13 正则表达式基础

正则表达式基础

正则表达式就是能用某种**模式**去**匹配**一类**字符串**的**公式**，它是由一串**字符**和**元字符**构成的**字符串**。

正则表达式**以行为单位**进行字符串的处理行为，通过一些**特殊符号**的辅助，让用户达到**查找、删除、替换**等目的。

正则表达式这个概念最初是由Unix中的工具（如 `grep` 和 `sed` ）普及开的。

一、 `grep` 命令格式

为了方便学习正则表达式，先了解下 `grep` 命令的使用方法。

1.1 `grep` 命令基本格式

`grep` 命令是一种强大的**文本搜索**工具，它能使用正则表达式搜索文本，在文件中全局查找指定的模式，并打印所有包含该表达式的**行**。

通常 `grep` 有三种版本，即 `grep` 、 `egrep` (等同于 `grep -E`)和 `fgrep` 。

- `egrep` 为扩展的 `grep` ，其支持更多的正则表达式**元字符**。
- `fgrep` 则为快速 `grep` (固定的字符串对文本进行搜索，不支持正则表达式的引用但查询极为快速)，它按字面解释所有的字符。

`grep` 命令的语法格式为： `grep [选项] 模式 [文件1 文件2……]`

模式即我们常说的正则表达式。

针对搜索字符串选项，使用正则表达式时必须用**单引号** `''` 括起来，避免与Shell中有特殊作用的字符，如 `>` 、 `|` 、 `&` 等冲突。

案例1： `grep` 命令基础应用

```shell
# 查找/etc/passwd包含对应模式的行
[root@Shell ~]# grep 'tom' /etc/passwd
[root@Shell ~]# grep 'bash' /etc/passwd
root:x:0:0:root:/root:/bin/bash
# 查找多个文件
[root@Shell ~]# grep 'root' /etc/passwd /etc/shadow /etc/group
/etc/passwd:root:x:0:0:root:/root:/bin/bash
/etc/passwd:operator:x:11:0:operator:/root:/sbin/nologin
/etc/shadow:root:$6$NZFAAMacJN2goGuO$JfNFM6GR49PTZCW26wqHChubDsKTekcn.5gzyI/1SMSkqgucb8/HIalLVqv2G.9KBTADEdQwHFD9s9D1Mq/JC/::0:99999:7:::
/etc/group:root:x:0:
```

1.2 `grep` 命令结合管道

`grep` 命令的输入可以来自**标准输入**或**管道**，而不仅仅是**文件**。这种方法在实际场景中应用更广泛。

案例2： `grep` 命令结合管道

```shell
#列出sshd相关进程
[root@Shell ~]# ps aux | grep 'sshd'
root        1429  0.0  0.3  82544  3580 ?        Ss   Feb06   0:00 /usr/sbin/sshd -D
root        2997  0.0  0.5 140772  5076 ?        Ss   Feb06   0:00 sshd: root@pts/0
root        3437  0.0  0.0 112644   952 pts/0    S+   00:29   0:00 grep --color=auto sshd
```

```shell
# 列出/root目录中的目录
[root@Shell ~]# ll /root | grep '^d'
drwxr-xr-x. 2 root root    6 Jan 28 20:02 test
drwxr-xr-x. 2 root root    6 Jan 28 20:01 tmp_dir
```

1.3 `grep` 命令的选项

`grep` 命令的常见选项如下。

```
-c, --count         显示成功匹配的行数
-q, --quiet, --silent    静默模式--quiet，--silent 即不输出任何信息
-o, --only-matching    仅显示匹配到的字符串本身
-v, --invert-match    反向查找，只显示不被模式匹配到的行
-l, --files-with-matches   只列出匹配行所在的文件名
-n, --line-number     在每一行前面加上它在文件中的相对行号

-i, --ignore-case   忽略字符的大小写
-R, -r,--recursive     递归针对目录
-s, --no-messages     禁止显示文件不存在或文件不可读的错误信息
--color             颜色
-A, --after-context=NUM  print NUM lines of trailing context  显示被模式匹配的行及其后#行
-B, --before-context=NUM  print NUM lines of leading context  显示被模式匹配的行及其前#行
-C, --context=NUM   print NUM lines of output context显示别模式匹配的行及其前后各#行
-G 支持基本正则表达式
```

