# 认识正则表达式

项目开发中，经常需要对表单中输入内容的文本框进行格式限制。例如，用户名、密码、手机号、身份证号的验证，这些内容遵循的规则繁多而又复杂，如果要匹配成功，可能需要进行多次的条件判断，这种做法显然不可取。此时，就需要实现正则表达式，利用最简短的描述语法完成查找、匹配、替换等功能。

## 什么是正则表达式

正则表达式（Regular Expression，简称RegExp）是一种描述字符串结构的语法规则，是一个特定的格式化操作，用于验证各种字符串是否匹配这个特征，进行实现高级的文本查找、替换、截取内容等操作。在项目开发中，手机号码指定位数的隐藏、数据采集、敏感词的过滤以及表单的验证等功能，都可以利用正则表达式来完成。

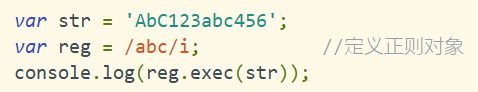
## 如何使用正则

在JavaScript中，可以使用RegExp对象提供的exec( )方法和String对象提供的match( ) 方法在一个指定字符串中执行匹配。

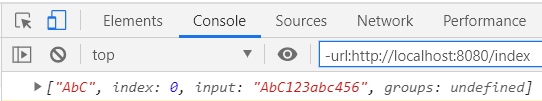
1. **RegExp对象的exec( )方法**

exec( )方法用于在目标字符中搜索匹配，一次仅返回一个匹配结果。其语法格式为：  
 **正则对象 . exec( str ) ;**

具体示例如下：



“/abc/i”中的“/”是正则表达式的定界符，“abc”表示正则表达式的模式文本，“i”是模式修饰符，表示忽略字母大小写。exec( )方法的参数时待匹配的字符串，匹配成功时，该方法返回值是一个数组，否则返回null。查看控制台输出：



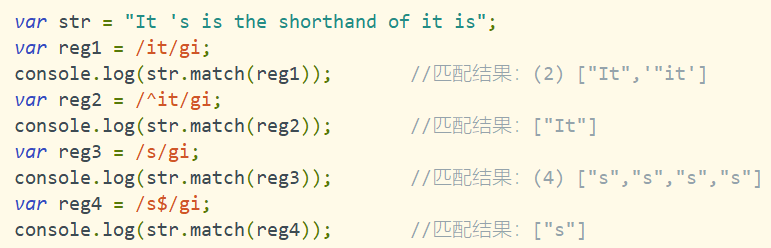
返回结果中，数组中的第一个元素表示匹配到的字符串；第二个元素index表示匹配到的字符串位于目标字符串中的索引值；第三参数input表示目标字符串。

1. **String对象的match( )方法**

String对象中的match( )除了可在字符串内检索指定的值外，还可以在目标字符窗中根据正则匹配出所有符合要求的内容，匹配成功后将其保存到数组中，匹配失败则返回false。其语法格式为：

**str . match( 正则对象 ) ;**

具体示例如下：



上述代码中，定位符“^”和“$”用于确定字符在字符串中的位置，前者可用于匹配字符串开始的位置，后者用于匹配字符串结尾的位置。其中，g表示全局匹配，用于在找到第一个匹配之后仍然继续查找。

## 创建正则对象

在JavaScript中，使用正则之前首先需要创建正则对象。创建正则对象的方式有两种：一是通过字面量的方式进行创建，二是通过RegExp对象的构造函数的方式创建。

1. 使用字面量方式创建正则对象的语法如下：

**/pattern/flags**

1. 使用RegExp对象的构造函数的方式创建语法如下：

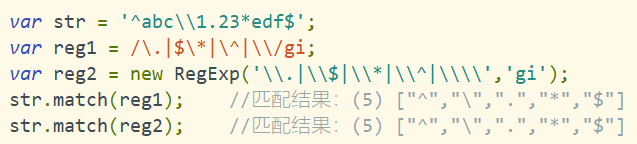
**new RegExp( pattern [ ,flags ] )**

其中，pattern是由元字符和文本字符组成的正则表达式模式文本，元字符是具有特殊含义的字符，如“^”，“.”或“\*”等，文本字符就是普通的文本，如字母和数字等。flags表示模式的修饰符，用于进一步对正则表达式进行设置。可选值如下：

|  |  |
| --- | --- |
| 模式符 | 说明 |
| g | 用于在目标字符串中实现全局匹配 |
| i | 匹配时忽略大小写 |
| m | 实现多行匹配 |
| u | 以Unicode编码执行正则表达式 |
| y | 粘性匹配，进匹配目标字符串中此正则表达式的lastIndex属性的索引 |

