**正则表达式**

**正则表达式在线可视化=https://jex.im/regulex/**

## 功能

* 检查文本中是否含有指定的特征词
* 找出文中匹配特征词的位置
* 从文本中提取信息，比如：字符串的子串
* 修改/替换文本

## 基本语法

**常见非打印字符**

* \n =换行 \r =回车 \r= 制表符 \s= 空白符(等价\n,\r,\t) \S =非空白符

**常见特殊字符(元字符)**

* \，转义=\\
* ^ =字符串开始位置，转义=\^
* $ =字符串结束位置，转义=\$
* () =标记子表达式开始和结束位置，转义=\(\)
* \* =前面表达式出现0次或多次，转义=\\*，等价{0,}
* + =前面表达式出现1次或多次，转义=\+，等价{1,}
* ? =前面表达式出现0次或1次，转义=\?，等价{0,1}
* . =匹配除换行符之外的单字符，转义=\.
* | =或逻辑，指明两项之间的一个选择，转义=\|
* [] =中括号表示单个字符，转义=\[\]
* {} =数量限定符

**常见元字符组合**

* {n} =出现n次 {n,}至少出现n次 {n,m}出现n到m次
* [a-z] =单个字符，包含a-z
* [^a-z] =单个字符，不包含a-z
* \b =单词边界，如er\b能匹配never，不能匹配very
* \B =非单词边界，如er\B能匹配very，不能匹配never
* \d =匹配单个数字字符，等价[0-9]
* \D =匹配单个非数字字符，等价[^0-9]
* \s =空白字符，包括空格、enter等
* \S =非空白字符
* \w =匹配包括下划线在内的单个字符，接近等价[a-zA-Z0-9\_]
* \W =匹配非单词字符，等价[^a-zA-Z0-9\_]
* ?=p =p前面的位置，如’hello’.replace(/?=l/g, ‘#’)，结果是he#l#lo
* ?!p =不是p前面的位置，如’hello’.replace(/?!p/g, ‘#’)，结果是#h#ell#o#

## 常见正则表达式

* 手机号码正则：^1[3456789]\d{9}$
* 邮箱地址正则：^[a-z0-9A-Z]+[-|a-z0-9A-Z.\_]+@([a-z0-9A-Z]+(-[a-z0-9A-Z]+)?\\.)+[a-z]{2,}$
* 网页地址正则：
* 中文正则(表示一个中文)：^[\u4e00-u9fa5]$
* 身份证：^(\d{15}|\d{17}([xX]|\d))$
* 千分位加逗号：’1123123123’.replace(/(?!^)(?=(\d{3})+$)/g, ‘,’)=’1,123,123,123’如果要求一个正则是匹配位置的话，那么所有的条件必须都是位置
* 驼峰改下划线：str.replace(/[A-Z]/g, ‘\_$1’).toLowerCase()
* 去掉字符串前后空格：str.replace(/(^\s\*)|(\s\*$)/g, ‘’)

## 常用技巧

* 分组：用括号表示分组()
* 分支结构：(a|b)
* 引用分组：如’2017-12-11’.replace(/(\d{4})-(\d{2})-(\d{2})/, ‘$1/$2/$3’)，结果是2017/12/11
* 反向引用：如/(\d{4})([-/.])(\d{2})\2(\d{2})/，其中\2表示在其前边的第二组正则，即([-/.])

## 在JavaScript中使用正则

### 正则表达式创建方式

* 字面量创建var reg = /pattern/flags，如var reg = /[0-9]{2}/
* 实例创建var reg = new RegExp(pattern, flags)，如var reg = new RegExp(‘\\d’)，或var reg = new RegExp(‘[\s]\*’, ‘g’)

注意：实例创建时正则特殊符号需要转义，如new RegExp(‘\\d’) = /\d/

其中pattern：正则表达式

flags：标识(修饰符)

标识主要包括：

i =忽略大小写匹配

m =多行匹配

g =全局匹配

### 正则的特性

* 贪婪性，每一次会尽可能多的捕获符合条件的内容。解决方式=量词字符后加？
* 懒惰性，成功捕获一次后返回，不管字符后面还有无符合条件字符。解决方式=加标识符g

var str = ‘123aaa456’;

var reg = /\d+/;

str.match(reg);// 因为贪婪性匹配多个、懒惰性匹配一次，会返回’123’

reg = /\d+?/g;// 去除贪婪性、懒惰性

str.match(reg);// 返回1,2,3,4,5,6

### 常用方法

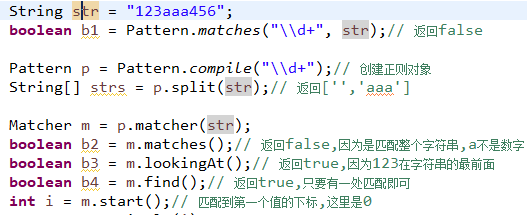
var reg = /[0-9]{1}/;

* test，如reg.test(str)返回true / false
* exec，如reg.exec(str)返回匹配到的字符串
* match，如str.metch(reg)返回匹配到的数组 / null，注意是str调用
* replace，如str.replace(reg, newStr)替换正则匹配结果，返回新字符串
* search，如str.search(reg)查找正则串，返回第一个匹配到的位置，为-1表示匹配不到
* split，如str.split(reg)用正则分割字符串，返回结果数组

## 在Java中使用正则

* Pattern类，使用正则表达式来构造
* Matcher类，使用Pattern来构造
* PatternSyntaxException类，正则匹配异常

实例如下



## 深入正则表达式引擎

**近代正则引擎都很类似但不完全一样，如.NET正则库 / JDK正则包，正则引擎分两种：**

* 文本导向引擎=DFA
* 正则导向引擎（最流行）=NFA