案例3：使用 `grep` 匹配文件中 `root` 字符串，只匹配的显示文件名

```shell
[root@Shell ~]# grep -l 'root' /etc/passwd /etc/shadow /etc/hosts
/etc/passwd
/etc/shadow
```

案例分析： `-l` 选项表示只列出匹配行的**文件名**

案例4：使用 `grep` 匹配文件中 `root` 字符串，并显示匹配行在文件中的行号

```shell
[root@Shell ~]# grep -n 'root' /etc/passwd /etc/shadow /etc/hosts
/etc/passwd:1:root:x:0:0:root:/root:/bin/bash
/etc/passwd:10:operator:x:11:0:operator:/root:/sbin/nologin
/etc/shadow:1:root:$6$NZFAAMacJN2goGuO$JfNFM6GR49PTZCW26wqHChubDsKTekcn.5gzyI/1SMSkqgucb8/HIalLVqv2G.9KBTADEdQwHFD9s9D1Mq/JC/::0:99999:7:::
```

案例分析： `-n` 选项表示显示匹配行在文件中的**行号**

案例5：查找 `/etc/ssh/ssh_config` 中未被注释的行

```shell
[root@Shell ~]# grep -v '^#' /etc/ssh/ssh_config



Host *
        GSSAPIAuthentication yes
        ForwardX11Trusted yes
        SendEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
        SendEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
        SendEnv LC_IDENTIFICATION LC_ALL LANGUAGE
        SendEnv XMODIFIERS
```

案例分析：`-v` 选项表示**反向查找**，只显示不被模式匹配到的行

案例6：使用 `grep` 匹配文件中 `root` 字符串，不显示输出信息

```shell
[root@shell ~]# grep 'root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
[root@shell ~]# grep -q 'root' /etc/passwd
```

案例分析：`-q` 选项**不显示输出信息**

案例7：使用 `grep` 匹配文件中 `root` 字符串，只显示匹配到的字符串

```shell
[root@shell ~]# grep 'root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
[root@shell ~]# grep -o 'root' /etc/passwd
root
root
root
root
```

案例分析：`-o` 选项只显示匹配到的**字符串**

案例8：使用 `grep` 匹配文件中 `root` 字符串，只显示匹配到的行数

```shell
[root@shell ~]# grep 'root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
[root@shell ~]# grep -c 'root' /etc/passwd
2
```

案例分析： `-c` 选项只显示匹配到的**行数**

1.4 `grep` 命令的返回状态

`grep` 命令的返回状态如下。

```shell
找到匹配的表达式:          grep返回的退出状态为0
没找到匹配的表达式:        grep返回的退出状态为1
找不到指定文件:           grep返回的退出状态为2
```

案例9： `grep` 命令的返回状态

```shell
#找到匹配的表达式
# -q表示不显示结果
[root@Shell ~]# grep -q 'root' /etc/passwd
[root@Shell ~]# echo $?
0
#没找到匹配的表达式
[root@Shell ~]# grep 'tom' /etc/passwd
[root@Shell ~]# echo $?
1
#找不到指定文件
[root@Shell ~]# grep 'root' /etc/passwd1
grep: /etc/passwd1: No such file or directory
[root@Shell ~]# echo $?
2
```

二、正则表式基本语法

正则表达式由**普通字符**和**元字符（Meta Characters）**组成。

普通字符包括大小写的字母和数字。

元字符则具有特殊的含义，**元字符表达的是不同于字面本身的含义**。

元字符通常由各种执行模式匹配操作的程序（如：`vi`、`grep`、`sed`、`awk`、`python`）来解析。

模式描述在搜索文本时要匹配的一个或多个字符。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

构建正则表达式的方法和数学表达式的方法一样，也就是用**多种元字符与运算符可以将小的表达式结合在一起**来创建更大的表达式。

正则表达式可以是单个字符、字符集合、字符范围、字符间的等任意组合。

要想达到熟练使用正则表达式元字符，就要掌握最基本的元字符。

2.1 最简单的正则表达式

在最简单的情况下，一个普通的字符串就是一个正则表达式，功能类似于查找字符串。

例如，正则表达式 `testing` 中没有包含任何元字符，它匹配包含 `testing` 的字符串，因此可以匹配 `testing` 和 `testing123` 等字符串，但不能匹配 `Testing` 。