模式修饰符可以根据实际需求多个组合在一起使用。例如，既要忽略大小写又要进行全局匹配，可以直接使用gi，并且在编写多个模式修饰符没有顺序要求。

示例：匹配特殊字符“^”、“$”、“\*”、“.”和“\”。



* 上述代码中，正则对象中的选择符“|”可以理解为“或”，经常用于查找的条件有多个时，只要其中一个条件满足即可成立的情况。
* 由于JavaScript中的字符串存在转义问题，因此代码中str里的“\\”表示反斜线“\”。

同时，在正则中匹配特殊字符串时，也需要反斜线（\）对特殊字符串进行转义。

* 值得一提的是，构造函数方式与字面量方式创建的正则对象，虽然在功能上完全一致，但它们在语法实现上有一定的区别，前者的pattern在使用时需要对反斜杠（\）进行转义，例如，上述示例中匹配特殊字符“.”时，除了需要对特殊字符转义，还需要再添加一个“\”对反斜杠进行转义，因此最后的正则表达式模式为“\\.”；而后者的pattern在编写时，要放在定界符“/”内，flags标记则放在结尾定界符之外。

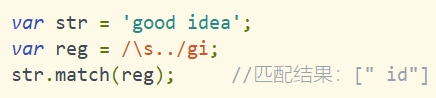
# 字符类别与集合

## 字符类别

JavaScript中给出的字符类别可以轻松的完成某些正则匹配。例如，大小写字母和数字 可以使用‘\w’表示；若要匹配0~9的数字可以使用‘\d’表示。常用的字符类别如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 字符 | 含义 | 字符 | 含义 |
| . | 匹配除“\n”外的任何单个字符 | \f | 匹配一个换页符（form-feed） |
| \d | 匹配任意一个数字（0~9） | \D | 匹配任意一个非数字字符 |
| \s | 匹配一个空白符，包括空格、制表符、换页符、换行符等 | \S | 匹配一个非空白符 |
| \w | 匹配任意一个字母（大小写）、数字和下划线 | \W | 匹配任意一个非字母（大小写）、数字和下划线的字符 |
| \b | 匹配单词分界符。如‘\bg’匹配‘best grade’，结果为‘g’ | \B | 非单词分界符。如‘\Bede’匹配‘best grade’，结果为‘ade’ |
| \t | 匹配一个水平制表符（tab） | \v | 匹配一个垂直制表符（vertical tab） |
| \n | 匹配一个换行符（linefeed） | \r | 匹配一个回车符（carriage return） |

为了读者理解字符类别的使用，以“.”和“\s”为例进行演示。具体示例如下：



上述代码中，正则对象reg用于匹配空白符后的任意两个字符（除换行外）。因此在控制台看到的结果中，id前有一个空格。

## 字符集合

正则表达式中的“[ ]”可以实现一个字符集合。与连字符“-”一起使用时，表示匹配 指定范围内的字符；与元字符“^”一起使用时，表示匹配指定范围外的字符。

以字符串**“ 'get好TB6’ . match ( /pattern/flags ) ”**为例进行匹配：

|  |  |  |
| --- | --- | --- |
| pattern | 说明 | 匹配结果 |
| [ cat ] | 匹配字符集合中的任意一个字符c、a、t | [ “t” ] |
| [ ^cat ] | 匹配c、a、t以外的字符 | 1. [ “g” , “e” , “好” , “T” , “B” , “6” ] |
| [ B-Z ] | 匹配字母B~Z范围内的字符 | 1. [ “T” , “B” ] |
| [ ^a-z ] | 匹配字母a~z范围外的字符 | (4) [ “好” , “T” , “B” , “6” ] |
| [ a-zA-Z0-9 ] | 匹配大小写字符和0~9范围内的字符 | (7) [ “g” , “e” , “t” , “T” , “B” , “6” ] |
| [ \u4e00-\u9fa5 ] | 匹配任意一个中文字符 | [ “好” ] |

