语法：getline var，变量var包含了特定行的内容。

awk getline从整体上来说，用法说明：

* **当其左右无重定向符|或<时：** getline作用于当前文件，读入当前文件的第一行给其后跟的变量var或$0（无变量），应该注意到，由于awk在处理getline之前已经读入了一行，所以getline得到的返回结果是隔行的。
* **当其左右有重定向符|或<时：** getline则作用于定向输入文件，由于该文件是刚打开，并没有被awk读入一行，只是getline读入，那么getline返回的是该文件的第一行，而不是隔行。

**示例：**

执行linux的date命令，并通过管道输出给getline，然后再把输出赋值给自定义变量out，并打印它：

awk 'BEGIN{ "date" | getline out; print out }' test

执行shell的date命令，并通过管道输出给getline，然后getline从管道中读取并将输入赋值给out，split函数把变量out转化成数组mon，然后打印数组mon的第二个元素：

awk 'BEGIN{ "date" | getline out; split(out,mon); print mon[2] }' test

命令ls的输出传递给geline作为输入，循环使getline从ls的输出中读取一行，并把它打印到屏幕。这里没有输入文件，因为BEGIN块在打开输入文件前执行，所以可以忽略输入文件。

awk 'BEGIN{ while( "ls" | getline) print }'

**关闭文件**

awk中允许在程序中关闭一个输入或输出文件，方法是使用awk的close语句。

close("filename")

filename可以是getline打开的文件，也可以是stdin，包含文件名的变量或者getline使用的确切命令。或一个输出文件，可以是stdout，包含文件名的变量或使用管道的确切命令。

**输出到一个文件**

awk中允许用如下方式将结果输出到一个文件：

echo | awk '{printf("hello word!n") > "datafile"}'

# 或

echo | awk '{printf("hello word!n") >> "datafile"}'

设置字段定界符

默认的字段定界符是空格，可以使用-F "定界符" 明确指定一个定界符：

awk -F: '{ print $NF }' /etc/passwd

# 或

awk 'BEGIN{ FS=":" } { print $NF }' /etc/passwd

在BEGIN语句块中则可以用OFS=“定界符”设置输出字段的定界符。

流程控制语句

在linux awk的while、do-while和for语句中允许使用break,continue语句来控制流程走向，也允许使用exit这样的语句来退出。break中断当前正在执行的循环并跳到循环外执行下一条语句。if 是流程选择用法。awk中，流程控制语句，语法结构，与c语言类型。有了这些语句，其实很多shell程序都可以交给awk，而且性能是非常快的。下面是各个语句用法。

**条件判断语句**

if(表达式)

语句1

else

语句2

格式中语句1可以是多个语句，为了方便判断和阅读，最好将多个语句用{}括起来。awk分枝结构允许嵌套，其格式为：

if(表达式)

{语句1}

else if(表达式)

{语句2}

else

{语句3}

示例：

awk 'BEGIN{

test=100;

if(test>90){

print "very good";

}

else if(test>60){

print "good";

}

else{

print "no pass";

}

}'

very good

每条命令语句后面可以用; **分号** 结尾。

**循环语句**

**while语句**

while(表达式)

{语句}

示例：

awk 'BEGIN{

test=100;

total=0;

while(i<=test){

total+=i;

i++;

}

print total;

}'

5050

**for循环**

for循环有两种格式：

格式1：

for(变量 in 数组)

{语句}

示例：

awk 'BEGIN{

for(k in ENVIRON){

print k"="ENVIRON[k];

}

}'

TERM=linux

G\_BROKEN\_FILENAMES=1

SHLVL=1

pwd=/root/text

...

logname=root

HOME=/root

SSH\_CLIENT=192.168.1.21 53087 22

注：ENVIRON是awk常量，是子典型数组。

格式2：

for(变量;条件;表达式)

{语句}

示例：

awk 'BEGIN{

total=0;

for(i=0;i<=100;i++){

total+=i;

}

print total;

}'

5050

**do循环**

do

{语句} while(条件)

例子：

awk 'BEGIN{

total=0;

i=0;

do {total+=i;i++;} while(i<=100)

print total;

}'

5050

**其他语句**

* **break** 当 break 语句用于 while 或 for 语句时，导致退出程序循环。
* **continue** 当 continue 语句用于 while 或 for 语句时，使程序循环移动到下一个迭代。
* **next** 能能够导致读入下一个输入行，并返回到脚本的顶部。这可以避免对当前输入行执行其他的操作过程。
* **exit** 语句使主输入循环退出并将控制转移到END,如果END存在的话。如果没有定义END规则，或在END中应用exit语句，则终止脚本的执行。

数组应用

数组是awk的灵魂，处理文本中最不能少的就是它的数组处理。因为数组索引（下标）可以是数字和字符串在awk中数组叫做关联数组(associative arrays)。awk 中的数组不必提前声明，也不必声明大小。数组元素用0或空字符串来初始化，这根据上下文而定。

**数组的定义**

数字做数组索引（下标）：

Array[1]="sun"

Array[2]="kai"

字符串做数组索引（下标）：

Array["first"]="www"

Array"[last"]="name"

Array["birth"]="1987"

使用中print Array[1]会打印出sun；使用print Array[2]会打印出kai；使用print["birth"]会得到1987。

**读取数组的值**

{ for(item in array) {print array[item]}; } #输出的顺序是随机的

{ for(i=1;i<=len;i++) {print array[i]}; } #Len是数组的长度

**数组相关函数**

**得到数组长度：**

awk 'BEGIN{info="it is a test";lens=split(info,tA," ");print length(tA),lens;}'

4 4

length返回字符串以及数组长度，split进行分割字符串为数组，也会返回分割得到数组长度。

awk 'BEGIN{info="it is a test";split(info,tA," ");print asort(tA);}'

4

asort对数组进行排序，返回数组长度。

**输出数组内容（无序，有序输出）：**

awk 'BEGIN{info="it is a test";split(info,tA," ");for(k in tA){print k,tA[k];}}'

4 test

1 it

2 is

3 a

for…in输出，因为数组是关联数组，默认是无序的。所以通过for…in得到是无序的数组。如果需要得到有序数组，需要通过下标获得。

awk 'BEGIN{info="it is a test";tlen=split(info,tA," ");for(k=1;k<=tlen;k++){print k,tA[k];}}'

1 it

2 is

3 a

4 test

注意：数组下标是从1开始，与C数组不一样。

**判断键值存在以及删除键值：**

# 错误的判断方法：

awk 'BEGIN{tB["a"]="a1";tB["b"]="b1";if(tB["c"]!="1"){print "no found";};for(k in tB){print k,tB[k];}}'

no found

a a1

b b1

c

以上出现奇怪问题，tB[“c”]没有定义，但是循环时候，发现已经存在该键值，它的值为空，这里需要注意，awk数组是关联数组，只要通过数组引用它的key，就会自动创建改序列。

# 正确判断方法：

awk 'BEGIN{tB["a"]="a1";tB["b"]="b1";if( "c" in tB){print "ok";};for(k in tB){print k,tB[k];}}'

a a1

b b1

if(key in array)通过这种方法判断数组中是否包含key键值。

#删除键值：

awk 'BEGIN{tB["a"]="a1";tB["b"]="b1";delete tB["a"];for(k in tB){print k,tB[k];}}'

b b1

delete array[key]可以删除，对应数组key的，序列值。

**二维、多维数组使用**

awk的多维数组在本质上是一维数组，更确切一点，awk在存储上并不支持多维数组。awk提供了逻辑上模拟二维数组的访问方式。例如，array[2,4]=1这样的访问是允许的。awk使用一个特殊的字符串SUBSEP(�34)作为分割字段，在上面的例子中，关联数组array存储的键值实际上是2�344。

类似一维数组的成员测试，多维数组可以使用if ( (i,j) in array)这样的语法，但是下标必须放置在圆括号中。类似一维数组的循环访问，多维数组使用for ( item in array )这样的语法遍历数组。与一维数组不同的是，多维数组必须使用split()函数来访问单独的下标分量。

awk 'BEGIN{

for(i=1;i<=9;i++){

for(j=1;j<=9;j++){

tarr[i,j]=i\*j; print i,"\*",j,"=",tarr[i,j];

}

}

}'

1 \* 1 = 1

1 \* 2 = 2

1 \* 3 = 3

1 \* 4 = 4

1 \* 5 = 5

1 \* 6 = 6

...

9 \* 6 = 54

9 \* 7 = 63

9 \* 8 = 72

9 \* 9 = 81

可以通过array[k,k2]引用获得数组内容。

另一种方法：

awk 'BEGIN{

for(i=1;i<=9;i++){

for(j=1;j<=9;j++){

tarr[i,j]=i\*j;

}

}

for(m in tarr){

split(m,tarr2,SUBSEP); print tarr2[1],"\*",tarr2[2],"=",tarr[m];

}

}'

内置函数

awk内置函数，主要分以下3种类似：算数函数、字符串函数、其它一般函数、时间函数。

**算术函数**

| **格式** | **描述** |
| --- | --- |
| atan2( y, x ) | 返回 y/x 的反正切。 |
| cos( x ) | 返回 x 的余弦；x 是弧度。 |
| sin( x ) | 返回 x 的正弦；x 是弧度。 |
| exp( x ) | 返回 x 幂函数。 |
| log( x ) | 返回 x 的自然对数。 |
| sqrt( x ) | 返回 x 平方根。 |
| int( x ) | 返回 x 的截断至整数的值。 |
| rand( ) | 返回任意数字 n，其中 0 <= n < 1。 |
| srand( [expr] ) | 将 rand 函数的种子值设置为 Expr 参数的值，或如果省略 Expr 参数则使用某天的时间。返回先前的种子值。 |

举例说明：

awk 'BEGIN{OFMT="%.3f";fs=sin(1);fe=exp(10);fl=log(10);fi=int(3.1415);print fs,fe,fl,fi;}'

0.841 22026.466 2.303 3

OFMT 设置输出数据格式是保留3位小数。

获得随机数：

awk 'BEGIN{srand();fr=int(100\*rand());print fr;}'

78

awk 'BEGIN{srand();fr=int(100\*rand());print fr;}'

31

awk 'BEGIN{srand();fr=int(100\*rand());print fr;}'

41

**字符串函数**

| **格式** | **描述** |
| --- | --- |
| gsub( Ere, Repl, [ In ] ) | 除了正则表达式所有具体值被替代这点，它和 sub 函数完全一样地执行。 |
| sub( Ere, Repl, [ In ] ) | 用 Repl 参数指定的字符串替换 In 参数指定的字符串中的由 Ere 参数指定的扩展正则表达式的第一个具体值。sub 函数返回替换的数量。出现在 Repl 参数指定的字符串中的 &（和符号）由 In 参数指定的与 Ere 参数的指定的扩展正则表达式匹配的字符串替换。如果未指定 In 参数，缺省值是整个记录（$0 记录变量）。 |
| index( String1, String2 ) | 在由 String1 参数指定的字符串（其中有出现 String2 指定的参数）中，返回位置，从 1 开始编号。如果 String2 参数不在 String1 参数中出现，则返回 0（零）。 |
| length [(String)] | 返回 String 参数指定的字符串的长度（字符形式）。如果未给出 String 参数，则返回整个记录的长度（$0 记录变量）。 |
| blength [(String)] | 返回 String 参数指定的字符串的长度（以字节为单位）。如果未给出 String 参数，则返回整个记录的长度（$0 记录变量）。 |
| substr( String, M, [ N ] ) | 返回具有 N 参数指定的字符数量子串。子串从 String 参数指定的字符串取得，其字符以 M 参数指定的位置开始。M 参数指定为将 String 参数中的第一个字符作为编号 1。如果未指定 N 参数，则子串的长度将是 M 参数指定的位置到 String 参数的末尾 的长度。 |
| match( String, Ere ) | 在 String 参数指定的字符串（Ere 参数指定的扩展正则表达式出现在其中）中返回位置（字符形式），从 1 开始编号，或如果 Ere 参数不出现，则返回 0（零）。RSTART 特殊变量设置为返回值。RLENGTH 特殊变量设置为匹配的字符串的长度，或如果未找到任何匹配，则设置为 -1（负一）。 |
| split( String, A, [Ere] ) | 将 String 参数指定的参数分割为数组元素 A[1], A[2], . . ., A[n]，并返回 n 变量的值。此分隔可以通过 Ere 参数指定的扩展正则表达式进行，或用当前字段分隔符（FS 特殊变量）来进行（如果没有给出 Ere 参数）。除非上下文指明特定的元素还应具有一个数字值，否则 A 数组中的元素用字符串值来创建。 |
| tolower( String ) | 返回 String 参数指定的字符串，字符串中每个大写字符将更改为小写。大写和小写的映射由当前语言环境的 LC\_CTYPE 范畴定义。 |
| toupper( String ) | 返回 String 参数指定的字符串，字符串中每个小写字符将更改为大写。大写和小写的映射由当前语言环境的 LC\_CTYPE 范畴定义。 |
| sprintf(Format, Expr, Expr, . . . ) | 根据 Format 参数指定的 printf 子例程格式字符串来格式化 Expr 参数指定的表达式并返回最后生成的字符串。 |

注：Ere都可以是正则表达式。

**gsub,sub使用**

awk 'BEGIN{info="this is a test2010test!";gsub(/[0-9]+/,"!",info);print info}'

this is a test!test!

在 info中查找满足正则表达式，/[0-9]+/ 用””替换，并且替换后的值，赋值给info 未给info值，默认是$0

**查找字符串（index使用）**

awk 'BEGIN{info="this is a test2010test!";print index(info,"test")?"ok":"no found";}'

ok

未找到，返回0

**正则表达式匹配查找(match使用）**

awk 'BEGIN{info="this is a test2010test!";print match(info,/[0-9]+/)?"ok":"no found";}'

ok

**截取字符串(substr使用）**

[wangsl@centos5 ~]$ awk 'BEGIN{info="this is a test2010test!";print substr(info,4,10);}'

s is a tes

从第 4个 字符开始，截取10个长度字符串

**字符串分割（split使用）**

awk 'BEGIN{info="this is a test";split(info,tA," ");print length(tA);for(k in tA){print k,tA[k];}}'

4

4 test

1 this

2 is

3 a

分割info，动态创建数组tA，这里比较有意思，awk for …in循环，是一个无序的循环。 并不是从数组下标1…n ，因此使用时候需要注意。

**格式化字符串输出（sprintf使用）**

格式化字符串格式：

其中格式化字符串包括两部分内容：一部分是正常字符，这些字符将按原样输出; 另一部分是格式化规定字符，以"%"开始，后跟一个或几个规定字符,用来确定输出内容格式。

| **格式** | **描述** | **格式** | **描述** |
| --- | --- | --- | --- |
| %d | 十进制有符号整数 | %u | 十进制无符号整数 |
| %f | 浮点数 | %s | 字符串 |
| %c | 单个字符 | %p | 指针的值 |
| %e | 指数形式的浮点数 | %x | %X 无符号以十六进制表示的整数 |
| %o | 无符号以八进制表示的整数 | %g | 自动选择合适的表示法 |

awk 'BEGIN{n1=124.113;n2=-1.224;n3=1.2345; printf("%.2f,%.2u,%.2g,%X,%on",n1,n2,n3,n1,n1);}'

124.11,18446744073709551615,1.2,7C,174

**一般函数**

| **格式** | **描述** |
| --- | --- |
| close( Expression ) | 用同一个带字符串值的 Expression 参数来关闭由 print 或 printf 语句打开的或调用 getline 函数打开的文件或管道。如果文件或管道成功关闭，则返回 0；其它情况下返回非零值。如果打算写一个文件，并稍后在同一个程序中读取文件，则 close 语句是必需的。 |
| system(command ) | 执行 Command 参数指定的命令，并返回退出状态。等同于 system 子例程。 |
| Expression | getline [ Variable ] | 从来自 Expression 参数指定的命令的输出中通过管道传送的流中读取一个输入记录，并将该记录的值指定给 Variable 参数指定的变量。如果当前未打开将 Expression 参数的值作为其命令名称的流，则创建流。创建的流等同于调用 popen 子例程，此时 Command 参数取 Expression 参数的值且 Mode 参数设置为一个是 r 的值。只要流保留打开且 Expression 参数求得同一个字符串，则对 getline 函数的每次后续调用读取另一个记录。如果未指定 Variable 参数，则 $0 记录变量和 NF 特殊变量设置为从流读取的记录。 |
| getline [ Variable ] < Expression | 从 Expression 参数指定的文件读取输入的下一个记录，并将 Variable 参数指定的变量设置为该记录的值。只要流保留打开且 Expression 参数对同一个字符串求值，则对 getline 函数的每次后续调用读取另一个记录。如果未指定 Variable 参数，则 $0 记录变量和 NF 特殊变量设置为从流读取的记录。 |
| getline [ Variable ] | 将 Variable 参数指定的变量设置为从当前输入文件读取的下一个输入记录。如果未指定 Variable 参数，则 $0 记录变量设置为该记录的值，还将设置 NF、NR 和 FNR 特殊变量。 |

**打开外部文件（close用法）**

awk 'BEGIN{while("cat /etc/passwd"|getline){print $0;};close("/etc/passwd");}'

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

**逐行读取外部文件(getline使用方法）**

awk 'BEGIN{while(getline < "/etc/passwd"){print $0;};close("/etc/passwd");}'

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

awk 'BEGIN{print "Enter your name:";getline name;print name;}'

Enter your name:

chengmo

chengmo

**调用外部应用程序(system使用方法）**

awk 'BEGIN{b=system("ls -al");print b;}'

total 42092

drwxr-xr-x 14 chengmo chengmo 4096 09-30 17:47 .

drwxr-xr-x 95 root root 4096 10-08 14:01 ..

b返回值，是执行结果。

**时间函数**

| **格式** | **描述** |
| --- | --- |
| 函数名 | 说明 |
| mktime( YYYY MM dd HH MM ss[ DST]) | 生成时间格式 |
| strftime([format [, timestamp]]) | 格式化时间输出，将时间戳转为时间字符串具体格式，见下表。 |
| systime() | 得到时间戳，返回从1970年1月1日开始到当前时间(不计闰年)的整秒数 |

**建指定时间(mktime使用）**

awk 'BEGIN{tstamp=mktime("2001 01 01 12 12 12");print strftime("%c",tstamp);}'

2001年01月01日 星期一 12时12分12秒

awk 'BEGIN{tstamp1=mktime("2001 01 01 12 12 12");tstamp2=mktime("2001 02 01 0 0 0");print tstamp2-tstamp1;}'

2634468

求2个时间段中间时间差，介绍了strftime使用方法

awk 'BEGIN{tstamp1=mktime("2001 01 01 12 12 12");tstamp2=systime();print tstamp2-tstamp1;}'

308201392

**strftime日期和时间格式说明符**

| **格式** | **描述** |
| --- | --- |
| %a | 星期几的缩写(Sun) |
| %A | 星期几的完整写法(Sunday) |
| %b | 月名的缩写(Oct) |
| %B | 月名的完整写法(October) |
| %c | 本地日期和时间 |
| %d | 十进制日期 |
| %D | 日期 08/20/99 |
| %e | 日期，如果只有一位会补上一个空格 |
| %H | 用十进制表示24小时格式的小时 |
| %I | 用十进制表示12小时格式的小时 |
| %j | 从1月1日起一年中的第几天 |
| %m | 十进制表示的月份 |
| %M | 十进制表示的分钟 |
| %p | 12小时表示法(AM/PM) |
| %S | 十进制表示的秒 |
| %U | 十进制表示的一年中的第几个星期(星期天作为一个星期的开始) |
| %w | 十进制表示的星期几(星期天是0) |
| %W | 十进制表示的一年中的第几个星期(星期一作为一个星期的开始) |
| %x | 重新设置本地日期(08/20/99) |
| %X | 重新设置本地时间(12:00:00) |
| %y | 两位数字表示的年(99) |
| %Y | 当前月份 |
| %% | 百分号(%) |

## sed

功能强大的流式文本编辑器

## 补充说明

**sed** 是一种流编辑器，它是文本处理中非常中的工具，能够完美的配合正则表达式使用，功能不同凡响。处理时，把当前处理的行存储在临时缓冲区中，称为“模式空间”（pattern space），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，这样不断重复，直到文件末尾。文件内容并没有 改变，除非你使用重定向存储输出。Sed主要用来自动编辑一个或多个文件；简化对文件的反复操作；编写转换程序等。

## sed的选项、命令、替换标记

**命令格式**

sed [options] 'command' file(s)

sed [options] -f scriptfile file(s)

### 选项

-e<script>或--expression=<script>：以选项中的指定的script来处理输入的文本文件；

-f<script文件>或--file=<script文件>：以选项中指定的script文件来处理输入的文本文件；

-h或--help：显示帮助；

-n或--quiet或——silent：仅显示script处理后的结果；

-V或--version：显示版本信息。

### 参数

文件：指定待处理的文本文件列表。

### sed命令

a\ # 在当前行下面插入文本。

i\ # 在当前行上面插入文本。

c\ # 把选定的行改为新的文本。

d # 删除，删除选择的行。

D # 删除模板块的第一行。

s # 替换指定字符

h # 拷贝模板块的内容到内存中的缓冲区。

H # 追加模板块的内容到内存中的缓冲区。

g # 获得内存缓冲区的内容，并替代当前模板块中的文本。

G # 获得内存缓冲区的内容，并追加到当前模板块文本的后面。

l # 列表不能打印字符的清单。

n # 读取下一个输入行，用下一个命令处理新的行而不是用第一个命令。

N # 追加下一个输入行到模板块后面并在二者间嵌入一个新行，改变当前行号码。

p # 打印模板块的行。

P # (大写) 打印模板块的第一行。

q # 退出Sed。

b lable # 分支到脚本中带有标记的地方，如果分支不存在则分支到脚本的末尾。

r file # 从file中读行。

t label # if分支，从最后一行开始，条件一旦满足或者T，t命令，将导致分支到带有标号的命令处，或者到脚本的末尾。

T label # 错误分支，从最后一行开始，一旦发生错误或者T，t命令，将导致分支到带有标号的命令处，或者到脚本的末尾。

w file # 写并追加模板块到file末尾。

W file # 写并追加模板块的第一行到file末尾。

! # 表示后面的命令对所有没有被选定的行发生作用。

= # 打印当前行号码。

# # 把注释扩展到下一个换行符以前。

### sed替换标记

g # 表示行内全面替换。

p # 表示打印行。

w # 表示把行写入一个文件。

x # 表示互换模板块中的文本和缓冲区中的文本。

y # 表示把一个字符翻译为另外的字符（但是不用于正则表达式）

\1 # 子串匹配标记

& # 已匹配字符串标记

### sed元字符集

^ # 匹配行开始，如：/^sed/匹配所有以sed开头的行。

$ # 匹配行结束，如：/sed$/匹配所有以sed结尾的行。

. # 匹配一个非换行符的任意字符，如：/s.d/匹配s后接一个任意字符，最后是d。

\* # 匹配0个或多个字符，如：/\*sed/匹配所有模板是一个或多个空格后紧跟sed的行。

[] # 匹配一个指定范围内的字符，如/[ss]ed/匹配sed和Sed。

[^] # 匹配一个不在指定范围内的字符，如：/[^A-RT-Z]ed/匹配不包含A-R和T-Z的一个字母开头，紧跟ed的行。

\(..\) # 匹配子串，保存匹配的字符，如s/\(love\)able/\1rs，loveable被替换成lovers。

& # 保存搜索字符用来替换其他字符，如s/love/ \*\*&\*\* /，love这成 \*\*love\*\* 。

\< # 匹配单词的开始，如:/\<love/匹配包含以love开头的单词的行。

\> # 匹配单词的结束，如/love\>/匹配包含以love结尾的单词的行。

x\{m\} # 重复字符x，m次，如：/0\{5\}/匹配包含5个0的行。

x\{m,\} # 重复字符x，至少m次，如：/0\{5,\}/匹配至少有5个0的行。

x\{m,n\} # 重复字符x，至少m次，不多于n次，如：/0\{5,10\}/匹配5~10个0的行。

## sed用法实例

### 替换操作：s命令

替换文本中的字符串：

sed 's/book/books/' file

**-n选项** 和 **p命令** 一起使用表示只打印那些发生替换的行：

sed -n 's/test/TEST/p' file

直接编辑文件 **选项-i** ，会匹配file文件中每一行的第一个book替换为books：

sed -i 's/book/books/g' file

### 全面替换标记g

使用后缀 /g 标记会替换每一行中的所有匹配：

sed 's/book/books/g' file

当需要从第N处匹配开始替换时，可以使用 /Ng：

echo sksksksksksk | sed 's/sk/SK/2g'

skSKSKSKSKSK

echo sksksksksksk | sed 's/sk/SK/3g'

skskSKSKSKSK

echo sksksksksksk | sed 's/sk/SK/4g'

skskskSKSKSK

### 定界符

以上命令中字符 / 在sed中作为定界符使用，也可以使用任意的定界符：

sed 's:test:TEXT:g'

sed 's|test|TEXT|g'

定界符出现在样式内部时，需要进行转义：

sed 's/\/bin/\/usr\/local\/bin/g'

### 删除操作：d命令

删除空白行：

sed '/^$/d' file

删除文件的第2行：

sed '2d' file

删除文件的第2行到末尾所有行：

sed '2,$d' file

删除文件最后一行：

sed '$d' file

删除文件中所有开头是test的行：

sed '/^test/'d file

### 已匹配字符串标记&

正则表达式 \w+ 匹配每一个单词，使用 [&] 替换它，& 对应于之前所匹配到的单词：

echo this is a test line | sed 's/\w\+/[&]/g'

[this] [is] [a] [test] [line]

所有以192.168.0.1开头的行都会被替换成它自已加localhost：

sed 's/^192.168.0.1/&localhost/' file

192.168.0.1localhost

### 子串匹配标记\1

匹配给定样式的其中一部分：

echo this is digit 7 in a number | sed 's/digit \([0-9]\)/\1/'

this is 7 in a number

命令中 digit 7，被替换成了 7。样式匹配到的子串是 7，(..) 用于匹配子串，对于匹配到的第一个子串就标记为 **\1** ，依此类推匹配到的第二个结果就是 **\2** ，例如：

echo aaa BBB | sed 's/\([a-z]\+\) \([A-Z]\+\)/\2 \1/'

BBB aaa

love被标记为1，所有loveable会被替换成lovers，并打印出来：

sed -n 's/\(love\)able/\1rs/p' file

### 组合多个表达式

sed '表达式' | sed '表达式'

等价于：

sed '表达式; 表达式'

### 引用

sed表达式可以使用单引号来引用，但是如果表达式内部包含变量字符串，就需要使用双引号。

test=hello

echo hello WORLD | sed "s/$test/HELLO"

HELLO WORLD

### 选定行的范围：,（逗号）

所有在模板test和check所确定的范围内的行都被打印：

sed -n '/test/,/check/p' file

打印从第5行开始到第一个包含以test开始的行之间的所有行：

sed -n '5,/^test/p' file

对于模板test和west之间的行，每行的末尾用字符串aaa bbb替换：

sed '/test/,/west/s/$/aaa bbb/' file

### 多点编辑：e命令

-e选项允许在同一行里执行多条命令：

sed -e '1,5d' -e 's/test/check/' file

上面sed表达式的第一条命令删除1至5行，第二条命令用check替换test。命令的执行顺序对结果有影响。如果两个命令都是替换命令，那么第一个替换命令将影响第二个替换命令的结果。

和 -e 等价的命令是 --expression：

sed --expression='s/test/check/' --expression='/love/d' file

### 从文件读入：r命令

file里的内容被读进来，显示在与test匹配的行后面，如果匹配多行，则file的内容将显示在所有匹配行的下面：

sed '/test/r file' filename

### 写入文件：w命令

在example中所有包含test的行都被写入file里：

sed -n '/test/w file' example

### 追加（行下）：a\命令

将 this is a test line 追加到 以test 开头的行后面：

sed '/^test/a\this is a test line' file

在 test.conf 文件第2行之后插入 this is a test line：

sed -i '2a\this is a test line' test.conf

### 插入（行上）：i\命令

将 this is a test line 追加到以test开头的行前面：

sed '/^test/i\this is a test line' file

在test.conf文件第5行之前插入this is a test line：

sed -i '5i\this is a test line' test.conf

### 下一个：n命令

如果test被匹配，则移动到匹配行的下一行，替换这一行的aa，变为bb，并打印该行，然后继续：

sed '/test/{ n; s/aa/bb/; }' file

### 变形：y命令

把1~10行内所有abcde转变为大写，注意，正则表达式元字符不能使用这个命令：

sed '1,10y/abcde/ABCDE/' file

### 退出：q命令

打印完第10行后，退出sed

sed '10q' file

### 保持和获取：h命令和G命令

在sed处理文件的时候，每一行都被保存在一个叫模式空间的临时缓冲区中，除非行被删除或者输出被取消，否则所有被处理的行都将 打印在屏幕上。接着模式空间被清空，并存入新的一行等待处理。

sed -e '/test/h' -e '$G' file

在这个例子里，匹配test的行被找到后，将存入模式空间，h命令将其复制并存入一个称为保持缓存区的特殊缓冲区内。第二条语句的意思是，当到达最后一行后，G命令取出保持缓冲区的行，然后把它放回模式空间中，且追加到现在已经存在于模式空间中的行的末尾。在这个例子中就是追加到最后一行。简单来说，任何包含test的行都被复制并追加到该文件的末尾。

### 保持和互换：h命令和x命令

互换模式空间和保持缓冲区的内容。也就是把包含test与check的行互换：

sed -e '/test/h' -e '/check/x' file

### 脚本scriptfile

sed脚本是一个sed的命令清单，启动Sed时以-f选项引导脚本文件名。Sed对于脚本中输入的命令非常挑剔，在命令的末尾不能有任何空白或文本，如果在一行中有多个命令，要用分号分隔。以#开头的行为注释行，且不能跨行。

sed [options] -f scriptfile file(s)

