

# 道正则表达式

傅利民(phoolimin@gmail.com)

September 19, 2009

## Contents

1	介绍	1
2	字符类	1
3	模式项	3
4	模式项重复	3
5	模式组和扑获	3
6	字符串函数	4
7	例子	6

## 1 介绍

正则表达式是一个用单个字符串表示的句法规则，用以描述或者匹配符合该规则的所有字符串。正则表达式主要被用来进行字符串操作，如在字符串里查找，提取或替换符合某规则的子字符串。

从最新的发布开始，道语言不再从语法上支持正则表达式，正则表达式的功能改由字符串方法支持。而且新的正则表达式的语法也有所改变，变得更象Lua语言所支持的正则表达式（主要区别在于字符类的表示符号）。

## 2 字符类

一个字符类表示从属某类的所有字符：



### 3 模式项

模式项可以是：

- 单个字符类；
- $\wedge$ ：匹配字符串开头；
- $\$$ ：匹配字符串结尾；
- $\%n$ ：匹配第 $n$ 个模式组的子字符串； $n$ 可以是一个或多个数字；
- $\%bxy$ ：匹配一平衡配对的字符 $x$ 和 $y$ ；这里平衡表示，从相同的匹配位置开始，被匹配的子字符串必须包含相同且最少数目的 $x$ 和 $y$ ；类似于Lua的字符串模式语法；
- $\%B\{pattern1\}\{pattern2\}$ ：匹配一平衡配对的模式 $pattern1$ 和 $pattern2$ ；类似于 $\%bxy$ ；

### 4 模式项重复

模式项 $e$ 可以选择性的被忽略或重复匹配，规则如下：

- $e?$ ：匹配零次或一次；
- $e^*$ ：匹配零次或任意次数；
- $e^+$ ：匹配一次或多次；
- $e\{n\}$ ：匹配 $n$ 次；
- $e\{n,\}$ ：匹配至少 $n$ 次；
- $e\{,n\}$ ：匹配至多 $n$ 次；
- $e\{n,m\}$ ：匹配至少 $n$ 次，且最多 $m$ 次；

### 5 模式组和扑获

在正则表达式里，可以用括号将一个或多个模式项括起来形成一个子模式，即模式组(group)。一个模式组里可包含多个可选子模式，以 $|$ 分开。如果用 $(|pattern)$ 或 $(pattern|)$ 包含一个空的可选子模式，那么这个模式组可在字符串匹配过程中被跳过。如果正则表达式里含有多个模式组，那么按模式组的左括号所出现的顺序，每个模式组都会自动获得一个标号。例如，在 $(\%a+)\%s*(\%d+(\%a+))$ 里，第

一个(*%a+*) 将拥有标号1, (*%d+(%a+)*) 拥有标号2, 而第二个(*%a+*) 将拥有标号3。为了方便起见, 整个正则表达式所表达的模式也被自动定义为一个模式组, 标号为0。

当一个字符串被匹配到一个正则表达式时, 那些与其中的模式组相匹配的子字符串将被标记(捕获), 以便于被引用或提取。如在进行字符串匹配或替换时, *%n* 可被用作表示第*n*个模式组所匹配的子字符串。

当一个正则表达式可以有多种方式匹配到一个字符串里起始于同一个下标的子字符串时, 匹配长度最长的匹配方式将被选中, 并返回相应的结果。匹配长度将被定义为该表达式里所有模式组所匹配的子字符串的长度的和。根据这种定义, 如果需要给予某个模式组更高的匹配优先权, 那么可以给该模式组增加更多层括号。例如(*%d%w\**)(*%w\*%d*) 可以有两种方式匹配到*1a2*, 一种是将*1a* 匹配给(*%d%w\**) 及*2* 匹配给(*%w\*%d*); 另一种是将*1* 匹配给(*%d%w\**) 及*a2* 匹配给(*%w\*%d*)。如果在(*%w\*%d*) 外再添加一层括号, 即(*%d%w\**)(*(%w\*%d)*), 那么它与*1a2* 的匹配将变得唯一, 也就是将*a* 匹配到后面的组里。