需要注意的是，字符“-”在通常情况下只表示一个普通字符，只有在表示字符范围时才作为元字符使用。

# 字符限定与分组

## 限定符

项目开发中，若需要匹配一个连续出现的字符，如6个连续出现的数字。可创建如下正则对象：



以上方式虽然可以实现，但是重复出现的“\d”既不便于阅读，书写又繁琐，此时可以使用限定符（？、+、\*、{ }）完成某个字符连续出现的匹配。

完成对于重复出现的字符可以使用限定符（ ？、+、\*、{ } ）表示，完成某个字符连续出现的匹配。常用的限定符有：

|  |  |  |  |
| --- | --- | --- | --- |
| 字符 | 说明 | 示例 | 结果 |
| ？ | 匹配前面的字符零次或一次 | hi?t | 可匹配ht和hit |
| + | 匹配前面的字符一次或多次 | bre+ad | 可匹配范围从bread+bre…ad |
| \* | 匹配前面的字符零次或多次 | ro\*se | 可匹配范围从rse到ro…se |
| { n} | 匹配前面的字符n次 | hit{ 2}er | 只能匹配hitter |
| { n, } | 匹配前面的字符最少n次 | hit{ 2 ,}er | 可匹配范围从hitter到hitt…er |
| { n , m} | 匹配前面的字符最少n次，最多m次 | fe{ 0 ,2}l | 可匹配fl、fel和feel三种情况 |

限定符的使用对象是限定符前面离限定符最近的单个字符。如hi?是对i的限定。

示例：匹配6个连续出现的数字：



## 贪婪与惰性匹配

当“.”字符和限定符连用时，可以实现匹配指定数量范围内的任意字符。例如 “^hello.\*world$”表示匹配以‘hello’开始、‘world’结束的字符串。

正则表达式在实现指定数量范围的任意字符匹配时，支持贪婪匹配和惰性匹配两种匹配方式。贪婪匹配表示匹配尽可能多的字符。惰性匹配表示尽可能匹配少的字符。默认情况下是贪婪匹配，若要实现惰性匹配，需要在上一个限定符的后面加上“？”符号，具体示例如下：

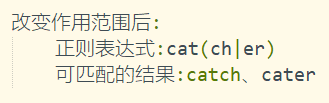
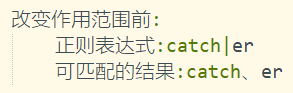


从上述代码的执行结果可以看出，贪婪匹配，会获取最先出现的w到最后出现的b，即可获得匹配结果为“webWEBWebwEb”；惰性匹配，会获取最先出现的w到最先出现的b，即可获取匹配结果“web”。

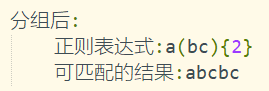
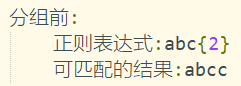
## 括号字符