### 打印奇数行或偶数行

方法1：

sed -n 'p;n' test.txt #奇数行

sed -n 'n;p' test.txt #偶数行

```shell

方法2：

```shell

sed -n '1~2p' test.txt #奇数行

sed -n '2~2p' test.txt #偶数行

```shell

### 打印匹配字符串的下一行

```shell

grep -A 1 SCC URFILE

sed -n '/SCC/{n;p}' URFILE

awk '/SCC/{getline; print}' URFILE

```shell

## ls

显示目录内容列表

## 补充说明

**ls命令** 用来显示目标列表，在Linux中是使用率较高的命令。ls命令的输出信息可以进行彩色加亮显示，以分区不同类型的文件。

### 语法

ls（选项）（参数）

### 选项

-a, --all # 不隐藏任何以. 开始的项目

-A, --almost-all # 列出除. 及.. 以外的任何项目

--author # 与-l 同时使用时列出每个文件的作者

-b, --escape # 以八进制溢出序列表示不可打印的字符

--block-size=SIZE # scale sizes by SIZE before printing them; e.g.,

# '--block-size=M' prints sizes in units of

# 1,048,576 bytes; see SIZE format below

-B, --ignore-backups # do not list implied entries ending with ~

-c # with -lt: sort by, and show, ctime (time of last

# modification of file status information);

# with -l: show ctime and sort by name;

# otherwise: sort by ctime, newest first

-C # list entries by columns

--color[=WHEN] # colorize the output; WHEN can be 'never', 'auto',

# or 'always' (the default); more info below

-d, --directory # list directories themselves, not their contents

-D, --dired # generate output designed for Emacs' dired mode

-f # do not sort, enable -aU, disable -ls --color

-F, --classify # append indicator (one of \*/=>@|) to entries

--file-type # likewise, except do not append '\*'

--format=WORD # across -x, commas -m, horizontal -x, long -l,

# single-column -1, verbose -l, vertical -C

--full-time # like -l --time-style=full-iso

-g # 类似-l，但不列出所有者

--group-directories-first

# group directories before files;

# can be augmented with a --sort option, but any

# use of --sort=none (-U) disables grouping

-G, --no-group # 以一个长列表的形式，不输出组名

-h, --human-readable # 与-l 一起，以易于阅读的格式输出文件大小

(例如 1K 234M 2G)

--si # 同上面类似，但是使用1000 为基底而非1024

-H, --dereference-command-line

# follow symbolic links listed on the command line

--dereference-command-line-symlink-to-dir

# follow each command line symbolic link

# that points to a directory

--hide=PATTERN # do not list implied entries matching shell PATTERN

# (overridden by -a or -A)

--indicator-style=WORD # append indicator with style WORD to entry names:

# none (default), slash (-p),

# file-type (--file-type), classify (-F)

-i, --inode # print the index number of each file

-I, --ignore=PATTERN # do not list implied entries matching shell PATTERN

-k, --kibibytes # default to 1024-byte blocks for disk usage

-l # 使用较长格式列出信息

-L, --dereference # 当显示符号链接的文件信息时，显示符号链接所指示

# 的对象而并非符号链接本身的信息

-m # 所有项目以逗号分隔，并填满整行行宽

-n, --numeric-uid-gid # 类似 -l，但列出UID 及GID 号

-N, --literal # 输出未经处理的项目名称 (如不特别处理控制字符)

-o # 类似 -l，但不列出有关组的信息

-p, --indicator-style=slash # 对目录加上表示符号"/"

-q, --hide-control-chars # print ? instead of nongraphic characters

--show-control-chars # show nongraphic characters as-is (the default,

# unless program is 'ls' and output is a terminal)

-Q, --quote-name # enclose entry names in double quotes

--quoting-style=WORD # use quoting style WORD for entry names:

# literal, locale, shell, shell-always, c, escape

-r, --reverse # 逆序排列

-R, --recursive # 递归显示子目录

-s, --size # 以块数形式显示每个文件分配的尺寸

-S # sort by file size

--sort=WORD # sort by WORD instead of name: none (-U), size (-S),

# time (-t), version (-v), extension (-X)

--time=WORD # with -l, show time as WORD instead of default

# modification time: atime or access or use (-u)

# ctime or status (-c); also use specified time

# as sort key if --sort=time

--time-style=STYLE # with -l, show times using style STYLE:

# full-iso, long-iso, iso, locale, or +FORMAT;

# FORMAT is interpreted like in 'date'; if FORMAT

# is FORMAT1<newline>FORMAT2, then FORMAT1 applies

# to non-recent files and FORMAT2 to recent files;

# if STYLE is prefixed with 'posix-', STYLE

# takes effect only outside the POSIX locale

-t # sort by modification time, newest first

-T, --tabsize=COLS # assume tab stops at each COLS instead of 8

-u # with -lt: sort by, and show, access time;

# with -l: show access time and sort by name;

# otherwise: sort by access time

-U # do not sort; list entries in directory order

-v # natural sort of (version) numbers within text

-w, --width=COLS # assume screen width instead of current value

-x # list entries by lines instead of by columns

-X # sort alphabetically by entry extension

-1 # list one file per line

SELinux options:

--lcontext # Display security context. Enable -l. Lines

# will probably be too wide for most displays.

-Z, --context # Display security context so it fits on most

# displays. Displays only mode, user, group,

# security context and file name.

--scontext # Display only security context and file name.

--help # 显示此帮助信息并退出

--version # 显示版本信息并退出

### 参数

目录：指定要显示列表的目录，也可以是具体的文件。

### 实例

$ ls # 仅列出当前目录可见文件

$ ls -l # 列出当前目录可见文件详细信息

$ ls -hl # 列出详细信息并以可读大小显示文件大小

$ ls -al # 列出所有文件（包括隐藏）的详细信息

显示当前目录下包括影藏文件在内的所有文件列表

[root@localhost ~]# ls -a

. anaconda-ks.cfg .bash\_logout .bashrc install.log .mysql\_history satools .tcshrc .vimrc

.. .bash\_history .bash\_profile .cshrc install.log.syslog .rnd .ssh .viminfo

输出长格式列表

[root@localhost ~]# ls -1

anaconda-ks.cfg

install.log

install.log.syslog

satools

显示文件的inode信息

索引节点（index inode简称为“inode”）是Linux中一个特殊的概念，具有相同的索引节点号的两个文本本质上是同一个文件（除文件名不同外）。

[root@localhost ~]# ls -i -l anaconda-ks.cfg install.log

2345481 -rw------- 1 root root 859 Jun 11 22:49 anaconda-ks.cfg

2345474 -rw-r--r-- 1 root root 13837 Jun 11 22:49 install.log

水平输出文件列表

[root@localhost /]# ls -m

bin, boot, data, dev, etc, home, lib, lost+found, media, misc, mnt, opt, proc, root, sbin, selinux, srv, sys, tmp, usr, var

修改最后一次编辑的文件

最近修改的文件显示在最上面。

[root@localhost /]# ls -t

tmp root etc dev lib boot sys proc data home bin sbin usr var lost+found media mnt opt selinux srv misc

显示递归文件

[root@localhost ~]# ls -R

.:

anaconda-ks.cfg install.log install.log.syslog satools

./satools:

black.txt freemem.sh iptables.sh lnmp.sh mysql php502\_check.sh ssh\_safe.sh

打印文件的UID和GID

[root@localhost /]# ls -n

total 254

drwxr-xr-x 2 0 0 4096 Jun 12 04:03 bin

drwxr-xr-x 4 0 0 1024 Jun 15 14:45 boot

drwxr-xr-x 6 0 0 4096 Jun 12 10:26 data

drwxr-xr-x 10 0 0 3520 Sep 26 15:38 dev

drwxr-xr-x 75 0 0 4096 Oct 16 04:02 etc

drwxr-xr-x 4 0 0 4096 Jun 12 10:26 home

drwxr-xr-x 14 0 0 12288 Jun 16 04:02 lib

drwx------ 2 0 0 16384 Jun 11 22:46 lost+found

drwxr-xr-x 2 0 0 4096 May 11 2011 media

drwxr-xr-x 2 0 0 4096 Nov 8 2010 misc

drwxr-xr-x 2 0 0 4096 May 11 2011 mnt

drwxr-xr-x 2 0 0 4096 May 11 2011 opt

dr-xr-xr-x 232 0 0 0 Jun 15 11:04 proc

drwxr-x--- 4 0 0 4096 Oct 15 14:43 root

drwxr-xr-x 2 0 0 12288 Jun 12 04:03 sbin

drwxr-xr-x 2 0 0 4096 May 11 2011 selinux

drwxr-xr-x 2 0 0 4096 May 11 2011 srv

drwxr-xr-x 11 0 0 0 Jun 15 11:04 sys

drwxrwxrwt 3 0 0 98304 Oct 16 08:45 tmp

drwxr-xr-x 13 0 0 4096 Jun 11 23:38 usr

drwxr-xr-x 19 0 0 4096 Jun 11 23:38 var

列出文件和文件夹的详细信息

[root@localhost /]# ls -l

total 254

drwxr-xr-x 2 root root 4096 Jun 12 04:03 bin

drwxr-xr-x 4 root root 1024 Jun 15 14:45 boot

drwxr-xr-x 6 root root 4096 Jun 12 10:26 data

drwxr-xr-x 10 root root 3520 Sep 26 15:38 dev

drwxr-xr-x 75 root root 4096 Oct 16 04:02 etc

drwxr-xr-x 4 root root 4096 Jun 12 10:26 home

drwxr-xr-x 14 root root 12288 Jun 16 04:02 lib

drwx------ 2 root root 16384 Jun 11 22:46 lost+found

drwxr-xr-x 2 root root 4096 May 11 2011 media

drwxr-xr-x 2 root root 4096 Nov 8 2010 misc

drwxr-xr-x 2 root root 4096 May 11 2011 mnt

drwxr-xr-x 2 root root 4096 May 11 2011 opt

dr-xr-xr-x 232 root root 0 Jun 15 11:04 proc

drwxr-x--- 4 root root 4096 Oct 15 14:43 root

drwxr-xr-x 2 root root 12288 Jun 12 04:03 sbin

drwxr-xr-x 2 root root 4096 May 11 2011 selinux

drwxr-xr-x 2 root root 4096 May 11 2011 srv

drwxr-xr-x 11 root root 0 Jun 15 11:04 sys

drwxrwxrwt 3 root root 98304 Oct 16 08:48 tmp

drwxr-xr-x 13 root root 4096 Jun 11 23:38 usr

drwxr-xr-x 19 root root 4096 Jun 11 23:38 var

列出可读文件和文件夹详细信息

[root@localhost /]# ls -lh

total 254K

drwxr-xr-x 2 root root 4.0K Jun 12 04:03 bin

drwxr-xr-x 4 root root 1.0K Jun 15 14:45 boot

drwxr-xr-x 6 root root 4.0K Jun 12 10:26 data

drwxr-xr-x 10 root root 3.5K Sep 26 15:38 dev

drwxr-xr-x 75 root root 4.0K Oct 16 04:02 etc

drwxr-xr-x 4 root root 4.0K Jun 12 10:26 home

drwxr-xr-x 14 root root 12K Jun 16 04:02 lib

drwx------ 2 root root 16K Jun 11 22:46 lost+found

drwxr-xr-x 2 root root 4.0K May 11 2011 media

drwxr-xr-x 2 root root 4.0K Nov 8 2010 misc

drwxr-xr-x 2 root root 4.0K May 11 2011 mnt

drwxr-xr-x 2 root root 4.0K May 11 2011 opt

dr-xr-xr-x 235 root root 0 Jun 15 11:04 proc

drwxr-x--- 4 root root 4.0K Oct 15 14:43 root

drwxr-xr-x 2 root root 12K Jun 12 04:03 sbin

drwxr-xr-x 2 root root 4.0K May 11 2011 selinux

drwxr-xr-x 2 root root 4.0K May 11 2011 srv

drwxr-xr-x 11 root root 0 Jun 15 11:04 sys

drwxrwxrwt 3 root root 96K Oct 16 08:49 tmp

drwxr-xr-x 13 root root 4.0K Jun 11 23:38 usr

drwxr-xr-x 19 root root 4.0K Jun 11 23:38 var

显示文件夹信息

[root@localhost /]# ls -ld /etc/

drwxr-xr-x 75 root root 4096 Oct 16 04:02 /etc/

按时间列出文件和文件夹详细信息

[root@localhost /]# ls -lt

total 254

drwxrwxrwt 3 root root 98304 Oct 16 08:53 tmp

drwxr-xr-x 75 root root 4096 Oct 16 04:02 etc

drwxr-x--- 4 root root 4096 Oct 15 14:43 root

drwxr-xr-x 10 root root 3520 Sep 26 15:38 dev

drwxr-xr-x 14 root root 12288 Jun 16 04:02 lib

drwxr-xr-x 4 root root 1024 Jun 15 14:45 boot

drwxr-xr-x 11 root root 0 Jun 15 11:04 sys

dr-xr-xr-x 232 root root 0 Jun 15 11:04 proc

drwxr-xr-x 6 root root 4096 Jun 12 10:26 data

drwxr-xr-x 4 root root 4096 Jun 12 10:26 home

drwxr-xr-x 2 root root 4096 Jun 12 04:03 bin

drwxr-xr-x 2 root root 12288 Jun 12 04:03 sbin

drwxr-xr-x 13 root root 4096 Jun 11 23:38 usr

drwxr-xr-x 19 root root 4096 Jun 11 23:38 var

drwx------ 2 root root 16384 Jun 11 22:46 lost+found

drwxr-xr-x 2 root root 4096 May 11 2011 media

drwxr-xr-x 2 root root 4096 May 11 2011 mnt

drwxr-xr-x 2 root root 4096 May 11 2011 opt

drwxr-xr-x 2 root root 4096 May 11 2011 selinux

drwxr-xr-x 2 root root 4096 May 11 2011 srv

drwxr-xr-x 2 root root 4096 Nov 8 2010 misc

按修改时间列出文件和文件夹详细信息

[root@localhost /]# ls -ltr

total 254

drwxr-xr-x 2 root root 4096 Nov 8 2010 misc

drwxr-xr-x 2 root root 4096 May 11 2011 srv

drwxr-xr-x 2 root root 4096 May 11 2011 selinux

drwxr-xr-x 2 root root 4096 May 11 2011 opt

drwxr-xr-x 2 root root 4096 May 11 2011 mnt

drwxr-xr-x 2 root root 4096 May 11 2011 media

drwx------ 2 root root 16384 Jun 11 22:46 lost+found

drwxr-xr-x 19 root root 4096 Jun 11 23:38 var

drwxr-xr-x 13 root root 4096 Jun 11 23:38 usr

drwxr-xr-x 2 root root 12288 Jun 12 04:03 sbin

drwxr-xr-x 2 root root 4096 Jun 12 04:03 bin

drwxr-xr-x 4 root root 4096 Jun 12 10:26 home

drwxr-xr-x 6 root root 4096 Jun 12 10:26 data

dr-xr-xr-x 232 root root 0 Jun 15 11:04 proc

drwxr-xr-x 11 root root 0 Jun 15 11:04 sys

drwxr-xr-x 4 root root 1024 Jun 15 14:45 boot

drwxr-xr-x 14 root root 12288 Jun 16 04:02 lib

drwxr-xr-x 10 root root 3520 Sep 26 15:38 dev

drwxr-x--- 4 root root 4096 Oct 15 14:43 root

drwxr-xr-x 75 root root 4096 Oct 16 04:02 etc

drwxrwxrwt 3 root root 98304 Oct 16 08:54 tmp

按照特殊字符对文件进行分类

[root@localhost nginx-1.2.1]# ls -F

auto/ CHANGES CHANGES.ru conf/ configure\* contrib/ html/ LICENSE Makefile man/ objs/ README src/

列出文件并标记颜色分类

[root@localhost nginx-1.2.1]# ls --color=auto

auto CHANGES CHANGES.ru conf configure contrib html LICENSE Makefile man objs README src

## cd

切换用户当前工作目录

## 补充说明

**cd命令** 用来切换工作目录至dirname。 其中dirName表示法可为绝对路径或相对路径。若目录名称省略，则变换至使用者的home directory(也就是刚login时所在的目录)。另外，~也表示为home directory的意思，.则是表示目前所在的目录，..则表示目前目录位置的上一层目录。

### 语法

cd (选项) (参数)

### 选项

-p 如果要切换到的目标目录是一个符号连接，直接切换到符号连接指向的目标目录

-L 如果要切换的目标目录是一个符号的连接，直接切换到字符连接名代表的目录，而非符号连接所指向的目标目录。

- 当仅实用"-"一个选项时，当前工作目录将被切换到环境变量"OLDPWD"所表示的目录。

### 实例

cd # 进入用户主目录；

cd ~ # 进入用户主目录；

cd - # 返回进入此目录之前所在的目录；

cd .. # 返回上级目录（若当前目录为“/“，则执行完后还在“/"；".."为上级目录的意思）；

cd ../.. # 返回上两级目录；

cd !$ # 把上个命令的参数作为cd参数使用。

## vi

功能强大的纯文本编辑器

## 补充说明

**vi命令** 是UNIX操作系统和类UNIX操作系统中最通用的全屏幕纯文本编辑器。Linux中的vi编辑器叫vim，它是vi的增强版（vi Improved），与vi编辑器完全兼容，而且实现了很多增强功能。

vi编辑器支持编辑模式和命令模式，编辑模式下可以完成文本的编辑功能，命令模式下可以完成对文件的操作命令，要正确使用vi编辑器就必须熟练掌握着两种模式的切换。默认情况下，打开vi编辑器后自动进入命令模式。从编辑模式切换到命令模式使用“esc”键，从命令模式切换到编辑模式使用“A”、“a”、“O”、“o”、“I”、“i”键。

vi编辑器提供了丰富的内置命令，有些内置命令使用键盘组合键即可完成，有些内置命令则需要以冒号“：”开头输入。常用内置命令如下：

Ctrl+u：向文件首翻半屏；

Ctrl+d：向文件尾翻半屏；

Ctrl+f：向文件尾翻一屏；

Ctrl+b：向文件首翻一屏；

Esc：从编辑模式切换到命令模式；

ZZ：命令模式下保存当前文件所做的修改后退出vi；

:行号：光标跳转到指定行的行首；

:$：光标跳转到最后一行的行首；

x或X：删除一个字符，x删除光标后的，而X删除光标前的；

D：删除从当前光标到光标所在行尾的全部字符；

dd：删除光标行正行内容；

ndd：删除当前行及其后n-1行；

nyy：将当前行及其下n行的内容保存到寄存器？中，其中？为一个字母，n为一个数字；

p：粘贴文本操作，用于将缓存区的内容粘贴到当前光标所在位置的下方；

P：粘贴文本操作，用于将缓存区的内容粘贴到当前光标所在位置的上方；

/字符串：文本查找操作，用于从当前光标所在位置开始向文件尾部查找指定字符串的内容，查找的字符串会被加亮显示；

？字符串：文本查找操作，用于从当前光标所在位置开始向文件头部查找指定字符串的内容，查找的字符串会被加亮显示；

a，bs/F/T：替换文本操作，用于在第a行到第b行之间，将F字符串换成T字符串。其中，“s/”表示进行替换操作；

a：在当前字符后添加文本；

A：在行末添加文本；

i：在当前字符前插入文本；

I：在行首插入文本；

o：在当前行后面插入一空行；

O：在当前行前面插入一空行；

:wq：在命令模式下，执行存盘退出操作；

:w：在命令模式下，执行存盘操作；

:w！：在命令模式下，执行强制存盘操作；

:q：在命令模式下，执行退出vi操作；

:q！：在命令模式下，执行强制退出vi操作；

:e文件名：在命令模式下，打开并编辑指定名称的文件；

:n：在命令模式下，如果同时打开多个文件，则继续编辑下一个文件；

:f：在命令模式下，用于显示当前的文件名、光标所在行的行号以及显示比例；

:set number：在命令模式下，用于在最左端显示行号；

:set nonumber：在命令模式下，用于在最左端不显示行号；

### 语法

vi(选项)(参数)

### 选项

+<行号>：从指定行号的行开始显示文本内容；

-b：以二进制模式打开文件，用于编辑二进制文件和可执行文件；

-c<指令>：在完成对第一个文件编辑任务后，执行给出的指令；

-d：以diff模式打开文件，当多个文件编辑时，显示文件差异部分；

-l：使用lisp模式，打开“lisp”和“showmatch”；

-m：取消写文件功能，重设“write”选项；

-M：关闭修改功能；

-n：不实用缓存功能；

-o<文件数目>：指定同时打开指定数目的文件；

-R：以只读方式打开文件；

-s：安静模式，不现实指令的任何错误信息。

### 参数

文件列表：指定要编辑的文件列表。多个文件之间使用空格分隔开。

## 知识扩展

vi编辑器有三种工作方式：命令方式、输入方式和ex转义方式。通过相应的命令或操作，在这三种工作方式之间可以进行转换。

**命令方式**

在Shell提示符后输入命令vi，进入vi编辑器，并处于vi的命令方式。此时，从键盘上输入的任何字符都被作为编辑命令来解释，例如，a(append）表示附加命令，i(insert）表示插入命令，x表示删除字符命令等。如果输入的字符不是vi的合法命令，则机器发出“报警声”，光标不移动。另外，在命令方式下输入的字符（即vi命令）并不在屏幕上显示出来，例如，输入i，屏幕上并无变化，但通过执行i命令，编辑器的工作方式却发生变化：由命令方式变为输入方式。

**输入方式**

通过输入vi的插入命令（i）、附加命令（a）、打开命令（o）、替换命令（s）、修改命令(c）或取代命令（r）可以从命令方式进入输入方式。在输入方式下，从键盘上输入的所有字符都被插入到正在编辑的缓冲区中，被当做该文件的正文。进入输入方式后，输入的可见字符都在屏幕上显示出来，而编辑命令不再起作用，仅作为普通字母出现。例如，在命令方式下输入字母i，进到输入方式，然后再输入i，就在屏幕上相应光标处添加一个字母i。

由输入方式回到命令方式的办法是按下Esc键。如果已在命令方式下，那么按下Esc键就会发出“嘟嘟”声。为了确保用户想执行的vi命令是在命令方式下输入的，不妨多按几下Esc键，听到嘟声后再输入命令。

**ex转义方式**

vi和ex编辑器的功能是相同的，二者的主要区别是用户界面。在vi中，命令通常是单个字母，如a,x,r等。而在ex中，命令是以Enter；键结束的命令行。vi有一个专门的“转义”命令，可访问很多面向行的ex命令。为使用ex转义方式，可输入一个冒号（:）。作为ex命令提示符，冒号出现在状态行（通常在屏幕最下一行）。按下中断键（通常是Del键），可终止正在执行的命令。多数文件管理命令都是在ex转义方式下执行的（例如，读取文件，把编辑缓冲区的内容写到文件中等）。转义命令执行后，自动回到命令方式。例如：

:1,$s/I/i/g 按Enter键

则从文件第一行至文件末尾（$）将大写I全部替换成小写i。vi编辑器的三种工作方式之间的转换如图所示。

!vi

## rm

用于删除给定的文件和目录

## 补充说明

**rm** **命令** 可以删除一个目录中的一个或多个文件或目录，也可以将某个目录及其下属的所有文件及其子目录均删除掉。对于链接文件，只是删除整个链接文件，而原有文件保持不变。

注意：使用rm命令要格外小心。因为一旦删除了一个文件，就无法再恢复它。所以，在删除文件之前，最好再看一下文件的内容，确定是否真要删除。rm命令可以用-i选项，这个选项在使用文件扩展名字符删除多个文件时特别有用。使用这个选项，系统会要求你逐一确定是否要删除。这时，必须输入y并按Enter键，才能删除文件。如果仅按Enter键或其他字符，文件不会被删除。

### 语法

rm (选项)(参数)

### 选项

-d：直接把欲删除的目录的硬连接数据删除成0，删除该目录；

-f：强制删除文件或目录；

-i：删除已有文件或目录之前先询问用户；

-r或-R：递归处理，将指定目录下的所有文件与子目录一并处理；

--preserve-root：不对根目录进行递归操作；

-v：显示指令的详细执行过程。

### 参数

文件：指定被删除的文件列表，如果参数中含有目录，则必须加上-r或者-R选项。

### 实例

交互式删除当前目录下的文件test和example

rm -i test example

Remove test ?n（不删除文件test)

Remove example ?y（删除文件example)

删除当前目录下除隐含文件外的所有文件和子目录

# rm -r \*

应注意，这样做是非常危险的!

**rm 命令删除文件**

# rm 文件1 文件2 ...

rm testfile.txt

**rm 命令删除目录**

rm -r [目录名称] -r 表示递归地删除目录下的所有文件和目录。 -f 表示强制删除

rm -rf testdir

rm -r testdir

**删除操作前有确认提示**

rm -i [文件/目录]

rm -r -i testdir

**rm 忽略不存在的文件或目录**

-f 选项（LCTT 译注：即 “force”）让此次操作强制执行，忽略错误提示

rm -f [文件...]

**仅在某些场景下确认删除**

选项 -I，可保证在删除超过 3 个文件时或递归删除时（LCTT 译注： 如删除目录）仅提示一次确认。

rm -I file1 file2 file3

**删除根目录**

当然，删除根目录（/）是 Linux 用户最不想要的操作，这也就是为什么默认 rm 命令不支持在根目录上执行递归删除操作。 然而，如果你非得完成这个操作，你需要使用 --no-preserve-root 选项。当提供此选项，rm 就不会特殊处理根目录（/）了。

不给实例了，操作系统都被你删除了，你太坏了????

**rm 显示当前删除操作的详情**

rm -v [文件/目录]

## echo

输出指定的字符串或者变量

补充说明

**echo命令** 用于在shell中打印shell变量的值，或者直接输出指定的字符串。linux的echo命令，在shell编程中极为常用, 在终端下打印变量value的时候也是常常用到的，因此有必要了解下echo的用法echo命令的功能是在显示器上显示一段文字，一般起到一个提示的作用。

**语法**

echo(选项)(参数)

**选项**

-e：激活转义字符。

使用-e选项时，若字符串中出现以下字符，则特别加以处理，而不会将它当成一般文字输出：

* \a 发出警告声；
* \b 删除前一个字符；
* \c 不产生进一步输出 (\c 后面的字符不会输出)；
* \f 换行但光标仍旧停留在原来的位置；
* \n 换行且光标移至行首；
* \r 光标移至行首，但不换行；
* \t 插入tab；
* \v 与\f相同；
* \\ 插入\字符；
* \nnn 插入 nnn（八进制）所代表的ASCII字符；

**参数**

变量：指定要打印的变量。

**实例**

用echo命令打印带有色彩的文字：

**文字色：**

echo -e "\e[1;31mThis is red text\e[0m"

This is red text

* \e[1;31m 将颜色设置为红色
* \e[0m 将颜色重新置回

颜色码：重置=0，黑色=30，红色=31，绿色=32，黄色=33，蓝色=34，洋红=35，青色=36，白色=37

**背景色** ：

echo -e "\e[1;42mGreed Background\e[0m"

Greed Background

颜色码：重置=0，黑色=40，红色=41，绿色=42，黄色=43，蓝色=44，洋红=45，青色=46，白色=47

**文字闪动：**

echo -e "\033[37;31;5mMySQL Server Stop...\033[39;49;0m"

红色数字处还有其他数字参数：0 关闭所有属性、1 设置高亮度（加粗）、4 下划线、5 闪烁、7 反显、8 消隐

## set

显示或设置shell特性及shell变量

## 补充说明

**set命令** 作用主要是显示系统中已经存在的shell变量，以及设置shell变量的新变量值。使用set更改shell特性时，符号"+"和"-"的作用分别是打开和关闭指定的模式。set命令不能够定义新的shell变量。如果要定义新的变量，可以使用declare命令以变量名=值的格式进行定义即可。

### 语法

set(选项)(参数)

### 选项

-a：标示已修改的变量，以供输出至环境变量。

-b：使被中止的后台程序立刻回报执行状态。

-C：转向所产生的文件无法覆盖已存在的文件。

-d：Shell预设会用杂凑表记忆使用过的指令，以加速指令的执行。使用-d参数可取消。

-e：若指令传回值不等于0，则立即退出shell。

-f：取消使用通配符。

-h：自动记录函数的所在位置。

-H Shell：可利用"!"加<指令编号>的方式来执行history中记录的指令。

-k：指令所给的参数都会被视为此指令的环境变量。

-l：记录for循环的变量名称。

-m：使用监视模式。

-n：只读取指令，而不实际执行。

-p：启动优先顺序模式。

-P：启动-P参数后，执行指令时，会以实际的文件或目录来取代符号连接。

-t：执行完随后的指令，即退出shell。

-u：当执行时使用到未定义过的变量，则显示错误信息。

-v：显示shell所读取的输入值。

-x：执行指令后，会先显示该指令及所下的参数。

### 参数

取消某个set曾启动的参数。

### 实例

使用declare命令定义一个新的环境变量"mylove"，并且将其值设置为"Visual C++"，输入如下命令：

declare mylove='Visual C++' #定义新环境变量

再使用set命令将新定义的变量输出为环境变量，输入如下命令：

set -a mylove #设置为环境变量

执行该命令后，将会新添加对应的环境变量。用户可以使用env命令和grep命令分别显示和搜索环境变量"mylove"，输入命令如下：

env | grep mylove #显示环境变量值

此时，该命令执行后，将输出查询到的环境变量值。

## env

显示系统中已存在的环境变量

补充说明

**env命令** 用于显示系统中已存在的环境变量，以及在定义的环境中执行指令。该命令只使用"-"作为参数选项时，隐藏了选项"-i"的功能。若没有设置任何选项和参数时，则直接显示当前的环境变量。

如果使用env命令在新环境中执行指令时，会因为没有定义环境变量"PATH"而提示错误信息"such file or directory"。此时，用户可以重新定义一个新的"PATH"或者使用绝对路径。

**语法**

env(选项)(参数)

**选项**

-i：开始一个新的空的环境；

-u<变量名>：从当前环境中删除指定的变量。

**参数**

* 变量定义：定义在新的环境中变量，定义多个变量定义用空格隔开。格式为“变量名=值”；
* 指定：指定要执行的指令和参数。

**实例**

[root@localhost ~]# env

hostname=LinServ-1

TERM=linux

SHELL=/bin/bash

HISTSIZE=1000

SSH\_CLIENT=192.168.2.111 2705 22

SSH\_TTY=/dev/pts/0

USER=root

LS\_COLORS=no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:bd=40;33;01:cd=40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=01;32:\*.cmd=01;32:\*.exe=01;32:\*.com=01;32:\*.btm=01;32:\*.bat=01;32:\*.sh=01;32:\*.csh=01;32:\*.tar=01;31:\*.tgz=01;31:\*.arj=01;31:\*.taz=01;31:\*.lzh=01;31:\*.zip=01;31:\*.z=01;31:\*.Z=01;31:\*.gz=01;31:\*.bz2=01;31:\*.bz=01;31:\*.tz=01;31:\*.rpm=01;31:\*.cpio=01;31:\*.jpg=01;35:\*.gif=01;35:\*.bmp=01;35:\*.xbm=01;35:\*.xpm=01;35:\*.png=01;35:\*.tif=01;35:

mail=/var/spool/mail/root

PATH=/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin

INPUTRC=/etc/inputrc

pwd=/root

LANG=zh\_CN.UTF-8

SHLVL=1

HOME=/root

logname=root

SSH\_CONNECTION=192.168.2.111 2705 192.168.2.2 22

LESSOPEN=|/usr/bin/lesspipe.sh %s

G\_BROKEN\_FILENAMES=1

\_=/bin/env

## cat

连接文件并打印到标准输出设备上

## 补充说明

**cat命令** 连接文件并打印到标准输出设备上，cat经常用来显示文件的内容，类似于下的type命令。

注意：当文件较大时，文本在屏幕上迅速闪过（滚屏），用户往往看不清所显示的内容。因此，一般用more等命令分屏显示。为了控制滚屏，可以按Ctrl+S键，停止滚屏；按Ctrl+Q键可以恢复滚屏。按Ctrl+C（中断）键可以终止该命令的执行，并且返回Shell提示符状态。

### 语法

cat(选项)(参数)

### 选项

-n或--number：从1开始对所有输出的行数编号；

-b或--number-nonblank：和-n相似，只不过对于空白行不编号；

-s或--squeeze-blank：当遇到有连续两行以上的空白行，就代换为一行的空白行；

-A：显示不可打印字符，行尾显示“$”；

-e：等价于"-vE"选项；

-t：等价于"-vT"选项；

### 参数

文件列表：指定要连接的文件列表。

### 实例

设m1和m2是当前目录下的两个文件

cat m1 （在屏幕上显示文件m1的内容）

cat m1 m2 （同时显示文件m1和m2的内容）

cat m1 m2 > file （将文件m1和m2合并后放入文件file中）

## more

显示文件内容，每次显示一屏

补充说明

**more命令** 是一个基于vi编辑器文本过滤器，它以全屏幕的方式按页显示文本文件的内容，支持vi中的关键字定位操作。more名单中内置了若干快捷键，常用的有H（获得帮助信息），Enter（向下翻滚一行），空格（向下滚动一屏），Q（退出命令）。

该命令一次显示一屏文本，满屏后停下来，并且在屏幕的底部出现一个提示信息，给出至今己显示的该文件的百分比：--More--（XX%）可以用下列不同的方法对提示做出回答：

* 按Space键：显示文本的下一屏内容。
* 按Enier键：只显示文本的下一行内容。
* 按斜线符|：接着输入一个模式，可以在文本中寻找下一个相匹配的模式。
* 按H键：显示帮助屏，该屏上有相关的帮助信息。
* 按B键：显示上一屏内容。
* 按Q键：退出rnore命令。

**语法**

more(语法)(参数)

**选项**

-<数字>：指定每屏显示的行数；

-d：显示“[press space to continue,'q' to quit.]”和“[Press 'h' for instructions]”；

-c：不进行滚屏操作。每次刷新这个屏幕；

-s：将多个空行压缩成一行显示；

-u：禁止下划线；

+<数字>：从指定数字的行开始显示。

**参数**

文件：指定分页显示内容的文件。

**实例**

显示文件file的内容，但在显示之前先清屏，并且在屏幕的最下方显示完核的百分比。

more -dc file

显示文件file的内容，每10行显示一次，而且在显示之前先清屏。

more -c -10 file

## tail

在屏幕上显示指定文件的末尾若干行

## 补充说明

**tail命令** 用于输入文件中的尾部内容。tail命令默认在屏幕上显示指定文件的末尾10行。如果给定的文件不止一个，则在显示的每个文件前面加一个文件名标题。如果没有指定文件或者文件名为“-”，则读取标准输入。

注意：如果表示字节或行数的N值之前有一个”+”号，则从文件开头的第N项开始显示，而不是显示文件的最后N项。N值后面可以有后缀：b表示512，k表示1024，m表示1 048576(1M)。

### 语法

tail(选项)(参数)

### 选项

--retry：即是在tail命令启动时，文件不可访问或者文件稍后变得不可访问，都始终尝试打开文件。使用此选项时需要与选项“——follow=name”连用；

-c<N>或——bytes=<N>：输出文件尾部的N（N为整数）个字节内容；

-f<name/descriptor>或；--follow<nameldescript>：显示文件最新追加的内容。“name”表示以文件名的方式监视文件的变化。“-f”与“-fdescriptor”等效；

-F：与选项“-follow=name”和“--retry"连用时功能相同；

-n<N>或——line=<N>：输出文件的尾部N（N位数字）行内容。

--pid=<进程号>：与“-f”选项连用，当指定的进程号的进程终止后，自动退出tail命令；

-q或——quiet或——silent：当有多个文件参数时，不输出各个文件名；

-s<秒数>或——sleep-interal=<秒数>：与“-f”选项连用，指定监视文件变化时间隔的秒数；

-v或——verbose：当有多个文件参数时，总是输出各个文件名；

--help：显示指令的帮助信息；

--version：显示指令的版本信息。

### 参数

文件列表：指定要显示尾部内容的文件列表。

### 实例

tail file #（显示文件file的最后10行）

tail -n +20 file #（显示文件file的内容，从第20行至文件末尾）

tail -c 10 file #（显示文件file的最后10个字符）

tail -25 mail.log # 显示 mail.log 最后的 25 行

tail -f mail.log # 等同于--follow=descriptor，根据文件描述符进行追踪，当文件改名或被删除，追踪停止

tail -F mail.log # 等同于--follow=name --retry，根据文件名进行追踪，并保持重试，即该文件被删除或改名后，如果再次创建相同的文件名，会继续追踪

## less

分屏上下翻页浏览文件内容

## 补充说明

**less命令** 的作用与more十分相似，都可以用来浏览文字档案的内容，不同的是less命令允许用户向前或向后浏览文件，而more命令只能向前浏览。用less命令显示文件时，用PageUp键向上翻页，用PageDown键向下翻页。要退出less程序，应按Q键。

### 语法

less(选项)(参数)

### 选项

-e：文件内容显示完毕后，自动退出；

-f：强制显示文件；

-g：不加亮显示搜索到的所有关键词，仅显示当前显示的关键字，以提高显示速度；

-l：搜索时忽略大小写的差异；

-N：每一行行首显示行号；

-s：将连续多个空行压缩成一行显示；

-S：在单行显示较长的内容，而不换行显示；

-x<数字>：将TAB字符显示为指定个数的空格字符。

### 参数

文件：指定要分屏显示内容的文件。

## 实例

sudo less /var/log/shadowsocks.log

## chgrp

用来变更文件或目录的所属群组

补充说明

**chgrp命令** 用来改变文件或目录所属的用户组。该命令用来改变指定文件所属的用户组。其中，组名可以是用户组的id，也可以是用户组的组名。文件名可以 是由空格分开的要改变属组的文件列表，也可以是由通配符描述的文件集合。如果用户不是该文件的文件主或超级用户(root)，则不能改变该文件的组。

在UNIX系统家族里，文件或目录权限的掌控以拥有者及所属群组来管理。您可以使用chgrp指令去变更文件与目录的所属群组，设置方式采用群组名称或群组识别码皆可。

**语法**

chgrp [选项][组群][文件|目录]

**选项**

-R 递归式地改变指定目录及其下的所有子目录和文件的所属的组

-c或——changes：效果类似“-v”参数，但仅回报更改的部分；

-f或--quiet或——silent：不显示错误信息；

-h或--no-dereference：只对符号连接的文件作修改，而不是该其他任何相关文件；

-H如果命令行参数是一个通到目录的符号链接，则遍历符号链接

-R或——recursive：递归处理，将指令目录下的所有文件及子目录一并处理；

-L遍历每一个遇到的通到目录的符号链接

-P不遍历任何符号链接（默认）

-v或——verbose：显示指令执行过程；

--reference=<参考文件或目录>：把指定文件或目录的所属群组全部设成和参考文件或目录的所属群组相同；

**参数**

* 组：指定新工作名称；
* 文件：指定要改变所属组的文件列表。多个文件或者目录之间使用空格隔开。

**实例**

将/usr/meng及其子目录下的所有文件的用户组改为mengxin

chgrp -R mengxin /usr/meng

更改文件ah的组群所有者为 newuser

[root@rhel ~]# chgrp newuser ah

## chmod

用来变更文件或目录的权限

## 补充说明

**chmod命令** 用来变更文件或目录的权限。在UNIX系统家族里，文件或目录权限的控制分别以读取、写入、执行3种一般权限来区分，另有3种特殊权限可供运用。用户可以使用chmod指令去变更文件与目录的权限，设置方式采用文字或数字代号皆可。符号连接的权限无法变更，如果用户对符号连接修改权限，其改变会作用在被连接的原始文件。

权限范围的表示法如下：

u User，即文件或目录的拥有者；  
g Group，即文件或目录的所属群组；  
o Other，除了文件或目录拥有者或所属群组之外，其他用户皆属于这个范围；  
a All，即全部的用户，包含拥有者，所属群组以及其他用户；  
r 读取权限，数字代号为“4”;  
w 写入权限，数字代号为“2”；  
x 执行或切换权限，数字代号为“1”；  
- 不具任何权限，数字代号为“0”；  
s 特殊功能说明：变更文件或目录的权限。

### 语法

chmod(选项)(参数)

### 选项

u # 操作对象简称，用户user，文件或目录的所有者。

g # 操作对象简称，同组用户group，文件或目录所属群组

o # 操作对象简称，其它用户others

a # 操作对象简称，所有用户all，系统默认使用此项

+ # 权限操作符，添加某些权限

- # 权限操作符，取消某些权限

= # 权限操作符，设置文件的权限为给定的权限

r # 权限设定（英文），表示可读权限

w # 权限设定（英文），表示可写权限

x # 权限设定（英文），表示可执行权限

- # 权限设定（英文字符），表示没有权限

X # 权限设定，如果目标文件是可执行文件或目录，可给其设置可执行权限

s # 权限设定，设置权限suid和sgid，使用权限组合“u+s”设定文件的用户的ID位，“g+s”设置组ID位

t # 权限设定，只有目录或文件的所有者才可以删除目录下的文件

-c或——changes # 效果类似“-v”参数，但仅回报更改的部分，如果文件权限已经改变，显示其操作信息；

-f或--quiet或——silent # 操作过程中不显示任何错误信息；

-R或——recursive # 递归处理，将指令目录下的所有文件及子目录一并处理；

-v或——verbose # 显示命令运行时的详细执行过程；

--reference=<参考文件或目录> # 把指定文件或目录的所属群组全部设成和参考文件或目录的所属群组相同；

<权限范围>+<权限设置> # 开启权限范围的文件或目录的该选项权限设置；

<权限范围>-<权限设置> # 关闭权限范围的文件或目录的该选项权限设置；

<权限范围>=<权限设置> # 指定权限范围的文件或目录的该选项权限设置；

--help # 显示帮助信息

--version # 显示版本信息

### 参数

权限模式：指定文件的权限模式；  
文件：要改变权限的文件。

### 知识扩展和实例

Linux用 户分为：拥有者、组群(Group)、其他（other），Linux系统中，预设的情況下，系统中所有的帐号与一般身份使用者，以及root的相关信 息， 都是记录在/etc/passwd文件中。每个人的密码则是记录在/etc/shadow文件下。 此外，所有的组群名称记录在/etc/group內！

linux文件的用户权限的分析图

-rw-r--r-- 1 user staff 651 Oct 12 12:53 .gitmodules

# ↑╰┬╯╰┬╯╰┬╯

# ┆ ┆ ┆ ╰┈ 0 其他人

# ┆ ┆ ╰┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈ g 属组

# ┆ ╰┈┈┈┈ u 属组

# ╰┈┈ 第一个字母 `d` 代表目录，`-` 代表普通文件

例：rwx　rw-　r--

r=读取属性　　//值＝4  
w=写入属性　　//值＝2  
x=执行属性　　//值＝1

chmod u+x,g+w f01　　# 为文件f01设置自己可以执行，组员可以写入的权限

chmod u=rwx,g=rw,o=r f01

chmod 764 f01

chmod a+x f01　　 # 对文件f01的u,g,o都设置可执行属性

文件的属主和属组属性设置

chown user:market f01　　# 把文件f01给uesr，添加到market组

ll -d f1 查看目录f1的属性

将/home/wwwroot/里的所有文件和文件夹设置为755权限

(1)直接指定路径修改

chmod -R 755 /home/wwwroot/\*

(2)手动进入该目录修改权限（并显示详细过程）

cd /home/wwwroot

chmod -Rv 755 \* #注意：“\*”表示通配符，指的是所有文件和文件

sudo chmod o+rw /shared #设置文件夹权限以保证其余用户可以访问它。

## chown

用来变更文件或目录的拥有者或所属群组

## 补充说明

**chown命令** 改变某个文件或目录的所有者和所属的组，该命令可以向某个用户授权，使该用户变成指定文件的所有者或者改变文件所属的组。用户可以是用户或者是用户D，用户组可以是组名或组id。文件名可以使由空格分开的文件列表，在文件名中可以包含通配符。

只有文件主和超级用户才可以便用该命令。

### 语法

chown(选项)(参数)

### 选项

-c或——changes：效果类似“-v”参数，但仅回报更改的部分；

-f或--quite或——silent：不显示错误信息；

-h或--no-dereference：只对符号连接的文件作修改，而不更改其他任何相关文件；

-R或——recursive：递归处理，将指定目录下的所有文件及子目录一并处理；

-v或——version：显示指令执行过程；

--dereference：效果和“-h”参数相同；

--help：在线帮助；

--reference=<参考文件或目录>：把指定文件或目录的拥有者与所属群组全部设成和参考文件或目录的拥有者与所属群组相同；

--version：显示版本信息。

### 参数

用户：组：指定所有者和所属工作组。当省略“：组”，仅改变文件所有者；  
文件：指定要改变所有者和工作组的文件列表。支持多个文件和目标，支持shell通配符。

### 实例

将目录/usr/meng及其下面的所有文件、子目录的文件主改成 liu：

chown -R liu /usr/meng

## cksum

检查文件的CRC是否正确

## 补充说明

**cksum命令** 是检查文件的CRC是否正确，确保文件从一个系统传输到另一个系统的过程中不被损坏。这种方法要求校验和在源系统中被计算出来，在目的系统中又被计算一次，两个数字进行比较，如果校验和相等，则该文件被认为是正确传输了。

注意：CRC是指一种排错检查方法，即循环冗余校验法。

指定文件交由cksum命令进行校验后，会返回校验结果供用户核对文件是否正确无误。若不指定任何文件名称或是所给予的文件名为"-"，则cksum命令会从标准输入设备中读取数据。

### 语法

cksum(选项)(参数)

### 选项

--help：在线帮助；

--version：显示版本信息。

### 参数

文件：指定要计算校验的版本信息。

### 实例

使用cksum命令计算文件"testfile1"的完整性，输入如下命令：

cksum testfile1 #对指定文件进行CRC校验

以上命令执行后，将输出校验码等相关的信息，具体输出信息如下所示：

1263453430 78 testfile1 #输出信息

上面的输出信息中，"1263453430"表示校验码，"78"表示字节数。

注意：如果文件中有任何字符被修改，都将改变计算后CRC校验码的值。

## cmp

比较两个文件是否有差异

## 补充说明

**cmp命令** 用来比较两个文件是否有差异。当相互比较的两个文件完全一样时，则该指令不会显示任何信息。若发现有差异，预设会标示出第一个不通之处的字符和列数编号。若不指定任何文件名称或是所给予的文件名为“-”，则cmp指令会从标准输入设备读取数据。

### 语法

cmp(选项)(参数)

### 选项

-c或--print-chars：除了标明差异处的十进制字码之外，一并显示该字符所对应字符；

-i<字符数目>或--ignore-initial=<字符数目>：指定一个数目；

-l或——verbose：标示出所有不一样的地方；

-s或--quiet或——silent：不显示错误信息；

-v或——version：显示版本信息；

--help：在线帮助。

### 参数

目录：比较两个文件的差异。

### 实例

使用cmp命令比较文件"testfile"和文件"testfile1"两个文件，则输入下面的命令：

cmp testfile testfile1 #比较两个指定的文件

在上述指令执行之前，使用cat命令查看两个指定的文件内容，如下所示：

cat testfile #查看文件内容

Absncn 50 #显示文件“testfile”

Asldssja 60

Jslkadjls 85

cat testfile1 #查看文件内容

Absncn 50 #显示文件“testfile1”

AsldssjE 62

Jslkadjls 85

然后，再执行cmp命令，并返回比较结果，具体如下所示：

cmp testfile testfile1 #比较两个文件

testfile testfile1 #有差异：第8字节，第2行

## diff

比较给定的两个文件的不同

补充说明

**diff命令** 在最简单的情况下，比较给定的两个文件的不同。如果使用“-”代替“文件”参数，则要比较的内容将来自标准输入。diff命令是以逐行的方式，比较文本文件的异同处。如果该命令指定进行目录的比较，则将会比较该目录中具有相同文件名的文件，而不会对其子目录文件进行任何比较操作。

**语法**

diff(选项)(参数)

**选项**

-<行数>：指定要显示多少行的文本。此参数必须与-c或-u参数一并使用；

-a或——text：diff预设只会逐行比较文本文件；

-b或--ignore-space-change：不检查空格字符的不同；

-B或--ignore-blank-lines：不检查空白行；

-c：显示全部内容，并标出不同之处；

-C<行数>或--context<行数>：与执行“-c-<行数>”指令相同；

-d或——minimal：使用不同的演算法，以小的单位来做比较；

-D<巨集名称>或ifdef<巨集名称>：此参数的输出格式可用于前置处理器巨集；

-e或——ed：此参数的输出格式可用于ed的script文件；

-f或-forward-ed：输出的格式类似ed的script文件，但按照原来文件的顺序来显示不同处；

-H或--speed-large-files：比较大文件时，可加快速度；

-l<字符或字符串>或--ignore-matching-lines<字符或字符串>：若两个文件在某几行有所不同，而之际航同时都包含了选项中指定的字符或字符串，则不显示这两个文件的差异；

-i或--ignore-case：不检查大小写的不同；

-l或——paginate：将结果交由pr程序来分页；

-n或——rcs：将比较结果以RCS的格式来显示；

-N或--new-file：在比较目录时，若文件A仅出现在某个目录中，预设会显示：Only in目录，文件A 若使用-N参数，则diff会将文件A 与一个空白的文件比较；

-p：若比较的文件为C语言的程序码文件时，显示差异所在的函数名称；

-P或--unidirectional-new-file：与-N类似，但只有当第二个目录包含了第一个目录所没有的文件时，才会将这个文件与空白的文件做比较；

-q或--brief：仅显示有无差异，不显示详细的信息；

-r或——recursive：比较子目录中的文件；

-s或--report-identical-files：若没有发现任何差异，仍然显示信息；

-S<文件>或--starting-file<文件>：在比较目录时，从指定的文件开始比较；

-t或--expand-tabs：在输出时，将tab字符展开；

-T或--initial-tab：在每行前面加上tab字符以便对齐；

-u，-U<列数>或--unified=<列数>：以合并的方式来显示文件内容的不同；

-v或——version：显示版本信息；

-w或--ignore-all-space：忽略全部的空格字符；

-W<宽度>或--width<宽度>：在使用-y参数时，指定栏宽；

-x<文件名或目录>或--exclude<文件名或目录>：不比较选项中所指定的文件或目录；

-X<文件>或--exclude-from<文件>；您可以将文件或目录类型存成文本文件，然后在=<文件>中指定此文本文件；

-y或--side-by-side：以并列的方式显示文件的异同之处；

--help：显示帮助；

--left-column：在使用-y参数时，若两个文件某一行内容相同，则仅在左侧的栏位显示该行内容；

--suppress-common-lines：在使用-y参数时，仅显示不同之处。

**参数**

* 文件1：指定要比较的第一个文件；
* 文件2：指定要比较的第二个文件。

**实例**

将目录/usr/li下的文件"test.txt"与当前目录下的文件"test.txt"进行比较，输入如下命令：

diff /usr/li test.txt #使用diff指令对文件进行比较

上面的命令执行后，会将比较后的不同之处以指定的形式列出，如下所示：

n1 a n3,n4

n1,n2 d n3

n1,n2 c n3,n4

其中，字母"a"、"d"、"c"分别表示添加、删除及修改操作。而"n1"、"n2"表示在文件1中的行号，"n3"、"n4"表示在文件2中的行号。

注意：以上说明指定了两个文件中不同处的行号及其相应的操作。在输出形式中，每一行后面将跟随受到影响的若干行。其中，以<开始的行属于文件1，以>开始的行属于文件2。

## file

用来探测给定文件的类型。

## 补充说明

**file命令** 用来探测给定文件的类型。file命令对文件的检查分为文件系统、魔法幻数检查和语言检查3个过程。

### 语法

file(选项)(参数)

### 选项

-b：列出辨识结果时，不显示文件名称；

-c：详细显示指令执行过程，便于排错或分析程序执行的情形；

-f<名称文件>：指定名称文件，其内容有一个或多个文件名称时，让file依序辨识这些文件，格式为每列一个文件名称；

-L：直接显示符号连接所指向的文件类别；

-m<魔法数字文件>：指定魔法数字文件；

-v：显示版本信息；

-z：尝试去解读压缩文件的内容。

### 参数

文件：要确定类型的文件列表，多个文件之间使用空格分开，可以使用shell通配符匹配多个文件。

### 实例

显示文件类型

[root@localhost ~]# file install.log

install.log: UTF-8 Unicode text

[root@localhost ~]# file -b install.log <== 不显示文件名称

UTF-8 Unicode text

[root@localhost ~]# file -i install.log <== 显示MIME类别。

install.log: text/plain; charset=utf-8

[root@localhost ~]# file -b -i install.log

text/plain; charset=utf-8

显示符号链接的文件类型

[root@localhost ~]# ls -l /var/mail

lrwxrwxrwx 1 root root 10 08-13 00:11 /var/mail -> spool/mail

[root@localhost ~]# file /var/mail

/var/mail: symbolic link to `spool/mail'

[root@localhost ~]# file -L /var/mail

/var/mail: directory

[root@localhost ~]# file /var/spool/mail

/var/spool/mail: directory

[root@localhost ~]# file -L /var/spool/mail

/var/spool/mail: directory

## find

在指定目录下查找文件

补充说明

**find命令** 用来在指定目录下查找文件。任何位于参数之前的字符串都将被视为欲查找的目录名。如果使用该命令时，不设置任何参数，则find命令将在当前目录下查找子目录与文件。并且将查找到的子目录和文件全部进行显示。

**语法**

find(选项)(参数)

**选项**

-amin<分钟>：查找在指定时间曾被存取过的文件或目录，单位以分钟计算；

-anewer<参考文件或目录>：查找其存取时间较指定文件或目录的存取时间更接近现在的文件或目录；

-atime<24小时数>：查找在指定时间曾被存取过的文件或目录，单位以24小时计算；

-cmin<分钟>：查找在指定时间之时被更改过的文件或目录；

-cnewer<参考文件或目录>查找其更改时间较指定文件或目录的更改时间更接近现在的文件或目录；

-ctime<24小时数>：查找在指定时间之时被更改的文件或目录，单位以24小时计算；

-daystart：从本日开始计算时间；

-depth：从指定目录下最深层的子目录开始查找；

-expty：寻找文件大小为0 Byte的文件，或目录下没有任何子目录或文件的空目录；

-exec<执行指令>：假设find指令的回传值为True，就执行该指令；

-false：将find指令的回传值皆设为False；

-fls<列表文件>：此参数的效果和指定“-ls”参数类似，但会把结果保存为指定的列表文件；

-follow：排除符号连接；

-fprint<列表文件>：此参数的效果和指定“-print”参数类似，但会把结果保存成指定的列表文件；

-fprint0<列表文件>：此参数的效果和指定“-print0”参数类似，但会把结果保存成指定的列表文件；

-fprintf<列表文件><输出格式>：此参数的效果和指定“-printf”参数类似，但会把结果保存成指定的列表文件；

-fstype<文件系统类型>：只寻找该文件系统类型下的文件或目录；

-gid<群组识别码>：查找符合指定之群组识别码的文件或目录；

-group<群组名称>：查找符合指定之群组名称的文件或目录；

-help或——help：在线帮助；

-ilname<范本样式>：此参数的效果和指定“-lname”参数类似，但忽略字符大小写的差别；

-iname<范本样式>：此参数的效果和指定“-name”参数类似，但忽略字符大小写的差别；

-inum<inode编号>：查找符合指定的inode编号的文件或目录；

-ipath<范本样式>：此参数的效果和指定“-path”参数类似，但忽略字符大小写的差别；

-iregex<范本样式>：此参数的效果和指定“-regexe”参数类似，但忽略字符大小写的差别；

-links<连接数目>：查找符合指定的硬连接数目的文件或目录；

-iname<范本样式>：指定字符串作为寻找符号连接的范本样式；

-ls：假设find指令的回传值为Ture，就将文件或目录名称列出到标准输出；

-maxdepth<目录层级>：设置最大目录层级；

-mindepth<目录层级>：设置最小目录层级；

-mmin<分钟>：查找在指定时间曾被更改过的文件或目录，单位以分钟计算；

-mount：此参数的效果和指定“-xdev”相同；

-mtime<24小时数>：查找在指定时间曾被更改过的文件或目录，单位以24小时计算；

-name<范本样式>：指定字符串作为寻找文件或目录的范本样式；

-newer<参考文件或目录>：查找其更改时间较指定文件或目录的更改时间更接近现在的文件或目录；

-nogroup：找出不属于本地主机群组识别码的文件或目录；

-noleaf：不去考虑目录至少需拥有两个硬连接存在；

-nouser：找出不属于本地主机用户识别码的文件或目录；

-ok<执行指令>：此参数的效果和指定“-exec”类似，但在执行指令之前会先询问用户，若回答“y”或“Y”，则放弃执行命令；

-path<范本样式>：指定字符串作为寻找目录的范本样式；

-perm<权限数值>：查找符合指定的权限数值的文件或目录；

-print：假设find指令的回传值为Ture，就将文件或目录名称列出到标准输出。格式为每列一个名称，每个名称前皆有“./”字符串；

-print0：假设find指令的回传值为Ture，就将文件或目录名称列出到标准输出。格式为全部的名称皆在同一行；

-printf<输出格式>：假设find指令的回传值为Ture，就将文件或目录名称列出到标准输出。格式可以自行指定；

-prune：不寻找字符串作为寻找文件或目录的范本样式;

-regex<范本样式>：指定字符串作为寻找文件或目录的范本样式；

-size<文件大小>：查找符合指定的文件大小的文件；

-true：将find指令的回传值皆设为True；

-typ<文件类型>：只寻找符合指定的文件类型的文件；

-uid<用户识别码>：查找符合指定的用户识别码的文件或目录；

-used<日数>：查找文件或目录被更改之后在指定时间曾被存取过的文件或目录，单位以日计算；

-user<拥有者名称>：查找符和指定的拥有者名称的文件或目录；

-version或——version：显示版本信息；

-xdev：将范围局限在先行的文件系统中；

-xtype<文件类型>：此参数的效果和指定“-type”参数类似，差别在于它针对符号连接检查。

**参数**

起始目录：查找文件的起始目录。

**实例**

# 当前目录搜索所有文件，文件内容 包含 “140.206.111.111” 的内容

find . -type f -name "\*" | xargs grep "140.206.111.111"

**根据文件或者正则表达式进行匹配**

列出当前目录及子目录下所有文件和文件夹

find .

在/home目录下查找以.txt结尾的文件名

find /home -name "\*.txt"

同上，但忽略大小写

find /home -iname "\*.txt"

当前目录及子目录下查找所有以.txt和.pdf结尾的文件

find . \( -name "\*.txt" -o -name "\*.pdf" \)

或

find . -name "\*.txt" -o -name "\*.pdf"

匹配文件路径或者文件

find /usr/ -path "\*local\*"

基于正则表达式匹配文件路径

find . -regex ".\*\(\.txt\|\.pdf\)$"

同上，但忽略大小写

find . -iregex ".\*\(\.txt\|\.pdf\)$"

**否定参数**

找出/home下不是以.txt结尾的文件

find /home ! -name "\*.txt"

**根据文件类型进行搜索**

find . -type 类型参数

类型参数列表：

* **f** 普通文件
* **l** 符号连接
* **d** 目录
* **c** 字符设备
* **b** 块设备
* **s** 套接字
* **p** Fifo

**基于目录深度搜索**

向下最大深度限制为3

find . -maxdepth 3 -type f

搜索出深度距离当前目录至少2个子目录的所有文件

find . -mindepth 2 -type f

**根据文件时间戳进行搜索**

find . -type f 时间戳

UNIX/Linux文件系统每个文件都有三种时间戳：

* **访问时间** （-atime/天，-amin/分钟）：用户最近一次访问时间。
* **修改时间** （-mtime/天，-mmin/分钟）：文件最后一次修改时间。
* **变化时间** （-ctime/天，-cmin/分钟）：文件数据元（例如权限等）最后一次修改时间。

搜索最近七天内被访问过的所有文件

find . -type f -atime -7

搜索恰好在七天前被访问过的所有文件

find . -type f -atime 7

搜索超过七天内被访问过的所有文件

find . -type f -atime +7

搜索访问时间超过10分钟的所有文件

find . -type f -amin +10

找出比file.log修改时间更长的所有文件

find . -type f -newer file.log

**根据文件大小进行匹配**

find . -type f -size 文件大小单元

文件大小单元：

* **b** —— 块（512字节）
* **c** —— 字节
* **w** —— 字（2字节）
* **k** —— 千字节
* **M** —— 兆字节
* **G** —— 吉字节

搜索大于10KB的文件

find . -type f -size +10k

搜索小于10KB的文件

find . -type f -size -10k

搜索等于10KB的文件

find . -type f -size 10k

**删除匹配文件**

删除当前目录下所有.txt文件

find . -type f -name "\*.txt" -delete

**根据文件权限/所有权进行匹配**

当前目录下搜索出权限为777的文件

find . -type f -perm 777

找出当前目录下权限不是644的php文件

find . -type f -name "\*.php" ! -perm 644

找出当前目录用户tom拥有的所有文件

find . -type f -user tom

找出当前目录用户组sunk拥有的所有文件

find . -type f -group sunk

**借助-exec选项与其他命令结合使用**

找出当前目录下所有root的文件，并把所有权更改为用户tom

find .-type f -user root -exec chown tom {} \;

上例中， **{}** 用于与 **-exec** 选项结合使用来匹配所有文件，然后会被替换为相应的文件名。

找出自己家目录下所有的.txt文件并删除

find $HOME/. -name "\*.txt" -ok rm {} \;

上例中， **-ok** 和 **-exec** 行为一样，不过它会给出提示，是否执行相应的操作。

查找当前目录下所有.txt文件并把他们拼接起来写入到all.txt文件中

find . -type f -name "\*.txt" -exec cat {} \;> all.txt

将30天前的.log文件移动到old目录中

find . -type f -mtime +30 -name "\*.log" -exec cp {} old \;

找出当前目录下所有.txt文件并以“File:文件名”的形式打印出来

find . -type f -name "\*.txt" -exec printf "File: %s\n" {} \;

因为单行命令中-exec参数中无法使用多个命令，以下方法可以实现在-exec之后接受多条命令

-exec ./text.sh {} \;

**搜索但跳出指定的目录**

查找当前目录或者子目录下所有.txt文件，但是跳过子目录sk

find . -path "./sk" -prune -o -name "\*.txt" -print

**find其他技巧收集**

要列出所有长度为零的文件

find . -empty

**其它实例**

find ~ -name '\*jpg' # 主目录中找到所有的 jpg 文件。 -name 参数允许你将结果限制为与给定模式匹配的文件。

find ~ -iname '\*jpg' # -iname 就像 -name，但是不区分大小写

find ~ ( -iname 'jpeg' -o -iname 'jpg' ) # 一些图片可能是 .jpeg 扩展名。幸运的是，我们可以将模式用“或”（表示为 -o）来组合。

find ~ \( -iname '\*jpeg' -o -iname '\*jpg' \) -type f # 如果你有一些以 jpg 结尾的目录呢？ （为什么你要命名一个 bucketofjpg 而不是 pictures 的目录就超出了本文的范围。）我们使用 -type 参数修改我们的命令来查找文件。

find ~ \( -iname '\*jpeg' -o -iname '\*jpg' \) -type d # 也许你想找到那些命名奇怪的目录，以便稍后重命名它们

最近拍了很多照片，所以让我们把它缩小到上周更改的文件

find ~ \( -iname '\*jpeg' -o -iname '\*jpg' \) -type f -mtime -7

你可以根据文件状态更改时间 （ctime）、修改时间 （mtime） 或访问时间 （atime） 来执行时间过滤。 这些是在几天内，所以如果你想要更细粒度的控制，你可以表示为在几分钟内（分别是 cmin、mmin 和 amin）。 除非你确切地知道你想要的时间，否则你可能会在 + （大于）或 - （小于）的后面加上数字。

但也许你不关心你的照片。也许你的磁盘空间不够用，所以你想在 log 目录下找到所有巨大的（让我们定义为“大于 1GB”）文件：

find /var/log -size +1G

或者，也许你想在 /data 中找到 bcotton 拥有的所有文件：

find /data -owner bcotton

你还可以根据权限查找文件。也许你想在你的主目录中找到对所有人可读的文件，以确保你不会过度分享。

find ~ -perm -o=r

删除 mac 下自动生成的文件

find ./ -name '\_\_MACOSX' -depth -exec rm -rf {} \;

统计代码行数

find . -name "\*.java"|xargs cat|grep -v ^$|wc -l # 代码行数统计, 排除空行

## git

是目前世界上最先进的分布式版本控制系统

补充说明

**git命令** 很多人都知道，Linus在1991年创建了开源的Linux，从此，Linux系统不断发展，已经成为最大的服务器系统软件了。

Linus虽然创建了Linux，但Linux的壮大是靠全世界热心的志愿者参与的，这么多人在世界各地为Linux编写代码，那Linux的代码是如何管理的呢？

事实是，在2002年以前，世界各地的志愿者把源代码文件通过diff的方式发给Linus，然后由Linus本人通过手工方式合并代码！

你也许会想，为什么Linus不把Linux代码放到版本控制系统里呢？不是有CVS、SVN这些免费的版本控制系统吗？因为Linus坚定地反对CVS和SVN，这些集中式的版本控制系统不但速度慢，而且必须联网才能使用。有一些商用的版本控制系统，虽然比CVS、SVN好用，但那是付费的，和Linux的开源精神不符。

不过，到了2002年，Linux系统已经发展了十年了，代码库之大让Linus很难继续通过手工方式管理了，社区的弟兄们也对这种方式表达了强烈不满，于是Linus选择了一个商业的版本控制系统BitKeeper，BitKeeper的东家BitMover公司出于人道主义精神，授权Linux社区免费使用这个版本控制系统。

安定团结的大好局面在2005年就被打破了，原因是Linux社区牛人聚集，不免沾染了一些梁山好汉的江湖习气。开发Samba的Andrew试图破解BitKeeper的协议（这么干的其实也不只他一个），被BitMover公司发现了（监控工作做得不错！），于是BitMover公司怒了，要收回Linux社区的免费使用权。

Linus可以向BitMover公司道个歉，保证以后严格管教弟兄们，嗯，这是不可能的。实际情况是这样的：

Linus花了两周时间自己用C写了一个分布式版本控制系统，这就是Git！一个月之内，Linux系统的源码已经由Git管理了！牛是怎么定义的呢？大家可以体会一下。

Git迅速成为最流行的分布式版本控制系统，尤其是2008年，GitHub网站上线了，它为开源项目免费提供Git存储，无数开源项目开始迁移至GitHub，包括jQuery，PHP，Ruby等等。

历史就是这么偶然，如果不是当年BitMover公司威胁Linux社区，可能现在我们就没有免费而超级好用的Git了。

[Git常用命令清单](https://github.com/jaywcjlove/handbook/blob/master/other/Git%E5%B8%B8%E7%94%A8%E5%91%BD%E4%BB%A4%E6%B8%85%E5%8D%95.md)

**语法**

git [--version] [--help] [-C <path>] [-c name=value]

[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

[-p | --paginate | --no-pager] [--no-replace-objects] [--bare]

[--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

<command> [<args>]

**选项**

add 将文件内容添加到索引

bisect 通过二进制查找引入错误的更改

branch 列出，创建或删除分支

checkout 检查分支或路径到工作树

clone 将存储库克隆到新目录中

commit 将更改记录到存储库

diff 显示提交，提交和工作树等之间的更改

fetch 从另一个存储库下载对象和引用

grep 打印匹配图案的行

init 创建一个空的Git仓库或重新初始化一个现有的

log 显示提交日志

merge 加入两个或更多的开发历史

mv 移动或重命名文件，目录或符号链接

pull 从另一个存储库或本地分支获取并合并

push 更新远程引用以及相关对象

rebase 转发端口本地提交到更新的上游头

reset 将当前HEAD复位到指定状态

rm 从工作树和索引中删除文件

show 显示各种类型的对象

status 显示工作树状态

tag 创建，列出，删除或验证使用GPG签名的标签对象

**例子**

init

git init #初始化

status

git status #获取状态

add

git add file # .或\*代表全部添加  
git rm --cached <added\_file\_to\_undo> # 在commit之前撤销git add操作  
git reset head # 好像比上面git rm --cached更方便

commit

git commit -m "message" #此处注意乱码

remote

git remote add origin git@github.com:JSLite/test.git #添加源

push

git push -u origin master # push同事设置默认跟踪分支

git push origin master

git push -f origin master # 强制推送文件，缩写 -f（全写--force）

clone

git clone git://github.com/JSLite/JSLite.js.git  
git clone git://github.com/JSLite/JSLite.js.git mypro #克隆到自定义文件夹  
git clone [user@]example.com:path/to/repo.git/ #SSH协议还有另一种写法。

git clone支持多种协议，除了HTTP(s)以外，还支持SSH、Git、本地文件协议等，下面是一些例子。git clone <版本库的网址> <本地目录名>

$ git clone http[s]://example.com/path/to/repo.git/

$ git clone ssh://example.com/path/to/repo.git/

$ git clone git://example.com/path/to/repo.git/

$ git clone /opt/git/project.git

$ git clone file:///opt/git/project.git

$ git clone ftp[s]://example.com/path/to/repo.git/

$ git clone rsync://example.com/path/to/repo.git/

配置

首先是配置帐号信息 ssh -T git@github.com 测试。

修改项目中的个人信息

git help config # 获取帮助信息，查看修改个人信息的参数

git config --global user.name "小弟调调" # 修改全局名字

git config --global user.email "wowohoo@qq.com" # 修改全局邮箱

git config --list # 查看配置的信息

**配置自动换行**

自动转换坑太大，提交到git是自动将换行符转换为lf

git config --global core.autocrlf input

常见使用场景

**创建SSH密钥**

这个密钥用来跟 github 通信，在本地终端里生成然后上传到 github

ssh-keygen -t rsa -C 'wowohoo@qq.com' # 生成密钥

ssh-keygen -t rsa -C "wowohoo@qq.com" -f ~/.ssh/ww\_rsa # 指定生成目录文件名字

ssh -T git@github.com # 测试是否成功

**多账号ssh配置**

**1.生成指定名字的密钥**

ssh-keygen -t rsa -C "邮箱地址" -f ~/.ssh/jslite\_rsa  
会生成 jslite\_rsa 和 jslite\_rsa.pub 这两个文件

**2.密钥复制到托管平台上**

vim ~/.ssh/jslite\_rsa.pub  
打开公钥文件 jslite\_rsa.pub ，并把内容复制至代码托管平台上

**3.修改config文件**

vim ~/.ssh/config #修改config文件，如果没有创建 config

Host jslite.github.com

HostName github.com

User git

IdentityFile ~/.ssh/jslite\_rsa

Host work.github.com

HostName github.com

# Port 服务器open-ssh端口（默认：22,默认时一般不写此行）

# PreferredAuthentications 配置登录时用什么权限认证

# publickey|password publickey|keyboard-interactive等

User git

IdentityFile ~/.ssh/work\_rsa

* Host 这里是个别名可以随便命名
* HostName 一般是网站如：git@ss.github.com:username/repo.git 填写 github.com
* User 通常填写git
* IdentityFile 使用的公钥文件地址

**4.测试**

ssh -T git@jslite.github.com # `@`后面跟上定义的Host

ssh -T work.github.com # 通过别名测试

ssh -i ~/公钥文件地址 Host别名 # 如 ssh -i ~/.ssh/work\_rsa work.github.com

**5.使用**

# 原来的写法

git clone git@github.com:<jslite的用户名>/learngit.git

# 现在的写法

git clone git@jslite.github.com:<jslite的用户名>/learngit.git

git clone git@work.github.com:<work的用户名>/learngit.git

**5.注意**

如果你修改了id\_rsa的名字，你需要将ssh key添加到SSH agent中，如：

ssh-add ~/.ssh/jslite\_rsa

ssh-add -l # 查看所有的key

ssh-add -D # 删除所有的key

ssh-add -d ~/.ssh/jslite\_rsa # 删除指定的key

**免密码登录远程服务器**

$ ssh-keygen -t rsa -P '' -f ~/.ssh/aliyunserver.key

$ ssh-copy-id -i ~/.ssh/aliyunserver.key.pub root@192.168.182.112 # 这里需要输入密码一次

编辑 ~/.ssh/config

Host aliyun1

HostName 192.168.182.112

User root

PreferredAuthentications publickey

IdentityFile ~/.ssh/aliyunserver.key

上面配置完了，可以通过命令登录，不需要输入IP地址和密码 ssh aliyun1

**https协议下提交代码免密码**

git clone https://github.com/username/rep.git

通过上面方式克隆可能需要密码，解决办法：进入当前克隆的项目 vi rep/.git/config 编辑 config, 按照下面方式修改，你就可以提交代码不用输入密码了。

[core]

repositoryformatversion = 0

filemode = true

bare = false

logallrefupdates = true

ignorecase = true

precomposeunicode = true

[remote "origin"]

- url = https://github.com/username/rep.git

+ url = https://用户名:密码@github.com/username/rep.git

fetch = +refs/heads/\*:refs/remotes/origin/\*

[branch "master"]

remote = origin

merge = refs/heads/master

**文件推向3个git库**

**1. 增加3个远程库地址**

git remote add origin https://github.com/JSLite/JSLite.git

git remote set-url --add origin https://gitlab.com/wang/JSLite.js.git

git remote set-url --add origin https://oschina.net/wang/JSLite.js.git

**2. 删除其中一个 set-url 地址**

usage: git remote set-url [--push] <name> <newurl> [<oldurl>]

or: git remote set-url --add <name> <newurl>

or: git remote set-url --delete <name> <url>

git remote set-url --delete origin https://oschina.net/wang/JSLite.js.git

**3.推送代码**

git push origin master

git push -f origin master # 强制推送

**4.拉代码**

只能拉取 origin 里的一个url地址，这个fetch-url  
默认为你添加的到 origin的第一个地址

git pull origin master

git pull --all # 获取远程所有内容包括tag

git pull origin next:master # 取回origin主机的next分支，与本地的master分支合并

git pull origin next # 远程分支是与当前分支合并

# 上面一条命令等同于下面两条命令

git fetch origin

git merge origin/next

如果远程主机删除了某个分支，默认情况下，git pull 不会在拉取远程分支的时候，删除对应的本地分支。这是为了防止，由于其他人操作了远程主机，导致git pull不知不觉删除了本地分支。  
但是，你可以改变这个行为，加上参数 -p 就会在本地删除远程已经删除的分支。

$ git pull -p

# 等同于下面的命令

$ git fetch --prune origin

$ git fetch -p

**5.更改pull**

只需要更改config文件里，那三个url的顺序即可，fetch-url会直接对应排行第一的那个utl连接。

**修改远程仓库地址**

git remote remove origin # 删除该远程路径

git remote add origin git@jslite.github.com:JSLite/JSLite.git # 添加远程路径

**撤销远程记录**

git reset --hard HEAD~1 # 撤销一条记录

git push -f origin HEAD:master # 同步到远程仓库

**放弃本地的文件修改**

git reset --hard FETCH\_HEAD # FETCH\_HEAD表示上一次成功git pull之后形成的commit点。然后git pull

git reset --hard FETCH\_HEAD 出现错误

git pull

You are not currently on a branch, so I cannot use any

'branch.<branchname>.merge' in your configuration file.

Please specify which remote branch you want to use on the command

line and try again (e.g. 'git pull <repository> <refspec>').

See git-pull(1) FOR details.

解决方法：

git checkout -b temp # 新建+切换到temp分支

git checkout master

**最简单放弃本地修改内容**

# 如果有的修改以及加入暂存区的话

git reset --hard

# 还原所有修改，不会删除新增的文件

git checkout .

# 下面命令会删除新增的文件

git clean -xdf

通过存储暂存区stash，在删除暂存区的方法放弃本地修改。

git stash && git stash drop

**回滚到某个commit提交**

git revert HEAD~1 # 撤销一条记录 会弹出 commit 编辑

git push # 提交回滚

**回退到某一个版本**

git reset --hard <hash>

# 例如 git reset --hard a3hd73r

# --hard代表丢弃工作区的修改，让工作区与版本代码一模一样，与之对应，

# --soft参数代表保留工作区的修改。

**去掉某个commit**

# 实质是新建了一个与原来完全相反的commit，抵消了原来commit的效果

git revert <commit-hash>

**新建一个空分支**

# 这种方式新建的分支(gh-pages)是没有 commit 记录的

git checkout --orphan gh-pages

# 删除新建的gh-pages分支原本的内容，如果不删除，提交将作为当前分支的第一个commit

git rm -rf .

# 查看一下状态 有可能上面一条命令，没有删除还没有提交的的文件

git state

**合并多个commit**

# 这个命令，将最近4个commit合并为1个，HEAD代表当前版本。

# 将进入VIM界面，你可以修改提交信息。

git rebase -i HEAD~4

# 可以看到其中分为两个部分，上方未注释的部分是填写要执行的指令，

# 而下方注释的部分则是指令的提示说明。指令部分中由前方的命令名称、commit hash 和 commit message 组成

# 当前我们只要知道 pick 和 squash 这两个命令即可。

# --> pick 的意思是要会执行这个 commit

# --> squash 的意思是这个 commit 会被合并到前一个commit

# 我们将 需要保留的 这个 commit 前方的命令改成 squash 或 s，然后输入:wq以保存并退出

# 这是我们会看到 commit message 的编辑界面

# 其中, 非注释部分就是两次的 commit message, 你要做的就是将这两个修改成新的 commit message。

#

# 输入wq保存并推出, 再次输入git log查看 commit 历史信息，你会发现这两个 commit 已经合并了。

# 将修改强制推送到前端

git push -f origin master

**修改远程Commit记录**

git commit --amend

# amend只能修改没有提交到线上的，最后一次commit记录

git rebase -i HEAD~3

# 表示要修改当前版本的倒数第三次状态

# 将要更改的记录行首单词 pick 改为 edit

pick 96dc3f9 doc: Update quick-start.md

pick f1cce8a test(Transition):Add transition test (#47)

pick 6293516 feat(Divider): Add Divider component.

# Rebase eeb03a4..6293516 onto eeb03a4 (3 commands)

#

# Commands:

# p, pick = use commit

# r, reword = use commit, but edit the commit message

# e, edit = use commit, but stop for amending

# s, squash = use commit, but meld into previous commit

# f, fixup = like "squash", but discard this commit's log message

# x, exec = run command (the rest of the line) using shell

# d, drop = remove commit

保存并退出，会弹出下面提示

# You can amend the commit now, with

#

# git commit --amend

#

# Once you are satisfied with your changes, run

#

# git rebase --continue

# 通过这条命令进入编辑页面更改commit，保存退出

git commit --amend

# 保存退出确认修改，继续执行 rebase,

git rebase --continue

# 如果修改多条记录反复执行上面两条命令直到完成所有修改

# 最后，确保别人没有提交进行push，最好不要加 -f 强制推送

git push -f origin master

**添加忽略文件**

echo node\_modules/ >> .gitignore

**利用commit关闭一个issue**

这个功能在Github上可以玩儿，Gitlab上特别老的版本不能玩儿哦，那么如何跟随着commit关闭一个issue呢? 在confirm merge的时候可以使用一下命令来关闭相关issue:

fixes #xxx、 fixed #xxx、 fix #xxx、 closes #xxx、 close #xxx、 closed #xxx、

**同步fork的上游仓库**

[Github教程同步fork教程](https://help.github.com/articles/syncing-a-fork/)，[在Github上同步一个分支(fork)](http://www.miss77.net/549.html)

**设置添加多个远程仓库地址。**

在同步之前，需要创建一个远程点指向上游仓库(repo).如果你已经派生了一个原始仓库，可以按照如下方法做。

$ git remote -v

# List the current remotes （列出当前远程仓库）

# origin https://github.com/user/repo.git (fetch)

# origin https://github.com/user/repo.git (push)

$ git remote add upstream https://github.com/otheruser/repo.git

# Set a new remote (设置一个新的远程仓库)

$ git remote -v

# Verify new remote (验证新的原唱仓库)

# origin https://github.com/user/repo.git (fetch)

# origin https://github.com/user/repo.git (push)

# upstream https://github.com/otheruser/repo.git (fetch)

# upstream https://github.com/otheruser/repo.git (push)

**同步更新仓库内容**

同步上游仓库到你的仓库需要执行两步：首先你需要从远程拉去，之后你需要合并你希望的分支到你的本地副本分支。从上游的存储库中提取分支以及各自的提交内容。 master 将被存储在本地分支机构 upstream/master

git fetch upstream

# remote: Counting objects: 75, done.

# remote: Compressing objects: 100% (53/53), done.

# remote: Total 62 (delta 27), reused 44 (delta 9)

# Unpacking objects: 100% (62/62), done.

# From https://github.com/ORIGINAL\_OWNER/ORIGINAL\_REPOSITORY

# \* [new branch] master -> upstream/master

检查你的 fork's 本地 master 分支

git checkout master

# Switched to branch 'master'

合并来自 upstream/master 的更改到本地 master 分支上。 这使你的前 fork's master 分支与上游资源库同步，而不会丢失你本地修改。

git merge upstream/master

# Updating a422352..5fdff0f

# Fast-forward

# README | 9 -------

# README.md | 7 ++++++

# 2 files changed, 7 insertions(+), 9 deletions(-)

# delete mode 100644 README

# create mode 100644 README.md

**批量修改历史commit中的名字和邮箱**

**1.克隆仓库**

注意参数，这个不是普通的clone，clone下来的仓库并不能参与开发

git clone --bare https://github.com/user/repo.git

cd repo.git

**2.命令行中运行代码**

OLD\_EMAIL原来的邮箱  
CORRECT\_NAME更正的名字  
CORRECT\_EMAIL更正的邮箱

将下面代码复制放到命令行中执行

git filter-branch -f --env-filter '

OLD\_EMAIL="wowohoo@qq.com"

CORRECT\_NAME="小弟调调"

CORRECT\_EMAIL="更正的邮箱@qq.com"

if [ "$GIT\_COMMITTER\_EMAIL" = "$OLD\_EMAIL" ]

then

export GIT\_COMMITTER\_NAME="$CORRECT\_NAME"

export GIT\_COMMITTER\_EMAIL="$CORRECT\_EMAIL"

fi

if [ "$GIT\_AUTHOR\_EMAIL" = "$OLD\_EMAIL" ]

then

export GIT\_AUTHOR\_NAME="$CORRECT\_NAME"

export GIT\_AUTHOR\_EMAIL="$CORRECT\_EMAIL"

fi

' --tag-name-filter cat -- --branches --tags

执行过程

Rewrite 160d4df2689ff6df3820563bfd13b5f1fb9ba832 (479/508) (16 seconds passed, remaining 0 predicted)

Ref 'refs/heads/dev' was rewritten

Ref 'refs/heads/master' was rewritten

**3.同步到远程仓库**

同步到push远程git仓库

git push --force --tags origin 'refs/heads/\*'

我还遇到了如下面错误，lab默认给master分支加了保护，不允许强制覆盖。Project(项目)->Setting->Repository 菜单下面的Protected branches把master的保护去掉就可以了。修改完之后，建议把master的保护再加回来，毕竟强推不是件好事。

remote: GitLab: You are not allowed to force push code to a protected branch on this project.

当上面的push 不上去的时候，先 git pull 确保最新代码

git pull --allow-unrelated-histories

# 或者指定分枝

git pull origin master --allow-unrelated-histories

**查看某个文件历史**

git log --pretty=oneline 文件名 # 列出文件的所有改动历史

git show c178bf49 # 某次的改动的修改记录

git log -p c178bf49 # 某次的改动的修改记录

git blame 文件名 # 显示文件的每一行是在那个版本最后修改。

git whatchanged 文件名 # 显示某个文件的每个版本提交信息：提交日期，提交人员，版本号，提交备注（没有修改细节）

**打造自己的git命令**

git config --global alias.st status

git config --global alias.br branch

git config --global alias.co checkout

git config --global alias.ci commit

配置好后再输入git命令的时候就不用再输入一大段了，例如我们要查看状态，只需：

git st

**中文乱码的解决方案**

git config --global core.quotepath false

新建仓库

**init**

git init #初始化

**status**

git status #获取状态

**add**

git add file # .或\*代表全部添加  
git rm --cached <added\_file\_to\_undo> # 在commit之前撤销git add操作  
git reset head # 好像比上面git rm --cached更方便

**commit**

git commit -m "message" #此处注意乱码

**remote**

git remote add origin git@github.com:JSLite/test.git #添加源

**push**

git push -u origin master # push同事设置默认跟踪分支

git push origin master

git push -f origin master # 强制推送文件，缩写 -f（全写--force）

clone

git clone git://github.com/JSLite/JSLite.js.git  
git clone git://github.com/JSLite/JSLite.js.git mypro #克隆到自定义文件夹  
git clone [user@]example.com:path/to/repo.git/ #SSH协议还有另一种写法。

git clone支持多种协议，除了HTTP(s)以外，还支持SSH、Git、本地文件协议等，下面是一些例子。git clone <版本库的网址> <本地目录名>

$ git clone http[s]://example.com/path/to/repo.git/

$ git clone ssh://example.com/path/to/repo.git/

$ git clone git://example.com/path/to/repo.git/

$ git clone /opt/git/project.git

$ git clone file:///opt/git/project.git

$ git clone ftp[s]://example.com/path/to/repo.git/

$ git clone rsync://example.com/path/to/repo.git/

本地

**help**

git help config # 获取帮助信息

**add**

git add \* # 跟踪新文件

git add -u [path] # 添加[指定路径下]已跟踪文件

**rm**

rm \*&git rm \* # 移除文件

git rm -f \* # 移除文件

git rm --cached \* # 取消跟踪

git mv file\_from file\_to # 重命名跟踪文件

git log # 查看提交记录

**commit**

git commit #提交更新

git commit -m 'message' #提交说明

git commit -a #跳过使用暂存区域，把所有已经跟踪过的文件暂存起来一并提交

git commit --amend #修改最后一次提交

git commit log #查看所有提交，包括没有push的commit

git commit -m "#133" #关联issue 任意位置带上# 符号加上issue号码

git commit -m "fix #133" commit关闭issue

git commit -m '概要描述'$'\n\n''1.详细描述'$'\n''2.详细描述' #提交简要描述和详细描述

**reset**

git reset HEAD \* # 取消已经暂存的文件

git reset --mixed HEAD \* # 同上

git reset --soft HEAD \* # 重置到指定状态，不会修改索引区和工作树

git reset --hard HEAD \* # 重置到指定状态，会修改索引区和工作树

git reset -- files \* # 重置index区文件

**revert**

git revert HEAD # 撤销前一次操作

git revert HEAD~ # 撤销前前一次操作

git revert commit # 撤销指定操作

**checkout**

git checkout -- file # 取消对文件的修改（从暂存区——覆盖worktree file）

git checkout branch|tag|commit -- file\_name # 从仓库取出file覆盖当前分支

git checkout HEAD~1 [文件] # 将会更新 working directory 去匹配某次 commit

git checkout -- . # 从暂存区取出文件覆盖工作区

git checkout -b gh-pages 0c304c9 # 这个表示 从当前分支 commit 哈希值为 0c304c9 的节点，分一个新的分支gh-pages出来，并切换到 gh-pages

**diff**

git diff file # 查看指定文件的差异

git diff --stat # 查看简单的diff结果

git diff # 比较Worktree和Index之间的差异

git diff --cached # 比较Index和HEAD之间的差异

git diff HEAD # 比较Worktree和HEAD之间的差异

git diff branch # 比较Worktree和branch之间的差异

git diff branch1 branch2 # 比较两次分支之间的差异

git diff commit commit # 比较两次提交之间的差异

git diff master..test # 上面这条命令只显示两个分支间的差异

git diff master...test # 你想找出‘master’,‘test’的共有 父分支和'test'分支之间的差异，你用3个‘.'来取代前面的两个'.'

**stash**

git stash # 将工作区现场（已跟踪文件）储藏起来，等以后恢复后继续工作。

git stash list # 查看保存的工作现场

git stash apply # 恢复工作现场

git stash drop # 删除stash内容

git stash pop # 恢复的同时直接删除stash内容

git stash apply stash@{0} # 恢复指定的工作现场，当你保存了不只一份工作现场时。

**merge**

git merge --squash test # 合并压缩，将test上的commit压缩为一条

**cherry-pick**

git cherry-pick commit # 拣选合并，将commit合并到当前分支

git cherry-pick -n commit # 拣选多个提交，合并完后可以继续拣选下一个提交

**rebase**

git rebase master # 将master分之上超前的提交，变基到当前分支

git rebase --onto master 169a6 # 限制回滚范围，rebase当前分支从169a6以后的提交

git rebase --interactive # 交互模式，修改commit

git rebase --continue # 处理完冲突继续合并

git rebase --skip # 跳过

git rebase --abort # 取消合并

分支branch

**删除**

git push origin :branchName # 删除远程分支

git push origin --delete new # 删除远程分支new

git branch -d branchName # 删除本地分支，强制删除用-D

git branch -d test # 删除本地test分支

git branch -D test # 强制删除本地test分支

git remote prune origin # 远程删除了，本地还能看到远程存在，这条命令删除远程不存在的分支

**提交**

git push -u origin branchName # 提交分支到远程origin主机中

**拉取**

git fetch -p #拉取远程分支时，自动清理 远程分支已删除，本地还存在的对应同名分支。

**分支合并**

git merge branchName # 合并分支 - 将分支branchName和当前所在分支合并

git merge origin/master # 在本地分支上合并远程分支。

git rebase origin/master # 在本地分支上合并远程分支。

git merge test # 将test分支合并到当前分支

**重命名**

git branch -m old new #重命名分支

**查看**

git branch # 列出本地分支

git branch -r # 列出远端分支

git branch -a # 列出所有分支

git branch -v # 查看各个分支最后一个提交对象的信息

git branch --merge # 查看已经合并到当前分支的分支

git branch --no-merge # 查看为合并到当前分支的分支

git remote show origin # 可以查看remote地址，远程分支

**新建**

git branch test # 新建test分支

git branch newBrach 3defc69 # 指定哈希3defc69，新建分支名字为newBrach

git checkout -b newBrach origin/master # 取回远程主机的更新以后，在它的基础上创建一个新的分支

git checkout -b newBrach 3defc69 # 以哈希值3defc69，新建 newBrach 分支，并切换到该分支

**连接**

git branch --set-upstream dev origin/dev # 将本地dev分支与远程dev分支之间建立链接

git branch --set-upstream master origin/next # 手动建立追踪关系

**分支切换**

git checkout test # 切换到test分支

git checkout -b test # 新建+切换到test分支

git checkout -b test dev # 基于dev新建test分支，并切换

远端

git fetch <远程主机名> <分支名> # fetch取回所有分支（branch）的更新

git fetch origin remotebranch[:localbranch] # 从远端拉去分支[到本地指定分支]

git merge origin/branch # 合并远端上指定分支

git pull origin remotebranch:localbranch # 拉去远端分支到本地分支

git push origin branch # 将当前分支，推送到远端上指定分支

git push origin localbranch:remotebranch # 推送本地指定分支，到远端上指定分支

git push origin :remotebranch # 删除远端指定分支

git checkout -b [--track] test origin/dev # 基于远端dev分支，新建本地test分支[同时设置跟踪]

submodule

克隆项目同时克隆submodule

git clone https://github.com/jaywcjlove/handbook.git --depth=1 --recurse-submodules

克隆项目，之后再手动克隆 submodule 子项目

git submodule add --force '仓库地址' '路径'

# 其中，仓库地址是指子模块仓库地址，路径指将子模块放置在当前工程下的路径。

# 注意：路径不能以 / 结尾（会造成修改不生效）、不能是现有工程已有的目录（不能順利 Clone）

git submodule init # 初始化submodule

git submodule update # 更新submodule(必须在根目录执行命令)

git submodule update --init --recursive # 下载的工程带有submodule

当使用git clone下来的工程中带有submodule时，初始的时候，submodule的内容并不会自动下载下来的，此时，只需执行如下命令：

git submodule foreach git pull # submodule 里有其他的 submodule 一次更新

git submodule foreach git pull origin master # submodule更新

git submodule foreach --recursive git submodule init

git submodule foreach --recursive git submodule update

删除文件

git rm -rf node\_modules/

remote

git是一个分布式代码管理工具，所以可以支持多个仓库，在git里，服务器上的仓库在本地称之为remote。个人开发时，多源用的可能不多，但多源其实非常有用。

git remote add origin1 git@github.com:yanhaijing/data.js.git

git remote # 显示全部源

git remote -v # 显示全部源+详细信息

git remote rename origin1 origin2 # 重命名

git remote rm origin # 删除

git remote show origin # 查看指定源的全部信息

标签tag

当开发到一定阶段时，给程序打标签是非常棒的功能。

git tag -a v0.1 -m 'my version 1.4' # 新建带注释标签

git push origin --tags # 一次性推送所有分支

git push origin v1.5 # 推送单个tag到orgin源上

git tag -v v1.4.2.1 # 验证标签，验证已经签署的标签

git show v1.5 # 看到对应的 GPG 签

git tag # 列出现有标签

git tag v0gi.1 # 新建标签

git checkout tagname # 切换到标签

git tag -d v0.1 # 删除标签

git push origin :refs/tags/v0.1 # 删除远程标签

git pull --all # 获取远程所有内容包括tag

git --git-dir='<绝对地址>/.git' describe --tags HEAD # 查看本地版本信息

日志log

git config format.pretty oneline #显示历史记录时，每个提交的信息只显示一行

git config color.ui true #彩色的 git 输出

git log #查看最近的提交日志

git log --pretty=oneline #单行显示提交日志

git log --graph --pretty=oneline --abbrev-commit

git log -num #显示第几条log（倒数）

git reflog #查看所有分支的所有操作记录

git log --since=1.day #一天内的提交；你可以给出各种时间格式，比如说具体的某一天（“2008-01-15”），或者是多久以前（“2 years 1 day 3 minutes ago”）。

git log --pretty="%h - %s" --author=自己的名字 #查看自己的日志

git log -p -2 #展开两次更新显示每次提交的内容差异

git log --stat #要快速浏览其他协作者提交的更新都作了哪些改动

git log --pretty=format:"%h - %an, %ar : %s"#定制要显示的记录格式

git log --pretty=format:'%h : %s' --date-order --graph # 拓扑顺序展示

git log --pretty=format:'%h : %s - %ad' --date=short #日期YYYY-MM-DD显示

git log <last tag> HEAD --pretty=format:%s # 只显示commit

