

千锋HTML5学院

第二阶段javascript课程课件

1

正则表达式是什么？

2

正则表达式的创建

3

使用正则表达式匹配字符串

4

常见正则表达式

正则表达式是什么？

正则表达式(Regular Expression)是一个描述字符模式的对象, 用于进行字符串的匹配, 一般用在有规律的字符串匹配中, 如匹配用户名是否正确, 邮箱是否正确等;

正则表达式常用于表单验证, 如在 HTML 表单中填写的用户名、地址、出生日期, 邮箱等信息, 在表单提交到服务器做进一步处理之前, 我们需要先检查表单中的信息是否符合要求, 做表单验证, 以确认用户确实输入了信息并且这些信息是符合要求的。

正则表达式的创建

创建正则表达式有两种方式：

1, 采用 new 运算符

```
var box = new RegExp("box"); //传入非空字符串  
console.log(box); // /box/  
console.log(typeof box); //object
```

```
var box = new RegExp("box", "gi"); //第二个参数是模式修饰符  
console.log(box); // /box/gi
```

2, 采用字面量方式

```
var box = /box/;  
console.log(box); // /box/  
console.log(typeof box); //object
```

```
var box = /box/gi;  
console.log(box); // /box/gi
```

其中g表示全局匹配, i表示忽略大小写

使用正则表达式匹配字符串

使用正则表达式匹配字符串有两种方式:

1, `test()` : 返回`true`则符合, `false`则不符合

2, `exec()` : 返回数组则符合, `null`则不符合

这两种方法使用正则表达式对象去调用, 参数为要匹配的字符串

```
var box = /box/gi;  
var str = "This is a Box bOX box";  
console.log(box.test(str));  
console.log(/box/gi.test(str));
```

```
var box = /box/gi;  
var str = "This is a Box boX"  
console.log(box.exec(str));  
console.log(/box/gi.exec(str));
```

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。

```
var str = "This is a Box box BoX";  
var matchArr = str.match(/box/gi);  
console.log(matchArr); //返回数组或null
```

```
//查找并替换, 返回替换后的新字符串  
var replaceStr = str.replace(/box/gi, "xxx");  
console.log(replaceStr);
```

```
//查找并返回匹配的字符串的起始位置,找不到匹配的则返回-1  
var searchIndex = str.search(/box/i);  
console.log(searchIndex);
```

```
//根据指定字符串拆分, 返回拆分后的数组, 否则返回原字符串  
var splitArr = str.split(/b/i);  
console.log(splitArr.length);
```

正则表达式

//.号元字符, 代表除了换行之外的所有单个字符

```
var pattern = /g..gle/; //一个点.匹配一个任意的字符  
var str = "goagle";  
console.log(pattern.test(str));
```

/*号元字符, 配合其他字符使用, 允许其他字符出现任意多次

// 重复多次匹配, 可以出现任意次,
var pattern = /g.*gle/; //.* 匹配0到多个字符
var str = "google"
console.log(pattern.test(str));

// [] : 表示字符可以出现的范围

//[a-z]*表示任意0到多个a-z的字母

```
var pattern = /g[a-z]*gle/;  
var str = "google";  
console.log(pattern.test(str));
```

正则表达式

//非字符: ^

```
var pattern = /g[^0-9]*gle/; //可以有任意多个非0-9的字符  
var str = "google";  
console.log(pattern.test(str));
```

//+ 表示至少出现1次

//[A-Z]+: 至少出现一个A-Z的字符

```
var pattern = /[a-z][A-Z]+/;  
var str = "gooGle";  
console.log(pattern.test(str));
```

//使用元符号匹配

//\w*: 匹配任意多个数字字母下划线, \w: 等价于[a-zA-Z0-9_]

```
var pattern = /g\w*gle/;  
var str = "gooA3gle";  
console.log(pattern.test(str));
```


正则表达式

/\d 代表数字, 等价于 **[0-9]**

/\d* 表示任意多个数字

```
var pattern = /\d*gle/;  
var str = "g3243gle";  
console.log(pattern.test(str));
```

/\D: 匹配非数字, 相当于 **[^0-9]**

```
var pattern = /\Dgle/;  
var str = "ga3gle";  
console.log(pattern.test(str));
```