在正则表达式中，被括号“（）”括起来的内容，称为“子表达式”。子表达式的作用有 两个作用：一是改变限定符的作用范围，二是分组。

1. 改变限定符的作用范围



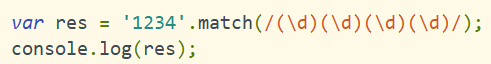
1. 分组



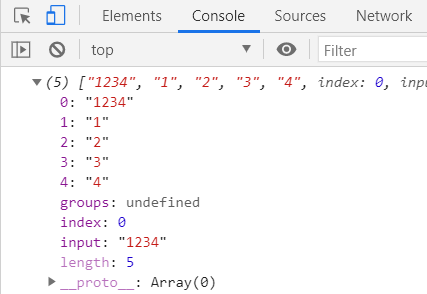
## 捕获与非捕获

正则表达式中，当子表达式匹配到相应的内容时，系统会自动捕获这个匹配的行为，然后将子表达式匹配到的内容存储到系统的缓存区中，这个称之为“捕获”。

1. 在利用match( )方法进行捕获时，其返回结果中会包含子表达式的匹配结果，具体示例如下：

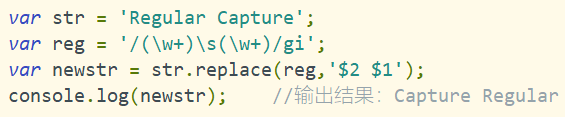


在浏览器的控制台查看捕获结果：



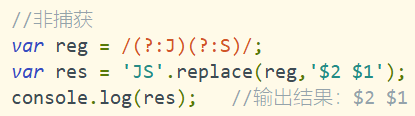
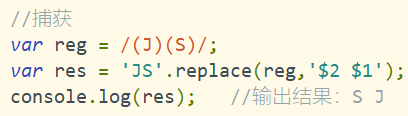
下标为1的元素保存第1个子表达式的捕获内容，下标为2的元素保存第2个子表达式的捕获内容，以此类推，即可得到所有的捕获内容。

1. 另外，还可以通过String对象的replace( )方法，直接利用$n（n是大于0的正整数）的方式捕获内容，完成对子表达式捕获的内容进行替换的操作。以颠倒“Regular Capture”中两个单词的顺序为例进行演示：



上述代码中，replace( )方法的第1个参数为正则表达式，用于与str字符串进行匹配，将符合规则的内容利用第2个参数设置的内容进行替换。其中，$2表示reg正则表达式中第2个子表达式被捕获的内容“Capture”，$1表示第1个子表达式被捕获的内容“Regular”。replace( )方法的返回值是替换后的新字符串，因此，并不会改变原字符串的内容。

1. 除此之外，若有在开发中不想将子表达式的匹配内容存放到系统的缓存中，则可以使用“( ? : x )”的方式实现非捕获匹配。捕获与非捕获的实现对比如下所示：

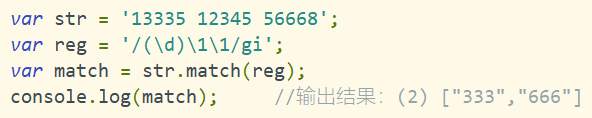


从上述代码中可以看出，捕获后可以通过$n的方式获取到子表达式匹配到的内容，而非捕获后，不能通过其他的方式获取子表达式匹配到的内容。

## 反向引用

在编写正则表达式时，若要在正则表达式中，获取存放在缓存区内的子表达式的捕获内容，则可以使用“\n”（n是大于0的正整数）的方式引用，这个过程就是“反向引用”。其中，“\1”表示第1个子表达式的捕获内容，“\2”表示第2个子表达式的捕获内容，依次类推。

示例：以查找连续3个相同的数字为例



“\d”用于匹配0~9之间的任意一个数字，为其添加圆括号“（）”后，即可通过反向引用获取捕获的内容。因此，最后的匹配结果为333和666。（我在学习中编码实现后的实际输出结果是null，对此，我存在异议）

## 零宽断言

零宽断言指的是一种零宽度的子表达式匹配，用于查找子表达式匹配的内容之前或之后是否含有特定的字符集。它分为正向预查和反向预查，但是在JavaScript中仅支持正向预查，即匹配含有或不含有捕获内容之前的数据，匹配的结果中不含捕获的内容。具体字符与示例如下表：

|  |  |  |
| --- | --- | --- |
| 字符 | 说明 | 示例 |
| x(?=y) | 仅当x后面紧跟着y时，才匹配x | Countr(?=ylies)用于匹配Country或Countries中的Countr |
| x(?!y) | 仅当x后不紧跟y时才匹配x | Countr(?!ylies)用于匹配Countr后不是y或ies的任意字符串中的Countr |