git config --global format.pretty '%h : %s - %ad' --date=short #日期YYYY-MM-DD显示 写入全局配置

| **选项** | **说明** | **选项** | **说明** |
| --- | --- | --- | --- |
| %H | 提交对象（commit）的完整哈希字串 | %ad | 作者修订日期（可以用 -date= 选项定制格式） |
| %h | 提交对象的简短哈希字串 | %ar | 作者修订日期，按多久以前的方式显示 |
| %T | 树对象（tree）的完整哈希字串 | %cn | 提交者(committer)的名字 |
| %t | 树对象的简短哈希字串 | %ce | 提交者的电子邮件地址 |
| %P | 父对象（parent）的完整哈希字串 | %cd | 提交日期 |
| %p | 父对象的简短哈希字串 | %cr | 提交日期，按多久以前的方式显示 |
| %an | 作者（author）的名字 | %s | 提交说明 |
| %ae | 作者的电子邮件地址 | - | - |

[Pretty Formats](https://git-scm.com/docs/git-log#_pretty_formats)

重写历史

git commit --amend # 改变最近一次提交

git rebase -i HEAD~3 # 修改最近三次的提交说明，或者其中任意一次

git commit --amend # 保存好了，这些指示很明确地告诉了你该干什么

git rebase --continue # 修改提交说明，退出编辑器。

pick f7f3f6d changed my name a bit

pick 310154e updated README formatting and added blame

pick a5f4a0d added cat-file

改成

pick 310154e updated README formatting and added blame

pick f7f3f6d changed my name a bit

**删除仓库**

cd ..

rm -rf repo.git

[Github官方教程](https://help.github.com/articles/changing-author-info/)

其它

git help \* # 获取命令的帮助信息

git status # 获取当前的状态，非常有用，因为git会提示接下来的能做的操作

报错问题解决

**1. git fatal: protocol error: bad line length character: No s**

解决办法：更换remote地址为 http/https 的

**2. The requested URL returned error: 403 Forbidden while accessing**

解决github push错误的办法：

#vim 编辑器打开 当前项目中的config文件

vim .git/config

#修改

[remote "origin"]

url = https://github.com/jaywcjlove/example.git

#为下面代码

[remote "origin"]

url = https://jaywcjlove@github.com/jaywcjlove/example.git

**3. git status 显示中文问题**

在查看状态的时候 git status 如果是中文就显示下面的情况

\344\272\247\345\223\201\351\234\200\346\261\202

解决这个问题方法是：

git config --global core.quotepath false

## cut

连接文件并打印到标准输出设备上

补充说明

**cut 命令** 用来显示行中的指定部分，删除文件中指定字段。cut 经常用来显示文件的内容，类似于 type 命令。

说明：该命令有两项功能，其一是用来显示文件的内容，它依次读取由参数 file 所指 明的文件，将它们的内容输出到标准输出上；其二是连接两个或多个文件，如cut fl f2 > f3将把文件 fl 和 f2 的内容合并起来，然后通过输出重定向符“>”的作用，将它们放入文件 f3 中。

当文件较大时，文本在屏幕上迅速闪过（滚屏），用户往往看不清所显示的内容。因此，一般用 more 等命令分屏显示。为了控制滚屏，可以按 Ctrl+S 键，停止滚屏；按 Ctrl+Q 键可以恢复滚屏。按 Ctrl+C（中断）键可以终止该命令的执行，并且返回 Shell 提示符状态。

**语法**

cut（选项）（参数）

**选项**

-b：仅显示行中指定直接范围的内容；

-c：仅显示行中指定范围的字符；

-d：指定字段的分隔符，默认的字段分隔符为“TAB”；

-f：显示指定字段的内容；

-n：与“-b”选项连用，不分割多字节字符；

--complement：补足被选择的字节、字符或字段；

--out-delimiter= 字段分隔符：指定输出内容是的字段分割符；

--help：显示指令的帮助信息；

--version：显示指令的版本信息。

**参数**

文件：指定要进行内容过滤的文件。

**实例**

例如有一个学生报表信息，包含 No、Name、Mark、Percent：

[root@localhost text]# cat test.txt

No Name Mark Percent

01 tom 69 91

02 jack 71 87

03 alex 68 98

使用 **-f** 选项提取指定字段（这里的 f 参数可以简单记忆为 --fields的缩写）：

[root@localhost text]# cut -f 1 test.txt

No

01

02

03

[root@localhost text]# cut -f2,3 test.txt

Name Mark

tom 69

jack 71

alex 68

**--complement** 选项提取指定字段之外的列（打印除了第二列之外的列）：

[root@localhost text]# cut -f2 --complement test.txt

No Mark Percent

01 69 91

02 71 87

03 68 98

使用 **-d** 选项指定字段分隔符：

[root@localhost text]# cat test2.txt

No;Name;Mark;Percent

01;tom;69;91

02;jack;71;87

03;alex;68;98

[root@localhost text]# cut -f2 -d";" test2.txt

Name

tom

jack

alex

**指定字段的字符或者字节范围**

cut 命令可以将一串字符作为列来显示，字符字段的记法：

* **N-** ：从第 N 个字节、字符、字段到结尾；
* **N-M** ：从第 N 个字节、字符、字段到第 M 个（包括 M 在内）字节、字符、字段；
* **-M** ：从第 1 个字节、字符、字段到第 M 个（包括 M 在内）字节、字符、字段。

上面是记法，结合下面选项将摸个范围的字节、字符指定为字段：

* **-b** 表示字节；
* **-c** 表示字符；
* **-f** 表示定义字段。

**示例**

[root@localhost text]# cat test.txt

abcdefghijklmnopqrstuvwxyz

abcdefghijklmnopqrstuvwxyz

abcdefghijklmnopqrstuvwxyz

abcdefghijklmnopqrstuvwxyz

abcdefghijklmnopqrstuvwxyz

打印第 1 个到第 3 个字符：

[root@localhost text]# cut -c1-3 test.txt

abc

abc

abc

abc

abc

打印前 2 个字符：

[root@localhost text]# cut -c-2 test.txt

ab

ab

ab

ab

ab

打印从第 5 个字符开始到结尾：

[root@localhost text]# cut -c5- test.txt

efghijklmnopqrstuvwxyz

efghijklmnopqrstuvwxyz

efghijklmnopqrstuvwxyz

efghijklmnopqrstuvwxyz

efghijklmnopqrstuvwxyz

## ln

用来为文件创件连接

补充说明

**ln命令** 用来为文件创件连接，连接类型分为硬连接和符号连接两种，默认的连接类型是硬连接。如果要创建符号连接必须使用"-s"选项。

注意：符号链接文件不是一个独立的文件，它的许多属性依赖于源文件，所以给符号链接文件设置存取权限是没有意义的。

**语法**

ln [选项]... [-T] 目标 链接名 (第一种格式)

　或：ln [选项]... 目标 (第二种格式)

　或：ln [选项]... 目标... 目录 (第三种格式)

　或：ln [选项]... -t 目录 目标... (第四种格式)

**选项**

--backup[=CONTROL] 为每个已存在的目标文件创建备份文件

-b 类似--backup，但不接受任何参数

-d, -F, --directory 创建指向目录的硬链接(只适用于超级用户)

-f, --force 强行删除任何已存在的目标文件

-i, --interactive 覆盖既有文件之前先询问用户；

-L, --logical 取消引用作为符号链接的目标

-n, --no-dereference 把符号连接的目的目录视为一般文件；

-P, --physical 直接将硬链接到符号链接

-r, --relative 创建相对于链接位置的符号链接

-s, --symbolic 对源文件建立符号连接，而非硬连接；

-S, --suffix=SUFFIX 用"-b"参数备份目标文件后，备份文件的字尾会被加上一个备份字符串，预设的备份字符串是符号“~”，用户可通过“-S”参数来改变它；

-t, --target-directory=DIRECTORY 指定要在其中创建链接的DIRECTORY

-T, --no-target-directory 将“LINK\_NAME”视为常规文件

-v, --verbose 打印每个链接文件的名称

--help 显示此帮助信息并退出

--version 显示版本信息并退出

**参数**

* 源文件：指定连接的源文件。如果使用-s选项创建符号连接，则“源文件”可以是文件或者目录。创建硬连接时，则“源文件”参数只能是文件；
* 目标文件：指定源文件的目标连接文件。

none, off # 不进行备份(即使使用了--backup 选项)

numbered, t # 备份文件加上数字进行排序

existing, nil # 若有数字的备份文件已经存在则使用数字，否则使用普通方式备份

simple, never # 永远使用普通方式备份

**实例**

将目录/usr/mengqc/mub1下的文件m2.c链接到目录/usr/liu下的文件a2.c

cd /usr/mengqc

ln /mub1/m2.c /usr/liu/a2.c

在执行ln命令之前，目录/usr/liu中不存在a2.c文件。执行ln之后，在/usr/liu目录中才有a2.c这一项，表明m2.c和a2.c链接起来（注意，二者在物理上是同一文件），利用ls -l命令可以看到链接数的变化。

在目录/usr/liu下建立一个符号链接文件abc，使它指向目录/usr/mengqc/mub1

ln -s /usr/mengqc/mub1 /usr/liu/abc

执行该命令后，/usr/mengqc/mub1代表的路径将存放在名为/usr/liu/abc的文件中。

扩展知识

Linux具有为一个文件起多个名字的功能，称为链接。被链接的文件可以存放在相同的目录下，但是必须有不同的文件名，而不用在硬盘上为同样的数据重复备份。另外，被链接的文件也可以有相同的文件名，但是存放在不同的目录下，这样只要对一个目录下的该文件进行修改，就可以完成对所有目录下同名链接文件的修改。对于某个文件的各链接文件，我们可以给它们指定不同的存取权限，以控制对信息的共享和增强安全性。

文件链接有两种形式，即硬链接和符号链接。

**硬链接**

建立硬链接时，在另外的目录或本目录中增加目标文件的一个目录项，这样，一个文件就登记在多个目录中。如图所示的m2.c文件就在目录mub1和liu中都建立了目录项。

创建硬链接后，己经存在的文件的I节点号（Inode）会被多个目录文件项使用。一个文件的硬链接数可以在目录的长列表格式的第二列中看到，无额外链接的文件的链接数为l。

在默认情况下，ln命令创建硬链接。ln命令会增加链接数，rm命令会减少链接数。一个文件除非链接数为0，否则不会从文件系统中被物理地删除。

对硬链接有如下限制：

* 不能对目录文件做硬链接。
* 不能在不同的文件系统之间做硬链接。就是说，链接文件和被链接文件必须位于同一个文件系统中。

**符号链接**

符号链接也称为软链接，是将一个路径名链接到一个文件。这些文件是一种特别类型的文件。事实上，它只是一个文本文件（如图中的abc文件），其中包含它提供链接的另一个文件的路径名，如图中虚线箭头所示。另一个文件是实际包含所有数据的文件。所有读、写文件内容的命令被用于符号链接时，将沿着链接方向前进来访问实际的文件。

!符号连接

与硬链接不同的是，符号链接确实是一个新文件，当然它具有不同的I节点号；而硬链接并没有建立新文件。

符号链接没有硬链接的限制，可以对目录文件做符号链接，也可以在不同文件系统之间做符号链接。

用ln -s命令建立符号链接时，源文件最好用绝对路径名。这样可以在任何工作目录下进行符号链接。而当源文件用相对路径时，如果当前的工作路径与要创建的符号链接文件所在路径不同，就不能进行链接。

符号链接保持了链接与源文件或目录之间的区别：

* 删除源文件或目录，只删除了数据，不会删除链接。一旦以同样文件名创建了源文件，链接将继续指向该文件的新数据。
* 在目录长列表中，符号链接作为一种特殊的文件类型显示出来，其第一个字母是l。
* 符号链接的大小是其链接文件的路径名中的字节数。
* 当用ln -s命令列出文件时，可以看到符号链接名后有一个箭头指向源文件或目录，例如lrwxrwxrwx … 14 jun 20 10:20 /etc/motd->/original\_file其中，表示“文件大小”的数字“14”恰好说明源文件名original\_file由14个字符构成。

## mv

用来对文件或目录重新命名

补充说明

**mv命令** 用来对文件或目录重新命名，或者将文件从一个目录移到另一个目录中。source表示源文件或目录，target表示目标文件或目录。如果将一个文件移到一个已经存在的目标文件中，则目标文件的内容将被覆盖。

mv命令可以用来将源文件移至一个目标文件中，或将一组文件移至一个目标目录中。源文件被移至目标文件有两种不同的结果：

1. 如果目标文件是到某一目录文件的路径，源文件会被移到此目录下，且文件名不变。
2. 如果目标文件不是目录文件，则源文件名（只能有一个）会变为此目标文件名，并覆盖己存在的同名文件。如果源文件和目标文件在同一个目录下，mv的作用就是改文件名。当目标文件是目录文件时，源文件或目录参数可以有多个，则所有的源文件都会被移至目标文件中。所有移到该目录下的文件都将保留以前的文件名。

注意事项：mv与cp的结果不同，mv好像文件“搬家”，文件个数并未增加。而cp对文件进行复制，文件个数增加了。

**语法**

mv(选项)(参数)

**选项**

--backup=<备份模式>：若需覆盖文件，则覆盖前先行备份；

-b：当文件存在时，覆盖前，为其创建一个备份；

-f：若目标文件或目录与现有的文件或目录重复，则直接覆盖现有的文件或目录；

-i：交互式操作，覆盖前先行询问用户，如果源文件与目标文件或目标目录中的文件同名，则询问用户是否覆盖目标文件。用户输入”y”，表示将覆盖目标文件；输入”n”，表示取消对源文件的移动。这样可以避免误将文件覆盖。

--strip-trailing-slashes：删除源文件中的斜杠“/”；

-S<后缀>：为备份文件指定后缀，而不使用默认的后缀；

--target-directory=<目录>：指定源文件要移动到目标目录；

-u：当源文件比目标文件新或者目标文件不存在时，才执行移动操作。

**参数**

* 源文件：源文件列表。
* 目标文件：如果“目标文件”是文件名则在移动文件的同时，将其改名为“目标文件”；如果“目标文件”是目录名则将源文件移动到“目标文件”下。

**实例**

将目录/usr/men中的所有文件移到当前目录（用.表示）中：

mv /usr/men/\* .

移动文件

mv file\_1.txt /home/office/

移动多个文件

mv file\_2.txt file\_3.txt file\_4.txt /home/office/

mv \*.txt /home/office/

移动目录

mv directory\_1/ /home/office/

重命名文件或目录

mv file\_1.txt file\_2.txt # 将文件file\_1.txt改名为file\_2.txt

重命名目录

mv directory\_1/ directory\_2/

打印移动信息

mv -v \*.txt /home/office

提示是否覆盖文件

mv -i file\_1.txt /home/office

源文件比目标文件新时才执行更新

mv -uv \*.txt /home/office

不要覆盖任何已存在的文件

mv -vn \*.txt /home/office

复制时创建备份

mv -bv \*.txt /home/office

无条件覆盖已经存在的文件

mv -f \*.txt /home/office

## cp

将源文件或目录复制到目标文件或目录中

补充说明

**cp命令** 用来将一个或多个源文件或者目录复制到指定的目的文件或目录。它可以将单个源文件复制成一个指定文件名的具体的文件或一个已经存在的目录下。cp命令还支持同时复制多个文件，当一次复制多个文件时，目标文件参数必须是一个已经存在的目录，否则将出现错误。

**语法**

cp(选项)(参数)

**选项**

-a：此参数的效果和同时指定"-dpR"参数相同；

-d：当复制符号连接时，把目标文件或目录也建立为符号连接，并指向与源文件或目录连接的原始文件或目录；

-f：强行复制文件或目录，不论目标文件或目录是否已存在；

-i：覆盖既有文件之前先询问用户；

-l：对源文件建立硬连接，而非复制文件；

-p：保留源文件或目录的属性；

-R/r：递归处理，将指定目录下的所有文件与子目录一并处理；

-s：对源文件建立符号连接，而非复制文件；

-u：使用这项参数后只会在源文件的更改时间较目标文件更新时或是名称相互对应的目标文件并不存在时，才复制文件；

-S：在备份文件时，用指定的后缀“SUFFIX”代替文件的默认后缀；

-b：覆盖已存在的文件目标前将目标文件备份；

-v：详细显示命令执行的操作。

**参数**

* 源文件：制定源文件列表。默认情况下，cp命令不能复制目录，如果要复制目录，则必须使用-R选项；
* 目标文件：指定目标文件。当“源文件”为多个文件时，要求“目标文件”为指定的目录。

**实例**

下面的第一行中是 cp 命令和具体的参数（-r 是“递归”， -u 是“更新”，-v 是“详细”）。接下来的三行显示被复制文件的信息，最后一行显示命令行提示符。这样，只拷贝新的文件到我的存储设备上，我就使用 cp 的“更新”和“详细”选项。

通常来说，参数 -r 也可用更详细的风格 --recursive。但是以简短的方式，也可以这么连用 -ruv。

cp -r -u -v /usr/men/tmp ~/men/tmp

版本备份 --backup=numbered 参数意思为“我要做个备份，而且是带编号的连续备份”。所以一个备份就是 1 号，第二个就是 2 号，等等。

$ cp --force --backup=numbered test1.py test1.py

$ ls

test1.py test1.py.~1~ test1.py.~2~

如果把一个文件复制到一个目标文件中，而目标文件已经存在，那么，该目标文件的内容将被破坏。此命令中所有参数既可以是绝对路径名，也可以是相对路径名。通常会用到点.或点点..的形式。例如，下面的命令将指定文件复制到当前目录下：

cp ../mary/homework/assign .

所有目标文件指定的目录必须是己经存在的，cp命令不能创建目录。如果没有文件复制的权限，则系统会显示出错信息。

将文件file复制到目录/usr/men/tmp下，并改名为file1

cp file /usr/men/tmp/file1

将目录/usr/men下的所有文件及其子目录复制到目录/usr/zh中

cp -r /usr/men /usr/zh

交互式地将目录/usr/men中的以m打头的所有.c文件复制到目录/usr/zh中

cp -i /usr/men m\*.c /usr/zh

我们在Linux下使用cp命令复制文件时候，有时候会需要覆盖一些同名文件，覆盖文件的时候都会有提示：需要不停的按Y来确定执行覆盖。文件数量不多还好，但是要是几百个估计按Y都要吐血了，于是折腾来半天总结了一个方法：

cp aaa/\* /bbb

# 复制目录aaa下所有到/bbb目录下，这时如果/bbb目录下有和aaa同名的文件，需要按Y来确认并且会略过aaa目录下的子目录。

cp -r aaa/\* /bbb

# 这次依然需要按Y来确认操作，但是没有忽略子目录。

cp -r -a aaa/\* /bbb

# 依然需要按Y来确认操作，并且把aaa目录以及子目录和文件属性也传递到了/bbb。

\cp -r -a aaa/\* /bbb

# 成功，没有提示按Y、传递了目录属性、没有略过目录。

递归强制复制目录到指定目录中覆盖已存在文件

cp -rfb ./\* ../backup

# 将当前目录下所有文件，复制到当前目录的兄弟目录 backup 文件夹中

拷贝目录下的隐藏文件如 .babelrc

cp -r aaa/.\* ./bbb

# 将 aaa 目录下的，所有`.`开头的文件，复制到 bbb 目录中。

cp -a aaa ./bbb/

# 记住后面目录最好的'/' 带上 `-a` 参数

## touch

创建新的空文件

## 补充说明

**touch命令** 有两个功能：一是用于把已存在文件的时间标签更新为系统当前的时间（默认方式），它们的数据将原封不动地保留下来；二是用来创建新的空文件。

### 语法

touch(选项)(参数)

### 选项

-a：或--time=atime或--time=access或--time=use 只更改存取时间；

-c：或--no-create 不建立任何文件；

-d：<时间日期> 使用指定的日期时间，而非现在的时间；

-f：此参数将忽略不予处理，仅负责解决BSD版本touch指令的兼容性问题；

-m：或--time=mtime或--time=modify 只更该变动时间；

-r：<参考文件或目录> 把指定文件或目录的日期时间，统统设成和参考文件或目录的日期时间相同；

-t：<日期时间> 使用指定的日期时间，而非现在的时间；

--help：在线帮助；

--version：显示版本信息。

### 参数

文件：指定要设置时间属性的文件列表。

### 实例

touch ex2

在当前目录下建立一个空文件ex2，然后，利用ls -l命令可以发现文件ex2的大小为0，表示它是空文件。

## umask

用来设置限制新建文件权限的掩码

## 补充说明

**umask命令** 用来设置限制新建文件权限的掩码。当新文件被创建时，其最初的权限由文件创建掩码决定。用户每次注册进入系统时，umask命令都被执行， 并自动设置掩码mode来限制新文件的权限。用户可以通过再次执行umask命令来改变默认值，新的权限将会把旧的覆盖掉。

### 语法

umask(选项)(参数)

### 选项

-p：输出的权限掩码可直接作为指令来执行；

-S：以符号方式输出权限掩码。

### 参数

权限掩码：指定权限掩码。

### 实例

利用umask命令可以指定哪些权限将在新文件的默认权限中被删除。例如，可以使用下面的命令创建掩码，使得组用户的写权限，其他用户的读、写和执行权限都被取消：

umask u=, g=w, o=rwx

执行该命令以后，对于下面创建的新文件，其文件主的权限未做任何改变，而组用户没有写权限，其他用户的所有权限都被取消。

应注意：操作符“=”在umask命令和chmod命令中的作用恰恰相反。在chmod命令中，利用它来设置指定的权限，而其余权限则被删除；但是在umask命令中，它将在原有权限的基础上删除指定的权限。

不能直接利用umask命令创建一个可执行的文件，用户只能在其后利用chmod命令使它具有执行权限。假设执行了命令umask u=, g=w, o=rwx，虽然在命令行中，没有删去文件主和组用户的执行权限，但默认的文件权限还是640（即 rw-r-----），而不是750(rwxr-x---）。但是，如果创建的是目录或者通过编译程序创建的一个可执行文件，将不受此限制。在这种情况 下，会设置文件的执行权限。

也可以使用八进制数值来设置mode。由于在umask中所指定的权限是要从文件中删除的，所以，如果该文件原来的初始化权限是777，那么执行命令umask 022以后，该文件的权限将变为755：如果该文件原来的初始化权限是666，那么该文件的权限将变为644。

可以使用下面的命令检查新创建文件的默认权限：

umask -S

## which

查找并显示给定命令的绝对路径

## 补充说明

**which命令** 用于查找并显示给定命令的绝对路径，环境变量PATH中保存了查找命令时需要遍历的目录。which指令会在环境变量$PATH设置的目录里查找符合条件的文件。也就是说，使用which命令，就可以看到某个系统命令是否存在，以及执行的到底是哪一个位置的命令。

### 语法

which(选项)(参数)

### 选项

-n<文件名长度>：制定文件名长度，指定的长度必须大于或等于所有文件中最长的文件名；

-p<文件名长度>：与-n参数相同，但此处的<文件名长度>包含了文件的路径；

-w：指定输出时栏位的宽度；

-V：显示版本信息。

### 参数

指令名：指令名列表。

### 实例

查找文件、显示命令路径：

[root@localhost ~]# which pwd

/bin/pwd

[root@localhost ~]# which adduser

/usr/sbin/adduser

说明：which是根据使用者所配置的 PATH 变量内的目录去搜寻可运行档的！所以，不同的 PATH 配置内容所找到的命令当然不一样的！

用 which 去找出 cd

[root@localhost ~]# which cd

cd: shell built-in command

cd 这个常用的命令竟然找不到啊！为什么呢？这是因为 cd 是bash 内建的命令！ 但是 which 默认是找 PATH 内所规范的目录，所以当然一定找不到的！

## whereis

查找二进制程序、代码等相关文件路径

## 补充说明

**whereis命令** 用来定位指令的二进制程序、源代码文件和man手册页等相关文件的路径。

whereis命令只能用于程序名的搜索，而且只搜索二进制文件（参数-b）、man说明文件（参数-m）和源代码文件（参数-s）。如果省略参数，则返回所有信息。

和find相比，whereis查找的速度非常快，这是因为linux系统会将 系统内的所有文件都记录在一个数据库文件中，当使用whereis和下面即将介绍的locate时，会从数据库中查找数据，而不是像find命令那样，通 过遍历硬盘来查找，效率自然会很高。 但是该数据库文件并不是实时更新，默认情况下时一星期更新一次，因此，我们在用whereis和locate 查找文件时，有时会找到已经被删除的数据，或者刚刚建立文件，却无法查找到，原因就是因为数据库文件没有被更新。

### 语法

whereis(选项)(参数)

### 选项

-b：只查找二进制文件；

-B<目录>：只在设置的目录下查找二进制文件；

-f：不显示文件名前的路径名称；

-m：只查找说明文件；

-M<目录>：只在设置的目录下查找说明文件；

-s：只查找原始代码文件；

-S<目录>只在设置的目录下查找原始代码文件；

-u：查找不包含指定类型的文件。

### 参数

指令名：要查找的二进制程序、源文件和man手册页的指令名。

### 实例

将相关的文件都查找出来

[root@localhost ~]# whereis tomcat

tomcat:

[root@localhost ~]# whereis svn

svn: /usr/bin/svn /usr/local/svn /usr/share/man/man1/svn.1.gz

说明：tomcat没安装，找不出来，svn安装找出了很多相关文件

只将二进制文件查找出来

[root@localhost ~]# whereis -b svn

svn: /usr/bin/svn /usr/local/svn

[root@localhost ~]# whereis -m svn

svn: /usr/share/man/man1/svn.1.gz

[root@localhost ~]# whereis -s svn

svn:

说明：whereis -m svn查出说明文档路径，whereis -s svn找source源文件。

## scp

加密的方式在本地主机和远程主机之间复制文件

补充说明

**scp命令** 用于在Linux下进行远程拷贝文件的命令，和它类似的命令有cp，不过cp只是在本机进行拷贝不能跨服务器，而且scp传输是加密的。可能会稍微影响一下速度。当你服务器硬盘变为只读read only system时，用scp可以帮你把文件移出来。另外，scp还非常不占资源，不会提高多少系统负荷，在这一点上，rsync就远远不及它了。虽然 rsync比scp会快一点，但当小文件众多的情况下，rsync会导致硬盘I/O非常高，而scp基本不影响系统正常使用。

**语法**

scp(选项)(参数)

**选项**

-1：使用ssh协议版本1；

-2：使用ssh协议版本2；

-4：使用ipv4；

-6：使用ipv6；

-B：以批处理模式运行；

-C：使用压缩；

-F：指定ssh配置文件；

-i：identity\_file 从指定文件中读取传输时使用的密钥文件（例如亚马逊云pem），此参数直接传递给ssh；

-l：指定宽带限制；

-o：指定使用的ssh选项；

-P：指定远程主机的端口号；

-p：保留文件的最后修改时间，最后访问时间和权限模式；

-q：不显示复制进度；

-r：以递归方式复制。

**参数**

* 源文件：指定要复制的源文件。
* 目标文件：目标文件。格式为user@host：filename（文件名为目标文件的名称）。

**实例**

从远程复制到本地的scp命令与上面的命令雷同，只要将从本地复制到远程的命令后面2个参数互换顺序就行了。

**从远处复制文件到本地目录**

scp root@10.10.10.10:/opt/soft/nginx-0.5.38.tar.gz /opt/soft/

从10.10.10.10机器上的/opt/soft/的目录中下载nginx-0.5.38.tar.gz 文件到本地/opt/soft/目录中。

**从亚马逊云复制OpenVPN到本地目录**

scp -i amazon.pem ubuntu@10.10.10.10:/usr/local/openvpn\_as/etc/exe/openvpn-connect-2.1.3.110.dmg openvpn-connect-2.1.3.110.dmg

从10.10.10.10机器上下载openvpn安装文件到本地当前目录来。

**从远处复制到本地**

scp -r root@10.10.10.10:/opt/soft/mongodb /opt/soft/

从10.10.10.10机器上的/opt/soft/中下载mongodb目录到本地的/opt/soft/目录来。

**上传本地文件到远程机器指定目录**

scp /opt/soft/nginx-0.5.38.tar.gz root@10.10.10.10:/opt/soft/scptest

# 指定端口 2222

scp -rp -P 2222 /opt/soft/nginx-0.5.38.tar.gz root@10.10.10.10:/opt/soft/scptest

复制本地/opt/soft/目录下的文件nginx-0.5.38.tar.gz到远程机器10.10.10.10的opt/soft/scptest目录。

**上传本地目录到远程机器指定目录**

scp -r /opt/soft/mongodb root@10.10.10.10:/opt/soft/scptest

上传本地目录/opt/soft/mongodb到远程机器10.10.10.10上/opt/soft/scptest的目录中去。

## read

从键盘读取变量值

## 补充说明

**read命令** 从键盘读取变量的值，通常用在shell脚本中与用户进行交互的场合。该命令可以一次读取多个变量的值，变量和输入的值都需要使用空格隔开。在read命令后面，如果没有指定变量名，读取的数据将被自动赋值给特定的变量REPLY

### 语法

read(选项)(参数)

### 选项

-p：指定读取值时的提示符；

-t：指定读取值时等待的时间（秒）。

### 参数

变量：指定读取值的变量名。

### 实例

下面的列表给出了read命令的常用方式：

read 1987name

从标准输入读取输入并赋值给变量1987name。

read first last

从标准输入读取输入到第一个空格或者回车，将输入的第一个单词放到变量first中，并将该行其他的输入放在变量last中。

read

从标准输入读取一行并赋值给特定变量REPLY。

read -a arrayname

把单词清单读入arrayname的数组里。

read -p "text"

打印提示（text），等待输入，并将输入存储在REPLY中。

read -r line

允许输入包含反斜杠。

read -t 3

指定读取等待时间为3秒。

read -n 2 var

从输入中读取两个字符并存入变量var，不需要按回车读取。

read -d ":" var

用定界符“:”结束输入行。

## read命令示例

从标准输入读取输入并赋值给变量1987name。

#read 1987name #等待读取输入，直到回车后表示输入完毕，并将输入赋值给变量answer

HelloWorld #控制台输入Hello

#echo $1987name #打印变量

HelloWorld

等待一组输入，每个单词之间使用空格隔开，直到回车结束，并分别将单词依次赋值给这三个读入变量。

#read one two three

1 2 3 #在控制台输入1 2 3，它们之间用空格隔开。

#echo "one = $one, two = $two, three = $three"

one = 1, two = 2, three = 3

REPLY示例

#read #等待控制台输入，并将结果赋值给特定内置变量REPLY。

This is REPLY #在控制台输入该行。

#echo $REPLY #打印输出特定内置变量REPLY，以确认是否被正确赋值。

This is REPLY

-p选项示例

#read -p "Enter your name: " #输出文本提示，同时等待输入，并将结果赋值给REPLY。

Enter you name: stephen #在提示文本之后输入stephen

#echo $REPLY

stephen

等待控制台输入，并将输入信息视为数组，赋值给数组变量friends，输入信息用空格隔开数组的每个元素。

#read -a friends

Tim Tom Helen

#echo "They are ${friends[0]}, ${friends[1]} and ${friends[2]}."

They are Tim, Tom and Helen.

**补充一个终端输入密码时候，不让密码显示出来的例子。**

方法1：

#!/bin/bash

read -p "输入密码：" -s pwd

echo

echo password read, is "$pwd"

方法2：

#!/bin/bash

stty -echo

read -p "输入密码：" pwd

stty echo

echo

echo 输入完毕。

其中，选项-echo禁止将输出发送到终端，而选项echo则允许发送输出。

使用read命令从键盘读取变量值，并且将值赋给指定的变量，输入如下命令：

read v1 v3 #读取变量值

执行上面的指令以后，要求键入两个数据，如下所示：

Linux c+ #输入数据

完成之后，可以使用echo命令将指定的变量值输出查看，输入如下命令：

echo $v1 $v3 #输出变量的值

执行输出变量值的命令以后，将显示用户所输入的数据值，如下所示：

Linux c+ #输出变量值

注意：使用echo命令输出变量值时，必须在变量名前添加符号$。否则，echo将直接输出变量名。

## sort

将文件进行排序并输出

## 补充说明

**sort命令** 是在Linux里非常有用，它将文件进行排序，并将排序结果标准输出。sort命令既可以从特定的文件，也可以从stdin中获取输入。

### 语法

sort(选项)(参数)

### 选项

-b：忽略每行前面开始出的空格字符；

-c：检查文件是否已经按照顺序排序；

-d：排序时，处理英文字母、数字及空格字符外，忽略其他的字符；

-f：排序时，将小写字母视为大写字母；

-i：排序时，除了040至176之间的ASCII字符外，忽略其他的字符；

-m：将几个排序号的文件进行合并；

-M：将前面3个字母依照月份的缩写进行排序；

-n：依照数值的大小排序；

-o<输出文件>：将排序后的结果存入制定的文件；

-r：以相反的顺序来排序；

-t<分隔字符>：指定排序时所用的栏位分隔字符；

+<起始栏位>-<结束栏位>：以指定的栏位来排序，范围由起始栏位到结束栏位的前一栏位。

### 参数

文件：指定待排序的文件列表。

### 实例

sort将文件/文本的每一行作为一个单位，相互比较，比较原则是从首字符向后，依次按ASCII码值进行比较，最后将他们按升序输出。

root@[mail text]# cat sort.txt

aaa:10:1.1

ccc:30:3.3

ddd:40:4.4

bbb:20:2.2

eee:50:5.5

eee:50:5.5

[root@mail text]# sort sort.txt

aaa:10:1.1

bbb:20:2.2

ccc:30:3.3

ddd:40:4.4

eee:50:5.5

eee:50:5.5

忽略相同行使用-u选项或者uniq：

[root@mail text]# cat sort.txt

aaa:10:1.1

ccc:30:3.3

ddd:40:4.4

bbb:20:2.2

eee:50:5.5

eee:50:5.5

[root@mail text]# sort -u sort.txt

aaa:10:1.1

bbb:20:2.2

ccc:30:3.3

ddd:40:4.4

eee:50:5.5

或者

[root@mail text]# uniq sort.txt

aaa:10:1.1

ccc:30:3.3

ddd:40:4.4

bbb:20:2.2

eee:50:5.5

sort的-n、-r、-k、-t选项的使用：

[root@mail text]# cat sort.txt

AAA:BB:CC

aaa:30:1.6

ccc:50:3.3

ddd:20:4.2

bbb:10:2.5

eee:40:5.4

eee:60:5.1

#将BB列按照数字从小到大顺序排列：

[root@mail text]# sort -nk 2 -t: sort.txt

AAA:BB:CC

bbb:10:2.5

ddd:20:4.2

aaa:30:1.6

eee:40:5.4

ccc:50:3.3

eee:60:5.1

#将CC列数字从大到小顺序排列：

[root@mail text]# sort -nrk 3 -t: sort.txt

eee:40:5.4

eee:60:5.1

ddd:20:4.2

ccc:50:3.3

bbb:10:2.5

aaa:30:1.6

AAA:BB:CC

# -n是按照数字大小排序，-r是以相反顺序，-k是指定需要爱排序的栏位，-t指定栏位分隔符为冒号

**-k选项的具体语法格式：**

-k选项的语法格式：

FStart.CStart Modifie,FEnd.CEnd Modifier

-------Start--------,-------End--------

FStart.CStart 选项 , FEnd.CEnd 选项

这个语法格式可以被其中的逗号,分为两大部分， **Start** 部分和 **End** 部分。Start部分也由三部分组成，其中的Modifier部分就是我们之前说过的类似n和r的选项部分。我们重点说说Start部分的FStart和C.Start。C.Start也是可以省略的，省略的话就表示从本域的开头部分开始。FStart.CStart，其中FStart就是表示使用的域，而CStart则表示在FStart域中从第几个字符开始算“排序首字符”。同理，在End部分中，你可以设定FEnd.CEnd，如果你省略.CEnd，则表示结尾到“域尾”，即本域的最后一个字符。或者，如果你将CEnd设定为0(零)，也是表示结尾到“域尾”。

从公司英文名称的第二个字母开始进行排序：

$ sort -t ' ' -k 1.2 facebook.txt

baidu 100 5000

sohu 100 4500

google 110 5000

guge 50 3000

使用了-k 1.2，表示对第一个域的第二个字符开始到本域的最后一个字符为止的字符串进行排序。你会发现baidu因为第二个字母是a而名列榜首。sohu和 google第二个字符都是o，但sohu的h在google的o前面，所以两者分别排在第二和第三。guge只能屈居第四了。

只针对公司英文名称的第二个字母进行排序，如果相同的按照员工工资进行降序排序：

$ sort -t ' ' -k 1.2,1.2 -nrk 3,3 facebook.txt

baidu 100 5000

google 110 5000

sohu 100 4500

guge 50 3000

由于只对第二个字母进行排序，所以我们使用了-k 1.2,1.2的表示方式，表示我们“只”对第二个字母进行排序。（如果你问“我使用-k 1.2怎么不行？”，当然不行，因为你省略了End部分，这就意味着你将对从第二个字母起到本域最后一个字符为止的字符串进行排序）。对于员工工资进行排 序，我们也使用了-k 3,3，这是最准确的表述，表示我们“只”对本域进行排序，因为如果你省略了后面的3，就变成了我们“对第3个域开始到最后一个域位置的内容进行排序” 了。

## wc

统计文件的字节数、字数、行数

## 补充说明

**wc命令** 统计指定文件中的字节数、字数、行数，并将统计结果显示输出。利用wc指令我们可以计算文件的Byte数、字数或是列数，若不指定文件名称，或是所给予的文件名为“-”，则wc指令会从标准输入设备读取数据。wc同时也给出所指定文件的总统计数。

### 语法

wc(选项)(参数)

wc [选项]... [文件]...

wc [选项]... --files0-from=F

### 选项

-c # 统计字节数，或--bytes或——chars：只显示Bytes数；。

-l # 统计行数，或——lines：只显示列数；。

-m # 统计字符数。这个标志不能与 -c 标志一起使用。

-w # 统计字数，或——words：只显示字数。一个字被定义为由空白、跳格或换行字符分隔的字符串。

-L # 打印最长行的长度。

-help # 显示帮助信息

--version # 显示版本信息

### 参数

文件：需要统计的文件列表。

## 例子

wc -l \* # 统计当前目录下的所有文件行数

wc -l \*.js   # 统计当前目录下的所有 .js 后缀的文件行数

find . \* | xargs wc -l # 当前目录以及子目录的所有文件行数

wc test.txt # 查看文件的字节数、字数、行数

查看文件的字节数、字数、行数

wc test.txt

# 输出结果

7 8 70 test.txt

行数 单词数 字节数 文件名

用wc命令怎么做到只打印统计数字不打印文件名

wc -l test.txt

# 输出结果

7 test.txt

用来统计当前目录下的文件数

ls -l | wc -l

# 输出结果

8

## ftp

用来设置文件系统相关功能

## 补充说明

**ftp命令** 用来设置文件系统相关功能。ftp服务器在网上较为常见，Linux ftp命令的功能是用命令的方式来控制在本地机和远程机之间传送文件，这里详细介绍Linux ftp命令的一些经常使用的命令，相信掌握了这些使用Linux进行ftp操作将会非常容易。

### 语法

ftp(选项)(参数)

### 选项

-d：详细显示指令执行过程，便于排错或分析程序执行的情况；

-i：关闭互动模式，不询问任何问题；

-g：关闭本地主机文件名称支持特殊字符的扩充特性；

-n：不使用自动登录；

-v：显示指令执行过程。

### 参数

主机：指定要连接的FTP服务器的主机名或ip地址。

### 实例

ftp> ascii # 设定以ASCII方式传送文件(缺省值)

ftp> bell # 每完成一次文件传送,报警提示.

ftp> binary # 设定以二进制方式传送文件.

ftp> bye # 终止主机FTP进程,并退出FTP管理方式.

ftp> case # 当为ON时,用MGET命令拷贝的文件名到本地机器中,全部转换为小写字母.

ftp> cd # 同UNIX的CD命令.

ftp> cdup # 返回上一级目录.

ftp> chmod # 改变远端主机的文件权限.

ftp> close # 终止远端的FTP进程,返回到FTP命令状态, 所有的宏定义都被删除.

ftp> delete # 删除远端主机中的文件.

ftp> dir [remote-directory] [local-file] # 列出当前远端主机目录中的文件.如果有本地文件,就将结果写至本地文件.

ftp> get [remote-file] [local-file] # 从远端主机中传送至本地主机中.

ftp> help [command] # 输出命令的解释.

ftp> lcd # 改变当前本地主机的工作目录,如果缺省,就转到当前用户的HOME目录.

ftp> ls [remote-directory] [local-file] # 同DIR.

ftp> macdef # 定义宏命令.

ftp> mdelete [remote-files] # 删除一批文件.

ftp> mget [remote-files] # 从远端主机接收一批文件至本地主机.

ftp> mkdir directory-name # 在远端主机中建立目录.

ftp> mput local-files # 将本地主机中一批文件传送至远端主机.

ftp> open host [port] # 重新建立一个新的连接.

ftp> prompt # 交互提示模式.

ftp> put local-file [remote-file] # 将本地一个文件传送至远端主机中.

ftp> pwd # 列出当前远端主机目录.

ftp> quit # 同BYE.

ftp> recv remote-file [local-file] # 同GET.

ftp> rename [from] [to] # 改变远端主机中的文件名.

ftp> rmdir directory-name # 删除远端主机中的目录.

ftp> send local-file [remote-file] # 同PUT.

ftp> status # 显示当前FTP的状态.

ftp> system # 显示远端主机系统类型.

ftp> user user-name [password] [account] # 重新以别的用户名登录远端主机.

ftp> ? [command] # 同HELP. [command]指定需要帮助的命令名称。如果没有指定 command，ftp 将显示全部命令的列表。

ftp> ! # 从 ftp 子系统退出到外壳。

关闭FTP连接

bye

exit

quit

下载文件

ftp> get readme.txt # 下载 readme.txt 文件

ftp> mget \*.txt # 下载

上传文件

ftp> put /path/readme.txt # 上传 readme.txt 文件

ftp> mput \*.txt # 可以上传多个文件

## du

显示每个文件和目录的磁盘使用空间

## 补充说明

**du命令** 也是查看使用空间的，但是与df命令不同的是Linux du命令是对文件和目录磁盘使用的空间的查看，还是和df命令有一些区别的。

### 语法

du [选项][文件]

### 选项

-a或-all 显示目录中个别文件的大小。

-b或-bytes 显示目录或文件大小时，以byte为单位。

-c或--total 除了显示个别目录或文件的大小外，同时也显示所有目录或文件的总和。

-k或--kilobytes 以KB(1024bytes)为单位输出。

-m或--megabytes 以MB为单位输出。

-s或--summarize 仅显示总计，只列出最后加总的值。

-h或--human-readable 以K，M，G为单位，提高信息的可读性。

-x或--one-file-xystem 以一开始处理时的文件系统为准，若遇上其它不同的文件系统目录则略过。

-L<符号链接>或--dereference<符号链接> 显示选项中所指定符号链接的源文件大小。

-S或--separate-dirs 显示个别目录的大小时，并不含其子目录的大小。

-X<文件>或--exclude-from=<文件> 在<文件>指定目录或文件。

--exclude=<目录或文件> 略过指定的目录或文件。

-D或--dereference-args 显示指定符号链接的源文件大小。

-H或--si 与-h参数相同，但是K，M，G是以1000为换算单位。

-l或--count-links 重复计算硬件链接的文件。

### 实例

显示目录或者文件所占空间：

root@localhost [test]# du

608 ./test6

308 ./test4

4 ./scf/lib

4 ./scf/service/deploy/product

4 ./scf/service/deploy/info

12 ./scf/service/deploy

16 ./scf/service

4 ./scf/doc

4 ./scf/bin

32 ./scf

8 ./test3

1288 .

只显示当前目录下面的子目录的目录大小和当前目录的总的大小，最下面的1288为当前目录的总大小

显示指定文件所占空间：

[root@localhost test]# du log2012.log

300 log2012.log

查看指定目录的所占空间：

[root@localhost test]# du scf

4 scf/lib

4 scf/service/deploy/product

4 scf/service/deploy/info

12 scf/service/deploy

16 scf/service

4 scf/doc

4 scf/bin

32 scf

显示多个文件所占空间：

[root@localhost test]# du log30.tar.gz log31.tar.gz

4 log30.tar.gz

4 log31.tar.gz

只显示总和的大小:

[root@localhost test]# du -s

1288 .

[root@localhost test]# du -s scf

32 scf

[root@localhost test]# cd ..

[root@localhost soft]# du -s test

1288 test

显示总和的大小且易读:

du -sh $dir

## mkdir

用来创建目录

## 补充说明

**mkdir命令** 用来创建目录。该命令创建由dirname命名的目录。如果在目录名的前面没有加任何路径名，则在当前目录下创建由dirname指定的目录；如果给出了一个已经存在的路径，将会在该目录下创建一个指定的目录。在创建目录时，应保证新建的目录与它所在目录下的文件没有重名。

注意：在创建文件时，不要把所有的文件都存放在主目录中，可以创建子目录，通过它们来更有效地组织文件。最好采用前后一致的命名方式来区分文件和目录。例如，目录名可以以大写字母开头，这样，在目录列表中目录名就出现在前面。

在一个子目录中应包含类型相似或用途相近的文件。例如，应建立一个子目录，它包含所有的数据库文件，另有一个子目录应包含电子表格文件，还有一个子目录应包含文字处理文档，等等。目录也是文件，它们和普通文件一样遵循相同的命名规则，并且利用全路径可以唯一地指定一个目录。

### 语法

mkdir (选项)(参数)

### 选项

-Z：设置安全上下文，当使用SELinux时有效；

-m<目标属性>或--mode<目标属性>建立目录的同时设置目录的权限；

-p或--parents 若所要建立目录的上层目录目前尚未建立，则会一并建立上层目录；

--version 显示版本信息。

### 参数

目录：指定要创建的目录列表，多个目录之间用空格隔开。

### 实例

在目录/usr/meng下建立子目录test，并且只有文件主有读、写和执行权限，其他人无权访问

mkdir -m 700 /usr/meng/test

在当前目录中建立bin和bin下的os\_1目录，权限设置为文件主可读、写、执行，同组用户可读和执行，其他用户无权访问

mkdir -p-m 750 bin/os\_1

## pwd

绝对路径方式显示用户当前工作目录

## 补充说明

**pwd命令** 以绝对路径的方式显示用户当前工作目录。命令将当前目录的全路径名称（从根目录）写入标准输出。全部目录使用/分隔。第一个/表示根目录，最后一个目录是当前目录。执行pwd命令可立刻得知您目前所在的工作目录的绝对路径名称。

### 语法

pwd（选项）

### 选项

--help：显示帮助信息；

--version：显示版本信息。

### 实例

[root@localhost ~]# pwd

/root

## tree

树状图列出目录的内容

## 补充说明

**tree命令** 以树状图列出目录的内容。

### 语法

tree(选项)(参数)

### 选项

------- 列表选项 -------

-a # 显示所有文件和目录。

-d # 先是目录名称而非文件。

-l # 如遇到性质为符号连接的目录，直接列出该连接所指向的原始目录。

-f # 在每个文件或目录之前，显示完整的相对路径名称。

-x # 将范围局限在现行的文件系统中，若指定目录下的某些子目录，其存放于另一个文件系统上，则将该目录予以排除在寻找范围外。

-L level # 限制目录显示层级。

-R # Rerun tree when max dir level reached.

-P pattern # <范本样式> 只显示符合范本样式的文件和目录名称。

-I pattern # Do not list files that match the given pattern.

--ignore-case # Ignore case when pattern matching.

--matchdirs # Include directory names in -P pattern matching.

--noreport # Turn off file/directory count at end of tree listing.

--charset X # Use charset X for terminal/HTML and indentation line output.

--filelimit # # Do not descend dirs with more than # files in them.

--timefmt <f> # Print and format time according to the format <f>.

-o filename # Output to file instead of stdout.

-------- 文件选项 ---------

-q # 用“？”号取代控制字符，列出文件和目录名称。

-N # 直接列出文件和目录名称，包括控制字符。

-Q # Quote filenames with double quotes.

-p # 列出权限标示。

-u # 列出文件或目录的拥有者名称，没有对应的名称时，则显示用户识别码。

-g # 列出文件或目录的所属群组名称，没有对应的名称时，则显示群组识别码。

-s # 列出文件和目录大小。

-h # Print the size in a more human readable way.

--si # Like -h, but use in SI units (powers of 1000).

-D # 列出文件或目录的更改时间。

-F # 在执行文件，目录，Socket，符号连接，管道名称名称，各自加上"\*"，"/"，"@"，"|"号。

--inodes # Print inode number of each file.

--device # Print device ID number to which each file belongs.

------- 排序选项 -------

-v # Sort files alphanumerically by version.

-t # 用文件和目录的更改时间排序。

-c # Sort files by last status change time.

-U # Leave files unsorted.

-r # Reverse the order of the sort.

--dirsfirst # List directories before files (-U disables).

--sort X # Select sort: name,version,size,mtime,ctime.

------- 图形选项 ------

-i # 不以阶梯状列出文件和目录名称。

-A # 使用ASNI绘图字符显示树状图而非以ASCII字符组合。

-S # Print with CP437 (console) graphics indentation lines.

-n # Turn colorization off always (-C overrides).

-C # 在文件和目录清单加上色彩，便于区分各种类型。

------- XML / HTML / JSON选项 -------

-X # Prints out an XML representation of the tree.

-J # Prints out an JSON representation of the tree.

-H baseHREF # Prints out HTML format with baseHREF as top directory.

-T string # Replace the default HTML title and H1 header with string.

--nolinks # Turn off hyperlinks in HTML output.

---- 杂项选项 ----

--version # 输入版本信息。

--help # 打印使用帮助信息。

-- # Options processing terminator.

### 参数

目录：执行tree指令，它会列出指定目录下的所有文件，包括子目录里的文件。

### 实例

列出目录/private/ 第一级文件名

tree /private/ -L 1

/private/

├── etc

├── tftpboot

├── tmp

└── var

忽略文件夹

tree -I node\_modules # 忽略当前目录文件夹node\_modules

tree -P node\_modules # 列出当前目录文件夹node\_modules的目录结构

tree -P node\_modules -L 2 # 显示目录node\_modules两层的目录树结构

tree -L 2 > /home/www/tree.txt # 当前目录结果存到 tree.txt 文件中

忽略多个文件夹

tree -I 'node\_modules|icon|font' -L 2

## umount

用于卸载已经加载的文件系统

## 补充说明

**umount命令** 用于卸载已经加载的文件系统。利用设备名或挂载点都能umount文件系统，不过最好还是通过挂载点卸载，以免使用绑定挂载（一个设备，多个挂载点）时产生混乱。

### 语法

umount(选项)(参数)

### 选项

-a：卸除/etc/mtab中记录的所有文件系统；

-h：显示帮助；

-n：卸除时不要将信息存入/etc/mtab文件中；

-r：若无法成功卸除，则尝试以只读的方式重新挂入文件系统；

-t<文件系统类型>：仅卸除选项中所指定的文件系统；

-v：执行时显示详细的信息；

-V：显示版本信息。

### 参数

文件系统：指定要卸载的文件系统或者其对应的设备文件名。

### 实例

下面两条命令分别通过设备名和挂载点卸载文件系统，同时输出详细信息：

通过设备名卸载

umount -v /dev/sda1

/dev/sda1 umounted

通过挂载点卸载

umount -v /mnt/mymount/

/tmp/diskboot.img umounted

如果设备正忙，卸载即告失败。卸载失败的常见原因是，某个打开的shell当前目录为挂载点里的某个目录：

umount -v /mnt/mymount/

umount: /mnt/mymount: device is busy

umount: /mnt/mymount: device is busy

有时，导致设备忙的原因并不好找。碰到这种情况时，可以用lsof列出已打开文件，然后搜索列表查找待卸载的挂载点：

lsof | grep mymount 查找mymount分区里打开的文件

bash 9341 francois cwd DIR 8,1 1024 2 /mnt/mymount

从上面的输出可知，mymount分区无法卸载的原因在于，francois运行的PID为9341的bash进程。

对付系统文件正忙的另一种方法是执行延迟卸载：

umount -vl /mnt/mymount/ 执行延迟卸载

延迟卸载（lazy unmount）会立即卸载目录树里的文件系统，等到设备不再繁忙时才清理所有相关资源。卸载可移动存储介质还可以用eject命令。下面这条命令会卸载cd并弹出CD：

eject /dev/cdrom 卸载并弹出CD

## sync

用于强制被改变的内容立刻写入磁盘

补充说明

**sync命令** 用于强制被改变的内容立刻写入磁盘，更新超块信息。

在Linux/Unix系统中，在文件或数据处理过程中一般先放到内存缓冲区中，等到适当的时候再写入磁盘，以提高系统的运行效率。sync命令则可用来强制将内存缓冲区中的数据立即写入磁盘中。用户通常不需执行sync命令，系统会自动执行update或bdflush操作，将缓冲区的数据写 入磁盘。只有在update或bdflush无法执行或用户需要非正常关机时，才需手动执行sync命令。

**语法**

sync(选项)

**选项**

--help：显示帮助；

--version：显示版本信息。

**buffer与cache**

* buffer：为了解决写磁盘的效率
* cache：为了解决读磁盘的效率

linux系统为了提高读写磁盘的效率，会先将数据放在一块buffer中。在写磁盘时并不是立即将数据写到磁盘中，而是先写入这块buffer中了。此时如果重启系统，就可能造成数据丢失。

sync命令用来flush文件系统buffer，这样数据才会真正的写到磁盘中，并且buffer才能够释放出来，flush就是用来清空buffer。sync命令会强制将数据写入磁盘中，并释放该数据对应的buffer，所以常常会在写磁盘后输入sync命令来将数据真正的写入磁盘。

如果不去手动的输入sync命令来真正的去写磁盘，linux系统也会周期性的去sync数据。

## rsync

远程数据同步工具

补充说明

**rsync命令** 是一个远程数据同步工具，可通过LAN/WAN快速同步多台主机间的文件。rsync使用所谓的“rsync算法”来使本地和远程两个主机之间的文件达到同步，这个算法只传送两个文件的不同部分，而不是每次都整份传送，因此速度相当快。 rsync是一个功能非常强大的工具，其命令也有很多功能特色选项，我们下面就对它的选项一一进行分析说明。

**语法**

rsync [OPTION]... SRC DEST

rsync [OPTION]... SRC [USER@]host:DEST

rsync [OPTION]... [USER@]HOST:SRC DEST

rsync [OPTION]... [USER@]HOST::SRC DEST

rsync [OPTION]... SRC [USER@]HOST::DEST

rsync [OPTION]... rsync://[USER@]HOST[:PORT]/SRC [DEST]

对应于以上六种命令格式，rsync有六种不同的工作模式：

1. 拷贝本地文件。当SRC和DES路径信息都不包含有单个冒号":"分隔符时就启动这种工作模式。如：rsync -a /data /backup
2. 使用一个远程shell程序(如rsh、ssh)来实现将本地机器的内容拷贝到远程机器。当DST路径地址包含单个冒号":"分隔符时启动该模式。如：rsync -avz \*.c foo:src
3. 使用一个远程shell程序(如rsh、ssh)来实现将远程机器的内容拷贝到本地机器。当SRC地址路径包含单个冒号":"分隔符时启动该模式。如：rsync -avz foo:src/bar /data
4. 从远程rsync服务器中拷贝文件到本地机。当SRC路径信息包含"::"分隔符时启动该模式。如：rsync -av root@192.168.78.192::www /databack
5. 从本地机器拷贝文件到远程rsync服务器中。当DST路径信息包含"::"分隔符时启动该模式。如：rsync -av /databack root@192.168.78.192::www
6. 列远程机的文件列表。这类似于rsync传输，不过只要在命令中省略掉本地机信息即可。如：rsync -v rsync://192.168.78.192/www

**选项**

-v, --verbose 详细模式输出。

-q, --quiet 精简输出模式。

-c, --checksum 打开校验开关，强制对文件传输进行校验。

-a, --archive 归档模式，表示以递归方式传输文件，并保持所有文件属性，等于-rlptgoD。

-r, --recursive 对子目录以递归模式处理。

-R, --relative 使用相对路径信息。

-b, --backup 创建备份，也就是对于目的已经存在有同样的文件名时，将老的文件重新命名为~filename。可以使用--suffix选项来指定不同的备份文件前缀。

--backup-dir 将备份文件(如~filename)存放在在目录下。

-suffix=SUFFIX 定义备份文件前缀。

-u, --update 仅仅进行更新，也就是跳过所有已经存在于DST，并且文件时间晚于要备份的文件，不覆盖更新的文件。

-l, --links 保留软链结。

-L, --copy-links 想对待常规文件一样处理软链结。

--copy-unsafe-links 仅仅拷贝指向SRC路径目录树以外的链结。

--safe-links 忽略指向SRC路径目录树以外的链结。

-H, --hard-links 保留硬链结。

-p, --perms 保持文件权限。

-o, --owner 保持文件属主信息。

-g, --group 保持文件属组信息。

-D, --devices 保持设备文件信息。

-t, --times 保持文件时间信息。

-S, --sparse 对稀疏文件进行特殊处理以节省DST的空间。

-n, --dry-run现实哪些文件将被传输。

-w, --whole-file 拷贝文件，不进行增量检测。

-x, --one-file-system 不要跨越文件系统边界。

-B, --block-size=SIZE 检验算法使用的块尺寸，默认是700字节。

-e, --rsh=command 指定使用rsh、ssh方式进行数据同步。

--rsync-path=PATH 指定远程服务器上的rsync命令所在路径信息。

-C, --cvs-exclude 使用和CVS一样的方法自动忽略文件，用来排除那些不希望传输的文件。

--existing 仅仅更新那些已经存在于DST的文件，而不备份那些新创建的文件。

--delete 删除那些DST中SRC没有的文件。

--delete-excluded 同样删除接收端那些被该选项指定排除的文件。

--delete-after 传输结束以后再删除。

--ignore-errors 及时出现IO错误也进行删除。

--max-delete=NUM 最多删除NUM个文件。

--partial 保留那些因故没有完全传输的文件，以是加快随后的再次传输。

--force 强制删除目录，即使不为空。

--numeric-ids 不将数字的用户和组id匹配为用户名和组名。

--timeout=time ip超时时间，单位为秒。

-I, --ignore-times 不跳过那些有同样的时间和长度的文件。

--size-only 当决定是否要备份文件时，仅仅察看文件大小而不考虑文件时间。

--modify-window=NUM 决定文件是否时间相同时使用的时间戳窗口，默认为0。

-T --temp-dir=DIR 在DIR中创建临时文件。

--compare-dest=DIR 同样比较DIR中的文件来决定是否需要备份。

-P 等同于 --partial。

--progress 显示备份过程。

-z, --compress 对备份的文件在传输时进行压缩处理。

--exclude=PATTERN 指定排除不需要传输的文件模式。

--include=PATTERN 指定不排除而需要传输的文件模式。

--exclude-from=FILE 排除FILE中指定模式的文件。

--include-from=FILE 不排除FILE指定模式匹配的文件。

--version 打印版本信息。

--address 绑定到特定的地址。

--config=FILE 指定其他的配置文件，不使用默认的rsyncd.conf文件。

--port=PORT 指定其他的rsync服务端口。

--blocking-io 对远程shell使用阻塞IO。

-stats 给出某些文件的传输状态。

--progress 在传输时现实传输过程。

--log-format=formAT 指定日志文件格式。

--password-file=FILE 从FILE中得到密码。

--bwlimit=KBPS 限制I/O带宽，KBytes per second。

-h, --help 显示帮助信息。

**实例**

**SSH方式**

首先在服务端启动ssh服务：

service sshd start

启动 sshd： [确定]

**使用rsync进行同步**

接下来就可以在客户端使用rsync命令来备份服务端上的数据了，SSH方式是通过系统用户来进行备份的，如下：

rsync -vzrtopg --progress -e ssh --delete work@172.16.78.192:/www/\* /databack/experiment/rsync

work@172.16.78.192's password:

receiving file list ...

5 files to consider

test/

a

0 100% 0.00kB/s 527:35:41 (1, 20.0% of 5)

b

67 100% 65.43kB/s 0:00:00 (2, 40.0% of 5)

c

0 100% 0.00kB/s 527:35:41 (3, 60.0% of 5)

dd

100663296 100% 42.22MB/s 0:00:02 (4, 80.0% of 5)

sent 96 bytes received 98190 bytes 11563.06 bytes/sec

total size is 100663363 speedup is 1024.19

上面的信息描述了整个的备份过程，以及总共备份数据的大小。

**后台服务方式**

启动rsync服务，编辑/etc/xinetd.d/rsync文件，将其中的disable=yes改为disable=no，并重启xinetd服务，如下：

vi /etc/xinetd.d/rsync

#default: off

# description: The rsync server is a good addition to an ftp server, as it \

# allows crc checksumming etc.

service rsync {

disable = no

socket\_type = stream

wait = no

user = root

server = /usr/bin/rsync

server\_args = --daemon

log\_on\_failure += USERID

}

/etc/init.d/xinetd restart

停止 xinetd： [确定]

启动 xinetd： [确定]

创建配置文件，默认安装好rsync程序后，并不会自动创建rsync的主配置文件，需要手工来创建，其主配置文件为“/etc/rsyncd.conf”，创建该文件并插入如下内容：

vi /etc/rsyncd.conf

uid=root

gid=root

max connections=4

log file=/var/log/rsyncd.log

pid file=/var/run/rsyncd.pid

lock file=/var/run/rsyncd.lock

secrets file=/etc/rsyncd.passwd

hosts deny=172.16.78.0/22

[www]

comment= backup web

path=/www

read only = no

exclude=test

auth users=work

创建密码文件，采用这种方式不能使用系统用户对客户端进行认证，所以需要创建一个密码文件，其格式为“username:password”，用户名可以和密码可以随便定义，最好不要和系统帐户一致，同时要把创建的密码文件权限设置为600，这在前面的模块参数做了详细介绍。

echo "work:abc123" > /etc/rsyncd.passwd

chmod 600 /etc/rsyncd.passwd

备份，完成以上工作，现在就可以对数据进行备份了，如下：

rsync -avz --progress --delete work@172.16.78.192::www /databack/experiment/rsync

Password:

receiving file list ...

6 files to consider

./ files...

a

0 100% 0.00kB/s 528:20:41 (1, 50.0% of 6)

b

67 100% 65.43kB/s 0:00:00 (2, 66.7% of 6)

c

0 100% 0.00kB/s 528:20:41 (3, 83.3% of 6)

dd

100663296 100% 37.49MB/s 0:00:02 (4, 100.0% of 6)

sent 172 bytes received 98276 bytes 17899.64 bytes/sec

total size is 150995011 speedup is 1533.75

恢复，当服务器的数据出现问题时，那么这时就需要通过客户端的数据对服务端进行恢复，但前提是服务端允许客户端有写入权限，否则也不能在客户端直接对服务端进行恢复，使用rsync对数据进行恢复的方法如下：

rsync -avz --progress /databack/experiment/rsync/ work@172.16.78.192::www

Password:

building file list ...

6 files to consider

./

a

b

67 100% 0.00kB/s 0:00:00 (2, 66.7% of 6)

c

sent 258 bytes received 76 bytes 95.43 bytes/sec

total size is 150995011 speedup is 452080.87

## fdisk

查看磁盘使用情况和磁盘分区

## 补充说明

**fdisk命令** 用于观察硬盘实体使用情况，也可对硬盘分区。它采用传统的问答式界面，而非类似DOS fdisk的cfdisk互动式操作界面，因此在使用上较为不便，但功能却丝毫不打折扣。

### 语法

fdisk(选项)(参数)

### 选项

-b <大小> 扇区大小(512、1024、2048或4096)

-c[=<模式>] 兼容模式：“dos”或“nondos”(默认)

-h 打印此帮助文本

-u[=<单位>] 显示单位：“cylinders”(柱面)或“sectors”(扇区，默认)

-v 打印程序版本

-C <数字> 指定柱面数

-H <数字> 指定磁头数

-S <数字> 指定每个磁道的扇区数

### 参数

设备文件：指定要进行分区或者显示分区的硬盘设备文件。

### 实例

首先选择要进行操作的磁盘：

[root@localhost ~]# fdisk /dev/sdb

输入m列出可以执行的命令：

command (m for help): m

Command action

a toggle a bootable flag

b edit bsd disklabel

c toggle the dos compatibility flag

d delete a partition

l list known partition types

m print this menu

n add a new partition

o create a new empty DOS partition table

p print the partition table

q quit without saving changes

s create a new empty Sun disklabel

t change a partition's system id

u change display/entry units

v verify the partition table

w write table to disk and exit

x extra functionality (experts only)

输入p列出磁盘目前的分区情况：

Command (m for help): p

Disk /dev/sdb: 3221 MB, 3221225472 bytes

255 heads, 63 sectors/track, 391 cylinders

Units = cylinders of 16065 \* 512 = 8225280 bytes

Device Boot Start End Blocks Id System

/dev/sdb1 1 1 8001 8e Linux LVM

/dev/sdb2 2 26 200812+ 83 Linux

输入d然后选择分区，删除现有分区：

Command (m for help): d

Partition number (1-4): 1

Command (m for help): d

Selected partition 2

查看分区情况，确认分区已经删除：

Command (m for help): print

Disk /dev/sdb: 3221 MB, 3221225472 bytes

255 heads, 63 sectors/track, 391 cylinders

Units = cylinders of 16065 \* 512 = 8225280 bytes

Device Boot Start End Blocks Id System

Command (m for help):

输入n建立新的磁盘分区，首先建立两个主磁盘分区：

Command (m for help): n

Command action

e extended

p primary partition (1-4)

p //建立主分区

Partition number (1-4): 1 //分区号

First cylinder (1-391, default 1): //分区起始位置

Using default value 1

last cylinder or +size or +sizeM or +sizeK (1-391, default 391): 100 //分区结束位置，单位为扇区

Command (m for help): n //再建立一个分区

Command action

e extended

p primary partition (1-4)

p

Partition number (1-4): 2 //分区号为2

First cylinder (101-391, default 101):

Using default value 101

Last cylinder or +size or +sizeM or +sizeK (101-391, default 391): +200M //分区结束位置，单位为M

确认分区建立成功：

Command (m for help): p

Disk /dev/sdb: 3221 MB, 3221225472 bytes

255 heads, 63 sectors/track, 391 cylinders

Units = cylinders of 16065 \* 512 = 8225280 bytes

Device Boot Start End Blocks Id System

/dev/sdb1 1 100 803218+ 83 Linux

/dev/sdb2 101 125 200812+ 83 Linux

再建立一个逻辑分区：

Command (m for help): n

Command action

e extended

p primary partition (1-4)

e //选择扩展分区

Partition number (1-4): 3

First cylinder (126-391, default 126):

Using default value 126

Last cylinder or +size or +sizeM or +sizeK (126-391, default 391):

Using default value 391

确认扩展分区建立成功：

Command (m for help): p

Disk /dev/sdb: 3221 MB, 3221225472 bytes

255 heads, 63 sectors/track, 391 cylinders

Units = cylinders of 16065 \* 512 = 8225280 bytes

Device Boot Start End Blocks Id System

/dev/sdb1 1 100 803218+ 83 Linux

/dev/sdb2 101 125 200812+ 83 Linux

/dev/sdb3 126 391 2136645 5 Extended

在扩展分区上建立两个逻辑分区：

Command (m for help): n

Command action

l logical (5 or over)

p primary partition (1-4)

l //选择逻辑分区

First cylinder (126-391, default 126):

Using default value 126

Last cylinder or +size or +sizeM or +sizeK (126-391, default 391): +400M

Command (m for help): n

Command action

l logical (5 or over)

p primary partition (1-4)

l

First cylinder (176-391, default 176):

Using default value 176

Last cylinder or +size or +sizeM or +sizeK (176-391, default 391):

Using default value 391

确认逻辑分区建立成功：

Command (m for help): p

Disk /dev/sdb: 3221 MB, 3221225472 bytes

255 heads, 63 sectors/track, 391 cylinders

Units = cylinders of 16065 \* 512 = 8225280 bytes

Device Boot Start End Blocks Id System

/dev/sdb1 1 100 803218+ 83 Linux

/dev/sdb2 101 125 200812+ 83 Linux

/dev/sdb3 126 391 2136645 5 Extended

/dev/sdb5 126 175 401593+ 83 Linux

/dev/sdb6 176 391 1734988+ 83 Linux

Command (m for help):

从上面的结果我们可以看到，在硬盘sdb我们建立了2个主分区（sdb1，sdb2），1个扩展分区（sdb3），2个逻辑分区（sdb5，sdb6）

注意：主分区和扩展分区的磁盘号位1-4，也就是说最多有4个主分区或者扩展分区，逻辑分区开始的磁盘号为5，因此在这个实验中试没有sdb4的。

最后对分区操作进行保存：

Command (m for help): w

The partition table has been altered!

Calling ioctl() to re-read partition table.

Syncing disks.

建立好分区之后我们还需要对分区进行格式化才能在系统中使用磁盘。

在sdb1上建立ext2分区：

[root@localhost ~]# mkfs.ext2 /dev/sdb1

mke2fs 1.39 (29-May-2006)

Filesystem label=

OS type: Linux

Block size=4096 (log=2)

Fragment size=4096 (log=2)

100576 inodes, 200804 blocks

10040 blocks (5.00%) reserved for the super user

First data block=0

Maximum filesystem blocks=209715200

7 block groups

32768 blocks per group, 32768 fragments per group

14368 inodes per group

Superblock backups stored on blocks:

32768, 98304, 163840

Writing inode tables: done

Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 32 mounts or

180 days, whichever comes first. Use tune2fs -c or -i to override.

在sdb6上建立ext3分区：

[root@localhost ~]# mkfs.ext3 /dev/sdb6

mke2fs 1.39 (29-May-2006)

Filesystem label=

OS type: Linux

Block size=4096 (log=2)

Fragment size=4096 (log=2)

217280 inodes, 433747 blocks

21687 blocks (5.00%) reserved for the super user

First data block=0

Maximum filesystem blocks=444596224

14 block groups

32768 blocks per group, 32768 fragments per group

15520 inodes per group

Superblock backups stored on blocks:

32768, 98304, 163840, 229376, 294912

Writing inode tables: done

Creating journal (8192 blocks): done

Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 32 mounts or

180 days, whichever comes first. Use tune2fs -c or -i to override.

[root@localhost ~]#

建立两个目录/oracle和/web，将新建好的两个分区挂载到系统：

[root@localhost ~]# mkdir /oracle

[root@localhost ~]# mkdir /web

[root@localhost ~]# mount /dev/sdb1 /oracle

[root@localhost ~]# mount /dev/sdb6 /web

查看分区挂载情况：

[root@localhost ~]# df -h

文件系统 容量 已用 可用 已用% 挂载点

/dev/mapper/VolGroup00-LogVol00

6.7G 2.8G 3.6G 44% /

/dev/sda1 99M 12M 82M 13% /boot

tmpfs 125M 0 125M 0% /dev/shm

/dev/sdb1 773M 808K 733M 1% /oracle

/dev/sdb6 1.7G 35M 1.6G 3% /web

如果需要每次开机自动挂载则需要修改/etc/fstab文件，加入两行配置：

[root@localhost ~]# vim /etc/fstab

/dev/VolGroup00/LogVol00 / ext3 defaults 1 1

LABEL=/boot /boot ext3 defaults 1 2

tmpfs /dev/shm tmpfs defaults 0 0

devpts /dev/pts devpts gid=5,mode=620 0 0

sysfs /sys sysfs defaults 0 0

proc /proc proc defaults 0 0

/dev/VolGroup00/LogVol01 swap swap defaults 0 0

/dev/sdb1 /oracle ext2 defaults 0 0

/dev/sdb6 /web ext3 defaults 0 0

## mkfs

用于在设备上创建Linux文件系统

补充说明

**mkfs命令** 用于在设备上（通常为硬盘）创建Linux文件系统。mkfs本身并不执行建立文件系统的工作，而是去调用相关的程序来执行。

**语法**

mkfs(选项)(参数)

**选项**

fs：指定建立文件系统时的参数；

-t<文件系统类型>：指定要建立何种文件系统；

-v：显示版本信息与详细的使用方法；

-V：显示简要的使用方法；

-c：在制做档案系统前，检查该partition是否有坏轨。

**参数**

* 文件系统：指定要创建的文件系统对应的设备文件名；
* 块数：指定文件系统的磁盘块数。

**实例**

在/dev/hda5上建一个msdos的档案系统，同时检查是否有坏轨存在，并且将过程详细列出来：

mkfs -V -t msdos -c /dev/hda5

mkfs -t ext3 /dev/sda6 //将sda6分区格式化为ext3格式

mkfs -t ext2 /dev/sda7 //将sda7分区格式化为ext2格式

## telnet

登录远程主机和管理(测试ip端口是否连通)

补充说明

**telnet命令** 用于登录远程主机，对远程主机进行管理。telnet因为采用明文传送报文，安全性不好，很多Linux服务器都不开放telnet服务，而改用更安全的ssh方式了。但仍然有很多别的系统可能采用了telnet方式来提供远程登录，因此弄清楚telnet客户端的使用方式仍是很有必要的。

**语法**

telnet(选项)(参数)

**选项**

-8：允许使用8位字符资料，包括输入与输出；

-a：尝试自动登入远端系统；

-b<主机别名>：使用别名指定远端主机名称；

-c：不读取用户专属目录里的.telnetrc文件；

-d：启动排错模式；

-e<脱离字符>：设置脱离字符；

-E：滤除脱离字符；

-f：此参数的效果和指定"-F"参数相同；

-F：使用Kerberos V5认证时，加上此参数可把本地主机的认证数据上传到远端主机；

-k<域名>：使用Kerberos认证时，加上此参数让远端主机采用指定的领域名，而非该主机的域名；

-K：不自动登入远端主机；

-l<用户名称>：指定要登入远端主机的用户名称；

-L：允许输出8位字符资料；

-n<记录文件>：指定文件记录相关信息；

-r：使用类似rlogin指令的用户界面；

-S<服务类型>：设置telnet连线所需的ip TOS信息；

-x：假设主机有支持数据加密的功能，就使用它；

-X<认证形态>：关闭指定的认证形态。

**参数**

* 远程主机：指定要登录进行管理的远程主机；
* 端口：指定TELNET协议使用的端口号。

**实例**

$ telnet 192.168.2.10

Trying 192.168.2.10...

Connected to 192.168.2.10 (192.168.2.10).

Escape character is '^]'.

localhost (Linux release 2.6.18-274.18.1.el5 #1 SMP Thu Feb 9 12:45:44 EST 2012) (1)

login: root

Password:

Login incorrect

一般情况下不允许root从远程登录，可以先用普通账号登录，然后再用su -切到root用户。

$ telnet 192.168.188.132

Trying 192.168.188.132...

telnet: connect to address 192.168.188.132: Connection refused

telnet: Unable to connect to remote host

处理这种情况方法：

1. 确认ip地址是否正确？
2. 确认ip地址对应的主机是否已经开机？
3. 如果主机已经启动，确认路由设置是否设置正确？（使用route命令查看）
4. 如果主机已经启动，确认主机上是否开启了telnet服务？（使用netstat命令查看，TCP的23端口是否有LISTEN状态的行）
5. 如果主机已经启动telnet服务，确认防火墙是否放开了23端口的访问？（使用iptables-save查看）

**启动telnet服务**

service xinetd restart

配置参数，通常的配置如下：

service telnet

{

disable = no #启用

flags = REUSE #socket可重用

socket\_type = stream #连接方式为TCP

wait = no #为每个请求启动一个进程

user = root #启动服务的用户为root

server = /usr/sbin/in.telnetd #要激活的进程

log\_on\_failure += USERID #登录失败时记录登录用户名

}

如果要配置允许登录的客户端列表，加入

only\_from = 192.168.0.2 #只允许192.168.0.2登录

如果要配置禁止登录的客户端列表，加入

no\_access = 192.168.0.{2,3,4} #禁止192.168.0.2、192.168.0.3、192.168.0.4登录

如果要设置开放时段，加入

access\_times = 9:00-12:00 13:00-17:00 # 每天只有这两个时段开放服务（我们的上班时间：P）

如果你有两个IP地址，一个是私网的IP地址如192.168.0.2，一个是公网的IP地址如218.75.74.83，如果你希望用户只能从私网来登录telnet服务，那么加入

bind = 192.168.0.2

各配置项具体的含义和语法可参考xined配置文件属性说明（man xinetd.conf）

配置端口，修改services文件：

# vi /etc/services

找到以下两句

telnet 23/tcp

telnet 23/udp

如果前面有#字符，就去掉它。telnet的默认端口是23，这个端口也是黑客端口扫描的主要对象，因此最好将这个端口修改掉，修改的方法很简单，就是将23这个数字修改掉，改成大一点的数字，比如61123。注意，1024以下的端口号是internet保留的端口号，因此最好不要用，还应该注意不要与其它服务的端口冲突。

启动服务：

service xinetd restart

## nc

用于设置路由器，是网络工具中的瑞士军刀。

## 补充说明

**nc命令** 全称**netcat**，用于设置路由器。它能通过 TCP 和 UDP 在网络中读写数据。通过与其他工具结合和重定向，你可以在脚本中以多种方式使用它。使用 netcat 命令所能完成的事情令人惊讶。

### 语法

nc [-hlnruz][-g<网关...>][-G<指向器数目>][-i<延迟秒数>][-o<输出文件>][-p<通信端口>]

[-s<来源位址>][-v...][-w<超时秒数>][主机名称][通信端口...]

### 选项

-g<网关> 设置路由器跃程通信网关，最丢哦可设置8个。

-G<指向器数目> 设置来源路由指向器，其数值为4的倍数。

-h 在线帮助。

-i<延迟秒数> 设置时间间隔，以便传送信息及扫描通信端口。

-l 使用监听模式，管控传入的资料。

-n 直接使用IP地址，而不通过域名服务器。

-o<输出文件> 指定文件名称，把往来传输的数据以16进制字码倾倒成该文件保存。

-p<通信端口> 设置本地主机使用的通信端口。

-r 乱数指定本地与远端主机的通信端口。

-s<来源位址> 设置本地主机送出数据包的IP地址。

-u 使用UDP传输协议。

-v 显示指令执行过程。

-w<超时秒数> 设置等待连线的时间。

-z 使用0输入/输出模式，只在扫描通信端口时使用。

### 实例

TCP端口扫描

[root@localhost ~]# nc -v -z -w2 192.168.0.3 1-100

192.168.0.3: inverse host lookup failed: Unknown host

(UNKNOWN) [192.168.0.3] 80 (http) open

(UNKNOWN) [192.168.0.3] 23 (telnet) open

(UNKNOWN) [192.168.0.3] 22 (ssh) open

扫描192.168.0.3 的端口 范围是 1-100 扫描UDP端口

[root@localhost ~]# nc -u -z -w2 192.168.0.1 1-1000 # 扫描192.168.0.3 的端口 范围是 1-1000

扫描指定端口

[root@localhost ~]# nc -nvv 192.168.0.1 80 # 扫描 80端口

(UNKNOWN) [192.168.0.1] 80 (?) open

y //用户输入

查看从服务器到目的地的出站端口 443 是否被防火墙阻止

nc -vz acme-v02.api.letsencrypt.org 443 -w2

# Ncat: Version 7.50 ( https://nmap.org/ncat )

# Ncat: Connected to 23.77.214.183:443.

# Ncat: 0 bytes sent, 0 bytes received in 0.07 seconds.

## ps

报告当前系统的进程状态

补充说明

**ps命令** 用于报告当前系统的进程状态。可以搭配kill指令随时中断、删除不必要的程序。ps命令是最基本同时也是非常强大的进程查看命令，使用该命令可以确定有哪些进程正在运行和运行的状态、进程是否结束、进程有没有僵死、哪些进程占用了过多的资源等等，总之大部分信息都是可以通过执行该命令得到的。

**语法**

ps(选项)

**选项**

-a：显示所有终端机下执行的程序，除了阶段作业领导者之外。

a：显示现行终端机下的所有程序，包括其他用户的程序。

-A：显示所有程序。

-c：显示CLS和PRI栏位。

c：列出程序时，显示每个程序真正的指令名称，而不包含路径，选项或常驻服务的标示。

-C<指令名称>：指定执行指令的名称，并列出该指令的程序的状况。

-d：显示所有程序，但不包括阶段作业领导者的程序。

-e：此选项的效果和指定"A"选项相同。

e：列出程序时，显示每个程序所使用的环境变量。

-f：显示UID,PPIP,C与STIME栏位。

f：用ASCII字符显示树状结构，表达程序间的相互关系。

-g<群组名称>：此选项的效果和指定"-G"选项相同，当亦能使用阶段作业领导者的名称来指定。

g：显示现行终端机下的所有程序，包括群组领导者的程序。

-G<群组识别码>：列出属于该群组的程序的状况，也可使用群组名称来指定。

h：不显示标题列。

-H：显示树状结构，表示程序间的相互关系。

-j或j：采用工作控制的格式显示程序状况。

-l或l：采用详细的格式来显示程序状况。

L：列出栏位的相关信息。

-m或m：显示所有的执行绪。

n：以数字来表示USER和WCHAN栏位。

-N：显示所有的程序，除了执行ps指令终端机下的程序之外。

-p<程序识别码>：指定程序识别码，并列出该程序的状况。

p<程序识别码>：此选项的效果和指定"-p"选项相同，只在列表格式方面稍有差异。

r：只列出现行终端机正在执行中的程序。

-s<阶段作业>：指定阶段作业的程序识别码，并列出隶属该阶段作业的程序的状况。

s：采用程序信号的格式显示程序状况。

S：列出程序时，包括已中断的子程序资料。

-t<终端机编号>：指定终端机编号，并列出属于该终端机的程序的状况。

t<终端机编号>：此选项的效果和指定"-t"选项相同，只在列表格式方面稍有差异。

-T：显示现行终端机下的所有程序。

-u<用户识别码>：此选项的效果和指定"-U"选项相同。

u：以用户为主的格式来显示程序状况。

-U<用户识别码>：列出属于该用户的程序的状况，也可使用用户名称来指定。

U<用户名称>：列出属于该用户的程序的状况。

v：采用虚拟内存的格式显示程序状况。

-V或V：显示版本信息。

-w或w：采用宽阔的格式来显示程序状况。

x：显示所有程序，不以终端机来区分。

X：采用旧式的Linux i386登陆格式显示程序状况。

-y：配合选项"-l"使用时，不显示F(flag)栏位，并以RSS栏位取代ADDR栏位　。

-<程序识别码>：此选项的效果和指定"p"选项相同。

--cols<每列字符数>：设置每列的最大字符数。

--columns<每列字符数>：此选项的效果和指定"--cols"选项相同。

--cumulative：此选项的效果和指定"S"选项相同。

--deselect：此选项的效果和指定"-N"选项相同。

--forest：此选项的效果和指定"f"选项相同。

--headers：重复显示标题列。

--help：在线帮助。

--info：显示排错信息。

--lines<显示列数>：设置显示画面的列数。

--no-headers：此选项的效果和指定"h"选项相同，只在列表格式方面稍有差异。

--group<群组名称>：此选项的效果和指定"-G"选项相同。

--Group<群组识别码>：此选项的效果和指定"-G"选项相同。

--pid<程序识别码>：此选项的效果和指定"-p"选项相同。

--rows<显示列数>：此选项的效果和指定"--lines"选项相同。

--sid<阶段作业>：此选项的效果和指定"-s"选项相同。

--tty<终端机编号>：此选项的效果和指定"-t"选项相同。

--user<用户名称>：此选项的效果和指定"-U"选项相同。

--User<用户识别码>：此选项的效果和指定"-U"选项相同。

--version：此选项的效果和指定"-V"选项相同。

--widty<每列字符数>：此选项的效果和指定"-cols"选项相同。

由于ps命令能够支持的系统类型相当的多，所以选项多的离谱！

**实例**

ps axo pid,comm,pcpu # 查看进程的PID、名称以及CPU 占用率

ps aux | sort -rnk 4 # 按内存资源的使用量对进程进行排序

ps aux | sort -nk 3 # 按 CPU 资源的使用量对进程进行排序

ps -A # 显示所有进程信息

ps -u root # 显示指定用户信息

ps -efL # 查看线程数

ps -e -o "%C : %p :%z : %a"|sort -k5 -nr # 查看进程并按内存使用大小排列

ps -ef # 显示所有进程信息，连同命令行

ps -ef | grep ssh # ps 与grep 常用组合用法，查找特定进程

ps -C nginx # 通过名字或命令搜索进程

ps aux --sort=-pcpu,+pmem # CPU或者内存进行排序,-降序，+升序

ps -f --forest -C nginx # 用树的风格显示进程的层次关系

ps -o pid,uname,comm -C nginx # 显示一个父进程的子进程

ps -e -o pid,uname=USERNAME,pcpu=CPU\_USAGE,pmem,comm # 重定义标签

ps -e -o pid,comm,etime # 显示进程运行的时间

ps -aux | grep named # 查看named进程详细信息

ps -o command -p 91730 | sed -n 2p # 通过进程id获取服务名称

将目前属于您自己这次登入的 PID 与相关信息列示出来

ps -l

# UID PID PPID F CPU PRI NI SZ RSS WCHAN S ADDR TTY TIME CMD

# 501 566 559 4006 0 31 0 4317620 228 - Ss 0 ttys001 0:00.05 /App...cOS/iTerm2 --server /usr/bin/login -fpl kenny /Ap...s/MacOS/iTerm2 --launch\_shel

# 501 592 577 4006 0 31 0 4297048 52 - S 0 ttys001 0:00.63 -zsh

* F 代表这个程序的旗标 (flag)， 4 代表使用者为 super user
* S 代表这个程序的状态 (STAT)，关于各 STAT 的意义将在内文介绍
* UID 程序被该 UID 所拥有
* PID 就是这个程序的 ID ！
* PPID 则是其上级父程序的ID
* C CPU 使用的资源百分比
* PRI 这个是 Priority (优先执行序) 的缩写，详细后面介绍
* NI 这个是 Nice 值，在下一小节我们会持续介绍
* ADDR 这个是 kernel function，指出该程序在内存的那个部分。如果是个 running的程序，一般就是 "-"
* SZ 使用掉的内存大小
* WCHAN 目前这个程序是否正在运作当中，若为 - 表示正在运作
* TTY 登入者的终端机位置
* TIME 使用掉的 CPU 时间。
* CMD 所下达的指令为何

在预设的情况下， ps 仅会列出与目前所在的 bash shell 有关的 PID 而已，所以， 当我使用 ps -l 的时候，只有三个 PID。

列出目前所有的正在内存当中的程序

ps aux

# USER PID %CPU %MEM VSZ RSS TT STAT STARTED TIME COMMAND

# kenny 6155 21.3 1.7 7969944 284912 ?? S 二03下午 199:14.14 /Appl...OS/WeChat

# kenny 559 20.4 0.8 4963740 138176 ?? S 二03下午 33:28.27 /Appl...S/iTerm2

# \_windowserver 187 18.0 0.6 7005748 95884 ?? Ss 二03下午 288:44.97 /Syst...Light.WindowServer -daemon

# kenny 1408 10.7 2.1 5838592 347348 ?? S 二03下午 138:51.63 /Appl...nts/MacOS/Google Chrome

# kenny 327 5.8 0.5 5771984 79452 ?? S 二03下午 2:51.58 /Syst...pp/Contents/MacOS/Finder

* USER：该 process 属于那个使用者账号的
* PID ：该 process 的号码
* %CPU：该 process 使用掉的 CPU 资源百分比
* %MEM：该 process 所占用的物理内存百分比
* VSZ ：该 process 使用掉的虚拟内存量 (Kbytes)
* RSS ：该 process 占用的固定的内存量 (Kbytes)
* TTY ：该 process 是在那个终端机上面运作，若与终端机无关，则显示 ?，另外， tty1-tty6 是本机上面的登入者程序，若为 pts/0 等等的，则表示为由网络连接进主机的程序。
* STAT：该程序目前的状态，主要的状态有
* R ：该程序目前正在运作，或者是可被运作
* S ：该程序目前正在睡眠当中 (可说是 idle 状态)，但可被某些讯号 (signal) 唤醒。
* T ：该程序目前正在侦测或者是停止了
* Z ：该程序应该已经终止，但是其父程序却无法正常的终止他，造成 zombie (疆尸) 程序的状态
* START：该 process 被触发启动的时间
* TIME ：该 process 实际使用 CPU 运作的时间
* COMMAND：该程序的实际指令

列出类似程序树的程序显示

ps -axjf

# USER PID PPID PGID SESS JOBC STAT TT TIME COMMAND UID C STIME TTY

# root 1 0 1 0 0 Ss ?? 10:51.90 /sbin/launchd 0 0 二03下午 ??

# root 50 1 50 0 0 Ss ?? 0:10.07 /usr/sbin/syslog 0 0 二03下午 ??

# root 51 1 51 0 0 Ss ?? 0:29.90 /usr/libexec/Use 0 0 二03下午 ??

找出与 cron 与 syslog 这两个服务有关的 PID 号码

ps aux | egrep '(cron|syslog)'

# root 50 0.0 0.0 4305532 1284 ?? Ss 二03下午 0:10.08 /usr/sbin/syslogd

# kenny 90167 0.0 0.0 4258468 184 s007 R+ 9:23下午 0:00.00 egrep (cron|syslog)

把所有进程显示出来，并输出到ps001.txt文件

## date

显示或设置系统时间与日期

## 补充说明

**date命令** 是显示或设置系统时间与日期。

很多shell脚本里面需要打印不同格式的时间或日期，以及要根据时间和日期执行操作。延时通常用于脚本执行过程中提供一段等待的时间。日期可以以多种格式去打印，也可以使用命令设置固定的格式。在类UNIX系统中，日期被存储为一个整数，其大小为自世界标准时间（UTC）1970年1月1日0时0分0秒起流逝的秒数。

### 语法

date(选项)(参数)

### 选项

-d<字符串>：显示字符串所指的日期与时间。字符串前后必须加上双引号；

-s<字符串>：根据字符串来设置日期与时间。字符串前后必须加上双引号；

-u：显示GMT；

--help：在线帮助；

--version：显示版本信息。

### 参数

<+时间日期格式>：指定显示时使用的日期时间格式。

### 日期格式字符串列表

%H # 小时，24小时制（00~23）

%I # 小时，12小时制（01~12）

%k # 小时，24小时制（0~23）

%l # 小时，12小时制（1~12）

%M # 分（00～59）

%p # 显示出上午或下午

%r # 时间，12小时制

%s # 从1970年1月1日0点到目前经历的秒数

%S # 秒（00～59）

%T # 时间（24小时制）（hh:mm:ss）

%X # 显示时间的格式（％H时％M分％S秒）

%Z # 按字母表排序的时区缩写

%a # 星期名缩写

%A # 星期名全称

%b # 月名缩写

%B # 月名全称

%c # 日期和时间

%d # 按月计的日期（01～31）

%D # 日期（mm/dd/yy）

%h # 和%b选项相同

%j # 一年的第几天（001~366）

%m # 月份（01～12）

%w # 一个星期的第几天（0代表星期天）

%W # 一年的第几个星期（00～53，星期一为第一天）

%x # 显示日期的格式（mm/dd/yy）

%y # 年份的最后两个数字（1999则是99）

%Y # 年份（比如1970、1996等）

%C # 世纪，通常为省略当前年份的后两位数字

%U # 一年中的第几周，以周日为每星期第一天

%e # 按月计的日期，添加空格，等于%\_d

### 实例

格式化输出：

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

2009-12-07

输出昨天日期：

date -d "1 day ago" +"%Y-%m-%d"

2012-11-19

2秒后输出：

date -d "2 second" +"%Y-%m-%d %H:%M.%S"

2012-11-20 14:21.31

传说中的 1234567890 秒：

date -d "1970-01-01 1234567890 seconds" +"%Y-%m-%d %H:%m:%S"

2009-02-13 23:02:30

普通转格式：

date -d "2009-12-12" +"%Y/%m/%d %H:%M.%S"

2009/12/12 00:00.00

apache格式转换：

date -d "Dec 5, 2009 12:00:37 AM" +"%Y-%m-%d %H:%M.%S"

2009-12-05 00:00.37

格式转换后时间游走：

date -d "Dec 5, 2009 12:00:37 AM 2 year ago" +"%Y-%m-%d %H:%M.%S"

2007-12-05 00:00.37

加减操作：

date +%Y%m%d # 显示前天年月日

date -d "+1 day" +%Y%m%d # 显示前一天的日期

date -d "-1 day" +%Y%m%d # 显示后一天的日期

date -d "-1 month" +%Y%m%d # 显示上一月的日期

date -d "+1 month" +%Y%m%d # 显示下一月的日期

date -d "-1 year" +%Y%m%d # 显示前一年的日期

date -d "+1 year" +%Y%m%d # 显示下一年的日期

设定时间：

date -s # 设置当前时间，只有root权限才能设置，其他只能查看

date -s 20120523 # 设置成20120523，这样会把具体时间设置成空00:00:00

date -s 01:01:01 # 设置具体时间，不会对日期做更改

date -s "01:01:01 2012-05-23" # 这样可以设置全部时间

date -s "01:01:01 20120523" # 这样可以设置全部时间

date -s "2012-05-23 01:01:01" # 这样可以设置全部时间

date -s "20120523 01:01:01" # 这样可以设置全部时间

有时需要检查一组命令花费的时间，举例：

#!/bin/bash

start=$(date +%s)

nmap href.lu &> /dev/null

end=$(date +%s)

difference=$(( end - start ))

echo $difference seconds.

## top

显示或管理执行中的程序

补充说明

**top命令** 可以实时动态地查看系统的整体运行情况，是一个综合了多方信息监测系统性能和运行信息的实用工具。通过top命令所提供的互动式界面，用热键可以管理。

**语法**

top(选项)

**选项**

-b：以批处理模式操作；

-c：显示完整的治命令；

-d：屏幕刷新间隔时间；

-I：忽略失效过程；

-s：保密模式；

-S：累积模式；

-i<时间>：设置间隔时间；

-u<用户名>：指定用户名；

-p<进程号>：指定进程；

-n<次数>：循环显示的次数。

**top交互命令**

在top命令执行过程中可以使用的一些交互命令。这些命令都是单字母的，如果在命令行中使用了-s选项， 其中一些命令可能会被屏蔽。

h：显示帮助画面，给出一些简短的命令总结说明；

k：终止一个进程；

i：忽略闲置和僵死进程，这是一个开关式命令；

q：退出程序；

r：重新安排一个进程的优先级别；

S：切换到累计模式；

s：改变两次刷新之间的延迟时间（单位为s），如果有小数，就换算成ms。输入0值则系统将不断刷新，默认值是5s；

f或者F：从当前显示中添加或者删除项目；

o或者O：改变显示项目的顺序；

l：切换显示平均负载和启动时间信息；

m：切换显示内存信息；

t：切换显示进程和CPU状态信息；

c：切换显示命令名称和完整命令行；

M：根据驻留内存大小进行排序；

P：根据CPU使用百分比大小进行排序；

T：根据时间/累计时间进行排序；

w：将当前设置写入~/.toprc文件中。

**实例**

top - 09:44:56 up 16 days, 21:23, 1 user, load average: 9.59, 4.75, 1.92

Tasks: 145 total, 2 running, 143 sleeping, 0 stopped, 0 zombie

Cpu(s): 99.8%us, 0.1%sy, 0.0%ni, 0.2%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st

Mem: 4147888k total, 2493092k used, 1654796k free, 158188k buffers

Swap: 5144568k total, 56k used, 5144512k free, 2013180k cached

**解释：**

* top - 09:44:56[当前系统时间],
* 16 days[系统已经运行了16天],
* 1 user[个用户当前登录],
* load average: 9.59, 4.75, 1.92[系统负载，即任务队列的平均长度]
* Tasks: 145 total[总进程数],
* 2 running[正在运行的进程数],
* 143 sleeping[睡眠的进程数],
* 0 stopped[停止的进程数],
* 0 zombie[冻结进程数],
* Cpu(s): 99.8%us[用户空间占用CPU百分比],
* 0.1%sy[内核空间占用CPU百分比],
* 0.0%ni[用户进程空间内改变过优先级的进程占用CPU百分比],
* 0.2%id[空闲CPU百分比], 0.0%wa[等待输入输出的CPU时间百分比],
* 0.0%hi[],
* 0.0%st[],
* Mem: 4147888k total[物理内存总量],
* 2493092k used[使用的物理内存总量],
* 1654796k free[空闲内存总量],
* 158188k buffers[用作内核缓存的内存量]
* Swap:  5144568k total[交换区总量],
* 56k used[使用的交换区总量],
* 5144512k free[空闲交换区总量],
* 2013180k cached[缓冲的交换区总量],

## rpm

RPM软件包的管理工具

补充说明

**rpm命令** 是RPM软件包的管理工具。rpm原本是Red Hat Linux发行版专门用来管理Linux各项套件的程序，由于它遵循GPL规则且功能强大方便，因而广受欢迎。逐渐受到其他发行版的采用。RPM套件管理方式的出现，让Linux易于安装，升级，间接提升了Linux的适用度。

**语法**

rpm(选项)(参数)

**选项**

-a：查询所有套件；

-b<完成阶段><套件档>+或-t <完成阶段><套件档>+：设置包装套件的完成阶段，并指定套件档的文件名称；

-c：只列出组态配置文件，本参数需配合"-l"参数使用；

-d：只列出文本文件，本参数需配合"-l"参数使用；

-e<套件档>或--erase<套件档>：删除指定的套件；

-f<文件>+：查询拥有指定文件的套件；

-h或--hash：套件安装时列出标记；

-i：显示套件的相关信息；

-i<套件档>或--install<套件档>：安装指定的套件档；

-l：显示套件的文件列表；

-p<套件档>+：查询指定的RPM套件档；

-q：使用询问模式，当遇到任何问题时，rpm指令会先询问用户；

-R：显示套件的关联性信息；

-s：显示文件状态，本参数需配合"-l"参数使用；

-U<套件档>或--upgrade<套件档>：升级指定的套件档；

-v：显示指令执行过程；

-vv：详细显示指令执行过程，便于排错。

**参数**

软件包：指定要操纵的rpm软件包。

**实例**

**如何安装rpm软件包**

rpm软件包的安装可以使用程序rpm来完成。执行下面的命令：

rpm -ivh your-package.rpm

其中your-package.rpm是你要安装的rpm包的文件名，一般置于当前目录下。

安装过程中可能出现下面的警告或者提示：

... conflict with ...

可能是要安装的包里有一些文件可能会覆盖现有的文件，缺省时这样的情况下是无法正确安装的可以用rpm --force -i强制安装即可

... is needed by ...

... is not installed ...

此包需要的一些软件你没有安装可以用rpm --nodeps -i来忽略此信息，也就是说rpm -i --force --nodeps可以忽略所有依赖关系和文件问题，什么包都能安装上，但这种强制安装的软件包不能保证完全发挥功能。

**如何安装.src.rpm软件包**

有些软件包是以.src.rpm结尾的，这类软件包是包含了源代码的rpm包，在安装时需要进行编译。这类软件包有两种安装方法：

方法一：

rpm -i your-package.src.rpm

cd /usr/src/redhat/SPECS

rpmbuild -bp your-package.specs #一个和你的软件包同名的specs文件

cd /usr/src/redhat/BUILD/your-package/ #一个和你的软件包同名的目录

./configure #这一步和编译普通的源码软件一样，可以加上参数

make

make install

方法二：

rpm -i you-package.src.rpm

cd /usr/src/redhat/SPECS

前两步和方法一相同

rpmbuild -bb your-package.specs #一个和你的软件包同名的specs文件

这时在/usr/src/redhat/RPM/i386/（根据具体包的不同，也可能是i686,noarch等等）在这个目录下，有一个新的rpm包，这个是编译好的二进制文件。

执行rpm -i new-package.rpm即可安装完成。

**如何卸载rpm软件包**

使用命令rpm -e包名，包名可以包含版本号等信息，但是不可以有后缀.rpm，比如卸载软件包proftpd-1.2.8-1，可以使用下列格式：

rpm -e proftpd-1.2.8-1

rpm -e proftpd-1.2.8

rpm -e proftpd-

rpm -e proftpd

不可以是下列格式：

rpm -e proftpd-1.2.8-1.i386.rpm

rpm -e proftpd-1.2.8-1.i386

rpm -e proftpd-1.2

rpm -e proftpd-1

有时会出现一些错误或者警告：

... is needed by ...

这说明这个软件被其他软件需要，不能随便卸载，可以用rpm -e --nodeps强制卸载

**如何不安装但是获取rpm包中的文件**

使用工具rpm2cpio和cpio

rpm2cpio xxx.rpm | cpio -vi

rpm2cpio xxx.rpm | cpio -idmv

rpm2cpio xxx.rpm | cpio --extract --make-directories

参数i和extract相同，表示提取文件。v表示指示执行进程，d和make-directory相同，表示根据包中文件原来的路径建立目录，m表示保持文件的更新时间。

**如何查看与rpm包相关的文件和其他信息**

下面所有的例子都假设使用软件包mysql-3.23.54a-11

1、我的系统中安装了那些rpm软件包。

rpm -qa 讲列出所有安装过的包

