# 正则表达式

在编写处理网页文本的程序时，经常会有查找符合某些复杂规则的字符串的需要。正则表达式就是用于描述这些规则的工具。正则表达式是由普通字符（例如字符 a 到 z）以及特殊字符（称为“元字符”）组成的文字模式。模式用于描述在搜索文本时要匹配的一个或多个字符串。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。

## 4.2.1　基本语法与使用

正则表达式功能非常强大，但是学好并不是很困难。一些初学者总是感觉到正则表达式很抽象，看到稍微长的表达式直接选择放弃。接下来从一个新手的角度，由浅及深，配合各种示例来讲解正则表达式的用法。

### 1.入门小例子

学习正则表达式最好的办法就是通过例子。在不断解决问题的过程中，就会不断理解正则表达式构造方法的灵活多变。

例如我们想找到一篇英文文献中所有的 we 单词，你可以使用正则表达式：we，这是最简单的正则表达式，可以精确匹配英文文献中的 we 单词。正则表达式工具一般可以设置为忽略大小写，那 we 这个正则表达式可以将文献中的 We、wE、we 和 WE 都匹配出来。如果仅仅使用 we 来匹配，会发现得出来的结果和预想的不一样，类似于 well、welcome 这样的单词也会被匹配出来，因为这些单词中也包含 we。如何仅仅将 we 单词匹配出来呢？我们需要使用这样的正则表达式：\bwe\b。

#### “\b”

是正则表达式规定的一个特殊代码，被称为元字符，代表着单词的开头或结尾，也就是单词的分界处，它不代表英语中空格、标点符号、换行等单词分隔符，只是用来匹配一个位置，这种理解方式很关键。

假如我们看到 we 单词不远处有一个 work 单词，想把 we、work 和它们之间的所有内容都匹配出来，那么我们需要了解另外两个元字符

#### “.”和“\*”

正则表达式可以写为 \bwe\b.\*\bwork\b。**“.”这个元字符的含义是匹配除了换行符的任意字符，“\*”元字符不是代表字符，而是代表数量，含义是“\*”前面的内容可以连续重复任意次使得整个表达式被匹配。“.\*”整体的意思就非常明显了，表示可以匹配任意数量不换行的字符**，那么 \bwe\b.\*\bwork\b 作用就是先匹配出 we 单词，接着再匹配任意的字符（非换行），直到匹配到 work 单词结束。通过上面的例子，我们看到元字符在正则表达式中非常关键，元字符的组合能构造出强大的功能。

接下来咱们开始讲解常用的元字符，在讲解之前，需要介绍一个正则表达式的测试工具

#### Match Tracer

这个工具可以将写的正则表达式生成树状结构，描述并高亮每一部分的语法，同时可以检验正则表达式写的是否正确，如图4-23所示。

### 2.常用元字符

元字符主要有四种作用：有的用来匹配字符，有的用来匹配位置，有的用来匹配数量，有的用来匹配模式。在上面的例子中，我们讲到了“.”“\*”这两个元字符，还有其他元字符，如表4-1所示。

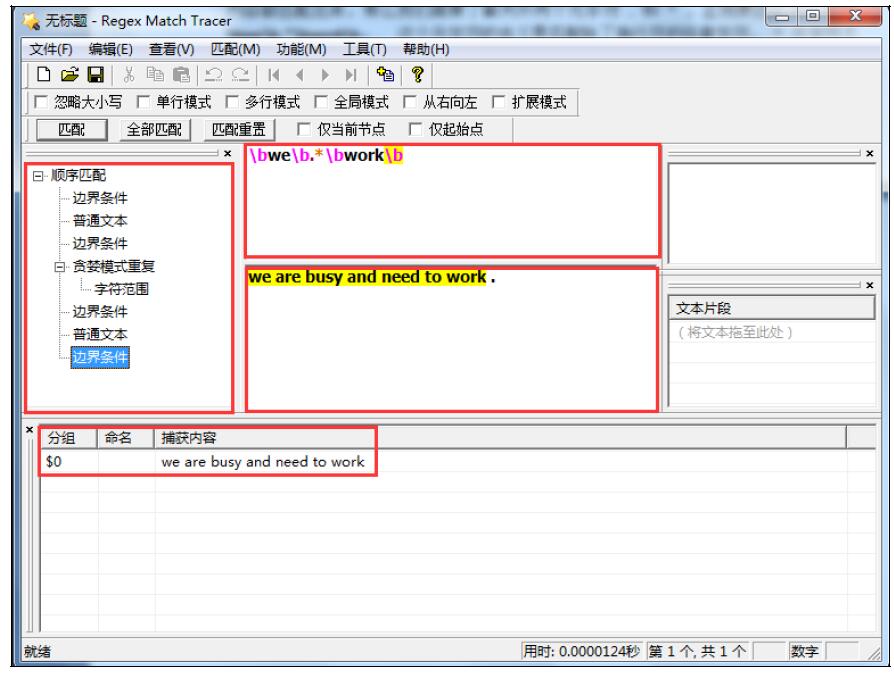
[](http://popimage/?src=)

