**第 10 章 正则表达式**

学习要点：

1.什么是正则表达式

2.创建正则表达式

3.获取控制

4.常用的正则

**主讲教师：李炎恢**

**合作网站：**[**http://www.ibeifeng.com**](http://www.ibeifeng.com/)

**讲师博客：**[**http://hi.baidu.com/李炎恢**](http://hi.baidu.com/%E6%9D%8E%E7%82%8E%E6%81%A2)

假设用户需要在 HTML 表单中填写姓名、地址、出生日期等。那么在将表单提交到服 务器进一步处理前，JavaScript 程序会检查表单以确认用户确实输入了信息并且这些信息是 符合要求的。

## 一．什么是正则表达式

正则表达式(regular expression)是一个描述字符模式的对象。ECMAScript 的 RegExp 类 表示正则表达式，而 String 和 RegExp 都定义了使用正则表达式进行强大的模式匹配和文本 检索与替换的函数。

正则表达式主要用来验证客户端的输入数据。用户填写完表单单击按钮之后，表单就会 被发送到服务器，在服务器端通常会用 PHP、ASP.NET 等服务器脚本对其进行进一步处理。 因为客户端验证，可以节约大量的服务器端的系统资源，并且提供更好的用户体验。

**二．创建正则表达式** 创建正则表达式和创建字符串类似，创建正则表达式提供了两种方法，一种是采用 new

运算符，另一个是采用字面量方式。

1.两种创建方式

var box = new RegExp('box'); //第一个参数字符串

var box = new RegExp('box', 'ig'); //第二个参数可选模式修饰符

模式修饰符的可选参数

|  |  |
| --- | --- |
| 参 数 | 含 义 |
| i | 忽略大小写 |
| g | 全局匹配 |
| m | 多行匹配 |

var box = /box/; //直接用两个反斜杠

var box = /box/ig; //在第二个斜杠后面加上模式修饰符

2.测试正则表达式

RegExp 对象包含两个方法：test()和 exec()，功能基本相似，用于测试字符串匹配。test() 方法在字符串中查找是否存在指定的正则表达式并返回布尔值，如果存在则返回 true，不存 在则返回 false。exec()方法也用于在字符串中查找指定正则表达式，如果 exec()方法执行成 功，则返回包含该查找字符串的相关信息数组。如果执行失败，则返回 null。

RegExp 对象的方法

|  |  |
| --- | --- |
| 方 法 | 功 能 |
| test | 在字符串中测试模式匹配，返回 true 或 false |
| exec | 在字符串中执行匹配搜索，返回结果数组 |

/\*使用 new 运算符的 test 方法示例\*/

var pattern = new RegExp('box', 'i'); //创建正则模式，不区分大小写 var str = 'This is a Box!'; //创建要比对的字符串

alert(pattern.test(str)); //通过 test()方法验证是否匹配

/\*使用字面量方式的 test 方法示例\*/

var pattern = /box/i; //创建正则模式，不区分大小写 var str = 'This is a Box!';

alert(pattern.test(str));

/\*使用一条语句实现正则匹配\*/

alert(/box/i.test('This is a Box!')); //模式和字符串替换掉了两个变量

/\*使用 exec 返回匹配数组\*/ var pattern = /box/i;

var str = 'This is a Box!';

alert(pattern.exec(str)); //匹配了返回数组，否则返回 null PS：exec 方法还有其他具体应用，我们在获取控制学完后再看。

3.使用字符串的正则表达式方法

除了 test()和 exec()方法，String 对象也提供了 4 个使用正则表达式的方法。

String 对象中的正则表达式方法

|  |  |
| --- | --- |
| 方 法 | 含 义 |
| match(pattern) | 返回 pattern 中的子串或 null |
| replace(pattern, replacement) | 用 replacement 替换 pattern |
| search(pattern) | 返回字符串中 pattern 开始位置 |
| split(pattern) | 返回字符串按指定 pattern 拆分的数组 |