如果要查找所有安装过的包含某个字符串sql的软件包

rpm -qa | grep sql

2、如何获得某个软件包的文件全名。

rpm -q mysql

可以获得系统中安装的mysql软件包全名，从中可以获得当前软件包的版本等信息。这个例子中可以得到信息mysql-3.23.54a-11

3、一个rpm包中的文件安装到那里去了？

rpm -ql 包名

注意这里的是不包括.rpm后缀的软件包的名称，也就是说只能用mysql或者mysql-3.23.54a-11而不是mysql-3.23.54a-11.rpm。如果只是想知道可执行程序放到那里去了，也可以用which，比如：

which mysql

4、一个rpm包中包含那些文件。

* 一个没有安装过的软件包，使用rpm -qlp \*\*\*\* .rpm
* 一个已经安装过的软件包，还可以使用rpm -ql \*\*\*\* .rpm

5、如何获取关于一个软件包的版本，用途等相关信息？

* 一个没有安装过的软件包，使用rpm -qip \*\*\*\* .rpm
* 一个已经安装过的软件包，还可以使用rpm -qi \*\*\*\* .rpm

6、某个程序是哪个软件包安装的，或者哪个软件包包含这个程序。

rpm -qf `which 程序名` #返回软件包的全名

rpm -qif `which 程序名` #返回软件包的有关信息

rpm -qlf `which 程序名` #返回软件包的文件列表

注意，这里不是引号，而是，就是键盘左上角的那个键。也可以使用rpm -qilf`，同时输出软件包信息和文件列表。

7、某个文件是哪个软件包安装的，或者哪个软件包包含这个文件。

注意，前一个问题中的方法，只适用与可执行的程序，而下面的方法，不仅可以用于可执行程序，也可以用于普通的任何文件。前提是知道这个文件名。首先获得这个程序的完整路径，可以用whereis或者which，然后使用rpm -qf例如：

whereis ftptop

ftptop: /usr/bin/ftptop /usr/share/man/man1/ftptop.1.gz

rpm -qf /usr/bin/ftptop

proftpd-1.2.8-1

rpm -qf /usr/share/doc/proftpd-1.2.8/rfc/rfc0959.txt

proftpd-1.2.8-1

更多实例

库依赖: <http://rpmfind.net/>

源码包 -> 编译 -> 二进制包(rpm 包 / 系统默认包)

rpm 命名规则: 软件(软件名, 软件版本) + 系统(os 版本, os 位数) rpm 校验: SM5DLUGT -> size modified(类型/权限) md5 device L(文件路径) user group time(modified time)

yum: 解决 rpm 依赖的问题

# rpm

mysql57-community-release-el6-8.noarch.rpm # 一个 rpm 包的例子

/var/lib/rpm/ # 包全名 -> 包名 的数据库

rpm -Uivh --dodeps xxx # upgrade install verbose hash

rpm -qilpfa|grep xxx # query info list(rpm包安装后的文件位置) package(rpm 包) file(文件属于哪个rpm文件) all

rpm -e # erase

rpm -V # verify

rpm2cpio | cpio -idv

# rpm 默认安装位置

/etc/ 配置文件

/usr/bin/ 可执行文件

/urs/lib/ 程序使用的函数库

/usr/share/doc/ 使用手册

/usr/share/man/ manual

## dmesg

显示Linux系统启动信息

## 补充说明

**dmesg命令** 被用于检查和控制内核的环形缓冲区。kernel会将开机信息存储在ring buffer中。您若是开机时来不及查看信息，可利用dmesg来查看。开机信息保存在/var/log/dmesg文件里。

### 语法

dmesg(选项)

### 选项

-c：显示信息后，清除ring buffer中的内容；

-s<缓冲区大小>：预设置为8196，刚好等于ring buffer的大小；

-n：设置记录信息的层级。

### 实例

[root@localhost ~]# dmesg | head

Linux version 2.6.18-348.6.1.el5 (mockbuild@builder17.centos.org) (gcc version 4.1.2 20080704 (Red Hat 4.1.2-54)) #1 SMP Tue May 21 15:34:22 EDT 2013

BIOS-provided physical RAM map:

BIOS-e820: 0000000000010000 - 000000000009f400 (usable)

BIOS-e820: 000000000009f400 - 00000000000a0000 (reserved)

BIOS-e820: 00000000000f0000 - 0000000000100000 (reserved)

BIOS-e820: 0000000000100000 - 000000007f590000 (usable)

BIOS-e820: 000000007f590000 - 000000007f5e3000 (ACPI NVS)

BIOS-e820: 000000007f5e3000 - 000000007f5f0000 (ACPI data)

BIOS-e820: 000000007f5f0000 - 000000007f600000 (reserved)

BIOS-e820: 00000000e0000000 - 00000000e8000000 (reserved)

查看硬盘基础信息

dmesg | grep sda

[ 2.442555] sd 0:0:0:0: [sda] 488281250 512-byte logical blocks: (250 GB/232 GiB)

[ 2.442590] sd 0:0:0:0: [sda] Write Protect is off

[ 2.442592] sd 0:0:0:0: [sda] Mode Sense: 00 3a 00 00

[ 2.442607] sd 0:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA

[ 2.447533] sda: sda1

[ 2.448503] sd 0:0:0:0: [sda] Attached SCSI disk

## tar

Linux下的归档使用工具，用来打包和备份。

## 补充说明

**tar命令** 可以为linux的文件和目录创建档案。利用tar，可以为某一特定文件创建档案（备份文件），也可以在档案中改变文件，或者向档案中加入新的文件。tar最初被用来在磁带上创建档案，现在，用户可以在任何设备上创建档案。利用tar命令，可以把一大堆的文件和目录全部打包成一个文件，这对于备份文件或将几个文件组合成为一个文件以便于网络传输是非常有用的。

首先要弄清两个概念：打包和压缩。打包是指将一大堆文件或目录变成一个总的文件；压缩则是将一个大的文件通过一些压缩算法变成一个小文件。

为什么要区分这两个概念呢？这源于Linux中很多压缩程序只能针对一个文件进行压缩，这样当你想要压缩一大堆文件时，你得先将这一大堆文件先打成一个包（tar命令），然后再用压缩程序进行压缩（gzip bzip2命令）。

### 语法

tar(选项)(参数)

### 选项

-A或--catenate：新增文件到以存在的备份文件；

-B：设置区块大小；

-c或--create：建立新的备份文件；

-C <目录>：这个选项用在解压缩，若要在特定目录解压缩，可以使用这个选项。

-d：记录文件的差别；

-x或--extract或--get：从备份文件中还原文件；

-t或--list：列出备份文件的内容；

-z或--gzip或--ungzip：通过gzip指令处理备份文件；

-Z或--compress或--uncompress：通过compress指令处理备份文件；

-f<备份文件>或--file=<备份文件>：指定备份文件；

-v或--verbose：显示指令执行过程；

-r：添加文件到已经压缩的文件；

-u：添加改变了和现有的文件到已经存在的压缩文件；

-j：支持bzip2解压文件；

-v：显示操作过程；

-l：文件系统边界设置；

-k：保留原有文件不覆盖；

-m：保留文件不被覆盖；

-w：确认压缩文件的正确性；

-p或--same-permissions：用原来的文件权限还原文件；

-P或--absolute-names：文件名使用绝对名称，不移除文件名称前的“/”号；

-N <日期格式> 或 --newer=<日期时间>：只将较指定日期更新的文件保存到备份文件里；

--exclude=<范本样式>：排除符合范本样式的文件。

### 参数

文件或目录：指定要打包的文件或目录列表。

### 实例

- z：有gzip属性的

- j：有bz2属性的

- Z：有compress属性的

- v：显示所有过程

- O：将文件解开到标准输出

下面的参数-f是必须的

-f: 使用档案名字，切记，这个参数是最后一个参数，后面只能接档案名。

tar -cf all.tar \*.jpg

# 这条命令是将所有.jpg的文件打成一个名为all.tar的包。-c是表示产生新的包，-f指定包的文件名。

tar -rf all.tar \*.gif

# 这条命令是将所有.gif的文件增加到all.tar的包里面去。-r是表示增加文件的意思。

tar -uf all.tar logo.gif

# 这条命令是更新原来tar包all.tar中logo.gif文件，-u是表示更新文件的意思。

tar -tf all.tar

# 这条命令是列出all.tar包中所有文件，-t是列出文件的意思

tar -cfv archive.tar foo bar # 从文件foo和bar创建archive.tar。

tar -tvf archive.tar # 详细列出archive.tar中的所有文件。

tar -xf archive.tar # 从archive.tar提取所有文件。

#### zip格式

压缩： zip -r [目标文件名].zip [原文件/目录名]  
解压： unzip [原文件名].zip  
注：-r参数代表递归

#### tar格式（该格式仅仅打包，不压缩）

打包：tar -cvf [目标文件名].tar [原文件名/目录名]  
解包：tar -xvf [原文件名].tar  
注：c参数代表create（创建），x参数代表extract（解包），v参数代表verbose（详细信息），f参数代表filename（文件名），所以f后必须接文件名。

#### tar.gz格式

方式一：利用前面已经打包好的tar文件，直接用压缩命令。

压缩：gzip [原文件名].tar  
解压：gunzip [原文件名].tar.gz

方式二：一次性打包并压缩、解压并解包

打包并压缩： tar -zcvf [目标文件名].tar.gz [原文件名/目录名]  
解压并解包： tar -zxvf [原文件名].tar.gz  
注：z代表用gzip算法来压缩/解压。

#### tar.bz2格式

方式一：利用已经打包好的tar文件，直接执行压缩命令：

压缩：bzip2 [原文件名].tar  
解压：bunzip2 [原文件名].tar.bz2  
方式二：一次性打包并压缩、解压并解包

打包并压缩： tar -jcvf [目标文件名].tar.bz2 [原文件名/目录名]  
解压并解包： tar -jxvf [原文件名].tar.bz2  
注：小写j代表用bzip2算法来压缩/解压。

#### tar.xz格式

方式一：利用已经打包好的tar文件，直接用压缩命令：

压缩：xz [原文件名].tar  
解压：unxz [原文件名].tar.xz  
方式二：一次性打包并压缩、解压并解包

打包并压缩： tar -Jcvf [目标文件名].tar.xz [原文件名/目录名]  
解压并解包： tar -Jxvf [原文件名].tar.xz  
注：大写J代表用xz算法来压缩/解压。

#### tar.Z格式（已过时）

方式一：利用已经打包好的tar文件，直接用压缩命令：

压缩：compress [原文件名].tar  
解压：uncompress [原文件名].tar.Z  
方式二：一次性打包并压缩、解压并解包

打包并压缩： tar -Zcvf [目标文件名].tar.Z [原文件名/目录名]  
解压并解包： tar -Zxvf [原文件名].tar.Z  
注：大写Z代表用ncompress算法来压缩/解压。另，ncompress是早期Unix系统的压缩格式，但由于ncompress的压缩率太低，现已过时。

#### jar格式

压缩：jar -cvf [目标文件名].jar [原文件名/目录名]  
解压：jar -xvf [原文件名].jar

注：如果是打包的是Java类库，并且该类库中存在主类，那么需要写一个META-INF/MANIFEST.MF配置文件，内容如下：

Manifest-Version: 1.0

Created-By: 1.6.0\_27 (Sun Microsystems Inc.)

Main-class: the\_name\_of\_the\_main\_class\_should\_be\_put\_here

然后用如下命令打包：

jar -cvfm [目标文件名].jar META-INF/MANIFEST.MF [原文件名/目录名]  
这样以后就能用“java -jar [文件名].jar”命令直接运行主类中的public static void main方法了。

#### 7z格式

压缩：7z a [目标文件名].7z [原文件名/目录名]  
解压：7z x [原文件名].7z  
注：这个7z解压命令支持rar格式，即：

7z x [原文件名].rar

#### 其它例子

**将文件全部打包成tar包** ：

tar -cvf log.tar log2012.log 仅打包，不压缩！

tar -zcvf log.tar.gz log2012.log 打包后，以 gzip 压缩

tar -jcvf log.tar.bz2 log2012.log 打包后，以 bzip2 压缩

在选项f之后的文件档名是自己取的，我们习惯上都用 .tar 来作为辨识。 如果加z选项，则以.tar.gz或.tgz来代表gzip压缩过的tar包；如果加j选项，则以.tar.bz2来作为tar包名。

**解压目录**

去掉第一层目录结构，要出除第二层，--strip-components 2

tar -xvf portal-web-v2.0.0.tar --strip-components 1 -C 指定目录

**查阅上述tar包内有哪些文件** ：

tar -ztvf log.tar.gz

由于我们使用 gzip 压缩的log.tar.gz，所以要查阅log.tar.gz包内的文件时，就得要加上z这个选项了。

**将tar包解压缩** ：

tar -zxvf /opt/soft/test/log.tar.gz

在预设的情况下，我们可以将压缩档在任何地方解开的

**只将tar内的部分文件解压出来** ：

tar -zxvf /opt/soft/test/log30.tar.gz log2013.log

我可以透过tar -ztvf来查阅 tar 包内的文件名称，如果单只要一个文件，就可以透过这个方式来解压部分文件！

**文件备份下来，并且保存其权限** ：

tar -zcvpf log31.tar.gz log2014.log log2015.log log2016.log

这个-p的属性是很重要的，尤其是当您要保留原本文件的属性时。

**在文件夹当中，比某个日期新的文件才备份** ：

tar -N "2012/11/13" -zcvf log17.tar.gz test

**备份文件夹内容是排除部分文件：**

tar --exclude scf/service -zcvf scf.tar.gz scf/\*

**其实最简单的使用 tar 就只要记忆底下的方式即可：**

压　缩：tar -jcv -f filename.tar.bz2 要被压缩的文件或目录名称

查　询：tar -jtv -f filename.tar.bz2

解压缩：tar -jxv -f filename.tar.bz2 -C 欲解压缩的目录

## zip

可以用来解压缩文件

补充说明

**zip命令** 可以用来解压缩文件，或者对文件进行打包操作。zip是个使用广泛的压缩程序，文件经它压缩后会另外产生具有“.zip”扩展名的压缩文件。

**语法**

zip(选项)(参数)

**选项**

-A：调整可执行的自动解压缩文件；

-b<工作目录>：指定暂时存放文件的目录；

-c：替每个被压缩的文件加上注释；

-d：从压缩文件内删除指定的文件；

-D：压缩文件内不建立目录名称；

-f：此参数的效果和指定“-u”参数类似，但不仅更新既有文件，如果某些文件原本不存在于压缩文件内，使用本参数会一并将其加入压缩文件中；

-F：尝试修复已损坏的压缩文件；

-g：将文件压缩后附加在已有的压缩文件之后，而非另行建立新的压缩文件；

-h：在线帮助；

-i<范本样式>：只压缩符合条件的文件；

-j：只保存文件名称及其内容，而不存放任何目录名称；

-J：删除压缩文件前面不必要的数据；

-k：使用MS-DOS兼容格式的文件名称；

-l：压缩文件时，把LF字符置换成LF+CR字符；

-ll：压缩文件时，把LF+cp字符置换成LF字符；

-L：显示版权信息；

-m：将文件压缩并加入压缩文件后，删除原始文件，即把文件移到压缩文件中；

-n<字尾字符串>：不压缩具有特定字尾字符串的文件；

-o：以压缩文件内拥有最新更改时间的文件为准，将压缩文件的更改时间设成和该文件相同；

-q：不显示指令执行过程；

-r：递归处理，将指定目录下的所有文件和子目录一并处理；

-S：包含系统和隐藏文件；

-t<日期时间>：把压缩文件的日期设成指定的日期；

-T：检查备份文件内的每个文件是否正确无误；

-u：更换较新的文件到压缩文件内；

-v：显示指令执行过程或显示版本信息；

-V：保存VMS操作系统的文件属性；

-w：在文件名称里假如版本编号，本参数仅在VMS操作系统下有效；

-x<范本样式>：压缩时排除符合条件的文件；

-X：不保存额外的文件属性；

-y：直接保存符号连接，而非该链接所指向的文件，本参数仅在UNIX之类的系统下有效；

-z：替压缩文件加上注释；

-$：保存第一个被压缩文件所在磁盘的卷册名称；

-<压缩效率>：压缩效率是一个介于1~9的数值。

**参数**

* zip压缩包：指定要创建的zip压缩包；
* 文件列表：指定要压缩的文件列表。

**实例**

将/home/Blinux/html/这个目录下所有文件和文件夹打包为当前目录下的html.zip：

zip -q -r html.zip /home/Blinux/html

上面的命令操作是将绝对地址的文件及文件夹进行压缩，以下给出压缩相对路径目录，比如目前在Bliux这个目录下，执行以下操作可以达到以上同样的效果：

zip -q -r html.zip html

比如现在我的html目录下，我操作的zip压缩命令是：

zip -q -r html.zip \*

压缩效率选择:

zip -9 # 1-9 faster->better

**问题解决**

CentOS7中命令找不到

-Bash: Unzip: Command Not Found

解决方法

yum install -y unzip zip

## gzip

用来压缩文件

## 补充说明

**gzip命令** 用来压缩文件。gzip是个使用广泛的压缩程序，文件经它压缩过后，其名称后面会多处“.gz”扩展名。

gzip是在Linux系统中经常使用的一个对文件进行压缩和解压缩的命令，既方便又好用。gzip不仅可以用来压缩大的、较少使用的文件以节省磁盘空间，还可以和tar命令一起构成Linux操作系统中比较流行的压缩文件格式。据统计，gzip命令对文本文件有60%～70%的压缩率。减少文件大小有两个明显的好处，一是可以减少存储空间，二是通过网络传输文件时，可以减少传输的时间。

### 语法

gzip(选项)(参数)

### 选项

-a或——ascii：使用ASCII文字模式；

-d或--decompress或----uncompress：解开压缩文件；

-f或——force：强行压缩文件。不理会文件名称或硬连接是否存在以及该文件是否为符号连接；

-h或——help：在线帮助；

-l或——list：列出压缩文件的相关信息；

-L或——license：显示版本与版权信息；

-n或--no-name：压缩文件时，不保存原来的文件名称及时间戳记；

-N或——name：压缩文件时，保存原来的文件名称及时间戳记；

-q或——quiet：不显示警告信息；

-r或——recursive：递归处理，将指定目录下的所有文件及子目录一并处理；

-S或<压缩字尾字符串>或----suffix<压缩字尾字符串>：更改压缩字尾字符串；

-t或——test：测试压缩文件是否正确无误；

-v或——verbose：显示指令执行过程；

-V或——version：显示版本信息；

-<压缩效率>：压缩效率是一个介于1~9的数值，预设值为“6”，指定愈大的数值，压缩效率就会愈高；

--best：此参数的效果和指定“-9”参数相同；

--fast：此参数的效果和指定“-1”参数相同。

-num 用指定的数字num调整压缩的速度，-1或--fast表示最快压缩方法（低压缩比），-9或--best表示最慢压缩方法（高压缩比）。系统缺省值为6。

### 参数

文件列表：指定要压缩的文件列表。

### 实例

把test6目录下的每个文件压缩成.gz文件

gzip \*

把上例中每个压缩的文件解压，并列出详细的信息

gzip -dv \*

详细显示例1中每个压缩的文件的信息，并不解压

gzip -l \*

压缩一个tar备份文件，此时压缩文件的扩展名为.tar.gz

gzip -r log.tar

递归的压缩目录

gzip -rv test6

这样，所有test下面的文件都变成了\*.gz，目录依然存在只是目录里面的文件相应变成了\*.gz.这就是压缩，和打包不同。因为是对目录操作，所以需要加上-r选项，这样也可以对子目录进行递归了。

递归地解压目录

gzip -dr test6

## ip

网络配置工具

## 补充说明

**ip命令** 用来显示或操纵Linux主机的路由、网络设备、策略路由和隧道，是Linux下较新的功能强大的网络配置工具。

### 语法

ip(选项)(参数)

Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }

ip [ -force ] -batch filename

### 选项

OBJECT := { link | address | addrlabel | route | rule | neigh | ntable |

tunnel | tuntap | maddress | mroute | mrule | monitor | xfrm |

netns | l2tp | macsec | tcp\_metrics | token }

-V：显示指令版本信息；

-s：输出更详细的信息；

-f：强制使用指定的协议族；

-4：指定使用的网络层协议是IPv4协议；

-6：指定使用的网络层协议是IPv6协议；

-0：输出信息每条记录输出一行，即使内容较多也不换行显示；

-r：显示主机时，不使用IP地址，而使用主机的域名。

### 参数

OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |

-h[uman-readable] | -iec |

-f[amily] { inet | inet6 | ipx | dnet | bridge | link } |

-4 | -6 | -I | -D | -B | -0 |

-l[oops] { maximum-addr-flush-attempts } |

-o[neline] | -t[imestamp] | -ts[hort] | -b[atch] [filename] |

-rc[vbuf] [size] | -n[etns] name | -a[ll] }

网络对象：指定要管理的网络对象；

具体操作：对指定的网络对象完成具体操作；

help：显示网络对象支持的操作命令的帮助信息。

### 实例

ip link show # 显示网络接口信息

ip link set eth0 up # 开启网卡

ip link set eth0 down # 关闭网卡

ip link set eth0 promisc on # 开启网卡的混合模式

ip link set eth0 promisc offi # 关闭网卡的混个模式

ip link set eth0 txqueuelen 1200 # 设置网卡队列长度

ip link set eth0 mtu 1400 # 设置网卡最大传输单元

ip addr show # 显示网卡IP信息

ip addr add 192.168.0.1/24 dev eth0 # 设置eth0网卡IP地址192.168.0.1

ip addr del 192.168.0.1/24 dev eth0 # 删除eth0网卡IP地址

ip route show # 显示系统路由

ip route add default via 192.168.1.254 # 设置系统默认路由

ip route list # 查看路由信息

ip route add 192.168.4.0/24 via 192.168.0.254 dev eth0 # 设置192.168.4.0网段的网关为192.168.0.254,数据走eth0接口

ip route add default via 192.168.0.254 dev eth0 # 设置默认网关为192.168.0.254

ip route del 192.168.4.0/24 # 删除192.168.4.0网段的网关

ip route del default # 删除默认路由

ip route delete 192.168.1.0/24 dev eth0 # 删除路由

**用ip命令显示网络设备的运行状态**

[root@localhost ~]# ip link list

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 16436 qdisc noqueue

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast qlen 1000

link/ether 00:16:3e:00:1e:51 brd ff:ff:ff:ff:ff:ff

3: eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast qlen 1000

link/ether 00:16:3e:00:1e:52 brd ff:ff:ff:ff:ff:ff

**显示更加详细的设备信息**

[root@localhost ~]# ip -s link list

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 16436 qdisc noqueue

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

RX: bytes packets errors dropped overrun mcast

5082831 56145 0 0 0 0

TX: bytes packets errors dropped carrier collsns

5082831 56145 0 0 0 0

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast qlen 1000

link/ether 00:16:3e:00:1e:51 brd ff:ff:ff:ff:ff:ff

RX: bytes packets errors dropped overrun mcast

3641655380 62027099 0 0 0 0

TX: bytes packets errors dropped carrier collsns

6155236 89160 0 0 0 0

3: eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast qlen 1000

link/ether 00:16:3e:00:1e:52 brd ff:ff:ff:ff:ff:ff

RX: bytes packets errors dropped overrun mcast

2562136822 488237847 0 0 0 0

TX: bytes packets errors dropped carrier collsns

3486617396 9691081 0 0 0 0

**显示核心路由表**

[root@localhost ~]# ip route list

112.124.12.0/22 dev eth1 proto kernel scope link src 112.124.15.130

10.160.0.0/20 dev eth0 proto kernel scope link src 10.160.7.81

192.168.0.0/16 via 10.160.15.247 dev eth0

172.16.0.0/12 via 10.160.15.247 dev eth0

10.0.0.0/8 via 10.160.15.247 dev eth0

default via 112.124.15.247 dev eth1

**显示邻居表**

[root@localhost ~]# ip neigh list

112.124.15.247 dev eth1 lladdr 00:00:0c:9f:f3:88 REACHABLE

10.160.15.247 dev eth0 lladdr 00:00:0c:9f:f2:c0 STALE

**获取主机所有网络接口**

ip link | grep ^[0-9] | awk -F: '{print $2}'

## ss

比 netstat 好用的socket统计信息，iproute2 包附带的另一个工具，允许你查询 socket 的有关统计信息。

## 补充说明

**ss命令** 用来显示处于活动状态的套接字信息。ss命令可以用来获取socket统计信息，它可以显示和netstat类似的内容。但ss的优势在于它能够显示更多更详细的有关TCP和连接状态的信息，而且比netstat更快速更高效。

当服务器的socket连接数量变得非常大时，无论是使用netstat命令还是直接cat /proc/net/tcp，执行速度都会很慢。可能你不会有切身的感受，但请相信我，当服务器维持的连接达到上万个的时候，使用netstat等于浪费 生命，而用ss才是节省时间。

天下武功唯快不破。ss快的秘诀在于，它利用到了TCP协议栈中tcp\_diag。tcp\_diag是一个用于分析统计的模块，可以获得Linux 内核中第一手的信息，这就确保了ss的快捷高效。当然，如果你的系统中没有tcp\_diag，ss也可以正常运行，只是效率会变得稍慢。

### 语法

ss [参数]

ss [参数] [过滤]

### 选项

-h, --help 帮助信息

-V, --version 程序版本信息

-n, --numeric 不解析服务名称

-r, --resolve 解析主机名

-a, --all 显示所有套接字（sockets）

-l, --listening 显示监听状态的套接字（sockets）

-o, --options 显示计时器信息

-e, --extended 显示详细的套接字（sockets）信息

-m, --memory 显示套接字（socket）的内存使用情况

-p, --processes 显示使用套接字（socket）的进程

-i, --info 显示 TCP内部信息

-s, --summary 显示套接字（socket）使用概况

-4, --ipv4 仅显示IPv4的套接字（sockets）

-6, --ipv6 仅显示IPv6的套接字（sockets）

-0, --packet 显示 PACKET 套接字（socket）

-t, --tcp 仅显示 TCP套接字（sockets）

-u, --udp 仅显示 UCP套接字（sockets）

-d, --dccp 仅显示 DCCP套接字（sockets）

-w, --raw 仅显示 RAW套接字（sockets）

-x, --unix 仅显示 Unix套接字（sockets）

-f, --family=FAMILY 显示 FAMILY类型的套接字（sockets），FAMILY可选，支持 unix, inet, inet6, link, netlink

-A, --query=QUERY, --socket=QUERY

QUERY := {all|inet|tcp|udp|raw|unix|packet|netlink}[,QUERY]

-D, --diag=FILE 将原始TCP套接字（sockets）信息转储到文件

-F, --filter=FILE 从文件中都去过滤器信息

FILTER := [ state TCP-STATE ] [ EXPRESSION ]

### 实例

ss -t -a # 显示TCP连接

ss -s # 显示 Sockets 摘要

ss -l # 列出所有打开的网络连接端口

ss -pl # 查看进程使用的socket

ss -lp | grep 3306 # 找出打开套接字/端口应用程序

ss -u -a 显示所有UDP Sockets

ss -o state established '( dport = :smtp or sport = :smtp )' # 显示所有状态为established的SMTP连接

ss -o state established '( dport = :http or sport = :http )' # 显示所有状态为Established的HTTP连接

ss -o state fin-wait-1 '( sport = :http or sport = :https )' dst 193.233.7/24 # 列举出处于 FIN-WAIT-1状态的源端口为 80或者 443，目标网络为 193.233.7/24所有 tcp套接字

# ss 和 netstat 效率对比

time netstat -at

time ss

# 匹配远程地址和端口号

# ss dst ADDRESS\_PATTERN

ss dst 192.168.1.5

ss dst 192.168.119.113:http

ss dst 192.168.119.113:smtp

ss dst 192.168.119.113:443

# 匹配本地地址和端口号

# ss src ADDRESS\_PATTERN

ss src 192.168.119.103

ss src 192.168.119.103:http

ss src 192.168.119.103:80

ss src 192.168.119.103:smtp

ss src 192.168.119.103:25

**将本地或者远程端口和一个数比较**

# ss dport OP PORT 远程端口和一个数比较；

# ss sport OP PORT 本地端口和一个数比较

# OP 可以代表以下任意一个:

# <= or le : 小于或等于端口号

# >= or ge : 大于或等于端口号

# == or eq : 等于端口号

# != or ne : 不等于端口号

# < or gt : 小于端口号

# > or lt : 大于端口号

ss sport = :http

ss dport = :http

ss dport \> :1024

ss sport \> :1024

ss sport \< :32000

ss sport eq :22

ss dport != :22

ss state connected sport = :http

ss \( sport = :http or sport = :https \)

ss -o state fin-wait-1 \( sport = :http or sport = :https \) dst 192.168.1/24

**用TCP 状态过滤Sockets**

ss -4 state closing

# ss -4 state FILTER-NAME-HERE

# ss -6 state FILTER-NAME-HERE

# FILTER-NAME-HERE 可以代表以下任何一个：

# established、 syn-sent、 syn-recv、 fin-wait-1、 fin-wait-2、 time-wait、 closed、 close-wait、 last-ack、 listen、 closing、

# all : 所有以上状态

# connected : 除了listen and closed的所有状态

# synchronized :所有已连接的状态除了syn-sent

# bucket : 显示状态为maintained as minisockets,如：time-wait和syn-recv.

# big : 和bucket相反.

**显示ICP连接**

[root@localhost ~]# ss -t -a

State Recv-Q Send-Q Local Address:Port Peer Address:Port

LISTEN 0 0 \*:3306 \*:\*

LISTEN 0 0 \*:http \*:\*

LISTEN 0 0 \*:ssh \*:\*

LISTEN 0 0 127.0.0.1:smtp \*:\*

ESTAB 0 0 112.124.15.130:42071 42.156.166.25:http

ESTAB 0 0 112.124.15.130:ssh 121.229.196.235:33398

**显示 Sockets 摘要**

[root@localhost ~]# ss -s

Total: 172 (kernel 189)

TCP: 10 (estab 2, closed 4, orphaned 0, synrecv 0, timewait 0/0), ports 5

Transport Total ip IPv6

\* 189 - -

RAW 0 0 0

UDP 5 5 0

TCP 6 6 0

INET 11 11 0

FRAG 0 0 0

列出当前的established, closed, orphaned and waiting TCP sockets

**列出所有打开的网络连接端口**

[root@localhost ~]# ss -l

Recv-Q Send-Q Local Address:Port Peer Address:Port

0 0 \*:3306 \*:\*

0 0 \*:http \*:\*

0 0 \*:ssh \*:\*

0 0 127.0.0.1:smtp \*:\*

**查看进程使用的socket**

[root@localhost ~]# ss -pl

Recv-Q Send-Q Local Address:Port Peer Address:Port

0 0 \*:3306 \*:\* users:(("mysqld",1718,10))

0 0 \*:http \*:\* users:(("nginx",13312,5),("nginx",13333,5))

0 0 \*:ssh \*:\* users:(("sshd",1379,3))

0 0 127.0.0.1:smtp \*:\* us

**找出打开套接字/端口应用程序**

[root@localhost ~]# ss -pl | grep 3306

0 0 \*:3306 \*:\* users:(("mysqld",1718,10))

**显示所有UDP Sockets**

[root@localhost ~]# ss -u -a

State Recv-Q Send-Q Local Address:Port Peer Address:Port

UNCONN 0 0 \*:syslog \*:\*

UNCONN 0 0 112.124.15.130:ntp \*:\*

UNCONN 0 0 10.160.7.81:ntp \*:\*

UNCONN 0 0 127.0.0.1:ntp \*:\*

UNCONN 0 0 \*:ntp \*:\*

#### 出所有端口为 22（ssh）的连接

ss state all sport = :ssh

Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port

tcp LISTEN 0 128 \*:ssh \*:\*

tcp ESTAB 0 0 192.168.0.136:ssh 192.168.0.102:46540

tcp LISTEN 0 128 :::ssh :::\*

## openssl

强大的安全套接字层密码库

## 补充说明

**OpenSSL** 是一个强大的安全套接字层密码库，囊括主要的密码算法、常用的密钥和证书封装管理功能及SSL协议，并提供丰富的应用程序供测试或其它目的使用。在OpenSSL被曝出现严重安全漏洞后，发现多数通过SSL协议加密的网站使用名为OpenSSL的开源软件包。由于这是互联网应用最广泛的安全传输方法，被网银、在线支付、电商网站、门户网站、电子邮件等重要网站广泛使用，所以该漏洞影响范围广大。

OpenSSL有两种运行模式：交互模式和批处理模式。

直接输入openssl回车进入交互模式，输入带命令选项的openssl进入批处理模式。

OpenSSL整个软件包大概可以分成三个主要的功能部分：密码算法库、SSL协议库以及应用程序。OpenSSL的目录结构自然也是围绕这三个功能部分进行规划的。

**对称加密算法**

OpenSSL一共提供了8种对称加密算法，其中7种是分组加密算法，仅有的一种流加密算法是RC4。这7种分组加密算法分别是AES、DES、Blowfish、CAST、IDEA、RC2、RC5，都支持电子密码本模式（ECB）、加密分组链接模式（CBC）、加密反馈模式（CFB）和输出反馈模式（OFB）四种常用的分组密码加密模式。其中，AES使用的加密反馈模式（CFB）和输出反馈模式（OFB）分组长度是128位，其它算法使用的则是64位。事实上，DES算法里面不仅仅是常用的DES算法，还支持三个密钥和两个密钥3DES算法。

**非对称加密算法**

OpenSSL一共实现了4种非对称加密算法，包括DH算法、RSA算法、DSA算法和椭圆曲线算法（EC）。DH算法一般用户密钥交换。RSA算法既可以用于密钥交换，也可以用于数字签名，当然，如果你能够忍受其缓慢的速度，那么也可以用于数据加密。DSA算法则一般只用于数字签名。

**信息摘要算法**

OpenSSL实现了5种信息摘要算法，分别是MD2、MD5、MDC2、SHA（SHA1）和RIPEMD。SHA算法事实上包括了SHA和SHA1两种信息摘要算法，此外，OpenSSL还实现了DSS标准中规定的两种信息摘要算法DSS和DSS1。

**密钥和证书管理**

密钥和证书管理是PKI的一个重要组成部分，OpenSSL为之提供了丰富的功能，支持多种标准。

首先，OpenSSL实现了ASN.1的证书和密钥相关标准，提供了对证书、公钥、私钥、证书请求以及CRL等数据对象的DER、PEM和BASE64的编解码功能。OpenSSL提供了产生各种公开密钥对和对称密钥的方法、函数和应用程序，同时提供了对公钥和私钥的DER编解码功能。并实现了私钥的PKCS#12和PKCS#8的编解码功能。OpenSSL在标准中提供了对私钥的加密保护功能，使得密钥可以安全地进行存储和分发。

在此基础上，OpenSSL实现了对证书的X.509标准编解码、PKCS#12格式的编解码以及PKCS#7的编解码功能。并提供了一种文本数据库，支持证书的管理功能，包括证书密钥产生、请求产生、证书签发、吊销和验证等功能。

事实上，OpenSSL提供的CA应用程序就是一个小型的证书管理中心（CA），实现了证书签发的整个流程和证书管理的大部分机制。

### 实例

**1、使用 openssl 生成密码**

几乎所有 Linux 发行版都包含 openssl。我们可以利用它的随机功能来生成可以用作密码的随机字母字符串。

openssl rand -base64 10

# nU9LlHO5nsuUvw==

nU9LlHO5nsuUvw==

**2、消息摘要算法应用例子**

用SHA1算法计算文件file.txt的哈西值，输出到stdout：

# openssl dgst -sha1 file.txt

用SHA1算法计算文件file.txt的哈西值，输出到文件digest.txt：

# openssl sha1 -out digest.txt file.txt

用DSS1(SHA1)算法为文件file.txt签名，输出到文件dsasign.bin。签名的private key必须为DSA算法产生的，保存在文件dsakey.pem中。

# openssl dgst -dss1 -sign dsakey.pem -out dsasign.bin file.txt

用dss1算法验证file.txt的数字签名dsasign.bin，验证的private key为DSA算法产生的文件dsakey.pem。

# openssl dgst -dss1 -prverify dsakey.pem -signature dsasign.bin file.txt

用sha1算法为文件file.txt签名,输出到文件rsasign.bin，签名的private key为RSA算法产生的文件rsaprivate.pem。

# openssl sha1 -sign rsaprivate.pem -out rsasign.bin file.txt

用sha1算法验证file.txt的数字签名rsasign.bin，验证的public key为RSA算法生成的rsapublic.pem。

# openssl sha1 -verify rsapublic.pem -signature rsasign.bin file.txt

**3、对称加密应用例子**

对称加密应用例子，用DES3算法的CBC模式加密文件plaintext.doc，加密结果输出到文件ciphertext.bin。

# openssl enc -des3 -salt -in plaintext.doc -out ciphertext.bin

用DES3算法的OFB模式解密文件ciphertext.bin，提供的口令为trousers，输出到文件plaintext.doc。注意：因为模式不同，该命令不能对以上的文件进行解密。

# openssl enc -des-ede3-ofb -d -in ciphertext.bin -out plaintext.doc -pass pass:trousers

用Blowfish的CFB模式加密plaintext.doc，口令从环境变量PASSWORD中取，输出到文件ciphertext.bin。

# openssl bf-cfb -salt -in plaintext.doc -out ciphertext.bin -pass env:PASSWORD

给文件ciphertext.bin用base64编码，输出到文件base64.txt。

# openssl base64 -in ciphertext.bin -out base64.txt

用RC5算法的CBC模式加密文件plaintext.doc，输出到文件ciphertext.bin，salt、key和初始化向量(iv)在命令行指定。

# openssl rc5 -in plaintext.doc -out ciphertext.bin -S C62CB1D49F158ADC -iv E9EDACA1BD7090C6 -K 89D4B1678D604FAA3DBFFD030A314B29

**4、Diffie-Hellman应用例子**

使用生成因子2和随机的1024-bit的素数产生D0ffie-Hellman参数，输出保存到文件dhparam.pem

# openssl dhparam -out dhparam.pem -2 1024

从dhparam.pem中读取Diffie-Hell参数，以C代码的形式，输出到stdout。

# openssl dhparam -in dhparam.pem -noout -C

**5、DSA应用例子应用例子**

生成1024位DSA参数集，并输出到文件dsaparam.pem。

# openssl dsaparam -out dsaparam.pem 1024

使用参数文件dsaparam.pem生成DSA私钥匙，采用3DES加密后输出到文件dsaprivatekey.pem

# openssl gendsa -out dsaprivatekey.pem -des3 dsaparam.pem

使用私钥匙dsaprivatekey.pem生成公钥匙，输出到dsapublickey.pem

# openssl dsa -in dsaprivatekey.pem -pubout -out dsapublickey.pem

从dsaprivatekey.pem中读取私钥匙，解密并输入新口令进行加密，然后写回文件dsaprivatekey.pem

# openssl dsa -in dsaprivatekey.pem -out dsaprivatekey.pem -des3 -passin

**6、RSA应用例子**

产生1024位RSA私匙，用3DES加密它，口令为trousers，输出到文件rsaprivatekey.pem

# openssl genrsa -out rsaprivatekey.pem -passout pass:trousers -des3 1024

从文件rsaprivatekey.pem读取私匙，用口令trousers解密，生成的公钥匙输出到文件rsapublickey.pem

# openssl rsa -in rsaprivatekey.pem -passin pass:trousers -pubout -out rsapubckey.pem

用公钥匙rsapublickey.pem加密文件plain.txt，输出到文件cipher.txt

# openssl rsautl -encrypt -pubin -inkey rsapublickey.pem -in plain.txt -out cipher.txt

使用私钥匙rsaprivatekey.pem解密密文cipher.txt，输出到文件plain.txt

# openssl rsautl -decrypt -inkey rsaprivatekey.pem -in cipher.txt -out plain.txt

用私钥匙rsaprivatekey.pem给文件plain.txt签名，输出到文件signature.bin

# openssl rsautl -sign -inkey rsaprivatekey.pem -in plain.txt -out signature.bin

用公钥匙rsapublickey.pem验证签名signature.bin，输出到文件plain.txt

# openssl rsautl -verify -pubin -inkey rsapublickey.pem -in signature.bin -out plain

从X.509证书文件cert.pem中获取公钥匙，用3DES加密mail.txt，输出到文件mail.enc

# openssl smime -encrypt -in mail.txt -des3 -out mail.enc cert.pem

从X.509证书文件cert.pem中获取接收人的公钥匙，用私钥匙key.pem解密S/MIME消息mail.enc，结果输出到文件mail.txt

# openssl smime -decrypt -in mail.enc -recip cert.pem -inkey key.pem -out mail.txt

cert.pem为X.509证书文件，用私匙key,pem为mail.txt签名，证书被包含在S/MIME消息中，输出到文件mail.sgn

# openssl smime -sign -in mail.txt -signer cert.pem -inkey key.pem -out mail.sgn

验证S/MIME消息mail.sgn，输出到文件mail.txt，签名者的证书应该作为S/MIME消息的一部分包含在mail.sgn中

# openssl smime -verify -in mail.sgn -out mail.txt

更多实例:

openssl version -a

openssl help

openssl genrsa -aes128 -out fd.key 2048 # pem format

openssl rsa -text -in fd.key

## ssh

openssh套件中的客户端连接工具

补充说明

**ssh命令** 是openssh套件中的客户端连接工具，可以给予ssh加密协议实现安全的远程登录服务器。

**语法**

ssh(选项)(参数)

**选项**

-1：强制使用ssh协议版本1；

-2：强制使用ssh协议版本2；

-4：强制使用IPv4地址；

-6：强制使用IPv6地址；

-A：开启认证代理连接转发功能；

-a：关闭认证代理连接转发功能；

-b：使用本机指定地址作为对应连接的源ip地址；

-C：请求压缩所有数据；

-F：指定ssh指令的配置文件；

-f：后台执行ssh指令；

-g：允许远程主机连接主机的转发端口；

-i：指定身份文件；

-l：指定连接远程服务器登录用户名；

-N：不执行远程指令；

-o：指定配置选项；

-p：指定远程服务器上的端口；

-q：静默模式；

-X：开启X11转发功能；

-x：关闭X11转发功能；

-y：开启信任X11转发功能。

**参数**

* 远程主机：指定要连接的远程ssh服务器；
* 指令：要在远程ssh服务器上执行的指令。

**实例**

# ssh 用户名@远程服务器地址

ssh user1@172.24.210.101

# 指定端口

ssh -p 2211 root@140.206.185.170

# ssh 大家族

ssh user@ip -p22 # 默认用户名为当前用户名，默认端口为 22

ssh-keygen # 为当前用户生成 ssh 公钥 + 私钥

ssh-keygen -f keyfile -i -m key\_format -e -m key\_format # key\_format: RFC4716/SSH2(default) PKCS8 PEM

ssh-copy-id user@ip:port # 将当前用户的公钥复制到需要 ssh 的服务器的 ~/.ssh/authorized\_keys，之后可以免密登录

**背后故事**

英文：Tatu Ylonen  
编译：Linux中国/kenxx  
来源：https://linux.cn/article-8476-1.html

为什么 SSH（安全终端）的端口号是 22 呢，这不是一个巧合，这其中有个我（Tatu Ylonen，SSH 协议的设计者）未曾诉说的故事。

**将 SSH 协议端口号设为 22 的故事**

1995 年春我编写了 SSH 协议的最初版本，那时候 telnet 和 FTP 正被广泛使用。

当时我设计 SSH 协议想着是为了替代 telnet（端口 23）和 ftp（端口21）两个协议的，而端口 22 是空闲的。我想当然地就选择了夹在 telnet 和 ftp 的端口中间的数字。我觉得端口号虽然是个小事但似乎又存在着某种信念。但我到底要怎么拿到那个端口号呢？我未曾拥有过任何一个端口号，但我却认识几个拥有端口号的人！

在那时取得端口号的事情其实说来挺简单的。毕竟当时的因特网（Internet）并不是很大，是因特网爆炸的早期。端口号分配的活儿是 IANA（Internet Assigned Numbers Authority，互联网数字分配机构）干的。在那时这机构可相当于是因特网先驱 [Jon Postel](https://en.wikipedia.org/wiki/Jon_Postel) 和 [Joyce K. Reynolds](https://en.wikipedia.org/wiki/Joyce_K._Reynolds) 一般的存在。Jon 参与编写了多项主要的协议标准，例如 IP（RFC 791）、ICMP（RFC 792）和 TCP（RFC 793）等一些你应该早有耳闻的协议。

我可以说是敬畏 Jon 先生的，他参与编写了几乎所有主要的因特网标准文档（Internet RFC）！

1995 年 7 月，就在我发布 ssh-1.0 前，我发送了一封邮件给 IANA：

From ylo Mon Jul 10 11:45:48 +0300 1995  
From: Tatu Ylonen  
To: Internet Assigned Numbers Authority  
Subject: 请求取得一个端口号  
Organization: 芬兰赫尔辛基理工大学

亲爱的机构成员：

我写了个可以在不安全的网络环境中安全地从一台机器登录到另一台机器的程序。它主要是对现有的 telnet 协议以及 rlogin 协议的功能性提升和安全性改进。说的具体些，就是可以防御 IP、DNS > 或路由等欺骗行为。我打算将我的软件免费地发布在因特网上，以得到广泛地使用。

我希望为该软件注册一个特权端口号，要是这个端口号在 1 到 255 > 之间就更好了，这样它就可以用在名字服务器的 WKS 字段中了。

我在附件中附上了协议标准的草案。这个软件已经在本地运行了几个月了，我已准备在获得端口号后就发布。如果端口号分配一事安排的及时，我希望这周就将要发布的软件准备好。我目前在 beta 版测试时使用的端口号是 > 22，如果要是能够分配到这个端口，我就不用做什么更改了（目前这个端口在列表中还是空闲的）。

软件中服务的名称叫 ssh（系 Secure Shell 的缩写）。

您最真诚的，  
Tatu Ylonen

（LCTT 译注：DNS 协议中的 WKS 记录类型意即“众所周知的业务描述”，是类似于 A、MX 这样的 DNS 记录类型，用于描述某个 IP 所提供的服务，目前鲜见使用。参见： <https://docs.oracle.com/cd/E19683-01/806-4077/dnsintro-154/index.html> 。）

第二天，我就收到了 Joyce 发来的邮件：

Date: Mon, 10 Jul 1995 15:35:33 -0700  
From: [jkrey@ISI.EDU](mailto:jkrey@ISI.EDU)  
To: [ylo@cs.hut.fi](mailto:ylo@cs.hut.fi)  
Subject: 回复：请求取得一个端口号  
Cc: [iana@ISI.EDU](mailto:iana@ISI.EDU)  
Tatu,  
我们将端口号 22 分配给 ssh 服务了，你目前是该服务的主要联系人。  
Joyce

这就搞定了！SSH 的端口正式使用 22！！！

1995 年 7 月 12 日上午 2 点 21 分，我给我在赫尔辛基理工大学的测试者们宣布了 SSH 的最后 beta 版本。当日下午 5 点 23 分，我给测试者们宣布了 ssh-1.0.0 版本。1995 年 7 月 12 日，下午 5 点 51 分，我将一份 SSH（安全终端）的宣告发给了 [cypherpunks@toad.com](mailto:cypherpunks@toad.com) 的邮件列表，此外我还将其发给了一些新闻组、邮件列表和一些在因特网上讨论相关话题的人们。

**如何更改 SSH 服务的端口号**

SSH 服务器是默认运行在 22 号端口上的。然而，由于某些原因需要，它也可以运行在别的端口上。比如为了方便测试使用，又比如在同一个宿主机上运行多个不同的配置。当然，极少情况下，不使用 root 权限运行它也可以，比如某些必须运行在非特权的端口的情况（端口号大于等于 1024）。

端口号可以在配置文件 /etc/ssh/sshd\_config 中将 Port 22 更改。也可以使用 -p 选项运行 sshd。SSH 客户端和 sftp 程序也可以使用 -p 选项。

**配置 SSH 协议穿越防火墙**

SSH 是少数通常被许可穿越防火墙的协议之一。通常的做法是不限制出站的 SSH 连接，尤其常见于一些较小的或者比较技术型的组织中，而入站的 SSH 连接通常会限制到一台或者是少数几台服务器上。

**出站的 SSH 连接**

在防火墙中配置出站的 SSH 连接十分简单。如果完全限制了外发连接，那么只需要创建一个允许 TCP 端口 22 可以外发的规则即可。如果你想限制目标地址，你可以限制该规则仅允许访问你的组织放在云端的外部服务器或保护该云端的跳板服务器即可。

**反向通道是有风险的**

其实不限制出站的 SSH 连接虽然是可以的，但是是存在风险的，SSH 协议是支持 通道访问 的。最初的想法是在外部服务器搭建一个 SSH 服务监听来自各处的连接，将进入的连接转发到组织，并让这个连接可以访问某个内部服务器。

在某些场景下这当然非常的方便。开发者和系统管理员经常使用它打开一个通道以便于他们可以远程访问，比如在家里或者在旅行中使用笔记本电脑等场景。

然而通常来讲这些做法是违背安全策略的，跳过了防火墙管理员和安全团队保护的控制无疑是违背安全策略的，比如这些： PCI、HIPAA、NIST SP 800-53 等。它可以被黑客和外国情报机构用来在组织内留下后门。

CryptoAuditor 是一款可以控制通道穿过防火墙或者一组云端服务器入口的产品。该款产品可以配合 通用 SSH 密钥管理器（Universal SSH Key Manager） 来获得对 主机密钥（host keys）的访问，以在启用防火墙并阻挡未授权转发的场景中解密 SSH 会话。

**入站的 SSH 访问**

对于入站访问而言，这里有几点需要说一下：

配置防火墙，并转发所有去往 22 端口的连接只能流向到一个特定的内部网络 IP 地址或者一个 DMZ 主机。在该 IP 上运行 CryptoAuditor 或者跳板机来控制和审查所有访问该组织的连接。 在防火墙上使用不同的端口访问不同的服务器。 只允许使用 IPsec 协议这样的 VPN（虚拟专用网）登录后连接 SSH 服务。

**通过 iptables 服务限制 SSH 访问**

iptables 是一款内建在 Linux 内核的宿主防火墙。通常配置用于保护服务器以防止被访问那些未明确开启的端口。

如果服务器上启用了 iptables，使用下面的命令将可以允许进入的 SSH 访问，当然命令需要以 root 身份运行。

iptables -A INPUT -p tcp --dport 22 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT

iptables -A OUTPUT -p tcp --sport 22 -m conntrack --ctstate ESTABLISHED -j ACCEPT

如果你想将上述命令创建的规则持久地保存，在某些系统版本中，可使用如下命令：

service iptables save

## passwd

用于让用户可以更改自己的密码

## 补充说明

**passwd命令** 用于设置用户的认证信息，包括用户密码、密码过期时间等。系统管理者则能用它管理系统用户的密码。只有管理者可以指定用户名称，一般用户只能变更自己的密码。

### 语法

passwd(选项)(参数)

### 选项

-d：删除密码，仅有系统管理者才能使用；

-f：强制执行；

-k：设置只有在密码过期失效后，方能更新；

-l：锁住密码；

-s：列出密码的相关信息，仅有系统管理者才能使用；

-u：解开已上锁的帐号。

### 参数

用户名：需要设置密码的用户名。

### 知识扩展

与用户、组账户信息相关的文件

存放用户信息：

/etc/passwd

/etc/shadow

存放组信息：

/etc/group

/etc/gshadow

用户信息文件分析（每项用:隔开）

例如：jack:X:503:504:::/home/jack/:/bin/bash

jack　　# 用户名

X　　# 口令、密码

503　　# 用户id（0代表root、普通新建用户从500开始）

504　　# 所在组

:　　# 描述

/home/jack/　　# 用户主目录

/bin/bash　　# 用户缺省Shell

组信息文件分析

例如：jack:$!$:???:13801:0:99999:7:\*:\*:

jack　　# 组名

$!$　　# 被加密的口令

13801　　# 创建日期与今天相隔的天数

0　　# 口令最短位数

99999　　# 用户口令

7　　# 到7天时提醒

\*　　# 禁用天数

\*　　# 过期天数

### 实例

如果是普通用户执行passwd只能修改自己的密码。如果新建用户后，要为新用户创建密码，则用passwd用户名，注意要以root用户的权限来创建。

[root@localhost ~]# passwd linuxde # 更改或创建linuxde用户的密码；

Changing password for user linuxde.

New UNIX password: # 请输入新密码；

Retype new UNIX password: # 再输入一次；

passwd: all authentication tokens updated successfully. # 成功；

普通用户如果想更改自己的密码，直接运行passwd即可，比如当前操作的用户是linuxde。

[linuxde@localhost ~]$ passwd

Changing password for user linuxde. # 更改linuxde用户的密码；

(current) UNIX password: # 请输入当前密码；

New UNIX password: # 请输入新密码；

Retype new UNIX password: # 确认新密码；

passwd: all authentication tokens updated successfully. # 更改成功；

比如我们让某个用户不能修改密码，可以用-l选项来锁定：

[root@localhost ~]# passwd -l linuxde # 锁定用户linuxde不能更改密码；

Locking password for user linuxde.

passwd: Success # 锁定成功；

[linuxde@localhost ~]# su linuxde # 通过su切换到linuxde用户；

[linuxde@localhost ~]$ passwd # linuxde来更改密码；

Changing password for user linuxde.

Changing password for linuxde

(current) UNIX password: # 输入linuxde的当前密码；

passwd: Authentication token manipulation error # 失败，不能更改密码；

再来一例：

[root@localhost ~]# passwd -d linuxde # 清除linuxde用户密码；

Removing password for user linuxde.

passwd: Success # 清除成功；

[root@localhost ~]# passwd -S linuxde # 查询linuxde用户密码状态；

Empty password. # 空密码，也就是没有密码；

注意：当我们清除一个用户的密码时，登录时就无需密码，这一点要加以注意。

## useradd

创建的新的系统用户

## 补充说明

**useradd命令** 用于Linux中创建的新的系统用户。useradd可用来建立用户帐号。帐号建好之后，再用passwd设定帐号的密码．而可用userdel删除帐号。使用useradd指令所建立的帐号，实际上是保存在/etc/passwd文本文件中。

在Slackware中，adduser指令是个script程序，利用交谈的方式取得输入的用户帐号资料，然后再交由真正建立帐号的useradd命令建立新用户，如此可方便管理员建立用户帐号。在Red Hat Linux中， **adduser命令** 则是useradd命令的符号连接，两者实际上是同一个指令。

### 语法

useradd(选项)(参数)

### 选项

-c<备注>：加上备注文字。备注文字会保存在passwd的备注栏位中；

-d<登入目录>：指定用户登入时的启始目录；

-D：变更预设值；

-e<有效期限>：指定帐号的有效期限；

-f<缓冲天数>：指定在密码过期后多少天即关闭该帐号；

-g<群组>：指定用户所属的群组；

-G<群组>：指定用户所属的附加群组；

-m：自动建立用户的登入目录；

-M：不要自动建立用户的登入目录；

-n：取消建立以用户名称为名的群组；

-r：建立系统帐号；

-s<shell>：指定用户登入后所使用的shell；

-u<uid>：指定用户id。

### 参数

用户名：要创建的用户名。

### 实例

新建用户加入组：

useradd –g sales jack –G company,employees //-g：加入主要组、-G：加入次要组

建立一个新用户账户，并设置ID：

useradd caojh -u 544

需要说明的是，设定ID值时尽量要大于500，以免冲突。因为Linux安装后会建立一些特殊用户，一般0到499之间的值留给bin、mail这样的系统账号。

## users

显示当前登录系统的所有用户

## 补充说明

**users命令** 用于显示当前登录系统的所有用户的用户列表。每个显示的用户名对应一个登录会话。如果一个用户有不止一个登录会话，那他的用户名将显示相同的次数。

### 语法

users(选项)

### 选项

--help：显示命令的帮助信息；

--version：显示命令的版本信息。

### 实例

[root@localhost ~]# users

root root

## kill

删除执行中的程序或工作

## 补充说明

**kill命令** 用来删除执行中的程序或工作。kill可将指定的信息送至程序。预设的信息为SIGTERM(15),可将指定程序终止。若仍无法终止该程序，可使用SIGKILL(9)信息尝试强制删除程序。程序或工作的编号可利用ps指令或job指令查看。

### 语法

kill(选项)(参数)

xkill # 图形化关闭程序

### 选项

-a：当处理当前进程时，不限制命令名和进程号的对应关系；

-l <信息编号>：若不加<信息编号>选项，则-l参数会列出全部的信息名称；

-p：指定kill 命令只打印相关进程的进程号，而不发送任何信号；

-s <信息名称或编号>：指定要送出的信息；

-u：指定用户。

### 参数

进程或作业识别号：指定要删除的进程或作业。

### 实例

列出所有信号名称：

kill -l

1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL

5) SIGTRAP 6) SIGABRT 7) SIGBUS 8) SIGFPE

9) SIGKILL 10) SIGUSR1 11) SIGSEGV 12) SIGUSR2

13) SIGPIPE 14) SIGALRM 15) SIGTERM 16) SIGSTKFLT

17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP

21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU

25) SIGXFSZ 26) SIGVTALRM 27) SIGPROF 28) SIGWINCH

29) SIGIO 30) SIGPWR 31) SIGSYS 34) SIGRTMIN

35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3 38) SIGRTMIN+4

39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8

43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12

47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14

51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10

55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7 58) SIGRTMAX-6

59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2

63) SIGRTMAX-1 64) SIGRTMAX

只有第9种信号(SIGKILL)才可以无条件终止进程，其他信号进程都有权利忽略， **下面是常用的信号：**

HUP 1 终端断线

INT 2 中断（同 Ctrl + C）

QUIT 3 退出（同 Ctrl + \）

TERM 15 终止

KILL 9 强制终止

CONT 18 继续（与STOP相反， fg/bg命令）

STOP 19 暂停（同 Ctrl + Z）

先用ps查找进程，然后用kill杀掉：

ps -ef | grep vim

root 3268 2884 0 16:21 pts/1 00:00:00 vim install.log

root 3370 2822 0 16:21 pts/0 00:00:00 grep vim

kill 3268

kill 3268

-bash: kill: (3268) - 没有那个进程

## ntpdate

使用网络计时协议（NTP）设置日期和时间

补充说明

**ntpdate命令** 是用来设置本地日期和时间。它从指定的每个服务器获得了一些样本，并应用标准 NTP 时钟过滤器和选择算法来选择最好的样本。

此 ntpdate 命令使用以下方法进行时间调整：

* 如果它确定时钟偏差超过 0.5 秒，它通过调用 settimeofday 子例程设置时钟时间。在引导时间，这是一个首选的方法。
* 如 果它确定时钟偏差小于 0.5 秒，它通过调用 adjtime 子例程和偏移量来调整时钟时间。此方法倾向于用牺牲一些稳定性来保持漂移时钟更加准确。 当不是通过运行一个守护程序而是从 cron 命令有规则的运行ntpdate 命令时，每一小时或两小时执行一次可以保证足够的走时精度，从而避免调整时钟。

使用很多服务器可以大幅度改善 ntpdate 命令的可靠性与精度。尽管能使用单一服务器，但您能通过提供至少三个或四个服务器以获得更好的性能。

如果一个类似 xntpd 守护程序的 NTP 服务器守护程序正在同一主机上运行，命令将拒绝ntpdate 设置日期。

你必须有 root 权限才能在主机上运行这个命令。

**语法**

ntpdate [ -b] [ -d] [ -s] [ -u] [ -aKeyid] [ -eAuthenticationDelay] [ -kKeyFile] [ -oVersion] [ -pSamples] [ -tTimeOut] Server...

**选项**

|  |  |
| --- | --- |
| -aKeyid | 使用 Keyid 来认证全部数据包。 |
| -b | 通过调用 settimeofday 子例程来增加时钟的时间。 |
| -d | 指定调试方式。判断 ntpdate 命令会产生什么结果（不产生实际的结果）。结果再现在屏幕上。这个标志使用无特权的端口。 |
| -eAuthenticationDelay | 指定延迟认证处理的时间秒数。 |
| -kKeyFile | 当不使用缺省值 /etc/ntp.keys 文件时，为包含密钥的文件指定一个不同的名称。 请参阅文件KeyFile的描述。 |
| -oVersion | 当轮询它的发出数据包时，指定使用的 NTP 版本实现。 Version 的值可以是 1，2，3。缺省值是 3。 |
| -pSamples | 指定从每个服务器获取的样本的数目。 Samples 的值在 1 和 8 之间，并包括 1 和 8。它的缺省值是 4。 |
| -s | 指定日志操作 syslog 设施的使用，而不是使用标准输出。 当运行 ntpdate 命令和 cron命令时，它是很有用的。 |
| -tTimeOut | 指定等待响应的时间。给定 TimeOut 的值四舍五入为 0.2 秒的倍数。缺省值是 1 秒。 |
| -u | 指定使用无特权的端口发送数据包。 当在一个对特权端口的输入流量进行阻拦的防火墙后是很有益的， 并希望在防火墙之外和主机同步。防火墙是一个系统或者计算机，它控制从外网对专用网的访问。 |

## ifconfig

配置和显示Linux系统网卡的网络参数

补充说明

**ifconfig命令** 被用于配置和显示Linux内核中网络接口的网络参数。用ifconfig命令配置的网卡信息，在网卡重启后机器重启后，配置就不存在。要想将上述的配置信息永远的存的电脑里，那就要修改网卡的配置文件了。

**语法**

ifconfig(参数)

**参数**

add<地址>：设置网络设备IPv6的ip地址；

del<地址>：删除网络设备IPv6的IP地址；

down：关闭指定的网络设备；

<hw<网络设备类型><硬件地址>：设置网络设备的类型与硬件地址；

io\_addr<I/O地址>：设置网络设备的I/O地址；

irq<IRQ地址>：设置网络设备的IRQ；

media<网络媒介类型>：设置网络设备的媒介类型；

mem\_start<内存地址>：设置网络设备在主内存所占用的起始地址；

metric<数目>：指定在计算数据包的转送次数时，所要加上的数目；

mtu<字节>：设置网络设备的MTU；

netmask<子网掩码>：设置网络设备的子网掩码；

tunnel<地址>：建立IPv4与IPv6之间的隧道通信地址；

up：启动指定的网络设备；

-broadcast<地址>：将要送往指定地址的数据包当成广播数据包来处理；

-pointopoint<地址>：与指定地址的网络设备建立直接连线，此模式具有保密功能；

-promisc：关闭或启动指定网络设备的promiscuous模式；

IP地址：指定网络设备的IP地址；

网络设备：指定网络设备的名称。

**实例**

**显示网络设备信息（激活状态的）：**

[root@localhost ~]# ifconfig

eth0 Link encap:Ethernet HWaddr 00:16:3E:00:1E:51

inet addr:10.160.7.81 Bcast:10.160.15.255 Mask:255.255.240.0

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:61430830 errors:0 dropped:0 overruns:0 frame:0

TX packets:88534 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:3607197869 (3.3 GiB) TX bytes:6115042 (5.8 MiB)

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

UP LOOPBACK RUNNING MTU:16436 Metric:1

RX packets:56103 errors:0 dropped:0 overruns:0 frame:0

TX packets:56103 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:5079451 (4.8 MiB) TX bytes:5079451 (4.8 MiB)

说明：

**eth0** 表示第一块网卡，其中HWaddr表示网卡的物理地址，可以看到目前这个网卡的物理地址(MAC地址）是00:16:3E:00:1E:51。

**inet addr** 用来表示网卡的IP地址，此网卡的IP地址是10.160.7.81，广播地址Bcast:10.160.15.255，掩码地址Mask:255.255.240.0。

**lo** 是表示主机的回坏地址，这个一般是用来测试一个网络程序，但又不想让局域网或外网的用户能够查看，只能在此台主机上运行和查看所用的网络接口。比如把 httpd服务器的指定到回坏地址，在浏览器输入127.0.0.1就能看到你所架WEB网站了。但只是您能看得到，局域网的其它主机或用户无从知道。

* 第一行：连接类型：Ethernet（以太网）HWaddr（硬件mac地址）。
* 第二行：网卡的IP地址、子网、掩码。
* 第三行：UP（代表网卡开启状态）RUNNING（代表网卡的网线被接上）MULTICAST（支持组播）MTU:1500（最大传输单元）：1500字节。
* 第四、五行：接收、发送数据包情况统计。
* 第七行：接收、发送数据字节数统计信息。

**启动关闭指定网卡：**

ifconfig eth0 up

ifconfig eth0 down

ifconfig eth0 up为启动网卡eth0，ifconfig eth0 down为关闭网卡eth0。ssh登陆linux服务器操作要小心，关闭了就不能开启了，除非你有多网卡。

**为网卡配置和删除IPv6地址：**

ifconfig eth0 add 33ffe:3240:800:1005::2/64 #为网卡eth0配置IPv6地址

ifconfig eth0 del 33ffe:3240:800:1005::2/64 #为网卡eth0删除IPv6地址

**用ifconfig修改MAC地址：**

ifconfig eth0 hw ether 00:AA:BB:CC:dd:EE

**配置IP地址：**

[root@localhost ~]# ifconfig eth0 192.168.2.10

[root@localhost ~]# ifconfig eth0 192.168.2.10 netmask 255.255.255.0

[root@localhost ~]# ifconfig eth0 192.168.2.10 netmask 255.255.255.0 broadcast 192.168.2.255

**启用和关闭arp协议：**

ifconfig eth0 arp #开启网卡eth0 的arp协议

ifconfig eth0 -arp #关闭网卡eth0 的arp协议

**设置最大传输单元：**

ifconfig eth0 mtu 1500 #设置能通过的最大数据包大小为 1500 bytes

**其它实例**

ifconfig #处于激活状态的网络接口

ifconfig -a #所有配置的网络接口，不论其是否激活

ifconfig eth0 #显示eth0的网卡信息

## service

控制系统服务的实用工具

补充说明

**service命令** 是Redhat Linux兼容的发行版中用来控制系统服务的实用工具，它以启动、停止、重新启动和关闭系统服务，还可以显示所有系统服务的当前状态。

**语法**

service(选项)(参数)

**选项**

-h：显示帮助信息；

--status-all：显示所服务的状态。

**参数**

* 服务名：自动要控制的服务名，即/etc/init.d目录下的脚本文件名；
* 控制命令：系统服务脚本支持的控制命令。

**实例**

当修改了主机名、ip地址等信息时，经常需要把网络重启使之生效。

service network status

配置设备：

lo eth0

当前的活跃设备：

lo eth0

service network restart

正在关闭接口 eth0： [ 确定 ]

关闭环回接口： [ 确定 ]

设置网络参数： [ 确定 ]

弹出环回接口： [ 确定 ]

弹出界面 eth0： [ 确定 ]

重启mysql

service mysqld status

mysqld (pid 1638) 正在运行...

service mysqld restart

停止 MySQL： [ 确定 ]

启动 MySQL： [ 确定 ]

## systemctl

系统服务管理器指令

## 补充说明

**systemctl命令** 是系统服务管理器指令，它实际上将 service 和 chkconfig 这两个命令组合到一起。

| **任务** | **旧指令** | **新指令** |
| --- | --- | --- |
| 使某服务自动启动 | chkconfig --level 3 httpd on | systemctl enable httpd.service |
| 使某服务不自动启动 | chkconfig --level 3 httpd off | systemctl disable httpd.service |
| 检查服务状态 | service httpd status | systemctl status httpd.service （服务详细信息） systemctl is-active httpd.service （仅显示是否 Active) |
| 显示所有已启动的服务 | chkconfig --list | systemctl list-units --type=service |
| 启动某服务 | service httpd start | systemctl start httpd.service |
| 停止某服务 | service httpd stop | systemctl stop httpd.service |
| 重启某服务 | service httpd restart | systemctl restart httpd.service |

### 实例

systemctl start nfs-server.service . # 启动nfs服务

systemctl enable nfs-server.service # 设置开机自启动

systemctl disable nfs-server.service # 停止开机自启动

systemctl status nfs-server.service # 查看服务当前状态

systemctl restart nfs-server.service # 重新启动某服务

systemctl list-units --type=service # 查看所有已启动的服务

开启防火墙22端口

iptables -I INPUT -p tcp --dport 22 -j accept

如果仍然有问题，就可能是SELinux导致的

关闭SElinux：

修改/etc/selinux/config文件中的SELINUX=""为disabled，然后重启。

彻底关闭防火墙：

sudo systemctl status firewalld.service

sudo systemctl stop firewalld.service

sudo systemctl disable firewalld.service

## shutdown

用来执行系统关机的命令

补充说明

**shutdown命令** 用来系统关机命令。shutdown指令可以关闭所有程序，并依用户的需要，进行重新开机或关机的动作。

**语法**

shutdown(选项)(参数)

**选项**

-c：当执行“shutdown -h 11:50”指令时，只要按+键就可以中断关机的指令；

-f：重新启动时不执行fsck；

-F：重新启动时执行fsck；

-h：将系统关机；

-k：只是送出信息给所有用户，但不会实际关机；

-n：不调用init程序进行关机，而由shutdown自己进行；

-r：shutdown之后重新启动；

-t<秒数>：送出警告信息和删除信息之间要延迟多少秒。

**参数**

* [时间]：设置多久时间后执行shutdown指令；
* [警告信息]：要传送给所有登入用户的信息。

**实例**

指定现在立即关机：

shutdown -h now

指定5分钟后关机，同时送出警告信息给登入用户：

shutdown +5 "System will shutdown after 5 minutes"

## wget

Linux系统下载文件工具

补充说明

**wget命令** 用来从指定的URL下载文件。wget非常稳定，它在带宽很窄的情况下和不稳定网络中有很强的适应性，如果是由于网络的原因下载失败，wget会不断的尝试，直到整个文件下载完毕。如果是服务器打断下载过程，它会再次联到服务器上从停止的地方继续下载。这对从那些限定了链接时间的服务器上下载大文件非常有用。

wget支持HTTP，HTTPS和FTP协议，可以使用HTTP代理。所谓的自动下载是指，wget可以在用户退出系统的之后在后台执行。这意味这你可以登录系统，启动一个wget下载任务，然后退出系统，wget将在后台执行直到任务完成，相对于其它大部分浏览器在下载大量数据时需要用户一直的参与，这省去了极大的麻烦。

用于从网络上下载资源，没有指定目录，下载资源回默认为当前目录。wget虽然功能强大，但是使用起来还是比较简单：

1. **支持断点下传功能** 这一点，也是网络蚂蚁和FlashGet当年最大的卖点，现在，Wget也可以使用此功能，那些网络不是太好的用户可以放心了；
2. **同时支持FTP和HTTP下载方式** 尽管现在大部分软件可以使用HTTP方式下载，但是，有些时候，仍然需要使用FTP方式下载软件；
3. **支持代理服务器** 对安全强度很高的系统而言，一般不会将自己的系统直接暴露在互联网上，所以，支持代理是下载软件必须有的功能；
4. **设置方便简单** 可能，习惯图形界面的用户已经不是太习惯命令行了，但是，命令行在设置上其实有更多的优点，最少，鼠标可以少点很多次，也不要担心是否错点鼠标；
5. **程序小，完全免费** 程序小可以考虑不计，因为现在的硬盘实在太大了；完全免费就不得不考虑了，即使网络上有很多所谓的免费软件，但是，这些软件的广告却不是我们喜欢的。

**语法**

wget [参数] [URL地址]

**选项**

启动参数：

-V, –-version 显示wget的版本后退出

-h, –-help 打印语法帮助

-b, –-background 启动后转入后台执行

-e, –-execute=COMMAND 执行 `.wgetrc’格式的命令，wgetrc格式参见/etc/wgetrc或~/.wgetrc

记录和输入文件参数：

-o, –-output-file=FILE 把记录写到FILE文件中

-a, –-append-output=FILE 把记录追加到FILE文件中

-d, –-debug 打印调试输出

-q, –-quiet 安静模式(没有输出)

-v, –-verbose 冗长模式(这是缺省设置)

-nv, –-non-verbose 关掉冗长模式，但不是安静模式

-i, –-input-file=FILE 下载在FILE文件中出现的URLs

-F, –-force-html 把输入文件当作HTML格式文件对待

-B, –-base=URL 将URL作为在-F -i参数指定的文件中出现的相对链接的前缀

–-sslcertfile=FILE 可选客户端证书

–-sslcertkey=KEYFILE 可选客户端证书的KEYFILE

–-egd-file=FILE 指定EGD socket的文件名

下载参数：

–-bind-address=ADDRESS 指定本地使用地址(主机名或IP，当本地有多个IP或名字时使用)

-t, –-tries=NUMBER 设定最大尝试链接次数(0 表示无限制).

-O –-output-document=FILE 把文档写到FILE文件中

-nc, –-no-clobber 不要覆盖存在的文件或使用.#前缀

-c, –-continue 接着下载没下载完的文件

–progress=TYPE 设定进程条标记

-N, –-timestamping 不要重新下载文件除非比本地文件新

-S, –-server-response 打印服务器的回应

–-spider 不下载任何东西

-T, –-timeout=SECONDS 设定响应超时的秒数

-w, –-wait=SECONDS 两次尝试之间间隔SECONDS秒

–waitretry=SECONDS 在重新链接之间等待1…SECONDS秒

–random-wait 在下载之间等待0…2\*WAIT秒

-Y, –-proxy=on/off 打开或关闭代理

-Q, –-quota=NUMBER 设置下载的容量限制

–limit-rate=RATE 限定下载输率

目录参数：

-nd –-no-directories 不创建目录

-x, –-force-directories 强制创建目录

-nH, –-no-host-directories 不创建主机目录

-P, –-directory-prefix=PREFIX 将文件保存到目录 PREFIX/…

–cut-dirs=NUMBER 忽略 NUMBER层远程目录

HTTP 选项参数：

-–http-user=USER 设定HTTP用户名为 USER.

-–http-passwd=PASS 设定http密码为 PASS

-C, –-cache=on/off 允许/不允许服务器端的数据缓存 (一般情况下允许)

-E, –-html-extension 将所有text/html文档以.html扩展名保存

-–ignore-length 忽略 `Content-Length’头域

-–header=STRING 在headers中插入字符串 STRING

-–proxy-user=USER 设定代理的用户名为 USER

-–proxy-passwd=PASS 设定代理的密码为 PASS

-–referer=URL 在HTTP请求中包含 `Referer: URL’头

-s, –-save-headers 保存HTTP头到文件

-U, –-user-agent=AGENT 设定代理的名称为 AGENT而不是 Wget/VERSION

-–no-http-keep-alive 关闭 HTTP活动链接 (永远链接)

–-cookies=off 不使用 cookies

–-load-cookies=FILE 在开始会话前从文件 FILE中加载cookie

-–save-cookies=FILE 在会话结束后将 cookies保存到 FILE文件中

FTP 选项参数：

-nr, -–dont-remove-listing 不移走 `.listing’文件

-g, -–glob=on/off 打开或关闭文件名的 globbing机制

-–passive-ftp 使用被动传输模式 (缺省值).

-–active-ftp 使用主动传输模式

-–retr-symlinks 在递归的时候，将链接指向文件(而不是目录)

递归下载参数：

-r, -–recursive 递归下载－－慎用!

-l, -–level=NUMBER 最大递归深度 (inf 或 0 代表无穷)

–-delete-after 在现在完毕后局部删除文件

-k, –-convert-links 转换非相对链接为相对链接

-K, –-backup-converted 在转换文件X之前，将之备份为 X.orig

-m, –-mirror 等价于 -r -N -l inf -nr

-p, –-page-requisites 下载显示HTML文件的所有图片

递归下载中的包含和不包含(accept/reject)：

-A, –-accept=LIST 分号分隔的被接受扩展名的列表

-R, –-reject=LIST 分号分隔的不被接受的扩展名的列表

-D, –-domains=LIST 分号分隔的被接受域的列表

–-exclude-domains=LIST 分号分隔的不被接受的域的列表

–-follow-ftp 跟踪HTML文档中的FTP链接

–-follow-tags=LIST 分号分隔的被跟踪的HTML标签的列表

-G, –-ignore-tags=LIST 分号分隔的被忽略的HTML标签的列表

-H, –-span-hosts 当递归时转到外部主机

-L, –-relative 仅仅跟踪相对链接

-I, –-include-directories=LIST 允许目录的列表

-X, –-exclude-directories=LIST 不被包含目录的列表

-np, –-no-parent 不要追溯到父目录

wget -S –-spider url 不下载只显示过程

**参数**

URL：下载指定的URL地址。

**实例**

**使用wget下载单个文件**

wget http://www.jsdig.com/testfile.zip

以下的例子是从网络下载一个文件并保存在当前目录，在下载的过程中会显示进度条，包含（下载完成百分比，已经下载的字节，当前下载速度，剩余下载时间）。

**下载并以不同的文件名保存**

wget -O wordpress.zip http://www.jsdig.com/download.aspx?id=1080

wget默认会以最后一个符合/的后面的字符来命令，对于动态链接的下载通常文件名会不正确。

错误：下面的例子会下载一个文件并以名称download.aspx?id=1080保存:

wget http://www.jsdig.com/download?id=1

即使下载的文件是zip格式，它仍然以download.php?id=1080命令。

正确：为了解决这个问题，我们可以使用参数-O来指定一个文件名：

wget -O wordpress.zip http://www.jsdig.com/download.aspx?id=1080

**wget限速下载**

wget --limit-rate=300k http://www.jsdig.com/testfile.zip

当你执行wget的时候，它默认会占用全部可能的宽带下载。但是当你准备下载一个大文件，而你还需要下载其它文件时就有必要限速了。

**使用wget断点续传**

wget -c http://www.jsdig.com/testfile.zip

使用wget -c重新启动下载中断的文件，对于我们下载大文件时突然由于网络等原因中断非常有帮助，我们可以继续接着下载而不是重新下载一个文件。需要继续中断的下载时可以使用-c参数。

**使用wget后台下载**

wget -b http://www.jsdig.com/testfile.zip

Continuing in background, pid 1840.

Output will be written to `wget-log'.

对于下载非常大的文件的时候，我们可以使用参数-b进行后台下载，你可以使用以下命令来察看下载进度：

tail -f wget-log

**伪装代理名称下载**

wget --user-agent="Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Chrome/10.0.648.204 Safari/534.16" http://www.jsdig.com/testfile.zip

有些网站能通过根据判断代理名称不是浏览器而拒绝你的下载请求。不过你可以通过--user-agent参数伪装。

**测试下载链接**

当你打算进行定时下载，你应该在预定时间测试下载链接是否有效。我们可以增加--spider参数进行检查。

wget --spider URL

如果下载链接正确，将会显示:

Spider mode enabled. Check if remote file exists.

HTTP request sent, awaiting response... 200 OK

Length: unspecified [text/html]

Remote file exists and could contain further links,

but recursion is disabled -- not retrieving.

这保证了下载能在预定的时间进行，但当你给错了一个链接，将会显示如下错误:

wget --spider url

Spider mode enabled. Check if remote file exists.

HTTP request sent, awaiting response... 404 Not Found

Remote file does not exist -- broken link!!!

你可以在以下几种情况下使用--spider参数：

* 定时下载之前进行检查
* 间隔检测网站是否可用
* 检查网站页面的死链接

**增加重试次数**

wget --tries=40 URL

如果网络有问题或下载一个大文件也有可能失败。wget默认重试20次连接下载文件。如果需要，你可以使用--tries增加重试次数。

**下载多个文件**

wget -i filelist.txt

首先，保存一份下载链接文件：

cat > filelist.txt

url1

url2

url3

url4

接着使用这个文件和参数-i下载。

**镜像网站**

wget --mirror -p --convert-links -P ./LOCAL URL

下载整个网站到本地。

* --miror开户镜像下载。
* -p下载所有为了html页面显示正常的文件。
* --convert-links下载后，转换成本地的链接。
* -P ./LOCAL保存所有文件和目录到本地指定目录。

**过滤指定格式下载**

wget --reject=gif ur

下载一个网站，但你不希望下载图片，可以使用这条命令。

**把下载信息存入日志文件**

wget -o download.log URL

不希望下载信息直接显示在终端而是在一个日志文件，可以使用。

**限制总下载文件大小**

wget -Q5m -i filelist.txt

当你想要下载的文件超过5M而退出下载，你可以使用。注意：这个参数对单个文件下载不起作用，只能递归下载时才有效。

**下载指定格式文件**

wget -r -A.pdf url

可以在以下情况使用该功能：

* 下载一个网站的所有图片。
* 下载一个网站的所有视频。
* 下载一个网站的所有PDF文件。

**FTP下载**

wget ftp-url

wget --ftp-user=USERNAME --ftp-password=PASSWORD url

可以使用wget来完成ftp链接的下载。

使用wget匿名ftp下载：

wget ftp-url

使用wget用户名和密码认证的ftp下载：

wget --ftp-user=USERNAME --ftp-password=PASSWORD url

## firewall-cmd

Linux上新用的防火墙软件，跟iptables差不多的工具。

补充说明

firewall-cmd 是 firewalld的字符界面管理工具，firewalld是centos7的一大特性，最大的好处有两个：支持动态更新，不用重启服务；第二个就是加入了防火墙的“zone”概念。

firewalld跟iptables比起来至少有两大好处：

1. firewalld可以动态修改单条规则，而不需要像iptables那样，在修改了规则后必须得全部刷新才可以生效。
2. firewalld在使用上要比iptables人性化很多，即使不明白“五张表五条链”而且对TCP/IP协议也不理解也可以实现大部分功能。

firewalld自身并不具备防火墙的功能，而是和iptables一样需要通过内核的netfilter来实现，也就是说firewalld和 iptables一样，他们的作用都是用于维护规则，而真正使用规则干活的是内核的netfilter，只不过firewalld和iptables的结 构以及使用方法不一样罢了。

**命令格式**

firewall-cmd [选项 ... ]

**选项**

通用选项

-h, --help # 显示帮助信息；

-V, --version # 显示版本信息. （这个选项不能与其他选项组合）；

-q, --quiet # 不打印状态消息；

状态选项

--state # 显示firewalld的状态；

--reload # 不中断服务的重新加载；

--complete-reload # 中断所有连接的重新加载；

--runtime-to-permanent # 将当前防火墙的规则永久保存；

--check-config # 检查配置正确性；

日志选项

--get-log-denied # 获取记录被拒绝的日志；

--set-log-denied=<value> # 设置记录被拒绝的日志，只能为 'all','unicast','broadcast','multicast','off' 其中的一个；

**实例**

# 安装firewalld

yum install firewalld firewall-config

systemctl start firewalld # 启动

systemctl status firewalld # 或者 firewall-cmd --state 查看状态

systemctl disable firewalld # 停止

systemctl stop firewalld # 禁用

# 关闭服务的方法

# 你也可以关闭目前还不熟悉的FirewallD防火墙，而使用iptables，命令如下：

systemctl stop firewalld

systemctl disable firewalld

yum install iptables-services

systemctl start iptables

systemctl enable iptables

配置firewalld

firewall-cmd --version # 查看版本

firewall-cmd --help # 查看帮助

# 查看设置：

firewall-cmd --state # 显示状态

firewall-cmd --get-active-zones # 查看区域信息

firewall-cmd --get-zone-of-interface=eth0 # 查看指定接口所属区域

firewall-cmd --panic-on # 拒绝所有包

firewall-cmd --panic-off # 取消拒绝状态

firewall-cmd --query-panic # 查看是否拒绝

firewall-cmd --reload # 更新防火墙规则

firewall-cmd --complete-reload

# 两者的区别就是第一个无需断开连接，就是firewalld特性之一动态添加规则，第二个需要断开连接，类似重启服务

# 将接口添加到区域，默认接口都在public

firewall-cmd --zone=public --add-interface=eth0

# 永久生效再加上 --permanent 然后reload防火墙

# 设置默认接口区域，立即生效无需重启

firewall-cmd --set-default-zone=public

# 查看所有打开的端口：

firewall-cmd --zone=dmz --list-ports

# 加入一个端口到区域：

firewall-cmd --zone=dmz --add-port=8080/tcp

# 若要永久生效方法同上

# 打开一个服务，类似于将端口可视化，服务需要在配置文件中添加，/etc/firewalld 目录下有services文件夹，这个不详细说了，详情参考文档

firewall-cmd --zone=work --add-service=smtp

# 移除服务

firewall-cmd --zone=work --remove-service=smtp

# 显示支持的区域列表

firewall-cmd --get-zones

# 设置为家庭区域

firewall-cmd --set-default-zone=home

# 查看当前区域

firewall-cmd --get-active-zones

# 设置当前区域的接口

firewall-cmd --get-zone-of-interface=enp03s

# 显示所有公共区域（public）

firewall-cmd --zone=public --list-all

# 临时修改网络接口（enp0s3）为内部区域（internal）

firewall-cmd --zone=internal --change-interface=enp03s

# 永久修改网络接口enp03s为内部区域（internal）

firewall-cmd --permanent --zone=internal --change-interface=enp03s

服务管理

# 显示服务列表

Amanda, FTP, Samba和TFTP等最重要的服务已经被FirewallD提供相应的服务，可以使用如下命令查看：

firewall-cmd --get-services

# 允许SSH服务通过

firewall-cmd --enable service=ssh

# 禁止SSH服务通过

firewall-cmd --disable service=ssh

# 打开TCP的8080端口

firewall-cmd --enable ports=8080/tcp

# 临时允许Samba服务通过600秒

firewall-cmd --enable service=samba --timeout=600

# 显示当前服务

firewall-cmd --list-services

# 添加HTTP服务到内部区域（internal）

firewall-cmd --permanent --zone=internal --add-service=http

firewall-cmd --reload # 在不改变状态的条件下重新加载防火墙

端口管理

# 打开443/TCP端口

firewall-cmd --add-port=443/tcp

# 永久打开3690/TCP端口

firewall-cmd --permanent --add-port=3690/tcp

# 永久打开端口好像需要reload一下，临时打开好像不用，如果用了reload临时打开的端口就失效了

# 其它服务也可能是这样的，这个没有测试

firewall-cmd --reload

# 查看防火墙，添加的端口也可以看到

firewall-cmd --list-all

直接模式

# FirewallD包括一种直接模式，使用它可以完成一些工作，例如打开TCP协议的9999端口

firewall-cmd --direct -add-rule ipv4 filter INPUT 0 -p tcp --dport 9000 -j ACCEPT

firewall-cmd --reload

**自定义服务管理**

选项

（末尾带有 [P only] 的话表示该选项除了与（--permanent）之外，不能与其他选项一同使用！）

--new-service=<服务名> 新建一个自定义服务 [P only]

--new-service-from-file=<文件名> [--name=<服务名>]

从文件中读取配置用以新建一个自定义服务 [P only]

--delete-service=<服务名>

删除一个已存在的服务 [P only]

--load-service-defaults=<服务名>

Load icmptype default settings [P only]

--info-service=<服务名>

显示该服务的相关信息

--path-service=<服务名>

显示该服务的文件的相关路径 [P only]

--service=<服务名> --set-description=<描述>

给该服务设置描述信息 [P only]

--service=<服务名> --get-description

显示该服务的描述信息 [P only]

--service=<服务名> --set-short=<描述>

给该服务设置一个简短的描述 [P only]

--service=<服务名> --get-short

显示该服务的简短描述 [P only]

--service=<服务名> --add-port=<端口号>[-<端口号>]/<protocol>

给该服务添加一个新的端口(端口段) [P only]

--service=<服务名> --remove-port=<端口号>[-<端口号>]/<protocol>

从该服务上移除一个端口(端口段) [P only]

--service=<服务名> --query-port=<端口号>[-<端口号>]/<protocol>

查询该服务是否添加了某个端口(端口段) [P only]

--service=<服务名> --get-ports

显示该服务添加的所有端口 [P only]

--service=<服务名> --add-protocol=<protocol>

为该服务添加一个协议 [P only]

--service=<服务名> --remove-protocol=<protocol>

从该服务上移除一个协议 [P only]

--service=<服务名> --query-protocol=<protocol>

查询该服务是否添加了某个协议 [P only]

--service=<服务名> --get-protocols

显示该服务添加的所有协议 [P only]

--service=<服务名> --add-source-port=<端口号>[-<端口号>]/<protocol>

添加新的源端口(端口段)到该服务 [P only]

--service=<服务名> --remove-source-port=<端口号>[-<端口号>]/<protocol>

从该服务中删除源端口(端口段) [P only]

--service=<服务名> --query-source-port=<端口号>[-<端口号>]/<protocol>

查询该服务是否添加了某个源端口(端口段) [P only]

--service=<服务名> --get-source-ports

显示该服务所有源端口 [P only]

--service=<服务名> --add-module=<module>

为该服务添加一个模块 [P only]

--service=<服务名> --remove-module=<module>

为该服务移除一个模块 [P only]

--service=<服务名> --query-module=<module>

查询该服务是否添加了某个模块 [P only]

--service=<服务名> --get-modules

显示该服务添加的所有模块 [P only]

--service=<服务名> --set-destination=<ipv>:<address>[/<mask>]

Set destination for ipv to address in service [P only]

--service=<服务名> --remove-destination=<ipv>

Disable destination for ipv i service [P only]

--service=<服务名> --query-destination=<ipv>:<address>[/<mask>]

Return whether destination ipv is set for service [P only]

--service=<服务名> --get-destinations

List destinations in service [P only]

**控制端口 / 服务**

可以通过两种方式控制端口的开放，一种是指定端口号另一种是指定服务名。虽然开放 http 服务就是开放了 80 端口，但是还是不能通过端口号来关闭，也就是说通过指定服务名开放的就要通过指定服务名关闭；通过指定端口号开放的就要通过指定端口号关闭。还有一个要注意的就是指定端口的时候一定要指定是什么协议，tcp 还是 udp。知道这个之后以后就不用每次先关防火墙了，可以让防火墙真正的生效。

firewall-cmd --add-service=mysql # 开放mysql端口

firewall-cmd --remove-service=http # 阻止http端口

firewall-cmd --list-services # 查看开放的服务

firewall-cmd --add-port=3306/tcp # 开放通过tcp访问3306

firewall-cmd --remove-port=80tcp # 阻止通过tcp访问3306

firewall-cmd --add-port=233/udp # 开放通过udp访问233

firewall-cmd --list-ports # 查看开放的端口

伪装 IP

firewall-cmd --query-masquerade # 检查是否允许伪装IP

firewall-cmd --add-masquerade # 允许防火墙伪装IP

firewall-cmd --remove-masquerade# 禁止防火墙伪装IP

**端口转发**

端口转发可以将指定地址访问指定的端口时，将流量转发至指定地址的指定端口。转发的目的如果不指定 ip 的话就默认为本机，如果指定了 ip 却没指定端口，则默认使用来源端口。 如果配置好端口转发之后不能用，可以检查下面两个问题：

1. 比如我将 80 端口转发至 8080 端口，首先检查本地的 80 端口和目标的 8080 端口是否开放监听了
2. 其次检查是否允许伪装 IP，没允许的话要开启伪装 IP

firewall-cmd --add-forward-port=port=80:proto=tcp:toport=8080 # 将80端口的流量转发至8080

firewall-cmd --add-forward-port=port=80:proto=tcp:toaddr=192.168.0.1 # 将80端口的流量转发至192.168.0.1

firewall-cmd --add-forward-port=port=80:proto=tcp:toaddr=192.168.0.1:toport=8080 # 将80端口的流量转发至192.168.0.1的8080端口

1. 当我们想把某个端口隐藏起来的时候，就可以在防火墙上阻止那个端口访问，然后再开一个不规则的端口，之后配置防火墙的端口转发，将流量转发过去。
2. 端口转发还可以做流量分发，一个防火墙拖着好多台运行着不同服务的机器，然后用防火墙将不同端口的流量转发至不同机器。

## iptables

Linux上常用的防火墙软件

补充说明

**iptables命令** 是Linux上常用的防火墙软件，是netfilter项目的一部分。可以直接配置，也可以通过许多前端和图形界面配置。

**语法**

iptables(选项)(参数)

**选项**

-t, --table table 对指定的表 table 进行操作， table 必须是 raw， nat，filter，mangle 中的一个。如果不指定此选项，默认的是 filter 表。

# 通用匹配：源地址目标地址的匹配

-p：指定要匹配的数据包协议类型；

-s, --source [!] address[/mask] ：把指定的一个／一组地址作为源地址，按此规则进行过滤。当后面没有 mask 时，address 是一个地址，比如：192.168.1.1；当 mask 指定时，可以表示一组范围内的地址，比如：192.168.1.0/255.255.255.0。

-d, --destination [!] address[/mask] ：地址格式同上，但这里是指定地址为目的地址，按此进行过滤。

-i, --in-interface [!] <网络接口name> ：指定数据包的来自来自网络接口，比如最常见的 eth0 。注意：它只对 INPUT，FORWARD，PREROUTING 这三个链起作用。如果没有指定此选项， 说明可以来自任何一个网络接口。同前面类似，"!" 表示取反。

-o, --out-interface [!] <网络接口name> ：指定数据包出去的网络接口。只对 OUTPUT，FORWARD，POSTROUTING 三个链起作用。

# 查看管理命令

-L, --list [chain] 列出链 chain 上面的所有规则，如果没有指定链，列出表上所有链的所有规则。

# 规则管理命令

-A, --append chain rule-specification 在指定链 chain 的末尾插入指定的规则，也就是说，这条规则会被放到最后，最后才会被执行。规则是由后面的匹配来指定。

-I, --insert chain [rulenum] rule-specification 在链 chain 中的指定位置插入一条或多条规则。如果指定的规则号是1，则在链的头部插入。这也是默认的情况，如果没有指定规则号。

-D, --delete chain rule-specification -D, --delete chain rulenum 在指定的链 chain 中删除一个或多个指定规则。

-R num：Replays替换/修改第几条规则

# 链管理命令（这都是立即生效的）

-P, --policy chain target ：为指定的链 chain 设置策略 target。注意，只有内置的链才允许有策略，用户自定义的是不允许的。

-F, --flush [chain] 清空指定链 chain 上面的所有规则。如果没有指定链，清空该表上所有链的所有规则。

-N, --new-chain chain 用指定的名字创建一个新的链。

-X, --delete-chain [chain] ：删除指定的链，这个链必须没有被其它任何规则引用，而且这条上必须没有任何规则。如果没有指定链名，则会删除该表中所有非内置的链。

-E, --rename-chain old-chain new-chain ：用指定的新名字去重命名指定的链。这并不会对链内部照成任何影响。

-Z, --zero [chain] ：把指定链，或者表中的所有链上的所有计数器清零。

-j, --jump target <指定目标> ：即满足某条件时该执行什么样的动作。target 可以是内置的目标，比如 ACCEPT，也可以是用户自定义的链。

-h：显示帮助信息；

基本参数

| **参数** | **作用** |
| --- | --- |
| -P | 设置默认策略:iptables -P INPUT (DROP |
| -F | 清空规则链 |
| -L | 查看规则链 |
| -A | 在规则链的末尾加入新规则 |
| -I | num 在规则链的头部加入新规则 |
| -D | num 删除某一条规则 |
| -s | 匹配来源地址IP/MASK，加叹号"!"表示除这个IP外。 |
| -d | 匹配目标地址 |
| -i | 网卡名称 匹配从这块网卡流入的数据 |
| -o | 网卡名称 匹配从这块网卡流出的数据 |
| -p | 匹配协议,如tcp,udp,icmp |
| --dport num | 匹配目标端口号 |
| --sport num | 匹配来源端口号 |

**命令选项输入顺序**

iptables -t 表名 <-A/I/D/R> 规则链名 [规则号] <-i/o 网卡名> -p 协议名 <-s 源IP/源子网> --sport 源端口 <-d 目标IP/目标子网> --dport 目标端口 -j 动作

**工作机制**

规则链名包括(也被称为五个钩子函数（hook functions）)：

* **INPUT链** ：处理输入数据包。
* **OUTPUT链** ：处理输出数据包。
* **FORWARD链** ：处理转发数据包。
* **PREROUTING链** ：用于目标地址转换（DNAT）。
* **POSTOUTING链** ：用于源地址转换（SNAT）。

**防火墙的策略**

防火墙策略一般分为两种，一种叫通策略，一种叫堵策略，通策略，默认门是关着的，必须要定义谁能进。堵策略则是，大门是洞开的，但是你必须有身份认证，否则不能进。所以我们要定义，让进来的进来，让出去的出去，所以通，是要全通，而堵，则是要选择。当我们定义的策略的时候，要分别定义多条功能，其中：定义数据包中允许或者不允许的策略，filter过滤的功能，而定义地址转换的功能的则是nat选项。为了让这些功能交替工作，我们制定出了“表”这个定义，来定义、区分各种不同的工作功能和处理方式。

我们现在用的比较多个功能有3个：

1. filter 定义允许或者不允许的，只能做在3个链上：INPUT ，FORWARD ，OUTPUT
2. nat 定义地址转换的，也只能做在3个链上：PREROUTING ，OUTPUT ，POSTROUTING
3. mangle功能:修改报文原数据，是5个链都可以做：PREROUTING，INPUT，FORWARD，OUTPUT，POSTROUTING

我们修改报文原数据就是来修改TTL的。能够实现将数据包的元数据拆开，在里面做标记/修改内容的。而防火墙标记，其实就是靠mangle来实现的。

小扩展:

* 对于filter来讲一般只能做在3个链上：INPUT ，FORWARD ，OUTPUT
* 对于nat来讲一般也只能做在3个链上：PREROUTING ，OUTPUT ，POSTROUTING
* 而mangle则是5个链都可以做：PREROUTING，INPUT，FORWARD，OUTPUT，POSTROUTING

iptables/netfilter（这款软件）是工作在用户空间的，它可以让规则进行生效的，本身不是一种服务，而且规则是立即生效的。而我们iptables现在被做成了一个服务，可以进行启动，停止的。启动，则将规则直接生效，停止，则将规则撤销。

iptables还支持自己定义链。但是自己定义的链，必须是跟某种特定的链关联起来的。在一个关卡设定，指定当有数据的时候专门去找某个特定的链来处理，当那个链处理完之后，再返回。接着在特定的链中继续检查。

注意：规则的次序非常关键，谁的规则越严格，应该放的越靠前，而检查规则的时候，是按照从上往下的方式进行检查的。

表名包括：

* **raw** ：高级功能，如：网址过滤。
* **mangle** ：数据包修改（QOS），用于实现服务质量。
* **nat** ：地址转换，用于网关路由器。
* **filter** ：包过滤，用于防火墙规则。

动作包括：

* **ACCEPT** ：接收数据包。
* **DROP** ：丢弃数据包。
* **REDIRECT** ：重定向、映射、透明代理。
