# JavaScript基础入门

第6天课堂笔记（本课程共10天）

目录

[JavaScript基础入门](#_Toc23207)

[一、字符串方法复习](#_Toc12099)

[二，正则表达式](#_Toc19786)

[2.1创建正则表达式](#_Toc25366)

[2.2 split](#_Toc18365)

[2.4 match](#_Toc1685)

[2.5 replace](#_Toc9938)

[2.6 replace](#_Toc1602)

[2.7 exec](#_Toc25644)

[2.8 test](#_Toc24497)

[三、 正则表达式语法](#_Toc20362)

[3.1 特殊字符](#_Toc12856)

[3.2 预定义特殊字符](#_Toc10034)

[3.3字符集](#_Toc5670)

[3.4 分界](#_Toc12498)

[3.5预定义类](#_Toc30373)

[3.6 量词](#_Toc15958)

[3.7 修饰符](#_Toc19378)

[3.8其他](#_Toc9860)

[3.9 分组](#_Toc15568)

[四，实战](#_Toc14169)

[4.1验证手机号](#_Toc14947)

[作业](#_Toc6276)

# 一、字符串方法复习

concat字符串拼接

charAt 获取某位置上的字符

charCodeAt获取某位置上的字符编码

slice 截取字符串

subString截取字符串

subStr截取字符串

indexOf 获取字符在字符串中的位置

lastIndexOf 获取字符在字符串中的位置 逆向

search 获取字符在字符串中的位置

split 字符串转化成数组

replace 字符串替换

# 二，正则表达式

常用的使用场景，[字符串替换，表单验证](https://www.ibm.com/developerworks/cn/web/0909_formvalid_hulx/)

正则表达式是用来匹配字符串的一种模式

数据类型是对象

今天我们要学习6个方法

正则相关方法： [exec](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/RegExp/exec)和[test](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/RegExp/test)

字符串相关方法： [search](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/String/search)， [match](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/String/match)， [split](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/String/split)，[replace](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/String/replace)

## 2.1创建正则表达式

你可以通过下面两种方法创建一个正则表达式：

使用一个正则表达式字面量，如下所示：  
var re = /abc/;  
正则表达式字面量在脚本加载后编译。若你的正则表达式是常量，使用这种方式可以获得更好的性能。

调用RegExp对象的构造函数，如下所示：  
var re = new RegExp("abc");  
（new操作符我们《JavaScript高级》课程详细介绍，这里记住即可）

## 2.2 split

切割字符串的方法，我们通常传递字符或者字符串来切割字符串，此时我们通常传递的参数是能看到的摸得着，是比较具体的，

当我们不能用一个完整的字符串表达这个切个符号的时候，但是我们能提炼换一个规则来切割，此时我们就可以用正则表达式来表示这个规则，切割字符串。

|  |
| --- |
| 1. var str = 'a b c'; 2. console.log(str.split('')) 3. console.log(str.split(/\s+/)) |



2.3 search

用来寻找字符或者字符串在父字符串中的位置，我们可以传递字符，也可以传递字符串。

|  |
| --- |
| 1. var str = 'abcde'; 2. console.log(str.indexOf('cd')) 3. console.log(str.search('cd')) 4. console.log(str.search(/cd/)) |



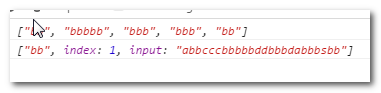
类数组：它是融合了数组和对象的一个对象，他可以使用数组的方法，有数组的属性，他也可以使用对象的方法，有对象的属性。

正则表达式中，是一个严格模式，一定注意，不要乱添加空格。

## 2.4 match

在父字符串中寻找匹配字符串，它是一个字符串方法，参数是配置字符串或者匹配的规则。

|  |
| --- |
| 1. var str = 'abbcccbbbbbddbbbdabbbsbb'; 2. console.log(str.match(/b+/g)) 3. console.log(str.match('bb')) |