/\*使用 match 方法获取获取匹配数组\*/

var pattern = /box/ig; //全局搜索 var str = 'This is a Box!，That is a Box too';

alert(str.match(pattern)); //匹配到两个 Box,Box

alert(str.match(pattern).length); //获取数组的长度

/\*使用 search 来查找匹配数据\*/ var pattern = /box/ig;

var str = 'This is a Box!，That is a Box too';

alert(str.search(pattern)); //查找到返回位置，否则返回-1 PS：因为 search 方法查找到即返回，也就是说无需 g 全局

/\*使用 replace 替换匹配到的数据\*/ var pattern = /box/ig;

var str = 'This is a Box!，That is a Box too';

alert(str.replace(pattern, 'Tom')); //将 Box 替换成了 Tom

/\*使用 split 拆分成字符串数组\*/ var pattern = / /ig;

var str = 'This is a Box!，That is a Box too';

alert(str.split(pattern)); //将空格拆开分组成数组

## RegExp 对象的静态属性

|  |  |  |
| --- | --- | --- |
| 属 性 | 短 名 | 含 义 |
| input | $\_ | 当前被匹配的字符串 |
| lastMatch | $& | 最后一个匹配字符串 |
| lastParen | $+ | 最后一对圆括号内的匹配子串 |
| leftContext | $` | 最后一次匹配前的子串 |
| multiline | $\* | 用于指定是否所有的表达式都用于多行的布尔值 |
| rightContext | $' | 在上次匹配之后的子串 |

**/\*使用静态属性\*/**

var pattern = /(g)oogle/;

var str = 'This is google！';

pattern.test(str); //执行一下

alert(RegExp.input); //This is google！

alert(RegExp.leftContext); //This is

alert(RegExp.rightContext); //！

alert(RegExp.lastMatch); //google

alert(RegExp.lastParen); //g

alert(RegExp.multiline); //false

PS：Opera 不支持 input、lastMatch、lastParen 和 multiline 属性。IE 不支持 multiline 属 性。

所有的属性可以使用短名来操作

RegExp.input 可以改写成 RegExp['$\_']，依次类推。但 RegExp.input 比较特殊，它还可 以写成 RegExp.$\_。

## RegExp 对象的实例属性

|  |  |
| --- | --- |
| 属 性 | 含 义 |
| global | Boolean 值，表示 g 是否已设置 |
| ignoreCase | Boolean 值，表示 i 是否已设置 |
| lastIndex | 整数，代表下次匹配将从哪里字符位置开始 |
| multiline | Boolean 值，表示 m 是否已设置 |
| Source | 正则表达式的源字符串形式 |

/\*使用实例属性\*/

var pattern = /google/ig;

alert(pattern.global); //true，是否全局了

alert(pattern.ignoreCase); //true，是否忽略大小写

alert(pattern.multiline); //false，是否支持换行

alert(pattern.lastIndex); //0，下次的匹配位置

alert(pattern.source); //google，正则表达式的源字符串

var pattern = /google/g;

var str = 'google google google';

pattern.test(str); //google，匹配第一次

alert(pattern.lastIndex); //6，第二次匹配的位

PS：以上基本没什么用。并且 lastIndex 在获取下次匹配位置上 IE 和其他浏览器有偏差， 主要表现在非全局匹配上。lastIndex 还支持手动设置，直接赋值操作。

## 三．获取控制

正则表达式元字符是包含特殊含义的字符。它们有一些特殊功能，可以控制匹配模式的 方式。反斜杠后的元字符将失去其特殊含义。

字符类：单个字符和数字

|  |  |
| --- | --- |
| 元字符/元符号 | 匹配情况 |
| . | 匹配除换行符外的任意字符 |
| [a-z0-9] | 匹配括号中的字符集中的任意字符 |
| [^a-z0-9] | 匹配任意不在括号中的字符集中的字符 |
| \d | 匹配数字 |
| \D | 匹配非数字，同[^0-9]相同 |
| \w | 匹配字母和数字及\_ |
| \W | 匹配非字母和数字及\_ |

字符类：空白字符

|  |  |
| --- | --- |
| 元字符/元符号 | 匹配情况 |
| \0 | 匹配 null 字符 |
| \b | 匹配空格字符 |
| \f | 匹配进纸字符 |
| \n | 匹配换行符 |
| \r | 匹配回车字符 |
| \t | 匹配制表符 |
| \s | 匹配空白字符、空格、制表符和换行符 |
| \S | 匹配非空白字符 |

字符类：锚字符

|  |  |
| --- | --- |
| 元字符/元符号 | 匹配情况 |
| ^ | 行首匹配 |
| $ | 行尾匹配 |
| \A | 只有匹配字符串开始处 |
| \b | 匹配单词边界，词在[]内时无效 |
| \B | 匹配非单词边界 |
| \G | 匹配当前搜索的开始位置 |
| \Z | 匹配字符串结束处或行尾 |
| \z | 只匹配字符串结束处 |