* **SNAT** ：源地址转换。
* **DNAT** ：目标地址转换。
* **MASQUERADE** ：IP伪装（NAT），用于ADSL。
* **LOG** ：日志记录。

┏╍╍╍╍╍╍╍╍╍╍╍╍╍╍╍┓

┌───────────────┐ ┃ Network ┃

│ table: filter │ ┗━━━━━━━┳━━━━━━━┛

│ chain: INPUT │◀────┐ │

└───────┬───────┘ │ ▼

│ │ ┌───────────────────┐

┌ ▼ ┐ │ │ table: nat │

│local process│ │ │ chain: PREROUTING │

└ ┘ │ └─────────┬─────────┘

│ │ │

▼ │ ▼ ┌─────────────────┐

┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅ │ ┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅ │table: nat │

Routing decision └───── outing decision ─────▶│chain: PREROUTING│

┅┅┅┅┅┅┅┅┅┳┅┅┅┅┅┅┅┅┅ ┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅ └────────┬────────┘

│ │

▼ │

┌───────────────┐ │

│ table: nat │ ┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅┅ │

│ chain: OUTPUT │ ┌─────▶ outing decision ◀──────────────┘

└───────┬───────┘ │ ┅┅┅┅┅┅┅┅┳┅┅┅┅┅┅┅┅

│ │ │

▼ │ ▼

┌───────────────┐ │ ┌────────────────────┐

│ table: filter │ │ │ chain: POSTROUTING │

│ chain: OUTPUT ├────┘ └──────────┬─────────┘

└───────────────┘ │

▼

┏╍╍╍╍╍╍╍╍╍╍╍╍╍╍╍┓

┃ Network ┃

┗━━━━━━━━━━━━━━━┛

**实例**

**空当前的所有规则和计数**

iptables -F # 清空所有的防火墙规则

iptables -X # 删除用户自定义的空链

iptables -Z # 清空计数

**配置允许ssh端口连接**

iptables -A INPUT -s 192.168.1.0/24 -p tcp --dport 22 -j ACCEPT

# 22为你的ssh端口， -s 192.168.1.0/24表示允许这个网段的机器来连接，其它网段的ip地址是登陆不了你的机器的。 -j ACCEPT表示接受这样的请求

**允许本地回环地址可以正常使用**

iptables -A INPUT -i lo -j ACCEPT

#本地圆环地址就是那个127.0.0.1，是本机上使用的,它进与出都设置为允许

iptables -A OUTPUT -o lo -j ACCEPT

**设置默认的规则**

iptables -P INPUT DROP # 配置默认的不让进

iptables -P FORWARD DROP # 默认的不允许转发

iptables -P OUTPUT ACCEPT # 默认的可以出去

**配置白名单**

iptables -A INPUT -p all -s 192.168.1.0/24 -j ACCEPT # 允许机房内网机器可以访问

iptables -A INPUT -p all -s 192.168.140.0/24 -j ACCEPT # 允许机房内网机器可以访问

iptables -A INPUT -p tcp -s 183.121.3.7 --dport 3380 -j ACCEPT # 允许183.121.3.7访问本机的3380端口

**开启相应的服务端口**

iptables -A INPUT -p tcp --dport 80 -j ACCEPT # 开启80端口，因为web对外都是这个端口

iptables -A INPUT -p icmp --icmp-type 8 -j ACCEPT # 允许被ping

iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT # 已经建立的连接得让它进来

**保存规则到配置文件中**

cp /etc/sysconfig/iptables /etc/sysconfig/iptables.bak # 任何改动之前先备份，请保持这一优秀的习惯

iptables-save > /etc/sysconfig/iptables

cat /etc/sysconfig/iptables

**列出已设置的规则**

iptables -L [-t 表名] [链名]

* 四个表名 raw，nat，filter，mangle
* 五个规则链名 INPUT、OUTPUT、FORWARD、PREROUTING、POSTROUTING
* filter表包含INPUT、OUTPUT、FORWARD三个规则链

iptables -L -t nat # 列出 nat 上面的所有规则

# ^ -t 参数指定，必须是 raw， nat，filter，mangle 中的一个

iptables -L -t nat --line-numbers # 规则带编号

iptables -L INPUT

iptables -L -nv # 查看，这个列表看起来更详细

**清除已有规则**

iptables -F INPUT # 清空指定链 INPUT 上面的所有规则

iptables -X INPUT # 删除指定的链，这个链必须没有被其它任何规则引用，而且这条上必须没有任何规则。

# 如果没有指定链名，则会删除该表中所有非内置的链。

iptables -Z INPUT # 把指定链，或者表中的所有链上的所有计数器清零。

**删除已添加的规则**

# 添加一条规则

iptables -A INPUT -s 192.168.1.5 -j DROP

将所有iptables以序号标记显示，执行：

iptables -L -n --line-numbers

比如要删除INPUT里序号为8的规则，执行：

iptables -D INPUT 8

**开放指定的端口**

iptables -A INPUT -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT #允许本地回环接口(即运行本机访问本机)

iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT #允许已建立的或相关连的通行

iptables -A OUTPUT -j ACCEPT #允许所有本机向外的访问

iptables -A INPUT -p tcp --dport 22 -j ACCEPT #允许访问22端口

iptables -A INPUT -p tcp --dport 80 -j ACCEPT #允许访问80端口

iptables -A INPUT -p tcp --dport 21 -j ACCEPT #允许ftp服务的21端口

iptables -A INPUT -p tcp --dport 20 -j ACCEPT #允许FTP服务的20端口

iptables -A INPUT -j reject #禁止其他未允许的规则访问

iptables -A FORWARD -j REJECT #禁止其他未允许的规则访问

**屏蔽IP**

iptables -A INPUT -p tcp -m tcp -s 192.168.0.8 -j DROP # 屏蔽恶意主机（比如，192.168.0.8

iptables -I INPUT -s 123.45.6.7 -j DROP #屏蔽单个IP的命令

iptables -I INPUT -s 123.0.0.0/8 -j DROP #封整个段即从123.0.0.1到123.255.255.254的命令

iptables -I INPUT -s 124.45.0.0/16 -j DROP #封IP段即从123.45.0.1到123.45.255.254的命令

iptables -I INPUT -s 123.45.6.0/24 -j DROP #封IP段即从123.45.6.1到123.45.6.254的命令是

**指定数据包出去的网络接口**

只对 OUTPUT，FORWARD，POSTROUTING 三个链起作用。

iptables -A FORWARD -o eth0

**查看已添加的规则**

iptables -L -n -v

Chain INPUT (policy DROP 48106 packets, 2690K bytes)

pkts bytes target prot opt in out source destination

5075 589K ACCEPT all -- lo \* 0.0.0.0/0 0.0.0.0/0

191K 90M ACCEPT tcp -- \* \* 0.0.0.0/0 0.0.0.0/0 tcp dpt:22

1499K 133M ACCEPT tcp -- \* \* 0.0.0.0/0 0.0.0.0/0 tcp dpt:80

4364K 6351M ACCEPT all -- \* \* 0.0.0.0/0 0.0.0.0/0 state RELATED,ESTABLISHED

6256 327K ACCEPT icmp -- \* \* 0.0.0.0/0 0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)

pkts bytes target prot opt in out source destination

Chain OUTPUT (policy ACCEPT 3382K packets, 1819M bytes)

pkts bytes target prot opt in out source destination

5075 589K ACCEPT all -- \* lo 0.0.0.0/0 0.0.0.0/0

**启动网络转发规则**

公网210.14.67.7让内网192.168.188.0/24上网

iptables -t nat -A POSTROUTING -s 192.168.188.0/24 -j SNAT --to-source 210.14.67.127

**端口映射**

本机的 2222 端口映射到内网 虚拟机的22 端口

iptables -t nat -A PREROUTING -d 210.14.67.127 -p tcp --dport 2222 -j DNAT --to-dest 192.168.188.115:22

**字符串匹配**

比如，我们要过滤所有TCP连接中的字符串test，一旦出现它我们就终止这个连接，我们可以这么做：

iptables -A INPUT -p tcp -m string --algo kmp --string "test" -j REJECT --reject-with tcp-reset

iptables -L

# Chain INPUT (policy ACCEPT)

# target prot opt source destination

# REJECT tcp -- anywhere anywhere STRING match "test" ALGO name kmp TO 65535 reject-with tcp-reset

#

# Chain FORWARD (policy ACCEPT)

# target prot opt source destination

#

# Chain OUTPUT (policy ACCEPT)

# target prot opt source destination

**阻止Windows蠕虫的攻击**

iptables -I INPUT -j DROP -p tcp -s 0.0.0.0/0 -m string --algo kmp --string "cmd.exe"

**防止SYN洪水攻击**

iptables -A INPUT -p tcp --syn -m limit --limit 5/second -j ACCEPT

更多实例

用iptables搭建一套强大的安全防护盾 <http://www.imooc.com/learn/389>

iptables: linux 下应用层防火墙工具

iptables 5链: 对应 Hook point netfilter: linux 操作系统核心层内部的一个数据包处理模块 Hook point: 数据包在 netfilter 中的挂载点; PRE\_ROUTING / INPUT / OUTPUT / FORWARD / POST\_ROUTING

iptables & netfilter

iptables 4表5链

iptables rules

* 4表

**filter**: 访问控制 / 规则匹配 **nat**: 地址转发 mangle / raw

* 规则

数据访问控制: ACCEPT / DROP / REJECT 数据包改写(nat -> 地址转换): snat / dnat 信息记录: log

使用场景实例

* 场景一

开放 tcp 10-22/80 端口 开放 icmp 其他未被允许的端口禁止访问

存在的问题: 本机无法访问本机; 本机无法访问其他主机

* 场景二

ftp: 默认被动模式(服务器产生随机端口告诉客户端, 客户端主动连接这个端口拉取数据) vsftpd: 使 ftp 支持主动模式(客户端产生随机端口通知服务器, 服务器主动连接这个端口发送数据)

* 场景三

允许外网访问: web http -> 80/tcp; https -> 443/tcp mail smtp -> 25/tcp; smtps -> 465/tcp pop3 -> 110/tcp; pop3s -> 995/tcp imap -> 143/tcp

内部使用: file nfs -> 123/udp samba -> 137/138/139/445/tcp ftp -> 20/21/tcp remote ssh -> 22/tcp sql mysql -> 3306/tcp oracle -> 1521/tcp

* 场景四

nat 转发

* 场景五

防CC攻击

iptables -L -F -A -D # list flush append delete

# 场景一

iptables -I INPUT -p tcp --dport 80 -j ACCEPT # 允许 tcp 80 端口

iptables -I INPUT -p tcp --dport 10:22 -j ACCEPT # 允许 tcp 10-22 端口

iptables -I INPUT -p icmp -j ACCEPT # 允许 icmp

iptables -A INPUT -j REJECT # 添加一条规则, 不允许所有

# 优化场景一

iptables -I INPUT -i lo -j ACCEPT # 允许本机访问

iptables -I INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT # 允许访问外网

iptables -I INPUT -p tcp --dport 80 -s 10.10.188.233 -j ACCEPT # 只允许固定ip访问80

# 场景二

vi /etc/vsftpd/vsftpd.conf # 使用 vsftpd 开启 ftp 主动模式

port\_enable=yes

connect\_from\_port\_20=YES

iptables -I INPUT -p tcp --dport 21 -j ACCEPT

vi /etc/vsftpd/vsftpd.conf # 建议使用 ftp 被动模式

pasv\_min\_port=50000

pasv\_max\_port=60000

iptables -I INPUT -p tcp --dport 21 -j ACCEPT

iptables -I INPUT -p tcp --dport 50000:60000 -j ACCEPT

# 还可以使用 iptables 模块追踪来自动开发对应的端口

# 场景三

iptables -I INPUT -i lo -j ACCEPT # 允许本机访问

iptables -I INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT # 允许访问外网

iptables -I INPUT -s 10.10.155.0/24 -j ACCEPT # 允许内网访问

iptables -I INPUT -p tcp -m multiport --dports 80,1723 -j ACCEPT # 允许端口, 80 -> http, 1723 -> vpn

iptables -A INPUT -j REJECT # 添加一条规则, 不允许所有

iptables-save # 保存设置到配置文件

# 场景四

iptables -t nat -L # 查看 nat 配置

iptables -t nat -A POST\_ROUTING -s 10.10.177.0/24 -j SNAT --to 10.10.188.232 # SNAT

vi /etc/sysconfig/network # 配置网关

iptables -t nat -A POST\_ROUTING -d 10.10.188.232 -p tcp --dport 80 -j DNAT --to 10.10.177.232:80 # DNAT

#场景五

iptables -I INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 100 -j REJECT # 限制并发连接访问数

iptables -I INPUT -m limit --limit 3/hour --limit-burst 10 -j ACCEPT # limit模块; --limit-burst 默认为5

# Linux中间件安装

## 1.CentOS6

### 1.1 操作系统配置

#### 1.1.1 服务器配置DNS

若提供的服务器可以正常解析外网地址，则可以忽略此步骤。

若提供的服务器，需要配置DNS才可以连接外网或软件源，配置如下：

配置命令：echo "nameserver 学校DNS或者是8.8.8.8或者是114.114.114.114">>/etc/resolv.conf

#### 1.1.2 配置yum源

以163源为例：

1）下载repo文件

wget http://mirrors.163.com/.help/CentOS6-Base-163.repo

2）备份并替换系统的repo文件

cp CentOS6-Base-163.repo /etc/yum.repos.d/

cd /etc/yum.repos.d/

mv CentOS-Base.repo CentOS-Base.repo.bak

mv CentOS6-Base-163.repo CentOS-Base.repo

3）执行yum源更新命令

yum clean all

yum makecache

配置完毕。

如果服务器不能连接外部软件源，则告诉该学校负责的老师将学校的内部源提供下，配置在/etc/yum.repos.d/中。

#### 1.1.3 更新操作系统

使用命令：

yum update -y

更新完成后，重启服务器。

#### 1.1.4 创建infoplus用户

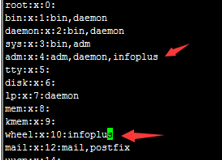
命令：useradd infoplus

命令：passwd infoplus

需要记录密码，密码建议为随机8—16位数，包括英文字母大小写、数字和特殊字符，需要注意键盘是英文模式输入。

将infoplus用户加入到adm和wheel组中，如下图：

vim /etc/group



然后修改/etc/sudoers 为 NOPASSWD，将前面#号去掉，如下图：



#### 1.1.5 配置主机名

修改/etc/sysconfig/network文件，添加主机名称，设置主机名为*所属业务名称*

例如，上海大学流程平台大厅服务器，名称为：

ip2\_taskcenter\_shu

:wq！ #保存退出

修改hosts文件，在每行后面添加主机名：

vi /etc/hosts

举例：

127.0.0.1 ip2\_taskcenter\_shu #添加ip2\_taskcenter\_shu

::1 ip2\_taskcenter\_shu #添加ip2\_taskcenter\_shu

:wq！ #保存退出

#### 1.1.6 禁用SELinux

SELinux始终保持关闭状态，使用如下命令查看SELinux状态

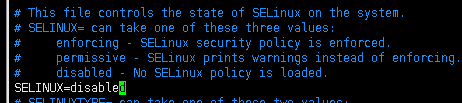
getenforce



此时为开启状态

第一种方法：

vim /etc/sysconfig/selinux #编辑SELinux状态



将SELINUX=enforce 改为SELINUX=disabled

重启服务器生效。

•第二种方法：

sed –i ‘s/SELINUX=enforcing/SELINUX=disabled/g’ /etc/sysconfig/selinux

重启服务器生效。

如果想立即生效（如果想临时性的改变）：

setenforce 0

**拓展说明：**

setenforce 1 设置SELinux 成为enforcing模式

setenforce 0 设置SELinux 成为permissive模式

查看状态 getenforce

#### 1.1.7 服务器修改文件打开数Openfile

1）先用ulimit -n或ulimit -a命令，发现文件打开数是1024

# ulimit -n

1024

2）用ulimit -n 40960把文件打开数改为最大值40960，再用ulimit -n验证发现修改成功

# ulimit -n 40960

# ulimit -n

40960

需要注意，有的centos6系统重启后，查看ulimit -n参数，又变回1024，则需要按照如下方法方法修改：

将ulimit 值添加到/etc/profile文件中（适用于有root权限登录的系统）

为了每次系统重新启动时，都可以获取更大的ulimit值，将ulimit 加入到/etc/profile 文件底部。

echo ulimit -n 40960 >>/etc/profile

source /etc/profile #加载修改后的profile

ulimit -n #显示40960

**拓展说明：**上述配置完成后，退出或重启系统，ulimit的值还是1024，这是为什么呢？

关键的原因是你登录的用户是什么身份，是不是root用户，由于服务器的root用户权限很大，一般是不能用来登录的，都是通过自己本人的登录权限进行登录，并通过sudo方式切换到root用户下进行工作。 用户登录的时候执行sh脚本的顺序：

/etc/profile.d/file

/etc/profile

/etc/bashrc

/mingjie/.bashrc

/mingjie/.bash\_profile

由于ulimit -n的脚本命令加载在第二部分，用户登录时由于权限原因在第二步还不能完成ulimit的修改，所以ulimit的值还是系统默认的1024。

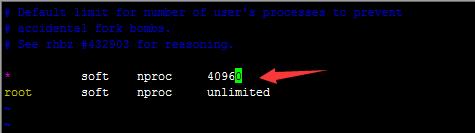
解决办法：

3）在/etc/security/limits.conf最后增加如下两行记录

\* soft nofile 40960

\* hard nofile 40960

4）进入/etc/security/limits.d目录中，默认就一个文件，文件名一般为conf为结尾，修改配置参数为40960，如下图：



#### 1.1.8 记录所有登录用户操作历史

需求：记录登陆后的IP地址和某用户名所操作的历史记录

实现方法：注意如下脚本中记录日志会存放在/opt/user\_log目录中。

##### 1）配置/etc/profile

USER\_IP=`who -u am i 2>/dev/null| awk '{print $NF}'|sed -e 's/[()]//g'`

if [ "$USER\_IP" = "" ]

then

USER\_IP=`hostname`

fi

if [ ! -d /opt/user\_log ]

then

mkdir /opt/user\_log

chmod -R 777 /opt/user\_log

fi

if [ ! -d /opt/user\_log/${LOGNAME} ]

then

mkdir /opt/user\_log/${LOGNAME}

chmod 300 /opt/user\_log/${LOGNAME}

fi

export HISTSIZE=4096

DT=`date "+%Y-%m-%d\_%H:%M:%S"`

export HISTFILE="/opt/user\_log/${LOGNAME}/${USER\_IP}-dbasky.$DT"

chmod 600 /opt/user\_log/${LOGNAME}/\*dbasky\* 2>/dev/null

##### 2）生效环境变量

source /etc/profile

##### 3）查看记录

说明：该脚本在系统的/opt新建个user\_log目录，记录所有登陆过系统的用户和IP地址（文件名），每当用户登录/退出会创建相应的文件，该文件保存这段用户登录时期内操作历史，可以用这个方法来监测系统的安全性。

#### 1.1.9 防火墙设置iptables

1）开通业务及管理端口

根据业务需求，调整服务器开放端口。

开启端口，修改vim /etc/sysconfig/iptables文件，来开启使用端口，centos系统默认开启22端口。

添加语句，分别开启80，443，8080，8009，3306 端口，根据不同业务服务器，开放不同端口。

-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT

-A INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT

-A INPUT -m state --state NEW -m tcp -p tcp --dport 8080 -j ACCEPT

-A INPUT -m state --state NEW -m tcp -p tcp --dport 8009 -j ACCEPT

-A INPUT -m state --state NEW -m tcp -p tcp --dport 3306 -j ACCEPT

保存退出，然后重启iptables服务并设置开机自启

service iptables restart

chkconfig iptables on #设置开机启动防火墙

关闭端口，则可以将添加的规则进行删除，然后重新加载防火墙规则：

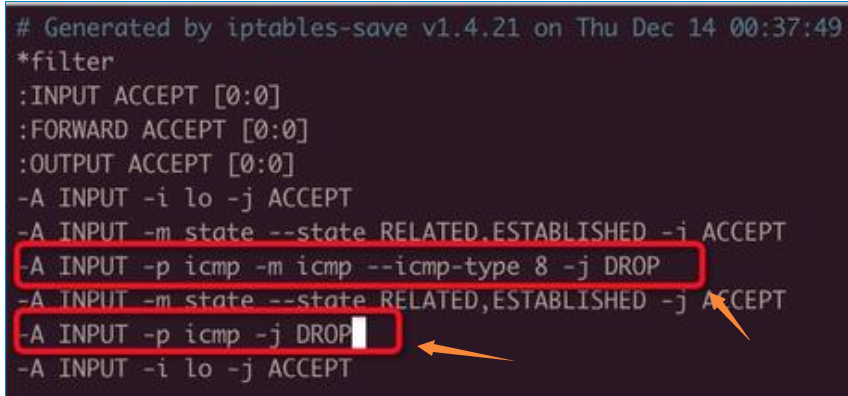
service iptables reload

2）禁用服务器ICMP回显响应

添加如下截图的两条规则：

-A INPUT -p icmp -m icmp --icmp-type 8 -j DROP

-A INPUT -p icmp -j DROP



**拓展说明：**

限定IP网段访问：

-A INPUT -m state --state NEW -m tcp -p tcp -s IP地址/24 --dport 3306 -j ACCEPT //限定IP网段访问3306端口

限定IP访问：

-A INPUT -m state --state NEW -m tcp -p tcp -s IP地址 --dport 3306 -j ACCEPT //限定IP访问3306端口

查看规则：

iptables -L -n

#### 1.1.10 操作系统内核参数优化

对TCP连接数优化：

[root@hadoop13 ~]# vim /etc/sysctl.conf

net.ipv4.tcp\_max\_tw\_buckets=360000

net.ipv4.tcp\_max\_syn\_backlog=8192

net.ipv4.tcp\_syncookies=1

net.ipv4.tcp\_tw\_reuse=1

net.ipv4.tcp\_tw\_recycle=1

net.ipv4.tcp\_fin\_timeout=10

net.ipv4.tcp\_sack=0

kernel.pid\_max=1000000

# 让参数生效

[root@hadoop13 ~]# sysctl -p

**拓展说明，具体每个参数的含义：**

net.ipv4.tcp\_max\_tw\_buckets = 360000 #改值的默认值是18w

net.ipv4.tcp\_max\_syn\_backlog = 8192 #默认为1024，等待连接的网络连接数

net.ipv4.tcp\_syncookies = 1 #表示开启SYN Cookies。当出现SYN等待队列溢出时，启用cookies来处理，可防范少量SYN攻击，默认为0，表示关闭；

net.ipv4.tcp\_tw\_reuse = 1 #表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭；

net.ipv4.tcp\_tw\_recycle = 1 #表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭。

net.ipv4.tcp\_fin\_timeout = 10s #修改系統默认的TIMEOUT时间,默认是60s.

net.ipv4.tcp\_sack=0 #禁用内核 SACK 配置防范漏洞利用；

#### 1.1.11 时间和时区设置

##### 1）有外网情况下，使用ntpdate做时间同步

注：需要咨询学校用户是否有时间服务器，如果有，则需要使用学校时间服务器进行时间同步；如果学校没有，我们就是用使用China ntp服务器，若不准确可以更换其它时间服务器。

ntpdate cn.pool.ntp.org（注意：同步时ntpd需要关闭）



同步之后将时间同步写入定时脚本

crontab -e

添加如下定时语句：

\*/1 \* \* \* \* /sbin/ntpdate cn.pool.ntp.org;hwclock -w //每天1分钟执行一次

保存退出并重新载入

service crond reload

##### 2）没有外网的情况下，使用手动更改时间

**注：需要咨询学校用户是否有时间服务器，如果有，则需要使用学校时间服务器进行时间同步；如果学校没有，则需要手动校准日期和时间。**

#date 查看系统时间



#date -s 修改日期



#date -s 修改时间



修改完后执行：clock -w ,强制将时间写入COMS！

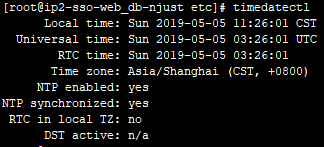


修改时区：（将Asia/shanghai-上海时区写入当前时区）

#cp -f /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

提示是否覆盖,输入Y回车

然后执行timedatectl命令，查看时区和时间：Asia/Shanghai（CST,中国时区+0800）



需要注意：有的系统会将/etc/localtime文件做成软连接，如下截图，这时在做时区覆盖后，不会生效，需要重新替换此链接。



修改时区软连接：

先将原有问题的localtime移走，命令为mv /etc/localtime /tmp

然后将上海的时区做软连接，命令为ln -s /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

#### 1.1.12 安全设置

**1）注释掉系统不需要的用户**

注意：不建议直接删除，当你需要某个用户时，自己重新添加会很麻烦。

cp /etc/passwd /etc/passwdbak #修改之前先备份

vi /etc/passwd #编辑用户，在前面加上#注释掉此行

#adm:x:3:4:adm:/var/adm:/sbin/nologin

#lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

#sync:x:5:0:sync:/sbin:/bin/sync

#shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

#halt:x:7:0:halt:/sbin:/sbin/halt

#uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin

#operator:x:11:0:operator:/root:/sbin/nologin

#games:x:12:100:games:/usr/games:/sbin/nologin

#gopher:x:13:30:gopher:/var/gopher:/sbin/nologin

#ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin #注释掉ftp匿名账号

cp /etc/group /etc/groupbak #修改之前先备份

vi /etc/group #编辑用户组，在前面加上#注释掉此行

#adm:x:4:root,adm,daemon

#lp:x:7:daemon,lp

#uucp:x:14:uucp

#games:x:20:

#dip:x:40:

**2）关闭系统不需要的服务**

service acpid stop chkconfig acpid off #停止服务，取消开机启动 #电源进阶设定，常用在 Laptop 上

service autofs stop chkconfig autofs off #停用自动挂载档桉系统与週边装置

service bluetooth stop chkconfig bluetooth off #停用Bluetooth蓝芽

service cpuspeed stop chkconfig cpuspeed off #停用控制CPU速度主要用来省电

service cups stop chkconfig cups off #停用 Common UNIX Printing System 使系统支援印表机

service ip6tables stop chkconfig ip6tables off #禁止IPv6

**3）禁止使用Ctrl+Alt+Del快捷键重启服务器**

cp /etc/inittab /etc/inittabbak

vi /etc/inittab #注释掉下面这一行

#ca::ctrlaltdel:/sbin/shutdown -t3 -r now

或者

vi /etc/init/control-alt-delete.conf

#注释掉

#exec /sbin/shutdown -r now "Control-Alt-Deletepressed"

**4）使用yum update更新系统时不升级内核，只更新软件包**

由于系统与硬件的兼容性问题，有可能升级内核后导致服务器不能正常启动，这是非常可怕的，没有特别的需要，建议不要随意升级内核。

cp /etc/yum.conf /etc/yum.confbak

1、修改yum的配置文件 vi /etc/yum.conf 在[main]的最后添加 exclude=kernel\*

2、直接在yum的命令后面加上如下的参数：

yum --exclude=kernel\* update

查看系统版本 cat /etc/issue

查看内核版本 uname -a

**5）关闭Centos自动更新**

chkconfig --list yum-updatesd #显示当前系统状态

yum-updatesd 0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭

service yum-updatesd stop #关闭 开启参数为start

停止 yum-updatesd： [确定]

service yum-updatesd status #查看是否关闭

yum-updatesd 已停

chkconfig --level 35 yum-updatesd off #禁止开启启动（系统模式为3、5）

chkconfig yum-updatesd off #禁止开启启动（所有启动模式全部禁止）

chkconfig --list yum-updatesd #显示当前系统状态

yum-updatesd 0:关闭 1:关闭 2:启用 3:关闭 4:启用 5:关闭 6:关闭

**6）linux限制登录失败次数并锁定**

在/etc/pam.d/login和/etc/pam.d/sshd第二行添加

auth required pam\_tally2.so deny=6 unlock\_time=180 even\_deny\_root root\_unlock\_time=180

登录失败5次锁定180秒，根据需要设置是否包括root

**7）拓展说明**

Linux定期做日志安全检查

将日志移动到专用的日志服务器里，这可避免入侵者轻易的改动本地日志。下面是常见linux的默认日志文件及其用处：

/var/log/message – 记录系统日志或当前活动日志。

/var/log/auth.log – 身份认证日志。

/var/log/cron – Crond 日志 (cron 任务).

/var/log/maillog – 邮件服务器日志。

/var/log/secure – 认证日志。

/var/log/wtmp 历史登录、注销、启动、停机日志和，lastb命令可以查看登录失败的用户

/var/run/utmp 当前登录的用户信息日志，w、who命令的信息便来源与此

/var/log/yum.log Yum 日志。

Linux安装logwatch

Logwatch是使用 Perl 开发的一个日志分析工具。能够对Linux 的日志文件进行分析，并自动发送mail给相关处理人员，可定制需求。

Logwatch的mail功能是借助宿主系统自带的 mail server 发邮件的，所以系统需安装mail server , 如sendmail,postfix,Qmail等。

#### 1.1.13 安装VMtools

安装命令参考：yum install open-vm-tools.x86\_64

#### 1.1.14 设置系统字符集

**01、查看Linux当前语言**

echo $LANG

C #C代表英文环境 ASCII 编码格式,通杀很多问题（乱码）

**02、查看系统支持的字符集**

locale -a #系统可以支持的语言

locale #当前系统的语言环境

**03、修改系统环境字符集**

export LANG=zh\_CN.utf8

LANG #系统主要语系

LC\_ALL #系统整体语系

vim /etc/profile

export LANG=zh\_CN.utf8

source /etc/profile

[root@t1 ~]# cat /etc/sysconfig/i18n #修改文件，也是一种方案

LANG="en\_US.UTF-8"

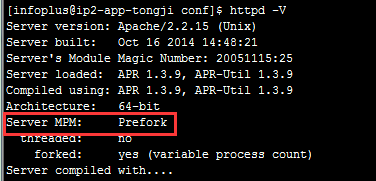
SYSFONT="latarcyrheb-sun16" #系统字体

### 1.2 Apache（httpd）安装配置

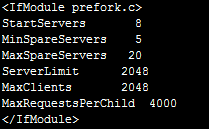
#### 1.2.1 Apache安装

yum install httpd mod\_ssl openssl

使用httpd -V 命令查看下server mpm的处理模式和版本，如下图所示：mpm所用模式是prefork，如果mpm所用的模式是其它模式（worker、event），则修改http.conf文件中所对应的代码。



Server MPM为Prefork模式，则修改/etc/httpd/conf目录中httpd.conf文件，修改ServerLimit，MaxClients和MaxRequestsPerChild，按照如下截图内容进行修改：



在httpd.conf中增加如下配置：

vim /etc/httpd/conf/httpd.conf 加入一句 ServerName localhost:80

修改apache 配置文件： vim /etc/httpd/conf/httpd.conf

Timeout 60参数修改成： Timeout 720

开启httpd服务：

service httpd start

设置开机自启：

chkconfig httpd on

如下代理配置，需根据实际需求调整。

在/etc/httpd/conf.d目录中新建infoplus.conf文件：

vim infoplus.conf

文件内容如下：（ centos版本）

<VirtualHost \*:80>

ServerAdmin i@marstone.net

DocumentRoot /var/www

ServerName 域名或IP地址

ServerAlias 域名或IP地址

LogFormat "%h %u %l %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %T %D" infoplus

CustomLog "| /usr/sbin/rotatelogs /var/log/httpd/infoplus-access-combined-ssl-log-%Y-%m-%d 86400 480" infoplus

　　　 ErrorLog "| /usr/sbin/rotatelogs /var/log/httpd/infoplus-error-ssl-log-%Y-%m-%d 86400 480"

RedirectMatch 302 ^/$ /taskcenter/

Header set P3P 'policyref="http://aitpro.local/w3c/p3p.xml", CP="NOI DSP COR NID CUR ADM DEV OUR BUS"'

<Location /infoplus>

ProxyPass ajp://engine服务器地址:8009/infoplus

</Location>

<Location /taskcenter>

ProxyPass ajp://localhost:8009/taskcenter

</Location>

<Location /taskcenter-beta>

ProxyPass ajp://localhost:8009/taskcenter-beta

</Location>

<Location /authorize>

ProxyPass ajp://localhost:8009/authorize

</Location>

<Location /dashboard>

ProxyPass ajp://localhost:8009/dashboard

</Location>

<Location /file>

ProxyPass ajp://API服务器地址:8009/fileapi/file

</Location>

<Location /manager-app/manager>

ProxyPass ajp://localhost:8009/manager

ProxyPassReverse ajp://localhost:8009/manager

</Location>

<Location /manager-engine/manager>

ProxyPass ajp://engine服务器地址:8009/manager

ProxyPassReverse ajp://engine服务器地址:8009/manager

</Location>

<Location /manager-api/manager>

ProxyPass ajp://API服务器地址:8009/manager

ProxyPassReverse ajp://API服务器地址:8009/manager

</Location>

</VirtualHost>

**需要注意，配置fileapi的location节，有特殊情况需要允许方法配置：**

<Location /file>  
   <Limit GET POST PUT DELETE OPTIONS>  
      Require all granted  
   </Limit>  
   ProxyPass ajp://IP地址:8009/fileapi/file  
</Location>

#### 1.2.2 Apache(httpd)编译安装

方法暂略

#### 1.2.3 Apache2性能优化

**拓展说明：**

apache2三种模式，从2.0开始，apache引入了MPM（Multi-Processing Module，多进程处理模块）。MPM有prefork, worker和event(在2.4版本中稳定发布)模式，三种模式拥有不同的特点和性能。

**prefork MPM**

prefork是比较古老而又稳定的apache模式，特点是每个进程都是单线程，在一个时间点只能处理一个连接，需要启动大量的进程来处理高并发的请求。由于是单线程进程，因而无须考虑线程安全的问题，可以使用非线程安全的库，例如mod\_php。

优点是成熟稳定，缺点是比较消耗内存，而且并发支持受限于进程数量，对高并发支持稍差。

**worker MPM**

worker同样使用多个进程，但每个进程又拥有多个线程，每个线程处理一个连接。由于线程是轻量级的，因而具有较高的并发性，同时，多个进程又获得了一定的稳定性。

worker模式特点是占用内存少，并发性比较高，缺点是必须考虑线程安全。如果使用了keep-alive方式，一个线程可能会被一直保持一个连接，但中间没有请求，直到超时。如果有多个线程被这样占据，在高并发场景下同样会出现无线程可用的情形。

**event MPM**

event模式是在2.4版本中才稳定发布的模式，它在worker的基础上，解决了keep-alive连接不能释放的问题。event MPM中，会有一个专门的线程来管理这些keep-alive类型的线程，当有真实请求过来的时候，将请求传递给服务线程，执行完毕后，又允许它释放。这样增强了高并发场景下的请求处理能力。

查看Apache工作模式，CentOS系统命令：

httpd -V |grep -i mpm

Ubuntu系统命令：

apachectl -V | grep -i mpm

需要注意apache（httpd）因版本的不同，而优化参数不同。

优化参数举例说明：

apache2的默认配置文件位于/etc/apache2/apache2.conf，不同模式对应的配置文件在/etc/apache2/mods-available/下，有mpm\_prefork.conf， mpm\_worker.conf和mpm\_event.conf。

##### 1）prefork MPM性能调优

mpm\_prefork.conf文件内容如下：

<IfModule mpm\_prefork\_module>

StartServers 10 # 启动时进程数

MinSpareServers 5 # 最小空闲进程数

MaxSpareServers 10 # 最大空闲进程数

MaxRequestWorkers 100 # 最大并发进程数

MaxConnectionsPerChild 10000 # 最大连接数限制

</IfModule>

各个指令的含义：

StartServers : apache2启动时创建的服务进程数量。

MinSpareServers：最小空闲进程数量，空闲进程指的是没有处理请求的进程。

MaxSpareServers : 最大空闲进程数量。

MaxRequestWorkers : 最大同时处理请求的进程数量，也是最大的同时连接数，表示了apache的最大请求并发能力，超过该数目后的请求，将排队。

MaxConnectionsPerChild : 进程生命周期内，处理的最大请求数目。达到该数目后，进程将死掉。如果设置为0，表示没有限制。该参数的意义在于，避免了可能存在的内存泄露带来的系统问题。

通过上面的介绍可以发现，prefork模式下，影响并发性能最重要的参数就是MaxRequestWorkers，它决定了apache的并发处理能力。但是这个参数不是越大越好，因为如果超出了系统硬件能力，机器会卡死。

如果确定合适的MaxRequestWorkers呢？

首先，通过top命令查看apache进程占用的资源，主要看%CPU和%MEM这两个指标，例如，每个进程的CPU占用率不超过1%，每个进程的内存占用率不超过2%，考虑内存限制，比较合适的apache进程数量为50个。

然后，逐步测试最大值。通过观测得来的CPU和内存的指标有一定的误差，一般可以适当调节这个数值，例如调到1.5或者2倍，再通过峰值场景下的机器是否卡顿来判断是继续上调还是下调。

调整完参数后，一般需要重启apache。

service apache2 restart

##### 2）worker MPM性能调优

mpm\_worker.conf文件内容如下：

<IfModule mpm\_worker\_module>

StartServers 2 # 启动时进程数

MinSpareThreads 25 # 最小空闲线程数

MaxSpareThreads 75 # 最大空闲线程数

ThreadLimit 64 # 每个进程可以启动的线程数量上限值

ThreadsPerChild 25 # 每个进程可以启动的线程数量

MaxRequestWorkers 400 # 线程数量最大值

MaxConnectionsPerChild 0 # 最大连接数限制

</IfModule>

##### 3）event MPM性能调优

httpd 2.4.35以后 增加配置

<IfModule mpm\_event\_module>

ServerLimit 1000

StartServers 20

MinSpareThreads 25

MaxSpareThreads 1200

ThreadsPerChild 50

MaxRequestWorkers 2000

MaxConnectionsPerChild 0

</IfModule>

Group apache

Timeout 720

Keepalive on

KeepAliveTimeout 5

TraceEnable off

#### 1.2.4 Apache2安全优化

##### 1）Apache隐藏版本号

**apache在默认404、403提示页面中隐藏apache版本和系统信息**



要隐藏红色部分的apache版本号和系统信息，在apache配置文件中（linux系统中是httpd.conf），找到如下两行：

ServerTokens OS

ServerSignature On

对应修改为：

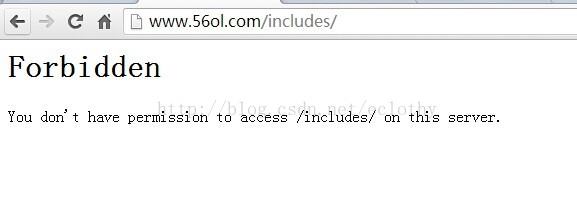
ServerTokens Prod

ServerSignature Off

然后执行：

service httpd restart

重启apache即可，修改后如图所示：



##### 2）Apache禁用默认icons目录

进入apache的配置文件 httpd.conf

vim /etc/httpd/conf/httpd.conf

找到：

Options Indexes FollowSymLinks

修改为：

Options FollowSymLinks

**详细举例：**

<Directory "/var/www/icons">

    Options Indexes MultiViews FollowSymLinks

    AllowOverride None

    Order allow,deny

    Allow from all

</Directory>

修改为：

<Directory "/var/www/icons">

    Options FollowSymLinks

    AllowOverride None

    Order allow,deny

    Allow from all

</Directory>

##### 3）Apache禁用默认的html目录列表

在/etc/httpd/conf/httpd.conf文件中，修改如下

<Directory "/var/www/html">

# Options Indexes FollowSymLinks ##注释该行

AllowOverride None

Require all granted

</Directory>

然后重启apache

service httpd restart

##### 4）移除apache欢迎页面

mv /etc/httpd/conf.d/welcome.conf /etc/httpd/conf.d/welcome.conf.bk

### 1.3 JDK安装配置

1、 检查系统是否自带有jdk  （不用卸载老的，直接安装新的也可以）

[root@VM\_0\_11\_centos ~]# rpm -qa |grep java

[root@VM\_0\_11\_centos ~]# rpm -qa |grep jdk

[root@VM\_0\_11\_centos ~]# rpm -qa |grep gcj

     如果有信息输出则需要卸载掉原来的jdk版本，用rpm -e ...  例如： 卸载掉java

[root@VM\_0\_11\_centos ~]# rpm -e java

2、 yum查找 java 1.8+ 的信息

[root@VM\_0\_11\_centos ~]# yum list java-1.8\*

Loaded plugins: fastestmirror, security

Loading mirror speeds from cached hostfile

\* webtatic: [uk.repo.webtatic.com](http://uk.repo.webtatic.com/)

nginx                                                                                                       | 2.9 kB     00:00

Available Packages

java-1.8.0-openjdk.x86\_64                                                1:1.8.0.171-8.b10.el6\_9                                 os

java-1.8.0-openjdk-debug.x86\_64                                          1:1.8.0.171-8.b10.el6\_9                                 os

java-1.8.0-openjdk-demo.x86\_64                                           1:1.8.0.171-8.b10.el6\_9                                 os

java-1.8.0-openjdk-demo-debug.x86\_64                                     1:1.8.0.171-8.b10.el6\_9                                 os

java-1.8.0-openjdk-devel.x86\_64                                          1:1.8.0.171-8.b10.el6\_9                                 os

java-1.8.0-openjdk-devel-debug.x86\_64                                    1:1.8.0.171-8.b10.el6\_9                                 os

java-1.8.0-openjdk-headless.x86\_64                                       1:1.8.0.171-8.b10.el6\_9                                 os

java-1.8.0-openjdk-headless-debug.x86\_64                                 1:1.8.0.171-8.b10.el6\_9                                 os

java-1.8.0-openjdk-javadoc.noarch                                        1:1.8.0.171-8.b10.el6\_9                                 os

java-1.8.0-openjdk-javadoc-debug.noarch                                  1:1.8.0.171-8.b10.el6\_9                                 os

java-1.8.0-openjdk-src.x86\_64                                            1:1.8.0.171-8.b10.el6\_9                                 os

java-1.8.0-openjdk-src-debug.x86\_64                                      1:1.8.0.171-8.b10.el6\_9                                 os

 3、 安装jdk1.8.0的所有文件

[root@VM\_0\_11\_centos ~]# yum install java-1.8.0-openjdk\* -y

4、 检查是否安装成功

[root@VM\_0\_11\_centos ~]# java -version

openjdk version "1.8.0\_171"

OpenJDK Runtime Environment (build 1.8.0\_171-b10)

OpenJDK 64-Bit Server VM (build 25.171-b10, mixed mode)

### 1.4 Tomcat安装配置

#### 1.4.1 Tomcat7安装与配置

安装命令：

yum install tomcat tomcat-admin-webapps

tomcat开机自启动命令：

systemctl enable tomcat.service

启动tomcat命令：

/bin/systemctl start tomcat.service

修改/etc/tomcat/目录中server.xml文件，修改HTTP的connector：

<Connector port="8080" protocol="HTTP/1.1" maxHttpHeaderSize="81920" URIEncoding="UTF-8" maxThreads="2048" acceptCount="2048" maxConnections="65536" redirectPort="8443" maxPostSize="10000000" />

若使用的是AJP，则还需要修改：

<Connector port="8009" URIEncoding="UTF-8" maxHttpHeaderSize="81920" maxThreads="2048" acceptCount="2048" maxConnections="65536" protocol="AJP/1.3" redirectPort="8443" maxPostSize="10000000"/>

Tomcat localhost日志输出访问耗时参数 %D，增加到如下红色位置：

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log." suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b %D" />

修改/etc/tomcat/tomcat-users.xml文件，在下面增加：

<user username="jenkins" password="xeJMkIOe" roles="manager-script"/>

修改/usr/share/tomcat/conf/目录中tomcat.conf文件，找到JAVA\_OPTS的配置的地方（默认是注释掉的）增加如下代码，注意如下是JDK1.7版本配置参数。

JAVA\_OPTS="-Djava.security.egd=file:/dev/./urandom -Djava.awt.headless=true -Dfile.encoding=UTF8 -Duser.timezone=GMT+08 -Xms128m -Xmx14400m -XX:PermSize=128m -XX:MaxPermSize=256m -XX:+UseConcMarkSweepGC"

**注意：Xmx的配置视服务器内存而定，8G内存可配到6400m左右，16G内存可配置为12800m或14400但切不可超出。**

**JDK1.8版本配置参数如下：**

JAVA\_OPTS="-Djava.security.egd=file:/dev/./urandom -Djava.awt.headless=true -Dfile.encoding=UTF8 -Duser.timezone=GMT+08 -Xms128m -Xmx14400m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=256m -XX:+UseConcMarkSweepGC"

#### 1.4.2 Tomcat7/8编译安装配置

**编译安装的tomcat则运行内存修改方法如下：**

默认将Tomcat7/8文件包放在/var/lib/或/usr/local目录下，然后进行解压，重名为为tomcat7或tomcat8。

完成后需要将tomcat配置开机自启动，方法如下：

在/etc/init.d目录下，创建tomcat7或者tomcat8文件，然后粘贴如下内容，需要注意如下内容中红色字体的修改，要符合部署环境。

#!/bin/sh

# chkconfig: 345 99 10

# description: Auto-starts tomcat

# /etc/init.d/tomcatd

# Tomcat auto-start

# Source function library.

#. /etc/init.d/functions

# source networking configuration.

#. /etc/sysconfig/network

RETVAL=0

export JDK\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.171-8.b10.el6\_9.x86\_64/jre

export CATALINA\_HOME=/var/lib/tomcat8

export CATALINA\_BASE=/var/lib/tomcat8

start()

{

if [ -f $CATALINA\_HOME/bin/startup.sh ];

then

echo $"Starting Tomcat"

$CATALINA\_HOME/bin/startup.sh

RETVAL=$?

echo " OK"

return $RETVAL

fi

}

stop()

{

if [ -f $CATALINA\_HOME/bin/shutdown.sh ];

then

echo $"Stopping Tomcat"

$CATALINA\_HOME/bin/shutdown.sh

RETVAL=$?

sleep 1

echo " OK"

ps -ef | grep tomcat|grep -v grep | grep -v PID | awk '{print $2}'|xargs kill -9

# [ $RETVAL -eq 0 ] && rm -f /var/lock/...

return $RETVAL

fi

}

case "$1" in

start)

start

;;

stop)

stop

;;

restart)

echo $"Restaring Tomcat"

$0 stop

sleep 1

$0 start

;;

\*)

echo $"Usage: $0 {start|stop|restart}"

exit 1

;;

esac

exit $RETVAL

然后设置开机自启：

chkconfig tomcat7 或tomcat8 on

或者

systemctl enable tomcat7 或tomcat8

修改/etc/tomcat/目录中server.xml文件，修改HTTP的connector：

<Connector port="8080" protocol="HTTP/1.1" maxHttpHeaderSize="81920" URIEncoding="UTF-8" maxThreads="2048" acceptCount="2048" maxConnections="65536" redirectPort="8443" maxPostSize="10000000"/>

若使用的是AJP，则还需要修改：

<Connector port="8009" URIEncoding="UTF-8" maxHttpHeaderSize="81920" maxThreads="2048" acceptCount="2048" maxConnections="65536" protocol="AJP/1.3" redirectPort="8443" maxPostSize="10000000"/>

Tomcat localhost日志输出访问耗时参数 %D，增加到如下红色位置：

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log." suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b %D" />

修改/etc/tomcat/tomcat-users.xml文件，在下面增加：

<user username="jenkins" password="xeJMkIOe" roles="manager-script"/>

**修改Tomcat运行内存：**

cd到tomcat/bin 下，编辑catalina.sh文件，向中插入以下代码，注意如下是JDK1.7版本配置参数。

JAVA\_OPTS="$JAVA\_OPTS -Xms128m -Xmx6400m -XX:PermSize=512m -XX:MaxPermSize=2048m -Djava.security.egd=file:/dev/./urandom -Dfile.encoding=UTF8 -Duser.timezone=GMT+08"

根据内存不同Xmx参数不同，注意如下是JDK1.7版本配置参数。

JAVA\_OPTS="$JAVA\_OPTS -Xms128m -Xmx12800m -XX:PermSize=512m -XX:MaxPermSize=2048m -Djava.security.egd=file:/dev/./urandom -Dfile.encoding=UTF8 -Duser.timezone=GMT+08"

**注意：Xmx的配置视服务器内存而定，8G内存可配到6400m左右，16G内存可配置为12800m或14400但切不可超出**

**JDK1.8版本配置参数如下：**

JAVA\_OPTS="$JAVA\_OPTS -Xms128m -Xmx6400m -XX:MetaspaceSize=512m -XX:MaxMetaspaceSize=2048m -Djava.security.egd=file:/dev/./urandom -Dfile.encoding=UTF8 -Duser.timezone=GMT+08"

根据内存不同Xmx参数不同，注意如下是JDK1.8版本配置参数。

JAVA\_OPTS="$JAVA\_OPTS -Xms128m -Xmx12800m -XX:MetaspaceSize=512m -XX:MaxMetaspaceSize=2048m -Djava.security.egd=file:/dev/./urandom -Dfile.encoding=UTF8 -Duser.timezone=GMT+08"

### 1.5 MariaDB10.X安装配置

**1.删除系统自带的mysql**

rpm -qa|grep mysql

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum remove mysql-libs

**2.创建/etc/yum.repos.d/MariaDB.repo文件，版本为10.4**

[mariadb]

name = MariaDB

baseurl = <http://yum.mariadb.org/10.4/centos7-amd64>

gpgkey=<https://yum.mariadb.org/RPM-GPG-KEY-MariaDB>

gpgcheck=1

**3.关闭并卸载旧版本的mariadb，安装新版本的mariadb。**

systemctl stop mariadb

yum remove mariadb-server mariadb mariadb-libs

rpm -qa|grep MariaDB

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum clean all

yum makecache

安装MariaDB，注意查看版本号

yum install MariaDB-client MariaDB-server MariaDB-devel

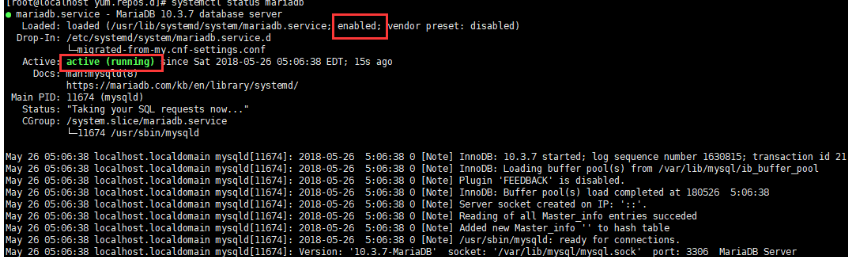
**4.启动数据库并设置为开机自启**

systemctl start mariadb

systemctl enable mariadb

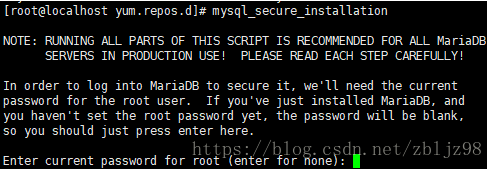
**5.查看MariaDB运行状态**

systemctl status mariadb

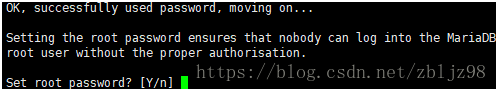


**6、初始化数据库，并删除测试数据库及更改权限和设置密码**

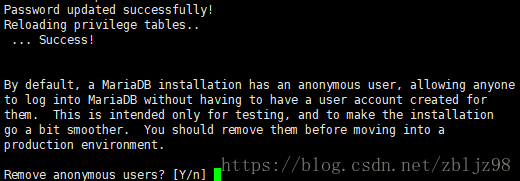
mysql\_secure\_installation



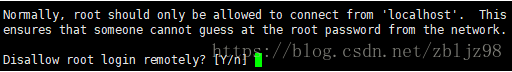
输入数据库设置密码



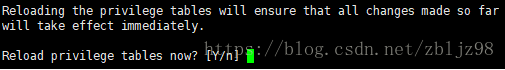
是否设置root密码，输入Y进行设置



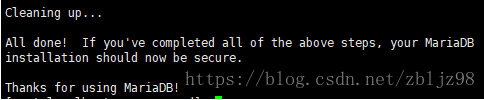
是否移除匿名用户，输入Y移除



是否拒绝root用户的远程登陆，根据实际情况选择



是否刷新权限表，输入Y刷新权限表



出现此界面，数据库安全设置完成。

注意：如果上述设置root密码失败，则使用如下方法初始化root密码。

首先停掉MariaDB服务：

systemctl stop mariadb

然后修改/etc/my.cnf.d目录中server.cnf文件，在[mysqld]节中增加skip-grant-tables

然后启动数据库：

systemctl start mariadb

用空密码进入mysql管理命令行，切换到mysql库。

abbuggy@abbuggy-ubuntu:~$ mysql

Welcome to the MySQL monitor.  Commands end with ; or \g.

mysql> use mysql;

Database changed

mysql> UPDATE user SET authentication\_string = password ( '密码' ) WHERE User = 'root' ;

Query OK, 0 rows affected (0.00 sec)

mysql>flush privileges;

mysql>quit

回到vim /etc/my.cnf.d/server.cnf，把刚才加入的那一行“skip-grant-tables”注释或删除掉。

再次重启MariaDB服务

systemctl restart mariadb

登陆数据库

$ mysql -uroot -p

再次设置密码：

mysql>SET PASSWORD = PASSWORD('密码');

mysql>flush privileges;

**7.连接数据库，并查询版本**

MariaDB [(none)]> select version();

+-----------------+

| version()       |

+-----------------+

| 10.3.14-MariaDB |

+-----------------+

1 row in set (0.000 sec)

**8.优化数据库配置**

（注意参数不可以copy直接用，一定要清楚配置内容，根据操作系统环境而定）

/etc/my.cnf.d/server.cnf文件中，在[server]节中配置内容如下

user= mysql

default-storage-engine=InnoDB

socket=/var/lib/mysql/mysql.sock

pid-file=/var/lib/mysql/mysql.pid

character-set-server=utf8

datadir=/var/lib/mysql/

log-bin=/var/lib/mysql/mysql-bin

expire-logs-days=14

sync-binlog=1

innodb\_flush\_method =O\_DIRECT

innodb-log-files-in-group =2

innodb\_log\_file\_size=256M

innodb-flush-log-at-trx-commit =1

innodb\_file\_per\_table=1

innodb\_buffer\_pool\_size=20G

tmp\_table\_size=32M

max\_heap\_table\_size=32M

max\_connections=2000

max\_connect\_errors =100000

query-cache-size=128M

query\_cache\_type=1

thread-cache-size=120

open-files-limit= 65535

table-definition-cache=4096

table-open-cache=4096

max\_allowed\_packet=50M

binlog\_format = MIXED

log-error=/home/mysqllog/mysql-error.log

slow\_query\_log=1

long\_query\_time=10

log\_queries\_not\_using\_indexes=1

slow\_query\_log\_file=/home/mysqllog/localhost-slow.log

**9）MariaDB搭建主从指导**

略，后续补充。

### 1.6 Redis安装配置

1）安装redis

yum install redis

如果提示：

No package redis available.  
Error: Nothing to do

则为yum添加epel源：

yum install epel-release

安装epel成功后，再执行yum install redis

启动服务：

service redis start

**设置开机启动redis**

chkconfig redis on

连接测试，执行命令：

redis-cli

2）防止服务器重启后，redis数据丢失，对redis配置如下

找到 /etc/sysctl.conf

vim /etc/sysctl.conf

在最后一行添加

vm.overcommit\_memory = 1

wq保存，用sysctl -p 使配置生效

sysctl -p

注意，上述问题只在Linux系统才存在，在Windows系统是没有问题的。

3）性能优化配置

4）安全优化配置

5）自动备份脚本

#!/bin/bash

echo "start..."

sdate=$(date "+ %Y-%m-%d %H:%M:%S")

msg=`redis-cli -p $1 -a $2 bgsave`

echo "start bgsave time:$sdate"

result=`redis-cli -p $1 -a $2 info Persistence | grep "rdb\_bgsave\_in\_progress" | awk -F":" '{print $2}'`

while [ `echo ${result} | awk -v tem="0" '{print($1>tem)? "1":"0"}'` -eq "1" ] ; do

    sleep 1

    result=`redis-cli -p $1 -a $2 info Persistence | grep "rdb\_bgsave\_in\_progress" | awk -F":" '{print $2}'`

done

edate=$(date "+ %Y-%m-%d %H:%M:%S")

echo "end bgsave time:$edate"

echo "renaming rdb file..."

date=$(date "+ %Y%m%d")

mv "../db03/dump.rdb" "../db03/${date}.rdb"

echo "finish backup rdb file"

echo "end"

### 1.7 MySQL5.6安装配置

1、删除系统自带的mysql

rpm -qa|grep mysql

yum remove mysql-libs

2、安装mysql YUM源

wget http://repo.mysql.com/mysql-community-release-el6-5.noarch.rpm

rpm -ivh mysql-community-release-el6-5.noarch.rpm

3、安装mysql

yum install mysql-server mysql-devel

4、root密码设置

service mysqld stop

mysqld\_safe --skip-grant-tables &

mysql -u root -p //回车直接进去，或者随便输入一个密码

use mysql;

update user set password=PASSWORD("8sDf3sfe@")where user="root"; //更改密码为 newpassord

flush privileges; //更新权限

quit

创建数据库脚本：

create database `数据库实例名称` character set 'utf8' collate 'utf8\_general\_ci';

grant all privileges on `数据库实例名称`.\* to 用户@'localhost' identified by '密码';

grant all privileges on `数据库实例名称`.\* to 用户@'%' identified by '密码';

flush privileges;

MySQL数据库主从配置方法：

分别修改master和slave机器中mysql配置文件my.cnf，该文件在/etc目录下 在[mysqld]配置段添加如下字段

**Master：**

server-id=1

log-bin=mysql-bin

binlog\_format = mixed

binlog-do-db=infoplus-v2 //需要同步的数据库，如果没有本行，即表示同步所有的数据库

binlog-ignore-db=mysql //被忽略的数据库

**slave：**

server-id=2

replicate-do-db=infoplus-v2 //同步的数据库，不写本行表示同步所有数据库

保存之后重启mysql服务

在master机上为slave机添加一同步帐号 用户为slave

grant replication slave on \*.\* to 'slave'@'slave服务器ip' identified by '密码';

flush privileges;

在master上执行命令查看master状态

mysql>show master status;

+-------------------------+---------------+-------------------+----------------------------+------------------------+

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |

+-------------------------+---------------+-------------------+----------------------------+------------------------+

| mysql-bin.000001 | 1533860 | infoplus-v2 | mysql | |

在slave上执行命令，使slave与master同步

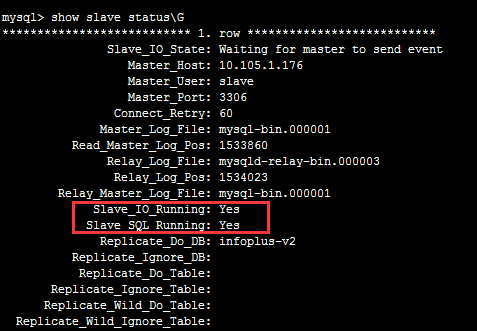
mysql>stop slave;

mysql>change master to master\_host='master服务器ip',master\_user='同步账号',master\_password='同步账号密码',master\_log\_file='master上的File值',master\_log\_pos=master上的Position值;

mysql>start slave;

查看slave连接状态，如果Slave\_IO\_Running、Slave\_SQL\_Running状态为Yes则设置成功。

mysql>show slave status\G



**设置日志保留天数**

在my.cnf文件中配置：

expire\_logs\_days = 3

表示保留3天的日志，这样老日志会自动被清理掉。

在my.cnf文件中增加expire\_logs\_days = 3参数后，若不能重启数据库，则通过如下命令来设置，即时生效：

set global expire\_logs\_days = 3;

附：查看日志方法

show binary logs;

show variables like '%log%';

set global expire\_logs\_days = 3;

首先创建备份目录

mkdir /opt/bak

chmod 711 bak

mkdir /opt/bak/bakmysql

chmod 711 bakmysql

mkdir /opt/bak/bakmysqlold

chmod 711 bakmysqlold

编写备份脚本

vim /opt/bak/backup.sh

chmod 777 /opt/bak/backup.sh

脚本内容：

#!/bin/sh

BK\_DR=/opt/bak/bakmysqlold

DAYS=7

cd /opt/bak/bakmysql

echo "You are in bakmysql directory"

mv infoplus-v2\* /opt/bak/bakmysqlold

echo "Old databases are moved to bakmysqlold folder"

Now=$(date +"%Y-%m-%d\_%H-%M-%S")

File=infoplus-v2-$Now.sql

mysqldump -uinfoplus -p'从数据库密码' infoplus-v2 | gzip > $File.gz

find $BK\_DR -name "infoplus-v2\*" -type f -mtime +$DAYS -exec rm {} \;

导入方法举例：gunzip < databasename-2018-07-25.sql.gz | mysql -uroot -proot databasename

定时备份设置，执行crontab -e 进行编辑

内容如下（每天凌晨1点，中午12点，下午18点开始执行备份脚本）：

00 01 \* \* \* /opt/bak/backup.sh

00 12 \* \* \* /opt/bak/backup.sh

00 18 \* \* \* /opt/bak/backup.sh

保存退出 并重新加载

service crond reload

### 1.8 MySQL5.7安装配置

删除系统自带的mysql

rpm -qa|grep mysql

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum remove mysql-libs

# 下载mysql5.7源安装包

shell> wget http://dev.mysql.com/get/mysql57-community-release-el7-8.noarch.rpm

# 安装mysql源

shell> yum localinstall mysql57-community-release-el7-8.noarch.rpm

检查mysql源是否安装成功

shell> yum repolist enabled | grep "mysql.\*-community.\*"

2、安装MySQL

shell> yum install mysql-community-server

3、启动MySQL服务

shell> systemctl start mysqld

查看MySQL的启动状态

shell> systemctl status mysqld

4、开机启动

shell> systemctl enable mysqld

shell> systemctl daemon-reload

5、修改root本地登录密码

mysql安装完成之后，在/var/log/mysqld.log文件中给root生成了一个默认密码。通过下面的方式找到root默认密码，然后登录mysql进行修改：

shell> grep 'temporary password' /var/log/mysqld.log

shell> mysql -u root -p

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';

或者

mysql> set password for 'root'@'localhost'=password('MyNewPass4!');

注意：mysql5.7默认安装了密码安全检查插件（validate\_password），默认密码检查策略要求密码必须包含：大小写字母、数字和特殊符号，并且长度不能少于8位。否则会提示ERROR 1819 (HY000): Your password does not satisfy the current policy requirements错误

MySQL5.7 关闭参数ONLY\_FULL\_GROUP\_BY

临时关闭方法：（命令行）

set GLOBAL sql\_mode ='STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION ';

彻底关闭方法：（修改配置文件）

在my.cnf里面的 [mysqld]下添加

set @@sql\_mode = 'STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION'

然后重启MySQL,再查看ONLY\_FULL\_GROUP\_BY已经不见了：

SELECT @@GLOBAL.sql\_mode;和SELECT @@SESSION.sql\_mode;

### 1.9 Memcached安装配置

#### 1）查找Memcached

yum search memcached

首先检查yum软件仓库中是否存在memcached，如果有，直接进入第3步安装即可，否则执行第2步。

#### 2）安装第三方软件库（可选）

标准的CentOS5软件仓库里面是没有memcache相应的包的,所以，我们的第一步就是导入第三方软件仓库，这里推荐的是RpmForge（RpmForge库现在已经拥有超过10000种的CentOS的软件包，被CentOS社区认为是最安全也是最稳定的一个第三方软件库），安装方法如下：

wget http://dag.wieers.com/rpm/packages/rpmforge-release/rpmforge-release-0.5.2-2.rf.src.rpm

rpm -ivh rpmforge-release-0.5.2-2.rf.src.rpm

#### 3）安装Memcached

yum -y install memcached

#### 4）验证安装

memcached -h

应该会输出一些帮助信息

#### 5）将memcache加入启动列表

chkconfig --level 2345 memcached on

#### 6）配置Memcache

vi /etc/sysconfig/memcached

文件中内容如下

PORT="11211″ 端口

USER="root" 使用的用户名

MAXCONN="1024″ 同时最大连接数

CACHESIZE="64″ 使用的内存大小

#OPTIONS="" 附加参数//默认是本机访问memcached，可以利用此参数设置其他服务器访问，配置参数为

OPTIONS="-l 0.0.0.0"

然后启动memcached服务：

service memcached start

#### 7）查看memcache状态

memcached-tool [Memcache Server IP]:[Memcache Server Port] stats

如：memcached-tool 127.0.0.1:11211 stats

### 1.10 OpenLDAP安装配置

#### 1.8.1 安装

#### 1.8.2 优化配置

1. 连接数过高时候(超过1024)，ldap就会出现故障，有类似下面日志打出。

daemon: accept(7) failed errno=24 (Too many open files)

解决方法：

a. 设定下客户端idle的超时时间(单位是秒)，超时的连接及时关闭掉。

如下:

#close idle connections, in case of too many open files(by lnzju)

idletimeout 20

默认是服务端不主动断。

b. 在slapd启动脚本中加入(ulimit -n 10240)来扩大打开文件数

如果还不行，可以修改代码slapd/daemon.c中的FD\_SETSIZE

2. 性能问题。关键数据需要加入index，否则会很慢。

日志中常常会有如下信息:

<= bdb\_equality\_candidates: (uidNumber) not indexed

调整如下：

index uidNumber eq

进一步的性能优化直接参见官方文档：

http://www.openldap.org/doc/admin24/tuning.html

idletimeout

以秒为单位，如果idle就强制断开client链接，默认是0，则关闭这个功能。

指定一个等待的秒数，如果超过这个时间客户都没有请求提交就关掉与客户的连接。默认情况下，idletimeout为0，表示禁用该功能。

sizelimit <integer>

设置一次search最多返回的entries数

该指令指定了一次搜索操作所能获得的最大条目数。

默认：

sizelimit 500

timelimit <integer>

以秒为单位，设置一次查询最多等待多少秒，超过时间，则返回，并表示超时。

该指令指定了slapd花在回答一个搜索请求上的最大秒数（实时）。如果在这段时间内请求没有完成，服务器端将返回一个超时给客户端。

默认：

timelimit 3600

loglevel <integer>

该指令指定了debug声明和统计数据应当被记入日志文件的级别(currently logged to the syslogd(icon\_cool.gif LOG\_LOCAL4 facility)。你必须将OpenLDAP配置为--enable-debug （默认）该指令才会工作(except for the two statistics levels, which are always enabled)。log levels是可以相加的。要想知道数字与debuglevel的对应关系，可以用-? 为参数启动slapd,你也可以参考下面的这个表。<integer>的可能值是：

表5.1: Debugging Levels

Level Description

-1 nable all debugging

0 no debugging

1 trace function calls

2 debug packet handling

4 heavy trace debugging

8 connection management

16 print out packets sent and received

32 search filter processing

64 configuration file processing

128 access control list processing

256 stats log connections/operations/results

512 stats log entries sent

1024 print communication with shell backends

2048 print entry parsing debugging

########################

openldap 优化 (2010-04-15 12:53)

/var//openldap/etc/openldap/slapd.conf

directory /var/openldap/var/openldap-data

cachesize 50000

idlcachesize 150000

checkpoint 1024 5

bash-2.05# cat DB\_CONFIG

set\_cachesize 0 536870912 1

set\_lg\_regionmax 10485760

set\_lg\_max 104857600

set\_lg\_bsize 20971520

#

# Automatically remove log files that are no longer needed.

set\_flags DB\_LOG\_AUTOREMOVE

# Just use these settings when doing slapadd...

#set\_flags DB\_TXN\_NOSYNC

#set\_flags DB\_TXN\_NOT\_DURABLE

如果做slapadd,可以把最后两行的注释去掉,slapadd速度会爆增，作完slapadd后再把最后两行注释加上，切记切记切记切记！！！！

# Just use these settings when doing slapadd...

set\_flags DB\_TXN\_NOSYNC

set\_flags DB\_TXN\_NOT\_DURABLE

bash-2.05# cat DB\_CONFIG

set\_cachesize 0 536870912 1

set\_lg\_regionmax 10485760

set\_lg\_max 104857600

set\_lg\_bsize 20971520

set\_cachesize 1 0 1

set\_lg\_regionmax 1048576

set\_lg\_max 10485760

set\_lg\_bsize 2097152

#

# Automatically remove log files that are no longer needed.

set\_flags DB\_LOG\_AUTOREMOVE

# Just use these settings when doing slapadd...

#set\_flags DB\_TXN\_NOSYNC

#set\_flags DB\_TXN\_NOT\_DURABLE

bash-2.05#

# Use of strong authentication encouraged.

rootpw aaaaa

#cachesize 655350

cachesize 65535

#dbcachesize 1024000000

dbcachesize 512000000

concurrency 10240

threads 10240

sockbuf\_max\_incoming 8294303s

### 1.11 Nginx安装配置

### 1.12 PHP安装配置

### 1.13 NFS安装配置

#### 一、NFS服务简介

  NFS 是Network File System的缩写，即网络文件系统。一种使用于分散式文件系统的协定，由Sun公司开发，于1984年向外公布。功能是通过网络让不同的机器、不同的操作系统能够彼此分享个别的数据，让应用程序在客户端通过网络访问位于服务器磁盘中的数据，是在类Unix系统间实现磁盘文件共享的一种方法。

  NFS 的基本原则是“容许不同的客户端及服务端通过一组RPC分享相同的文件系统”，它是独立于操作系统，容许不同硬件及操作系统的系统共同进行文件的分享。

  NFS在文件传送或信息传送过程中依赖于RPC协议。RPC，远程过程调用 (Remote Procedure Call) 是能使客户端执行其他系统中程序的一种机制。NFS本身是没有提供信息传输的协议和功能的，但NFS却能让我们通过网络进行资料的分享，这是因为NFS使用了一些其它的传输协议。而这些传输协议用到这个RPC功能的。可以说NFS本身就是使用RPC的一个程序。或者说NFS也是一个RPC SERVER。所以只要用到NFS的地方都要启动RPC服务，不论是NFS SERVER或者NFS CLIENT。这样SERVER和CLIENT才能通过RPC来实现PROGRAM PORT的对应。可以这么理解RPC和NFS的关系：NFS是一个文件系统，而RPC是负责负责信息的传输。

#### 二、环境介绍

系统平台：CentOS\_6.5

nfs服务器ip：192.168.10.233

linux客户端ip：192.168.10.234

Windows客户端：192.168.10.235

SELINUX=disabled

注意：防火墙需要开放nfs指定端口。

#### 三、服务器安装NFS服务

1、查看系统是否已安装NFS：

[root@localhost ~]# rpm -qa | grep nfs

[root@localhost ~]# rpm -qa | grep rpcbind

2、如果当前系统中没有安装NFS所需的软件包，安装nfs 和 rpcbind 软件包：

[root@localhost ~]# yum -y install nfs-utils  rpcbind

#### 四、NFS服务器的配置

**1、创建NFS共享目录，设置读写权限：**

[root@localhost ~]# mkdir /share

[root@localhost ~]# chmod -R 777 /share

**2、 编辑export文件，这个文件是NFS的主要配置文件：**

[root@localhost ~]# vi /etc/exports

/share 192.168.10.234(insecure,rw,no\_root\_squash)        ##注意地址和参数表之间没有空格##

**参数说明：**

***内容格式：*** *<输出目录>  [客户端1 选项（访问权限,用户映射,其他）]  [客户端2 选项（访问权限,用户映射,其他）]*

**客户端地址的指定方式**

指定ip地址的主机：192.168.10.234

指定子网中的所有主机：192.168.10.0/24或者192.168.10.0/255.255.255.0

指定域名的主机：whuang.test.com

指定域中的所有主机：\*.test.com

所有主机：\*

**访问权限选项**

设置输出目录只读：ro

设置输出目录读写：rw

**用户映射选项**

all\_squash：屏蔽所有远程用户的权限，将远程访问的所有普通用户及所属组都映射为匿名用户或用户组（nfsnobody）；

no\_all\_squash：与all\_squash取反（默认设置）；

root\_squash：屏蔽远程root权限，将root用户及所属组都映射为匿名用户或用户组（默认设置）；

no\_root\_squash：NFS客户端连接服务端时如果使用的是root的话，那么对服务端分享的目录来说，也拥有root权限。显然开启这项是不安全的；

anonuid=xxx：将远程访问的所有用户都映射为匿名用户，并指定该用户为本地用户（UID=xxx）；

anongid=xxx：将远程访问的所有用户组都映射为匿名用户组账户，并指定该匿名用户组账户为本地用户组账户（GID=xxx）；

**其它选项**

secure：限制客户端只能从小于1024的tcp/ip端口连接nfs服务器（默认设置）；

insecure：允许客户端从大于1024的tcp/ip端口连接服务器；

sync：同步写操作，，将数据同步写入内存缓冲区与磁盘中，效率低，但可以保证数据的一致性；

async：异步写操作，将数据先保存在内存缓冲区中，必要时才写入磁盘；

wdelay：延迟写操作，检查是否有相关的写操作，如果有则将这些写操作一起执行，这样可以提高效率（默认设置）；

no\_wdelay：若有写操作则立即执行，应与sync配合使用；

subtree：若输出目录是一个子目录，则nfs服务器将检查其父目录的权限(默认设置)；

no\_subtree：即使输出目录是一个子目录，nfs服务器也不检查其父目录的权限，这样可以提高效率；

**3、配置生效：**

[root@localhost ~]# exportfs -r

#### 五、启动NFS服务器rpcbind和nfs服务

[root@localhost ~]# service rpcbind  restart

[root@localhost ~]# service nfs  restart

**注意：一定是先启动rpcbind服务，然后再启动NFS服务**

[root@localhost ~]# chkconfig --level 123456 rpcbind on

[root@localhost ~]# chkconfig --level 123456 nfs on

#### 六、Linux客户端挂载NFS

**1、安装NFS：**

[root@localhost ~]# yum -y install nfs-utils

**2、查看服务器的共享目录信息：**

[root@localhost ~]#showmount -e 192.168.10.233

Export list for 192.168.10.233:

/share 192.168.8.0/22

**3、为了提高NFS的稳定性，使用TCP协议挂载，NFS默认用UDP协议：**

[root@localhost ~]# mkdir /mnt/share

[root@localhost ~]#mount -t nfs 192.168.10.233:/share /mnt/share -o proto=tcp -o nolock

**4、卸载已挂在的NFS:**

[root@localhost /]# umount /mnt/share/

**5、开机自动挂载:**

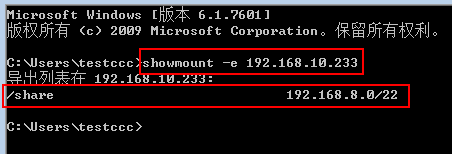
[root@localhost /]#echo "192.168.10.233:/share /mnt/share nfs defaults 0 0" >>/etc/fstab

#### 七、Windows客户端挂载NFS

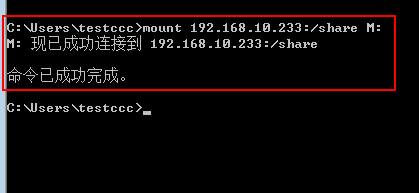
1、打开《控制面板》->《程序》->《打开或关闭windows功能》->《NFS客户端》；勾选《NFS客户端》，即开启windows NFS客户端服务；

2、win+R打开cmd终端：

C:\Windows\system32>showmount -e 192.168.10.233



C:\Windows\system32>mount 192.168.10.233:/share M:



成功挂载，打开计算机，你即可在你网络位置看到 M:盘了

3、取消挂载:

umount -a（卸载全部挂接点）或者直接在计算机 里面鼠标点击取消映射网络驱动器 M:

#### 八、常用命令

**1、exportfs**

启动了NFS之后又修改了/etc/exports，可以用exportfs 命令来使改动立刻生效：

**# exportfs [-aruv]**

-a 全部挂载或卸载 /etc/exports中的内容

-r 重新读取/etc/exports 中的信息 ，并同步更新/etc/exports、/var/lib/nfs/xtab

-u 卸载单一目录（和-a一起使用为卸载所有/etc/exports文件中的目录）

-v 在export的时候，将详细的信息输出到屏幕上。

具体例子：

# exportfs -au 卸载所有共享目录

# exportfs -rv 重新共享所有目录并输出详细信息

**2、nfsstat**

查看NFS的运行状态，对于调整NFS的运行有很大帮助。

**3、rpcinfo**

查看rpc执行信息，可以用于检测rpc运行情况的工具，利用rpcinfo -p 可以查看出RPC开启的端口所提供的程序有哪些。

**4、showmount**

-a 显示已经于客户端连接上的目录信息

-e IP或者hostname 显示此IP地址分享出来的目录

## 2.CentOS7

### 2.1 操作系统配置

#### 2.1.1 服务器配置DNS

若提供的服务器可以正常解析外网地址，则可以忽略此步骤。

若提供的服务器，需要配置DNS才可以连接外网或软件源，配置如下：

配置命令：echo "nameserver 学校DNS或者是8.8.8.8或者是114.114.114.114">>/etc/resolv.conf

#### 2.1.2 配置yum源

以163源为例：

1）下载repo文件

cd /tmp

wget http://mirrors.163.com/.help/CentOS7-Base-163.repo

2）备份并替换系统的repo文件

cp CentOS7-Base-163.repo /etc/yum.repos.d/

cd /etc/yum.repos.d/

mv CentOS-Base.repo CentOS-Base.repo.bak

mv CentOS7-Base-163.repo CentOS-Base.repo

3）执行yum源更新命令

yum clean all

yum makecache

配置完毕。

如果服务器不能连接外部软件源，则告诉该学校负责的老师将学校的内部源提供下，配置在/etc/yum.repos.d/中。

#### 2.1.3 更新操作系统

使用命令：

yum update -y

更新完成后，重启服务器。

#### 2.1.4 创建infoplus用户

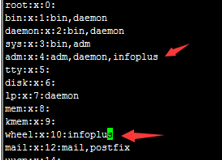
命令：useradd infoplus

命令：passwd infoplus

需要记录密码，密码建议为随机8—16位数，包括英文字母大小写、数字和特殊字符，需要注意键盘是英文模式输入。

将infoplus用户加入到adm和wheel组中，如下图：

vim /etc/group



然后修改/etc/sudoers 为 NOPASSWD，将前面#号去掉，如下图：



#### 2.1.5 配置主机名

设置主机名为*所属业务名称*

vi /etc/hostname #编辑配置文件，添加主机名

vi /etc/hosts #编辑配置文件，在每行后面添加主机名

举例：

127.0.0.1 主机名 #添加主机名

::1 主机名 #添加主机名

:wq！ #保存退出

#### 2.1.6 禁用SELinux

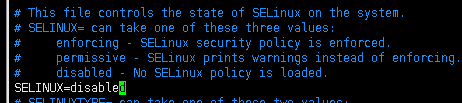
SELinux始终保持关闭状态，使用如下命令查看SELinux状态

getenforce



此时为开启状态，

vim /etc/sysconfig/selinux #编辑SELinux状态



将SELINUX=enforce 改为SELINUX=disabled

保存退出并重启服务器。

第二种方法：

sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux

重启服务器生效。

如果想立即生效（如果想临时性的改变）：

setenforce 0

**拓展说明：**

setenforce 1 设置SELinux 成为enforcing模式

setenforce 0 设置SELinux 成为permissive模式

查看状态 getenforce

#### 2.1.7 服务器修改文件打开数Openfile

1）先用ulimit -n或ulimit -a命令，发现文件打开数是1024

# ulimit -n

1024

2）用ulimit -n 40960把文件打开数改为最大值40960，再用ulimit -n验证发现修改成功

# ulimit -n 40960

# ulimit -n

40960

需要注意，有的centos6系统重启后，查看ulimit -n参数，又变回1024，则需要按照如下方法方法修改：

将ulimit 值添加到/etc/profile文件中（适用于有root权限登录的系统）

为了每次系统重新启动时，都可以获取更大的ulimit值，将ulimit 加入到/etc/profile 文件底部。

echo ulimit -n 40960 >>/etc/profile

source /etc/profile #加载修改后的profile

ulimit -n #显示40960

上述配置完成后，退出或重启系统，ulimit的值还是1024，这是为什么呢？

关键的原因是你登录的用户是什么身份，是不是root用户，由于服务器的root用户权限很大，一般是不能用来登录的，都是通过自己本人的登录权限进行登录，并通过sudo方式切换到root用户下进行工作。 用户登录的时候执行sh脚本的顺序：

/etc/profile.d/file

/etc/profile

/etc/bashrc

/mingjie/.bashrc

/mingjie/.bash\_profile

由于ulimit -n的脚本命令加载在第二部分，用户登录时由于权限原因在第二步还不能完成ulimit的修改，所以ulimit的值还是系统默认的1024。

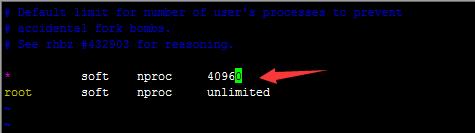
解决办法：

3）在/etc/security/limits.conf最后增加如下两行记录

\* soft nofile 40960

\* hard nofile 40960

4）进入/etc/security/limits.d目录中，默认就一个文件，文件名一般为conf为结尾，修改配置参数为40960，如下图：



#### 2.1.8 记录所有登录用户操作历史

需求：记录登陆后的IP地址和某用户名所操作的历史记录

实现：

##### 1）配置/etc/profile

USER\_IP=`who -u am i 2>/dev/null| awk '{print $NF}'|sed -e 's/[()]//g'`

if [ "$USER\_IP" = "" ]

then

USER\_IP=`hostname`

fi

if [ ! -d /opt/user\_log ]

then

mkdir /opt/user\_log

chmod -R 777 /opt/user\_log

fi

if [ ! -d /opt/user\_log/${LOGNAME} ]

then

mkdir /opt/user\_log/${LOGNAME}

chmod 300 /opt/user\_log/${LOGNAME}

fi

export HISTSIZE=4096

DT=`date "+%Y-%m-%d\_%H:%M:%S"`

export HISTFILE="/opt/user\_log/${LOGNAME}/${USER\_IP}-dbasky.$DT"

chmod 600 /opt/user\_log/${LOGNAME}/\*dbasky\* 2>/dev/null

##### 2）生效环境变量

source /etc/profile

##### 3）查看记录

说明：该脚本在系统的/tmp新建个dbasky目录，记录所有登陆过系统的用户和IP地址（文件名），每当用户登录/退出会创建相应的文件，该文件保存这段用户登录时期内操作历史，可以用这个方法来监测系统的安全性。

最后重启操作系统：

shutdown -r now #重启系统

#### 2.1.9 防火墙设置Firewalld

开启端口命令：

[root@localhost zones]# firewall-cmd --zone=public --permanent --add-port=8080/tcp

success

[root@localhost zones]# firewall-cmd --reload

Success

关闭端口命令：

[root@localhost zones]# firewall-cmd --zone=public --permanent --remove-port=8080/tcp

success

[root@localhost zones]# firewall-cmd --reload

success

限定IP网段访问：

firewall-cmd --permanent --zone=public --add-rich-rule='rule family="ipv4" source address="202.119.86.0/24" service name="mysql" accept'

限定IP访问：

firewall-cmd --permanent --add-rich-rule 'rule family=ipv4 source address=202.119.86.51 port port=3306 protocol=tcp accept'

重新加载规则：

firewall-cmd --reload

查看规则：

iptables -L -n

删除限定IP网段访问：

修改配置文件：/etc/firewalld/zones/public.xml

或firewall-cmd --permanent --remove-rich-rule 'rule family=ipv4 source address=192.168.0.1/2 port port=80 protocol=tcp accept' //删除配置

1.firewalld的基本使用

启动： systemctl start firewalld

查状态：systemctl status firewalld

停止： systemctl disable firewalld

禁用： systemctl stop firewalld

在开机时启用一个服务：systemctl enable firewalld.service

在开机时禁用一个服务：systemctl disable firewalld.service

查看服务是否开机启动：systemctl is-enabled firewalld.service

查看已启动的服务列表：systemctl list-unit-files|grep enabled

查看启动失败的服务列表：systemctl --failed

2.配置firewalld-cmd

查看版本： firewall-cmd --version

查看帮助： firewall-cmd --help

显示状态： firewall-cmd --state

查看所有打开的端口： firewall-cmd --zone=public --list-ports

更新防火墙规则： firewall-cmd --reload

查看区域信息: firewall-cmd --get-active-zones

查看指定接口所属区域： firewall-cmd --get-zone-of-interface=eth0

拒绝所有包：firewall-cmd --panic-on

取消拒绝状态： firewall-cmd --panic-off

查看是否拒绝： firewall-cmd --query-panic

#### 2.1.10 操作系统内核参数优化

对TCP连接数优化：

[root@hadoop13 ~]# vim /etc/sysctl.conf

net.ipv4.tcp\_max\_tw\_buckets = 360000

net.ipv4.tcp\_max\_syn\_backlog = 8192

net.ipv4.tcp\_syncookies = 1

net.ipv4.tcp\_tw\_reuse = 1

net.ipv4.tcp\_tw\_recycle = 1

net.ipv4.tcp\_fin\_timeout = 10

net.ipv4.tcp\_sack=0

kernel.pid\_max=1000000

# 让参数生效

[root@hadoop13 ~]# sysctl -p

**拓展说明，具体每个参数的含义：**

net.ipv4.tcp\_max\_tw\_buckets = 360000 #改值的默认值是18w

net.ipv4.tcp\_max\_syn\_backlog = 8192 #默认为1024，等待连接的网络连接数

net.ipv4.tcp\_syncookies = 1 #表示开启SYN Cookies。当出现SYN等待队列溢出时，启用cookies来处理，可防范少量SYN攻击，默认为0，表示关闭；

net.ipv4.tcp\_tw\_reuse = 1 #表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭；

net.ipv4.tcp\_tw\_recycle = 1 #表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭。

net.ipv4.tcp\_fin\_timeout = 10s #修改系統默认的TIMEOUT时间,默认是60s.

net.ipv4.tcp\_sack=0 #禁用内核 SACK 配置防范漏洞利用；

#### 2.1.11 时间和时区设置

##### 1）有外网情况下，使用ntpdate做时间同步

注：需要咨询学校用户是否有时间服务器，如果有，则需要使用学校时间服务器进行时间同步；如果学校没有，我们就是用使用China ntp服务器，若不准确可以更换其它时间服务器。

ntpdate cn.pool.ntp.org（注意：同步时ntpd需要关闭）



同步之后将时间同步写入定时脚本

crontab -e

添加如下定时语句：

\*/1 \* \* \* \* /sbin/ntpdate cn.pool.ntp.org;hwclock -w //每天1分钟执行一次

保存退出并重新载入

service crond reload

##### 2）没有外网的情况下，使用手动更改时间

**注：需要咨询学校用户是否有时间服务器，如果有，则需要使用学校时间服务器进行时间同步；如果学校没有，则需要手动校准日期和时间。**

#date 查看系统时间



#date -s 修改日期



#date -s 修改时间



修改完后执行：clock -w ,强制将时间写入COMS！

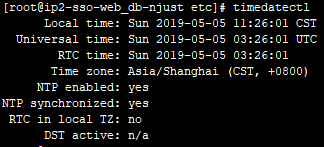


修改时区：（将Asia/shanghai-上海时区写入当前时区）

#cp -f /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

提示是否覆盖,输入Y回车

然后执行timedatectl命令，查看时区和时间：Asia/Shanghai（CST,中国时区+0800）



需要注意：有的系统会将/etc/localtime文件做成软连接，如下截图，这时在做时区覆盖后，不会生效，需要重新替换此链接。



修改时区软连接：

先将原有问题的localtime移走，命令为mv /etc/localtime /tmp

然后将上海的时区做软连接，命令为ln -s /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

#### 2.1.12 安全设置

**1）注释掉系统不需要的用户**

注意：不建议直接删除，当你需要某个用户时，自己重新添加会很麻烦。

cp /etc/passwd /etc/passwdbak #修改之前先备份

vi /etc/passwd #编辑用户，在前面加上#注释掉此行

#adm:x:3:4:adm:/var/adm:/sbin/nologin

#lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

#sync:x:5:0:sync:/sbin:/bin/sync

#shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

#halt:x:7:0:halt:/sbin:/sbin/halt

#uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin

#operator:x:11:0:operator:/root:/sbin/nologin

#games:x:12:100:games:/usr/games:/sbin/nologin

#gopher:x:13:30:gopher:/var/gopher:/sbin/nologin

#ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin #注释掉ftp匿名账号

cp /etc/group /etc/groupbak #修改之前先备份

vi /etc/group #编辑用户组，在前面加上#注释掉此行

#adm:x:4:root,adm,daemon

#lp:x:7:daemon,lp

#uucp:x:14:uucp

#games:x:20:

#dip:x:40:

**2）关闭系统不需要的服务**

service acpid stop chkconfig acpid off #停止服务，取消开机启动 #电源进阶设定，常用在 Laptop 上

service autofs stop chkconfig autofs off #停用自动挂载档桉系统与週边装置

service bluetooth stop chkconfig bluetooth off #停用Bluetooth蓝芽

service cpuspeed stop chkconfig cpuspeed off #停用控制CPU速度主要用来省电

service cups stop chkconfig cups off #停用 Common UNIX Printing System 使系统支援印表机

service ip6tables stop chkconfig ip6tables off #禁止IPv6

**3）禁止使用Ctrl+Alt+Del快捷键重启服务器**

cp /etc/inittab /etc/inittabbak

vi /etc/inittab #注释掉下面这一行

#ca::ctrlaltdel:/sbin/shutdown -t3 -r now

或者

vi /etc/init/control-alt-delete.conf

#注释掉

#exec /sbin/shutdown -r now "Control-Alt-Deletepressed"

**4）使用yum update更新系统时不升级内核，只更新软件包**

由于系统与硬件的兼容性问题，有可能升级内核后导致服务器不能正常启动，这是非常可怕的，没有特别的需要，建议不要随意升级内核。

cp /etc/yum.conf /etc/yum.confbak

1、修改yum的配置文件 vi /etc/yum.conf 在[main]的最后添加 exclude=kernel\*

2、直接在yum的命令后面加上如下的参数：

yum --exclude=kernel\* update

查看系统版本 cat /etc/issue

查看内核版本 uname -a

