



# .NET架构

主讲教师 张智  
计算机学院

## 第6章 JavaScript编程



JavaScript是一种广泛用于 Web 编程的轻量级脚本语言，被设计为向 HTML 页面增加交互性。

JavaScript标准：由ECMAScript制定，简称ES。

ES6：2015.6月发布，目前最新 ES9。

European Computer Manufacturers Association

## JavaScript基本特点

- 是一种**解释性脚本语言**（代码不进行预编译）
- 主要用来**向HTML页面添加交互行为**
- 可**直接嵌入**HTML页面，但写成**单独js文件**有利于结构和行为的分离
- 基于对象的**事件驱动**的简单**弱类型**脚本语言
- **跨平台特性**：在绝大多数浏览器的支持下，可以在多种平台下运行（如Windows、Linux、Mac、Android、iOS等）

## 主要内容

6.1 JS编程基础

6.2 DOM操作

6.3 浏览器对象

6.4 JSON对象

【附录】RegExp正则式对象

## 6.1 JS编程基础

- [JavaScript声明](#)
- [JavaScript变量](#)
- [JavaScript函数](#)

**【[返回](#)】**

# 1. JavaScript声明

## ■ 基本格式

```
<script>
```

```
...
```

```
JavaScript代码;
```

```
...
```

```
</script>
```

JavaScript可出现在HTML的任意地方  
建议将JS代码放在body元素的底部

## 示例1：JS位于body

当页面载入时，会直接执行位于body内的脚本

```
<html>
```

```
<body>
```

```
<script>
```

```
document.write("<h1>Hello World!</h1>");
```

→ 在web页面上直接输出

```
console.log("Hello World!");
```

→ 在控制台上输出

```
window.alert("hello world!");
```

→ 在web页面上弹出警告框

```
</script>
```

```
</body>
```

window可省略

```
</html>
```

备注：如果出现中文乱码，head中添加字符集：

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

## 示例2: JS位于head

将脚本写为函数形式：避免页面载入时被直接执行

```
<html>
<head>
  <script>
    function disp() {
      alert("Hello World!");
    }
  </script>
</head>
<body>
  <input type="button" onclick="disp()" value="OK" />
</body>
</html>
```

当函数被调用时  
JS代码才会被执行

简单语句可直接嵌入：onclick="alert('hello world')"

说明js可位于元素内部



### 示例3：JS位于外部文件

```
<html>
<head>
  <script src="scripts/my.js"> </script>
</head>
<body>
  <input type="button" onclick="disp()" value="OK" />
</body>
</html>
```

引用外部 js 文件

外部 js 文件不用  
包含 <script> 标签

my.js文件

```
function disp()
{
    alert("Hello World!");
}
```

## JavaScript代码调试

控制台窗口

源码窗口

console.log 输出

输出某个变量名查看其值

点击行号添加或去除断点

调试工具栏

```
<script>
var a = 1;
var b = a + 2 * 3;
console.log("a=" + a);
console.log("b=" + b);
</script>
```

Paused on breakpoint

Call Stack

(anonymous)

Breakpoints

testJ.html:6  
var b = a + 2 \* 3;

XHR/fetch Breakpoints

DOM Breakpoints

Global Listeners

Event Listener Breakpoints

Scope

Watch

Global

Window

[【返回】](#)

## 2. JavaScript变量

常用 **var** 来声明JavaScript变量：

```
var x=5;    //Number类型  
var flag=true; //Boolean类型  
var car="Benz"; //String类型，注：单引号也行
```

思考一下：可以吗？  
var a = 123;  
a = 'ABC';

ES6 引入新关键字 **let**，用 **let** 替代 **var**

```
let x=5;  
let flag=true;  
let car="Benz";
```

## 变量命名规则

- 规则：由字母或下划线、\$开头，后接任意个由一个数字、字母或者下划线、\$组成的字符串
- 变量名不能是JavaScript的关键字（比较多）
- JavaScript变量是弱类型
- JavaScript变量对大小写敏感

例如：

正确的变量命名：\$a 、 \_a 、 \$\_a、 s\_007 、 \$\_\$、 Answer

错误的变量命名：this、 1a、 b%

## JS示例：字符串用法

```
var s="How are you?";  
var n = s.indexOf('a'); //n=4  
var ss = s.substring(1,3); //取子串，索引1至3止(不含) ss="ow"  
var arr = s.split(" "); //用空格分割字符串 arr=["How", "are", "you?"]  
var b = 'a1b22c333d'.split(/\d+/) //使用正则式分割，返b=["a", "b", "c", "d"]  
var c = "This is a map".match("is"); //字符串匹配，只匹配一次 c=["is"], 匹配不成功null  
var d = "1 plus 2 equal 3".match(/\d+/g); //使用正则式全局匹配， d=['1','2','3']  
var e = 'This is a map'.replace('is','ere') //只替换一次 e='There is a map'  
var f = 'a1b22c3d'.replace(/\d+/g,':') //全局替换 f='a:b:c:d'
```