字符类：重复字符

|  |  |
| --- | --- |
| 元字符/元符号 | 匹配情况 |
| x? | 匹配 0 个或 1 个 x |
| x\* | 匹配 0 个或任意多个 x |
| x+ | 匹配至少一个 x |
| (xyz)+ | 匹配至少一个(xyz) |
| x{m,n} | 匹配最少 m 个、最多 n 个 x |

字符类：替代字符

|  |  |
| --- | --- |
| 元字符/元符号 | 匹配情况 |
| this|where|logo | 匹配 this 或 where 或 logo 中任意一个 |

字符类：记录字符

|  |  |
| --- | --- |
| 元字符/元符号 | 匹配情况 |
| (string) | 用于反向引用的分组 |

|  |  |
| --- | --- |
| \1 或$1 | 匹配第一个分组中的内容 |
| \2 或$2 | 匹配第二个分组中的内容 |
| \3 或$3 | 匹配第三个分组中的内容 |

/\*使用点元字符\*/

var pattern = /g..gle/; //.匹配一个任意字符 var str = 'google';

alert(pattern.test(str));

/\*重复匹配\*/

var pattern = /g.\*gle/; //.匹配 0 个一个或多个

var str = 'google'; //\*,?,+,{n,m}

alert(pattern.test(str));

/\*使用字符类匹配\*/

var pattern = /g[a-zA-Z\_]\*gle/; //[a-z]\*表示任意个 a-z 中的字符

var str = 'google'; alert(pattern.test(str));

var pattern = /g[^0-9]\*gle/; //[^0-9]\*表示任意个非 0-9 的字符

var str = 'google'; alert(pattern.test(str));

var pattern = /[a-z][A-Z]+/; //[A-Z]+表示 A-Z 一次或多次

var str = 'gOOGLE'; alert(pattern.test(str));

/\*使用元符号匹配\*/

var pattern = /g\w\*gle/; //\w\*匹配任意多个所有字母数字\_ var str = 'google';

alert(pattern.test(str));

var pattern = /google\d\*/; //\d\*匹配任意多个数字 var str = 'google444';

alert(pattern.test(str));

var pattern = /\D{7,}/; //\D{7,}匹配至少 7 个非数字

var str = 'google8'; alert(pattern.test(str));

/\*使用锚元字符匹配\*/

var pattern = /^google$/; //^从开头匹配，$从结尾开始匹配 var str = 'google';

alert(pattern.test(str));

var pattern = /goo\sgle/; //\s 可以匹配到空格

var str = 'goo gle'; alert(pattern.test(str));

var pattern = /google\b/; //\b 可以匹配是否到了边界

var str = 'google'; alert(pattern.test(str));

/\*使用或模式匹配\*/

var pattern = /google|baidu|bing/; //匹配三种其中一种字符串 var str = 'google';

alert(pattern.test(str));

/\*使用分组模式匹配\*/

var pattern = /(google){4,8}/; //匹配分组里的字符串 4-8 次

var str = 'googlegoogle'; alert(pattern.test(str));

var pattern = /8(.\*)8/; //获取 8..8 之间的任意字符

var str = 'This is 8google8'; str.match(pattern);

alert(RegExp.$1); //得到第一个分组里的字符串内容

var pattern = /8(.\*)8/;

var str = 'This is 8google8';

var result = str.replace(pattern,'<strong>$1</strong>'); //得到替换的字符串输出 document.write(result);

var pattern = /(.\*)\s(.\*)/; var str = 'google baidu';

var result = str.replace(pattern, '$2 $1'); //将两个分组的值替换输出

document.write(result);

|  |  |
| --- | --- |
| 贪 婪 | 惰 性 |
| + | +? |
| ? | ?? |
| \* | \*? |
| {n} | {n}? |
| {n,} | {n,}? |
| {n,m} | {n,m}? |

/\*关于贪婪和惰性\*/

var pattern = /[a-z]+?/; //?号关闭了贪婪匹配，只替换了第一个 var str = 'abcdefjhijklmnopqrstuvwxyz';

var result = str.replace(pattern, 'xxx'); alert(result);

var pattern = /8(.+?)8/g; //禁止了贪婪，开启的全局 var str = 'This is 8google8, That is 8google8, There is 8google8';

var result = str.replace(pattern,'<strong>$1</strong>'); document.write(result);

var pattern = /8([^8]\*)8/g; //另一种禁止贪婪 var str = 'This is 8google8, That is 8google8, There is 8google8';

var result = str.replace(pattern,'<strong>$1</strong>'); document.write(result);

