## 正则表达式-理论基础篇[实验楼](http://mp.weixin.qq.com/javascript:void(0);)

**什么是正则表达式?**

可以称作规则一个描述字符模式的对象，正则表达式中的字母和数字，都是按照字面含义进行匹配的。强大的字符串匹配工具

**如何建立一个正则表达式？**

1.使用RegExp()构造函数来创建RegExp对象

2.特殊的直接量语法来创建，定义为包含在一对斜杠（/）之间的字符

例：

var reg = /s$/;

var reg = new RegExp(s$);

两者效果完全相同，用来匹配所有以字母“s”结尾的字符串。

**“文本模式”的正则表达式语法**

**1.直接量字符**

非字母的字符匹配，这些字符需要通过反斜线（\）作前缀进行转义.

| **字 符** | **匹 配** |
| --- | --- |
| 字母和数字字符 | 自身 |
| \o | NUL字符 |
| \t | 制表符 |
| \n | 换行符 |
| \v | 垂直制表符 |
| \f | 换页符 |
| \r | 回车符 |
| \xnn | 由十六进制数nn指定的拉丁字符,例:\x0A等价于\n |
| \uxxxx | 由十六进制数xxxx指定的unicode字符,例:\u0009等价于\t |
| \cX | 控制字符^X,例:\cJ等价于换行符\n |

具有特殊含义的标点符号:

^, $, ., \*, +, ?, =, !, :, |, \, /,(,),[,],{, }

想在正则表达式中使用这些字符的直接量进行匹配，则必须使用前缀"\"，其他标点符号(比如@和引号)无特殊含义，在正则表达式中按照字面含义进行匹配。

**2.字符类**

方括号,或,范围

将直接量字符单独放进方括号内就组成了字符类。一个字符类可以匹配它包含的任意字符。字符类可以使用连字符来表示字符范围。

| **字 符** | **含 义** |
| --- | --- |
| [...] | 方括号内的任意字符 |
| [^...] | 不在方括号内的任意字符 |
| . | 除换行符合其他unicode行终止符之外的任意字符 |
| \w | 任何ASCII符组成的的单词,等同[a-zA-Z0-9] |
| \W | 任何不是ASCII符组成的的单词,等同[^a-zA-Z0-9] |
| \s | 任何unicode空白字符 |
| \S | 任何非unicode空白符的字符,注意\W与\s不同 |
| \d | 任何ASCII数字等价于[0-9] |
| \D | 除了ASCII数字之外的任何字符,等价于[^0-9] |
| [\b] | 退格直接量(特例) |

例:

/[a,b,c]/:表示和字母"a","b","c"中的任意一个都匹配

/[a-z]/:匹配拉丁字母表中的小写字母

/[\s\d]/:匹配任意空白符或数字

**3. 重复**

用以指定字符重复的标记

| **字 符** | **含 义** |
| --- | --- |
| {n,m} | 匹配前一项至少n次,但不能超过m次 |
| {n,} | 匹配前一项n次或者更多次 |
| {n} | 匹配前一项n次 |
| ? | 匹配前一项0次或者1次(也就是前一项是可选的,等价于{0,1}) |
| + | 匹配前一项1次或者多次(等价于{1,}) |
| \* | 匹配前一项0次或者多次,等价于{0,} |

例:

/\d{2,4}/:匹配2-4个数字。

/\w[3]/d?/:精确匹配三个单词和一个可选的数字。

/\s+java\s+/:匹配前后有1个或多个空格的字符串"java"

/[^(]\*/:匹配0个或多个非左括号的字符(这里实际上能与"("匹配)

在使用"\*",和"?"时要注意,由于这些字符能匹配0个字符,因此它们允许什么都不匹配，例:

/a\*/能匹配字符串"bbbb"，因其含有0个a。

上表中列出的匹配字符为"贪婪的"匹配，它总是尽可能多地匹配,且允许后续的正则表达式继续匹配。

非贪婪匹配,只需在待匹配的字符后跟随一个问号即可,"?",+?,\*?或{1,5}?。例:

/a+/:匹配"aaa"时,三个字符都被匹配,

/a+?/:匹配"aaa"时,因其尽可能少的匹配,只能匹配第一个a

**4.选择-分组和引用**

包括：指定选择项，自表达式分组，和引用前一子表达式的特殊字符

选择项的尝试匹配次序是从左到右，直到发现匹配项，若左边成功匹配就忽略右边的项。

正则表达式中圆括号的作用：

1.把单独的项组成子表达式，以便于可以像处理独立单元一样，用\|,\*,+或者?等等来对单元内的项进行处理

1. 在完整的模式中定义子模式

3.允许在同一正则表达式的后部,引用前面的子表达式

| **字 符** | **含 义** |  |
| --- | --- | --- |
| `"|" | (这里是竖线)选择,匹配的是该符号左边的子表达式或右边的子表达式 |  |
| (...) | 组合,将几个项组合成为一个单元,这个单元可通过`\,\*,+或者?`等符号,加以修饰,而且可以记住和这个相匹配的字符串以供以后的引用使用 |  |
| (?:..) | 只组合,把项组合到一个单元,但不记忆与该组相匹配的字符 |  |
| \n | 和第n个分组第一次匹配的字符相匹配,组是圆括号中的子表达式(也有可能嵌套的),组索引是从左到右的左括号数,"(?:形式的分组不编码" |  |

例:

/ab|cd|ef/:可以匹配"ab",也可以匹配"cd",还可以匹配"ef".