## JS示例：数组用法

```
var a1 = [1, 3.14, 'Hello', null, true, new Date()]; //数组可以有包含任意数据类型
a1[2] = false; //可通过索引读取/修改对应元素, 'hello'修改为false值
var n = a1.length // n=6
for (var i = 0; i < a1.length; i++) { console.log(a1[i]); } //数组遍历
a1.forEach( function (x) { console.log(x); } ); //数组遍历
var a3 = [1, 2];
a3.push('A', 'B'); //进堆 a3 = [1, 2, 'A', 'B']
var p = a3.pop(); //出栈 a3= [1, 2, 'A'] p='B'
var a4 = [1, 2];
a4.unshift('A', 'B'); //在数组头部添加若干元素 arr = ['A', 'B', 1, 2]
var q = a4.shift(); //把数组第一个元素删除掉 arr = ['B', 1, 2] q = 'A'
var a5 = ['Big', 'Cat', 'Apple', 'big'];
a5.sort(); //排序 a5=["Apple", "Big", "Cat", "big"]
a5.reverse(); //反转 a5=["big", "Cat", "Big", "Apple"]
```

## JS示例：对象用法

```
var p1 = {  
    name: "小明",  
    age: 20  
}  
console.log( p1.name ); // '小明'  
var p2 = {  
    name: '小明',  
    birth: 1998,  
    age: function () { //定义对象方法  
        var y = new Date().getFullYear();  
        return y - this.birth;  
    }  
};  
console.log( p2.age() ); //调用对象方法
```

```
var p3 = {  
    name: '小明',  
    age: 20,  
    school: 'wust',  
    address: { // 对象可以嵌套  
        city: '武汉',  
        street: '白沙洲达到黄家湖西路',  
        zipcode: '430068'  
    }  
};  
console.log( p3.address.city ); // '武汉'
```

[【返回】](#)

### 3. JavaScript函数

- 函数一般先定义后使用 → 函数后定义也是可以的，匿名函数除外
- 函数一般格式：

函数名前没有类型

```
function 函数名称([参数列表]) {  
    //执行的语句  
    [ return [值]; ]  
}
```

只写参数名，不指定类型

函数名大小写敏感

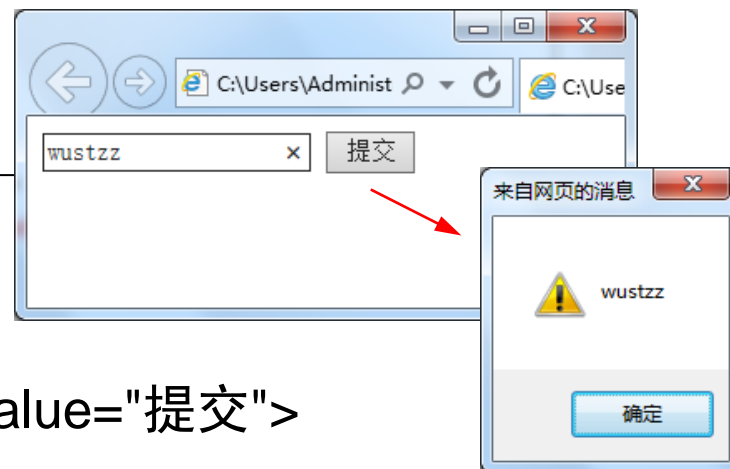
如果没有return，默认return **undefined**;



## 示例：不带参数函数

```
<form>
  <input type="text" id="username"/>
  <input type="button" onclick="show()" value="提交">
</form>

<script>
  function show() {
    alert(document.getElementById("username").value);
  }
</script>
```



返回属性id值对应的元素

## 示例：带参数函数

```
<script>
```

```
function add(a,b)  {
```

```
    return a+b;
```

```
}
```

```
</script>
```

```
6+5=<script>document.write( add(6,5) )</script>
```

参数只写参数名，不指定类型

即使有返回值也不能加函数类型

思考一下：本例js代码可以放到后面吗？

## 匿名函数

匿名函数没有函数名，必须先定义后使用

### ■ 用法1：匿名函数赋值给一般变量（函数变量）

```
var foo = function(name) {  
    alert("hello "+name);  
};  
foo("小明"); //调用
```

→ 匿名函数

### ■ 用法2：匿名函数赋值给特殊变量

```
window.onload = function() {  
    alert("hello");  
};
```

onload是window对象的一个事件属性  
onload 当在页面中所有元素以及内容全部加载完成以后触发

## 默认参数：

- 函数默认参数允许在没有值或undefined被传入时使用默认形参
- 示例：

默认参数

```
function multiply( a, b = 1 ) {  
    return a * b;  
}  
  
console.log(multiply(5, 2)); // 10  
console.log(multiply(5));    // 5
```