**5）关闭Centos自动更新**

chkconfig --list yum-updatesd #显示当前系统状态

yum-updatesd 0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭

service yum-updatesd stop #关闭 开启参数为start

停止 yum-updatesd： [确定]

service yum-updatesd status #查看是否关闭

yum-updatesd 已停

chkconfig --level 35 yum-updatesd off #禁止开启启动（系统模式为3、5）

chkconfig yum-updatesd off #禁止开启启动（所有启动模式全部禁止）

chkconfig --list yum-updatesd #显示当前系统状态

yum-updatesd 0:关闭 1:关闭 2:启用 3:关闭 4:启用 5:关闭 6:关闭

**6）linux限制登录失败次数并锁定**

在/etc/pam.d/login和/etc/pam.d/sshd第二行添加

auth required pam\_tally2.so deny=6 unlock\_time=180 even\_deny\_root root\_unlock\_time=180

登录失败5次锁定180秒，根据需要设置是否包括root

**7）拓展说明**

Linux定期做日志安全检查

将日志移动到专用的日志服务器里，这可避免入侵者轻易的改动本地日志。下面是常见linux的默认日志文件及其用处：

/var/log/message – 记录系统日志或当前活动日志。

/var/log/auth.log – 身份认证日志。

/var/log/cron – Crond 日志 (cron 任务).

/var/log/maillog – 邮件服务器日志。

/var/log/secure – 认证日志。

/var/log/wtmp 历史登录、注销、启动、停机日志和，lastb命令可以查看登录失败的用户

/var/run/utmp 当前登录的用户信息日志，w、who命令的信息便来源与此

/var/log/yum.log Yum 日志。

Linux安装logwatch

Logwatch是使用 Perl 开发的一个日志分析工具。能够对Linux 的日志文件进行分析，并自动发送mail给相关处理人员，可定制需求。

Logwatch的mail功能是借助宿主系统自带的 mail server 发邮件的，所以系统需安装mail server , 如sendmail,postfix,Qmail等。

#### 2.1.13 安装VMtools

安装命令参考：yum install open-vm-tools.x86\_64

#### 2.1.14 设置系统字符集

**01、查看Linux当前语言**

echo $LANG

C #C代表英文环境 ASCII 编码格式,通杀很多问题（乱码）

**02、查看系统支持的字符集**

locale -a #系统可以支持的语言

locale #当前系统的语言环境

**03、修改系统环境字符集**

export LANG=zh\_CN.utf8

LANG #系统主要语系

LC\_ALL #系统整体语系

vim /etc/profile

export LANG=zh\_CN.utf8

source /etc/profile

[root@t1 ~]# cat /etc/sysconfig/i18n #修改文件，也是一种方案

LANG="en\_US.UTF-8"

SYSFONT="latarcyrheb-sun16" #系统字体

### 2.2 Apache（httpd）安装配置

#### 2.2.1 安装Apache

如下Apache的安装方法需要根据部署条件，选择其中一种：

* 如学校对安全比较重视，且后续还会做安全加固，服务器可以连接外网，所以需要安装apache的最新版本，方法选择2.2.1.1，建议首选安装最新版本；
* 如学校服务器在内网，允许连接外网，且平台不需要外网访问，则apache可以安装默认版本，方法选择2.2.1.2；
* 如果学校服务器不允许连接外网，且与用户沟通后，也不开通外网访问权限，那么只能编译安装apache，且要选择最新版本。

##### 2.2.1.1 安装最新版本Apache

默认情况下， Centos7为软件设置了一些预定义的软件库。不幸的是，这些预定义软件库上的很多软件已近过时了。

**第一步：找出默认软件库上的Apache版本**

为了查找Centos上软件库里的Apache版本，在命令行下输入以下指令：

yum info httpd

如果你的centos保持着默认的配置的话，你很可能会按到类似的结果：

[root@crosp ~]# yum info httpd

Loaded plugins: fastestmirror

epel/x86\_64/metalink                                     |  19 kB     00:00

epel                                                     | 4.3 kB     00:00

(1/3): epel/x86\_64/group\_gz                                | 170 kB   00:00

(2/3): epel/x86\_64/primary\_db                              | 4.6 MB   00:00

(3/3): epel/x86\_64/updateinfo                              | 780 kB   00:00

Loading mirror speeds from cached hostfile

\* base: mirrors.ircam.fr

\* epel: mirrors.ircam.fr

\* extras: [centos.mirror.fr.planethoster.net](http://centos.mirror.fr.planethoster.net/)

\* updates: [centos.mirror.fr.planethoster.net](http://centos.mirror.fr.planethoster.net/)

Available Packages

Name        : httpd

Arch        : x86\_64

Version     : 2.4.6

Release     : 45.el7.centos.4

Size        : 2.7 M

Repo        : updates/7/x86\_64

Summary     : Apache HTTP Server

URL         : <http://httpd.apache.org/>

License     : ASL 2.0

Description : The Apache HTTP Server is a powerful, efficient, and extensible

            : web server.

正如你从上面的输出结果中看到，默认软件库中只有2.4.6版本的apache。

注意：Centos使用 向后修复安全实践 来将新的软件修复应用到更早期的版本。所以，默认库中的早期版本软件不意味着就更脆弱。不过对于我来说，并不算是个满足需求的更新。

你可以使用changelog来查看版本更新：

sudo yum install yum-changelog

yum changelog httpd

**第二步：安装CodeIT库**

CodeIT的人提供了一个很好的自定义库。这个库提供了最新版本的服务器软件(Apache & Nginx)。

在安装CodeIT库之前，你需要开启 EPEL 。EPEL提供了CodeIT库需要的依赖。

sudo yum install -y epel-release

最后，我们来安装CodeIT库

cd /etc/yum.repos.d && wget <https://repo.codeit.guru/codeit.el>`rpm -q --qf "%{VERSION}" $(rpm -q --whatprovides redhat-release)`.repo

接下来，我们再次确认Apache的包版本，你应该会看到类似下面的输出

[root@server-zujug2lu yum.repos.d]# yum info httpd

Loaded plugins: changelog, fastestmirror

Loading mirror speeds from cached hostfile

\* epel: [mirrors.yun-idc.com](http://mirrors.yun-idc.com/)

CodeIT                                                                                                                                                                  | 2.9 kB  00:00:00

CodeIT/x86\_64/primary\_db                                                                                                                                                | 114 kB  00:00:02

Installed Packages

Name        : httpd

Arch        : x86\_64

Version     : 2.4.6

Release     : 88.el7.centos

Size        : 9.4 M

Repo        : installed

From repo   : base

Summary     : Apache HTTP Server

URL         : <http://httpd.apache.org/>

License     : ASL 2.0

Description : The Apache HTTP Server is a powerful, efficient, and extensible

            : web server.

Available Packages

Name        : httpd

Arch        : x86\_64

Version     : 2.4.41

Release     : 1.codeit.el7

Size        : 1.4 M

Repo        : CodeIT/x86\_64

Summary     : Apache HTTP Server

URL         : <https://httpd.apache.org/>

License     : ASL 2.0

Description : The Apache HTTP Server is a powerful, efficient, and extensible

            : web server.

**第三步：安装Apache**

现在你应该有了安装最新版apache所需要的所有东西：

yum install httpd mod\_ssl openssl -y

一旦Apache安装好，我们就可以测试apache是否正常工作：

[root@crosp ~]# systemctl start httpd

输出应该是类似下图：

[root@crosp ~]# curl localhost

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "<http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd>">

<html xmlns="<http://www.w3.org/1999/xhtml>" xml:lang="en">

    <head>

        <title>Test Page for the Apache HTTP Server on CentOS</title>

        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

                ...

最后，让systemctl将Apache设置为开机自启动：

[root@crosp ~]# systemctl enable httpd

查看apache版本为2.4.39 大功告成！

[root@server-zujug2lu yum.repos.d]# httpd -V

Server version: Apache/2.4.39 (codeit)

Server built:   Apr  3 2019 18:54:14

Server's Module Magic Number: 20120211:84

Server loaded:  APR 1.5.2, APR-UTIL 1.5.2

Compiled using: APR 1.5.2, APR-UTIL 1.5.2

Architecture:   64-bit

Server MPM:     event

  threaded:     yes (fixed thread count)

    forked:     yes (variable process count)

Server compiled with....

-D APR\_HAS\_SENDFILE

-D APR\_HAS\_MMAP

-D APR\_HAVE\_IPV6 (IPv4-mapped addresses enabled)

-D APR\_USE\_SYSVSEM\_SERIALIZE

-D APR\_USE\_PTHREAD\_SERIALIZE

-D SINGLE\_LISTEN\_UNSERIALIZED\_ACCEPT

-D APR\_HAS\_OTHER\_CHILD

-D AP\_HAVE\_RELIABLE\_PIPED\_LOGS

-D DYNAMIC\_MODULE\_LIMIT=256

-D HTTPD\_ROOT="/etc/httpd"

-D SUEXEC\_BIN="/usr/sbin/suexec"

-D DEFAULT\_PIDLOG="/run/httpd/httpd.pid"

-D DEFAULT\_SCOREBOARD="logs/apache\_runtime\_status"

-D DEFAULT\_ERRORLOG="logs/error\_log"

-D AP\_TYPES\_CONFIG\_FILE="conf/mime.types"

-D SERVER\_CONFIG\_FILE="conf/httpd.conf"

##### 2.2.1.2 安装2.4默认版本

需要注意，首选安装Apache最新版本，CentOS7默认版本并不是最新版本。

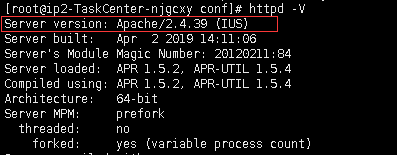
yum install httpd mod\_ssl openssl

##### 2.2.1.3 编译安装配置

略，后续补充。

#### 2.2.2 配置Apache

使用httpd -V 命令查看下server mpm的处理模式，如下图所示：mpm所用模式是prefork。



**学习小结：**

prefork MPM：

prefork是比较古老而又稳定的apache模式，特点是每个进程都是单线程，在一个时间点只能处理一个连接，需要启动大量的进程来处理高并发的请求。由于是单线程进程，因而无须考虑线程安全的问题，可以使用非线程安全的库，例如mod\_php。

优点是成熟稳定，缺点是比较消耗内存，而且并发支持受限于进程数量，对高并发支持稍差。

worker MPM：

worker同样使用多个进程，但每个进程又拥有多个线程，每个线程处理一个连接。由于线程是轻量级的，因而具有较高的并发性，同时，多个进程又获得了一定的稳定性。

worker模式特点是占用内存少，并发性比较高，缺点是必须考虑线程安全。如果使用了keep-alive方式，一个线程可能会被一直保持一个连接，但中间没有请求，直到超时。如果有多个线程被这样占据，在高并发场景下同样会出现无线程可用的情形。

event MPM：

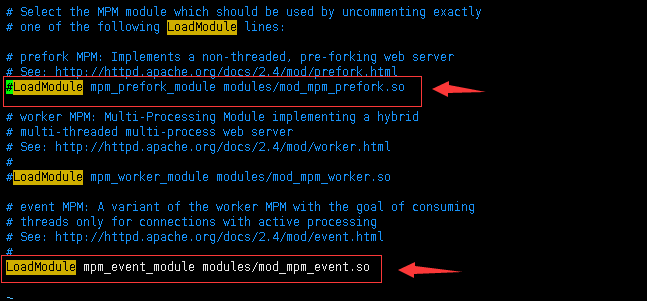
event模式是在2.4版本中才稳定发布的模式，它在worker的基础上，解决了keep-alive连接不能释放的问题。event MPM中，会有一个专门的线程来管理这些keep-alive类型的线程，当有真实请求过来的时候，将请求传递给服务线程，执行完毕后，又允许它释放。这样增强了高并发场景下的请求处理能力。

需要将Server MPM模式修改为event模式，方法如下：

将/etc/httpd/conf.modules.d目录中00-mpm.conf文件进行修改，修改内容如下：

将LoadModule mpm\_prefork\_module modules/mod\_mpm\_prefork.so注释掉，

将#LoadModule mpm\_event\_module modules/mod\_mpm\_event.so的#号去掉，如下截图，然后重启httpd，重启名为为service httpd restart



在httpd.conf中增加如下配置：

vim /etc/httpd/conf/httpd.conf 加入一句 ServerName localhost:80

修改apache 配置文件：

vim /etc/httpd/conf/httpd.conf

Timeout 60参数修改成： Timeout 720

开启httpd服务：

service httpd start或者/bin/systemctl start httpd.service

设置开机自启动：

systemctl enable httpd.service

在/etc/httpd/conf.d目录中新建infoplus.conf文件：

vim infoplus.conf

文件内容如下：（ centos版本）如下代理配置，需根据实际需求调整。

<VirtualHost \*:80>

ServerAdmin i@marstone.net

DocumentRoot /var/www

ServerName 域名或IP地址

ServerAlias 域名或IP地址

LogFormat "%h %u %l %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %T %D" infoplus

CustomLog "| /usr/sbin/rotatelogs /var/log/httpd/infoplus-access-combined-ssl-log-%Y-%m-%d 86400 480" infoplus

　　　 ErrorLog "| /usr/sbin/rotatelogs /var/log/httpd/infoplus-error-ssl-log-%Y-%m-%d 86400 480"

RedirectMatch 302 ^/$ /taskcenter/

Header set P3P 'policyref="http://aitpro.local/w3c/p3p.xml", CP="NOI DSP COR NID CUR ADM DEV OUR BUS"'

<Location /infoplus>

ProxyPass ajp://engine服务器地址:8009/infoplus

</Location>

<Location /taskcenter>

ProxyPass ajp://localhost:8009/taskcenter

</Location>

<Location /taskcenter-beta>

ProxyPass ajp://localhost:8009/taskcenter-beta

</Location>

<Location /authorize>

ProxyPass ajp://localhost:8009/authorize

</Location>

<Location /dashboard>

ProxyPass ajp://localhost:8009/dashboard

</Location>

<Location /file>

ProxyPass ajp://API服务器地址:8009/fileapi/file

</Location>

<Location /manager-app/manager>

ProxyPass ajp://localhost:8009/manager

ProxyPassReverse ajp://localhost:8009/manager

</Location>

<Location /manager-engine/manager>

ProxyPass ajp://engine服务器地址:8009/manager

ProxyPassReverse ajp://engine服务器地址:8009/manager

</Location>

<Location /manager-api/manager>

ProxyPass ajp://API服务器地址:8009/manager

ProxyPassReverse ajp://API服务器地址:8009/manager

</Location>

</VirtualHost>

需要注意，配置fileapi的location节，有特殊情况需要允许方法配置：

<Location /file>  
   <Limit GET POST PUT DELETE OPTIONS>  
      Require all granted  
   </Limit>  
   ProxyPass ajp://IP地址:8009/fileapi/file  
</Location>

#### 2.2.3 Apache2性能优化

**拓展说明：**

apache2三种模式，从2.0开始，apache引入了MPM（Multi-Processing Module，多进程处理模块）。MPM有prefork, worker和event(在2.4版本中稳定发布)模式，三种模式拥有不同的特点和性能。

**prefork MPM**

prefork是比较古老而又稳定的apache模式，特点是每个进程都是单线程，在一个时间点只能处理一个连接，需要启动大量的进程来处理高并发的请求。由于是单线程进程，因而无须考虑线程安全的问题，可以使用非线程安全的库，例如mod\_php。

优点是成熟稳定，缺点是比较消耗内存，而且并发支持受限于进程数量，对高并发支持稍差。

**worker MPM**

worker同样使用多个进程，但每个进程又拥有多个线程，每个线程处理一个连接。由于线程是轻量级的，因而具有较高的并发性，同时，多个进程又获得了一定的稳定性。

worker模式特点是占用内存少，并发性比较高，缺点是必须考虑线程安全。如果使用了keep-alive方式，一个线程可能会被一直保持一个连接，但中间没有请求，直到超时。如果有多个线程被这样占据，在高并发场景下同样会出现无线程可用的情形。

**event MPM**

event模式是在2.4版本中才稳定发布的模式，它在worker的基础上，解决了keep-alive连接不能释放的问题。event MPM中，会有一个专门的线程来管理这些keep-alive类型的线程，当有真实请求过来的时候，将请求传递给服务线程，执行完毕后，又允许它释放。这样增强了高并发场景下的请求处理能力。

查看Apache工作模式，CentOS系统命令：

httpd -V |grep -i mpm

Ubuntu系统命令：

apachectl -V | grep -i mpm

需要注意apache（httpd）因版本的不同，而优化参数不同。

优化参数举例说明：

apache2的默认配置文件位于/etc/apache2/apache2.conf，不同模式对应的配置文件在/etc/apache2/mods-available/下，有mpm\_prefork.conf， mpm\_worker.conf和mpm\_event.conf。

##### 1）prefork MPM性能调优

mpm\_prefork.conf文件内容如下：

<IfModule mpm\_prefork\_module>

StartServers 10 # 启动时进程数

MinSpareServers 5 # 最小空闲进程数

MaxSpareServers 10 # 最大空闲进程数

MaxRequestWorkers 100 # 最大并发进程数

MaxConnectionsPerChild 10000 # 最大连接数限制

</IfModule>

各个指令的含义：

StartServers : apache2启动时创建的服务进程数量。

MinSpareServers：最小空闲进程数量，空闲进程指的是没有处理请求的进程。

MaxSpareServers : 最大空闲进程数量。

MaxRequestWorkers : 最大同时处理请求的进程数量，也是最大的同时连接数，表示了apache的最大请求并发能力，超过该数目后的请求，将排队。

MaxConnectionsPerChild : 进程生命周期内，处理的最大请求数目。达到该数目后，进程将死掉。如果设置为0，表示没有限制。该参数的意义在于，避免了可能存在的内存泄露带来的系统问题。

通过上面的介绍可以发现，prefork模式下，影响并发性能最重要的参数就是MaxRequestWorkers，它决定了apache的并发处理能力。但是这个参数不是越大越好，因为如果超出了系统硬件能力，机器会卡死。

如果确定合适的MaxRequestWorkers呢？

首先，通过top命令查看apache进程占用的资源，主要看%CPU和%MEM这两个指标，例如，每个进程的CPU占用率不超过1%，每个进程的内存占用率不超过2%，考虑内存限制，比较合适的apache进程数量为50个。

然后，逐步测试最大值。通过观测得来的CPU和内存的指标有一定的误差，一般可以适当调节这个数值，例如调到1.5或者2倍，再通过峰值场景下的机器是否卡顿来判断是继续上调还是下调。

调整完参数后，一般需要重启apache。

service apache2 restart

##### 2）worker MPM性能调优

mpm\_worker.conf文件内容如下：

<IfModule mpm\_worker\_module>

StartServers 2 # 启动时进程数

MinSpareThreads 25 # 最小空闲线程数

MaxSpareThreads 75 # 最大空闲线程数

ThreadLimit 64 # 每个进程可以启动的线程数量上限值

ThreadsPerChild 25 # 每个进程可以启动的线程数量

MaxRequestWorkers 400 # 线程数量最大值

MaxConnectionsPerChild 0 # 最大连接数限制

</IfModule>

##### 3）event MPM性能调优

httpd 2.4.35以后 增加配置

<IfModule mpm\_event\_module>

ServerLimit 1000

StartServers 20

MinSpareThreads 25

MaxSpareThreads 1200

ThreadsPerChild 50

MaxRequestWorkers 2000

MaxConnectionsPerChild 0

</IfModule>

Group apache

Timeout 720

Keepalive on

KeepAliveTimeout 5

TraceEnable off

#### 2.2.4 Apache2安全优化

##### 1）Apache隐藏版本号

**apache在默认404、403提示页面中隐藏apache版本和系统信息**



要隐藏红色部分的apache版本号和系统信息，在apache配置文件中（linux系统中是httpd.conf），找到如下两行：

ServerTokens OS

ServerSignature On

对应修改为：

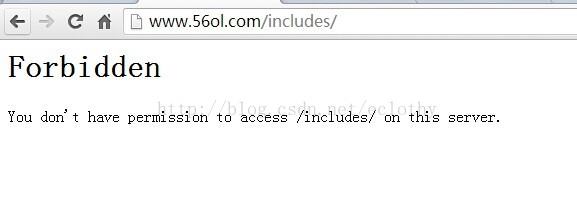
ServerTokens Prod

ServerSignature Off

然后执行：

service httpd restart

重启apache即可，修改后如图所示：



##### 2）Apache禁用默认icons目录

进入apache的配置文件 httpd.conf

vim /etc/httpd/conf/httpd.conf

找到：

Options Indexes FollowSymLinks

修改为：

Options FollowSymLinks

**详细举例：**

<Directory "/var/www/icons">

    Options Indexes MultiViews FollowSymLinks

    AllowOverride None

    Order allow,deny

    Allow from all

</Directory>

修改为：

<Directory "/var/www/icons">

    Options FollowSymLinks

    AllowOverride None

    Order allow,deny

    Allow from all

</Directory>

##### 3）Apache禁用默认的html目录列表

在/etc/httpd/conf/httpd.conf文件中，修改如下

<Directory "/var/www/html">

# Options Indexes FollowSymLinks ##注释该行

AllowOverride None

Require all granted

</Directory>

然后重启apache

service httpd restart

##### 4）移除apache欢迎页面

mv /etc/httpd/conf.d/welcome.conf /etc/httpd/conf.d/welcome.conf.bk

### 2.3 JDK安装配置

安装JDK版本根据程序而定：

安装jdk1.7

yum install java-1.7.0-openjdk -y

安装jdk1.8

yum install java-1.8.0-openjdk -y

### 2.4 Tomcat安装配置

#### 2.4.1 Tomcat7安装与配置

安装命令：

yum install tomcat tomcat-admin-webapps

tomcat开机自启动命令：

systemctl enable tomcat.service

启动tomcat命令：

/bin/systemctl start tomcat.service

修改/etc/tomcat/目录中server.xml文件，修改HTTP的connector：

<Connector port="8080" protocol="HTTP/1.1" maxHttpHeaderSize="81920" URIEncoding="UTF-8" maxThreads="2048" acceptCount="2048" maxConnections="65536" redirectPort="8443" maxPostSize="10000000"/>

若使用的是AJP，则还需要修改：

<Connector port="8009" URIEncoding="UTF-8" maxThreads="2048" acceptCount="2048" maxConnections="65536" protocol="AJP/1.3" maxHttpHeaderSize="81920" redirectPort="8443" maxPostSize="10000000"/>

Tomcat localhost日志输出访问耗时参数 %D，增加到如下红色位置：

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log." suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b %D" />

修改/etc/tomcat/tomcat-users.xml文件，在下面增加：

<user username="jenkins" password="xeJMkIOe" roles="manager-script"/>

修改/usr/share/tomcat/conf/目录中tomcat.conf文件，找到JAVA\_OPTS的配置的地方（默认是注释掉的）增加如下代码，注意如下是JDK1.7版本配置参数。

JAVA\_OPTS="-Djava.security.egd=file:/dev/./urandom -Djava.awt.headless=true -Dfile.encoding=UTF8 -Duser.timezone=GMT+08 -Xms128m -Xmx14400m -XX:PermSize=128m -XX:MaxPermSize=256m -XX:+UseConcMarkSweepGC"

**注意：Xmx的配置视服务器内存而定，8G内存可配到6400m左右，16G内存可配置为12800m或14400但切不可超出。**

**JDK1.8版本配置参数如下：**

JAVA\_OPTS="-Djava.security.egd=file:/dev/./urandom -Djava.awt.headless=true -Dfile.encoding=UTF8 -Duser.timezone=GMT+08 -Xms128m -Xmx14400m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=256m -XX:+UseConcMarkSweepGC"

#### 2.4.2 Tomcat7/8编译安装配置

**编译安装的tomcat则运行内存修改方法如下：**

默认将Tomcat7/8文件包放在/var/lib/或/usr/local目录下，然后进行解压，重名为为tomcat7或tomcat8。

完成后需要将tomcat配置开机自启动，方法如下：

添加非登录用户

useradd -s /sbin/nologin tomcat8

vi /etc/systemd/system/tomcat8.service

#在文件中写入以下内容

# Systemd unit file for tomcat

[Unit]

Description=Apache Tomcat Web Application Container

After=syslog.target network.target

[Service]

Type=forking

ExecStart=/usr/local/tomcat8/bin/startup.sh

ExecStop=/usr/local/tomcat8/bin/shutdown.sh

User=tomcat8

Group=tomcat8

[Install]

WantedBy=multi-user.target

chown -R tomcat8:tomcat8 tomcat8/

systemctl enable tomcat8 && systemctl start tomcat8

**另一种方法：**

在/etc/init.d目录下，创建tomcat7或者tomcat8文件，然后粘贴如下内容，需要注意如下内容中红色字体的修改，要符合部署环境。

#!/bin/sh

# chkconfig: 345 99 10

# description: Auto-starts tomcat

# /etc/init.d/tomcatd

# Tomcat auto-start

# Source function library.

#. /etc/init.d/functions

# source networking configuration.

#. /etc/sysconfig/network

RETVAL=0

export JDK\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.262.b10-0.el7\_8.x86\_64/jre

export CATALINA\_HOME=/usr/local/tomcat8

export CATALINA\_BASE=/usr/local/tomcat8

export TOMCAT8\_USER=tomcat8

export TOMCAT8\_GROUP=tomcat8

start()

{

if [ -f $CATALINA\_HOME/bin/startup.sh ];

then

echo $"Starting Tomcat"

$CATALINA\_HOME/bin/startup.sh

RETVAL=$?

echo " OK"

return $RETVAL

fi

}

stop()

{

if [ -f $CATALINA\_HOME/bin/shutdown.sh ];

then

echo $"Stopping Tomcat"

$CATALINA\_HOME/bin/shutdown.sh

RETVAL=$?

sleep 1

echo " OK"

ps -ef | grep tomcat|grep -v grep | grep -v PID | awk '{print $2}'|xargs kill -9

# [ $RETVAL -eq 0 ] && rm -f /var/lock/...

return $RETVAL

fi

}

case "$1" in

start)

start

;;

stop)

stop

;;

restart)

echo $"Restaring Tomcat"

$0 stop

sleep 1

$0 start

;;

\*)

echo $"Usage: $0 {start|stop|restart}"

exit 1

;;

esac

exit $RETVAL

然后设置开机自启：

chkconfig tomcat7 或tomcat8 on

或者

systemctl enable tomcat7 或tomcat8

修改/etc/tomcat/目录中server.xml文件，修改HTTP的connector：

<Connector port="8080" protocol="HTTP/1.1" maxHttpHeaderSize="81920" URIEncoding="UTF-8" maxThreads="2048" acceptCount="2048" maxConnections="65536" redirectPort="8443" maxPostSize="10000000"/>

若使用的是AJP，则还需要修改：

<Connector port="8009" URIEncoding="UTF-8" maxThreads="2048" acceptCount="2048" maxConnections="65536" protocol="AJP/1.3" maxHttpHeaderSize="81920" redirectPort="8443" maxPostSize="10000000"/>

Tomcat localhost日志输出访问耗时参数 %D，增加到如下红色位置：

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log." suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b %D" />

修改/etc/tomcat/tomcat-users.xml文件，在下面增加：

<user username="jenkins" password="xeJMkIOe" roles="manager-script"/>

**修改Tomcat运行内存：**

cd到/opt/tomcat/bin 下，编辑catalina.sh文件，向中插入以下代码，注意如下是JDK1.7版本配置参数。

JAVA\_OPTS="$JAVA\_OPTS -Xms128m -Xmx6400m -XX:PermSize=512m -XX:MaxPermSize=2048m -Djava.security.egd=file:/dev/./urandom -Dfile.encoding=UTF8 -Duser.timezone=GMT+08"

根据内存不同Xmx参数不同，注意如下是JDK1.7版本配置参数。

JAVA\_OPTS="$JAVA\_OPTS -Xms128m -Xmx12800m -XX:PermSize=512m -XX:MaxPermSize=2048m -Djava.security.egd=file:/dev/./urandom -Dfile.encoding=UTF8 -Duser.timezone=GMT+08"

**注意：Xmx的配置视服务器内存而定，8G内存可配到6400m左右，16G内存可配置为12800m或14400但切不可超出**

**JDK1.8版本配置参数如下：**

JAVA\_OPTS="$JAVA\_OPTS -Xms128m -Xmx6400m -XX:MetaspaceSize=512m -XX:MaxMetaspaceSize=2048m -Djava.security.egd=file:/dev/./urandom -Dfile.encoding=UTF8 -Duser.timezone=GMT+08"

根据内存不同Xmx参数不同，注意如下是JDK1.8版本配置参数。

JAVA\_OPTS="$JAVA\_OPTS -Xms128m -Xmx12800m -XX:MetaspaceSize=512m -XX:MaxMetaspaceSize=2048m -Djava.security.egd=file:/dev/./urandom -Dfile.encoding=UTF8 -Duser.timezone=GMT+08"

### 2.5 MariaDB10.X安装配置

1.删除系统自带的mysql

rpm -qa|grep mysql

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum remove mysql-libs

2.创建/etc/yum.repos.d/MariaDB.repo文件，版本10.4，内容如下：

说明：如果需要安装MariaDB其它版本，则可以数据源文件的版本路径

[mariadb]  
name = MariaDB  
baseurl = C:\Users\Administrator\AppData\Roaming\Tencent\TIM\Temp\%W@GJ$ACOF(TYDYECOKVDYB.pnghttps://mirrors.ustc.edu.cn/mariadb/yum/10.4/centos/7/x86\_64/  
gpgcheck=0

3.关闭并卸载旧版本的mariadb，安装新版本的mariadb。

systemctl stop mariadb

yum remove mariadb-server mariadb mariadb-libs

rpm -qa|grep MariaDB

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum clean all

yum makecache

yum install MariaDB-client MariaDB-server MariaDB-devel -y

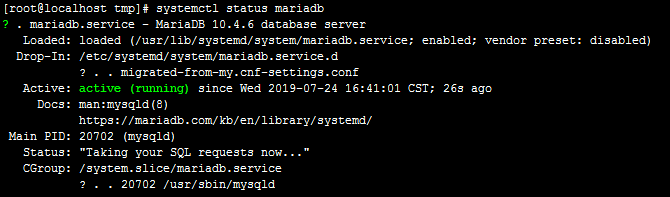
4.启动数据库并设置为开机自启

systemctl start mariadb

systemctl enable mariadb

5.查看MariaDB运行状态

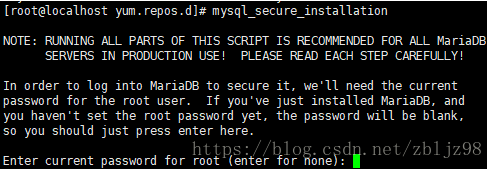
systemctl status mariadb

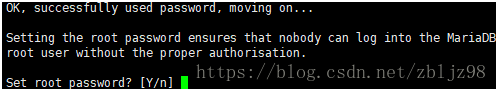


6、初始化数据库，并删除测试数据库及更改权限和设置密码

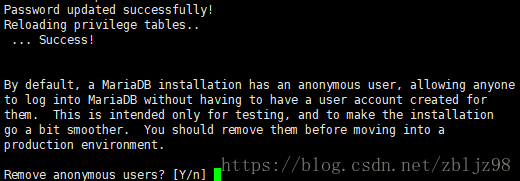
mysql\_secure\_installation

输入数据库设置密码

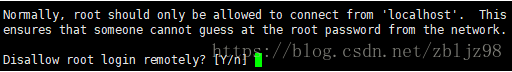




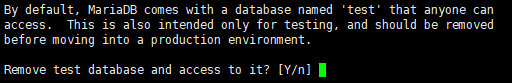
是否设置root密码，输入Y进行设置



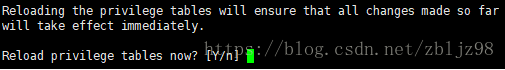
是否移除匿名用户，输入Y移除



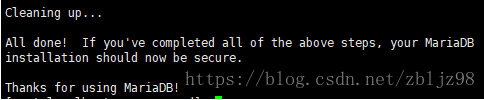
是否拒绝root用户的远程登陆，根据实际情况选择



是否删除test库，选择y



是否刷新权限表，输入Y刷新权限表



出现此界面，数据库安全设置完成。

注意：如果上述设置root密码失败，则使用如下方法初始化root密码。

首先停掉MariaDB服务：systemctl stop mariadb

然后修改/etc/my.cnf.d目录中server.cnf文件，在[mysqld]节中增加skip-grant-tables

然后启动数据库：systemctl start mariadb

用空密码进入mysql管理命令行，切换到mysql库。

abbuggy@abbuggy-ubuntu:~$ mysql

Welcome to the MySQL monitor.  Commands end with ; or \g.

mysql> use mysql;

Database changed

mysql> UPDATE user SET authentication\_string = password ( '密码' ) WHERE User = 'root' ;

Query OK, 0 rows affected (0.00 sec)

#或者，使用 set 指令设置root密码：

MariaDB [(none)]> SET password for 'root'@'localhost'=password('newpassword');

mysql>flush privileges;

mysql>quit

回到vim /etc/my.cnf.d/server.cnf，把刚才加入的那一行“skip-grant-tables”注释或删除掉。

再次重启MariaDB服务

systemctl restart mariadb

$ mysql -uroot -p

再次设置密码：

mysql>SET PASSWORD = PASSWORD('密码');

mysql>flush privileges;

7.连接数据库，并查询版本

MariaDB [(none)]> select version();

+-----------------+

| version()       |

+-----------------+

| 10.4.14-MariaDB |

+-----------------+

1 row in set (0.000 sec)

8.优化数据库配置（注意参数不可以copy直接用，一定要清楚配置内容，根据操作系统环境而定）

/etc/my.cnf.d/server.cnf文件中，在[server]节中配置内容如下

user= mysql

default-storage-engine=InnoDB

socket=/var/lib/mysql/mysql.sock

pid-file=/var/lib/mysql/mysql.pid

character-set-server=utf8

datadir=/var/lib/mysql/

log-bin=/var/lib/mysql/mysql-bin

expire-logs-days=14

sync-binlog=1

binlog\_format = MIXED

innodb\_flush\_method =O\_DIRECT

innodb-log-files-in-group =2

innodb\_log\_file\_size=256M

innodb-flush-log-at-trx-commit =1

innodb\_file\_per\_table=1

innodb\_buffer\_pool\_size=6G

tmp\_table\_size=32M

max\_heap\_table\_size=32M

max\_connections=2000

max\_connect\_errors =100000

query-cache-size=128M

query\_cache\_type=1

thread-cache-size=120

open-files-limit= 65535

table-definition-cache=4096

table-open-cache=4096

max\_allowed\_packet=1024M

ft\_min\_word\_len = 1

log-error=/home/mysqllog/mysql-error.log

slow\_query\_log=1

long\_query\_time=10

log\_queries\_not\_using\_indexes=1

slow\_query\_log\_file=/home/mysqllog/localhost-slow.log

log\_error=/var/lib/mysql/error.log  
innodb\_print\_all\_deadlocks=1

**MariaDB的卸载**

重要卸载命令：

yum --setopt=tsflags=noscripts remove MariaDB-\*

### 2.6 Redis安装配置

1）安装redis-server

yum install epel-release

yum install redis

systemctl enable redis.service

/bin/systemctl start redis.service

2）防止服务器重启后，redis数据丢失，对redis配置如下

找到 /etc/sysctl.conf

vim /etc/sysctl.conf

在最后一行添加

vm.overcommit\_memory = 1

wq保存，用sysctl -p 使配置生效

sysctl -p

注意，上述问题只在Linux系统才存在，在Windows系统是没有问题的。

3）性能优化配置

4）安全优化配置

5）自动备份脚本

#!/bin/bash

echo "start..."

sdate=$(date "+ %Y-%m-%d %H:%M:%S")

msg=`redis-cli -p $1 -a $2 bgsave`

echo "start bgsave time:$sdate"

result=`redis-cli -p $1 -a $2 info Persistence | grep "rdb\_bgsave\_in\_progress" | awk -F":" '{print $2}'`

while [ `echo ${result} | awk -v tem="0" '{print($1>tem)? "1":"0"}'` -eq "1" ] ; do

    sleep 1

    result=`redis-cli -p $1 -a $2 info Persistence | grep "rdb\_bgsave\_in\_progress" | awk -F":" '{print $2}'`

done

edate=$(date "+ %Y-%m-%d %H:%M:%S")

echo "end bgsave time:$edate"

echo "renaming rdb file..."

date=$(date "+ %Y%m%d")

mv "../db03/dump.rdb" "../db03/${date}.rdb"

echo "finish backup rdb file"

echo "end"

### 2.7 MySQL5.6安装配置

1、删除系统自带的mysql

rpm -qa|grep mysql

yum remove mysql-libs

重要卸载命令：

yum --setopt=tsflags=noscripts remove MariaDB-\*

2、安装mysql YUM源

wget http://repo.mysql.com/mysql-community-release-el6-5.noarch.rpm

rpm -ivh mysql-community-release-el6-5.noarch.rpm

3、安装mysql

yum install mysql-server mysql-devel

4、root密码设置

service mysqld stop

mysqld\_safe --skip-grant-tables &

mysql -u root -p //回车直接进去，或者随便输入一个密码

use mysql;

update user set password=PASSWORD("8sDf3sfe@")where user="root"; //更改密码为 newpassord

flush privileges; //更新权限

quit

创建数据库脚本：

create database `数据库实例名称` character set 'utf8' collate 'utf8\_general\_ci';

grant all privileges on `数据库实例名称`.\* to 用户@'localhost' identified by '密码';

grant all privileges on `数据库实例名称`.\* to 用户@'%' identified by '密码';

flush privileges;

MySQL数据库主从配置方法：

分别修改master和slave机器中mysql配置文件my.cnf，该文件在/etc目录下 在[mysqld]配置段添加如下字段

**Master：**

server-id=1

log-bin=mysql-bin

binlog\_format = mixed

binlog-do-db=infoplus-v2 //需要同步的数据库，如果没有本行，即表示同步所有的数据库

binlog-ignore-db=mysql //被忽略的数据库

**slave：**

server-id=2

replicate-do-db=infoplus-v2 //同步的数据库，不写本行表示同步所有数据库

保存之后重启mysql服务

在master机上为slave机添加一同步帐号 用户为slave

grant replication slave on \*.\* to 'slave'@'slave服务器ip' identified by '密码';

flush privileges;

在master上执行命令查看master状态

mysql>show master status;

+-------------------------+---------------+-------------------+----------------------------+------------------------+

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |

+-------------------------+---------------+-------------------+----------------------------+------------------------+

| mysql-bin.000001 | 1533860 | infoplus-v2 | mysql | |

在slave上执行命令，使slave与master同步

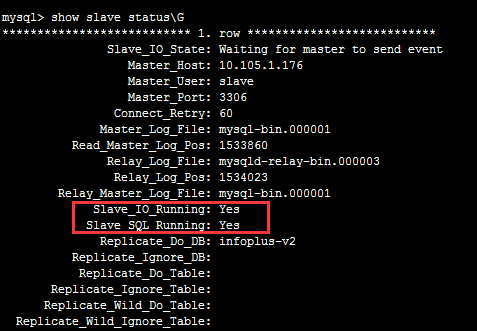
mysql>stop slave;

mysql>change master to master\_host='master服务器ip',master\_user='同步账号',master\_password='同步账号密码',master\_log\_file='master上的File值',master\_log\_pos=master上的Position值;

mysql>start slave;

查看slave连接状态，如果Slave\_IO\_Running、Slave\_SQL\_Running状态为Yes则设置成功。

mysql>show slave status\G



**设置日志保留天数**

在my.cnf文件中配置：

expire\_logs\_days = 3

表示保留3天的日志，这样老日志会自动被清理掉。

在my.cnf文件中增加expire\_logs\_days = 3参数后，若不能重启数据库，则通过如下命令来设置，即时生效：

set global expire\_logs\_days = 3;

附：查看日志方法

show binary logs;

show variables like '%log%';

set global expire\_logs\_days = 3;

首先创建备份目录

mkdir /opt/bak

chmod 711 bak

mkdir /opt/bak/bakmysql

chmod 711 bakmysql

mkdir /opt/bak/bakmysqlold

chmod 711 bakmysqlold

编写备份脚本

vim /opt/bak/backup.sh

chmod 777 /opt/bak/backup.sh

脚本内容：

#!/bin/sh

BK\_DR=/opt/bak/bakmysqlold

DAYS=7

cd /opt/bak/bakmysql

echo "You are in bakmysql directory"

mv infoplus-v2\* /opt/bak/bakmysqlold

echo "Old databases are moved to bakmysqlold folder"

Now=$(date +"%Y-%m-%d\_%H-%M-%S")

File=infoplus-v2-$Now.sql

mysqldump -uinfoplus -p'从数据库密码' infoplus-v2 gzip > $File.gz

find $BK\_DR -name "infoplus-v2\*" -type f -mtime +$DAYS -exec rm {} \;

定时备份设置

执行crontab -e 进行编辑

内容如下（每天凌晨1点，中午12点，下午18点开始执行备份脚本）：

00 01 \* \* \* /opt/bak/backup.sh

00 12 \* \* \* /opt/bak/backup.sh

00 18 \* \* \* /opt/bak/backup.sh

保存退出 并重新加载

service crond reload

### 2.8 MySQL5.7安装配置

删除系统自带的mysql

rpm -qa|grep mysql

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum remove mysql-libs

# 下载mysql5.7源安装包

shell> wget http://dev.mysql.com/get/mysql57-community-release-el7-8.noarch.rpm

# 安装mysql源

shell> yum localinstall mysql57-community-release-el7-8.noarch.rpm

检查mysql源是否安装成功

shell> yum repolist enabled | grep "mysql.\*-community.\*"

2、安装MySQL

shell> yum install mysql-community-server

3、启动MySQL服务

shell> systemctl start mysqld

查看MySQL的启动状态

shell> systemctl status mysqld

4、开机启动

shell> systemctl enable mysqld

shell> systemctl daemon-reload

5、修改root本地登录密码

mysql安装完成之后，在/var/log/mysqld.log文件中给root生成了一个默认密码。通过下面的方式找到root默认密码，然后登录mysql进行修改：

shell> grep 'temporary password' /var/log/mysqld.log

shell> mysql -u root -p

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';

或者

mysql> set password for 'root'@'localhost'=password('MyNewPass4!');

注意：mysql5.7默认安装了密码安全检查插件（validate\_password），默认密码检查策略要求密码必须包含：大小写字母、数字和特殊符号，并且长度不能少于8位。否则会提示ERROR 1819 (HY000): Your password does not satisfy the current policy requirements错误

MySQL5.7 关闭参数ONLY\_FULL\_GROUP\_BY

临时关闭方法：（命令行）

set GLOBAL sql\_mode ='STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION ';

彻底关闭方法：（修改配置文件）

在my.cnf里面的 [mysqld]下添加

set @@sql\_mode = 'STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION'

然后重启MySQL,再查看ONLY\_FULL\_GROUP\_BY已经不见了：

SELECT @@GLOBAL.sql\_mode;和SELECT @@SESSION.sql\_mode;

### 2.9 Memcached安装配置

#### 1）查找Memcached

yum search memcached

首先检查yum软件仓库中是否存在memcached，如果有，直接进入第3步安装即可，否则执行第2步。

#### 2）安装第三方软件库（可选）

标准的CentOS5软件仓库里面是没有memcache相应的包的,所以，我们的第一步就是导入第三方软件仓库，这里推荐的是RpmForge（RpmForge库现在已经拥有超过10000种的CentOS的软件包，被CentOS社区认为是最安全也是最稳定的一个第三方软件库），安装方法如下：

wget http://dag.wieers.com/rpm/packages/rpmforge-release/rpmforge-release-0.5.2-2.rf.src.rpm

rpm -ivh rpmforge-release-0.5.2-2.rf.src.rpm

#### 3）安装Memcached

yum -y install memcached

#### 4）验证安装

memcached -h

应该会输出一些帮助信息

#### 5）将memcache加入启动列表

chkconfig --level 2345 memcached on

#### 6）配置Memcache

vi /etc/sysconfig/memcached

文件中内容如下

PORT=”11211″ 端口

USER=”root” 使用的用户名

MAXCONN=”1024″ 同时最大连接数

CACHESIZE=”64″ 使用的内存大小

#OPTIONS="" 附加参数//默认是本机访问memcached，可以利用此参数设置其他服务器访问，配置参数为

OPTIONS="-l 0.0.0.0"

然后启动memcached服务：

service memcached start

#### 7）查看memcache状态

memcached-tool [Memcache Server IP]:[Memcache Server Port] stats

如：memcached-tool 127.0.0.1:11211 stats

### 2.10 OpenLDAP安装配置

#### 1）安装程序

# yum install -y openldap openldap-clients openldap-servers migrationtools

#### 2）配置OpenLDAP Server

vim /etc/openldap/slapd.d/cn\=config/olcDatabase\=\{2\}hdb.ldif

修改两行:

#change dc=yooma

例如：

olcSuffix: dc=jsit,dc=edu

olcRootDN: cn=Manager,dc=jsit,dc=edu

添加一行:

olcRootPW: 123456

#密码根据自己需要修改

:wq!

#### 3）配置Monitoring Database 文件

vim /etc/openldap/slapd.d/cn\=config/olcDatabase\=\{1\}monitor.ldif

#修改dn.base=""中的cn、dc项与step2中的相同

olcAccess: {0}to \* by dn.base="gidNumber=0+uidNumber=0,cn=peercred,cn=extern

al,cn=auth" read by dn.base=" cn=Manager,dc=jsit,dc=edu " read by \* none

:wq!

#### 4）配置OpenLDAP数据库

cp /usr/share/openldap-servers/DB\_CONFIG.example /var/lib/ldap/DB\_CONFIG

分配数据库属主属组：

chown -R ldap.ldap /var/lib/ldap

#### 5）测试配置文件

slaptest -u

提示如下：

56e7c83d ldif\_read\_file: checksum error on "/etc/openldap/slapd.d/cn=config/olcDatabase={1}monitor.ldif"

56e7c83d ldif\_read\_file: checksum error on "/etc/openldap/slapd.d/cn=config/olcDatabase={2}hdb.ldif"

config file testing succeeded

#验证成功

#### 6）重启slapd服务并设置开机自启动

systemctl start slapd

systemctl enable slapd

#### 7）核实LDAP 状态:

命令：

netstat -lt | grep ldap

输出：

tcp 0 0 0.0.0.0:ldap 0.0.0.0:\* LISTEN

tcp6 0 0 [::]:ldap [::]:\* LISTEN

命令：

netstat -tunlp | egrep "389|636"

输出：

tcp 0 0 0.0.0.0:389 0.0.0.0:\* LISTEN 18814/slapd

tcp6 0 0 :::389 :::\* LISTEN 18814/slapd

#### 8）添加LDAP schemas

cd /etc/openldap/schema/

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f cosine.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f nis.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f collective.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f corba.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f core.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f duaconf.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f dyngroup.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f inetorgperson.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f java.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f misc.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f openldap.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f pmi.ldif

# ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f ppolicy.ldif

#注意事项: You can add schema files according to your need: #

#### 9）使用Migration工具创建 LDAP DIT

命令：

cd /usr/share/migrationtools/

命令：

vim migrate\_common.ph

修改如下内容：

在61行

$NAMINGCONTEXT{'group'} = "ou=Groups";

在 71行,

$DEFAULT\_MAIL\_DOMAIN = "jsit.edu";

在74行

$DEFAULT\_BASE = "dc=jsit,dc=edu";

在 90行

$EXTENDED\_SCHEMA = 1;

:wq!

#### 10）生成 base.ldif 文件

命令：

./migrate\_base.pl >/root/base.ldif

#### 11）导入"base.ldif"到 LDAP 数据库

命令：

ldapadd -x -W -D " cn=Manager,dc=jsit,dc=edu" -f /root/base.ldif

#### 12）开启OpenLDAP日志访问功能

默认情况下OpenLDAP是没有启用日志记录功能的，但是在实际使用过程中，我们为了定位问题需要使用到OpenLDAP日志。

新建日志配置ldif文件，如下：

vim/root/loglevel.ldif

添加如下内容：

dn: cn=config

changetype: modify

replace: olcLogLevel

olcLogLevel: stats

导入到OpenLDAP中，并重启OpenLDAP服务，如下：

ldapmodify -Y EXTERNAL -H ldapi:/// -f /root/loglevel.ldif

重启slapd

systemctl restart slapd

修改rsyslog配置文件，并重启rsyslog服务，如下：

vim/etc/rsyslog.conf

添加如下内容：

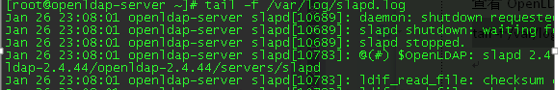
local4.\* /var/log/ldap/slapd.log

重启rsyslog

systemctl restart rsyslog

查看OpenLDAP日志，如下：

tail -f /var/log/slapd.log



#### 13）添加activity eq自定义索引

1. 创建一个 ldif 文件，用于存放索引命令

命令：

cat >> hdb-index.ldif << EOF

dn: olcDatabase={2}hdb,cn=config

changetype: modify

add: olcDbIndex

olcDbIndex: activity eq

EOF

2. 通过 ldapmodify 命令创建 olcDatabase={2}hdb 数据库相关索引条目。

命令：

ldapmodify -Q -Y EXTERNAL -H ldapi:/// -f hdb-index.ldif

3. 通过 ldapsearch 进行验证，是否已经添加成功

ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b cn=config '(olcDatabase={2}hdb)' olcDbIndex

### 2.11 Nginx安装配置

### 2.12 PHP安装配置

### 2.13 NFS安装配置

系统环境：CentOS Linux release 7.4.1708 (Core) 3.10.0-693.el7.x86\_64

软件版本：nfs-utils-1.3.0-0.48.el7\_4.x86\_64

网络环境中配置了DNS服务器，NFS服务器对应的域名是nfs.st.local，IP是192.168.1.18。配置过程中全部使用域名。

一、安装

nfs客户端和服务端都安装nfs-utils包，同时自动安装rpcbind。安装后会创建nfsnobody用户和组，uid和gid都是65534。

# yum -y install nfs-utils

二、配置端口

nfs除了主程序端口2049和rpcbind的端口111是固定以外，还会使用一些随机端口，以下配置将定义这些端口，以便配置防火墙

# vim /etc/sysconfig/nfs

#追加端口配置

MOUNTD\_PORT=4001

STATD\_PORT=4002

LOCKD\_TCPPORT=4003

LOCKD\_UDPPORT=4003

RQUOTAD\_PORT=4004

三、NFS权限说明

1、普通用户

当设置all\_squash时：访客时一律被映射为匿名用户(nfsnobody)

当设置no\_all\_squash时：访客被映射为服务器上相同uid的用户，因此在客户端应建立与服务端uid一致的用户，否则也映射为nfsnobody。root除外，因为root\_suqash为默认选项，除非指定了no\_root\_squash

2、root用户

当设置root\_squash时：访客以root用户访问NFS服务端时，被映射为nfsnobody用户

当设置no\_root\_squash时：访客以root用户访问NFS服务端时，被映射为root用户。以其他用户访问时同样映射为对应uid的用户，因为no\_all\_squash是默认选项

选项说明

ro：共享目录只读

rw：共享目录可读可写

all\_squash：所有访问用户都映射为匿名用户或用户组

no\_all\_squash（默认）：访问用户先与本机用户匹配，匹配失败后再映射为匿名用户或用户组

root\_squash（默认）：将来访的root用户映射为匿名用户或用户组

no\_root\_squash：来访的root用户保持root帐号权限

anonuid=<U ID>：指定匿名访问用户的本地用户UID，默认为nfsnobody（65534）

anongid=<GID>：指定匿名访问用户的本地用户组GID，默认为nfsnobody（65534）

secure（默认）：限制客户端只能从小于1024的tcp/ip端口连接服务器

insecure：允许客户端从大于1024的tcp/ip端口连接服务器

sync：将数据同步写入内存缓冲区与磁盘中，效率低，但可以保证数据的一致性

async：将数据先保存在内存缓冲区中，必要时才写入磁盘

wdelay（默认）：检查是否有相关的写操作，如果有则将这些写操作一起执行，这样可以提高效率

no\_wdelay：若有写操作则立即执行，应与sync配合使用

subtree\_check（默认） ：若输出目录是一个子目录，则nfs服务器将检查其父目录的权限

no\_subtree\_check ：即使输出目录是一个子目录，nfs服务器也不检查其父目录的权限，这样可以提高效率

以nfsuser(uid=1000)创建共享目录，参数默认rw

# mkdir /var/nfs

# chown nfsuser. -R /var/nfs

# vim /etc/exports

/var/nfs 192.168.1.0/24(rw)

# exportfs -r　　#重载exports配置

# exportfs -v　　#查看共享参数

/var/nfs 192.168.1.0/24(rw,sync,wdelay,hide,no\_subtree\_check,sec=sys,secure,root\_squash,no\_all\_squash)

复制代码

exportfs参数说明

-a 全部挂载或卸载 /etc/exports中的内容

-r 重新读取/etc/exports 中的信息 ，并同步更新/etc/exports、/var/lib/nfs/xtab

-u 卸载单一目录（和-a一起使用为卸载所有/etc/exports文件中的目录）

-v 输出详细的共享参数

四、防火墙

复制代码

# iptables -I INPUT 5 -p tcp -m tcp --dport 111 -j ACCEPT

# iptables -I INPUT 5 -p udp -m udp --dport 111 -j ACCEPT

# iptables -I INPUT 5 -p tcp -m tcp --dport 2049 -j ACCEPT

# iptables -I INPUT 5 -p udp -m udp --dport 2049 -j ACCEPT

# iptables -I INPUT 5 -p tcp -m tcp --dport 4001:4004 -j ACCEPT

# iptables -I INPUT 5 -p udp -m udp --dport 4001:4004 -j ACCEPT

# iptables-save >/etc/sysconfig/iptables

五、启动服务

# systemctl start rpcbind.service

# systemctl enable rpcbind.service

# systemctl start nfs.service

# systemctl enable nfs.service

启动顺序一定是rpcbind->nfs，否则有可能出现错误

六、Linux客户端挂载

1、直接挂载

# mount -t nfs nfs.st.local:/var/nfs /mnt

也可将挂载配置写入fstab文件中，与普通磁盘挂载一样，挂载时同样可以指定权限，只是类型为nfs。

2、autofs挂载

# yum -y install autofs

# vi /etc/auto.master

#添加一行

/- /etc/auto.mount

# vi /etc/auto.mount

#添加一行

/mnt -fstype=nfs,rw nfs.st.local:/var/nfs

#启动服务

# systemctl start autofs

# systemctl enable autofs

七、故障解决

1、nfs只能挂载为nobody

同时修改服务端、客户端/etc/idmapd.conf中的Domain为一样的值，随后重启rpcidmapd服务，或重启所有服务

2、客户端无法卸载nfs目录

umount.nfs4: /var/nfs: device is busy

执行fuser -km /var/nfs/，然后再执行umount

## 3.Ubuntu 14.04.4 LTS

### 3.1 操作系统配置

apt-get update

apt-get upgrade

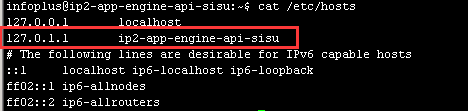
apt-get dist-upgrade

修改主机名

vim /etc/hostname



vim /etc/hosts



时间和时区同步

#date 查看系统时间



#date -s 修改日期



#date -s 修改时间



修改完后执行：clock -w ,强制将时间写入COMS！



修改时区：（将Asia/shanghai-上海时区写入当前时区）

#cp -f /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

提示是否覆盖,输入Y回车

然后#date

查看时区和时间（CST,中国时区）



增加infoplus用户

输入以下命令，根据提示输入密码及相关信息：

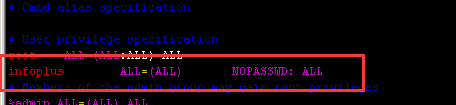
# adduser infoplus

将infoplus用户加入到adm和sudo组中，如下图：

vi /etc/group

修改/etc/sudoers，添加：infoplus ALL=(ALL) NOPASSWD: ALL

如下图：



### 3.2 Apache（httpd）安装配置

apachectl -V | grep -i mpm

接下来，针对相应的模式，进行配置。apache2的默认配置文件位于/etc/apache2/apache2.conf，不同模式对应的配置文件在/etc/apache2/mods-available/下，有mpm\_prefork.conf， mpm\_worker.conf和mpm\_event.conf。

Apache(httpd)编译安装配置

### 3.3 JDK安装配置

JDK默认是安装完成的，如下截图：



### 3.4 Tomcat安装配置

#sudo apt-get install tomcat7

#sudo apt-get install tomcat7-admin

需修改三个配置文件（还有一个可选）：

**配置/etc/tomcat7/server.xml文件：**

修改HTTP的connector：

<Connector

port="8080" protocol="HTTP/1.1" URIEncoding="UTF-8"

maxThreads="2048" acceptCount="2048" maxConnections="65536" redirectPort="8443" />

**若安装的是Apache，则还需要修改：**

<Connector port="8009" URIEncoding="UTF-8" maxThreads="2048" acceptCount="2048" maxConnections="65536" protocol="AJP/1.3" redirectPort="8443" />

Tomcat日志输出增加%D

#### 3.4.X Tomcat编译安装配置

### 3.5 MariaDB10.3安装配置

1.删除系统自带的mysql

rpm -qa|grep mysql

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum remove mysql-libs

2.创建/etc/yum.repos.d/MariaDB.repo文件，版本未10.3，内容如下：

[mariadb]

name = MariaDB

baseurl = <http://yum.mariadb.org/10.3/centos7-amd64>

gpgkey=<https://yum.mariadb.org/RPM-GPG-KEY-MariaDB>

gpgcheck=1

3.关闭并卸载旧版本的mariadb，安装新版本的mariadb。

systemctl stop mariadb

yum remove mariadb-server mariadb mariadb-libs

rpm -qa|grep MariaDB

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum clean all

yum makecache

yum install MariaDB-client MariaDB-server MariaDB-devel -y

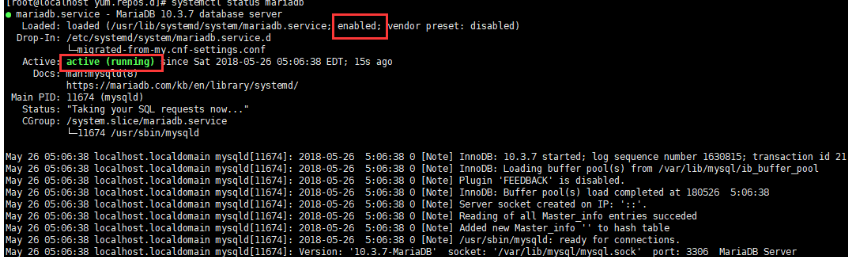
4.启动数据库并设置为开机自启

systemctl start mariadb

systemctl enable mariadb

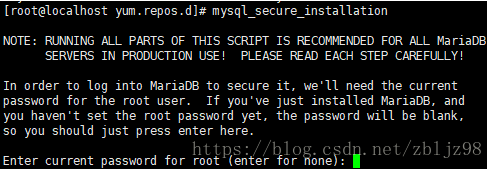
5.查看MariaDB运行状态

systemctl status mariadb

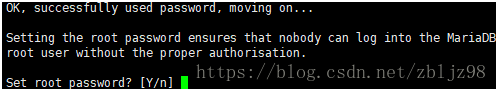


6、初始化数据库，并删除测试数据库及更改权限和设置密码

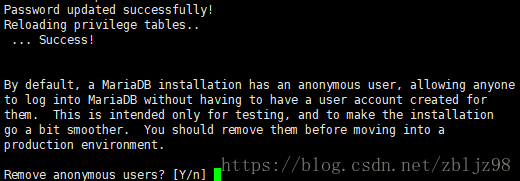
mysql\_secure\_installation



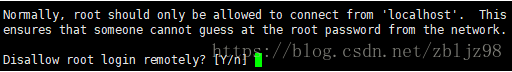
输入数据库设置密码



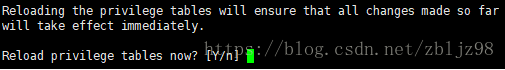
是否设置root密码，输入Y进行设置



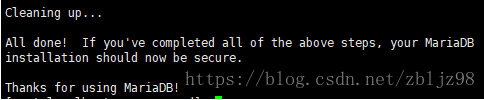
是否移除匿名用户，输入Y移除



是否拒绝root用户的远程登陆，根据实际情况选择



是否刷新权限表，输入Y刷新权限表



出现此界面，数据库安全设置完成。

注意：如果上述设置root密码失败，则使用如下方法初始化root密码。

首先停掉MariaDB服务：systemctl stop mariadb

然后修改/etc/my.cnf.d目录中server.cnf文件，在[mysqld]节中增加skip-grant-tables

然后启动数据库：systemctl start mariadb

用空密码进入mysql管理命令行，切换到mysql库。

abbuggy@abbuggy-ubuntu:~$ mysql

Welcome to the MySQL monitor.  Commands end with ; or \g.

mysql> use mysql;

Database changed

mysql> UPDATE user SET authentication\_string = password ( '密码' ) WHERE User = 'root' ;

Query OK, 0 rows affected (0.00 sec)

mysql>flush privileges;

mysql>quit

回到vim /etc/my.cnf.d/server.cnf，把刚才加入的那一行“skip-grant-tables”注释或删除掉。

再次重启MariaDB服务

systemctl restart mariadb

$ mysql -uroot -p8sDf3sfe

再次设置密码：

mysql>SET PASSWORD = PASSWORD('密码');

mysql>flush privileges;

7.连接数据库，并查询版本

MariaDB [(none)]> select version();

+-----------------+

| version()       |

+-----------------+

| 10.3.14-MariaDB |

+-----------------+

1 row in set (0.000 sec)

8.优化数据库配置（注意参数不可以copy直接用，一定要清楚配置内容，根据操作系统环境而定）

/etc/my.cnf.d/server.cnf文件中，在[server]节中配置内容如下

user= mysql

default-storage-engine=InnoDB

socket=/var/lib/mysql/mysql.sock

pid-file=/var/lib/mysql/mysql.pid

character-set-server=utf8

datadir=/var/lib/mysql/

log-bin=/var/lib/mysql/mysql-bin

expire-logs-days=14

sync-binlog=1

innodb\_flush\_method =O\_DIRECT

innodb-log-files-in-group =2

innodb\_log\_file\_size=256M

innodb-flush-log-at-trx-commit =1

innodb\_file\_per\_table=1

innodb\_buffer\_pool\_size=20G

tmp\_table\_size=32M

max\_heap\_table\_size=32M

max\_connections=2000

max\_connect\_errors =100000

query-cache-size=128M

query\_cache\_type=1

thread-cache-size=120

open-files-limit= 65535

table-definition-cache=4096

table-open-cache=4096

max\_allowed\_packet=50M

binlog\_format = MIXED

log-error=/home/mysqllog/mysql-error.log

slow\_query\_log=1

long\_query\_time=10

log\_queries\_not\_using\_indexes=1

slow\_query\_log\_file=/home/mysqllog/localhost-slow.log

### 3.6 Redis安装配置

1）安装redis

2）防止服务器重启后，redis数据丢失，对redis配置如下

找到 /etc/sysctl.conf

vim /etc/sysctl.conf

在最后一行添加

vm.overcommit\_memory = 1

wq保存，用sysctl -p 使配置生效

sysctl -p

注意，上述问题只在Linux系统才存在，在Windows系统是没有问题的。

3）性能优化配置

4）安全优化配置

5）自动备份脚本

#!/bin/bash

echo "start..."

sdate=$(date "+ %Y-%m-%d %H:%M:%S")

msg=`redis-cli -p $1 -a $2 bgsave`

echo "start bgsave time:$sdate"

result=`redis-cli -p $1 -a $2 info Persistence | grep "rdb\_bgsave\_in\_progress" | awk -F":" '{print $2}'`

while [ `echo ${result} | awk -v tem="0" '{print($1>tem)? "1":"0"}'` -eq "1" ] ; do

    sleep 1

    result=`redis-cli -p $1 -a $2 info Persistence | grep "rdb\_bgsave\_in\_progress" | awk -F":" '{print $2}'`

done

edate=$(date "+ %Y-%m-%d %H:%M:%S")

echo "end bgsave time:$edate"

echo "renaming rdb file..."

date=$(date "+ %Y%m%d")

mv "../db03/dump.rdb" "../db03/${date}.rdb"

echo "finish backup rdb file"

echo "end"

### 3.6 MySQL5.6安装配置

### 3.7 Memcached安装配置

### 3.8 OpenLDAP安装配置

### 3.9 Nginx安装配置

### 3.10 PHP安装配置

### 3.11 NFS安装配置

### 3.12 cron定时任务

启动 cron 服务

通常 ubuntu 下自带 cron，如果没有也可以通过以下命令进行安装：

sudo apt-get install cron1

若已经安装，输入以下命令判断 cron 服务是否启动：

pgrep cron1

如果有 pid （一串数字）输出则说明 cron 服务已经启动，没有任何输出说明需要手动启动 cron 服务。

启动 cron 服务：

service cron start

ubuntu 14.04 安装jdk 1.8

一，如何删除低版本的open JDK？

在ubuntn的软件中心中，如果输入"Java"，我们会看到open JDK，但是最高版本是1.7，也有1.6版本的，如果我们安装上去，可能使用1.8JDK的程序就不能运行了，假设现在我们安装的是1.7版本的openSDK，这种情况下我们如何处理？

删除先前安装的open JDK：

我们可以在软件管理中心中删除，随后在shell中输入“java -version”，如果还没有彻底删除，那么在shell里面输入：“sudo apt-get autoremove openjdk-7-jre”，执行完这个命令之后，我们注销一下，随后在shell中输入“java -version”，我们会发现已经删除我们先前安装的1.7版本的openSDK。

二，安装1.8版本的JDK：我使用方式是从PPA源安装。

Oracle java的PPA源：https://launchpad.net/~webupd8team/+archive/ubuntu/java

安装步骤：在shell里依次执行下面命令。

1，sudo cp /etc/apt/sources.list /etc/apt/sources.list\_backup

2，sudo add-apt-repository ppa:webupd8team/java

3，sudo apt-get update

4，apt-get install oracle-java8-installer

然后选择yes，yes

经过上面操作后，我们 应该已经成功安装了oracle 1.8版本的JDK了。

检验安装结果：在shell里输入：“java -version”，如果安装成功就会输出JAVA的版本信息了，应该是1.8的了！！

若要启动tomcat7，会提示no JDK or JRE found - please set JAVA\_HOME，则需要配置下tomcat7启动的JAVA\_HOME，如下截图。

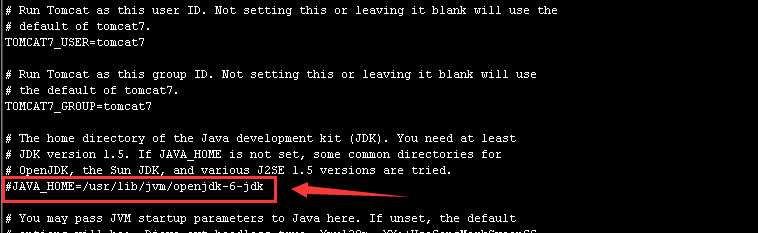
root@shu-pim-api2:/usr/lib/jvm/java-8-oracle# service tomcat7 start

\* no JDK or JRE found - please set JAVA\_HOME

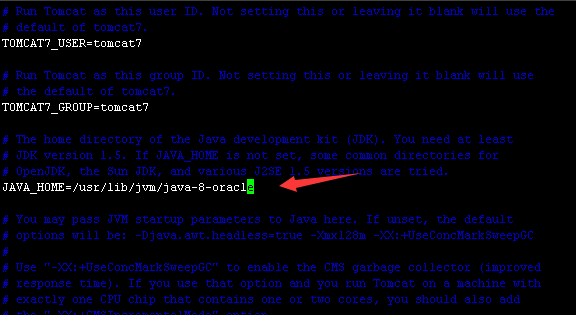
修改文件：

vim /etc/default/tomcat7

将如下截图的#去掉，然后配置正确的jdk目录（可以使用echo $JAVA\_HOME命令查看）



修改后截图如下：



配置完成后，再service tomcat7 start启动tomcat7就可以启动成功了。

## 4.Ubuntu 16.04.6 LTS

### 4.1 操作系统配置

### 4.2 Apache（httpd）安装配置

Apache(httpd)编译安装配置

### 4.3 JDK安装配置

### 4.4 Tomcat安装配置

Tomcat日志输出增加%D

#### 4.4.X Tomcat编译安装配置

### 4.5 MariaDB10.3安装配置

1.删除系统自带的mysql

rpm -qa|grep mysql

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum remove mysql-libs

2.创建/etc/yum.repos.d/MariaDB.repo文件，版本未10.3，内容如下：

[mariadb]

name = MariaDB

baseurl = <http://yum.mariadb.org/10.3/centos7-amd64>

gpgkey=<https://yum.mariadb.org/RPM-GPG-KEY-MariaDB>

gpgcheck=1

3.关闭并卸载旧版本的mariadb，安装新版本的mariadb。

systemctl stop mariadb

yum remove mariadb-server mariadb mariadb-libs

rpm -qa|grep MariaDB

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum clean all

yum makecache

yum install MariaDB-client MariaDB-server MariaDB-devel -y

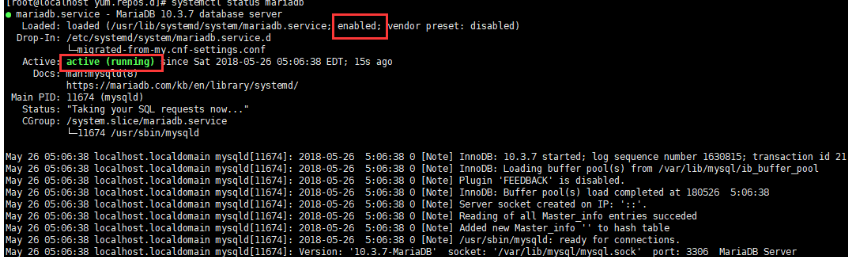
4.启动数据库并设置为开机自启

systemctl start mariadb

systemctl enable mariadb

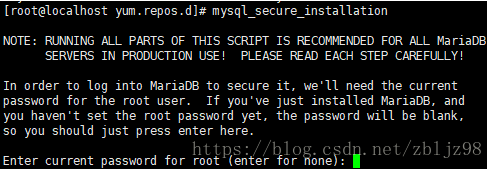
5.查看MariaDB运行状态

systemctl status mariadb

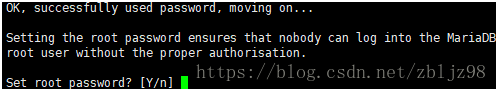


6、初始化数据库，并删除测试数据库及更改权限和设置密码

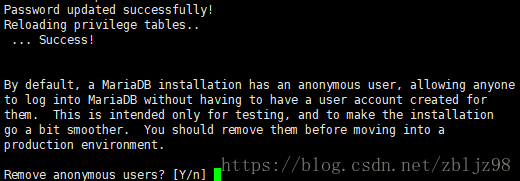
mysql\_secure\_installation



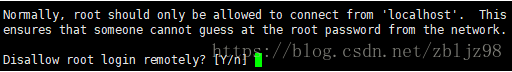
输入数据库设置密码



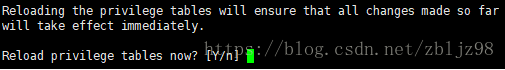
是否设置root密码，输入Y进行设置



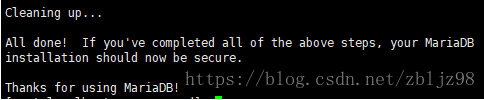
是否移除匿名用户，输入Y移除



是否拒绝root用户的远程登陆，根据实际情况选择



是否刷新权限表，输入Y刷新权限表



出现此界面，数据库安全设置完成。

注意：如果上述设置root密码失败，则使用如下方法初始化root密码。

首先停掉MariaDB服务：systemctl stop mariadb

然后修改/etc/my.cnf.d目录中server.cnf文件，在[mysqld]节中增加skip-grant-tables

然后启动数据库：systemctl start mariadb

用空密码进入mysql管理命令行，切换到mysql库。

abbuggy@abbuggy-ubuntu:~$ mysql

Welcome to the MySQL monitor.  Commands end with ; or \g.

mysql> use mysql;

Database changed

mysql> UPDATE user SET authentication\_string = password ( '密码' ) WHERE User = 'root' ;

Query OK, 0 rows affected (0.00 sec)

mysql>flush privileges;

mysql>quit

回到vim /etc/my.cnf.d/server.cnf，把刚才加入的那一行“skip-grant-tables”注释或删除掉。

再次重启MariaDB服务

systemctl restart mariadb

$ mysql -uroot -p8sDf3sfe

再次设置密码：

mysql>SET PASSWORD = PASSWORD('密码');

mysql>flush privileges;

7.连接数据库，并查询版本

MariaDB [(none)]> select version();

+-----------------+

| version()       |

+-----------------+

| 10.3.14-MariaDB |

+-----------------+

1 row in set (0.000 sec)

8.优化数据库配置（注意参数不可以copy直接用，一定要清楚配置内容，根据操作系统环境而定）

/etc/my.cnf.d/server.cnf文件中，在[server]节中配置内容如下

user= mysql

default-storage-engine=InnoDB

socket=/var/lib/mysql/mysql.sock

pid-file=/var/lib/mysql/mysql.pid

character-set-server=utf8

datadir=/var/lib/mysql/

log-bin=/var/lib/mysql/mysql-bin

expire-logs-days=14

sync-binlog=1

innodb\_flush\_method =O\_DIRECT

innodb-log-files-in-group =2

innodb\_log\_file\_size=256M

innodb-flush-log-at-trx-commit =1

innodb\_file\_per\_table=1

innodb\_buffer\_pool\_size=20G

tmp\_table\_size=32M

max\_heap\_table\_size=32M

max\_connections=2000

max\_connect\_errors =100000

query-cache-size=128M

query\_cache\_type=1

thread-cache-size=120

open-files-limit= 65535

table-definition-cache=4096

table-open-cache=4096

max\_allowed\_packet=50M

binlog\_format = MIXED

log-error=/home/mysqllog/mysql-error.log

slow\_query\_log=1

long\_query\_time=10

log\_queries\_not\_using\_indexes=1

slow\_query\_log\_file=/home/mysqllog/localhost-slow.log

### 4.6 Redis安装配置

1）安装redis

2）防止服务器重启后，redis数据丢失，对redis配置如下

找到 /etc/sysctl.conf

vim /etc/sysctl.conf

在最后一行添加

vm.overcommit\_memory = 1

wq保存，用sysctl -p 使配置生效

sysctl -p

注意，上述问题只在Linux系统才存在，在Windows系统是没有问题的。

3）性能优化配置

4）安全优化配置

5）自动备份脚本

#!/bin/bash

echo "start..."

sdate=$(date "+ %Y-%m-%d %H:%M:%S")

msg=`redis-cli -p $1 -a $2 bgsave`

echo "start bgsave time:$sdate"

result=`redis-cli -p $1 -a $2 info Persistence | grep "rdb\_bgsave\_in\_progress" | awk -F":" '{print $2}'`

while [ `echo ${result} | awk -v tem="0" '{print($1>tem)? "1":"0"}'` -eq "1" ] ; do

    sleep 1

    result=`redis-cli -p $1 -a $2 info Persistence | grep "rdb\_bgsave\_in\_progress" | awk -F":" '{print $2}'`

done

edate=$(date "+ %Y-%m-%d %H:%M:%S")

echo "end bgsave time:$edate"

echo "renaming rdb file..."

date=$(date "+ %Y%m%d")

mv "../db03/dump.rdb" "../db03/${date}.rdb"

echo "finish backup rdb file"

echo "end"

### 4.6 MySQL5.6安装配置

### 4.7 Memcached安装配置

### 4.8 OpenLDAP安装配置

### 4.9 Nginx安装配置

### 4.10 PHP安装配置

### 4.11 NFS安装配置

## 5.Ubuntu 18.04 LTS

### 5.1 操作系统配置

### 5.2 Apache（httpd）安装配置

Apache(httpd)编译安装配置

### 5.3 JDK安装配置

### 5.4 Tomcat安装配置

#### 5.4.X Tomcat编译安装配置

### 5.5 MariaDB10.3安装配置

1.删除系统自带的mysql

rpm -qa|grep mysql

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum remove mysql-libs

2.创建/etc/yum.repos.d/MariaDB.repo文件，版本未10.3，内容如下：

[mariadb]

name = MariaDB

baseurl = <http://yum.mariadb.org/10.3/centos7-amd64>

gpgkey=<https://yum.mariadb.org/RPM-GPG-KEY-MariaDB>

gpgcheck=1

3.关闭并卸载旧版本的mariadb，安装新版本的mariadb。

systemctl stop mariadb

yum remove mariadb-server mariadb mariadb-libs

rpm -qa|grep MariaDB

(有rpm包的，则使用rpm -e --nodeps进行卸载）

yum clean all

yum makecache

yum install MariaDB-client MariaDB-server MariaDB-devel -y

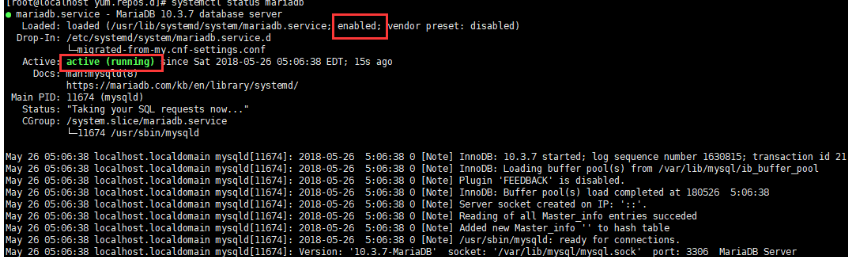
4.启动数据库并设置为开机自启

systemctl start mariadb

systemctl enable mariadb

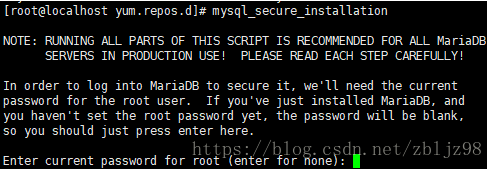
5.查看MariaDB运行状态

systemctl status mariadb

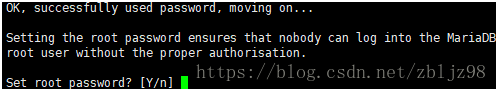


6、初始化数据库，并删除测试数据库及更改权限和设置密码

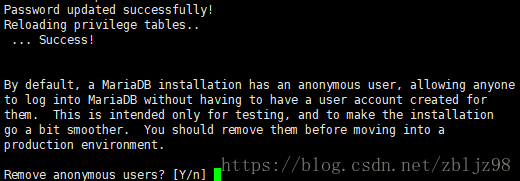
mysql\_secure\_installation



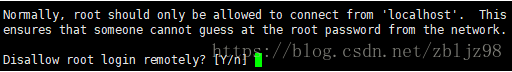
输入数据库设置密码



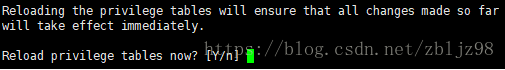
是否设置root密码，输入Y进行设置



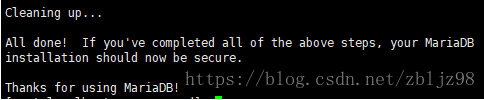
是否移除匿名用户，输入Y移除



是否拒绝root用户的远程登陆，根据实际情况选择



是否刷新权限表，输入Y刷新权限表



出现此界面，数据库安全设置完成。

注意：如果上述设置root密码失败，则使用如下方法初始化root密码。

首先停掉MariaDB服务：systemctl stop mariadb

然后修改/etc/my.cnf.d目录中server.cnf文件，在[mysqld]节中增加skip-grant-tables

然后启动数据库：systemctl start mariadb

用空密码进入mysql管理命令行，切换到mysql库。

abbuggy@abbuggy-ubuntu:~$ mysql

Welcome to the MySQL monitor.  Commands end with ; or \g.

mysql> use mysql;

Database changed

mysql> UPDATE user SET authentication\_string = password ( '密码' ) WHERE User = 'root' ;

Query OK, 0 rows affected (0.00 sec)

mysql>flush privileges;

mysql>quit

回到vim /etc/my.cnf.d/server.cnf，把刚才加入的那一行“skip-grant-tables”注释或删除掉。

再次重启MariaDB服务

systemctl restart mariadb

$ mysql -uroot -p8sDf3sfe

再次设置密码：

mysql>SET PASSWORD = PASSWORD('密码');

mysql>flush privileges;

7.连接数据库，并查询版本

MariaDB [(none)]> select version();

+-----------------+

| version()       |

+-----------------+

| 10.3.14-MariaDB |

+-----------------+

1 row in set (0.000 sec)

8.优化数据库配置（注意参数不可以copy直接用，一定要清楚配置内容，根据操作系统环境而定）

/etc/my.cnf.d/server.cnf文件中，在[server]节中配置内容如下

user= mysql

default-storage-engine=InnoDB

socket=/var/lib/mysql/mysql.sock

pid-file=/var/lib/mysql/mysql.pid

character-set-server=utf8

datadir=/var/lib/mysql/

log-bin=/var/lib/mysql/mysql-bin

expire-logs-days=14

sync-binlog=1

innodb\_flush\_method =O\_DIRECT

innodb-log-files-in-group =2

innodb\_log\_file\_size=256M

innodb-flush-log-at-trx-commit =1

innodb\_file\_per\_table=1

innodb\_buffer\_pool\_size=20G

tmp\_table\_size=32M

max\_heap\_table\_size=32M

max\_connections=2000

max\_connect\_errors =100000

query-cache-size=128M

query\_cache\_type=1

thread-cache-size=120

open-files-limit= 65535

table-definition-cache=4096

table-open-cache=4096

max\_allowed\_packet=50M

binlog\_format = MIXED

log-error=/home/mysqllog/mysql-error.log

slow\_query\_log=1

long\_query\_time=10

log\_queries\_not\_using\_indexes=1

slow\_query\_log\_file=/home/mysqllog/localhost-slow.log

### 5.6 Redis安装配置

1）安装Redis

2）防止服务器重启后，redis数据丢失，对redis配置如下

找到 /etc/sysctl.conf

vim /etc/sysctl.conf

在最后一行添加

vm.overcommit\_memory = 1

wq保存，用sysctl -p 使配置生效

sysctl -p

注意，上述问题只在Linux系统才存在，在Windows系统是没有问题的。

3）性能优化配置

4）安全优化配置

5）自动备份脚本

#!/bin/bash

echo "start..."

sdate=$(date "+ %Y-%m-%d %H:%M:%S")

msg=`redis-cli -p $1 -a $2 bgsave`

echo "start bgsave time:$sdate"

result=`redis-cli -p $1 -a $2 info Persistence | grep "rdb\_bgsave\_in\_progress" | awk -F":" '{print $2}'`

while [ `echo ${result} | awk -v tem="0" '{print($1>tem)? "1":"0"}'` -eq "1" ] ; do

    sleep 1

    result=`redis-cli -p $1 -a $2 info Persistence | grep "rdb\_bgsave\_in\_progress" | awk -F":" '{print $2}'`

done

edate=$(date "+ %Y-%m-%d %H:%M:%S")

echo "end bgsave time:$edate"

echo "renaming rdb file..."

date=$(date "+ %Y%m%d")

mv "../db03/dump.rdb" "../db03/${date}.rdb"

echo "finish backup rdb file"

echo "end"

### 5.6 MySQL5.6安装配置

### 5.7 Memcached安装配置

### 5.8 OpenLDAP安装配置

### 5.9 Nginx安装配置

### 5.10 PHP安装配置

### 5.11 NFS安装配置

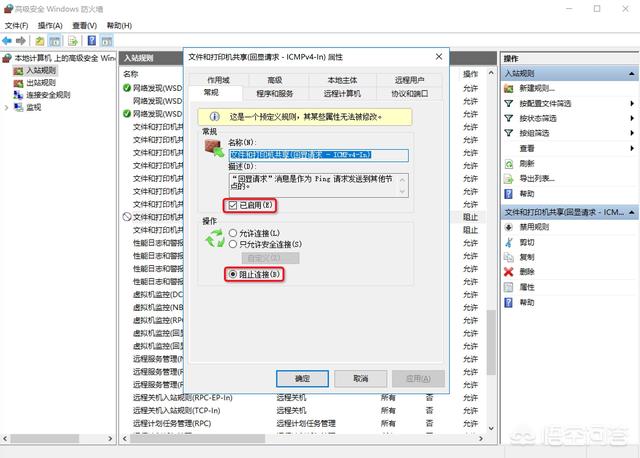
## 6.Windows server 2008/2012/2016

6.1操作系统配置

禁用服务器ICMP回显响应

控制面板 》查看方式“大图标”》Windows 防火墙 》左侧“高级设置”》入站规则 》找到“文件和打印机共享(回显请求-ICMPv4-In)”和“文件和打印机共享(回显请求-ICMPv6-In)”双击，然后选中“已启用”和“阻止连接”，如下图示：





#### 6.3.X Tomcat安装配置

#### 6.4.X Tomcat编译安装配置

# Kubernetes命令

kubectl get cs 查看组件状态信息

kubectl get ns 查看命名空间名称

kubectl describe node node01 查看node01的详细信息

kubectl get node 查看node状态

kubectl get pod -n命名空间 查看服务的状态

kubectl exec –it [option] –n命名空间 bash 以bash进入服务里面

kubectl get deploy -n命名空间 查看deploy类型的命名服务

kubectl edit deploy [option] -n命名空间 eg:kubectl edit deploy mysql -nketan 编辑mysql配置文件

kubectl delete pod PODNAME -n NAMESPACE --grace-period=0 –force

强制删除一个pod

kubectl get deployment -n jenkins ：查询 jenkins 命名空间下的 deployment

kubectl delete deployment jenkins2 -n jenkins  ：删除 jenkins 命名空间下的 jenkins2 的 deployment ，删除deployment 之后 pod 自动删除

kubectl delete ns 命名空间的名字

# 或者使用

kubectl delete ns 命名空间的名字 --force --grace-period=0

kubectl scale --replicas=0 sts infoplus –nketanyun将infoplus pod变为0

kubectl get deployment -w (-w表示监控)

kubectl get pods -l app 只看标签为app的pod

kubectl label pods pod-demo release=canary 修改标签

kubectl label pods pod-demo release=stable --overwrite 修改新标签并覆盖原有标签

kubeadm config images list 查看镜像列表

kubectl cp infoplus-v2.0-2019083.sql ketanyun/mysql-669bb87597-cxn45:/opt/.

将sql文件拷贝到mysql中

kubectl cp podname:logs  ./ -n ketanyun

将pod内logs文件夹拷贝到 ./

kubectl exec -it mysql-669bb87597-cxn45 -n ketanyun bash

进入到mysql中

kubectl scale --replicas=2 deploy taskcenter -n ketanyun # --replicas=2表示将pod节点扩为两个

kubectl get ingress -n ketanyun 查看ingress

kubectl create secret tls tls-hrbeu --cert=./214932563040839.pem --key=./214932563040839.key --namespace=ketanyun 添加集群证书

kubectl run 创建并运行一个特定的映像，可能是复制的 eg：kubectl run nginx --image=nginx控制器的名称

#### 查看所有namespace的pods运行情况

kubectl get pods --all-namespaces

#### 查看具体pods，记得后边跟namespace名字哦

kubectl get pods kubernetes-dashboard-76479d66bb-nj8wr --namespace=kube-system

#### 查看pods具体信息

kubectl get pods -o wide kubernetes-dashboard-76479d66bb-nj8wr --namespace=kube-system

#### 查看集群健康状态

kubectl get cs

#### 获取所有deployment

kubectl get deployment --all-namespaces

#### 列出该 namespace 中的所有 pod 包括未初始化的

kubectl get pods --include-uninitialized

#### 查看deployment()

kubectl get deployment nginx-app

#### 查看rc和servers

kubectl get rc,services

#### 查看pods结构信息（重点，通过这个看日志分析错误）对控制器和服务，node同样有效

kubectl describe pods xxxxpodsname --namespace=xxxnamespace

#### 其他控制器类似吧，就是kubectl get 控制器 控制器具体名称

#### 查看pod日志

kubectl logs $POD\_NAME

#### 查看pod变量

kubectl exec my-nginx-5j8ok -- printenv | grep SERVICE

kubectl get cs #### 集群健康情况  
kubectl cluster-info #### 集群核心组件运行情况  
kubectl get namespaces #### 表空间名  
kubectl version #### 版本  
kubectl api-versions #### API  
kubectl get events #### 查看事件  
kubectl get nodes //获取全部节点  
kubectl delete node k8s2 //删除节点  
kubectl rollout status deploy nginx-test

#### 创建

kubectl create -f ./nginx.yaml #### 创建资源  
kubectl create -f . #### 创建当前目录下的所有yaml资源  
kubectl create -f ./nginx1.yaml -f ./mysql2.yaml #### 使用多个文件创建资源  
kubectl create -f ./dir #### 使用目录下的所有清单文件来创建资源  
kubectl create -f https://git.io/vPieo #### 使用 url 来创建资源  
kubectl run -i --tty busybox --image=busybox ----创建带有终端的pod  
kubectl run nginx --image=nginx #### 启动一个 nginx 实例  
kubectl run mybusybox --image=busybox --replicas=5 ----启动多个pod  
kubectl explain pods,svc #### 获取 pod 和 svc 的文档

#### 更新

kubectl rolling-update python-v1 -f python-v2.json #### 滚动更新 pod frontend-v1  
kubectl rolling-update python-v1 python-v2 --image=image:v2 #### 更新资源名称并更新镜像  
kubectl rolling-update python --image=image:v2 #### 更新 frontend pod 中的镜像  
kubectl rolling-update python-v1 python-v2 --rollback #### 退出已存在的进行中的滚动更新  
cat pod.json | kubectl replace -f - #### 基于 stdin 输入的 JSON 替换 pod  
强制替换，删除后重新创建资源。会导致服务中断。  
kubectl replace --force -f ./pod.json  
为 nginx RC 创建服务，启用本地 80 端口连接到容器上的 8000 端口  
kubectl expose rc nginx --port=80 --target-port=8000

更新单容器 pod 的镜像版本（tag）到 v4  
kubectl get pod nginx-pod -o yaml | sed 's/(image: myimage):.\*$/\1:v4/' | kubectl replace -f -  
kubectl label pods nginx-pod new-label=awesome #### 添加标签  
kubectl annotate pods nginx-pod icon-url=http://goo.gl/XXBTWq #### 添加注解  
kubectl autoscale deployment foo --min=2 --max=10 #### 自动扩展 deployment “foo”

#### 编辑资源

kubectl edit svc/[**docker**](https://debian.cn/tag/docker/)-registry #### 编辑名为 docker-registry 的 service  
KUBE\_EDITOR="nano" kubectl edit svc/docker-registry #### 使用其它编辑器

#### 动态伸缩pod

kubectl scale --replicas=3 rs/foo #### 将foo副本集变成3个  
kubectl scale --replicas=3 -f foo.yaml #### 缩放“foo”中指定的资源。  
kubectl scale --current-replicas=2 --replicas=3 deployment/mysql #### 将deployment/mysql从2个变成3个  
kubectl scale --replicas=5 rc/foo rc/bar rc/baz #### 变更多个控制器的数量  
kubectl rollout status deploy deployment/mysql #### 查看变更进度

#### 删除

kubectl delete -f ./pod.json #### 删除 pod.json 文件中定义的类型和名称的 pod  
kubectl delete pod,service baz foo #### 删除名为“baz”的 pod 和名为“foo”的 service  
kubectl delete pods,services -l name=myLabel #### 删除具有 name=myLabel 标签的 pod 和 serivce  
kubectl delete pods,services -l name=myLabel --include-uninitialized #### 删除具有 name=myLabel 标签的 pod 和 service，包括尚未初始化的  
kubectl -n my-ns delete po,svc --all #### 删除 my-ns namespace下的所有 pod 和 serivce，包括尚未初始化的  
