14 正则表达式应用

正则表达式应用

一、正则表达式语法规则

1.1 常用正则表达式元字符

元字符	描述	实例	类别
`\`	相当于其他编程语言中都的转义字符的概念。	例如：\\n匹配\n。\n匹配换行符。	基础字符
`[xyz]`	字符集合。匹配所包含的任意一个字符。	例如：[abc]可以匹配plain中的a。	字符簇
`[^xyz]`	负值字符集合。匹配未包含的任意字符。	例如：[^abc]可以匹配plain中的plin任一字符。	字符簇
`[a-z]`	字符范围。匹配指定范围内的任意字符。注意：只有连字符在字符组内部时，并且出现在两个字符之间时，才能表示字符的范围；如果出字符组的开头，则只能表示连字符本身。	例如，[a-z]可以匹配a到z范围内的任意小写字母字符。	字符簇
`^`	匹配输入行首。	[root@shell ~]# echo loveu | grep -o '^love' love	定位符

$	匹配输入行尾。	[root@shell ~]# echo ilove | grep –o 'love$' love [root@shell ~]# echo love \| grep –o '^love$' love	定位符
*	匹配前面的子表达式任意次。 *等价于{0,}	[root@shell ~]# echo z | grep 'zo*' z [root@shell ~]# echo zoo | grep 'zo*' zoo	次数匹配符
+	匹配前面的子表达式一次或多次(大于等于1次)。 +等价于{1,}。 扩展元字符	[root@shell ~]# echo zoo | grep 'zo\+' zoo [root@shell ~]# echo zoo | grep –E 'zo+' zoo [root@shell ~]# echo z | grep –E 'zo+'	次数匹配符
?	匹配前面的子表达式零次或一次。 等价于{0,1}。 扩展元字符	[root@shell ~]# echo do | grep –E 'do(es)?' do [root@shell ~]# echo does | grep –E 'do(es)?' does	次数匹配符

`{n}`	n是一个非负整数。匹配确定的n次。 扩展元字符	[root@shell ~]# echo food | grep –E 'o{2}' food [root@shell ~]# echo food | grep 'o\{2\}' food [root@shell ~]# echo Bob | grep –E 'o{2}'	次数匹配符
`{n,}`	n是一个非负整数。至少匹配n次。	#o{2,}不能匹配Bob中的o，但能匹配foooood中的所有o。 [root@shell ~]# echo Bob | grep –E 'o{2,}' [root@shell ~]# echo foooood| grep –E 'o{2,}' foooood	次数匹配符
`{n,m}`	m和n均为非负整数，其中n<=m。最少匹配n次且最多匹配m次。	[root@shell ~]# echo foooood| grep –oE 'o{1,3}' ooo oo	次数匹配符
`.`	点匹配除\n\r之外的任何单个字符。	[root@shell ~]# echo love | grep 'l..e' love	字符簇
`(pattern)`	匹配pattern并获取这一匹配。	要匹配圆括号字符，请使用\(或\)。	字符簇

`x|y`	匹配x或y。	#z|food能匹配z或food。 [root@shell ~]# echo z | grep –E 'z|food' z #[zf]ood则匹配zood或food。 [root@shell ~]# echo food | grep –E 'z|food' food	字符簇
`\b`	匹配一个单词的边界，也就是指单词和空格间的位置（即正则表达式的“匹配”有两种概念，一种是匹配字符，一种是匹配位置，这里的\b是匹配位置的）。	#er\b可以匹配never中的er，但不能匹配verb中的er； [root@shell ~]# echo never | grep –o 'er\b' er [root@shell ~]# echo verb | grep –o 'er\b' #\b1_可以匹配1_23中的1_，但不能匹配21_3中的1_。 [root@shell ~]# echo 1_23 | grep –o '\b1_' 1_ [root@shell ~]# echo 21_3 | grep –o '\b1_'	定位符

\B	匹配非单词边界。	#er\B能匹配verb中的er，但不能匹配never中的er。 [root@shell ~]# echo verb | grep -o 'er\B' er [root@shell ~]# echo never | grep -o 'er\B'	定位符
\d	匹配一个数字字符。等价于[0-9]。	#grep要加上-P，perl正则支持。 [root@shell ~]# echo a1 | grep -oP '\d' 1	字符簇
\D	匹配一个非数字字符。等价于[^0-9]。	#grep要加上-P，perl正则支持。 [root@shell ~]# echo a1 | grep -oP '\D' a	字符簇
\w	匹配包括下划线的任何单词字符。类似但不等价于“[A-Za-z0-9_]”，这里的“单词”字符使用Unicode字符集。	[root@shell ~]# echo /a | grep -o '\w' a	字符簇
\W	匹配任何非单词字符。等价于“[^A-Za-z0-9_]”。	[root@shell ~]# echo /a | grep -o '\W' /	字符簇
\s	匹配任何不可见字符，包括空格、制表符、换页符等等。等价于[\f\n\r\t\v]	[root@shell ~]# cat temp.txt aa bb cc [root@shell ~]# grep '\s' -o temp.txt	字符簇

