正则表达式记录表

元字符

```
1、 \b : 单词的开始或结束

2、 .* : 任意数量的不包含换行的字符
    .  : 匹配除换行符以外的任意字符

3、 \d : 是新的元字符，匹配一位数字
  \d{n} : 必须连续重复匹配n次

4、 \s : 匹配任意的空白符（空格，制表格，换行符，中文全角空格等）

5、 \w : 匹配字母或者数字或者下划线或者汉字等

6、  ^  : 匹配字符串的开始
    ¥  : 匹配字符串的结束
```

注意：

```
 * 字符转义

 如果你想查找元字符本身，比如.和* ，这时候你就应该使用\.和\*
```

重复

代码	说明
*	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

注意：

*字符类

如果想匹配没有预定义的元字符集合，比如a，e，o。这时候你就要[aeo]就匹配任何一个英文元音的字母。[.?!]匹配任意一个标点符号

分枝条件

1. 满足任意一种规则都应该当成匹配，用 | 把不同的规则分割开来

   0\d{2}-\d{8}|0\d{3}-\d{7} 这个表达式能匹配两种以连字号分隔的电话号码：一种是三位区号，8位本地号(如010-12345678)，一种是4位区号，7位本地号(0376-2233445)

   \(0\d{2}\)[-]?\d{8}|0\d{2}[-]?\d{8}这个表达式匹配3位区号的电话号码，其中区号可以用小括号括起来，也可以不用，区号与本地号间可以用连字号或空格间隔，也可以没有间隔

2. 使用分枝条件时，要注意个分枝的顺序

   \d{5}-\d{4}|\d{5} 换成 \d{5}|\d{5}-\d{4}，那么就只会匹配5位的邮编。原因是匹配分枝条件时，将会从左到右地测试每个条件，如果满足了某个分枝的话，就不会去再管其它的条件了

3. 重复多个字符，使用分组

   (\d{1,3}\.){3}\d{1,3}是一个简单的IP地址匹配表达式

   (\d{1,3}\.){3}匹配三位数字加上一个英文句号(这个整体也就是这个分组)重复3次，最后再加上一个一到三位的数字(\d{1,3})

   不幸的是，它也将匹配256.300.888.999这种不可能存在的IP地址

   ((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)。解析正确IP：01-09,0-255
   1. 2[0-4]\d : (200-249)
   2. 25[0-5] :(250-255)
   3. [01]?\d\d? : (01-09,0-199)

反义

代码	语法
\W	匹配任意不是字母，数字，下划线，汉字的字符
\S	匹配任意不是空白符的字符
\D	匹配任意非数字的字符
\B	匹配不是单词开头或结束的位置
[^x]	匹配除了x以外的任意字符
[^aeiou]	匹配除了aeiou这几个字母以外的任意字符

例如：

`\S+`匹配不包含空白符的字符串。

`<a[^>]+>`匹配用尖括号括起来的以a开头的字符串

后向引用

常用分组语法

分类	代码/语法	说明
捕获	(exp)	匹配exp,并捕获文本到自动命名的组里
	(?exp	匹配exp,并捕获文本到名称为name的组里，也可以写成(?'name'exp)
	(?:exp)	匹配exp,不捕获匹配的文本，也不给此分组分配组号
零宽断言	(?=exp)	匹配exp前面的位置
	(?<=exp)	匹配exp后面的位置
	(?!exp)	匹配后面跟的不是exp的位置
	(?<!exp)	匹配前面不是exp的位置
注释	(?#comment)	这种类型的分组不对正则表达式的处理产生任何影响，用于提供注释让人阅读

注意：

分组0对应整个正则表达式
实际上组号分配过程是要从左向右扫描两遍的：第一遍只给未命名组分配，第二遍只给命名组分配－－因此所有命名组的组号都大于未命名的组号
你可以使用(?:exp)这样的语法来剥夺一个分组对组号分配的参与权

例如：

\b(\w+)\b\s+\1\b可以用来匹配重复的单词：html html

零宽断言

```
          <?!w+> ?<!w+
```

\b\w*q[^u]\w*\b匹配包含后面不是字母u的字母q的单词。

你会发现，如果q出现在单词的结尾的话，像Iraq,Benq，这个表达式就会出错。这是因为[^u]总要匹配一个字符，所以如果q是单词的最后一个字符的话，后面的[^u]将会匹配q后面的单词分隔符(可能是空格，或者是句号或其它的什么)，后面的\w*\b将会匹配下一个单词，于是\b\w*q[^u]\w*\b就能匹配整个Iraq fighting