## 2.5 replace

替换父字符串中匹配的字符串。

接收两个参数，

第一个是：匹配的字符串，可以是字符串也可以是正则表达式

第二个是：替换的字符串，可以是简单类型如字符串，也可以是函数，注意函数要有返回值，并且将这个返回值作为字符串

|  |
| --- |
| 1. var str = 'www.ucai.cn'; 2. // var str = 'www.ucai.com'; 3. console.log(str.replace(/ucai\.cn/, 'ucai.com')); 4. console.log(str.replace(/ucai\.cn/, function() { 5. return 'ucai.com'; 6. })) |

字符串方法返回值有三类：

返回数字类型：indexOf，lastIndexOf，search , charCodeAt

返回数组的方法：split，match

返回字符串的方法：charAt，substring，substr, slice, replace

## 2.6 replace

第一个参数是字符串或者是正则表达式，要是的是一个结果，当然也可以传递函数，注意，当传递的是函数时候一定要执行，并且有返回值，并返回值是字符串或者正则表达式。

第二个参数是字符串或者函数，字符串表示结果，当匹配过程中不会执行，如果是函数的话，分成两种情况，如果执行了跟字符串效果一样，如果没执行，会在匹配过程中执行。

很多情况下，一次性匹配需要写的正则表达式比较复杂，此时我们可以将其拆分开来，一步一步替换处理。

\s+ 它是贪婪模式，她会尽可能多的去匹配，这样匹配的次数比较少，性能更高

\s 它只表示一个空格，所以匹配的次数多，性能相对低些。

## 2.7 exec

它是正则对象的方法

正则表达式我们有两种创建方式。

字面量形式： var str = /abc/g; 表示全局匹配abc字符串

实例化形式：var reg = new RegExp('abc', 'g') 表示创建一个匹配abc字符串的正则对象，注意，转义字符要用双反斜杠’\\’,原因是第一个参数是字符串不是正则表达式。

所以使用exec方法是，只需要对正则对象直接调用即可，他表示在字符串中匹配符合正则规则的字符串

|  |
| --- |
| 1. // 匹配abc字符串 2. var str = /abc/g; 3. var reg = new RegExp('abc', 'g') 4. var demo = 'aaaabccccbacabc'; 5. // str.exec('abcd') => [abc] 6. console.log(str.exec(demo)); 7. // str.exec('abcd') => [abc] 8. console.log(reg.exec('aaaabccccbacabc')); |



## 2.8 test

正则对象的一个方法，它表示该字符串（参数）是否匹配我们的正则规则。

|  |
| --- |
| 1. // var reg = /hello/; 2. var reg = /iqianduan/; 3. var str = 'hello icketang'; 4. // 判断str字符串是否匹配我们reg规则 5. console.log(reg.test(str)); 6. var demo = prompt('不准输入空格'); 7. alert(/\s/.test(demo)); |



# 正则表达式语法

## 3.1 特殊字符

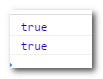
[正则表达式](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Guide/Regular_Expressions)由普通字符和特殊字符（元字符）组成。

普通字符包括：字母（abcABC），数字（123），\_等等

特殊字符：() [] {} ^ $ \* ? \ | + .

普通字符我们可以直接拿来用，但是，特殊字符是一定要转义。

|  |
| --- |
| 1. // 匹配 ^a 2. var reg = /\^a/ 3. console.log(reg.test('bb^acc')) 4. // 匹配\m 5. var reg = /\\m/; // 特殊字符要转义 6. console.log(reg.test('asdf\\msadfasd')); // 字符串中反斜杠要转义 |



## 3.2 预定义特殊字符

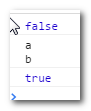
\t /\t/ 制表符

\n /\n/ 回车符

\f /\f/ 换页符

\b /\b/ 与回退字符

|  |
| --- |
| 1. // 匹配制表符 2. var str = 'a b'; 3. var reg = /\t/; 4. console.log(reg.test(str)); 5. // 匹配回车符 6. var str2 = `a\nb`; 7. var reg2 = /\n/; 8. console.log(str2) 9. console.log(reg2.test(str2)); |