kubectl delete pods prometheus-7fcfcb9f89-qkkf7 --grace-period=0 --force 强制删除

#### 交互

kubectl logs nginx-pod #### dump 输出 pod 的日志（stdout）  
kubectl logs nginx-pod -c my-container #### dump 输出 pod 中容器的日志（stdout，pod 中有多个容器的情况下使用）  
kubectl logs -f nginx-pod #### 流式输出 pod 的日志（stdout）  
kubectl logs -f nginx-pod -c my-container #### 流式输出 pod 中容器的日志（stdout，pod 中有多个容器的情况下使用）  
kubectl run -i --tty busybox --image=busybox -- sh #### 交互式 shell 的方式运行 pod  
kubectl attach nginx-pod -i #### 连接到运行中的容器  
kubectl port-forward nginx-pod 5000:6000 #### 转发 pod 中的 6000 端口到本地的 5000 端口  
kubectl exec nginx-pod -- ls / #### 在已存在的容器中执行命令（只有一个容器的情况下）  
kubectl exec nginx-pod -c my-container -- ls / #### 在已存在的容器中执行命令（pod 中有多个容器的情况下）  
kubectl top pod POD\_NAME --containers #### 显示指定 pod和容器的指标度量

#### 调度配置

kubectl cordon k8s-node #### 标记 my-node 不可调度  
kubectl drain k8s-node #### 清空 my-node 以待维护  
kubectl uncordon k8s-node #### 标记 my-node 可调度  
kubectl top node k8s-node #### 显示 my-node 的指标度量  
kubectl cluster-info dump #### 将当前集群状态输出到 stdout  
kubectl cluster-info dump --output-directory=/path/to/cluster-state #### 将当前集群状态输出到 /path/to/cluster-state

#### 如果该键和影响的污点（taint）已存在，则使用指定的值替换

kubectl taint nodes foo dedicated=special-user:NoSchedule

# docker命令

## **docker run ：**创建一个新的容器并运行一个命令

### 语法

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

OPTIONS说明：

* **-a stdin:** 指定标准输入输出内容类型，可选 STDIN/STDOUT/STDERR 三项；
* **-d:** 后台运行容器，并返回容器ID；
* **-i:** 以交互模式运行容器，通常与 -t 同时使用；
* **-P:** 随机端口映射，容器内部端口**随机**映射到主机的端口
* **-p:** 指定端口映射，格式为：主机(宿主)端口:容器端口
* **-t:** 为容器重新分配一个伪输入终端，通常与 -i 同时使用；
* **--name="nginx-lb":** 为容器指定一个名称；
* **--dns 8.8.8.8:** 指定容器使用的DNS服务器，默认和宿主一致；
* **--dns-search example.com:** 指定容器DNS搜索域名，默认和宿主一致；
* **-h "mars":** 指定容器的hostname；
* **-e username="ritchie":** 设置环境变量；
* **--env-file=[]:** 从指定文件读入环境变量；
* **--cpuset="0-2" or --cpuset="0,1,2":** 绑定容器到指定CPU运行；
* **-m :**设置容器使用内存最大值；
* **--net="bridge":** 指定容器的网络连接类型，支持 bridge/host/none/container: 四种类型；
* **--link=[]:** 添加链接到另一个容器；
* **--expose=[]:** 开放一个端口或一组端口；
* **--volume , -v:**绑定一个卷

### 实例

使用docker镜像nginx:latest以后台模式启动一个容器,并将容器命名为mynginx。

docker run --name mynginx -d nginx:latest

使用镜像nginx:latest以后台模式启动一个容器,并将容器的80端口映射到主机随机端口。

docker run -P -d nginx:latest

使用镜像 nginx:latest，以后台模式启动一个容器,将容器的 80 端口映射到主机的 80 端口,主机的目录 /data 映射到容器的 /data。

docker run -p 80:80 -v /data:/data -d nginx:latest

绑定容器的 8080 端口，并将其映射到本地主机 127.0.0.1 的 80 端口上。

$ docker run -p 127.0.0.1:80:8080/tcp ubuntu bash

使用镜像nginx:latest以交互模式启动一个容器,在容器内执行/bin/bash命令。

runoob@runoob:~$ docker run -it nginx:latest /bin/bash

root@b8573233d675:/#

## Docker start/stop/restart 命令

**docker start** :启动一个或多个已经被停止的容器

**docker stop** :停止一个运行中的容器

**docker restart** :重启容器

### 语法

docker start [OPTIONS] CONTAINER [CONTAINER...]

docker stop [OPTIONS] CONTAINER [CONTAINER...]

docker restart [OPTIONS] CONTAINER [CONTAINER...]

### 实例

启动已被停止的容器myrunoob

docker start myrunoob

停止运行中的容器myrunoob

docker stop myrunoob

重启容器myrunoob

docker restart myrunoob

## Docker kill 命令

**docker kill** :杀掉一个运行中的容器。

### 语法

docker kill [OPTIONS] CONTAINER [CONTAINER...]

OPTIONS说明：

* **-s :**向容器发送一个信号

### 实例

杀掉运行中的容器mynginx

runoob@runoob:~$ docker kill -s KILL mynginx

mynginx

## Docker rm 命令

**docker rm ：**删除一个或多个容器。

#### 语法

docker rm [OPTIONS] CONTAINER [CONTAINER...]

OPTIONS说明：

* **-f :**通过 SIGKILL 信号强制删除一个运行中的容器。
* **-l :**移除容器间的网络连接，而非容器本身。
* **-v :**删除与容器关联的卷。

#### 实例

强制删除容器 db01、db02：

docker rm -f db01 db02

移除容器 nginx01 对容器 db01 的连接，连接名 db：

docker rm -l db

删除容器 nginx01, 并删除容器挂载的数据卷：

docker rm -v nginx01

删除所有已经停止的容器：

docker rm $(docker ps -a -q)

## Docker pause/unpause 命令

**docker pause** :暂停容器中所有的进程。

**docker unpause** :恢复容器中所有的进程。

### 语法

docker pause CONTAINER [CONTAINER...]

docker unpause CONTAINER [CONTAINER...]

### 实例

暂停数据库容器db01提供服务。

docker pause db01

恢复数据库容器 db01 提供服务。

docker unpause db01

## Docker create 命令

**docker create ：**创建一个新的容器但不启动它

用法同 [docker run](https://m.runoob.com/docker/docker-run-command.html)

### 语法

docker create [OPTIONS] IMAGE [COMMAND] [ARG...]

语法同 [docker run](https://m.runoob.com/docker/docker-run-command.html)

### 实例

使用docker镜像nginx:latest创建一个容器,并将容器命名为myrunoob

runoob@runoob:~$ docker create --name myrunoob nginx:latest

09b93464c2f75b7b69f83d56a9cfc23ceb50a48a9db7652ee4c27e3e2cb1961f

## Docker exec 命令

**docker exec ：**在运行的容器中执行命令

### 语法

docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

OPTIONS说明：

* **-d :**分离模式: 在后台运行
* **-i :**即使没有附加也保持STDIN 打开
* **-t :**分配一个伪终端

### 实例

在容器 mynginx 中以交互模式执行容器内 /root/runoob.sh 脚本:

runoob@runoob:~$ docker exec -it mynginx /bin/sh /root/runoob.sh

http://www.runoob.com/

在容器 mynginx 中开启一个交互模式的终端:

runoob@runoob:~$ docker exec -i -t mynginx /bin/bash

root@b1a0703e41e7:/#

也可以通过 docker ps -a 命令查看已经在运行的容器，然后使用容器 ID 进入容器。

查看已经在运行的容器 ID：

# docker ps -a

...

9df70f9a0714 openjdk "/usercode/script.sh…"

...

第一列的 9df70f9a0714 就是容器 ID。

通过 exec 命令对指定的容器执行 bash:

# docker exec -it 9df70f9a0714 /bin/bash

## Docker ps 命令

**docker ps :**列出容器

### 语法

docker ps [OPTIONS]

OPTIONS说明：

* **-a :**显示所有的容器，包括未运行的。
* **-f :**根据条件过滤显示的内容。
* **--format :**指定返回值的模板文件。
* **-l :**显示最近创建的容器。
* **-n :**列出最近创建的n个容器。
* **--no-trunc :**不截断输出。
* **-q :**静默模式，只显示容器编号。
* **-s :**显示总的文件大小。

### 实例

列出所有在运行的容器信息。

runoob@runoob:~$ docker ps

CONTAINER ID IMAGE COMMAND ... PORTS NAMES

09b93464c2f7 nginx:latest "nginx -g 'daemon off" ... 80/tcp, 443/tcp myrunoob

96f7f14e99ab mysql:5.6 "docker-entrypoint.sh" ... 0.0.0.0:3306->3306/tcp mymysql

输出详情介绍：

**CONTAINER ID:** 容器 ID。

**IMAGE:** 使用的镜像。

**COMMAND:** 启动容器时运行的命令。

**CREATED:** 容器的创建时间。

**STATUS:** 容器状态。

状态有7种：

* created（已创建）
* restarting（重启中）
* running（运行中）
* removing（迁移中）
* paused（暂停）
* exited（停止）
* dead（死亡）

**PORTS:** 容器的端口信息和使用的连接类型（tcp\udp）。

**NAMES:** 自动分配的容器名称。

列出最近创建的5个容器信息。

runoob@runoob:~$ docker ps -n 5

CONTAINER ID IMAGE COMMAND CREATED

09b93464c2f7 nginx:latest "nginx -g 'daemon off" 2 days ago ...

b8573233d675 nginx:latest "/bin/bash" 2 days ago ...

b1a0703e41e7 nginx:latest "nginx -g 'daemon off" 2 days ago ...

f46fb1dec520 5c6e1090e771 "/bin/sh -c 'set -x \t" 2 days ago ...

a63b4a5597de 860c279d2fec "bash" 2 days ago ...

列出所有创建的容器ID。

runoob@runoob:~$ docker ps -a -q

09b93464c2f7

b8573233d675

b1a0703e41e7

f46fb1dec520

a63b4a5597de

6a4aa42e947b

de7bb36e7968

43a432b73776

664a8ab1a585

ba52eb632bbd

...

## Docker inspect 命令

**docker inspect :** 获取容器/镜像的元数据。

### 语法

docker inspect [OPTIONS] NAME|ID [NAME|ID...]

OPTIONS说明：

* **-f :**指定返回值的模板文件。
* **-s :**显示总的文件大小。
* **--type :**为指定类型返回JSON。

### 实例

获取镜像mysql:5.6的元信息。

runoob@runoob:~$ docker inspect mysql:5.6

[

{

"Id": "sha256:2c0964ec182ae9a045f866bbc2553087f6e42bfc16074a74fb820af235f070ec",

"RepoTags": [

"mysql:5.6"

],

"RepoDigests": [],

"Parent": "",

"Comment": "",

"Created": "2016-05-24T04:01:41.168371815Z",

"Container": "e0924bc460ff97787f34610115e9363e6363b30b8efa406e28eb495ab199ca54",

"ContainerConfig": {

"Hostname": "b0cf605c7757",

"Domainname": "",

"User": "",

"AttachStdin": false,

"AttachStdout": false,

"AttachStderr": false,

"ExposedPorts": {

"3306/tcp": {}

},

...

获取正在运行的容器mymysql的 IP。

runoob@runoob:~$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mymysql

172.17.0.3

## Docker top 命令

**docker top :**查看容器中运行的进程信息，支持 ps 命令参数。

### 语法

docker top [OPTIONS] CONTAINER [ps OPTIONS]

容器运行时不一定有/bin/bash终端来交互执行top命令，而且容器还不一定有top命令，可以使用docker top来实现查看container中正在运行的进程。

### 实例

查看容器mymysql的进程信息。

runoob@runoob:~/mysql$ docker top mymysql

UID PID PPID C STIME TTY TIME CMD

999 40347 40331 18 00:58 ? 00:00:02 mysqld

查看所有运行容器的进程信息。

for i in `docker ps |grep Up|awk '{print $1}'`;do echo \ &&docker top $i; done

## Docker attach 命令

**docker attach :**连接到正在运行中的容器。

### 语法

docker attach [OPTIONS] CONTAINER

要attach上去的容器必须正在运行，可以同时连接上同一个container来共享屏幕（与screen命令的attach类似）。

官方文档中说attach后可以通过CTRL-C来detach，但实际上经过我的测试，如果container当前在运行bash，CTRL-C自然是当前行的输入，没有退出；如果container当前正在前台运行进程，如输出nginx的access.log日志，CTRL-C不仅会导致退出容器，而且还stop了。这不是我们想要的，detach的意思按理应该是脱离容器终端，但容器依然运行。好在attach是可以带上--sig-proxy=false来确保CTRL-D或CTRL-C不会关闭容器。

### 实例

容器mynginx将访问日志指到标准输出，连接到容器查看访问信息。

runoob@runoob:~$ docker attach --sig-proxy=false mynginx

192.168.239.1 - - [10/Jul/2016:16:54:26 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.93 Safari/537.36" "-"

## Docker events 命令

**docker events :**从服务器获取实时事件

### 语法

docker events [OPTIONS]

OPTIONS说明：

* **-f ：**根据条件过滤事件；
* **--since ：**从指定的时间戳后显示所有事件;
* **--until ：**流水时间显示到指定的时间为止；

### 实例

显示docker 2016年7月1日后的所有事件。

runoob@runoob:~/mysql$ docker events --since="1467302400"

2016-07-08T19:44:54.501277677+08:00 network connect 66f958fd13dc4314ad20034e576d5c5eba72e0849dcc38ad9e8436314a4149d4 (container=b8573233d675705df8c89796a2c2687cd8e36e03646457a15fb51022db440e64, name=bridge, type=bridge)

2016-07-08T19:44:54.723876221+08:00 container start b8573233d675705df8c89796a2c2687cd8e36e03646457a15fb51022db440e64 (image=nginx:latest, name=elegant\_albattani)

2016-07-08T19:44:54.726110498+08:00 container resize b8573233d675705df8c89796a2c2687cd8e36e03646457a15fb51022db440e64 (height=39, image=nginx:latest, name=elegant\_albattani, width=167)

2016-07-08T19:46:22.137250899+08:00 container die b8573233d675705df8c89796a2c2687cd8e36e03646457a15fb51022db440e64 (exitCode=0, image=nginx:latest, name=elegant\_albattani)

...

显示docker 镜像为mysql:5.6 2016年7月1日后的相关事件。

runoob@runoob:~/mysql$ docker events -f "image"="mysql:5.6" --since="1467302400"

2016-07-11T00:38:53.975174837+08:00 container start 96f7f14e99ab9d2f60943a50be23035eda1623782cc5f930411bbea407a2bb10 (image=mysql:5.6, name=mymysql)

2016-07-11T00:51:17.022572452+08:00 container kill 96f7f14e99ab9d2f60943a50be23035eda1623782cc5f930411bbea407a2bb10 (image=mysql:5.6, name=mymysql, signal=9)

2016-07-11T00:51:17.132532080+08:00 container die 96f7f14e99ab9d2f60943a50be23035eda1623782cc5f930411bbea407a2bb10 (exitCode=137, image=mysql:5.6, name=mymysql)

2016-07-11T00:51:17.514661357+08:00 container destroy 96f7f14e99ab9d2f60943a50be23035eda1623782cc5f930411bbea407a2bb10 (image=mysql:5.6, name=mymysql)

2016-07-11T00:57:18.551984549+08:00 container create c8f0a32f12f5ec061d286af0b1285601a3e33a90a08ff1706de619ac823c345c (image=mysql:5.6, name=mymysql)

2016-07-11T00:57:18.557405864+08:00 container attach c8f0a32f12f5ec061d286af0b1285601a3e33a90a08ff1706de619ac823c345c (image=mysql:5.6, name=mymysql)

2016-07-11T00:57:18.844134112+08:00 container start c8f0a32f12f5ec061d286af0b1285601a3e33a90a08ff1706de619ac823c345c (image=mysql:5.6, name=mymysql)

2016-07-11T00:57:19.140141428+08:00 container die c8f0a32f12f5ec061d286af0b1285601a3e33a90a08ff1706de619ac823c345c (exitCode=1, image=mysql:5.6, name=mymysql)

2016-07-11T00:58:05.941019136+08:00 container destroy c8f0a32f12f5ec061d286af0b1285601a3e33a90a08ff1706de619ac823c345c (image=mysql:5.6, name=mymysql)

2016-07-11T00:58:07.965128417+08:00 container create a404c6c174a21c52f199cfce476e041074ab020453c7df2a13a7869b48f2f37e (image=mysql:5.6, name=mymysql)

2016-07-11T00:58:08.188734598+08:00 container start a404c6c174a21c52f199cfce476e041074ab020453c7df2a13a7869b48f2f37e (image=mysql:5.6, name=mymysql)

2016-07-11T00:58:20.010876777+08:00 container top a404c6c174a21c52f199cfce476e041074ab020453c7df2a13a7869b48f2f37e (image=mysql:5.6, name=mymysql)

2016-07-11T01:06:01.395365098+08:00 container top a404c6c174a21c52f199cfce476e041074ab020453c7df2a13a7869b48f2f37e (image=mysql:5.6, name=mymysql)

如果指定的时间是到秒级的，需要将时间转成时间戳。如果时间为日期的话，可以直接使用，如--since="2016-07-01"。

## Docker logs 命令

**docker logs :**获取容器的日志

### 语法

docker logs [OPTIONS] CONTAINER

OPTIONS说明：

* **-f :**跟踪日志输出
* **--since :**显示某个开始时间的所有日志
* **-t :**显示时间戳
* **--tail :**仅列出最新N条容器日志

### 实例

跟踪查看容器mynginx的日志输出。

runoob@runoob:~$ docker logs -f mynginx

192.168.239.1 - - [10/Jul/2016:16:53:33 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.93 Safari/537.36" "-"

2016/07/10 16:53:33 [error] 5#5: \*1 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 192.168.239.1, server: localhost, request: "GET /favicon.ico HTTP/1.1", host: "192.168.239.130", referrer: "http://192.168.239.130/"

192.168.239.1 - - [10/Jul/2016:16:53:33 +0000] "GET /favicon.ico HTTP/1.1" 404 571 "http://192.168.239.130/" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.93 Safari/537.36" "-"

192.168.239.1 - - [10/Jul/2016:16:53:59 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.93 Safari/537.36" "-"

...

查看容器mynginx从2016年7月1日后的最新10条日志。

docker logs --since="2016-07-01" --tail=10 mynginx

## Docker wait 命令

**docker wait :** 阻塞运行直到容器停止，然后打印出它的退出代码。

### 语法

docker wait [OPTIONS] CONTAINER [CONTAINER...]

### 实例

docker wait CONTAINER

## Docker export 命令

**docker export :**将文件系统作为一个tar归档文件导出到STDOUT。

### 语法

docker export [OPTIONS] CONTAINER

OPTIONS说明：

* **-o :**将输入内容写到文件。

### 实例

将id为a404c6c174a2的容器按日期保存为tar文件。

runoob@runoob:~$ docker export -o mysql-`date +%Y%m%d`.tar a404c6c174a2

runoob@runoob:~$ ls mysql-`date +%Y%m%d`.tar

mysql-20160711.tar

## Docker port 命令

**docker port :**列出指定的容器的端口映射，或者查找将PRIVATE\_PORT NAT到面向公众的端口。

### 语法

docker port [OPTIONS] CONTAINER [PRIVATE\_PORT[/PROTO]]

### 实例

查看容器mynginx的端口映射情况。

runoob@runoob:~$ docker port mymysql

3306/tcp -> 0.0.0.0:3306

## Docker commit 命令

**docker commit :**从容器创建一个新的镜像。

### 语法

docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

OPTIONS说明：

* **-a :**提交的镜像作者；
* **-c :**使用Dockerfile指令来创建镜像；
* **-m :**提交时的说明文字；
* **-p :**在commit时，将容器暂停。

### 实例

将容器a404c6c174a2 保存为新的镜像,并添加提交人信息和说明信息。

runoob@runoob:~$ docker commit -a "runoob.com" -m "my apache" a404c6c174a2 mymysql:v1

sha256:37af1236adef1544e8886be23010b66577647a40bc02c0885a6600b33ee28057

runoob@runoob:~$ docker images mymysql:v1

REPOSITORY TAG IMAGE ID CREATED SIZE

mymysql v1 37af1236adef 15 seconds ago 329 MB

## Docker cp 命令

**docker cp :**用于容器与主机之间的数据拷贝。

#### 语法

docker cp [OPTIONS] CONTAINER:SRC\_PATH DEST\_PATH|-

docker cp [OPTIONS] SRC\_PATH|- CONTAINER:DEST\_PATH

OPTIONS说明：

* **-L :**保持源目标中的链接

#### 实例

将主机/www/runoob目录拷贝到容器96f7f14e99ab的/www目录下。

docker cp /www/runoob 96f7f14e99ab:/www/

将主机/www/runoob目录拷贝到容器96f7f14e99ab中，目录重命名为www。

docker cp /www/runoob 96f7f14e99ab:/www

将容器96f7f14e99ab的/www目录拷贝到主机的/tmp目录中。

docker cp 96f7f14e99ab:/www /tmp/

## Docker diff 命令

**docker diff :**检查容器里文件结构的更改。

### 语法

docker diff [OPTIONS] CONTAINER

### 实例

查看容器mymysql的文件结构更改。

runoob@runoob:~$ docker diff mymysql

A /logs

A /mysql\_data

C /run

C /run/mysqld

A /run/mysqld/mysqld.pid

A /run/mysqld/mysqld.sock

C /tmp

## Docker login/logout 命令

**docker login :** 登陆到一个Docker镜像仓库，如果未指定镜像仓库地址，默认为官方仓库 Docker Hub

**docker logout :** 登出一个Docker镜像仓库，如果未指定镜像仓库地址，默认为官方仓库 Docker Hub

### 语法

docker login [OPTIONS] [SERVER]

docker logout [OPTIONS] [SERVER]

OPTIONS说明：

* **-u :**登陆的用户名
* **-p :**登陆的密码

### 实例

登陆到Docker Hub

docker login -u 用户名 -p 密码

登出Docker Hub

docker logout

## Docker pull 命令

**docker pull :**从镜像仓库中拉取或者更新指定镜像

### 语法

docker pull [OPTIONS] NAME[:TAG|@DIGEST]

OPTIONS说明：

* **-a :**拉取所有 tagged 镜像
* **--disable-content-trust :**忽略镜像的校验,默认开启

### 实例

从Docker Hub下载java最新版镜像。

docker pull java

从Docker Hub下载REPOSITORY为java的所有镜像。

docker pull -a java

## Docker push 命令

**docker push :**将本地的镜像上传到镜像仓库,要先登陆到镜像仓库

### 语法

docker push [OPTIONS] NAME[:TAG]

OPTIONS说明：

* **--disable-content-trust :**忽略镜像的校验,默认开启

### 实例

上传本地镜像myapache:v1到镜像仓库中。

docker push myapache:v1

## Docker search 命令

**docker search :** 从Docker Hub查找镜像

### 语法

docker search [OPTIONS] TERM

OPTIONS说明：

* **--automated :**只列出 automated build类型的镜像；
* **--no-trunc :**显示完整的镜像描述；
* **-f <过滤条件>:**列出收藏数不小于指定值的镜像。

### 实例

从 Docker Hub 查找所有镜像名包含 java，并且收藏数大于 10 的镜像

runoob@runoob:~$ docker search -f stars=10 java

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

java Java is a concurrent, class-based... 1037 [OK]

anapsix/alpine-java Oracle Java 8 (and 7) with GLIBC ... 115 [OK]

develar/java 46 [OK]

isuper/java-oracle This repository contains all java... 38 [OK]

lwieske/java-8 Oracle Java 8 Container - Full + ... 27 [OK]

nimmis/java-centos This is docker images of CentOS 7... 13 [OK]

参数说明：

**NAME:**镜像仓库源的名称

**DESCRIPTION:**镜像的描述

**OFFICIAL:**是否 docker 官方发布

**stars:**类似 Github 里面的 star，表示点赞、喜欢的意思。

**AUTOMATED:**自动构建。

## Docker images 命令

**docker images :**列出本地镜像。

### 语法

docker images [OPTIONS] [REPOSITORY[:TAG]]

OPTIONS说明：

* **-a :**列出本地所有的镜像（含中间映像层，默认情况下，过滤掉中间映像层）；
* **--digests :**显示镜像的摘要信息；
* **-f :**显示满足条件的镜像；
* **--format :**指定返回值的模板文件；
* **--no-trunc :**显示完整的镜像信息；
* **-q :**只显示镜像ID。

### 实例

查看本地镜像列表。

runoob@runoob:~$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

mymysql v1 37af1236adef 5 minutes ago 329 MB

runoob/ubuntu v4 1c06aa18edee 2 days ago 142.1 MB

<none> <none> 5c6e1090e771 2 days ago 165.9 MB

httpd latest ed38aaffef30 11 days ago 195.1 MB

alpine latest 4e38e38c8ce0 2 weeks ago 4.799 MB

mongo 3.2 282fd552add6 3 weeks ago 336.1 MB

redis latest 4465e4bcad80 3 weeks ago 185.7 MB

php 5.6-fpm 025041cd3aa5 3 weeks ago 456.3 MB

python 3.5 045767ddf24a 3 weeks ago 684.1 MB

...

列出本地镜像中REPOSITORY为ubuntu的镜像列表。

root@runoob:~# docker images ubuntu

REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu 14.04 90d5884b1ee0 9 weeks ago 188 MB

ubuntu 15.10 4e3b13c8a266 3 months ago 136.3 MB

## Docker rmi 命令

**docker rmi :**删除本地一个或多少镜像。

### 语法

docker rmi [OPTIONS] IMAGE [IMAGE...]

OPTIONS说明：

* **-f :**强制删除；
* **--no-prune :**不移除该镜像的过程镜像，默认移除；

### 实例

强制删除本地镜像 runoob/ubuntu:v4。

root@runoob:~# docker rmi -f runoob/ubuntu:v4

Untagged: runoob/ubuntu:v4

Deleted: sha256:1c06aa18edee44230f93a90a7d88139235de12cd4c089d41eed8419b503072be

Deleted: sha256:85feb446e89a28d58ee7d80ea5ce367eebb7cec70f0ec18aa4faa874cbd97c73

## Docker tag 命令

**docker tag :**标记本地镜像，将其归入某一仓库。

### 语法

docker tag [OPTIONS] IMAGE[:TAG] [REGISTRYHOST/][USERNAME/]NAME[:TAG]

### 实例

将镜像ubuntu:15.10标记为 runoob/ubuntu:v3 镜像。

root@runoob:~# docker tag ubuntu:15.10 runoob/ubuntu:v3

root@runoob:~# docker images runoob/ubuntu:v3

REPOSITORY TAG IMAGE ID CREATED SIZE

runoob/ubuntu v3 4e3b13c8a266 3 months ago 136.3 MB

## Docker build 命令

**docker build**命令用于使用 Dockerfile 创建镜像。

### 语法

docker build [OPTIONS] PATH | URL | -

OPTIONS说明：

* **--build-arg=[] :**设置镜像创建时的变量；
* **--cpu-shares :**设置 cpu 使用权重；
* **--cpu-period :**限制 CPU CFS周期；
* **--cpu-quota :**限制 CPU CFS配额；
* **--cpuset-cpus :**指定使用的CPU id；
* **--cpuset-mems :**指定使用的内存 id；
* **--disable-content-trust :**忽略校验，默认开启；
* **-f :**指定要使用的Dockerfile路径；
* **--force-rm :**设置镜像过程中删除中间容器；
* **--isolation :**使用容器隔离技术；
* **--label=[] :**设置镜像使用的元数据；
* **-m :**设置内存最大值；
* **--memory-swap :**设置Swap的最大值为内存+swap，"-1"表示不限swap；
* **--no-cache :**创建镜像的过程不使用缓存；
* **--pull :**尝试去更新镜像的新版本；
* **--quiet, -q :**安静模式，成功后只输出镜像 ID；
* **--rm :**设置镜像成功后删除中间容器；
* **--shm-size :**设置/dev/shm的大小，默认值是64M；
* **--ulimit :**Ulimit配置。
* **--squash :**将 Dockerfile 中所有的操作压缩为一层。
* **--tag, -t:** 镜像的名字及标签，通常 name:tag 或者 name 格式；可以在一次构建中为一个镜像设置多个标签。
* **--network:** 默认 default。在构建期间设置RUN指令的网络模式

### 实例

使用当前目录的 Dockerfile 创建镜像，标签为 runoob/ubuntu:v1。

docker build -t runoob/ubuntu:v1 .

使用URL **github.com/creack/docker-firefox** 的 Dockerfile 创建镜像。

docker build github.com/creack/docker-firefox

也可以通过 -f Dockerfile 文件的位置：

$ docker build -f /path/to/a/Dockerfile .

在 Docker 守护进程执行 Dockerfile 中的指令前，首先会对 Dockerfile 进行语法检查，有语法错误时会返回：

$ docker build -t test/myapp .

Sending build context to Docker daemon 2.048 kB

Error response from daemon: Unknown instruction: RUNCMD

## Docker history 命令

**docker history :**查看指定镜像的创建历史。

### 语法

docker history [OPTIONS] IMAGE

OPTIONS说明：

* **-H :**以可读的格式打印镜像大小和日期，默认为true；
* **--no-trunc :**显示完整的提交记录；
* **-q :**仅列出提交记录ID。

### 实例

查看本地镜像runoob/ubuntu:v3的创建历史。

root@runoob:~# docker history runoob/ubuntu:v3

IMAGE CREATED CREATED BY SIZE COMMENT

4e3b13c8a266 3 months ago /bin/sh -c #(nop) CMD ["/bin/bash"] 0 B

<missing> 3 months ago /bin/sh -c sed -i 's/^#\s\*\(deb.\*universe\)$/ 1.863 kB

<missing> 3 months ago /bin/sh -c set -xe && echo '#!/bin/sh' > /u 701 B

<missing> 3 months ago /bin/sh -c #(nop) ADD file:43cb048516c6b80f22 136.3 MB

## Docker save 命令

**docker save :**将指定镜像保存成 tar 归档文件。

### 语法

docker save [OPTIONS] IMAGE [IMAGE...]

OPTIONS 说明：

* **-o :**输出到的文件。

### 实例

将镜像 runoob/ubuntu:v3 生成 my\_ubuntu\_v3.tar 文档

runoob@runoob:~$ docker save -o my\_ubuntu\_v3.tar runoob/ubuntu:v3

runoob@runoob:~$ ll my\_ubuntu\_v3.tar

-rw------- 1 runoob runoob 142102016 Jul 11 01:37 my\_ubuntu\_v3.ta

## Docker load 命令

**docker load :**导入使用 [docker save](https://www.runoob.com/docker/docker-save-command.html) 命令导出的镜像。

### 语法

docker load [OPTIONS]

OPTIONS 说明：

* **--input , -i :**指定导入的文件，代替 STDIN。
* **--quiet , -q :**精简输出信息。

### 实例

导入镜像：

$ docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

$ docker load < busybox.tar.gz

Loaded image: busybox:latest

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox latest 769b9341d937 7 weeks ago 2.489 MB

$ docker load --input fedora.tar

Loaded image: fedora:rawhide

Loaded image: fedora:20

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox latest 769b9341d937 7 weeks ago 2.489 MB

fedora rawhide 0d20aec6529d 7 weeks ago 387 MB

fedora 20 58394af37342 7 weeks ago 385.5 MB

fedora heisenbug 58394af37342 7 weeks ago 385.5 MB

fedora latest 58394af37342 7 weeks ago 385.5 MB

## Docker import 命令

**docker import :** 从归档文件中创建镜像。

### 语法

docker import [OPTIONS] file|URL|- [REPOSITORY[:TAG]]

OPTIONS说明：

* **-c :**应用docker 指令创建镜像；
* **-m :**提交时的说明文字；

### 实例

从镜像归档文件my\_ubuntu\_v3.tar创建镜像，命名为runoob/ubuntu:v4

runoob@runoob:~$ docker import my\_ubuntu\_v3.tar runoob/ubuntu:v4

sha256:63ce4a6d6bc3fabb95dbd6c561404a309b7bdfc4e21c1d59fe9fe4299cbfea39

runoob@runoob:~$ docker images runoob/ubuntu:v4

REPOSITORY TAG IMAGE ID CREATED SIZE

runoob/ubuntu v4 63ce4a6d6bc3 20 seconds ago 142.1 MB

## Docker info 命令

docker info : 显示 Docker 系统信息，包括镜像和容器数。。

### 语法

docker info [OPTIONS]

### 实例

查看docker系统信息。

$ docker info

Containers: 12

Images: 41

Storage Driver: aufs

Root Dir: /var/lib/docker/aufs

Backing Filesystem: extfs

Dirs: 66

Dirperm1 Supported: false

Execution Driver: native-0.2

Logging Driver: json-file

Kernel Version: 3.13.0-32-generic

Operating System: Ubuntu 14.04.1 LTS

CPUs: 1

Total Memory: 1.954 GiB

Name: iZ23mtq8bs1Z

ID: M5N4:K6WN:PUNC:73ZN:AONJ:AUHL:KSYH:2JPI:CH3K:O4MK:6OCX:5OYW

## Docker version 命令

docker version :显示 Docker 版本信息。

### 语法

docker version [OPTIONS]

OPTIONS说明：

* **-f :**指定返回值的模板文件。

### 实例

显示 Docker 版本信息。

$ docker version

Client:

Version: 1.8.2

API version: 1.20

Go version: go1.4.2

Git commit: 0a8c2e3

Built: Thu Sep 10 19:19:00 UTC 2015

OS/Arch: linux/amd64

Server:

Version: 1.8.2

API version: 1.20

Go version: go1.4.2

Git commit: 0a8c2e3

Built: Thu Sep 10 19:19:00 UTC 2015

OS/Arch: linux/amd64

# Redis

## Redis 简介

Redis 是完全开源的，遵守 BSD 协议，是一个高性能的 key-value 数据库。

Redis 与其他 key - value 缓存产品有以下三个特点：

* Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
* Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。
* Redis支持数据的备份，即master-slave模式的数据备份。

## Redis 优势

* 性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。
* 丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。
* 原子 – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。
* 丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

## Redis与其他key-value存储有什么不同？

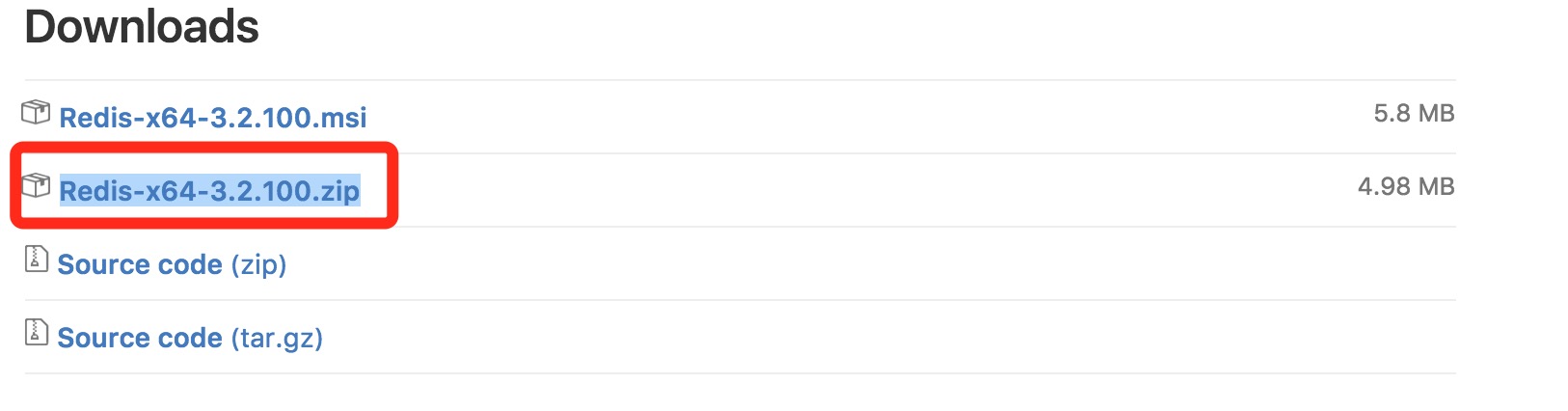
* Redis有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。
* Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

## Redis 安装

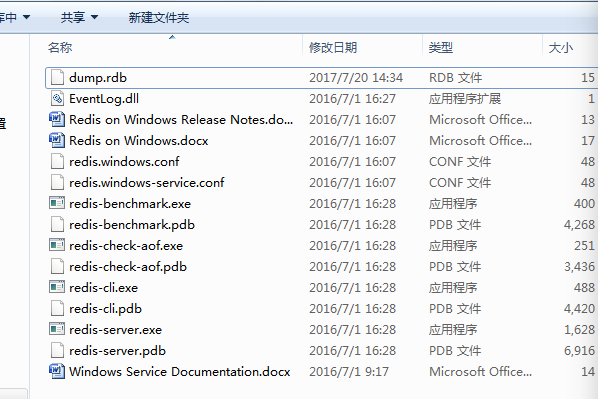
### Windows 下安装

**下载地址：**<https://github.com/tporadowski/redis/releases>。

Redis 支持 32 位和 64 位。这个需要根据你系统平台的实际情况选择，这里我们下载 **Redis-x64-xxx.zip**压缩包到 C 盘，解压后，将文件夹重新命名为 **redis**。



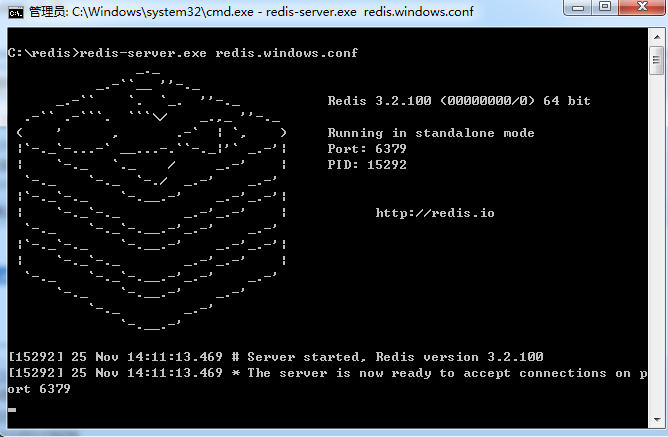
打开文件夹，内容如下：



打开一个 **cmd** 窗口 使用 cd 命令切换目录到 **C:\redis** 运行：

redis-server.exe redis.windows.conf

如果想方便的话，可以把 redis 的路径加到系统的环境变量里，这样就省得再输路径了，后面的那个 redis.windows.conf 可以省略，如果省略，会启用默认的。输入之后，会显示如下界面：



这时候另启一个 cmd 窗口，原来的不要关闭，不然就无法访问服务端了。

切换到 redis 目录下运行:

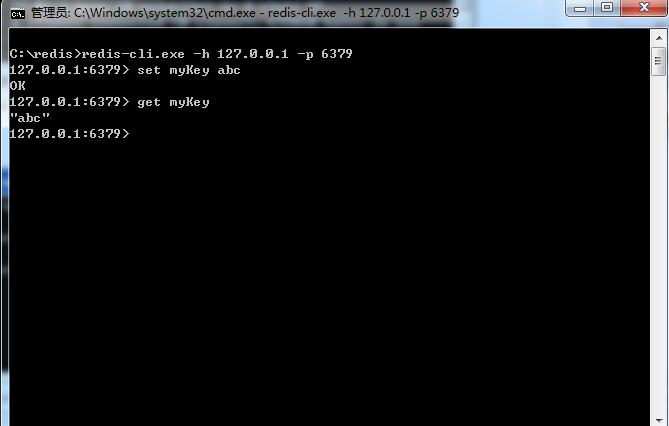
redis-cli.exe -h 127.0.0.1 -p 6379

设置键值对:

set myKey abc

取出键值对:

get myKey



### Linux 源码安装

**下载地址：**<http://redis.io/download>，下载最新稳定版本。

本教程使用的最新文档版本为 2.8.17，下载并安装：

# wget http://download.redis.io/releases/redis-6.0.8.tar.gz

# tar xzf redis-6.0.8.tar.gz

# cd redis-6.0.8

# make

执行完 **make** 命令后，redis-6.0.8 的 **src** 目录下会出现编译后的 redis 服务程序 redis-server，还有用于测试的客户端程序 redis-cli：

下面启动 redis 服务：

# cd src

# ./redis-server

注意这种方式启动 redis 使用的是默认配置。也可以通过启动参数告诉 redis 使用指定配置文件使用下面命令启动。

# cd src

# ./redis-server ../redis.conf

**redis.conf** 是一个默认的配置文件。我们可以根据需要使用自己的配置文件。

启动 redis 服务进程后，就可以使用测试客户端程序 redis-cli 和 redis 服务交互了。 比如：

# cd src

# ./redis-cli

redis> set foo bar

OK

redis> get foo

"bar"

### Ubuntu apt 命令安装

在 Ubuntu 系统安装 Redis 可以使用以下命令:

# sudo apt update

# sudo apt install redis-server

#### 启动 Redis

# redis-server

#### 查看 redis 是否启动？

# redis-cli

以上命令将打开以下终端：

redis 127.0.0.1:6379>

127.0.0.1 是本机 IP ，6379 是 redis 服务端口。现在我们输入 PING 命令。

redis 127.0.0.1:6379> ping

PONG

以上说明我们已经成功安装了redis。

## Redis 配置

Redis 的配置文件位于 Redis 安装目录下，文件名为 **redis.conf**(Windows 名为 redis.windows.conf)。

你可以通过 **CONFIG** 命令查看或设置配置项。

**语法**

**Redis CONFIG 命令格式如下：**

**redis 127.0.0.1:6379> CONFIG GET CONFIG\_SETTING\_NAME**

**实例**

redis 127.0.0.1:6379> CONFIG GET loglevel

1) "loglevel"

2) "notice"

使用 **\*** 号获取所有配置项：

**实例**

redis 127.0.0.1:6379> CONFIG GET \*

1) "dbfilename"

2) "dump.rdb"

3) "requirepass"

4) ""

5) "masterauth"

6) ""

7) "unixsocket"

8) ""

9) "logfile"

10) ""

11) "pidfile"

12) "/var/run/redis.pid"

13) "maxmemory"

14) "0"

15) "maxmemory-samples"

16) "3"

17) "timeout"

18) "0"

19) "tcp-keepalive"

20) "0"

21) "auto-aof-rewrite-percentage"

22) "100"

23) "auto-aof-rewrite-min-size"

24) "67108864"

25) "hash-max-ziplist-entries"

26) "512"

27) "hash-max-ziplist-value"

28) "64"

29) "list-max-ziplist-entries"

30) "512"

31) "list-max-ziplist-value"

32) "64"

33) "set-max-intset-entries"

34) "512"

35) "zset-max-ziplist-entries"

36) "128"

37) "zset-max-ziplist-value"

38) "64"

39) "hll-sparse-max-bytes"

40) "3000"

41) "lua-time-limit"

42) "5000"

43) "slowlog-log-slower-than"

44) "10000"

45) "latency-monitor-threshold"

46) "0"

47) "slowlog-max-len"

48) "128"

49) "port"

50) "6379"

51) "tcp-backlog"

52) "511"

53) "databases"

54) "16"

55) "repl-ping-slave-period"

56) "10"

57) "repl-timeout"

58) "60"

59) "repl-backlog-size"

60) "1048576"

61) "repl-backlog-ttl"

62) "3600"

63) "maxclients"

64) "4064"

65) "watchdog-period"

66) "0"

67) "slave-priority"

68) "100"

69) "min-slaves-to-write"

70) "0"

71) "min-slaves-max-lag"

72) "10"

73) "hz"

74) "10"

75) "no-appendfsync-on-rewrite"

76) "no"

77) "slave-serve-stale-data"

78) "yes"

79) "slave-read-only"

80) "yes"

81) "stop-writes-on-bgsave-error"

82) "yes"

83) "daemonize"

84) "no"

85) "rdbcompression"

86) "yes"

87) "rdbchecksum"

88) "yes"

89) "activerehashing"

90) "yes"

91) "repl-disable-tcp-nodelay"

92) "no"

93) "aof-rewrite-incremental-fsync"

94) "yes"

95) "appendonly"

96) "no"

97) "dir"

98) "/home/deepak/Downloads/redis-2.8.13/src"

99) "maxmemory-policy"

100) "volatile-lru"

101) "appendfsync"

102) "everysec"

103) "save"

104) "3600 1 300 100 60 10000"

105) "loglevel"

106) "notice"

107) "client-output-buffer-limit"

108) "normal 0 0 0 slave 268435456 67108864 60 pubsub 33554432 8388608 60"

109) "unixsocketperm"

110) "0"

111) "slaveof"

112) ""

113) "notify-keyspace-events"

114) ""

115) "bind"

116) ""

### 编辑配置

你可以通过修改 redis.conf 文件或使用 **CONFIG set** 命令来修改配置。

**语法**

**CONFIG SET** 命令基本语法：

redis 127.0.0.1:6379> CONFIG SET CONFIG\_SETTING\_NAME NEW\_CONFIG\_VALUE

**实例**

redis 127.0.0.1:6379> CONFIG SET loglevel "notice"

OK

redis 127.0.0.1:6379> CONFIG GET loglevel

1) "loglevel"

2) "notice"

### 参数说明

redis.conf 配置项说明如下：

|  |  |  |
| --- | --- | --- |
| **序号** | **配置项** | **说明** |
| 1 | daemonize no | Redis 默认不是以守护进程的方式运行，可以通过该配置项修改，使用 yes 启用守护进程（Windows 不支持守护线程的配置为 no ） |
| 2 | pidfile /var/run/redis.pid | 当 Redis 以守护进程方式运行时，Redis 默认会把 pid 写入 /var/run/redis.pid 文件，可以通过 pidfile 指定 |
| 3 | port 6379 | 指定 Redis 监听端口，默认端口为 6379，作者在自己的一篇博文中解释了为什么选用 6379 作为默认端口，因为 6379 在手机按键上 MERZ 对应的号码，而 MERZ 取自意大利歌女 Alessia Merz 的名字 |
| 4 | bind 127.0.0.1 | 绑定的主机地址 |
| 5 | timeout 300 | 当客户端闲置多长秒后关闭连接，如果指定为 0 ，表示关闭该功能 |
| 6 | loglevel notice | 指定日志记录级别，Redis 总共支持四个级别：debug、verbose、notice、warning，默认为 notice |
| 7 | logfile stdout | 日志记录方式，默认为标准输出，如果配置 Redis 为守护进程方式运行，而这里又配置为日志记录方式为标准输出，则日志将会发送给 /dev/null |
| 8 | databases 16 | 设置数据库的数量，默认数据库为0，可以使用SELECT 命令在连接上指定数据库id |
| 9 | save <seconds> <changes>  Redis 默认配置文件中提供了三个条件：  **save 900 1**  **save 300 10**  **save 60 10000**  分别表示 900 秒（15 分钟）内有 1 个更改，300 秒（5 分钟）内有 10 个更改以及 60 秒内有 10000 个更改。 | 指定在多长时间内，有多少次更新操作，就将数据同步到数据文件，可以多个条件配合 |
| 10 | rdbcompression yes | 指定存储至本地数据库时是否压缩数据，默认为 yes，Redis 采用 LZF 压缩，如果为了节省 CPU 时间，可以关闭该选项，但会导致数据库文件变的巨大 |
| 11 | dbfilename dump.rdb | 指定本地数据库文件名，默认值为 dump.rdb |
| 12 | dir ./ | 指定本地数据库存放目录 |
| 13 | slaveof <masterip> <masterport> | 设置当本机为 slave 服务时，设置 master 服务的 IP 地址及端口，在 Redis 启动时，它会自动从 master 进行数据同步 |
| 14 | masterauth <master-password> | 当 master 服务设置了密码保护时，slav 服务连接 master 的密码 |
| 15 | requirepass foobared | 设置 Redis 连接密码，如果配置了连接密码，客户端在连接 Redis 时需要通过 AUTH <password> 命令提供密码，默认关闭 |
| 16 | maxclients 128 | 设置同一时间最大客户端连接数，默认无限制，Redis 可以同时打开的客户端连接数为 Redis 进程可以打开的最大文件描述符数，如果设置 maxclients 0，表示不作限制。当客户端连接数到达限制时，Redis 会关闭新的连接并向客户端返回 max number of clients reached 错误信息 |
| 17 | maxmemory <bytes> | 指定 Redis 最大内存限制，Redis 在启动时会把数据加载到内存中，达到最大内存后，Redis 会先尝试清除已到期或即将到期的 Key，当此方法处理 后，仍然到达最大内存设置，将无法再进行写入操作，但仍然可以进行读取操作。Redis 新的 vm 机制，会把 Key 存放内存，Value 会存放在 swap 区 |
| 18 | appendonly no | 指定是否在每次更新操作后进行日志记录，Redis 在默认情况下是异步的把数据写入磁盘，如果不开启，可能会在断电时导致一段时间内的数据丢失。因为 redis 本身同步数据文件是按上面 save 条件来同步的，所以有的数据会在一段时间内只存在于内存中。默认为 no |
| 19 | appendfilename appendonly.aof | 指定更新日志文件名，默认为 appendonly.aof |
| 20 | appendfsync everysec | 指定更新日志条件，共有 3 个可选值：   * **no**：表示等操作系统进行数据缓存同步到磁盘（快） * **always**：表示每次更新操作后手动调用 fsync() 将数据写到磁盘（慢，安全） * **everysec**：表示每秒同步一次（折中，默认值） |
| 21 | vm-enabled no | 指定是否启用虚拟内存机制，默认值为 no，简单的介绍一下，VM 机制将数据分页存放，由 Redis 将访问量较少的页即冷数据 swap 到磁盘上，访问多的页面由磁盘自动换出到内存中（在后面的文章我会仔细分析 Redis 的 VM 机制） |
| 22 | vm-swap-file /tmp/redis.swap | 虚拟内存文件路径，默认值为 /tmp/redis.swap，不可多个 Redis 实例共享 |
| 23 | vm-max-memory 0 | 将所有大于 vm-max-memory 的数据存入虚拟内存，无论 vm-max-memory 设置多小，所有索引数据都是内存存储的(Redis 的索引数据 就是 keys)，也就是说，当 vm-max-memory 设置为 0 的时候，其实是所有 value 都存在于磁盘。默认值为 0 |
| 24 | vm-page-size 32 | Redis swap 文件分成了很多的 page，一个对象可以保存在多个 page 上面，但一个 page 上不能被多个对象共享，vm-page-size 是要根据存储的 数据大小来设定的，作者建议如果存储很多小对象，page 大小最好设置为 32 或者 64bytes；如果存储很大大对象，则可以使用更大的 page，如果不确定，就使用默认值 |
| 25 | vm-pages 134217728 | 设置 swap 文件中的 page 数量，由于页表（一种表示页面空闲或使用的 bitmap）是在放在内存中的，，在磁盘上每 8 个 pages 将消耗 1byte 的内存。 |
| 26 | vm-max-threads 4 | 设置访问swap文件的线程数,最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的，可能会造成比较长时间的延迟。默认值为4 |
| 27 | glueoutputbuf yes | 设置在向客户端应答时，是否把较小的包合并为一个包发送，默认为开启 |
| 28 | hash-max-zipmap-entries 64  hash-max-zipmap-value 512 | 指定在超过一定的数量或者最大的元素超过某一临界值时，采用一种特殊的哈希算法 |
| 29 | activerehashing yes | 指定是否激活重置哈希，默认为开启（后面在介绍 Redis 的哈希算法时具体介绍） |
| 30 | include /path/to/local.conf | 指定包含其它的配置文件，可以在同一主机上多个Redis实例之间使用同一份配置文件，而同时各个实例又拥有自己的特定配置文件 |

## Redis 数据类型

Redis支持五种数据类型：string（字符串），hash（哈希），list（列表），set（集合）及zset(sorted set：有序集合)。

### String（字符串）

string 是 redis 最基本的类型，你可以理解成与 Memcached 一模一样的类型，一个 key 对应一个 value。

string 类型是二进制安全的。意思是 redis 的 string 可以包含任何数据。比如jpg图片或者序列化的对象。

string 类型是 Redis 最基本的数据类型，string 类型的值最大能存储 512MB。

#### 实例

redis 127.0.0.1:6379> SET runoob "菜鸟教程"

OK

redis 127.0.0.1:6379> GET runoob

"菜鸟教程"

在以上实例中我们使用了 Redis 的 **SET** 和 **GET** 命令。键为 runoob，对应的值为 **菜鸟教程**。

**注意：**一个键最大能存储 512MB。

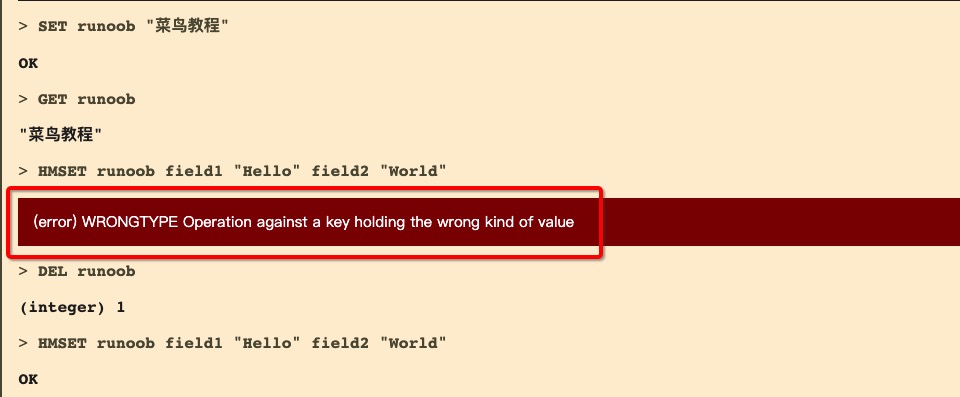
### Hash（哈希）

Redis hash 是一个键值(key=>value)对集合。

Redis hash 是一个 string 类型的 field 和 value 的映射表，hash 特别适合用于存储对象。

#### 实例

**DEL runoob** 用于删除前面测试用过的 key，不然会报错：**(error) WRONGTYPE Operation against a key holding the wrong kind of value**



redis 127.0.0.1:6379> DEL runoob

redis 127.0.0.1:6379> HMSET runoob field1 "Hello" field2 "World"

"OK"

redis 127.0.0.1:6379> HGET runoob field1

"Hello"

redis 127.0.0.1:6379> HGET runoob field2

"World"

实例中我们使用了 Redis **HMSET, HGET** 命令，**HMSET** 设置了两个 **field=>value** 对, HGET 获取对应 **field** 对应的 **value**。

每个 hash 可以存储 232 -1 键值对（40多亿）。

### List（列表）

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）。

#### 实例

redis 127.0.0.1:6379> DEL runoob

redis 127.0.0.1:6379> lpush runoob redis

(integer) 1

redis 127.0.0.1:6379> lpush runoob mongodb

(integer) 2

redis 127.0.0.1:6379> lpush runoob rabbitmq

(integer) 3

redis 127.0.0.1:6379> lrange runoob 0 10

1) "rabbitmq"

2) "mongodb"

3) "redis"

redis 127.0.0.1:6379>

列表最多可存储 232 - 1 元素 (4294967295, 每个列表可存储40多亿)。

### Set（集合）

Redis 的 Set 是 string 类型的无序集合。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是 O(1)。

#### sadd 命令

添加一个 string 元素到 key 对应的 set 集合中，成功返回 1，如果元素已经在集合中返回 0。

sadd key member

#### 实例

redis 127.0.0.1:6379> DEL runoob

redis 127.0.0.1:6379> sadd runoob redis

(integer) 1

redis 127.0.0.1:6379> sadd runoob mongodb

(integer) 1

redis 127.0.0.1:6379> sadd runoob rabbitmq

(integer) 1

redis 127.0.0.1:6379> sadd runoob rabbitmq

(integer) 0

redis 127.0.0.1:6379> smembers runoob

1) "redis"

2) "rabbitmq"

3) "mongodb"

**注意：**以上实例中 rabbitmq 添加了两次，但根据集合内元素的唯一性，第二次插入的元素将被忽略。

集合中最大的成员数为 232 - 1(4294967295, 每个集合可存储40多亿个成员)。

### zset(sorted set：有序集合)

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

zset的成员是唯一的,但分数(score)却可以重复。

#### zadd 命令

添加元素到集合，元素在集合中存在则更新对应score

zadd key score member

#### 实例

redis 127.0.0.1:6379> DEL runoob

redis 127.0.0.1:6379> zadd runoob 0 redis

(integer) 1

redis 127.0.0.1:6379> zadd runoob 0 mongodb

(integer) 1

redis 127.0.0.1:6379> zadd runoob 0 rabbitmq

(integer) 1

redis 127.0.0.1:6379> zadd runoob 0 rabbitmq

(integer) 0

redis 127.0.0.1:6379> ZRANGEBYSCORE runoob 0 1000

1) "mongodb"

2) "rabbitmq"

3) "redis"

各个数据类型应用场景：

| **类型** | **简介** | **特性** | **场景** |
| --- | --- | --- | --- |
| String(字符串) | 二进制安全 | 可以包含任何数据,比如jpg图片或者序列化的对象,一个键最大能存储512M | --- |
| Hash(字典) | 键值对集合,即编程语言中的Map类型 | 适合存储对象,并且可以像数据库中update一个属性一样只修改某一项属性值(Memcached中需要取出整个字符串反序列化成对象修改完再序列化存回去) | 存储、读取、修改用户属性 |
| List(列表) | 链表(双向链表) | 增删快,提供了操作某一段元素的API | 1,最新消息排行等功能(比如朋友圈的时间线) 2,消息队列 |
| Set(集合) | 哈希表实现,元素不重复 | 1、添加、删除,查找的复杂度都是O(1) 2、为集合提供了求交集、并集、差集等操作 | 1、共同好友 2、利用唯一性,统计访问网站的所有独立ip 3、好友推荐时,根据tag求交集,大于某个阈值就可以推荐 |
| Sorted Set(有序集合) | 将Set中的元素增加一个权重参数score,元素按score有序排列 | 数据插入集合时,已经进行天然排序 | 1、排行榜 2、带权重的消息队列 |

[FK](javascript:;)

   FK

  429\*\*\*967@qq.com

3年前 (2018-05-30)

1. fsisjacky

  fsi\*\*\*cky@163.com

[参考地址](https://www.cnblogs.com/EasonJim/p/7818004.html)

525

注意：Redis支持多个数据库，并且每个数据库的数据是隔离的不能共享，并且基于单机才有，如果是集群就没有数据库的概念。

Redis是一个字典结构的存储服务器，而实际上一个Redis实例提供了多个用来存储数据的字典，客户端可以指定将数据存储在哪个字典中。这与我们熟知的在一个关系数据库实例中可以创建多个数据库类似，所以可以将其中的每个字典都理解成一个独立的数据库。

每个数据库对外都是一个从0开始的递增数字命名，Redis默认支持16个数据库（可以通过配置文件支持更多，无上限），可以通过配置databases来修改这一数字。客户端与Redis建立连接后会自动选择0号数据库，不过可以随时使用SELECT命令更换数据库，如要选择1号数据库：

redis> SELECT 1

OK

redis [1] > GET foo

(nil)

然而这些以数字命名的数据库又与我们理解的数据库有所区别。首先Redis不支持自定义数据库的名字，每个数据库都以编号命名，开发者必须自己记录哪些数据库存储了哪些数据。另外Redis也不支持为每个数据库设置不同的访问密码，所以一个客户端要么可以访问全部数据库，要么连一个数据库也没有权限访问。最重要的一点是多个数据库之间并不是完全隔离的，比如FLUSHALL命令可以清空一个Redis实例中所有数据库中的数据。综上所述，这些数据库更像是一种命名空间，而不适宜存储不同应用程序的数据。比如可以使用0号数据库存储某个应用生产环境中的数据，使用1号数据库存储测试环境中的数据，但不适宜使用0号数据库存储A应用的数据而使用1号数据库B应用的数据，不同的应用应该使用不同的Redis实例存储数据。由于Redis非常轻量级，一个空Redis实例占用的内在只有1M左右，所以不用担心多个Redis实例会额外占用很多内存。

## Redis 命令

Redis 命令用于在 redis 服务上执行操作。

要在 redis 服务上执行命令需要一个 redis 客户端。Redis 客户端在我们之前下载的的 redis 的安装包中。

#### 语法

Redis 客户端的基本语法为：

$ redis-cli

#### 实例

以下实例讲解了如何启动 redis 客户端：

启动 redis 服务器，打开终端并输入命令 **redis-cli**，该命令会连接本地的 redis 服务。

$ redis-cli

redis 127.0.0.1:6379>

redis 127.0.0.1:6379> PING

PONG

在以上实例中我们连接到本地的 redis 服务并执行 **PING** 命令，该命令用于检测 redis 服务是否启动。

### 在远程服务上执行命令

如果需要在远程 redis 服务上执行命令，同样我们使用的也是 **redis-cli** 命令。

#### 语法

$ redis-cli -h host -p port -a password

#### 实例

以下实例演示了如何连接到主机为 127.0.0.1，端口为 6379 ，密码为 mypass 的 redis 服务上。

$redis-cli -h 127.0.0.1 -p 6379 -a "mypass"

redis 127.0.0.1:6379>

redis 127.0.0.1:6379> PING

PONG

## Redis 键(key)

Redis 键命令用于管理 redis 的键。

#### 语法

Redis 键命令的基本语法如下：

redis 127.0.0.1:6379> COMMAND KEY\_NAME

#### 实例

redis 127.0.0.1:6379> SET runoobkey redis

OK

redis 127.0.0.1:6379> DEL runoobkey

(integer) 1

在以上实例中 **DEL** 是一个命令， **runoobkey** 是一个键。 如果键被删除成功，命令执行后输出 **(integer) 1**，否则将输出 **(integer) 0**

### Redis keys 命令

下表给出了与 Redis 键相关的基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [DEL key](https://www.runoob.com/redis/keys-del.html) 该命令用于在 key 存在时删除 key。 |
| 2 | [DUMP key](https://www.runoob.com/redis/keys-dump.html) 序列化给定 key ，并返回被序列化的值。 |
| 3 | [EXISTS key](https://www.runoob.com/redis/keys-exists.html) 检查给定 key 是否存在。 |
| 4 | [EXPIRE key](https://www.runoob.com/redis/keys-expire.html) seconds 为给定 key 设置过期时间，以秒计。 |
| 5 | [EXPIREAT key timestamp](https://www.runoob.com/redis/keys-expireat.html) EXPIREAT 的作用和 EXPIRE 类似，都用于为 key 设置过期时间。 不同在于 EXPIREAT 命令接受的时间参数是 UNIX 时间戳(unix timestamp)。 |
| 6 | [PEXPIRE key milliseconds](https://www.runoob.com/redis/keys-pexpire.html) 设置 key 的过期时间以毫秒计。 |
| 7 | [PEXPIREAT key milliseconds-timestamp](https://www.runoob.com/redis/keys-pexpireat.html) 设置 key 过期时间的时间戳(unix timestamp) 以毫秒计 |
| 8 | [KEYS pattern](https://www.runoob.com/redis/keys-keys.html) 查找所有符合给定模式( pattern)的 key 。 |
| 9 | [MOVE key db](https://www.runoob.com/redis/keys-move.html) 将当前数据库的 key 移动到给定的数据库 db 当中。 |
| 10 | [PERSIST key](https://www.runoob.com/redis/keys-persist.html) 移除 key 的过期时间，key 将持久保持。 |
| 11 | [PTTL key](https://www.runoob.com/redis/keys-pttl.html) 以毫秒为单位返回 key 的剩余的过期时间。 |
| 12 | [TTL key](https://www.runoob.com/redis/keys-ttl.html) 以秒为单位，返回给定 key 的剩余生存时间(TTL, time to live)。 |
| 13 | [RANDOMKEY](https://www.runoob.com/redis/keys-randomkey.html) 从当前数据库中随机返回一个 key 。 |
| 14 | [RENAME key newkey](https://www.runoob.com/redis/keys-rename.html) 修改 key 的名称 |
| 15 | [RENAMENX key newkey](https://www.runoob.com/redis/keys-renamenx.html) 仅当 newkey 不存在时，将 key 改名为 newkey 。 |
| 16 | [SCAN cursor [MATCH pattern] [COUNT count]](https://www.runoob.com/redis/keys-scan.html) 迭代数据库中的数据库键。 |
| 17 | [TYPE key](https://www.runoob.com/redis/keys-type.html) 返回 key 所储存的值的类型。 |

## Redis 字符串(String)

Redis 字符串数据类型的相关命令用于管理 redis 字符串值，基本语法如下：

#### 语法

redis 127.0.0.1:6379> COMMAND KEY\_NAME

#### 实例

redis 127.0.0.1:6379> SET runoobkey redis

OK

redis 127.0.0.1:6379> GET runoobkey

"redis"

在以上实例中我们使用了 **SET** 和 **GET** 命令，键为 **runoobkey**。

### Redis 字符串命令

下表列出了常用的 redis 字符串命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [SET key value](https://www.runoob.com/redis/strings-set.html) 设置指定 key 的值 |
| 2 | [GET key](https://www.runoob.com/redis/strings-get.html) 获取指定 key 的值。 |
| 3 | [GETRANGE key start end](https://www.runoob.com/redis/strings-getrange.html) 返回 key 中字符串值的子字符 |
| 4 | [GETSET key value](https://www.runoob.com/redis/strings-getset.html) 将给定 key 的值设为 value ，并返回 key 的旧值(old value)。 |
| 5 | [GETBIT key offset](https://www.runoob.com/redis/strings-getbit.html) 对 key 所储存的字符串值，获取指定偏移量上的位(bit)。 |
| 6 | [MGET key1 [key2..]](https://www.runoob.com/redis/strings-mget.html) 获取所有(一个或多个)给定 key 的值。 |
| 7 | [SETBIT key offset value](https://www.runoob.com/redis/strings-setbit.html) 对 key 所储存的字符串值，设置或清除指定偏移量上的位(bit)。 |
| 8 | [SETEX key seconds value](https://www.runoob.com/redis/strings-setex.html) 将值 value 关联到 key ，并将 key 的过期时间设为 seconds (以秒为单位)。 |
| 9 | [SETNX key value](https://www.runoob.com/redis/strings-setnx.html) 只有在 key 不存在时设置 key 的值。 |
| 10 | [SETRANGE key offset value](https://www.runoob.com/redis/strings-setrange.html) 用 value 参数覆写给定 key 所储存的字符串值，从偏移量 offset 开始。 |
| 11 | [STRLEN key](https://www.runoob.com/redis/strings-strlen.html) 返回 key 所储存的字符串值的长度。 |
| 12 | [MSET key value [key value ...]](https://www.runoob.com/redis/strings-mset.html) 同时设置一个或多个 key-value 对。 |
| 13 | [MSETNX key value [key value ...]](https://www.runoob.com/redis/strings-msetnx.html) 同时设置一个或多个 key-value 对，当且仅当所有给定 key 都不存在。 |
| 14 | [PSETEX key milliseconds value](https://www.runoob.com/redis/strings-psetex.html) 这个命令和 SETEX 命令相似，但它以毫秒为单位设置 key 的生存时间，而不是像 SETEX 命令那样，以秒为单位。 |
| 15 | [INCR key](https://www.runoob.com/redis/strings-incr.html) 将 key 中储存的数字值增一。 |
| 16 | [INCRBY key increment](https://www.runoob.com/redis/strings-incrby.html) 将 key 所储存的值加上给定的增量值（increment） 。 |
| 17 | [INCRBYFLOAT key increment](https://www.runoob.com/redis/strings-incrbyfloat.html) 将 key 所储存的值加上给定的浮点增量值（increment） 。 |
| 18 | [DECR key](https://www.runoob.com/redis/strings-decr.html) 将 key 中储存的数字值减一。 |
| 19 | [DECRBY key decrement](https://www.runoob.com/redis/strings-decrby.html) key 所储存的值减去给定的减量值（decrement） 。 |
| 20 | [APPEND key value](https://www.runoob.com/redis/strings-append.html) 如果 key 已经存在并且是一个字符串， APPEND 命令将指定的 value 追加到该 key 原来值（value）的末尾。 |

## Redis 哈希(Hash)

Redis hash 是一个 string 类型的 field（字段） 和 value（值） 的映射表，hash 特别适合用于存储对象。

Redis 中每个 hash 可以存储 232 - 1 键值对（40多亿）。

#### 实例

127.0.0.1:6379> HMSET runoobkey name "redis tutorial" description "redis basic commands for caching" likes 20 visitors 23000

OK

127.0.0.1:6379> HGETALL runoobkey

1) "name"

2) "redis tutorial"

3) "description"

4) "redis basic commands for caching"

5) "likes"

6) "20"

7) "visitors"

8) "23000"

在以上实例中，我们设置了 redis 的一些描述信息(name, description, likes, visitors) 到哈希表的 **runoobkey** 中。

### Redis hash 命令

下表列出了 redis hash 基本的相关命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [HDEL key field1 [field2]](https://www.runoob.com/redis/hashes-hdel.html) 删除一个或多个哈希表字段 |
| 2 | [HEXISTS key field](https://www.runoob.com/redis/hashes-hexists.html) 查看哈希表 key 中，指定的字段是否存在。 |
| 3 | [HGET key field](https://www.runoob.com/redis/hashes-hget.html) 获取存储在哈希表中指定字段的值。 |
| 4 | [HGETALL key](https://www.runoob.com/redis/hashes-hgetall.html) 获取在哈希表中指定 key 的所有字段和值 |
| 5 | [HINCRBY key field increment](https://www.runoob.com/redis/hashes-hincrby.html) 为哈希表 key 中的指定字段的整数值加上增量 increment 。 |
| 6 | [HINCRBYFLOAT key field increment](https://www.runoob.com/redis/hashes-hincrbyfloat.html) 为哈希表 key 中的指定字段的浮点数值加上增量 increment 。 |
| 7 | [HKEYS key](https://www.runoob.com/redis/hashes-hkeys.html) 获取所有哈希表中的字段 |
| 8 | [HLEN key](https://www.runoob.com/redis/hashes-hlen.html) 获取哈希表中字段的数量 |
| 9 | [HMGET key field1 [field2]](https://www.runoob.com/redis/hashes-hmget.html) 获取所有给定字段的值 |
| 10 | [HMSET key field1 value1 [field2 value2 ]](https://www.runoob.com/redis/hashes-hmset.html) 同时将多个 field-value (域-值)对设置到哈希表 key 中。 |
| 11 | [HSET key field value](https://www.runoob.com/redis/hashes-hset.html) 将哈希表 key 中的字段 field 的值设为 value 。 |
| 12 | [HSETNX key field value](https://www.runoob.com/redis/hashes-hsetnx.html) 只有在字段 field 不存在时，设置哈希表字段的值。 |
| 13 | [HVALS key](https://www.runoob.com/redis/hashes-hvals.html) 获取哈希表中所有值。 |
| 14 | [HSCAN key cursor [MATCH pattern] [COUNT count]](https://www.runoob.com/redis/hashes-hscan.html) 迭代哈希表中的键值对。 |

## Redis 列表(List)

Redis列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）

一个列表最多可以包含 232 - 1 个元素 (4294967295, 每个列表超过40亿个元素)。

#### 实例

redis 127.0.0.1:6379> LPUSH runoobkey redis

(integer) 1

redis 127.0.0.1:6379> LPUSH runoobkey mongodb

(integer) 2

redis 127.0.0.1:6379> LPUSH runoobkey mysql

(integer) 3

redis 127.0.0.1:6379> LRANGE runoobkey 0 10

1) "mysql"

2) "mongodb"

3) "redis"

在以上实例中我们使用了 **LPUSH** 将三个值插入了名为 **runoobkey** 的列表当中。

#### Redis 列表命令

下表列出了列表相关的基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [BLPOP key1 [key2 ] timeout](https://www.runoob.com/redis/lists-blpop.html) 移出并获取列表的第一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。 |
| 2 | [BRPOP key1 [key2 ] timeout](https://www.runoob.com/redis/lists-brpop.html) 移出并获取列表的最后一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。 |
| 3 | [BRPOPLPUSH source destination timeout](https://www.runoob.com/redis/lists-brpoplpush.html) 从列表中弹出一个值，将弹出的元素插入到另外一个列表中并返回它； 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。 |
| 4 | [LINDEX key index](https://www.runoob.com/redis/lists-lindex.html) 通过索引获取列表中的元素 |
| 5 | [LINSERT key BEFORE|AFTER pivot value](https://www.runoob.com/redis/lists-linsert.html) 在列表的元素前或者后插入元素 |
| 6 | [LLEN key](https://www.runoob.com/redis/lists-llen.html) 获取列表长度 |
| 7 | [LPOP key](https://www.runoob.com/redis/lists-lpop.html) 移出并获取列表的第一个元素 |
| 8 | [LPUSH key value1 [value2]](https://www.runoob.com/redis/lists-lpush.html) 将一个或多个值插入到列表头部 |
| 9 | [LPUSHX key value](https://www.runoob.com/redis/lists-lpushx.html) 将一个值插入到已存在的列表头部 |
| 10 | [LRANGE key start stop](https://www.runoob.com/redis/lists-lrange.html) 获取列表指定范围内的元素 |
| 11 | [LREM key count value](https://www.runoob.com/redis/lists-lrem.html) 移除列表元素 |
| 12 | [LSET key index value](https://www.runoob.com/redis/lists-lset.html) 通过索引设置列表元素的值 |
| 13 | [LTRIM key start stop](https://www.runoob.com/redis/lists-ltrim.html) 对一个列表进行修剪(trim)，就是说，让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。 |
| 14 | [RPOP key](https://www.runoob.com/redis/lists-rpop.html) 移除列表的最后一个元素，返回值为移除的元素。 |
| 15 | [RPOPLPUSH source destination](https://www.runoob.com/redis/lists-rpoplpush.html) 移除列表的最后一个元素，并将该元素添加到另一个列表并返回 |
| 16 | [RPUSH key value1 [value2]](https://www.runoob.com/redis/lists-rpush.html) 在列表中添加一个或多个值 |
| 17 | [RPUSHX key value](https://www.runoob.com/redis/lists-rpushx.html) 为已存在的列表添加值 |

## Redis 集合(Set)

Redis 的 Set 是 String 类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。

Redis 中集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是 O(1)。

集合中最大的成员数为 232 - 1 (4294967295, 每个集合可存储40多亿个成员)。

#### 实例

redis 127.0.0.1:6379> SADD runoobkey redis

(integer) 1

redis 127.0.0.1:6379> SADD runoobkey mongodb

(integer) 1

redis 127.0.0.1:6379> SADD runoobkey mysql

(integer) 1

redis 127.0.0.1:6379> SADD runoobkey mysql

(integer) 0

redis 127.0.0.1:6379> SMEMBERS runoobkey

1) "mysql"

2) "mongodb"

3) "redis"

在以上实例中我们通过 **SADD** 命令向名为 **runoobkey** 的集合插入的三个元素。

### Redis 集合命令

下表列出了 Redis 集合基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [SADD key member1 [member2]](https://www.runoob.com/redis/sets-sadd.html) 向集合添加一个或多个成员 |
| 2 | [SCARD key](https://www.runoob.com/redis/sets-scard.html) 获取集合的成员数 |
| 3 | [SDIFF key1 [key2]](https://www.runoob.com/redis/sets-sdiff.html) 返回第一个集合与其他集合之间的差异。 |
| 4 | [SDIFFSTORE destination key1 [key2]](https://www.runoob.com/redis/sets-sdiffstore.html) 返回给定所有集合的差集并存储在 destination 中 |
| 5 | [SINTER key1 [key2]](https://www.runoob.com/redis/sets-sinter.html) 返回给定所有集合的交集 |
| 6 | [SINTERSTORE destination key1 [key2]](https://www.runoob.com/redis/sets-sinterstore.html) 返回给定所有集合的交集并存储在 destination 中 |
| 7 | [SISMEMBER key member](https://www.runoob.com/redis/sets-sismember.html) 判断 member 元素是否是集合 key 的成员 |
| 8 | [SMEMBERS key](https://www.runoob.com/redis/sets-smembers.html) 返回集合中的所有成员 |
| 9 | [SMOVE source destination member](https://www.runoob.com/redis/sets-smove.html) 将 member 元素从 source 集合移动到 destination 集合 |
| 10 | [SPOP key](https://www.runoob.com/redis/sets-spop.html) 移除并返回集合中的一个随机元素 |
| 11 | [SRANDMEMBER key [count]](https://www.runoob.com/redis/sets-srandmember.html) 返回集合中一个或多个随机数 |
| 12 | [SREM key member1 [member2]](https://www.runoob.com/redis/sets-srem.html) 移除集合中一个或多个成员 |
| 13 | [SUNION key1 [key2]](https://www.runoob.com/redis/sets-sunion.html) 返回所有给定集合的并集 |
| 14 | [SUNIONSTORE destination key1 [key2]](https://www.runoob.com/redis/sets-sunionstore.html) 所有给定集合的并集存储在 destination 集合中 |
| 15 | [SSCAN key cursor [MATCH pattern] [COUNT count]](https://www.runoob.com/redis/sets-sscan.html) 迭代集合中的元素 |

## Redis 有序集合(sorted set)

Redis 有序集合和集合一样也是 string 类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个 double 类型的分数。redis 正是通过分数来为集合中的成员进行从小到大的排序。

有序集合的成员是唯一的,但分数(score)却可以重复。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是 O(1)。 集合中最大的成员数为 232 - 1 (4294967295, 每个集合可存储40多亿个成员)。

#### 实例

redis 127.0.0.1:6379> ZADD runoobkey 1 redis

(integer) 1

redis 127.0.0.1:6379> ZADD runoobkey 2 mongodb

(integer) 1

redis 127.0.0.1:6379> ZADD runoobkey 3 mysql

(integer) 1

redis 127.0.0.1:6379> ZADD runoobkey 3 mysql

(integer) 0

redis 127.0.0.1:6379> ZADD runoobkey 4 mysql

(integer) 0

redis 127.0.0.1:6379> ZRANGE runoobkey 0 10 WITHSCORES

1) "redis"

2) "1"

3) "mongodb"

4) "2"

5) "mysql"

6) "4"

在以上实例中我们通过命令 **ZADD** 向 redis 的有序集合中添加了三个值并关联上分数。

### Redis 有序集合命令

下表列出了 redis 有序集合的基本命令:

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [ZADD key score1 member1 [score2 member2]](https://www.runoob.com/redis/sorted-sets-zadd.html) 向有序集合添加一个或多个成员，或者更新已存在成员的分数 |
| 2 | [ZCARD key](https://www.runoob.com/redis/sorted-sets-zcard.html) 获取有序集合的成员数 |
| 3 | [ZCOUNT key min max](https://www.runoob.com/redis/sorted-sets-zcount.html) 计算在有序集合中指定区间分数的成员数 |
| 4 | [ZINCRBY key increment member](https://www.runoob.com/redis/sorted-sets-zincrby.html) 有序集合中对指定成员的分数加上增量 increment |
| 5 | [ZINTERSTORE destination numkeys key [key ...]](https://www.runoob.com/redis/sorted-sets-zinterstore.html) 计算给定的一个或多个有序集的交集并将结果集存储在新的有序集合 destination 中 |
| 6 | [ZLEXCOUNT key min max](https://www.runoob.com/redis/sorted-sets-zlexcount.html) 在有序集合中计算指定字典区间内成员数量 |
| 7 | [ZRANGE key start stop [WITHSCORES]](https://www.runoob.com/redis/sorted-sets-zrange.html) 通过索引区间返回有序集合指定区间内的成员 |
| 8 | [ZRANGEBYLEX key min max [LIMIT offset count]](https://www.runoob.com/redis/sorted-sets-zrangebylex.html) 通过字典区间返回有序集合的成员 |
| 9 | [ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT]](https://www.runoob.com/redis/sorted-sets-zrangebyscore.html) 通过分数返回有序集合指定区间内的成员 |
| 10 | [ZRANK key member](https://www.runoob.com/redis/sorted-sets-zrank.html) 返回有序集合中指定成员的索引 |
| 11 | [ZREM key member [member ...]](https://www.runoob.com/redis/sorted-sets-zrem.html) 移除有序集合中的一个或多个成员 |
| 12 | [ZREMRANGEBYLEX key min max](https://www.runoob.com/redis/sorted-sets-zremrangebylex.html) 移除有序集合中给定的字典区间的所有成员 |
| 13 | [ZREMRANGEBYRANK key start stop](https://www.runoob.com/redis/sorted-sets-zremrangebyrank.html) 移除有序集合中给定的排名区间的所有成员 |
| 14 | [ZREMRANGEBYSCORE key min max](https://www.runoob.com/redis/sorted-sets-zremrangebyscore.html) 移除有序集合中给定的分数区间的所有成员 |
| 15 | [ZREVRANGE key start stop [WITHSCORES]](https://www.runoob.com/redis/sorted-sets-zrevrange.html) 返回有序集中指定区间内的成员，通过索引，分数从高到低 |
| 16 | [ZREVRANGEBYSCORE key max min [WITHSCORES]](https://www.runoob.com/redis/sorted-sets-zrevrangebyscore.html) 返回有序集中指定分数区间内的成员，分数从高到低排序 |
| 17 | [ZREVRANK key member](https://www.runoob.com/redis/sorted-sets-zrevrank.html) 返回有序集合中指定成员的排名，有序集成员按分数值递减(从大到小)排序 |
| 18 | [ZSCORE key member](https://www.runoob.com/redis/sorted-sets-zscore.html) 返回有序集中，成员的分数值 |
| 19 | [ZUNIONSTORE destination numkeys key [key ...]](https://www.runoob.com/redis/sorted-sets-zunionstore.html) 计算给定的一个或多个有序集的并集，并存储在新的 key 中 |
| 20 | [ZSCAN key cursor [MATCH pattern] [COUNT count]](https://www.runoob.com/redis/sorted-sets-zscan.html) 迭代有序集合中的元素（包括元素成员和元素分值） |

## Redis HyperLogLog

Redis 在 2.8.9 版本添加了 HyperLogLog 结构。

Redis HyperLogLog 是用来做基数统计的算法，HyperLogLog 的优点是，在输入元素的数量或者体积非常非常大时，计算基数所需的空间总是固定 的、并且是很小的。

在 Redis 里面，每个 HyperLogLog 键只需要花费 12 KB 内存，就可以计算接近 2^64 个不同元素的基 数。这和计算基数时，元素越多耗费内存就越多的集合形成鲜明对比。

但是，因为 HyperLogLog 只会根据输入元素来计算基数，而不会储存输入元素本身，所以 HyperLogLog 不能像集合那样，返回输入的各个元素。

### 什么是基数?

比如数据集 {1, 3, 5, 7, 5, 7, 8}， 那么这个数据集的基数集为 {1, 3, 5 ,7, 8}, 基数(不重复元素)为5。 基数估计就是在误差可接受的范围内，快速计算基数。

### 实例

以下实例演示了 HyperLogLog 的工作过程：

redis 127.0.0.1:6379> PFADD runoobkey "redis"

1) (integer) 1

redis 127.0.0.1:6379> PFADD runoobkey "mongodb"

1) (integer) 1

redis 127.0.0.1:6379> PFADD runoobkey "mysql"

1) (integer) 1

redis 127.0.0.1:6379> PFCOUNT runoobkey

(integer) 3

### Redis HyperLogLog 命令

下表列出了 redis HyperLogLog 的基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [PFADD key element [element ...]](https://www.runoob.com/redis/hyperloglog-pfadd.html) 添加指定元素到 HyperLogLog 中。 |
| 2 | [PFCOUNT key [key ...]](https://www.runoob.com/redis/hyperloglog-pfcount.html) 返回给定 HyperLogLog 的基数估算值。 |
| 3 | [PFMERGE destkey sourcekey [sourcekey ...]](https://www.runoob.com/redis/hyperloglog-pfmerge.html) 将多个 HyperLogLog 合并为一个 HyperLogLog |

## Redis 发布订阅

Redis 发布订阅 (pub/sub) 是一种消息通信模式：发送者 (pub) 发送消息，订阅者 (sub) 接收消息。

Redis 客户端可以订阅任意数量的频道。

下图展示了频道 channel1 ， 以及订阅这个频道的三个客户端 —— client2 、 client5 和 client1 之间的关系：



当有新消息通过 PUBLISH 命令发送给频道 channel1 时， 这个消息就会被发送给订阅它的三个客户端：



### 实例

以下实例演示了发布订阅是如何工作的，需要开启两个 redis-cli 客户端。

在我们实例中我们创建了订阅频道名为 **runoobChat**:

### 第一个 redis-cli 客户端

redis 127.0.0.1:6379**>** SUBSCRIBE runoobChat  
  
Reading messages... **(**press Ctrl-C to quit**)**  
1**)** "subscribe"  
2**)** "redisChat"  
3**)** **(**integer**)** 1

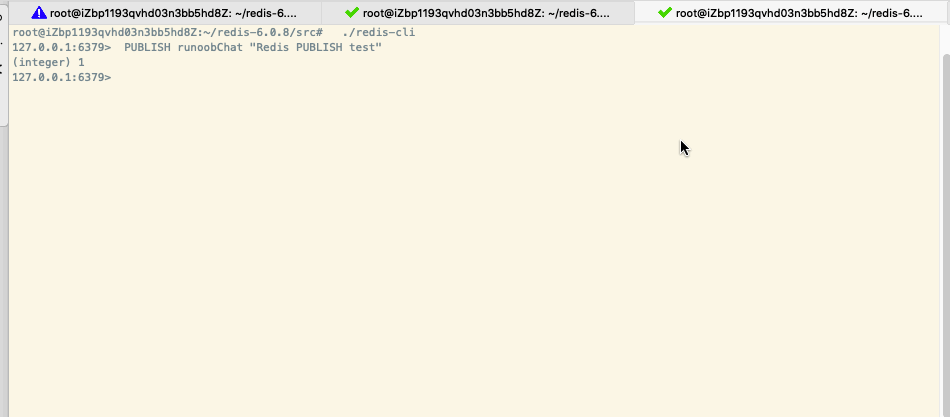
现在，我们先重新开启个 redis 客户端，然后在同一个频道 runoobChat 发布两次消息，订阅者就能接收到消息。

### 第二个 redis-cli 客户端

redis 127.0.0.1:6379> PUBLISH runoobChat "Redis PUBLISH test"  
  
(integer) 1  
  
redis 127.0.0.1:6379> PUBLISH runoobChat "Learn redis by runoob.com"  
  
(integer) 1  
  
# 订阅者的客户端会显示如下消息  
 1) "message"  
2) "runoobChat"  
3) "Redis PUBLISH test"  
 1) "message"  
2) "runoobChat"  
3) "Learn redis by runoob.com"

gif 演示如下：

* 开启本地 Redis 服务，开启两个 redis-cli 客户端。
* 在**第一个 redis-cli 客户端**输入 **SUBSCRIBE runoobChat**，意思是订阅 runoobChat 频道。
* 在**第二个 redis-cli 客户端**输入 **PUBLISH runoobChat "Redis PUBLISH test"** 往 runoobChat 频道发送消息，这个时候在第一个 redis-cli 客户端就会看到由第二个 redis-cli 客户端发送的测试消息。



### Redis 发布订阅命令

下表列出了 redis 发布订阅常用命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [PSUBSCRIBE pattern [pattern ...]](https://www.runoob.com/redis/pub-sub-psubscribe.html) 订阅一个或多个符合给定模式的频道。 |
| 2 | [PUBSUB subcommand [argument [argument ...]]](https://www.runoob.com/redis/pub-sub-pubsub.html) 查看订阅与发布系统状态。 |
| 3 | [PUBLISH channel message](https://www.runoob.com/redis/pub-sub-publish.html) 将信息发送到指定的频道。 |
| 4 | [PUNSUBSCRIBE [pattern [pattern ...]]](https://www.runoob.com/redis/pub-sub-punsubscribe.html) 退订所有给定模式的频道。 |
| 5 | [SUBSCRIBE channel [channel ...]](https://www.runoob.com/redis/pub-sub-subscribe.html) 订阅给定的一个或多个频道的信息。 |
| 6 | [UNSUBSCRIBE [channel [channel ...]]](https://www.runoob.com/redis/pub-sub-unsubscribe.html) 指退订给定的频道。 |

## Redis 事务

Redis 事务可以一次执行多个命令， 并且带有以下三个重要的保证：

* 批量操作在发送 EXEC 命令前被放入队列缓存。
* 收到 EXEC 命令后进入事务执行，事务中任意命令执行失败，其余的命令依然被执行。
* 在事务执行过程，其他客户端提交的命令请求不会插入到事务执行命令序列中。

一个事务从开始到执行会经历以下三个阶段：

* 开始事务。
* 命令入队。
* 执行事务。

### 实例

以下是一个事务的例子， 它先以 **MULTI** 开始一个事务， 然后将多个命令入队到事务中， 最后由 **EXEC** 命令触发事务， 一并执行事务中的所有命令：

redis 127.0.0.1:6379> MULTI

OK

redis 127.0.0.1:6379> SET book-name "Mastering C++ in 21 days"

QUEUED

redis 127.0.0.1:6379> GET book-name

QUEUED

redis 127.0.0.1:6379> SADD tag "C++" "Programming" "Mastering Series"

QUEUED

redis 127.0.0.1:6379> SMEMBERS tag

QUEUED

redis 127.0.0.1:6379> EXEC

1) OK

2) "Mastering C++ in 21 days"

3) (integer) 3

4) 1) "Mastering Series"

2) "C++"

3) "Programming"

单个 Redis 命令的执行是原子性的，但 Redis 没有在事务上增加任何维持原子性的机制，所以 Redis 事务的执行并不是原子性的。

事务可以理解为一个打包的批量执行脚本，但批量指令并非原子化的操作，中间某条指令的失败不会导致前面已做指令的回滚，也不会造成后续的指令不做。

***这是官网上的说明 From redis docs on***[***transactions***](http://redis.io/topics/transactions)***:***

*It's important to note that even when a command fails, all the other commands in the queue are processed – Redis will not stop the processing of commands.*

比如：

redis 127.0.0.1:7000> multi

OK

redis 127.0.0.1:7000> set a aaa

QUEUED

redis 127.0.0.1:7000> set b bbb

QUEUED

redis 127.0.0.1:7000> set c ccc

QUEUED

redis 127.0.0.1:7000> exec

1) OK

2) OK

3) OK

如果在 set b bbb 处失败，set a 已成功不会回滚，set c 还会继续执行。

### Redis 事务命令

下表列出了 redis 事务的相关命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [DISCARD](https://www.runoob.com/redis/transactions-discard.html) 取消事务，放弃执行事务块内的所有命令。 |
| 2 | [EXEC](https://www.runoob.com/redis/transactions-exec.html) 执行所有事务块内的命令。 |
| 3 | [MULTI](https://www.runoob.com/redis/transactions-multi.html) 标记一个事务块的开始。 |
| 4 | [UNWATCH](https://www.runoob.com/redis/transactions-unwatch.html) 取消 WATCH 命令对所有 key 的监视。 |
| 5 | [WATCH key [key ...]](https://www.runoob.com/redis/transactions-watch.html) 监视一个(或多个) key ，如果在事务执行之前这个(或这些) key 被其他命令所改动，那么事务将被打断。 |

## Redis 脚本

Redis 脚本使用 Lua 解释器来执行脚本。 Redis 2.6 版本通过内嵌支持 Lua 环境。执行脚本的常用命令为 **EVAL**。

#### 语法

Eval 命令的基本语法如下：

redis 127.0.0.1:6379> EVAL script numkeys key [key ...] arg [arg ...]

#### 实例

以下实例演示了 redis 脚本工作过程：

redis 127.0.0.1:6379> EVAL "return {KEYS[1],KEYS[2],ARGV[1],ARGV[2]}" 2 key1 key2 first second

1) "key1"

2) "key2"

3) "first"

4) "second"

### Redis 脚本命令

下表列出了 redis 脚本常用命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [EVAL script numkeys key [key ...] arg [arg ...]](https://www.runoob.com/redis/scripting-eval.html) 执行 Lua 脚本。 |