负向零宽断言能解决这样的问题，因为它只匹配一个位置，并不消费任何字符。现在，我们可以这样来解决这个问题：\b\w*q(?!u)\w*\b

例如

\d{3}(?!\d)匹配三位数字，而且这三位数字的后面不能是数字；\b((?!abc)\w)+\b匹配不包含连续字符串abc的单词。

(?<![a-z])\d{7}匹配前面不是小写字母的七位数字。

(?<=<(\w+)>).*(?=<\/\1>)匹配不包含属性的简单HTML标签内里的内容：被尖括号括起来的单词(比如可能是)，然后是.*(任意的字符串),最后是一个后缀(?=<\/\1>)。注意后缀里的\/，它用到了前面提过的字符转义；\1则是一个反向引用，引用的正是捕获的第一组，前面的(\w+)匹配的内容，这样如果前缀实际上是的话，后缀就是了。整个表达式匹配的是和之间的内容(再次提醒，不包括前缀和后缀本身)

注释

代码	语法
(?<=	# 断言要匹配的文本的前缀
<(\w+)>	# 查找尖括号括起来的字母或数字(即HTML/XML标签)
(	# 前缀结束
.*	# 匹配任意文本
(?=	# 断言要匹配的文本的后缀
<\ / \1>	# 查找尖括号括起来的内容：前面是一个"/"，后面是先前捕获的标签
)	# 后缀结束

例如：
`2[0-4]\d(?#200-249)|25[0-5](?#250-255)|[01]?\d\d?(?#0-199)`。

贪婪和懒惰

1. a.*b，它将会匹配最长的以a开始，以b结束的字符串。如果用它来搜索aabab的话，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。它会匹配整个字符串aabab。这被称为贪婪匹配

2. a.*?b，它将会匹配最长的以a开始，以b结束的字符串。如果用它来搜索aabab的话，匹配任意数量的重复，(但是在能使整个匹配成功的前提下)使用最少的重复,会匹配aab和ab。这被称为懒惰匹配

   注意：

   > 为什么第一个匹配是`aab`（第一到第三个字符）而不是`ab`（第二到第三个字符）？
   > 简单地说，因为正则表达式有另一条规则，比懒惰 / 贪婪规则的优先级更高：
   > 最先开始的匹配拥有最高的优先权

3. 懒惰限定符

代码	说明
*?	重复任意次，但尽可能少重复
+?	重复1次或更多次，但尽可能少重复
？？	重复0次或1次，但尽可能少重复
{n,m}?	重复n到m次，但尽可能少重复
{n,}?	重复n次以上，但尽可能少重复

处理选项

注意:

```
Regex regex = new Regex(@"\ba\w{6}\b", RegexOptions.IgnoreCase);
```

常用的处理选项

名称	说明
IgnoreCase	匹配时不区分大小写
Multiline	更改^和$的含义，使它们分别在任意一行的行首和行尾匹配，而不仅仅在整个字符串的开头和结尾匹配。(在此模式下,$的精确含意是:匹配\n之前的位置以及字符串结束前的位置.)
Singleline	更改.的含义，使它与每一个字符匹配（包括换行符\n）
IgnorePatternWhitespace	忽略表达式中的非转义空白并启用由#标记的注释。
ExplicitCapture	仅捕获已被显式命名的组。

常用事例

说明	正则表达式
网址(url)	[a-zA-z]+://[^\s]*
IP地址	((2[0-4]\d
电子邮件(email)	\w+([-+.]\w+)*@\w+([-.]\w+)*.\w+([-.]\w+)*
QQ号码	[1-9]\d{4,}
HTML标签	<(.*)(.*)>.*<\/\1>
密码(有数字/大写字母/小写字母/标点，8位以上)	(?=^.{8,}$)(?=.*\d)(?=.*\W+)(?=.*[A-Z])(?=.*[a-z])(?!.*\n).*
日期	(\d{4}
汉字	[\u4e00-\u9fa5]
中文及全角标点符号	[\u3000-\u301e\ufe10-\ufe19\ufe30-\ufe44\ufe50-\ufe6b\uff01-\uffee]

注意:

ios 开发使用 RegexKit.framework 框架

使用规则：

```
 昵称验证：(4-8)位汉字
 + (BOOL) validateNickname:(NSString *)nickname
{
    NSString *nicknameRegex = @"^[\u4e00-\u9fa5]{4,8}$";

    NSPredicate *passWordPredicate = [NSPredicate predicateWithFormat:@"SELF MATCH
ES %@",nicknameRegex];

    return [passWordPredicate evaluateWithObject:nickname];
}

密码验证：(6-20)位英文不分大小写和数字
+ (BOOL) validatePassword:(NSString *)passWord
{
   NSString *passWordRegex = @"^[a-zA-Z0-9]{6,20}+$";

   NSPredicate *passWordPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHE
S %@",passWordRegex];

   return [passWordPredicate evaluateWithObject:passWord];
}

邮箱验证：
+ (BOOL) validateEmail:(NSString *)email
{
   NSString *emailRegex = @"[A-Z0-9a-z._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,4}";

   NSPredicate *emailTest = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", e
mailRegex];

   return [emailTest evaluateWithObject:email];
```

}