`\S`	匹配任何可见字符。等价于[^\f\n\r\t\v]。	root@shell ~]# grep '\S' temp.txt a a b b c c	字符簇
`\cx`	匹配由x指明的控制字符。		特殊符号
`\f`	匹配一个换页符，等价于\x0c和\cL。		特殊符号
`\n`	匹配一个换行符。等价于\x0a和\cJ。		特殊符号
`\r`	匹配一个回车符。等价于\x0d和\cM。		特殊符号
`\t`	匹配一个制表符。等价于\x09和\cI。		特殊符号
`\v`	匹配一个垂直制表符。等价于\x0b和\cK。		特殊符号

1.2 正则表达式优先级

正则表达式**从左到右**进行计算，并遵循**优先级**顺序，这与算术表达式非常类似。相同优先级的正则表达式从左到右进行运算，不同优先级的正则表达式运算先高后低。

运算符优先级顺序如下表所示。

运算符	描述
`\`	转义符

`( ) (?:)(?=) []`	圆括号和方括号
`*`	表示匹配零次到多次。
`+ ? {n} {n,} {n,m}`	限定符
`^ $ \` 任何元字符、任何字符	定位点和序列 （即：位置和顺序）
`|`	替换，“或”字符具有高于替换运算符的优先级，使得“m|food”匹配“food”。若要匹配“mood”或“food”，请使用括号创建子表达式，从而产生“(m|f)ood”

二、grep正则表达式实例

- **案例1**：显示 `/proc/meminfo` 文件中以大小s开头的行

案例分析

```
^表示匹配行首，[sS]表示匹配s或S
```

代码

```
[root@Shell ~]# cat /proc/meminfo | grep '^[sS]'
SwapCached:            0 kB
SwapTotal:       2097148 kB
SwapFree:        2097148 kB
Shmem:              6812 kB
Slab:              51860 kB
SReclaimable:      18220 kB
SUnreclaim:        33640 kB
```

- **案例2**：显示默认shell不是 `/bin/shell` 的用户

案例分析

```
/bin/bash$表示行以/bin/bash结尾，-v表示反向匹配模式，即不匹配/bin/bash$的行。
```

代码

```shell
[root@Shell ~]# cat /etc/passwd | grep -v "/bin/bash$"
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
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
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
avahi-autoipd:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/sbin/nologin
systemd-bus-proxy:x:999:997:systemd Bus Proxy:/:/sbin/nologin
systemd-network:x:998:996:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:997:995:User for polkitd:/:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
```

扩展：满足条件的有多少用户?

```shell
cat /etc/passwd | grep -cv "/bin/bash$"
cat /etc/passwd | grep -v "/bin/bash$" |wc -l
```

- **案例3**：显示 `/etc/inittab` 中以 `# 空格` 开头的行

代码

```shell
[root@Shell ~]# grep '^#[[:space:]]' /etc/inittab
# inittab is no longer used when using systemd.
# ADDING CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.
# Ctrl-Alt-Delete is handled by /usr/lib/systemd/system/ctrl-alt-del.target
# systemd uses 'targets' instead of runlevels. By default, there are two main targets:
# multi-user.target: analogous to runlevel 3
# graphical.target: analogous to runlevel 5
# To view current default target, run:
# systemctl get-default
# To set a default target, run:
# systemctl set-default TARGET.target

# 或者grep '^#\s' /etc/inittab
```

- **案例4**：提取IP地址

案例分析

```shell
([0-9]{1,3}\.){3}中[0-9]{1,3}表示1-3位数字，\.表示.，{…}表示前面的模式匹配3次。由于
```

代码

```shell
[root@Shell ~]# egrep '([0-9]{1,3}\.){3}[0-9]{1,3}' /etc/sysconfig/network-scripts/ifcfg-ens33
IPADDR=192.168.68.2
NETMASK=255.255.255.0
GATEWAY=192.168.68.1
```

- **案例5**：找出 `/etc/rc.d/init.d/functions` 文件中行首为某单词（包括下划线）后面跟一个小括号的行

案例分析

Shell 复制代码

```
^[a-zA-Z]*_*.*\(\)中[a-zA-Z]*表示匹配字母任意次（包括0次）；_*表示匹配_任意次（包括0次）；.*表示前面的模式匹配任意次，\(\)表示匹配括号。
```

代码

Shell 复制代码

```
[root@Shell ~]# cat /etc/rc.d/init.d/functions | grep -Eo "^[a-zA-Z]*_*.*\(\)"
systemctl_redirect ()
checkpid()
__pids_var_run()
__pids_pidof()
daemon()
killproc()
pidfileofproc()
pidofproc()
status()
echo_success()
echo_failure()
echo_passed()
echo_warning()
update_boot_stage()
success()
failure()
passed()
warning()
action()
strstr()
is_ignored_file()
is_true()
is_false()
apply_sysctl()
```