案例10：最简单的正则表达式

Shell 复制代码

```
[root@Shell ~]# echo 'testing' | grep 'testing'
testing
[root@Shell ~]# echo 'testing123' | grep 'testing'
testing123
[root@Shell ~]# echo 'Testing123' | grep 'testing'
```

2.2 定位符

定位符用来描述字符串或单词的边界，`^` 和 `$` **分别指字符串的开始与结束**，`\b` 描述单词的前或后边界，`\B` 表示非单词边界。

案例11：定位符 `^` 的使用

```shell
[root@Shell ~]# echo 'love you' | grep ^love
love you
[root@Shell ~]# echo 'i love you' | grep ^love
```

`^love` 这个模式包含一个特殊的字符 `^`，表示该模式只匹配那些以 `love` 开头的字符串。

该模式与字符串 `love you` 匹配，与 `I love you` 不匹配。

案例12：定位符 `$` 的使用

```shell
[root@Shell ~]# echo ploceman | grep man$
ploceman
[root@Shell ~]# echo plocy | grep man$
```

正如 `^` 符号表示开头一样， `$` 符号表示用来匹配哪些以给定模式结尾的字符串。

该模式与 `policeman` 匹配，与 `policy` 不匹配。

案例13：使用 `^` 、 `$` 表示精确匹配

```shell
[root@Shell ~]# echo /bin/bash | grep bash
/bin/bash
[root@Shell ~]# echo /bin/bash | grep bash$
/bin/bash
[root@Shell ~]# echo /bin/bash | grep ^bash$
```

字符 `^` 和 `$` 同时使用时，表示精确匹配（字符串与模式一样）。例如：只匹配字符串 `bash` 。

稍微复杂的字符，如标点符号和白字符（空格、制表符等），要用到转义符。**所有的转义序列都用反斜杠 `\` 开头**。例如：制表符的转义序列是 `\t` 。如果要检测一个字符串是否以制表符开头，可以用如下模式： `^\t` 。

2.3 字符簇（匹配字符）

在程序中，要判断输入的电话号码、地址、EMAIL地址、信用卡号码等是否有效，用普通基于字面的字符是不够的。因此需要使用相应的**字符模式**的方法来描述，它就是**字符簇**。

2.3.1 使用元字符 [ ] 匹配单个字符

匹配字符时常用到元字符 `[]` ，表示**匹配1个字符**。

当 `[]` 中有多个字符时，表示匹配其中的**任意一个字符**。 `[]` 中还可以使用 `-` 表示范围。

案例14：匹配单个字符

```shell
# 匹配a或c
[root@Shell ~]# echo a | grep [ac]
a
# 匹配所有a或c
[root@Shell ~]# echo aadac | grep [ac]
aadac
[root@Shell ~]# echo aadac | grep -o [ac]
a
a
a
c
# 匹配a或c，所以b不匹配
[root@Shell ~]# echo b | grep [ac]
# 匹配a、b或c
[root@Shell ~]# echo b | grep [a-c]
b
```

2.3.2 使用元字符 [ ] 匹配多个字符

前面的例子只能匹配单个字符，如果需要匹配**多个字符**可以使用**多个 `[]` 元字符，每个 `[]` 元字符表示1个字符的模式**。

案例15：匹配一个由一个小写字母和一个数字组成的字符串

```shell
# 只要是一个小写字母在前，紧挨着一个数字的都匹配
[root@Shell ~]# echo r21d13 | grep -o [a-z][0-9]
r2
d1
# 限定匹配一个小写字母开头，紧挨着一个数字结尾
[root@Shell ~]# echo r21d13 | grep -o ^[a-z][0-9]$
[root@Shell ~]# echo r2 | grep  ^[a-z][0-9]$
r2
```

2.3.3 使用 [ ] 中的 ^ 排除字符

当在一组**方括号 `[]` 中使用 `^`** 时，它表示“非”或“排除”的意思，常常用来**剔除**某个字符。

案例16：排除字符

```shell
#限定小写字母开头
[root@Shell ~]# echo r2 | grep ^[a-z]
r2
#限定非小写字母开头
[root@Shell ~]# echo r2 | grep ^[^a-z]
```

