正则表达式

曹东刚

caodg@sei.pku.edu.cn

Linux 程序设计环境

http://c.pku.edu.cn/

内容提要

基本概念

正则表达式 (REs: Regular Expressions)

用于描述一组字符串的表达式, 或称为模式。英文缩写: regexp, regex, regxp, regexps, regexes, regexen.

- 正则表达式由普通字符 (例如字符 a 到 z) 以及元字符组成.

匹配 (matched)

一个正则表达式和一个字符串匹配, 如果该字符串中的字符序列和正则表达式定义的字符序列对应.

正则表达式文法

有多种不同的正则表达式, 文法有差异

- 传统 Unix 正则表达式, POSIX 基本正则表达式 (BRE)
- POSIX 扩展正则表达式 (ERE)
- Perl-Compatible Regular Expression (PCRE)

```
$ sudo apt-get install pcregrep ack
$ man pcrepattern
$ echo hello | pcregrep -o h
$ echo hello | ack -o h
```

正则表达式引擎

不同的正则表达式引擎对正则表达式有不同的解释

DFA 引擎

顺序输入，速度快，表达力弱．最左最长匹配

- awk, egrep, mysql, lex...

传统 NFA 引擎

回溯 (backtracking)，速度慢，灵活，表达力强．最左匹配

- Perl, Java, .NET, PHP, Python, Ruby, sed, GNU emacs, vi...

正则表达式引擎 (cont.)

POSIX NFA 引擎

匹配成功后仍然尝试所有可能. 最左最长匹配

混合引擎

DFA 和 NFA 的优点

- GNU awk 与 grep(两个单独的引擎), Tcl (单引擎)

NFA 引擎

NFA 引擎由正则表达式驱动

1. 引擎从 regex 开始，尝试匹配字符串的首字符.
   - 如果两者不匹配，则尝试首字符后面的字符.
   - 如果二者匹配，引擎会继续读入 regex，并和剩下的字符串匹配.
2. 如果引擎处理完 regex，且字符串匹配，则认为匹配成功，放弃其它选择.

NFA 引擎 (cont.)

若 regex 有多个选择项，此时

- NFA 引擎将选择其中的一个，并记录下其它的选择以及当前字符串位置.
- 如果随后的匹配失败，且有选择未尝试，引擎将回溯到该选择点，尝试其它选择.
- 如果所有选择都不合法，则匹配失败，引擎处理下一个字符.

NFA 回溯规则

- 总是首先回溯到最近保存的可选点, 如 LIFO 堆栈
- 在存在选择 (alternation) 的情况下, 按给出的顺序依次尝试
- 对于贪心限定符 (greedy quantifier), 首先尝试另一个匹配, 最后选择跳过
- 对于非贪心限定符 (lazy quantifier), 首先选择跳过, 最后尝试其它匹配

NFA 回溯规则 (cont.)

- 对于原子组合 (atomic grouping) 或占有限定符 (possessive quantifier) 不保存后向引用. 它们总是作为一个整体被处理.
- 占有限定符会取消回溯

小心给出选择的顺序, 尽量优化以减少回溯

元字符概览

字符	描述
.	匹配任何单个字符 (是否匹配换行符依赖于匹配模式及引擎实现)
\	将下一个字符转义. 例: \n 匹配一个换行符, \\ 匹配 '\'
^	匹配输入字符串的开始位置. 对于多行匹配, ^ 也匹配 '\n' 或 '\r' 之后的位置
$	匹配输入字符串的结束位置. 对于多行匹配, $ 也匹配 '\n' 或 '\r' 之前的位置
*	匹配前面的子表达式零次或多次. 例: ab* 能匹配 "a", "ab"
[xy-z]	字符集合, 匹配所包含的任意一个字符. 例: [abc] 可以匹配 "ascii" 中的 'a'。
[^xyz]	负值字符集合, 匹配未包含的任意字符. 例: [^abc] 可以匹配 "dog" 中的 'd'。