图4-23　Match Tracer

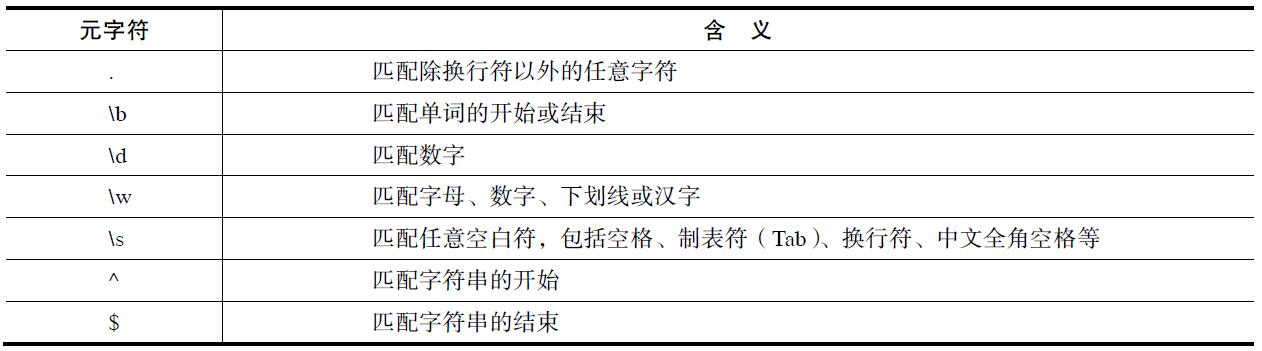
#### 表4-1　常见元字符

##### .除换行符以外任意字符

##### \d数字

##### \w字母、数字、下划线、或汉字

##### \s任意空白符

[](http://popimage/?src=)

上面的元字符是用来匹配字符和位置的，接下来讲解其他功能时，会依次列出匹配数量和模式的元字符。下面对上面列出的元字符使用一些小例子来进行一下练习。

假如一行文本为：we are still studying and so busy，我们想匹配出所有以u s 开头的单词，那么正则表达式可以写为：\bs\w\*\b。\bs\w\*\b 的匹配顺序：先是某个单词开始处（\b），然后是字母s，然后是任意数量的字母或数字（\w\*），最后是单词结束处（\b）。同理，如果匹配 s100 这样的字符串（不是单词），需要用到“^”和“$”，一个匹配开头，一个匹配结束，可以写为^s\d\*$。

### 3.字符转义

如果你想查找元字符本身的话，比如你查找“.”或者“\*”就会出现问题，因为它们具有特定功能，没办法把它们指定为普通字符。这个时候就需要用到转义，使用“\”来取消这些字符的特殊意义。因此如果查找“.”、“\”或者“\*”时，必须写成“\.”、“\\”和“\\*”。例如匹配 www.google.com 这个网址时，可以表达式可以写为 www\.google\.com。

### 4.重复{}

首先列举出匹配重复的限定符（指定数量的代码），如表4-2所示。

#### 表4-2　常用限定符

##### \*重复零次或更多次

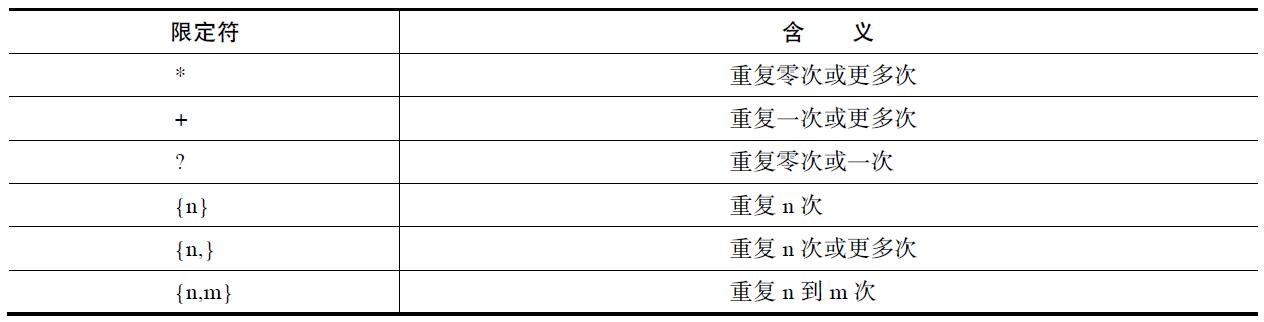
##### +重复一次或更多次

##### ?重复零次或一次

##### {n}重复n次

##### {n，}重复n次或更多次

##### {n，m}重复n到m次

[](http://popimage/?src=)

