Atitit 正则表达式知识点简化

目录

[1. 定义 1](#_Toc18303)

[2. 用途 1](#_Toc21531)

[3. 使用法code 1](#_Toc11356)

[4. Tool 类 可视化工具 notepad++等编辑器ide 2](#_Toc11718)

[4.1. 类库 2](#_Toc15793)

[5. 3常用正则表达式 2](#_Toc31077)

[6. 4正则表达式语法 3](#_Toc19485)

# 定义

正则表达式被作为用来描述其称之为“正则集的代数”的一种表达式，因而采用了“正则表达式”这个术语。

正则表达式是一个特殊的字符序列，它能帮助你方便的检查一个字符串是否与某种模式匹配

正则表达式是一种描述字符串结果的语法规则，是一个特定的格式化模式，可以匹配、替换、截取匹配的字符串

正则表达式 ：：：简单理解为一个最简化的编程语言

# 用途

比如检测手机号码 电子邮箱

|  |  |  |
| --- | --- | --- |
|  | Js | java |
| 测试是否匹配 | RegExp对象 test() | Matcher matcher = pattern.matcher(str);      // 字符串是否与正则表达式相匹配      boolean rs = matcher.matches(); |
| 提取匹配信息 | RegExp。exec() | Matcher m = Pattern.matcher(str);  **while** (m.find()) { >>((m.group()));  } |
|  |  |  |
|  |  |  |
|  |  |  |

## 项目用途范例

搜索音乐的年份，从爬虫爬取的资料里面

提取人物电话从ocr出来的文本里面

# 使用法code

Java

Js

# Tool 类 可视化工具 notepad++等编辑器ide

## 类库

# 3常用正则表达式

比较容易记住的表达式 ，

，年代 4个数字，邮编，6个数字 ，qq号码 7-10个数字

手机号码 11个数字，1开头 。身份证 18位数字

邮箱 .\*@.\*

任意字符.\*

|  |  |
| --- | --- |
| 规则 | 正则表达式语法 |
| 一个或多个汉字 | ^[\u0391-\uFFE5]+$ |
| 邮政编码 | ^[1-9]\d{5}$ |
| QQ号码 | ^[1-9]\d{4,10}$ |
| 邮箱 | ^[a-zA-Z\_]{1,}[0-9]{0,}@(([a-zA-z0-9]-\*){1,}\.){1,3}[a-zA-z\-]{1,}$ |
| 用户名（字母开头 + 数字/字母/下划线） | ^[A-Za-z][A-Za-z1-9\_-]+$ |
| 手机号码 | ^1[3|4|5|8][0-9]\d{8}$ |
| URL | ^((http|https)://)?([\w-]+\.)+[\w-]+(/[\w-./?%&=]\*)?$ |
| 18位身份证号 | ^(\d{6})(18|19|20)?(\d{2})([01]\d)([0123]\d)(\d{3})(\d|X|x)?$ |