## 正则运算符优先级

正则表达式中运算符有很多。在实际应用时，各种运算符会遵循优先级顺序进行匹配。正则表达式中常用运算符优先级，由高到低的顺序如下表所示：

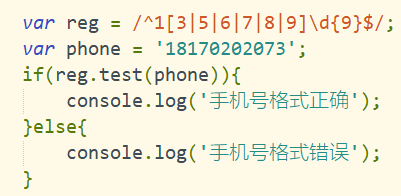
|  |  |
| --- | --- |
| 运算符 | 说明 |
| \ | 转义符 |
| ( )、(?:)、(?=)、[] | 括号和中括号 |
| \*、+、?、{n}、{n , }、{n , m } | 限定符 |
| ^、$、\任何元字符、任何字符 | 定位点或序列 |
| | | “或”操作 |

# 与正则相关的方法

JavaScript中除了前面讲解的RegExp类中的exec( )方法，String类中的match( )和replace( )方法外，RegExp类和String类中还有一些在开发中较为常用的方法。

## RegExp类中的方法

开发中，若无需获取正则与字符串匹配的结果，是需要检测正则表达式与指定的字符串是否匹配，则可以利用RegExp类提供的test( )方法。具体使用示例如下：



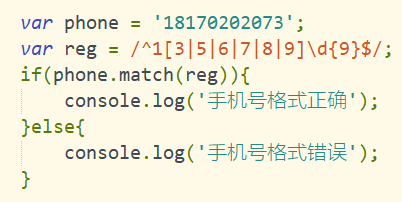
上述示例演示了验证手机号格式是否正确，如果匹配成功，则返回true，否则返回false。

## String类中的方法

在JavaScript中，除了String类的match( )和replace( )方法外，还有search( )和split( )方法可以根据正则进行相关的操作，但为了巩固学习，将match( )和repalce( )再讲一遍，因为它们也经常使用，特别是match( )。

1. **match( )方法**

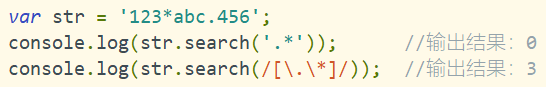
match( )方法的作用与RegExp类中的test( )方法的作用相同，用于判断某一字符串是否匹配指定的正则表达式。以验证手机号为例，如下：



匹配成功返回true，否则返回false

**2、search( )方法**

search( )方法可以返回指定模式的子串在字符串首次出现的位置，相对于indexOf( )方法来说功能更强大。具体示例如下：



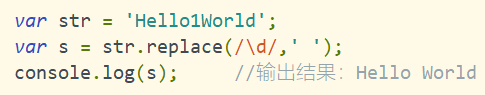
·search( )方法的参数是一个正则对象，如果传入一个非正则表达式对象，则会使

用“new RegExp(传入的参数)”隐式地将其转换为正则表达式对象。

·search( )方法如果匹配成功则返回索引下标，匹配失败的返回值为-1。

**3、replace( )方法**

replace方法可以将符合正则表达式的内容进行替换。示例如下：



将数字替换为空格。

**4、split( )方法**

split( )方法用于根据指定的分隔符将一个字符串分割成字符串数组，其分割后的字符串数组中不包括分割符。当分隔符不只一个时，需要定义正则对象才能够完成字符串的分割操作。

1. 按照规则分割



·split( )方法的参数为正则表达式模式设置的分隔符，返回值是以数组形式保存

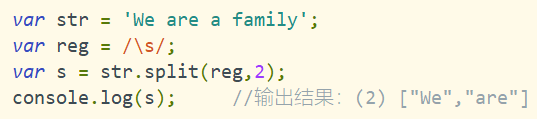
的分割后的结果，结果中不包含分割符。

·需要注意的是，当字符串为空时，split( )方法返回的是一个包含一个空字符的

数组“[“”]”，如果字符串和分隔符都是空字符串，则返回一个空数组“[]”。

1. 指定分割次数

在使用正则匹配方式分割字符串时，还可以指定字符串分隔的次数。具体示例如下：



当指定字符串分割次数后，若指定的次数小于实际字符串中符合规则分割的次数，则最后的返回结果中会忽略其他的分割结果。