下面是一些重复的例子：

* hello\d+：匹配 hello 后面跟1个或更多数字，例如可以匹配 hello1、hello10 等情况。
* ^\d{5，12}$：匹配5到12个数字的字符串，例如 QQ 号符合要求。
* we\d：匹配 we 后面跟0个或者一个数字，例如 we、we0 符合情况。

### 5.字符集合[]

通过上面介绍的元字符，可以看到查找数字、字母或数字、空格是很简单的，因为已经有了对应这些字符的集合，但是如果想匹配没有预定义元字符的字符集合，例如匹配 a、b、c、d 和 e 中任意一个字符，这时候就需要自定义字符集合。正则表达式是通过[]来实现自定义字符集合，[abcde]就是匹配 abcde 中的任意一个字符，[.！]匹配标点符号（“.”、“”或“！”）。

除了将需要自定义的字符都写入[]中，还可以指定一个字符范围。[0-9]代表的含义与“\d”是完全一致的，代表一位数字；[a-z0-9A-Z\_]也完全等同于“\w”（只考虑英文），代表着26个字母中的大小写、0～9的数字和下划线中的任一个字符。

### 6.分支条件

正则表达式里的分支条件指的是有几种匹配规则，如果满足其中任意一种规则都应该当成匹配，具体方法是用“|”把不同的规则分隔开。例如匹配电话号码，电话号码中一种是3位区号，8位本地号，形如010-11223344，另一种是4位区号，7位本地号，形如0321-1234567。如果想把电话号码匹配出来，就需要用到分支条件：0\d{2}-\d{8}|0\d{3}-\d{7}。在分支条件中有一点需要注意，匹配分支条件时，将会从左到右地测试每个条件，如果满足了某个分支的话，就不会去再管其他条件了，条件之间是一种或的关系，例如从1234567890匹配出连续的4个数字或者连续8个数字，如果写成 \d{4}|\d{8}，其实\d{8}是失效的，既然能匹配出来8位数字，肯定就能匹配出4位数字。

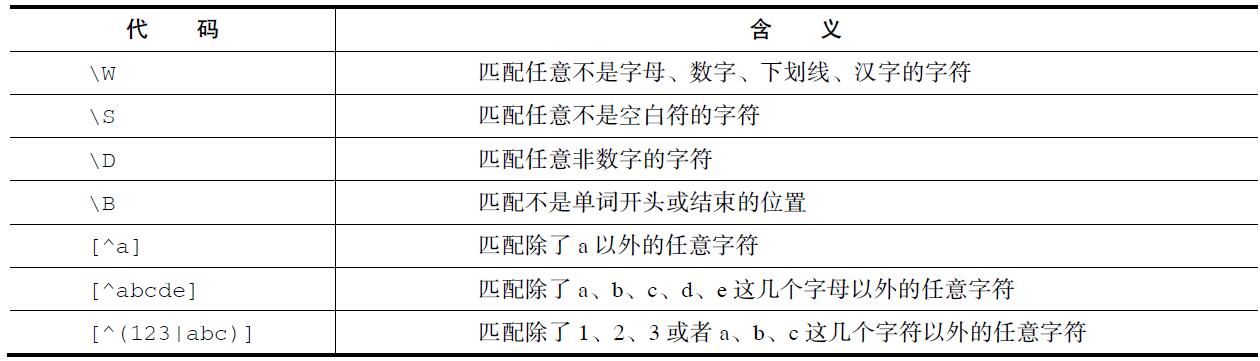
### 7.分组

先以简单的 IP 地址匹配为例子，想匹配类似192.168.1.1这样的 IP 地址，可以这样写正则表达式（（\d{1，3}）\.）{3}\d{1，3}。下面分析一下这个正则表达式：\d{1，3}代表着1～3位的数字，（（\d{1，3}）.）{3}代表着将1～3位数字加上一个“.”重复3次，匹配出类似192.168.1.这部分，之后再加上 \d{1，3}，表示1～3位的数字。但是上述的正则表达式会匹配出类似333.444.555.666这些不可能存在的 IP 地址，因为 IP 地址中每个数字都不能大于255，所以要写出一个完整的 IP 地址匹配表达式，还需要关注一下细节，下面给出一个使用分组的完整 IP 表达式：（（25[0-5]|2[0-4]\d|[0-1]\d{2}|[1-9]\d）.）{3}（（25[0-5]|2[0-4]\d|[0-1]\d{2}|[1-9]\d））。其中的关键是（25[0-5]|2[0-4]\d|[0-1]\d{2}|[1-9]\d）部分，大家应该有能力分析出来。