# 4正则表达式语法

|  |  |
| --- | --- |
| 元字符 | 描述 |
| \ | 将下一个字符标记符、或一个向后引用、或一个八进制转义符。例如，“\\n”匹配\n。“\n”匹配换行符。序列“\\”匹配“\”而“\(”则匹配“(”。即相当于多种编程语言中都有的“转义字符”的概念。 |
| ^ | 匹配输入字符串的开始位置。如果设置了RegExp对象的Multiline属性，^也匹配“\n”或“\r”之后的位置。 |
| $ | 匹配输入字符串的结束位置。如果设置了RegExp对象的Multiline属性，$也匹配“\n”或“\r”之前的位置。 |
| **\*** | **匹配前面的子表达式任意次。例如，zo\*能匹配“z”，“zo”以及“zoo”。\*等价于{0,}。** |
| + | 匹配前面的子表达式一次或多次(大于等于1次）。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。 |
| ? | 匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“do”或“does”中的“do”。?等价于{0,1}。 |
| **{n}** | **n是一个非负整数。匹配确定的n次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个o。** |
| {n,} | n是一个非负整数。至少匹配n次。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooood”中的所有o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o\*”。 |
| {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。例如，“o{1,3}”将匹配“fooooood”中的前三个o。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格。 |
| ? | 当该字符紧跟在任何一个其他限制符（\*,+,?，{n}，{n,}，{n,m}）后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串“oooo”，“o+?”将匹配单个“o”，而“o+”将匹配所有“o”。 |
| **.点** | **匹配除“\r\n”之外的任何单个字符。要匹配包括“\r\n”在内的任何字符，请使用像“[\s\S]”的模式。** |
| (pattern) | 匹配pattern并获取这一匹配。所获取的匹配可以从产生的Matches集合得到，在VBScript中使用SubMatches集合，在JScript中则使用$0…$9属性。要匹配圆括号字符，请使用“\(”或“\)”。 |
| (?:pattern) | 匹配pattern但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用或字符“(|)”来组合一个模式的各个部分是很有用。例如“industr(?:y|ies)”就是一个比“industry|industries”更简略的表达式。 |
| (?=pattern) | 正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如，“Windows(?=95|98|NT|2000)”能匹配“Windows2000”中的“Windows”，但不能匹配“Windows3.1”中的“Windows”。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。 |
| (?!pattern) | 正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如“Windows(?!95|98|NT|2000)”能匹配“Windows3.1”中的“Windows”，但不能匹配“Windows2000”中的“Windows”。 |
| (?<=pattern) | 反向肯定预查，与正向肯定预查类似，只是方向相反。例如，“(?<=95|98|NT|2000)Windows”能匹配“2000Windows”中的“Windows”，但不能匹配“3.1Windows”中的“Windows”。 |
| (?<!pattern) | 反向否定预查，与正向否定预查类似，只是方向相反。例如“(?<!95|98|NT|2000)Windows”能匹配“3.1Windows”中的“Windows”，但不能匹配“2000Windows”中的“Windows”。 |
| x|y | 匹配x或y。例如，“z|food”能匹配“z”或“food”或"zood"(此处请谨慎)。“(z|f)ood”则匹配“zood”或“food”。 |
| [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”。 |
| [^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”。 |
| [a-z] | 字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。  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. |
| [^a-z] | 负值字符范围。匹配任何不在指定范围内的任意字符。例如，“[^a-z]”可以匹配任何不在“a”到“z”范围内的任意字符。 |
| \b | 匹配一个单词边界，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就是匹配位置的）。例如，“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。“er\B”能匹配“verb”中的“er”，但不能匹配“never”中的“er”。 |
| \cx | 匹配由x指明的控制字符。例如，\cM匹配一个Control-M或回车符。x的值必须为A-Z或a-z之一。否则，将c视为一个原义的“c”字符。 |
| **\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]。 |
| \f | 匹配一个换页符。等价于\x0c和\cL。 |
| \n | 匹配一个换行符。等价于\x0a和\cJ。 |
| \r | 匹配一个回车符。等价于\x0d和\cM。 |
| \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]。 |
| \S | 匹配任何可见字符。等价于[^ \f\n\r\t\v]。 |
| \t | 匹配一个制表符。等价于\x09和\cI。 |
| \v | 匹配一个垂直制表符。等价于\x0b和\cK。 |
| \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。 |
| \W | 匹配任何非单词字符。等价于“[^A-Za-z0-9\_]”。 |
| \xn | 匹配n，其中n为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，“\x41”匹配“A”。“\x041”则等价于“\x04&1”。正则表达式中可以使用ASCII编码。 |
| \num | 匹配num，其中num是一个正整数。对所获取的匹配的引用。例如，“(.)\1”匹配两个连续的相同字符。 |
| \n | 标识一个八进制转义值或一个向后引用。如果\n之前至少n个获取的子表达式，则n为向后引用。否则，如果n为八进制数字（0-7），则n为一个八进制转义值。 |
| \nm | 标识一个八进制转义值或一个向后引用。如果\nm之前至少有nm个获得子表达式，则nm为向后引用。如果\nm之前至少有n个获取，则n为一个后跟文字m的向后引用。如果前面的条件都不满足，若n和m均为八进制数字（0-7），则\nm将匹配八进制转义值nm。 |
| \nml | 如果n为八进制数字（0-7），且m和l均为八进制数字（0-7），则匹配八进制转义值nml。 |
| \un | 匹配n，其中n是一个用四个十六进制数字表示的Unicode字符。例如，\u00A9匹配版权符号（&copy;）。 |
| \< \> | 匹配词（word）的开始（\<）和结束（\>）。例如正则表达式\<the\>能够匹配字符串"for the wise"中的"the"，但是不能匹配字符串"otherwise"中的"the"。注意：这个元字符不是所有的软件都支持的。 |
| \( \) | 将 \( 和 \) 之间的表达式定义为“组”（group），并且将匹配这个表达式的字符保存到一个临时区域（一个正则表达式中最多可以保存9个），它们可以用 \1 到\9 的符号来引用。 |
| | | 将两个匹配条件进行逻辑“或”（Or）运算。例如正则表达式(him|her) 匹配"it belongs to him"和"it belongs to her"，但是不能匹配"it belongs to them."。注意：这个元字符不是所有的软件都支持的。 |
| + | 匹配1或多个正好在它之前的那个字符。例如正则表达式9+匹配9、99、999等。注意：这个元字符不是所有的软件都支持的。 |
| ? | 匹配0或1个正好在它之前的那个字符。注意：这个元字符不是所有的软件都支持的。 |
| {i} {i,j} | 匹配指定数目的字符，这些字符是在它之前的表达式定义的。例如正则表达式A[0-9]{3} 能够匹配字符"A"后面跟着正好3个数字字符的串，例如A123、A348等，但是不匹配A1234。而正则表达式[0-9]{4,6} 匹配连续的任意4个、5个或者6个数字 |