案例17：查找 `/etc/ssh/ssh_config` 中未被注释的行改进版（另一种写法加去除空行）

```shell
[root@Shell ~]#  grep ^[^#] /etc/ssh/ssh_config | grep -v ^$
Host *
        GSSAPIAuthentication yes
        ForwardX11Trusted yes
        SendEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
        SendEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
        SendEnv LC_IDENTIFICATION LC_ALL LANGUAGE
        SendEnv XMODIFIERS
```

2.3.4 常用的字符匹配模式

元字符	描述
`[:digit:]` 或 `[0-9]`	匹配任意单个字符
`[:lower:]` 或 `[a-z]`	匹配任意单个小写字母
`[:upper:]` 或 `[A-Z]`	匹配任意单个大写字母
`[:alpha:]` 或 `[a-zA-Z]`	匹配任意单个大写字母或小写字母
`[:alnum:]` 或 `[0-9a-zA-Z]`	匹配任意单个字母或数字
`[[:space:]]` 或 `TAB`	匹配单个空格
`[[:punct:]]`	表示任意单个标点
`[:alnum:]` 或 `[0-9a-zA-Z]`	匹配任意单个字母或数字

`[^]`	匹配指定集合外的任意单个字符

2.4 匹配次数

在实际工作中，经常要匹配一个单词或一组数字。在前面的例子中，需要单独指定字符的模式，非常繁琐。利用指定匹配次数的元字符可以简化操作。

元字符	描述
`{m}`	匹配其前面的字符m次
`{m,n}`	匹配前面的字符至少m次，至多n次
`.`	匹配除了换行符之外的任意单个字符。
`*`	匹配其前面的字符任意次，0，1或多次
`?`	匹配其前面的字符0次或1次
`+`	匹配前面的字符1次或多次
`()`	将一个或多个字符捆绑在一起，当做一个整体进行处理，反向引用照常使用

案例18：指定匹配次数

```shell
# 在字符串查找中，?作为元字符必须加\转义
[root@Shell ~]# echo cat | grep 'c?'
[root@Shell ~]# echo cat | grep 'c\?'
cat
#?可匹配前面字符0次
[root@Shell ~]# echo cat | grep "b\?"
cat
# +要求至少匹配1次
[root@Shell ~]# echo cat | grep "b\+"
[root@Shell ~]# echo ccat | grep "c\+"
ccat
#*可匹配前面字符任意次
[root@Shell ~]# echo cat | grep 'c*'
cat
[root@Shell ~]# echo cat | grep 'b*'
cat
#.匹配单个字符
[root@Shell ~]# echo cat | grep -o 'c.'
ca
[root@Shell ~]# echo cat | grep -o 'c..'
cat
#{2}指定匹配两次c，字符串匹配时需要为{}加上\
[root@Shell ~]# echo ccat | grep 'c\{2\}'
ccat
[root@Shell ~]# echo cat | grep 'c\{2\}'
#{m,n}指定匹配前面字符的次数的下限和上限，无上限表示无限
[root@Shell ~]# echo goooal | grep 'o\{2,5\}'
goooal
[root@Shell ~]# echo goooal | grep 'o\{2,\}'
goooal
[root@Shell ~]# echo goooal | grep 'o\{4,\}'
# ()表示将多个模式组合为整体
[root@Shell ~]# echo cacabca | grep -o '\(ca\)\{2\}'
caca
# 使用egrep命令匹配次数元字符不用转义
[root@shell ~]# echo cacabca | egrep -o '(ca){2}'
caca

```

`?` 等价于 `{0,1}`； `*` 等价于 `{0,}`； `+` 等价于 `{1,}`。

⚠ **使用 `egrep` 命令匹配次数元字符不用转义。**

小结

- `grep` 命令格式
- 正则表达式：定位元字符、字符簇、匹配次数元字符

课程目标

- 知识目标：熟练掌握grep命令和正则表达式的基本语法。
- 技能目标：能够根据实际需求编写简单的正则表达式。

课外拓展

- 进一步了解正则表达式的应用场景。

参考资料

- 编程胶囊：https://codejiaonang.com/#/courses
- 《Linux Shell核心编程指南》，丁明一，电子工业出版社