### 8.反义

有时需要查找除某一类字符集合之外的字符。比如想查找除了数字以外，包含其他任意字符的情况，这时就需要用到反义，如表4-3所示。

#### 表4-3　常用的反义

[](http://popimage/?src=)

例如“\D+”匹配非数字的一个或者多个字符。

### 9.后向引用

前面我们讲到了分组，使用小括号指定一个表达式就可以看做是一个分组。默认情况下，每个分组会自动拥有一个组号，规则是：从左向右，以分组的左括号为标志，第一个出现的分组的组号为1，第二个为2，以此类推。还是以简单的 IP 匹配表达式（（\d{1，3}）.）{3}\d{1，3}为例，这里面有两个分组1和2，使用 Match Tracer 这个工具可以很明显地看出来，如图4-24所示。

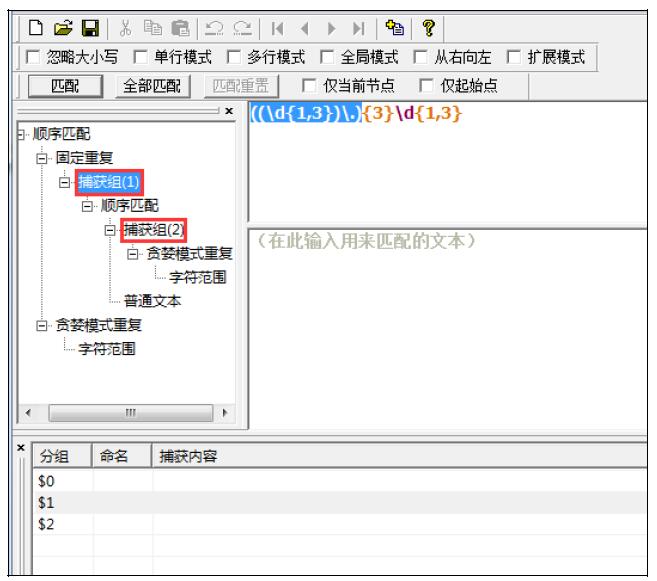
[](http://popimage/?src=)

图4-24　捕获组

如果想匹配类似192.168.1.1这种最后两位相等的 ip，上面的表达式可以改写成（（\d{1，3}）.）{3}\2。有一点要注意，引用的是内容而非模式。

你也可以自己指定子表达式的组名。要指定一个子表达式的组名，使用这样的语法：（\d+）或者（‘Digit’\d+）），这样就把“\d+”的组名指定为 Digit 了。要反向引用这个分组捕获的内容，你可以使用 \k，所以上面的 IP 匹配表达式写成（（\d{1，3}）.）{3}\k。使用小括号的地方很多，主要是用来分组，表4-4中列出了一些常用的形式。

#### 表4-4　常用分组形式

[](http://popimage/?src=)

在捕获这个表项里，我们讲解了前两种用法，还有（？：exp）没有进行讲解。（？：exp）不会改变正则表达式的处理方式，只是这样的组所匹配的内容不会像前两种那样被捕获到某个组里面，也不会拥有组号，这样做有什么意义？一般来说是为了节省资源，提高效率。比如说验证输入是否为整数，可以这样写^（[1-9][0-9]\*|0）$。这时候我们需要用到“（）”来限制“|”表示“或”关系的范围，但我们只是要判断规则，没必要把 exp 匹配的内容保存到组里，这时就可以用非捕获组了^（：\[1-9\]\[0-9\]\*|0）$。

### 10.零宽断言

##### (?=exp)匹配exp前面的位置

##### (?<=exp)匹配exp后面的位置

##### (?!exp)匹配后面不是exp的位置

##### (?<!exp)匹配前面不是exp的位置

在表4-4中，零宽断言总共有四种形式。前两种是正向零宽断言，后两种是负向零宽断言。什么是零宽断言呢？我们知道元字符“\b”、“^”匹配的是一个位置，而且这个位置需要满足一定的条件，我们把这个条件称为断言或零宽度断言。断言用来声明一个应该为真的事实，正则表达式中只有当断言为真时才会继续进行匹配。可能大家感到有些抽象，下面通过一些例子进行讲解。