元字符概览

字符	描述
\(\)	将\(\) 之间的表达式标记为“组”(group)，并且保存匹配该表达式的字符 (最多可以保存 9 个)，用 \1 到 \9 的符号来引用
\{m,n\}	最少匹配 m 次且最多匹配 n 次. 例: A[0-9]\{3\} 能够匹配 A123,A348 等，但不匹配 A1234. A[0-9]\{5,6\} 匹配 A12345, A234345
\< \>	匹配词的开始和结束. 例: \<the 能够匹配字符串“in the day”中的“the”，但不匹配“otherwise”中的“the”
|	将两个匹配条件进行逻辑“或”运算. 例: (him|her) 匹配“him”与“her”
+	匹配它之前的那个字符 1 次或多次; 占有限定符
?	匹配它之前的那个字符 0 次或 1 次; 非贪心限定符

普通字符

匹配最先满足 regex 的字符串

regex **is** 匹配 “this is a good toy” 中 “this” 的 “is”

大小写敏感

regex **Cat** 匹配 “Cat”, 不匹配 “cat”

需要转义的特殊字符: `[ \ ^ $ . | ? * + ( )`

例: regex 1+1=2 与1\+1=2

例: regex “”

字符集合 —1

- 例: regex gr[ae]y 匹配 “gray” 或者 “grey”
- 例: regex [0-9a-fA-FX] 匹配一个 16 进制数字字母, 或字母 “X”
- 例: regex [A-Za-z_][A-Za-z_0-9]* 程序语言中的标识符
- 例: regex q[^u] 匹配 “Iraq␣”, 不匹配 “Iraq”

字符集合 —2

- 字符集合 (方括号中) 中的特殊字符有 "] - \ ^", 在特定位置有特殊含义[1].
- 例: regex [+*], [x^], []x], [^]x], [-x]
- 字符集合作为一个整体重复匹配.

  例: regex [0-9]+

[1] 转义字符的处理在不同的引擎中有可能有差别, 请参考手册说明

POSIX 预定义字符集合

类别	含义	解释
[:upper:]	[A-Z]	大写字母
[:lower:]	[a-z]	小写字母
[:alpha:]	[A-Za-z]	大小写字母
[:alnum:]	[A-Za-z0-9]	数字与大小写字母
[:digit:]	[0-9]	数字
[:xdigit:]	[0-9A-Fa-f]	十六进制数字
[:punct:]	[.,!?:...]	标点符号
[:blank:]	[␣\t]	空格与制表符
[:space:]	[␣\t\n\r\f\v]	空白字符
[:cntrl:]		控制字符
[:graph:]	[^␣\t\n\r\f\v]	可打印字符
[:print:]	[^\t\n\r\f\v]	可打印字符与空格

使用的时候，前面要加括号，如[[:upper:]]

PCRE 预定义字符集合

类别	含义
\s	匹配任何空白字符，等价于[␣\f\n\r\t\v]
\S	匹配任何非空白字符，等价于[^␣\f\n\r\t\v]
\b	匹配一个单词边界，指单词和空格间的位置. 例: er\b 可以匹配“never”中的‘er’，但不能匹配“verb”中的‘er’
\B	匹配非单词边界. 例: er\B 可以匹配“verb”中的‘er’，但不能匹配“never”中的‘er’
\d	匹配一个数字字符，等价于 [0-9].
\D	匹配一个非数字字符，等价于 [^0-9].
\w	匹配包括下划线的任何单词字符，等价于[A-Za-z0-9_]
\W	匹配任何非单词字符，等价于 [^A-Za-z0-9_]

圆点 (.)

缺省情况下, 圆点. 等价于[^\n] 或者[^\n\r]

- 单行模式: 圆点. 匹配换行符. PCRE 中用/s 控制
- 多行模式: ^ 和 $ 分别扩展匹配字符串中换行符的前后位置. PCRE 中用/m 控制
- 单行模式只影响圆点, 与多行模式无关

圆点 (.) 示例

匹配 mm/dd/yy, 允许多种日期分隔符

1. `\d\d.\d\d.\d\d`
2. `\d\d[-␣/.]\d\d[-␣/.]\d\d`
3. `[0-1]\d[-␣/.][0-3]\d[-␣/.]\d\d`

多行模式示例

找出 grep.txt 中的 “my ice tea”

```
$ cat grep.txt
this is my
ice tea.
```

多行模式示例

找出 grep.txt 中的 “my ice tea”

```
$ cat grep.txt
this is my
ice tea.

$ pcregrep -M -o 'my\s+ice\s+tea' grep.txt
```

另一个例子

对下面的输入字符串，要求匹配一对双引号之间的字符串

Today, he said “word one” and “word two”. What a puzzle.