/\d{3}|[a-z]{4}/:可以匹配3个数字,或者4个小写字母.

(/[a-z]+(\d+)/):匹配一个或多个小写字母后跟随了一位或多位数字,使用括号包裹后,可以从检索到的匹配中抽取数字了.

**5.指定匹配位置**

我们称这些元素为正则表达式的锚,它们将模式定位在搜索字符串的特定位置上.

| **字 符** | **含 义** |
| --- | --- |
| ^ | 匹配字符串的开头,在多行检索中,匹配一行的开头 |
| $ | 匹配字符串的结尾,在多行检索中,匹配一行的结尾 |
| \b | 匹配一个单词边界,简言之,就是位于字符\w和\W之间的位置,或位于字符\w和字符串的开头或结尾之间的位置(需要注意:[\b]匹配的是退格符) |
| \B | 匹配非单词边界的位置 |

例:

/^JavaScript$/:匹配"JavaScript"单词

/\B[Ss]cript/:与"JavaScript"和"/postscript"匹配,但不与"script"与"Script".

**6.修饰符**

说明高级匹配的规则,放在"/"符号之外,第二条斜线后

| **字 符** | **含 义** |
| --- | --- |
| i | 执行不区分大小写的匹配 |
| g | 执行一个全局匹配,简言之,就是要找到所有的匹配,而不是在找到一个之后就停止 |
| m | 多行匹配模式,^匹配一行的开头和字符串的开头,$匹配行的结束和字符串的结束 |

例:

/java$/im:可以匹配"java"也可以匹配"Java\nis fun"

**用于模式匹配的string方法**

正则表达式的作用就是用来解析字符串

**1.search():**

查找字符串,接受一个参数为正则表达式,返回返回一个与之匹配的子串的起始位置,若未找到返回-1.

例:

JavaScript.search(/script/i);//返回4

search()方法不支持全局属性检索,所以会忽略修饰符g

**2. replace():**

用以执行检索与替换操作,接受两个参数,第一个参数是一个正则,第二个是要进行替换的字符串;

若第一个参数是字符串,replace()则会直接搜索这个字符串;

若替换字符串中出现了$加数字,那么replace()将用与指定的子表达式相匹配的文本来替换这两个字符.

例:

text.replace(/javascript/gi,"JavaScript");//将所有不区分大小写的javascript都替换成,大小写正确的JavaScript.  
  
//一段引用文本起始于引号,结束与引号.  
//中间的内容区域,不能包含引号  
var quote = /"([^"]\*)"/g;  
//引用中文半角引号替换英文引号,同时要保持引号之间的内容(储存在$1中)没有被修改.  
text.replace(quote,'“$1”');

**3. match()**

接受唯一一个参数,就是正则表达式,返回的是一个由匹配结果组成的数组.例:

"1 plus 2 equals 3".match(/\d+/g);//返回["1","2","3"]

若在这个正则表达式中没有设置修饰符g,match()就不会进行全局检索,它只检索第一个匹配。但即使match()执行的不是全局检索，它也返回一个数组。在这种情况下，数组的第一个元素就是匹配的字符串，余下的元素则是正则表达式中用圆括号括起来的子表达式。

因此，如果macth()返回一个数组a，那么a[0]存放的是完整的匹配，a[1]存放的是与第一个圆括号括起来的表达式相匹配的子串，以此类推。为了和方法replace()保持以后，a[n]存放的是$n的内容。

例:解析一个url

var url = /(\w+):\/\/([\w.]+)\/(\S\*)/;  
var text = "Visit my blog at http://www.example.com/~david";  
  
var result = text.match(url);  
  
if (result != null) {   
 var fullurl = result[0]; //包含 "http://www.example.com/"

var protocol = result[1]; //包含 "http"

var host = result[2]; //包含 "www.example.com"

var path = result[3]; //包含~david  
 }

**4. split()**

用以将调用它的字符串拆分为子串组成的数组,使用分隔符是split()的参数

参数为字符串:

"123,456,789".split(",");//返回["123","456","789"]

参数为正则:

"1,2, 3,4 ,5".split(/\s\*,\s\*/)//返回["1","2"," 3","4 ","5"]

指定分隔符",",允许两边可以留有任意多的空白符

**RegExp对象**

RegExp对象用以创建新的RegExp对象,可接受两个参数,第二个可选.

RegExp对象的第一个参数为正则表达式的主体部分,也就是/...../之间的文本,无论是字符串直接量,还是正则表达式都使用"\"作为转义字符的前缀.

第二个可选参数为指定正则表达式的修饰符,i,g,m或者他们的组合.参考:文本模式的则表达式-修饰符;

例:

var reg = new RegExp("\\d{5}","g");//匹配全局模式下的5个数字  
alert(reg.test("1J2a35786"));//true  
alert(reg.test("1J2a356"));//false

**RegExp属性**

| **属性名** | **类型** | **含义** |
| --- | --- | --- |
| source | 只读字符串 | 包含正则表达式的文本 |
| global | 只读布尔值 | 用以说明正则表达式是否带有修饰g |
| ignoreCase | 只读布尔值 | 用以说明正则表达式是否带有修饰i |
| multiline | 只读布尔值 | 用以说明正则表达式是否带有修饰m |
| lastIndex | 可读性的整数 | 若匹配模式中含有g,这个属性储存在整个字符串中下一次检索的开始位置 |