首先说一下正向零宽断言的两种形式：

* （？=exp）叫零宽度正预测先行断言，它断言此位置的后面能匹配表达式 exp。比如[a-z]\*（？=exp）匹配以 ing 结尾的单词的前面部分（除了 ing 以外的部分），查找 I love cooking and singing 时会匹配出中的 cook 与 sing。先行断言的执行步骤应该是从要匹配字符的最右端找到第一个“ing”，再匹配前面的表达式，如无法匹配则查找第二个“ing”。
* （？<=exp）叫零宽度正回顾后发断言，它断言此位置的前面能匹配表达式 exp。比如（？<=abc）.\*匹配以 abc 开头的字符串的后面部分，可以匹配 abcdefgabc 中的 defgabc 而不是 abcdefg。通过比较很容易看出后发断言和先行断言正好相反：它先从要匹配的字符串的最左端开始查找断言表达式，之后再匹配后面的字符串，如果无法匹配则继续查找第二个断言表达式，如此反复。

再说一下负向零宽断言的两种形式：

* （？！exp）叫零宽度负预测先行断言，断言此位置的后面不能匹配表达式 exp。比如 \b（（？！abc）\w）+\b 匹配不包含连续字符串 abc 的单词，查找“abc123，ade123”这个字符串，可以匹配出 ade123，可以使用 Match Tracer 进行查看分析。
* （？<！exp）叫零宽度负回顾后发断言，断言此位置的前面不能匹配表达式 exp。比如（？<！[a-z]）\d{7}匹配前面不是小写字母的七位数字。还有一个复杂的例子：（？<=<（\w+）>）.\*（？=<\/\1>），用于匹配不包含属性的简单 HTML 标记内的内容。该表达式可以从 python 爬虫中提取出“python 爬虫”，这在 Python 爬虫开发中常用到。大家可以思考一下是如何提取出包含属性的 HTML 标记内的内容。

### 11.注释

正则表达式可以包含注释进行解释说明，通过语法（#comment）来实现，例如 \b\w+（#字符串）\b。要包含注释的话，最好是启用“忽略模式里的空白符”选项，这样在编写表达式时能任意地添加空格、Tab、换行，而实际使用时这些都将被忽略。

### 12.贪婪与懒惰

当正则表达式中包含能接受重复的限定符时，通常的行为是（在使整个表达式能得到匹配的前提下）匹配尽可能多的字符，这就是贪婪模式。以表达式 a\w+b 为例，如果搜索 a12b34b，会尽可能匹配更多的个数，最后就会匹配整个 a12b34b，而不是 a12b。但是如果想匹配出 a12b 怎么办呢？这时候就需要使用懒惰模式，尽可能匹配个数较少的情况，因此需要将上面的 a\w+b 表达式改为 a\w+?b，使用“”来启用懒惰模式。表4-5列举了懒惰限定符的使用方式。

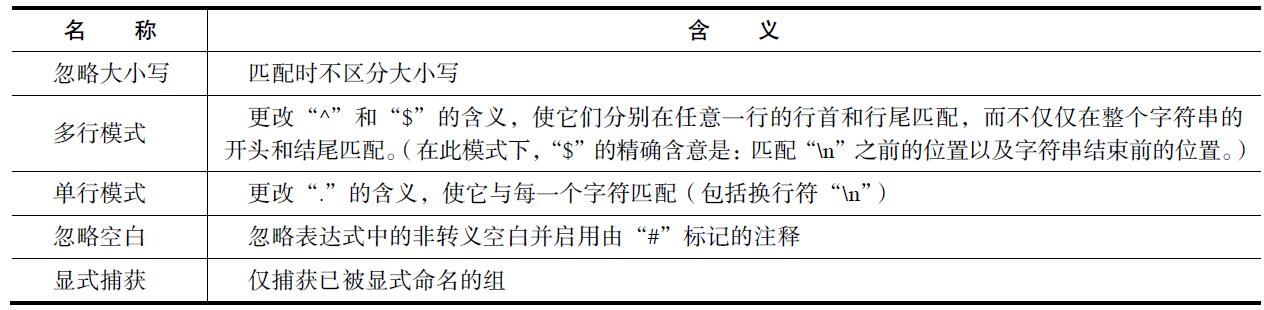
表4-5　懒惰限定符的使用方式

[](http://popimage/?src=)

### 13.处理选项

一般正则表达式的实现库都提供了用来改变正则表达式处理选项的方式，表4-6提供了常用的处理选项。

表4-6　常用的处理选项

[](http://popimage/?src=)

正则表达式中还有平衡组/递归匹配的概念，对于初学者来说，一般用不了这么复杂，此处不进行讲解。