/\*使用 exec 返回数组\*/

var pattern = /^[a-z]+\s[0-9]{4}$/i; var str = 'google 2012';

alert(pattern.exec(str)); //返回整个字符串

var pattern = /^[a-z]+/i; //只匹配字母 var str = 'google 2012';

alert(pattern.exec(str)); //返回 google

var pattern = /^([a-z]+)\s([0-9]{4})$/i; //使用分组 var str = 'google 2012';

alert(pattern.exec(str)[0]); //google 2012

alert(pattern.exec(str)[1]); //google

alert(pattern.exec(str)[2]); //2012

/\*捕获性分组和非捕获性分组\*/

var pattern = /(\d+)([a-z])/; //捕获性分组 var str = '123abc';

alert(pattern.exec(str));

var pattern = /(\d+)(?:[a-z])/; //非捕获性分组 var str = '123abc';

alert(pattern.exec(str));

/\*使用分组嵌套\*/

var pattern = /(A?(B?(C?)))/; //从外往内获取 var str = 'ABC';

alert(pattern.exec(str));

## 四．常用的正则

|  |  |  |
| --- | --- | --- |
| /\*使用前瞻捕获\*/  var pattern = /(goo(?=gle))/; |  | //goo 后面必须跟着 gle 才能捕获 |
| var str = 'google'; |  |  |
| alert(pattern.exec(str)); |  |  |
| /\*使用特殊字符匹配\*/ var pattern = /\.\[\/b\]/; |  | //特殊字符，用\符号转义即可 |
| var str = '.[/b]'; |  |  |
| alert(pattern.test(str)); |  |  |
| /\*使用换行模式\*/  var pattern = /^\d+/mg; |  | //启用了换行模式 |
| var str = '1.baidu\n2.google\n3.bing'; |  |  |
| var result = str.replace(pattern, '#'); |  |  |
| alert(result); |  |  |
| 1.检查邮政编码  var pattern = /[1-9][0-9]{5}/; |  | //共 6 位数字，第一位不能为 0 |
| var str = '224000'; |  |  |
| alert(pattern.test(str)); |  |  |
| 2.检查文件压缩包  var pattern = /[\w]+\.zip|rar|gz/; |  | //\w 表示所有数字和字母加下划线 |
| var str = '123.zip'; |  | //\.表示匹配.，后面是一个选择 |
| alert(pattern.test(str)); |  |  |
| 3.删除多余空格 var pattern = /\s/g; |  | //g 必须全局，才能全部匹配 |
| var str = '111 222 333';  var result = str.replace(pattern,''); |  | //把空格匹配成无空格 |
| alert(result); |  |  |
| 4.删除首尾空格 var pattern = /^\s+/; |  | //强制首 |
| var str = ' goo gle | '; |  |
| var result = str.replace(pattern, ''); pattern = /\s+$/; |  | //强制尾 |
| result = result.replace(pattern, ''); |  |  |
| alert('|' + result + '|'); |  |  |

var pattern = /^\s\*(.+?)\s\*$/; //使用了非贪婪捕获

var str = ' google '; alert('|' + pattern.exec(str)[1] + '|');

var pattern = /^\s\*(.+?)\s\*$/;

var str = ' google ';

alert('|' + str.replace(pattern, '$1') + '|'); //使用了分组获取

5.简单的电子邮件验证

var pattern = /^([a-zA-Z0-9\_\.\-]+)@([a-zA-Z0-9\_\.\-]+)\.([a-zA-Z]{2,4})$/; var str = 'yc60.com@gmail.com';

alert(pattern.test(str));

var pattern = /^([\w\.\-]+)@([\w\.\-]+)\.([\w]{2,4})$/; var str = 'yc60.com@gmail.com'; alert(pattern.test(str));

PS：以上是简单电子邮件验证，复杂的要比这个复杂很多，大家可以搜一下。

**感谢收看本次教程！**

# 本课程是由北风网(ibeifeng.com) 瓢城 Web 俱乐部(yc60.com)联合提供：

**本次主讲老师：李炎恢 我的博客：hi.baidu.com/李炎恢/ 我的邮件：**[**yc60.com@gmail.com**](mailto:yc60.com@gmail.com)