- **案例6**：列出 `/etc/` 目录下所有以 `.conf` 结尾的文件名，并将其名字转换为大写 。

案例分析

Shell 复制代码

```
"[^/]*(\.conf)$"中[^/]*表示不以/开头的字符，(\.conf)$表示以.conf结尾。
```

代码

Shell | 复制代码

```
[root@Shell ~]# find /etc -name '*.conf'| grep -Eo "[^/]*(\.conf)$" | tr 'a-z' 'A-Z'
RESOLV.CONF
CA-LEGACY.CONF
FASTESTMIRROR.CONF
LANGPACKS.CONF
SYSTEMD.CONF
VERSION-GROUPS.CONF
YUM-CRON-HOURLY.CONF
YUM-CRON.CONF
LVM.CONF
LVMLOCAL.CONF
DRACUT.CONF
DIST.CONF
LIBUSER.CONF
AUDITD.CONF
SESTATUS.CONF
MLX4.CONF
RDMA.CONF
MLX4.CONF
BOOTCHART.CONF
COREDUMP.CONF
JOURNALD.CONF
LOGIND.CONF
SYSTEM.CONF
USER.CONF
99-SYSCTL.CONF
LISTEN.CONF
UDEV.CONF
HOST.CONF
MAN_DB.CONF
SYSCTL.CONF
00-KEYBOARD.CONF
NSS-SOFTOKN-PRELINK.CONF
FIPSCHECK.CONF
GRUB2.CONF
LD.SO.CONF
MARIADB-X86_64.CONF
KERNEL-3.10.0-327.EL7.X86_64.CONF
NSSWITCH.CONF
NETWORKMANAGER.CONF
10-IBFT-PLUGIN.CONF
FIREWALLD.CONF
ORG.FREEDESKTOP.HOSTNAME1.CONF
ORG.FREEDESKTOP.LOCALE1.CONF
```

```
ORG.FREEDESKTOP.LOGIN1.CONF
ORG.FREEDESKTOP.MACHINE1.CONF
ORG.FREEDESKTOP.SYSTEMD1.CONF
ORG.FREEDESKTOP.TIMEDATE1.CONF
ORG.FREEDESKTOP.POLICYKIT1.CONF
DNSMASQ.CONF
WPA_SUPPLICANT.CONF
TEAMD.CONF
NM-AVAHI-AUTOIPD.CONF
NM-DISPATCHER.CONF
NM-IFCFG-RH.CONF
ORG.FREEDESKTOP.NETWORKMANAGER.CONF
COM.REDHAT.TUNED.CONF
FIREWALLD.CONF
SESSION.CONF
SYSTEM.CONF
LIBAUDIT.CONF
DNSMASQ.CONF
WPA_SUPPLICANT.CONF
TCSD.CONF
PLYMOUTHD.CONF
SMTPD.CONF
AUDISPD.CONF
AF_UNIX.CONF
SYSLOG.CONF
SEMANAGE.CONF
SETRANS.CONF
KRB5.CONF
RSYSLOG.CONF
PYTHON.CONF
ACCESS.CONF
CHROOT.CONF
GROUP.CONF
LIMITS.CONF
20-NPROC.CONF
NAMESPACE.CONF
PAM_ENV.CONF
SEPERMIT.CONF
TIME.CONF
PWQUALITY.CONF
KDUMP.CONF
LOGROTATE.CONF
ASOUND.CONF
LDAP.CONF
TUNED-MAIN.CONF
YUM.CONF
SUDO-LDAP.CONF
SUDO.CONF
```

```
E2FSCK.CONF
MKE2FS.CONF
VCONSOLE.CONF
LOCALE.CONF
```

- **案例7**：显示系统中所有系统用户的用户名和UID

案例分析

```shell
在/etc/passwd文件中第一列为用户名，第三列为UID，因此需要提取第1/3列。
bin:x:1:1:bin:/bin:/sbin/nologin
cut -d ":" -f1,3 含义为以:为分隔符分割文档，提取第1/3列
普通用户UID默认从1000开始，因此排除1000以上的UID即只留下系统用户
```

代码

```shell
[root@Shell ~]# cat /etc/passwd | cut -d ":" -f1,3 | grep -v 'root' | grep -v '[0-9]{4,}'
bin:1
daemon:2
adm:3
lp:4
sync:5
shutdown:6
halt:7
mail:8
operator:11
games:12
ftp:14
nobody:99
avahi-autoipd:170
systemd-bus-proxy:999
systemd-network:998
dbus:81
polkitd:997
tss:59
postfix:89
sshd:74
```

小结

- 常用元字符

- 优先级

课程目标

- 知识目标：熟练掌握正则表达式的语法规则。
- 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