## 3.3字符集

Javascript的正则表达式中有四类字符集

简单类

它是有一一对应的字符组成的集合，通过[]包裹住，来表示这几个字母组成的一个集合

如： [abDF45]表示由abDF45六个字符组成的一个集合

范围类

通过首位字母以及-组成的一个范围集合

如：[a-z] 表示小写字母集合。[A-Z] 表示大写字母集合。[0-9] 表示数字集合。[i-y] 表示小写字母i到y的集合

负向类

通过在[]内部最前面添加^来表示不包含该集合的字符集

如：[^abc] 表示不包含abc的任意字符集合，[^ab89CD] 表示不包含ab89CD这六个字符的集合

组合类

通过[]将几个集合拼接在一起表示一个组合的集合。

如：[a-zA-Z0-9] 表示大小写字母以及数字的结合

|  |
| --- |
| 1. // 简单类 nba 三个字母组成的简单集合 2. // var reg = /[nba]/; 3. // console.log(reg.test('xyzn')) 4. // 范围类 A-Z 大写字母集合 5. // var reg = /[A-Z]/; 6. // console.log(reg.test('a')) 7. // 负向类 不包含yellow 8. // var reg = /[^9]/; 9. // console.log(reg.test('9')) 10. // 组合类 11. var reg = /[0-9a-z]/; 12. console.log(reg.test('9a')) |

## 3.4 分界

字符串是有边界，这里的边界指的就是字符串的首尾

字符串的起始位置我们用^，

如： /^abc/ 判断字符串是否是以abc开始的

字符串的结束位置我们用$表示

如：/xyz$/ 判断字符串是否是一xyz结尾的

对于单词的分界，正则表达式给我定义了一些分隔符

单词分隔符我们用\b表示 [a-zA-Z\_0-9]之外的字符

非单词分隔符我们用\B表示

|  |
| --- |
| 1. // 判断字符串是否是以abc开头 2. var reg = /^abc/; 3. console.log(reg.test('abcdef')) 4. console.log(reg.test('hello')) 5. // 判断字符串是否是以xyz结尾的 6. var reg = /xyz$/; 7. console.log(reg.test('uvwxyz')); 8. console.log(reg.test('helloyz')); 9. // 'how old are you'切割成单词 10. var str = 'how old are you|abc'; 11. // 大小写字母表示：[a-zA-Z]+ 12. console.log(str.match(/\b[a-zA-Z]+\b/g)); |

## 3.5预定义类

. [^\n\r] 除了换行和回车之外的任意字符

\d [0-9] 数字字符

\D [^0-9] 非数字字符

\s [ \t\n\x0B\f\r] 空白字符

\S [^ \t\n\x0B\f\r] 非空白字符

\w [a-zA-Z\_0-9] 单词字符(所有的字母)

\W [^a-zA-Z\_0-9] 非单词字符

## 3.6 量词

? 软性量词 出现零次或一次

\* 软性量词 出现零次或多次(任意次)

+ 软性量词 出现一次或多次（至道一次）

{n} 硬性量词 对应零次或者n次

{n,m} 软性量词 至少出现n次但不超过m次(中间不能有空格)

{n,} 软性量词 至少出现n次(+的升级版)

## 3.7 修饰符

g 表示全局匹配

i 忽略大小写

m 多行匹配

## 3.8其他

[\u4e00-\u9fa5]表示中文集

| 表示或者的意思

alert(/a|b/.test('b')) =》 true

alert(/a|b/.test('c')) =》 false

## 3.9 分组

分组

1 解决多个字符串重复

2 可以在外面获取

格式 (任意个字符)

获取

在正则表达式内部获取：\数字

在正则表达式外部获取：$数字

注意数字是从1开始的。

# 四，实战

## 4.1验证手机号

|  |
| --- |
| 1. // 验证手机号 2. /\*\* 3. \* 11位 {11} 4. \* 第一位是1 ^1 5. \* 都是数字 [0-9] \d 6. \*/ 7. var reg = /^1\d{10}$/; 8. var str = prompt('请输入手机号'); 9. if (reg.test(str)) { 10. alert('成功') 11. } else { 12. alert('失败'); 13. } |