/\D{7,}: 匹配至少7个非数字, 相当于 **[^0-9]{7,}**

```
var pattern = /\D{7,}/;  
var str = "g3243gle";  
console.log(pattern.test(str));
```

正则表达式

//使用锚元字符

// **/^** 匹配开始,从头开始匹配

// **\$/** 匹配结尾,从结尾开始匹配

```
var pattern = /^google$/;  
var str = "google";  
console.log(pattern.test(str));
```

// **\s** 匹配空格

```
var pattern = /goo\sgle/;  
var str = "goo gle";  
console.log(pattern.test(str));
```

// **\b** 可以匹配是否到了边界,按单词搜索,把**google**当成一个单词来搜索

```
var pattern = /google\b/;  
var str = "googleaa go3oglexgoogle xxgooglde xxc";  
console.log(pattern.test(str));
```

正则表达式

//使用或模式匹配: |

// | 代表或者的意思, 匹配其中一种字符串

var pattern = /google|baidu|bing/; // |: 匹配三个中的其中一个字符串

var str = "googl2e bai3du;bingl"

console.log(pattern.test(str));

//分组模式匹配: ()

// ()加上小括号, 将内容进行分组, 可以作为一个整体进行多次匹配

var pattern = /(google){4,8}/; //匹配分组中的字符出现4-8次

var str = "googlegooglegooglegoogle"

console.log(pattern.test(str));

//获取8..8之间的任意字符

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

var str = "this is 8google8 baab 8ggg8";

console.log(str.match(pattern));

console.log(RegExp.\$1); //"google8 baab 8ggg"

正则表达式

使用exec返回数组

//exec 方法将匹配到的内容返回, 并将()分组的内容也放入数组返回

//以字母开头,至少一个, 忽略大小写

```
var pattern = /^[a-z]+/i;  
var str = "google 2016";  
console.log(pattern.exec(str)); // "google"
```

// /^[a-z]+\s[0-9]{4}\$/i : 表示以字母开头, 至少有一个字母, 有一个空格, 和4个数字, 并以数字结尾, 忽略大小写

```
var pattern = /^[a-z]+\s[0-9]{4}$/i;  
var str = "google 2016";  
console.log(pattern.exec(str)); // "google 2016"
```

正则表达式

//使用分组

```
var pattern = /^[a-z]+\s([0-9]{4})$/i;  
var str = "google 2016";  
console.log(pattern.exec(str)); //"google 2016,google,2016"  
console.log(pattern.exec(str)[0]); //google 2016  
console.log(pattern.exec(str)[1]); //google  RegExp.$1  
console.log(pattern.exec(str)[2]); //2016  RegExp.$2  
console.log(RegExp.$1); //google  RegExp.$1  
console.log(RegExp.$2); //2016  RegExp.$1
```

正则表达式

//捕获性分组 和非捕获性分组

//捕获性分组

```
var pattern = /(\d+)([a-z])/; //一个或多个数字,和一个a-z的字母  
var str = "123abc";  
console.log(pattern.exec(str)); //["123a,123,a"]
```

//非捕获性分组(添加?:后不会捕获第二个括号中的内容)

```
var pattern = /(\d+)(?:[a-z])/;  
var str = "123abc";  
console.log(pattern.exec(str)); //["123a,123"]
```

//使用分组嵌套

//从外往内捕获

```
var pattern = /(A?(B?(C?)))/;  
var str = "ABC";  
console.log(pattern.exec(str)); //["ABC,ABC,BC,C"]
```

正则表达式

//使用特殊字符匹配

//\:转义符,可以将本来有语义的字符没有语义,这里的.和[]都是字符,不代表任何正则匹配的语义

```
var pattern = /\.\[Vb\]/;  
var str = ".[/b]"  
console.log(pattern.test(str));
```

//使用换行模式

//m: 换行模式, 换行后又重新开始匹配

// ^表示以数字开头

```
var pattern = /^d+/mg;  
var str = "1,baidu\n2,google\n3,bing";  
console.log(str.replace(pattern, "#"));
```

常用正则表达式

检查邮政编码

//共6位数字, 第一位不能为0

```
var pattern = /^[1-9]\d{5}/;  
var str = "518000"  
console.log(pattern.test(str));
```

检查文件压缩包

//xxx.zip, xxx.gz, xxx.rar

```
var pattern = /^\\w+\\. (zip|gz|rar)$/;  
var str = "518000.zip"  
console.log(pattern.test(str));
```