## 6 字符串函数

象Lua里一样, 正则表达式的功能需要通过字符串的成员方法使用。正则表达式需要以字符串的形式作为参数传递给那些方法。每个正则表达式字符串都在第一次使用时被编译为一内部表达结构, 并与之一一对应。每当正则表达式字符串被使用时, 其相应的内部表达结构将被取用, 以避免多次编译正则表达式。实际上, 每个正则表达式字符串在每一个虚拟进程里仅被编译一次。

### 6.0.1 pfind(): 查找匹配模式的子字符串

```
string.pfind( pt : string, index=0, start=0, end=0 )=>list<tuple<int,int> >
```

此方法查找匹配于模式 $pt$ 的子字符串的位置(下标和长度)。如果 $index$ 大于零, 查找第 $index$ 个匹配的子字符串, 否则查找所有的。此查找从字符串下标 $start$ 开始, 直到下标 $end$ 结束。如果 $end$ 等于零, 一直查找到字符串末尾。

### 6.0.2 match(): 匹配模式

```
string.match( pt : string, start=0, end=0, substring=1 )  
=>tuple<start:int,end:int,substring:string>
```

从字符串下标`start`开始查找匹配于模式`pt`的子字符串,直到下标`end`结束。如果`end`等于零,一直查找到字符串末尾。如果`substring`非零,返回元组的`substring`成员将保存匹配的子字符串。

### 6.0.3 `extract()`: 提取子字符串

```
string.extract( pt : string, matched=1, mask='', rev=0 )=>list<string>
```

如果只使用参数`pt` (`matched`为正),此方法将从`string`里提取所有能匹配到模式`pt`的子字符串。当`matched`为负时,字符串`string`将与模式`pt`相匹配的子字符串所分割(所匹配子字符串的互补),并返回分割后的子字符串;当`matched`为零时,返回所匹配子字符串及其互补。

如果另一个模式`mask`也作为参数传入给此方法(且`rev`是零),对于模式`pt`的查找将被局限于与`mask`匹配的子字符串里;如果`rev`不是零,与`mask`匹配的子字符串将从对于模式`pt`的查找中被排除。

### 6.0.4 `capture()`: 捕获模式组

```
string.capture( pt : string, start=0, end=0 )=>list<string>
```

查找与模式`pt`匹配的子字符串,获取并返回与模式`pt`里模式组相匹配的子字符串。返回的列表里的第`i`个元素对应于与第`i`个模式组相匹配的子字符串。

### 6.0.5 `change()`: 替换子字符串

```
string.change( pt : string, s : string, index=0, start=0, end=0 )=>int
```

将所有与模式`pt`的子字符串替换为字符串`s`。`s`可以包含对模式组的向后引用,例如,如果`s = 'abc%1'`,这表示`s`将是`abc`与匹配于`pt`里标号为1的模式组的子字符串的连接。

如果参数`index`为正,仅替换第`index`个匹配的子字符串。如果`end`为正,替换将被局限于下标从`start`到`end`的子字符串里。

返回被替换的子字符串数目。

## 7 例子

```
s = 'abc123def456';

stdio.println( s.pfind( '%d+' ) );
stdio.println( s.match( '%d+' ) );
stdio.println( s.extract( '%d+' ) );
stdio.println( s.extract( '%d+', -1 ) );
stdio.println( s.extract( '%d+', 0 ) );
stdio.println( s.capture( '%a+)%d+' ) );

s.change( '%a+', '==%1==' );
stdio.println( s );

stdio.println( s.extract( '%d+', 1, '==%d+==' ) );
```

### Output

```
{ ( 3, 5 ), ( 9, 11 ) }
( 3, 5, 123 )
{ 123, 456 }
{ abc, def }
{ abc, 123, def, 456 }
{ abc123, abc, 123 }
==abc==123==def==456
{ 123 }
```