1. regex ".*" 匹配结果为 “word one” and “word two”
   星号 * 是贪心限定符
2. regex "[^"]*" 匹配结果为 “word one”

定位符

常用定位符: ^ $ \b \B

验证用户输入合法数字串

- regex \d+ (Wrong!)
- regex ^\d+$ (Correct)

匹配单字母 4

regex \b4\b

选择符

选择符 | 的优先级最低

- reg \b(cat|dog)\b : 匹配单词 cat 或 dog
  - 比较 reg \bcat|dog\b
- 输入字符串为 “SetValue”,
  regex 为Get|GetValue|Set|SetValue
  - 比较: regex Get|GetValue|SetValue|Set
  - 比较: regex Get(Value)?|Set(Value)?

数量限定符

- 数量限定符? * + 以及{m,n} 缺省情况下是“贪心”的
- 例: 找到下面字符串中的 HTML 标识符,

  This is a <EM>first</EM> test

  - regexp <.+> 匹配的结果是: <EM>first</EM>

非贪心限定符

- 将数量限定符改为“非贪心”(Laziness): 将问号**?** 放到表达式的后面
  - 对上面例子应用 regexp <.+?>
  - 比较负值运算: regexp <[^>]+>
  - 对于输入字符串“1234567 122”, 比较 regexp [1-9][0-9]{2,4} 和[1-9][0-9]{2,4}?

组合与后向引用

一对圆括号 () 将其中的所有正则表达式组合为一个子表达式, 并创建一个后向引用 (backreference).

- regex `Set(Value)?` 匹配 “Set” 或者 “SetValue”, 对于前者, 后向引用的值为空; 对于后者, 后向引用的值为 “Value”
- 例: regex `([a-c])x\1` 匹配: “axa”, “bxb”, “cxc”

组合与后向引用 (cont.)

更多例子

- 对 “Testing <B><I>bold italic</I></B> text”
  应用 regex <([A-Z][A-Z0-9]*)[^>]*>.*?</\1>
  - 结果为: “<B><I>bold italic</I></B>”
- 例: 对字符串 “cab”, 比较 regex ([abc]+) 和 regex ([abc])+
  - 前者后向引用为 “cab”, 后者为 “b”(后面的结果覆盖前面的)

断言

问题: 找到字符串 <td>Nice</td> 中 td tag 的内容.

方法: 前向 (look-ahead) 与后向 (look-behind) 断言.

前向断言

- 肯定断言 (?=)
  - 例: \w+(?=;)
- 否定断言 (?!)
  - 例: foo(?!bar)

断言 — cont.

后向断言

- 肯定断言 (?<=)
  - 例: (?<=linu)X
- 否定断言 (?<!)
  - 例: (?<!foo)bar

<td>Nice</td> 的 tag 内容的匹配:

(?<=<td>).*?(?=</td>)

占有限定符 possessive quantifier

取消贪心匹配的回溯，数量限定符后加加号 +

```
$ echo ababacd | ack -o '(ab)*a'
$ echo ababacd | ack -o '(ab)*?a'
$ echo ababacd | ack -o '(ab)*+a'
$ echo aaaaaaa | ack -o 'a?+a'
$ echo aaaaaaa | ack -o 'a+a'
$ echo aaaaaaa | ack -o 'a++a'
```

优先级顺序

操作符	描述
\	转义符
(), []	组合符和集合符
*, +, ?, {n}, {n,}, {n,m}	数量限定符
^, $	定位符和顺序
|	逻辑“或”操作

内容提要

grep

grep(全局正则表达式), 允许对文本文件进行模式查找。如果找到匹配模式, grep打印包含模式的所有行

grep 有三种变型

- grep: 标准 grep, 支持基本正则表达式
- egrep: 扩展 grep, 支持基本及扩展的正则表达式
- fgrep: 快速 grep, 允许查找字符串而不是一个模式

ack 被设计为取代 99% 的 grep 使用, ack 支持 PCRE

grep (cont.)

grep 命令格式:

grep [选项] 正则表达式 [文件]

```
$ echo abcdabcd | grep -o -P '(ab)*a' # pcre grep
```

单引号与双引号

shell 在处理命令时, 要对输入进行解释. 为了把参数正确传递给程序, 有时候需要阻止 shell 扩展参数中的特殊字符, 这可通过单引号或双引号实现.