删除多余空格

```
var str = " zhang san";  
var pattern = /\s+/g;  
str.replace(pattern, ""); //zhangsan
```


常用正则表达式

删除首尾空格

```
str.replace(/^s+/, "");  
str.replace(/\s+$/, "");  
var str = "  Li  ss  ";  
console.log(str.replace(/^s+/, ""));  
console.log(str.replace(/\s+$/, ""));
```

电子邮件

```
//xxxx@xxx(.xxx)+  
var str = "zhansan@1000phone.com"  
var pattern = /^[a-z0-9_\-\.]+\@([\da-z\-\\.]+\.)\.[a-z\]{2,6}$/;  
console.log(pattern.test(str));
```

手机号

```
var pattern = /^[1-3]\d{10}$/;  
var str = "18676273422";  
console.log(pattern.test(str));
```

常用正则表达式

身份证

```
var pattern = /^\\d{17}(\\d|X)$/;  
var str = "425173728482737711";  
console.log(pattern.test(str));
```

用户名只能使用数字字母下划线, 且数字不能开头, 长度在**6-15**位

```
var pattern = /^[a-zA-Z_][a-zA-Z0-9_]{5,14}/;
```

练习

1, 找出下面字符串中的**bag,beg,big,bog**, 忽略大小写, 并将其改为**bug**:

**I am a Big man, I have so mach bag, so veryone call me
beg man, bog bog bog, I hate you!**

练习

2, 假设有一个多字符的片断重复出现,
把**"really"**、**"really really"**, 以及任意数量连续出现的
"really"字符串换成一个简单的**"very "**

Billy tried really hard Sally tried really really hard Timmy tried
really really really hard Johnny tried really really really really
hard

练习

3, 在第二个输入框中随机输入一个字符串, 然后在第一个输入框中输入特定字符串, 点击替换, 将第二个输入框中的匹配的字符串替换成**, 如图

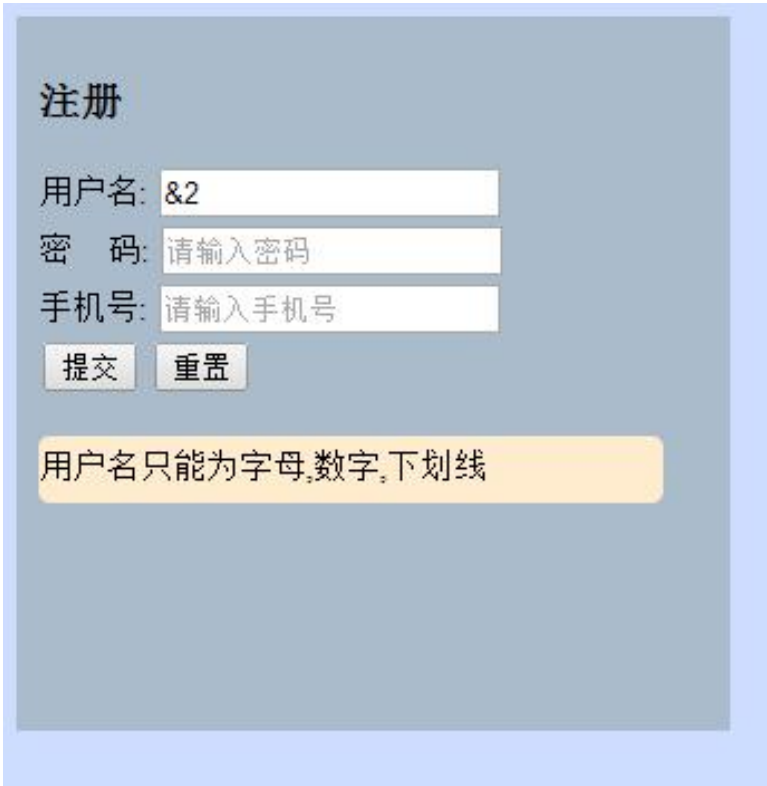
练习

4, 有以下表单, 验证用户名, 密码, 手机号

1, 用户名只包含数字, 字母, 下划线, 且长度不小于6位

2, 密码长度在8到16位

3, 手机号只能输入数字, 且长度为11



注册

用户名:

密 码:

手机号:

用户名只能为字母, 数字, 下划线