| 2 | [EVALSHA sha1 numkeys key [key ...] arg [arg ...]](https://www.runoob.com/redis/scripting-evalsha.html) 执行 Lua 脚本。 |
| 3 | [SCRIPT EXISTS script [script ...]](https://www.runoob.com/redis/scripting-script-exists.html) 查看指定的脚本是否已经被保存在缓存当中。 |
| 4 | [SCRIPT FLUSH](https://www.runoob.com/redis/scripting-script-flush.html) 从脚本缓存中移除所有脚本。 |
| 5 | [SCRIPT KILL](https://www.runoob.com/redis/scripting-script-kill.html) 杀死当前正在运行的 Lua 脚本。 |
| 6 | [SCRIPT LOAD script](https://www.runoob.com/redis/scripting-script-load.html) 将脚本 script 添加到脚本缓存中，但并不立即执行这个脚本。 |

## Redis 连接

Redis 连接命令主要是用于连接 redis 服务。

#### 实例

以下实例演示了客户端如何通过密码验证连接到 redis 服务，并检测服务是否在运行：

redis 127.0.0.1:6379> AUTH "password"

OK

redis 127.0.0.1:6379> PING

PONG

### Redis 连接命令

下表列出了 redis 连接的基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [AUTH password](https://www.runoob.com/redis/connection-auth.html) 验证密码是否正确 |
| 2 | [ECHO message](https://www.runoob.com/redis/connection-echo.html) 打印字符串 |
| 3 | [PING](https://www.runoob.com/redis/connection-ping.html) 查看服务是否运行 |
| 4 | [QUIT](https://www.runoob.com/redis/connection-quit.html) 关闭当前连接 |
| 5 | [SELECT index](https://www.runoob.com/redis/connection-select.html) 切换到指定的数据库 |

## Redis 服务器

Redis 服务器命令主要是用于管理 redis 服务。

#### 实例

以下实例演示了如何获取 redis 服务器的统计信息：

redis 127.0.0.1:6379> INFO

# Server

redis\_version:2.8.13

redis\_git\_sha1:00000000

redis\_git\_dirty:0

redis\_build\_id:c2238b38b1edb0e2

redis\_mode:standalone

os:Linux 3.5.0-48-generic x86\_64

arch\_bits:64

multiplexing\_api:epoll

gcc\_version:4.7.2

process\_id:3856

run\_id:0e61abd297771de3fe812a3c21027732ac9f41fe

tcp\_port:6379

uptime\_in\_seconds:11554

uptime\_in\_days:0

hz:10

lru\_clock:16651447

config\_file:

# Clients

connected\_clients:1

client-longest\_output\_list:0

client-biggest\_input\_buf:0

blocked\_clients:0

# Memory

used\_memory:589016

used\_memory\_human:575.21K

used\_memory\_rss:2461696

used\_memory\_peak:667312

used\_memory\_peak\_human:651.67K

used\_memory\_lua:33792

mem\_fragmentation\_ratio:4.18

mem\_allocator:jemalloc-3.6.0

# Persistence

loading:0

rdb\_changes\_since\_last\_save:3

rdb\_bgsave\_in\_progress:0

rdb\_last\_save\_time:1409158561

rdb\_last\_bgsave\_status:ok

rdb\_last\_bgsave\_time\_sec:0

rdb\_current\_bgsave\_time\_sec:-1

aof\_enabled:0

aof\_rewrite\_in\_progress:0

aof\_rewrite\_scheduled:0

aof\_last\_rewrite\_time\_sec:-1

aof\_current\_rewrite\_time\_sec:-1

aof\_last\_bgrewrite\_status:ok

aof\_last\_write\_status:ok

# Stats

total\_connections\_received:24

total\_commands\_processed:294

instantaneous\_ops\_per\_sec:0

rejected\_connections:0

sync\_full:0

sync\_partial\_ok:0

sync\_partial\_err:0

expired\_keys:0

evicted\_keys:0

keyspace\_hits:41

keyspace\_misses:82

pubsub\_channels:0

pubsub\_patterns:0

latest\_fork\_usec:264

# Replication

role:master

connected\_slaves:0

master\_repl\_offset:0

repl\_backlog\_active:0

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:0

repl\_backlog\_histlen:0

# CPU

used\_cpu\_sys:10.49

used\_cpu\_user:4.96

used\_cpu\_sys\_children:0.00

used\_cpu\_user\_children:0.01

# Keyspace

db0:keys=94,expires=1,avg\_ttl=41638810

db1:keys=1,expires=0,avg\_ttl=0

db3:keys=1,expires=0,avg\_ttl=0

### Redis 服务器命令

下表列出了 redis 服务器的相关命令:

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [BGREWRITEAOF](https://www.runoob.com/redis/server-bgrewriteaof.html) 异步执行一个 AOF（AppendOnly File） 文件重写操作 |
| 2 | [BGSAVE](https://www.runoob.com/redis/server-bgsave.html) 在后台异步保存当前数据库的数据到磁盘 |
| 3 | [CLIENT KILL [ip:port] [ID client-id]](https://www.runoob.com/redis/server-client-kill.html) 关闭客户端连接 |
| 4 | [CLIENT LIST](https://www.runoob.com/redis/server-client-list.html) 获取连接到服务器的客户端连接列表 |
| 5 | [CLIENT GETNAME](https://www.runoob.com/redis/server-client-getname.html) 获取连接的名称 |
| 6 | [CLIENT PAUSE timeout](https://www.runoob.com/redis/server-client-pause.html) 在指定时间内终止运行来自客户端的命令 |
| 7 | [CLIENT SETNAME connection-name](https://www.runoob.com/redis/server-client-setname.html) 设置当前连接的名称 |
| 8 | [CLUSTER SLOTS](https://www.runoob.com/redis/server-cluster-slots.html) 获取集群节点的映射数组 |
| 9 | [COMMAND](https://www.runoob.com/redis/server-command.html) 获取 Redis 命令详情数组 |
| 10 | [COMMAND COUNT](https://www.runoob.com/redis/server-command-count.html) 获取 Redis 命令总数 |
| 11 | [COMMAND GETKEYS](https://www.runoob.com/redis/server-command-getkeys.html) 获取给定命令的所有键 |
| 12 | [TIME](https://www.runoob.com/redis/server-time.html) 返回当前服务器时间 |
| 13 | [COMMAND INFO command-name [command-name ...]](https://www.runoob.com/redis/server-command-info.html) 获取指定 Redis 命令描述的数组 |
| 14 | [CONFIG GET parameter](https://www.runoob.com/redis/server-config-get.html) 获取指定配置参数的值 |
| 15 | [CONFIG REWRITE](https://www.runoob.com/redis/server-config-rewrite.html) 对启动 Redis 服务器时所指定的 redis.conf 配置文件进行改写 |
| 16 | [CONFIG SET parameter value](https://www.runoob.com/redis/server-config-set.html) 修改 redis 配置参数，无需重启 |
| 17 | [CONFIG RESETSTAT](https://www.runoob.com/redis/server-config-resetstat.html) 重置 INFO 命令中的某些统计数据 |
| 18 | [DBSIZE](https://www.runoob.com/redis/server-dbsize.html) 返回当前数据库的 key 的数量 |
| 19 | [DEBUG OBJECT key](https://www.runoob.com/redis/server-debug-object.html) 获取 key 的调试信息 |
| 20 | [DEBUG SEGFAULT](https://www.runoob.com/redis/server-debug-segfault.html) 让 Redis 服务崩溃 |
| 21 | [FLUSHALL](https://www.runoob.com/redis/server-flushall.html) 删除所有数据库的所有key |
| 22 | [FLUSHDB](https://www.runoob.com/redis/server-flushdb.html) 删除当前数据库的所有key |
| 23 | [INFO [section]](https://www.runoob.com/redis/server-info.html) 获取 Redis 服务器的各种信息和统计数值 |
| 24 | [LASTSAVE](https://www.runoob.com/redis/server-lastsave.html) 返回最近一次 Redis 成功将数据保存到磁盘上的时间，以 UNIX 时间戳格式表示 |
| 25 | [MONITOR](https://www.runoob.com/redis/server-monitor.html) 实时打印出 Redis 服务器接收到的命令，调试用 |
| 26 | [ROLE](https://www.runoob.com/redis/server-role.html) 返回主从实例所属的角色 |
| 27 | [SAVE](https://www.runoob.com/redis/server-save.html) 同步保存数据到硬盘 |
| 28 | [SHUTDOWN [NOSAVE] [SAVE]](https://www.runoob.com/redis/server-shutdown.html) 异步保存数据到硬盘，并关闭服务器 |
| 29 | [SLAVEOF host port](https://www.runoob.com/redis/server-slaveof.html) 将当前服务器转变为指定服务器的从属服务器(slave server) |
| 30 | [SLOWLOG subcommand [argument]](https://www.runoob.com/redis/server-showlog.html) 管理 redis 的慢日志 |
| 31 | [SYNC](https://www.runoob.com/redis/server-sync.html) 用于复制功能(replication)的内部命令 |

## Redis GEO

Redis GEO 主要用于存储地理位置信息，并对存储的信息进行操作，该功能在 Redis 3.2 版本新增。

Redis GEO 操作方法有：

* geoadd：添加地理位置的坐标。
* geopos：获取地理位置的坐标。
* geodist：计算两个位置之间的距离。
* georadius：根据用户给定的经纬度坐标来获取指定范围内的地理位置集合。
* georadiusbymember：根据储存在位置集合里面的某个地点获取指定范围内的地理位置集合。
* geohash：返回一个或多个位置对象的 geohash 值。

**geoadd**

geoadd 用于存储指定的地理空间位置，可以将一个或多个经度(longitude)、纬度(latitude)、位置名称(member)添加到指定的 key 中。

geoadd 语法格式如下：

GEOADD key longitude latitude member [longitude latitude member ...]

以下实例中 key 为 Sicily，Palermo 和 Catania 为位置名称 ：

### 实例

redis**>** GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.502669 "Catania"  
**(**integer**)** 2  
redis**>** GEODIST Sicily Palermo Catania  
"166274.1516"  
redis**>** GEORADIUS Sicily 15 37 100 km  
1**)** "Catania"  
redis**>** GEORADIUS Sicily 15 37 200 km  
1**)** "Palermo"  
2**)** "Catania"  
redis**>**

**geopos**

geopos 用于从给定的 key 里返回所有指定名称(member)的位置（经度和纬度），不存在的返回 nil。

geopos 语法格式如下：

GEOPOS key member [member ...]

### 实例

redis**>** GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.502669 "Catania"  
**(**integer**)** 2  
redis**>** GEOPOS Sicily Palermo Catania NonExisting  
1**)** 1**)** "13.36138933897018433"  
   2**)** "38.11555639549629859"  
2**)** 1**)** "15.08726745843887329"  
   2**)** "37.50266842333162032"  
3**)** **(**nil**)**  
redis**>**

**geodist**

geodist 用于返回两个给定位置之间的距离。

geodist 语法格式如下：

GEODIST key member1 member2 [m|km|ft|mi]

member1 member2 为两个地理位置。

最后一个距离单位参数说明：

* m ：米，默认单位。
* km ：千米。
* mi ：英里。
* ft ：英尺。

> 计算 Palermo 与 Catania 之间的距离：

### 实例

redis**>** GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.502669 "Catania"  
**(**integer**)** 2  
redis**>** GEODIST Sicily Palermo Catania  
"166274.1516"  
redis**>** GEODIST Sicily Palermo Catania km  
"166.2742"  
redis**>** GEODIST Sicily Palermo Catania mi  
"103.3182"  
redis**>** GEODIST Sicily Foo Bar  
**(**nil**)**  
redis**>**

**georadius、georadiusbymember**

georadius 以给定的经纬度为中心， 返回键包含的位置元素当中， 与中心的距离不超过给定最大距离的所有位置元素。

georadiusbymember 和 GEORADIUS 命令一样， 都可以找出位于指定范围内的元素， 但是 georadiusbymember 的中心点是由给定的位置元素决定的， 而不是使用经度和纬度来决定中心点。

georadius 与 georadiusbymember 语法格式如下：

GEORADIUS key longitude latitude radius m|km|ft|mi [WITHCOORD] [WITHDIST] [WITHHASH] [COUNT count] [ASC|DESC] [STORE key] [STOREDIST key]

GEORADIUSBYMEMBER key member radius m|km|ft|mi [WITHCOORD] [WITHDIST] [WITHHASH] [COUNT count] [ASC|DESC] [STORE key] [STOREDIST key]

参数说明：

* + m ：米，默认单位。
  + km ：千米。
  + mi ：英里。
  + ft ：英尺。
  + WITHDIST: 在返回位置元素的同时， 将位置元素与中心之间的距离也一并返回。
  + WITHCOORD: 将位置元素的经度和维度也一并返回。
  + WITHHASH: 以 52 位有符号整数的形式， 返回位置元素经过原始 geohash 编码的有序集合分值。 这个选项主要用于底层应用或者调试， 实际中的作用并不大。
  + COUNT 限定返回的记录数。
  + ASC: 查找结果根据距离从近到远排序。
  + DESC: 查找结果根据从远到近排序。

georadius 实例：

### 实例

redis**>** GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.502669 "Catania"  
**(**integer**)** 2  
redis**>** GEORADIUS Sicily 15 37 200 km WITHDIST  
1**)** 1**)** "Palermo"  
   2**)** "190.4424"  
2**)** 1**)** "Catania"  
   2**)** "56.4413"  
redis**>** GEORADIUS Sicily 15 37 200 km WITHCOORD  
1**)** 1**)** "Palermo"  
   2**)** 1**)** "13.36138933897018433"  
      2**)** "38.11555639549629859"  
2**)** 1**)** "Catania"  
   2**)** 1**)** "15.08726745843887329"  
      2**)** "37.50266842333162032"  
redis**>** GEORADIUS Sicily 15 37 200 km WITHDIST WITHCOORD  
1**)** 1**)** "Palermo"  
   2**)** "190.4424"  
   3**)** 1**)** "13.36138933897018433"  
      2**)** "38.11555639549629859"  
2**)** 1**)** "Catania"  
   2**)** "56.4413"  
   3**)** 1**)** "15.08726745843887329"  
      2**)** "37.50266842333162032"  
redis**>**

georadiusbymember 实例：

### 实例

redis**>** GEOADD Sicily 13.583333 37.316667 "Agrigento"  
**(**integer**)** 1  
redis**>** GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.502669 "Catania"  
**(**integer**)** 2  
redis**>** GEORADIUSBYMEMBER Sicily Agrigento 100 km  
1**)** "Agrigento"  
2**)** "Palermo"  
redis**>**

**geohash**

Redis GEO 使用 geohash 来保存地理位置的坐标。

geohash 用于获取一个或多个位置元素的 geohash 值。

geohash 语法格式如下：

GEOHASH key member [member ...]

实例：

### 实例

redis**>** GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.502669 "Catania"  
**(**integer**)** 2  
redis**>** GEOHASH Sicily Palermo Catania  
1**)** "sqc8b49rny0"  
2**)** "sqdtr74hyu0"  
redis**>**

## Redis Stream

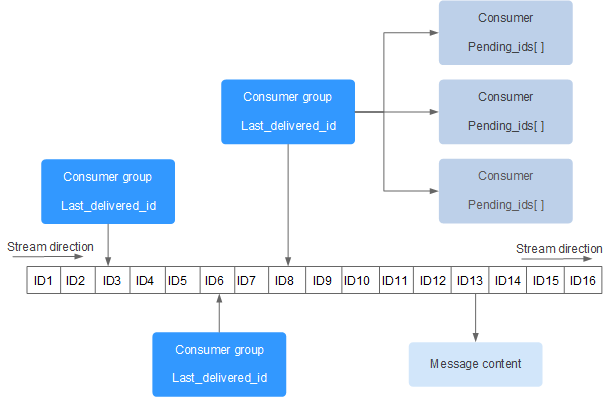
Redis Stream 是 Redis 5.0 版本新增加的数据结构。

Redis Stream 主要用于消息队列（MQ，Message Queue），Redis 本身是有一个 Redis 发布订阅 (pub/sub) 来实现消息队列的功能，但它有个缺点就是消息无法持久化，如果出现网络断开、Redis 宕机等，消息就会被丢弃。

简单来说发布订阅 (pub/sub) 可以分发消息，但无法记录历史消息。

而 Redis Stream 提供了消息的持久化和主备复制功能，可以让任何客户端访问任何时刻的数据，并且能记住每一个客户端的访问位置，还能保证消息不丢失。

Redis Stream 的结构如下所示，它有一个消息链表，将所有加入的消息都串起来，每个消息都有一个唯一的 ID 和对应的内容：



每个 Stream 都有唯一的名称，它就是 Redis 的 key，在我们首次使用 xadd 指令追加消息时自动创建。

上图解析：

* **Consumer Group** ：消费组，使用 XGROUP CREATE 命令创建，一个消费组有多个消费者(Consumer)。
* **last\_delivered\_id** ：游标，每个消费组会有个游标 last\_delivered\_id，任意一个消费者读取了消息都会使游标 last\_delivered\_id 往前移动。
* **pending\_ids** ：消费者(Consumer)的状态变量，作用是维护消费者的未确认的 id。 pending\_ids 记录了当前已经被客户端读取的消息，但是还没有 ack (Acknowledge character：确认字符）。

**消息队列相关命令：**

* **XADD** - 添加消息到末尾
* **XTRIM** - 对流进行修剪，限制长度
* **XDEL** - 删除消息
* **XLEN** - 获取流包含的元素数量，即消息长度
* **XRANGE** - 获取消息列表，会自动过滤已经删除的消息
* **XREVRANGE** - 反向获取消息列表，ID 从大到小
* **XREAD** - 以阻塞或非阻塞方式获取消息列表

**消费者组相关命令：**

* **XGROUP CREATE** - 创建消费者组
* **XREADGROUP GROUP** - 读取消费者组中的消息
* **XACK** - 将消息标记为"已处理"
* **XGROUP SETID** - 为消费者组设置新的最后递送消息ID
* **XGROUP DELCONSUMER** - 删除消费者
* **XGROUP DESTROY** - 删除消费者组
* **XPENDING** - 显示待处理消息的相关信息
* **XCLAIM** - 转移消息的归属权
* **XINFO**- 查看流和消费者组的相关信息；
* **XINFO GROUPS** - 打印消费者组的信息；
* **XINFO STREAM** - 打印流信息

**XADD**

使用 XADD 向队列添加消息，如果指定的队列不存在，则创建一个队列，XADD 语法格式：

XADD key ID field value [field value ...]

* **key** ：队列名称，如果不存在就创建
* **ID** ：消息 id，我们使用 \* 表示由 redis 生成，可以自定义，但是要自己保证递增性。
* **field value** ： 记录。

### 实例

redis**>** XADD mystream **\*** name Sara surname OConnor  
"1601372323627-0"  
redis**>** XADD mystream **\*** field1 value1 field2 value2 field3 value3  
"1601372323627-1"  
redis**>** XLEN mystream  
**(**integer**)** 2  
redis**>** XRANGE mystream - +  
1**)** 1**)** "1601372323627-0"  
   2**)** 1**)** "name"  
      2**)** "Sara"  
      3**)** "surname"  
      4**)** "OConnor"  
2**)** 1**)** "1601372323627-1"  
   2**)** 1**)** "field1"  
      2**)** "value1"  
      3**)** "field2"  
      4**)** "value2"  
      5**)** "field3"  
      6**)** "value3"  
redis**>**

**XTRIM**

使用 XTRIM 对流进行修剪，限制长度， 语法格式：

XTRIM key MAXLEN [~] count

* **key** ：队列名称
* **MAXLEN** ：长度
* **count** ：数量

### 实例

127.0.0.1:6379**>** XADD mystream **\*** field1 A field2 B field3 C field4 D  
"1601372434568-0"  
127.0.0.1:6379**>** XTRIM mystream MAXLEN 2  
**(**integer**)** 0  
127.0.0.1:6379**>** XRANGE mystream - +  
1**)** 1**)** "1601372434568-0"  
   2**)** 1**)** "field1"  
      2**)** "A"  
      3**)** "field2"  
      4**)** "B"  
      5**)** "field3"  
      6**)** "C"  
      7**)** "field4"  
      8**)** "D"  
127.0.0.1:6379**>**  
  
redis**>**

**XDEL**

使用 XDEL 删除消息，语法格式：

XDEL key ID [ID ...]

* **key**：队列名称
* **ID** ：消息 ID

### 实例

**>** XADD mystream **\*** a 1  
1538561698944-0  
**>** XADD mystream **\*** b 2  
1538561700640-0  
**>** XADD mystream **\*** c 3  
1538561701744-0  
**>** XDEL mystream 1538561700640-0  
**(**integer**)** 1  
127.0.0.1:6379**>** XRANGE mystream - +  
1**)** 1**)** 1538561698944-0  
   2**)** 1**)** "a"  
      2**)** "1"  
2**)** 1**)** 1538561701744-0  
   2**)** 1**)** "c"  
      2**)** "3"

**XLEN**

使用 XLEN 获取流包含的元素数量，即消息长度，语法格式：

XLEN key

* **key**：队列名称

### 实例

redis**>** XADD mystream **\*** item 1  
"1601372563177-0"  
redis**>** XADD mystream **\*** item 2  
"1601372563178-0"  
redis**>** XADD mystream **\*** item 3  
"1601372563178-1"  
redis**>** XLEN mystream  
**(**integer**)** 3  
redis**>**

**XRANGE**

使用 XRANGE 获取消息列表，会自动过滤已经删除的消息 ，语法格式：

XRANGE key start end [COUNT count]

* **key** ：队列名
* **start** ：开始值， **-** 表示最小值
* **end** ：结束值， **+** 表示最大值
* **count** ：数量

### 实例

redis**>** XADD writers **\*** name Virginia surname Woolf  
"1601372577811-0"  
redis**>** XADD writers **\*** name Jane surname Austen  
"1601372577811-1"  
redis**>** XADD writers **\*** name Toni surname Morrison  
"1601372577811-2"  
redis**>** XADD writers **\*** name Agatha surname Christie  
"1601372577812-0"  
redis**>** XADD writers **\*** name Ngozi surname Adichie  
"1601372577812-1"  
redis**>** XLEN writers  
**(**integer**)** 5  
redis**>** XRANGE writers - + COUNT 2  
1**)** 1**)** "1601372577811-0"  
   2**)** 1**)** "name"  
      2**)** "Virginia"  
      3**)** "surname"  
      4**)** "Woolf"  
2**)** 1**)** "1601372577811-1"  
   2**)** 1**)** "name"  
      2**)** "Jane"  
      3**)** "surname"  
      4**)** "Austen"  
redis**>**

**XREVRANGE**

使用 XREVRANGE 获取消息列表，会自动过滤已经删除的消息 ，语法格式：

XREVRANGE key end start [COUNT count]

* **key** ：队列名
* **end** ：结束值， **+** 表示最大值
* **start** ：开始值， **-** 表示最小值
* **count** ：数量

### 实例

redis**>** XADD writers **\*** name Virginia surname Woolf  
"1601372731458-0"  
redis**>** XADD writers **\*** name Jane surname Austen  
"1601372731459-0"  
redis**>** XADD writers **\*** name Toni surname Morrison  
"1601372731459-1"  
redis**>** XADD writers **\*** name Agatha surname Christie  
"1601372731459-2"  
redis**>** XADD writers **\*** name Ngozi surname Adichie  
"1601372731459-3"  
redis**>** XLEN writers  
**(**integer**)** 5  
redis**>** XREVRANGE writers + - COUNT 1  
1**)** 1**)** "1601372731459-3"  
   2**)** 1**)** "name"  
      2**)** "Ngozi"  
      3**)** "surname"  
      4**)** "Adichie"  
redis**>**

**XREAD**

使用 XREAD 以阻塞或非阻塞方式获取消息列表 ，语法格式：

XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] id [id ...]

* **count** ：数量
* **milliseconds** ：可选，阻塞毫秒数，没有设置就是非阻塞模式
* **key** ：队列名
* **id** ：消息 ID

### 实例

*# 从 Stream 头部读取两条消息*  
**>** XREAD COUNT 2 STREAMS mystream writers 0-0 0-0  
1**)** 1**)** "mystream"  
   2**)** 1**)** 1**)** 1526984818136-0  
         2**)** 1**)** "duration"  
            2**)** "1532"  
            3**)** "event-id"  
            4**)** "5"  
            5**)** "user-id"  
            6**)** "7782813"  
      2**)** 1**)** 1526999352406-0  
         2**)** 1**)** "duration"  
            2**)** "812"  
            3**)** "event-id"  
            4**)** "9"  
            5**)** "user-id"  
            6**)** "388234"  
2**)** 1**)** "writers"  
   2**)** 1**)** 1**)** 1526985676425-0  
         2**)** 1**)** "name"  
            2**)** "Virginia"  
            3**)** "surname"  
            4**)** "Woolf"  
      2**)** 1**)** 1526985685298-0  
         2**)** 1**)** "name"  
            2**)** "Jane"  
            3**)** "surname"  
            4**)** "Austen"

**XGROUP CREATE**

使用 XGROUP CREATE 创建消费者组，语法格式：

XGROUP [CREATE key groupname id-or-$] [SETID key groupname id-or-$] [DESTROY key groupname] [DELCONSUMER key groupname consumername]

* **key** ：队列名称，如果不存在就创建
* **groupname** ：组名。
* **$** ： 表示从尾部开始消费，只接受新消息，当前 Stream 消息会全部忽略。

从头开始消费:

XGROUP CREATE mystream consumer-group-name 0-0

从尾部开始消费:

XGROUP CREATE mystream consumer-group-name $

**XREADGROUP GROUP**

使用 XREADGROUP GROUP 读取消费组中的消息，语法格式：

XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] [NOACK] STREAMS key [key ...] ID [ID ...]

* **group** ：消费组名
* **consumer** ：消费者名。
* **count** ： 读取数量。
* **milliseconds** ： 阻塞毫秒数。
* **key** ： 队列名。
* **ID** ： 消息 ID。

XREADGROUP GROUP consumer-group-name consumer-name COUNT 1 STREAMS mystream >

## Redis 数据备份与恢复

Redis **SAVE** 命令用于创建当前数据库的备份。

#### 语法

redis Save 命令基本语法如下：

redis 127.0.0.1:6379> SAVE

#### 实例

redis 127.0.0.1:6379> SAVE

OK

该命令将在 redis 安装目录中创建dump.rdb文件。

### 恢复数据

如果需要恢复数据，只需将备份文件 (dump.rdb) 移动到 redis 安装目录并启动服务即可。获取 redis 目录可以使用 **CONFIG** 命令，如下所示：

redis 127.0.0.1:6379> CONFIG GET dir

1) "dir"

2) "/usr/local/redis/bin"

以上命令 **CONFIG GET dir** 输出的 redis 安装目录为 /usr/local/redis/bin。

### Bgsave

创建 redis 备份文件也可以使用命令 **BGSAVE**，该命令在后台执行。

#### 实例

127.0.0.1:6379> BGSAVE

Background saving started

## Redis 安全

我们可以通过 redis 的配置文件设置密码参数，这样客户端连接到 redis 服务就需要密码验证，这样可以让你的 redis 服务更安全。

#### 实例

我们可以通过以下命令查看是否设置了密码验证：

127.0.0.1:6379> CONFIG get requirepass

1) "requirepass"

2) ""

默认情况下 requirepass 参数是空的，这就意味着你无需通过密码验证就可以连接到 redis 服务。

你可以通过以下命令来修改该参数：

127.0.0.1:6379> CONFIG set requirepass "runoob"

OK

127.0.0.1:6379> CONFIG get requirepass

1) "requirepass"

2) "runoob"

设置密码后，客户端连接 redis 服务就需要密码验证，否则无法执行命令。

#### 语法

**AUTH** 命令基本语法格式如下：

127.0.0.1:6379> AUTH password

#### 实例

127.0.0.1:6379> AUTH "runoob"

OK

127.0.0.1:6379> SET mykey "Test value"

OK

127.0.0.1:6379> GET mykey

"Test value"

## Redis 性能测试

Redis 性能测试是通过同时执行多个命令实现的。

#### 语法

redis 性能测试的基本命令如下：

redis-benchmark [option] [option value]

**注意**：该命令是在 redis 的目录下执行的，而不是 redis 客户端的内部指令。

#### 实例

以下实例同时执行 10000 个请求来检测性能：

$ redis-benchmark -n 10000 -q

PING\_INLINE: 141043.72 requests per second

PING\_BULK: 142857.14 requests per second

SET: 141442.72 requests per second

GET: 145348.83 requests per second

INCR: 137362.64 requests per second

LPUSH: 145348.83 requests per second

LPOP: 146198.83 requests per second

SADD: 146198.83 requests per second

SPOP: 149253.73 requests per second

LPUSH (needed to benchmark LRANGE): 148588.42 requests per second

LRANGE\_100 (first 100 elements): 58411.21 requests per second

LRANGE\_300 (first 300 elements): 21195.42 requests per second

LRANGE\_500 (first 450 elements): 14539.11 requests per second

LRANGE\_600 (first 600 elements): 10504.20 requests per second

MSET (10 keys): 93283.58 requests per second

redis 性能测试工具可选参数如下所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **选项** | **描述** | **默认值** |
| 1 | **-h** | 指定服务器主机名 | 127.0.0.1 |
| 2 | **-p** | 指定服务器端口 | 6379 |
| 3 | **-s** | 指定服务器 socket |  |
| 4 | **-c** | 指定并发连接数 | 50 |
| 5 | **-n** | 指定请求数 | 10000 |
| 6 | **-d** | 以字节的形式指定 SET/GET 值的数据大小 | 2 |
| 7 | **-k** | 1=keep alive 0=reconnect | 1 |
| 8 | **-r** | SET/GET/INCR 使用随机 key, SADD 使用随机值 |  |
| 9 | **-P** | 通过管道传输 <numreq> 请求 | 1 |
| 10 | **-q** | 强制退出 redis。仅显示 query/sec 值 |  |
| 11 | **--csv** | 以 CSV 格式输出 |  |
| 12 | **-l（L 的小写字母）** | 生成循环，永久执行测试 |  |
| 13 | **-t** | 仅运行以逗号分隔的测试命令列表。 |  |
| 14 | **-I（i 的大写字母）** | Idle 模式。仅打开 N 个 idle 连接并等待。 |  |

#### 实例

以下实例我们使用了多个参数来测试 redis 性能：

$ redis-benchmark -h 127.0.0.1 -p 6379 -t set,lpush -n 10000 -q

SET: 146198.83 requests per second

LPUSH: 145560.41 requests per second

以上实例中主机为 127.0.0.1，端口号为 6379，执行的命令为 set,lpush，请求数为 10000，通过 -q 参数让结果只显示每秒执行的请求数。

## Redis 客户端连接

Redis 通过监听一个 TCP 端口或者 Unix socket 的方式来接收来自客户端的连接，当一个连接建立后，Redis 内部会进行以下一些操作：

* 首先，客户端 socket 会被设置为非阻塞模式，因为 Redis 在网络事件处理上采用的是非阻塞多路复用模型。
* 然后为这个 socket 设置 TCP\_NODELAY 属性，禁用 Nagle 算法
* 然后创建一个可读的文件事件用于监听这个客户端 socket 的数据发送

### 最大连接数

在 Redis2.4 中，最大连接数是被直接硬编码在代码里面的，而在2.6版本中这个值变成可配置的。

maxclients 的默认值是 10000，你也可以在 redis.conf 中对这个值进行修改。

config get maxclients

1) "maxclients"

2) "10000"

**实例**

以下实例我们在服务启动时设置最大连接数为 100000：

redis-server --maxclients 100000

### 客户端命令

|  |  |  |
| --- | --- | --- |
| **S.N.** | **命令** | **描述** |
| 1 | **CLIENT LIST** | 返回连接到 redis 服务的客户端列表 |
| 2 | **CLIENT SETNAME** | 设置当前连接的名称 |
| 3 | **CLIENT GETNAME** | 获取通过 CLIENT SETNAME 命令设置的服务名称 |
| 4 | **CLIENT PAUSE** | 挂起客户端连接，指定挂起的时间以毫秒计 |
| 5 | **CLIENT KILL** | 关闭客户端连接 |

## Redis 管道技术

Redis是一种基于客户端-服务端模型以及请求/响应协议的TCP服务。这意味着通常情况下一个请求会遵循以下步骤：

* 客户端向服务端发送一个查询请求，并监听Socket返回，通常是以阻塞模式，等待服务端响应。
* 服务端处理命令，并将结果返回给客户端。

### Redis 管道技术

Redis 管道技术可以在服务端未响应时，客户端可以继续向服务端发送请求，并最终一次性读取所有服务端的响应。

**实例**

查看 redis 管道，只需要启动 redis 实例并输入以下命令：

$(echo -en "PING\r\n SET runoobkey redis\r\nGET runoobkey\r\nINCR visitor\r\nINCR visitor\r\nINCR visitor\r\n"; sleep 10) | nc localhost 6379

+PONG

+OK

redis

:1

:2

:3

以上实例中我们通过使用 **PING** 命令查看redis服务是否可用， 之后我们设置了 runoobkey 的值为 redis，然后我们获取 runoobkey 的值并使得 visitor 自增 3 次。

在返回的结果中我们可以看到这些命令一次性向 redis 服务提交，并最终一次性读取所有服务端的响应

### 管道技术的优势

管道技术最显著的优势是提高了 redis 服务的性能。

**一些测试数据**

在下面的测试中，我们将使用Redis的Ruby客户端，支持管道技术特性，测试管道技术对速度的提升效果。

require 'rubygems'

require 'redis'

def bench(descr)

start = Time.now

yield

puts "#{descr} #{Time.now-start} seconds"

end

def without\_pipelining

r = Redis.new

10000.times {

    r.ping

}

end

def with\_pipelining

r = Redis.new

r.pipelined {

    10000.times {

        r.ping

    }

}

end

bench("without pipelining") {

    without\_pipelining

}

bench("with pipelining") {

    with\_pipelining

}

从处于局域网中的Mac OS X系统上执行上面这个简单脚本的数据表明，开启了管道操作后，往返延时已经被改善得相当低了。

without pipelining 1.185238 seconds

with pipelining 0.250783 seconds

如你所见，开启管道后，我们的速度效率提升了5倍。

## Redis 分区

分区是分割数据到多个Redis实例的处理过程，因此每个实例只保存key的一个子集。

**分区的优势**

* 通过利用多台计算机内存的和值，允许我们构造更大的数据库。
* 通过多核和多台计算机，允许我们扩展计算能力；通过多台计算机和网络适配器，允许我们扩展网络带宽。

**分区的不足**

redis的一些特性在分区方面表现的不是很好：

* 涉及多个key的操作通常是不被支持的。举例来说，当两个set映射到不同的redis实例上时，你就不能对这两个set执行交集操作。
* 涉及多个key的redis事务不能使用。
* 当使用分区时，数据处理较为复杂，比如你需要处理多个rdb/aof文件，并且从多个实例和主机备份持久化文件。
* 增加或删除容量也比较复杂。redis集群大多数支持在运行时增加、删除节点的透明数据平衡的能力，但是类似于客户端分区、代理等其他系统则不支持这项特性。然而，一种叫做presharding的技术对此是有帮助的。

### 分区类型

Redis 有两种类型分区。 假设有4个Redis实例 R0，R1，R2，R3，和类似user:1，user:2这样的表示用户的多个key，对既定的key有多种不同方式来选择这个key存放在哪个实例中。也就是说，有不同的系统来映射某个key到某个Redis服务。

**范围分区**

最简单的分区方式是按范围分区，就是映射一定范围的对象到特定的Redis实例。

比如，ID从0到10000的用户会保存到实例R0，ID从10001到 20000的用户会保存到R1，以此类推。

这种方式是可行的，并且在实际中使用，不足就是要有一个区间范围到实例的映射表。这个表要被管理，同时还需要各 种对象的映射表，通常对Redis来说并非是好的方法。

**哈希分区**

另外一种分区方法是hash分区。这对任何key都适用，也无需是object\_name:这种形式，像下面描述的一样简单：

* 用一个hash函数将key转换为一个数字，比如使用crc32 hash函数。对key foobar执行crc32(foobar)会输出类似93024922的整数。
* 对这个整数取模，将其转化为0-3之间的数字，就可以将这个整数映射到4个Redis实例中的一个了。93024922 % 4 = 2，就是说key foobar应该被存到R2实例中。注意：取模操作是取除的余数，通常在多种编程语言中用%操作符实现。

## Java 使用 Redis

### 安装

开始在 Java 中使用 Redis 前， 我们需要确保已经安装了 redis 服务及 Java redis 驱动，且你的机器上能正常使用 Java。 Java的安装配置可以参考我们的 [Java 开发环境配置](https://www.runoob.com/redis/java-environment-setup.html)接下来让我们安装 Java redis 驱动：

* 首先你需要下载驱动包 [**下载 jedis.jar**](https://mvnrepository.com/artifact/redis.clients/jedis)，确保下载最新驱动包。
* 在你的 classpath 中包含该驱动包。

*本站提供了 2.9.0 jar 版本下载：*[*jedis-2.9.0.jar*](https://static.runoob.com/download/jedis-2.9.0.jar)

### 连接到 redis 服务

### 实例

import redis.clients.jedis.Jedis; public class RedisJava { public static void main(String[] args) { //连接本地的 Redis 服务 Jedis jedis = new Jedis("localhost"); // 如果 Redis 服务设置了密码，需要下面这行，没有就不需要 // jedis.auth("123456"); System.out.println("连接成功"); //查看服务是否运行 System.out.println("服务正在运行: "+jedis.ping()); } }

编译以上 Java 程序，确保驱动包的路径是正确的。

连接成功

服务正在运行: PONG

### Redis Java String(字符串) 实例

### 实例

import redis.clients.jedis.Jedis;

public class RedisStringJava {

public static void main(String[] args) {

//连接本地的 Redis 服务

Jedis jedis = new Jedis("localhost");

System.out.println("连接成功");

//设置 redis 字符串数据

jedis.set("runoobkey", "www.runoob.com");

// 获取存储的数据并输出

System.out.println("redis 存储的字符串为: "+ jedis.get("runoobkey"));

}

}

编译以上程序。

连接成功

redis 存储的字符串为: www.runoob.com

### Redis Java List(列表) 实例

### 实例

import java.util.List;

import redis.clients.jedis.Jedis;

public class RedisListJava {

public static void main(String[] args) {

//连接本地的 Redis 服务

Jedis jedis = new Jedis("localhost");

System.out.println("连接成功");

//存储数据到列表中

jedis.lpush("site-list", "Runoob");

jedis.lpush("site-list", "Google");

jedis.lpush("site-list", "Taobao");

// 获取存储的数据并输出

List<String> list = jedis.lrange("site-list", 0 ,2);

for(int i=0; i<list.size(); i++) {

System.out.println("列表项为: "+list.get(i));

}

}

}

编译以上程序。

连接成功

列表项为: Taobao

列表项为: Google

列表项为: Runoob

### Redis Java Keys 实例

### 实例

import java.util.Iterator;

import java.util.Set;

import redis.clients.jedis.Jedis;

public class RedisKeyJava {

public static void main(String[] args) {

//连接本地的 Redis 服务

Jedis jedis = new Jedis("localhost");

System.out.println("连接成功");

// 获取数据并输出

Set<String> keys = jedis.keys("\*");

Iterator<String> it=keys.iterator() ;

while(it.hasNext()){

String key = it.next();

System.out.println(key);

}

}

}

编译以上程序。

连接成功

runoobkey

site-list

## PHP 使用 Redis

### 安装

开始在 PHP 中使用 Redis 前， 我们需要确保已经安装了 redis 服务及 PHP redis 驱动，且你的机器上能正常使用 PHP。 接下来让我们安装 PHP redis 驱动：下载地址为:[**https://github.com/phpredis/phpredis/releases**](https://github.com/phpredis/phpredis/releases)。

#### PHP安装redis扩展

以下操作需要在下载的 phpredis 目录中完成：

$ wget https://github.com/phpredis/phpredis/archive/3.1.4.tar.gz

$ tar zxvf 3.1.4.tar.gz # 解压

$ cd phpredis-3.1.4 # 进入 phpredis 目录

$ /usr/local/php/bin/phpize # php安装后的路径

$ ./configure --with-php-config=/usr/local/php/bin/php-config

$ make && make install

#### 修改php.ini文件

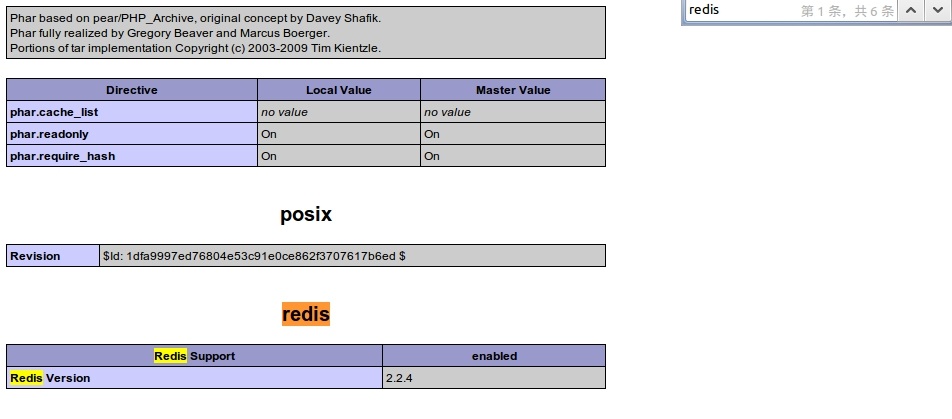
vi /usr/local/php/lib/php.ini

增加如下内容:

extension\_dir = "/usr/local/php/lib/php/extensions/no-debug-zts-20090626"

extension=redis.so

安装完成后重启php-fpm 或 apache。查看phpinfo信息，就能看到redis扩展。



### 连接到 redis 服务

<?php

//连接本地的 Redis 服务

$redis = new Redis();

$redis->connect('127.0.0.1', 6379);

echo "Connection to server successfully";

//查看服务是否运行

echo "Server is running: " . $redis->ping();

?>

执行脚本，输出结果为：

Connection to server sucessfully

Server is running: PONG

### Redis PHP String(字符串) 实例

<?php

//连接本地的 Redis 服务

$redis = new Redis();

$redis->connect('127.0.0.1', 6379);

echo "Connection to server successfully";

//设置 redis 字符串数据

$redis->set("tutorial-name", "Redis tutorial");

// 获取存储的数据并输出

echo "Stored string in redis:: " . $redis->get("tutorial-name");

?>

执行脚本，输出结果为：

Connection to server sucessfully

Stored string in redis:: Redis tutorial

### Redis PHP List(列表) 实例

<?php

//连接本地的 Redis 服务

$redis = new Redis();

$redis->connect('127.0.0.1', 6379);

echo "Connection to server successfully";

//存储数据到列表中

$redis->lpush("tutorial-list", "Redis");

$redis->lpush("tutorial-list", "Mongodb");

$redis->lpush("tutorial-list", "Mysql");

// 获取存储的数据并输出

$arList = $redis->lrange("tutorial-list", 0 ,5);

echo "Stored string in redis";

print\_r($arList);

?>

执行脚本，输出结果为：

Connection to server sucessfully

Stored string in redis

Mysql

Mongodb

Redis

### Redis PHP Keys 实例

<?php

//连接本地的 Redis 服务

$redis = new Redis();

$redis->connect('127.0.0.1', 6379);

echo "Connection to server successfully";

// 获取数据并输出

$arList = $redis->keys("\*");

echo "Stored keys in redis:: ";

print\_r($arList);

?>

执行脚本，输出结果为：

Connection to server sucessfully

Stored string in redis::

tutorial-name

tutorial-list

# Memcached

## Linux Memcached 安装

Memcached 支持许多平台：Linux、FreeBSD、Solaris、Mac OS，也可以安装在Windows上。

Linux系统安装memcached，首先要先安装libevent库。

sudo apt-get install libevent ibevent-dev 自动下载安装（Ubuntu/Debian）

yum install libevent libevent-devel 自动下载安装（Redhat/Fedora/Centos）

### 安装 Memcached

#### 自动安装

**Ubuntu/Debian**

sudo apt-get install memcached

**Redhat/Fedora/Centos**

yum install memcached

**FreeBSD**

portmaster databases/memcached

安装完后可以使用 whereis 查看命令的路径：

whereis memcached

发现安装位置在 **/usr/bin/memcached**。

#### 源代码安装

从其官方网站（http://memcached.org）下载memcached最新版本。

wget http://memcached.org/latest 下载最新版本

tar -zxvf memcached-1.x.x.tar.gz 解压源码

cd memcached-1.x.x 进入目录

./configure --prefix=/usr/local/memcached 配置

make && make test 编译

sudo make install 安装

### Memcached 运行

Memcached命令的运行：

$ /usr/local/memcached/bin/memcached -h 命令帮助

注意：如果使用自动安装 memcached 命令位于 **/usr/local/bin/memcached**。

**启动选项：**

* -d是启动一个守护进程；
* -m是分配给Memcache使用的内存数量，单位是MB；
* -u是运行Memcache的用户；
* -l是监听的服务器IP地址，可以有多个地址；
* -p是设置Memcache监听的端口，，最好是1024以上的端口；
* -c是最大运行的并发连接数，默认是1024；
* -P是设置保存Memcache的pid文件。

#### （1）作为前台程序运行：

从终端输入以下命令，启动memcached:

/usr/local/memcached/bin/memcached -p 11211 -m 64m -vv

slab class 1: chunk size 88 perslab 11915

slab class 2: chunk size 112 perslab 9362

slab class 3: chunk size 144 perslab 7281

中间省略

slab class 38: chunk size 391224 perslab 2

slab class 39: chunk size 489032 perslab 2

<23 server listening

<24 send buffer was 110592, now 268435456

<24 server listening (udp)

<24 server listening (udp)

<24 server listening (udp)

<24 server listening (udp)

这里显示了调试信息。这样就在前台启动了memcached，监听TCP端口11211，最大内存使用量为64M。调试信息的内容大部分是关于存储的信息。

#### （2）作为后台服务程序运行：

# /usr/local/memcached/bin/memcached -p 11211 -m 64m -d

或者

/usr/local/memcached/bin/memcached -d -m 64M -u root -l 192.168.0.200 -p 11211 -c 256 -P /tmp/memcached.pid

[Memcached 教程](https://www.runoob.com/memcached/memcached-tutorial.html)

[Window Memcached 安装](https://www.runoob.com/memcached/window-install-memcached.html)

### 1 篇笔记 写笔记

**Memcached 的启动和停止**

查找允许 memcached 的进程 id:

# ps -ef|grep memcached

root 9662 1 0 10:13 ? 00:00:00 /usr/bin/memcached -d -m 1024 -u root -l 127.0.0.1 -p 11211 -c 1024 -P /tmp/memcached.pid

root 10155 8703 0 10:14 pts/0 00:00:00 grep memcached

第一行的第二个就是进程 id，使用 kill 命令停止进程：

# kill 9662

# ps -ef|grep memcached

root 10298 8703 0 10:14 pts/0 00:00:00 grep memcached

可以看到 memcached 服务已经消失。

我们可以使用以下命令来启动：

# /usr/bin/memcached -d -m 1024 -u root -l 127.0.0.1 -p 11211 -c 1024 -P /tmp/memcached.pid

## Windows 下安装 Memcached

官网上并未提供 Memcached 的 Windows 平台安装包，我们可以使用以下链接来下载，你需要根据自己的系统平台及需要的版本号点击对应的链接下载即可：

* 32位系统 1.2.5版本：<http://static.runoob.com/download/memcached-1.2.5-win32-bin.zip>
* 32位系统 1.2.6版本：<http://static.runoob.com/download/memcached-1.2.6-win32-bin.zip>
* 32位系统 1.4.4版本：<http://static.runoob.com/download/memcached-win32-1.4.4-14.zip>
* 64位系统 1.4.4版本：<http://static.runoob.com/download/memcached-win64-1.4.4-14.zip>
* 32位系统 1.4.5版本：<http://static.runoob.com/download/memcached-1.4.5-x86.zip>
* 64位系统 1.4.5版本：<http://static.runoob.com/download/memcached-1.4.5-amd64.zip>

在 1.4.5 版本以前 memcached 可以作为一个服务安装，而在 1.4.5 及之后的版本删除了该功能。因此我们以下介绍两个不同版本 1.4.4 及 1.4.5的不同安装方法：

### memcached <1.4.5 版本安装

1、解压下载的安装包到指定目录。

2、在 1.4.5 版本以前 memcached 可以作为一个服务安装，使用管理员权限运行以下命令：

c:\memcached\memcached.exe -d install

**注意：**你需要使用真实的路径替代 c:\memcached\memcached.exe。

3、然后我们可以使用以下命令来启动和关闭 memcached 服务：

c:\memcached\memcached.exe -d start

c:\memcached\memcached.exe -d stop

4、如果要修改 memcached 的配置项, 可以在命令行中执行 regedit.exe 命令打开注册表并找到 "HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\memcached" 来进行修改。

如果要提供 memcached 使用的缓存配置 可以修改 ImagePath 为:

"c:\memcached\memcached.exe" -d runservice -m 512

**-m 512** 意思是设置 memcached 最大的缓存配置为512M。

此外我们还可以通过使用 "c:\memcached\memcached.exe -h" 命令查看更多的参数配置。

5、如果我们需要卸载 memcached ，可以使用以下命令：

c:\memcached\memcached.exe -d uninstall

### memcached >= 1.4.5 版本安装

1、解压下载的安装包到指定目录。

2、在 memcached1.4.5 版本之后，memcached 不能作为服务来运行，需要使用任务计划中来开启一个普通的进程，在 window 启动时设置 memcached自动执行。

我们使用管理员身份执行以下命令将 memcached 添加来任务计划表中：

schtasks /create /sc onstart /tn memcached /tr "'c:\memcached\memcached.exe' -m 512"

**注意：**你需要使用真实的路径替代 c:\memcached\memcached.exe。

**注意：-m 512** 意思是设置 memcached 最大的缓存配置为512M。

**注意：**我们可以通过使用 "c:\memcached\memcached.exe -h" 命令查看更多的参数配置。

3、如果需要删除 memcached 的任务计划可以执行以下命令：

schtasks /delete /tn memcached

## Memcached 连接

我们可以通过 telnet 命令并指定主机ip和端口来连接 Memcached 服务。

#### 语法

telnet HOST PORT

命令中的 **HOST** 和 **PORT** 为运行 Memcached 服务的 IP 和 端口。

#### 实例

以下实例演示了如何连接到 Memcached 服务并执行简单的 set 和 get 命令。

本实例的 Memcached 服务运行的主机为 127.0.0.1（本机） 、端口为 11211。

telnet 127.0.0.1 11211

Trying 127.0.0.1...

Connected to 127.0.0.1.

Escape character is '^]'.

set foo 0 0 3 保存命令

bar 数据

STORED 结果

get foo 取得命令

VALUE foo 0 3 数据

bar 数据

END 结束行

quit 退出

## Memcached set 命令

Memcached set 命令用于将 **value(数据值)** 存储在指定的 **key(键)** 中。

如果set的key已经存在，该命令可以更新该key所对应的原来的数据，也就是实现更新的作用。

**语法：**

set 命令的基本语法格式如下：

set key flags exptime bytes [noreply]

value

参数说明如下：

* **key：**键值 key-value 结构中的 key，用于查找缓存值。
* **flags**：可以包括键值对的整型参数，客户机使用它存储关于键值对的额外信息 。
* **exptime**：在缓存中保存键值对的时间长度（以秒为单位，0 表示永远）
* **bytes**：在缓存中存储的字节数
* **noreply（可选）**： 该参数告知服务器不需要返回数据
* **value**：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）

**实例**

以下实例中我们设置：

* key → runoob
* flag → 0
* exptime → 900 (以秒为单位)
* bytes → 9 (数据存储的字节数)
* value → memcached

set runoob 0 900 9

memcached

STORED

get runoob

VALUE runoob 0 9

memcached

END

**输出**

如果数据设置成功，则输出：

STORED

输出信息说明：

* **STORED**：保存成功后输出。
* **ERROR**：在保存失败后输出。

## Memcached add 命令

Memcached add 命令用于将 **value(数据值)** 存储在指定的 **key(键)** 中。

如果 add 的 key 已经存在，则不会更新数据(过期的 key 会更新)，之前的值将仍然保持相同，并且您将获得响应 **NOT\_STORED**。

**语法：**

add 命令的基本语法格式如下：

add key flags exptime bytes [noreply]

value

参数说明如下：

* **key：**键值 key-value 结构中的 key，用于查找缓存值。
* **flags**：可以包括键值对的整型参数，客户机使用它存储关于键值对的额外信息 。
* **exptime**：在缓存中保存键值对的时间长度（以秒为单位，0 表示永远）
* **bytes**：在缓存中存储的字节数
* **noreply（可选）**： 该参数告知服务器不需要返回数据
* **value**：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）

**实例**

以下实例中我们设置：

* key → new\_key
* flag → 0
* exptime → 900 (以秒为单位)
* bytes → 10 (数据存储的字节数)
* value → data\_value

add new\_key 0 900 10

data\_value

STORED

get new\_key

VALUE new\_key 0 10

data\_value

END

**输出**

如果数据添加成功，则输出：

STORED

输出信息说明：

* **STORED**：保存成功后输出。
* **NOT\_STORED**：在保存失败后输出。

## Memcached replace 命令

Memcached replace 命令用于替换已存在的 **key(键)** 的 **value(数据值)**。

如果 key 不存在，则替换失败，并且您将获得响应 **NOT\_STORED**。

**语法：**

replace 命令的基本语法格式如下：

replace key flags exptime bytes [noreply]

value

参数说明如下：

* **key：**键值 key-value 结构中的 key，用于查找缓存值。
* **flags**：可以包括键值对的整型参数，客户机使用它存储关于键值对的额外信息 。
* **exptime**：在缓存中保存键值对的时间长度（以秒为单位，0 表示永远）
* **bytes**：在缓存中存储的字节数
* **noreply（可选）**： 该参数告知服务器不需要返回数据
* **value**：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）

**实例**

以下实例中我们设置：

* key → mykey
* flag → 0
* exptime → 900 (以秒为单位)
* bytes → 10 (数据存储的字节数)
* value → data\_value

以下实例中我们使用的键位 'mykey' 并存储对应的值 data\_value。执行后我们替换相同的 key 的值为 'some\_other\_value'。

add mykey 0 900 10

data\_value

STORED

get mykey

VALUE mykey 0 10

data\_value

END

replace mykey 0 900 16

some\_other\_value

get mykey

VALUE mykey 0 16

some\_other\_value

END

**输出**

如果数据添加成功，则输出：

STORED

输出信息说明：

* **STORED**：保存成功后输出。
* **NOT\_STORED**：执行替换失败后输出。

## Memcached append 命令

Memcached append 命令用于向已存在 **key(键)** 的 **value(数据值)** 后面追加数据 。

**语法：**

append 命令的基本语法格式如下：

append key flags exptime bytes [noreply]

value

参数说明如下：

* **key：**键值 key-value 结构中的 key，用于查找缓存值。
* **flags**：可以包括键值对的整型参数，客户机使用它存储关于键值对的额外信息 。
* **exptime**：在缓存中保存键值对的时间长度（以秒为单位，0 表示永远）
* **bytes**：在缓存中存储的字节数
* **noreply（可选）**： 该参数告知服务器不需要返回数据
* **value**：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）

**实例**

实例如下：

* 首先我们在 Memcached 中存储一个键 runoob，其值为 memcached。
* 然后，我们使用 get 命令检索该值。
* 然后，我们使用 **append** 命令在键为 runoob 的值后面追加 "redis"。
* 最后，我们再使用 get 命令检索该值。

set runoob 0 900 9

memcached

STORED

get runoob

VALUE runoob 0 9

memcached

END

append runoob 0 900 5

redis

STORED

get runoob

VALUE runoob 0 14

memcachedredis

END

**输出**

如果数据添加成功，则输出：

STORED

输出信息说明：

* **STORED**：保存成功后输出。
* **NOT\_STORED**：该键在 Memcached 上不存在。
* **CLIENT\_ERROR**：执行错误。

## Memcached prepend 命令

Memcached prepend 命令用于向已存在 **key(键)** 的 **value(数据值)** 前面追加数据 。

**语法：**

prepend 命令的基本语法格式如下：

prepend key flags exptime bytes [noreply]

value

参数说明如下：

* **key：**键值 key-value 结构中的 key，用于查找缓存值。
* **flags**：可以包括键值对的整型参数，客户机使用它存储关于键值对的额外信息 。
* **exptime**：在缓存中保存键值对的时间长度（以秒为单位，0 表示永远）
* **bytes**：在缓存中存储的字节数
* **noreply（可选）**： 该参数告知服务器不需要返回数据
* **value**：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）

**实例**

实例如下：

* 首先我们在 Memcached 中存储一个键 runoob，其值为 memcached。
* 然后，我们使用 get 命令检索该值。
* 然后，我们使用 **prepend** 命令在键为 runoob 的值前面追加 "redis"。
* 最后，我们再使用 get 命令检索该值。

set runoob 0 900 9

memcached

STORED

get runoob

VALUE runoob 0 9

memcached

END

prepend runoob 0 900 5

redis

STORED

get runoob

VALUE runoob 0 14

redismemcached

END

**输出**

如果数据添加成功，则输出：

STORED

输出信息说明：

* **STORED**：保存成功后输出。
* **NOT\_STORED**：该键在 Memcached 上不存在。
* **CLIENT\_ERROR**：执行错误。

## Memcached CAS 命令

Memcached CAS（Check-And-Set 或 Compare-And-Swap） 命令用于执行一个"检查并设置"的操作

它仅在当前客户端最后一次取值后，该key 对应的值没有被其他客户端修改的情况下， 才能够将值写入。

检查是通过cas\_token参数进行的， 这个参数是Memcach指定给已经存在的元素的一个唯一的64位值。

**语法：**

CAS 命令的基本语法格式如下：

cas key flags exptime bytes unique\_cas\_token [noreply]

value

参数说明如下：

* **key：**键值 key-value 结构中的 key，用于查找缓存值。
* **flags**：可以包括键值对的整型参数，客户机使用它存储关于键值对的额外信息 。
* **exptime**：在缓存中保存键值对的时间长度（以秒为单位，0 表示永远）
* **bytes**：在缓存中存储的字节数
* **unique\_cas\_token**通过 gets 命令获取的一个唯一的64位值。
* **noreply（可选）**： 该参数告知服务器不需要返回数据
* **value**：存储的值（始终位于第二行）（可直接理解为key-value结构中的value）

**实例**

要在 Memcached 上使用 CAS 命令，你需要从 Memcached 服务商通过 gets 命令获取令牌（token）。

gets 命令的功能类似于基本的 get 命令。两个命令之间的差异在于，gets 返回的信息稍微多一些：64 位的整型值非常像名称/值对的 "版本" 标识符。

实例步骤如下：

* 如果没有设置唯一令牌，则 CAS 命令执行错误。
* 如果键 key 不存在，执行失败。
* 添加键值对。
* 通过 gets 命令获取唯一令牌。
* 使用 cas 命令更新数据
* 使用 get 命令查看数据是否更新

cas tp 0 900 9

ERROR <− 缺少 token

cas tp 0 900 9 2

memcached

NOT\_FOUND <− 键 tp 不存在

set tp 0 900 9

memcached

STORED

gets tp

VALUE tp 0 9 1

memcached

END

cas tp 0 900 5 1

redis

STORED

get tp

VALUE tp 0 5

redis

END

**输出**

如果数据添加成功，则输出：

STORED

输出信息说明：

* **STORED**：保存成功后输出。
* **ERROR**：保存出错或语法错误。
* **EXISTS**：在最后一次取值后另外一个用户也在更新该数据。
* **NOT\_FOUND**：Memcached 服务上不存在该键值。

## Memcached get 命令

Memcached get 命令获取存储在 **key(键)** 中的 **value(数据值)** ，如果 key 不存在，则返回空。

**语法：**

get 命令的基本语法格式如下：

get key

多个 key 使用空格隔开，如下:

get key1 key2 key3

参数说明如下：

* **key：**键值 key-value 结构中的 key，用于查找缓存值。

**实例**

在以下实例中，我们使用 runoob 作为 key，过期时间设置为 900 秒。

set runoob 0 900 9

memcached

STORED

get runoob

VALUE runoob 0 9

memcached

END

## Memcached gets 命令

Memcached gets 命令获取带有 CAS 令牌存 的 **value(数据值)** ，如果 key 不存在，则返回空。

**语法：**

gets 命令的基本语法格式如下：

gets key

多个 key 使用空格隔开，如下:

gets key1 key2 key3

参数说明如下：

* **key：**键值 key-value 结构中的 key，用于查找缓存值。

**实例**

在以下实例中，我们使用 runoob 作为 key，过期时间设置为 900 秒。

set runoob 0 900 9

memcached

STORED

gets runoob

VALUE runoob 0 9 1

memcached

END

在 使用 gets 命令的输出结果中，在最后一列的数字 1 代表了 key 为 runoob 的 CAS 令牌。

## Memcached delete 命令

Memcached delete 命令用于删除已存在的 key(键)。

**语法：**

delete 命令的基本语法格式如下：

delete key [noreply]

参数说明如下：

* **key：**键值 key-value 结构中的 key，用于查找缓存值。
* **noreply（可选）**： 该参数告知服务器不需要返回数据

**实例**

在以下实例中，我们使用 runoob 作为 key，过期时间设置为 900 秒。之后我们使用 delete 命令删除该 key。

set runoob 0 900 9

memcached

STORED

get runoob

VALUE runoob 0 9

memcached

END

delete runoob

DELETED

get runoob

END

delete runoob

NOT\_FOUND

**输出**

输出信息说明：

* **DELETED**：删除成功。
* **ERROR**：语法错误或删除失败。
* **NOT\_FOUND**：key 不存在。

## Memcached incr 与 decr 命令

Memcached incr 与 decr 命令用于对已存在的 key(键) 的数字值进行自增或自减操作。

incr 与 decr 命令操作的数据必须是十进制的32位无符号整数。

如果 key 不存在返回 **NOT\_FOUND**，如果键的值不为数字，则返回 **CLIENT\_ERROR**，其他错误返回 **ERROR**。

### incr 命令

**语法：**

incr 命令的基本语法格式如下：

incr key increment\_value

参数说明如下：

* **key：**键值 key-value 结构中的 key，用于查找缓存值。
* **increment\_value**： 增加的数值。

**实例**

在以下实例中，我们使用 visitors 作为 key，初始值为 10，之后进行加 5 操作。

set visitors 0 900 2

10

STORED

get visitors

VALUE visitors 0 2

10

END

incr visitors 5

15

get visitors

VALUE visitors 0 2

15

END

**输出**

输出信息说明：

* **NOT\_FOUND**：key 不存在。
* **CLIENT\_ERROR**：自增值不是对象。
* **ERROR**其他错误，如语法错误等。

### decr 命令

decr 命令的基本语法格式如下：

decr key decrement\_value

参数说明如下：

* **key：**键值 key-value 结构中的 key，用于查找缓存值。
* **decrement\_value**： 减少的数值。

**实例**

set visitors 0 900 2

10

STORED

get visitors

VALUE visitors 0 2

10

END

decr visitors 5

5

get visitors

VALUE visitors 0 1

5

END

在以下实例中，我们使用 visitors 作为 key，初始值为 10，之后进行减 5 操作。

**输出**

输出信息说明：

* **NOT\_FOUND**：key 不存在。
* **CLIENT\_ERROR**：自增值不是对象。
* **ERROR**其他错误，如语法错误等。

## Memcached stats 命令

Memcached stats 命令用于返回统计信息例如 PID(进程号)、版本号、连接数等。

**语法：**

stats 命令的基本语法格式如下：

stats

**实例**

在以下实例中，我们使用了 stats 命令来输出 Memcached 服务信息。

stats

STAT pid 1162

STAT uptime 5022

STAT time 1415208270

STAT version 1.4.14

STAT libevent 2.0.19-stable

STAT pointer\_size 64

STAT rusage\_user 0.096006

STAT rusage\_system 0.152009

STAT curr\_connections 5

STAT total\_connections 6

STAT connection\_structures 6

STAT reserved\_fds 20

STAT cmd\_get 6

STAT cmd\_set 4

STAT cmd\_flush 0

STAT cmd\_touch 0

STAT get\_hits 4

STAT get\_misses 2

STAT delete\_misses 1

STAT delete\_hits 1

STAT incr\_misses 2

STAT incr\_hits 1

STAT decr\_misses 0

STAT decr\_hits 1

STAT cas\_misses 0

STAT cas\_hits 0

STAT cas\_badval 0

STAT touch\_hits 0

STAT touch\_misses 0

STAT auth\_cmds 0

STAT auth\_errors 0

STAT bytes\_read 262

STAT bytes\_written 313

STAT limit\_maxbytes 67108864

STAT accepting\_conns 1

STAT listen\_disabled\_num 0

STAT threads 4

STAT conn\_yields 0

STAT hash\_power\_level 16

STAT hash\_bytes 524288

STAT hash\_is\_expanding 0

STAT expired\_unfetched 1

STAT evicted\_unfetched 0

STAT bytes 142

STAT curr\_items 2

STAT total\_items 6

STAT evictions 0

STAT reclaimed 1

END

这里显示了很多状态信息，下边详细解释每个状态项：

* **pid**： memcache服务器进程ID
* **uptime**：服务器已运行秒数
* **time**：服务器当前Unix时间戳
* **version**：memcache版本
* **pointer\_size**：操作系统指针大小
* **rusage\_user**：进程累计用户时间
* **rusage\_system**：进程累计系统时间
* **curr\_connections**：当前连接数量
* **total\_connections**：Memcached运行以来连接总数
* **connection\_structures**：Memcached分配的连接结构数量
* **cmd\_get**：get命令请求次数
* **cmd\_set**：set命令请求次数
* **cmd\_flush**：flush命令请求次数
* **get\_hits**：get命令命中次数
* **get\_misses**：get命令未命中次数
* **delete\_misses**：delete命令未命中次数
* **delete\_hits**：delete命令命中次数
* **incr\_misses**：incr命令未命中次数
* **incr\_hits**：incr命令命中次数
* **decr\_misses**：decr命令未命中次数
* **decr\_hits**：decr命令命中次数
* **cas\_misses**：cas命令未命中次数
* **cas\_hits**：cas命令命中次数
* **cas\_badval**：使用擦拭次数
* **auth\_cmds**：认证命令处理的次数
* **auth\_errors**：认证失败数目
* **bytes\_read**：读取总字节数
* **bytes\_written**：发送总字节数
* **limit\_maxbytes**：分配的内存总大小（字节）
* **accepting\_conns**：服务器是否达到过最大连接（0/1）
* **listen\_disabled\_num**：失效的监听数
* **threads**：当前线程数
* **conn\_yields**：连接操作主动放弃数目
* **bytes**：当前存储占用的字节数
* **curr\_items**：当前存储的数据总数
* **total\_items**：启动以来存储的数据总数
* **evictions**：LRU释放的对象数目
* **reclaimed**：已过期的数据条目来存储新数据的数目

## Memcached stats items 命令

Memcached stats items 命令用于显示各个 slab 中 item 的数目和存储时长(最后一次访问距离现在的秒数)。

### 语法：

stats items 命令的基本语法格式如下：

stats items

### 实例

stats items

STAT items:1:number 1

STAT items:1:age 7

STAT items:1:evicted 0

STAT items:1:evicted\_nonzero 0

STAT items:1:evicted\_time 0

STAT items:1:outofmemory 0

STAT items:1:tailrepairs 0

STAT items:1:reclaimed 0

STAT items:1:expired\_unfetched 0

STAT items:1:evicted\_unfetched 0

END

## Memcached stats slabs 命令

Memcached stats slabs 命令用于显示各个slab的信息，包括chunk的大小、数目、使用情况等。

### 语法：

stats slabs 命令的基本语法格式如下：

stats slabs

### 实例

stats slabs

STAT 1:chunk\_size 96

STAT 1:chunks\_per\_page 10922

STAT 1:total\_pages 1

STAT 1:total\_chunks 10922

STAT 1:used\_chunks 1

STAT 1:free\_chunks 10921

STAT 1:free\_chunks\_end 0

STAT 1:mem\_requested 71

STAT 1:get\_hits 0

STAT 1:cmd\_set 1

STAT 1:delete\_hits 0

STAT 1:incr\_hits 0

STAT 1:decr\_hits 0

STAT 1:cas\_hits 0

STAT 1:cas\_badval 0

STAT 1:touch\_hits 0

STAT active\_slabs 1

STAT total\_malloced 1048512

END

## Memcached stats sizes 命令

Memcached stats sizes 命令用于显示所有item的大小和个数。

该信息返回两列，第一列是 item 的大小，第二列是 item 的个数。

### 语法：

stats sizes 命令的基本语法格式如下：

stats sizes

### 实例

stats sizes

STAT 96 1

END

## Memcached flush\_all 命令

Memcached flush\_all 命令用于清理缓存中的所有 **key=>value(键=>值)** 对。

该命令提供了一个可选参数 **time**，用于在制定的时间后执行清理缓存操作。

### 语法：

flush\_all 命令的基本语法格式如下：

flush\_all [time] [noreply]

### 实例

清理缓存:

set runoob 0 900 9

memcached

STORED

get runoob

VALUE runoob 0 9

memcached

END

flush\_all

OK

get runoob

END

## Java 连接 Memcached 服务

使用 Java 程序连接 Memcached，需要在你的 classpath 中添加 Memcached jar 包。

本站 jar 包下载地址：[spymemcached-2.10.3.jar](https://www.runoob.com/try/download/spymemcached-2.10.3.jar)。

Google Code jar 包下载地址：[spymemcached-2.10.3.jar](http://code.google.com/p/spymemcached/downloads/list)（需要科学上网）。

以下程序假定 Memcached 服务的主机为 127.0.0.1，端口为 11211。

### 连接实例

Java 连接 Memcached

### MemcachedJava.java 文件：

import net.spy.memcached.MemcachedClient; import java.net.\*; public class MemcachedJava { public static void main(String[] args) { try{ // 本地连接 Memcached 服务 MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211)); System.out.println("Connection to server successful."); // 关闭连接 mcc.shutdown(); }catch(Exception ex){ System.out.println( ex.getMessage() ); } } }

该程序中我们使用 InetSocketAddress 连接 IP 为 127.0.0.1 端口 为 11211 的 memcached 服务。

执行以上代码，如果连接成功会输出以下信息：

Connection to server successful.

#### set 操作实例

以下使用 java.util.concurrent.Future 来存储数据

### MemcachedJava.java 文件：

import java.net.InetSocketAddress; import java.util.concurrent.Future; import net.spy.memcached.MemcachedClient; public class MemcachedJava { public static void main(String[] args) { try{ // 连接本地的 Memcached 服务 MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211)); System.out.println("Connection to server successful."); // 存储数据 Future fo = mcc.set("runoob", 900, "Free Education"); // 查看存储状态 System.out.println("set status:" + fo.get()); // 输出值 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 关闭连接 mcc.shutdown(); }catch(Exception ex){ System.out.println( ex.getMessage() ); } } }

执行程序，输出结果为：

Connection to server successful.

set status:true

runoob value in cache - Free Education

#### add 操作实例

### MemcachedJava.java 文件：

import java.net.InetSocketAddress; import java.util.concurrent.Future; import net.spy.memcached.MemcachedClient; public class MemcachedJava { public static void main(String[] args) { try{ // 连接本地的 Memcached 服务 MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211)); System.out.println("Connection to server successful."); // 添加数据 Future fo = mcc.set("runoob", 900, "Free Education"); // 打印状态 System.out.println("set status:" + fo.get()); // 输出 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 添加 fo = mcc.add("runoob", 900, "memcached"); // 打印状态 System.out.println("add status:" + fo.get()); // 添加新key fo = mcc.add("runoob", 900, "All Free Compilers"); // 打印状态 System.out.println("add status:" + fo.get()); // 输出 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 关闭连接 mcc.shutdown(); }catch(Exception ex){ System.out.println(ex.getMessage()); } } }

#### replace 操作实例

### MemcachedJava.java 文件：

import java.net.InetSocketAddress; import java.util.concurrent.Future; import net.spy.memcached.MemcachedClient; public class MemcachedJava { public static void main(String[] args) { try { //连接本地的 Memcached 服务 MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211)); System.out.println("Connection to server successful."); // 添加第一个 key=》value 对 Future fo = mcc.set("runoob", 900, "Free Education"); // 输出执行 add 方法后的状态 System.out.println("add status:" + fo.get()); // 获取键对应的值 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 添加新的 key fo = mcc.replace("runoob", 900, "Largest Tutorials' Library"); // 输出执行 set 方法后的状态 System.out.println("replace status:" + fo.get()); // 获取键对应的值 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 关闭连接 mcc.shutdown(); }catch(Exception ex){ System.out.println( ex.getMessage() ); } } }

#### append 操作实例

### MemcachedJava.java 文件：

import java.net.InetSocketAddress; import java.util.concurrent.Future; import net.spy.memcached.MemcachedClient; public class MemcachedJava { public static void main(String[] args) { try{ // 连接本地的 Memcached 服务 MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211)); System.out.println("Connection to server successful."); // 添加数据 Future fo = mcc.set("runoob", 900, "Free Education"); // 输出执行 set 方法后的状态 System.out.println("set status:" + fo.get()); // 获取键对应的值 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 对存在的key进行数据添加操作 fo = mcc.append("runoob", 900, " for All"); // 输出执行 set 方法后的状态 System.out.println("append status:" + fo.get()); // 获取键对应的值 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 关闭连接 mcc.shutdown(); }catch(Exception ex) { System.out.println(ex.getMessage()); ] } }

#### prepend 操作实例

### MemcachedJava.java 文件：

import java.net.InetSocketAddress; import java.util.concurrent.Future; import net.spy.memcached.MemcachedClient; public class MemcachedJava { public static void main(String[] args) { try{ // 连接本地的 Memcached 服务 MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211)); System.out.println("Connection to server successful."); // 添加数据 Future fo = mcc.set("runoob", 900, "Education for All"); // 输出执行 set 方法后的状态 System.out.println("set status:" + fo.get()); // 获取键对应的值 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 对存在的key进行数据添加操作 fo = mcc.prepend("runoob", 900, "Free "); // 输出执行 set 方法后的状态 System.out.println("prepend status:" + fo.get()); // 获取键对应的值 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 关闭连接 mcc.shutdown(); }catch(Exception ex) { System.out.println(ex.getMessage()); } } }

#### CAS 操作实例

### MemcachedJava.java 文件：

import java.net.InetSocketAddress; import java.util.concurrent.Future; import net.spy.memcached.CASValue; import net.spy.memcached.CASResponse; import net.spy.memcached.MemcachedClient; public class MemcachedJava { public static void main(String[] args) { try{ // 连接本地的 Memcached 服务 MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211)); System.out.println("Connection to server successful."); // 添加数据 Future fo = mcc.set("runoob", 900, "Free Education"); // 输出执行 set 方法后的状态 System.out.println("set status:" + fo.get()); // 使用 get 方法获取数据 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 通过 gets 方法获取 CAS token（令牌） CASValue casValue = mcc.gets("runoob"); // 输出 CAS token（令牌） 值 System.out.println("CAS token - " + casValue); // 尝试使用cas方法来更新数据 CASResponse casresp = mcc.cas("runoob", casValue.getCas(), 900, "Largest Tutorials-Library"); // 输出 CAS 响应信息 System.out.println("CAS Response - " + casresp); // 输出值 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 关闭连接 mcc.shutdown(); }catch(Exception ex) { System.out.println(ex.getMessage()); } } }

#### get 操作实例

### MemcachedJava.java 文件：

import java.net.InetSocketAddress; import java.util.concurrent.Future; import net.spy.memcached.MemcachedClient; public class MemcachedJava { public static void main(String[] args) { try{ // 连接本地的 Memcached 服务 MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211)); System.out.println("Connection to server successful."); // 添加数据 Future fo = mcc.set("runoob", 900, "Free Education"); // 输出执行 set 方法后的状态 System.out.println("set status:" + fo.get()); // 使用 get 方法获取数据 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 关闭连接 mcc.shutdown(); }catch(Exception ex) { System.out.println(ex.getMessage()); } } }

#### gets 操作实例、CAS

### MemcachedJava.java 文件：

import java.net.InetSocketAddress; import java.util.concurrent.Future; import net.spy.memcached.CASValue; import net.spy.memcached.CASResponse; import net.spy.memcached.MemcachedClient; public class MemcachedJava { public static void main(String[] args) { try{ // 连接本地的 Memcached 服务 MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211)); System.out.println("Connection to server successful."); // 添加数据 Future fo = mcc.set("runoob", 900, "Free Education"); // 输出执行 set 方法后的状态 System.out.println("set status:" + fo.get()); // 从缓存中获取键为 runoob 的值 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 通过 gets 方法获取 CAS token（令牌） CASValue casValue = mcc.gets("runoob"); // 输出 CAS token（令牌） 值 System.out.println("CAS value in cache - " + casValue); // 关闭连接 mcc.shutdown(); }catch(Exception ex) { System.out.println(ex.getMessage()); } } }

#### delete 操作实例

### MemcachedJava.java 文件：

import java.net.InetSocketAddress; import java.util.concurrent.Future; import net.spy.memcached.MemcachedClient; public class MemcachedJava { public static void main(String[] args) { try{ // 连接本地的 Memcached 服务 MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211)); System.out.println("Connection to server successful."); // 添加数据 Future fo = mcc.set("runoob", 900, "World's largest online tutorials library"); // 输出执行 set 方法后的状态 System.out.println("set status:" + fo.get()); // 获取键对应的值 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 对存在的key进行数据添加操作 fo = mcc.delete("runoob"); // 输出执行 delete 方法后的状态 System.out.println("delete status:" + fo.get()); // 获取键对应的值 System.out.println("runoob value in cache - " + mcc.get("runoob")); // 关闭连接 mcc.shutdown(); }catch(Exception ex) { System.out.println(ex.getMessage()); } } }

#### Incr/Decr 操作实例

### MemcachedJava.java 文件：

import java.net.InetSocketAddress; import java.util.concurrent.Future; import net.spy.memcached.MemcachedClient; public class MemcachedJava { public static void main(String[] args) { try{ // 连接本地的 Memcached 服务 MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211)); System.out.println("Connection to server successful."); // 添加数字值 Future fo = mcc.set("number", 900, "1000"); // 输出执行 set 方法后的状态 System.out.println("set status:" + fo.get()); // 获取键对应的值 System.out.println("value in cache - " + mcc.get("number")); // 自增并输出 System.out.println("value in cache after increment - " + mcc.incr("number", 111)); // 自减并输出 System.out.println("value in cache after decrement - " + mcc.decr("number", 112)); // 关闭连接 mcc.shutdown(); }catch(Exception ex) { System.out.println(ex.getMessage()); } } }

## PHP 连接 Memcached 服务

在前面章节中我们已经介绍了如何安装 Memcached 服务，接下来我们为大家介绍 PHP 如何使用 Memcached 服务。

#### PHP Memcache 扩展安装

PHP Memcache 扩展包下载地址：<http://pecl.php.net/package/memcache>，你可以下载最新稳定包(stable)。

wget http://pecl.php.net/get/memcache-2.2.7.tgz

tar -zxvf memcache-2.2.7.tgz

cd memcache-2.2.7

/usr/local/php/bin/phpize

./configure --with-php-config=/usr/local/php/bin/php-config

make && make install

**注意：**/usr/local/php/ 为php的安装路径，需要根据你安装的实际目录调整。

安装成功后会显示你的memcache.so扩展的位置，比如我的：

Installing shared extensions: /usr/local/php/lib/php/extensions/no-debug-non-zts-20090626/

最后我们需要把这个扩展添加到php中，打开你的php.ini文件在最后添加以下内容：

[Memcache]

extension\_dir = "/usr/local/php/lib/php/extensions/no-debug-non-zts-20090626/"

extension = memcache.so

添加完后 重新启动php,我使用的是nginx+php-fpm进程所以命令如下：

kill -USR2 `cat /usr/local/php/var/run/php-fpm.pid`

如果是apache的使用以下命令:

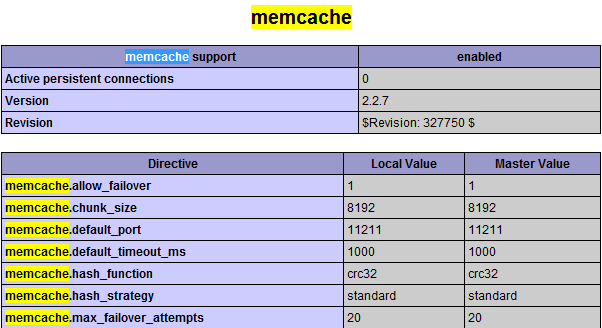
/usr/local/apache2/bin/apachectl restart

检查安装结果

/usr/local/php/bin/php -m | grep memcache

安装成功会输出：memcache。

或者通过浏览器访问 phpinfo() 函数来查看，如下图：



### PHP 连接 Memcached

<?php

$memcache = new Memcache; //创建一个memcache对象

$memcache->connect('localhost', 11211) or die ("Could not connect"); //连接Memcached服务器

$memcache->set('key', 'test'); //设置一个变量到内存中，名称是key 值是test

$get\_value = $memcache->get('key'); //从内存中取出key的值

echo $get\_value;

?>

更多 PHP 操作 Memcached 请参阅： <http://php.net/manual/zh/book.memcache.php>

[Java 连接 Memcached 服务](https://www.runoob.com/memcached/java-memcached.html)

### 1 篇笔记 写笔记

Centos 下使用 yum 命令快速安装 Memcached 与 php-memcached：

rpm qa|grep memcached //首先检查memcache是否已经安装完成

yum install memcached //（提示你是否确认安装输入y）检查完成后执行安装命令

yum install php-memcached //安装php的memcache的扩展

systemctl start httpd //开启apache

systemctl start memcached //开启memcached

setenforce 0

安装完成后可以使用 php 的 phpinfo(); 函数检查 memcache 是否安装。