- 双引号: 只对输入字符串中的$ `` \ 进行扩展

```
$ BOY=boy; echo "The $BOY did well"
$ echo "The $DAY is `date`"
```

单引号与双引号 (cont.)

- 单引号: 对输入字符串不做任何特殊处理

```
caodg@debian:~$ BOY=boy; echo 'The $BOY did well'
```

- 输入模式参数给 grep 时, 用单括号将模式括起来

```
caodg@debian:~$ grep '[ab]*|45$' input.txt
```

grep 示例

- 在当前目录下的 *.c 文件中查找字符串 “printf” (不分大小写)

```
caodg@debian:~$ grep -i 'printf' *.c
```

- 统计文件 main.c 中包含字符串 “printf” 的总行数

```
caodg@debian:~$ grep -c 'printf' main.c
```

- 列出 main.c 中包含字符串 “printf” 的每一行

```
caodg@debian:~$ grep -n 'printf' main.c
```

grep 示例 (cont.)

- 列出 main.c 中不包含字符串 “printf” 的每一行

```
caodg@debian:$ grep -v 'printf' main.c
```

- 递归搜索目录 worm 下的包含字符串 “trojan” 的文本文件

```
grep -r --binary-files=without-match 'trojan' worm
```

内容提要

find

find 查找文件系统中具有特定特征的文件

命令格式: find [路径] [表达式]

- 查找/home 目录下所有的 core 文件

```
caodg@debian:~$ find /home -name core -print
```

注意: 尽量输出相对路径, 当备份时可以在临时目录恢复.

find—示例

- 查找/home 目录下文件名包含“core”的文件

```
caodg@debian:~$ find /home -name '*core*' -print
```

- 查找属主帐户被删除的文件

```
caodg@debian:~$ find /var -nouser
```

查找特定类型的文件

- 查找/home 目录下所有的目录文件

```
caodg@debian:~$ find /home -type d
```

类型	描述
f	普通文件
d	目录
b	块设备文件
c	字符设备文件
l	符号链接文件
p	管道文件

按大小查找文件

- 查找/home 目录下大于 100M 的文件

```
caodg@debian:~$ find /home -size +100M
```

类型	描述
b	512 byte 磁盘块 (default)
c	字节
w	双字节
k	kilobyte
M	megabyte
G	gigabyte

- 注意: 数字前面加加号, 表示大于给定值; 减号, 表示小于给定值; 什么都不加表示等于

按时间查找文件

- 查找/home 目录下最近 5 天内内容修改过的文件

```
caodg@debian:~$ find /home -mtime -5
```

- 查找/home 目录下 5 天前访问过的文件

```
caodg@debian:~$ find /home -atime +5
```

- 查找/home 目录下最近 5 天内状态改变过的文件

```
caodg@debian:~$ find /home -ctime -5
```

- 查找当前目录下比 a.txt 新的文件

```
caodg@debian:~$ find . -newer a.txt
```

选项组合

- 查找/home 目录下最近 5 天内内容修改过的普通文件

```
~$ find /home -type f -mtime -5
```

- 查找当前目录下比 a.txt 新, 但比 b.txt 老的文件

```
~$ find . -newer a.txt ! -newer b.txt
```

- 查找 core 文件或者 tmp 目录

```
~$ find . -name core -o \( -type d -name tmp \)
```

  - 注意空格: "\(␣expression␣\)"

对找到的文件执行动作

- 删掉/home 目录下的所有 core 文件

```
~$ find /home -name core -exec rm -f {} \;
```

```
~$ find /home -name core -delete
```

- 将所有 tgz 文件移动到/tmp 目录下并确认

```
~$ find /home -name "*.tgz" -ok mv {} /tmp \;
```