## 不定参数：

■ 不定参数：形参前面加三个点，表示为一个数组

■ 示例：

不定参数

```
function foo( ...args ) {  
    console.log( args.length );  
}  
foo();    // 0  
foo(5);   // 1  
foo(5, 6, 7); // 3
```

```
function foo( x, ...args ) {  
    console.log(x); // 5  
    console.log(args); // [6,7]  
}  
foo(5, 6, 7);
```

# javascript:void(...)

void(exp)表示计算一个表达式但是不返回值

示例：

■ `<a href="javascript:void(0)">点我</a>`

当用户点击链接时，void(0) 计算为 0，超链接无跳转。

■ `<a href="javascript:void(alert('hehe'))">点我</a>`

当用户点击链接时，弹出alert窗口，超链接无跳转。

## Methods

## Object

toString  
toLocaleString  
valueOf  
hasOwnProperty  
isPrototypeOf  
propertyIsEnumerable

## String

charAt  
charCodeAt  
fromCharCode  
concat  
indexOf  
lastIndexOf  
localeCompare  
match  
replace  
search  
slice  
split  
substring  
substr  
toLowerCase  
toUpperCase  
toLocaleLowerCase  
toLocaleUpperCase

## RegExp

test  
match  
exec

## Array

concat  
join

## JavaScript 函数(部分)

## XMLHttpRequest

## Safari, Mozilla, Opera:

```
var req = new XMLHttpRequest();
```

## Internet Explorer:

```
var req = new  
ActiveXObject("Microsoft.XMLHTTP");
```

## XMLHttpRequest Object Methods

```
abort()  
getAllResponseHeaders()  
getResponseHeader(header)  
open(method, URL)  
send(body)  
setRequestHeader(header, value)
```

## XMLHttpRequest Object Properties

```
onreadystatechange  
readyState  
responseText  
responseXML  
status  
statusText
```

## REGULAR EXPRESSIONS - FORMAT

Regular expressions in JavaScript take the form:

```
var RegEx = /pattern/modifiers;
```

## REGULAR EXPRESSIONS - MODIFIERS

```
/g    Global matching  
/i    Case insensitive  
/s    Single line mode  
/m    Multi line mode
```

## REGULAR EXPRESSIONS - PATTERNS

```
^      Start of string  
$      End of string  
.      Any single character  
(a|b)  a or b  
(...)  Group section  
[abc]  Item in range (a or b or c)  
[^abc] Not in range (not a or b or c)  
a?     Zero or one of a  
a*     Zero or more of a  
a+     One or more of a
```

## DOM Methods

## Document

```
clear  
createDocument  
createDocumentFragment  
createElement  
createEvent  
createEventObject  
createRange  
createTextNode  
getElementsByName  
getElementById  
write
```

## Node

```
addEventListener  
appendChild  
attachEvent  
cloneNode  
createTextRange  
detachEvent  
dispatchEvent  
fireEvent  
getAttributeNS  
getAttributeNode  
hasChildNodes  
hasAttribute  
hasAttributes  
insertBefore  
removeChild  
removeEventListener  
replaceChild  
scrollIntoView
```

## Form

```
submit
```

push  
pop  
reverse  
shift  
slice  
sort  
splice  
unshift

**Number**  
toFixed  
toExponential  
toPrecision

**Date**  
parse  
toDate  
toString  
getDate  
getDay  
getFullYear  
getHours  
getMilliseconds  
getMinutes  
getMonth  
getSeconds  
getTime  
getTimezoneOffset  
getYear  
setDate  
setHours  
setMilliseconds  
setMinutes  
setMonth  
setSeconds  
setYear  
toLocaleTimeString

### XMLHttpRequest readyState Values

0	Uninitiated
1	Loading
2	Loaded
3	Interactive
4	Complete

### JAVASCRIPT IN HTML

#### External JavaScript File

```
<script type="text/javascript"
src="javascript.js"></script>
```

#### Inline JavaScript

```
<script type="text/javascript">
<!--
    // JavaScript Here
//-->
</script>
```

### Functions

#### Window

alert  
blur  
clearTimeout  
close  
focus  
open  
print  
setTimeout

#### Built In

eval  
parseInt  
parseFloat  
isNaN  
isFinite  
decodeURI  
decodeURIComponent  
encodeURIComponent  
encodeURIComponent  
escape  
unescape

a{3}	Exactly 3 of a
a{3,}	3 or more of a
a{3,6}	Between 3 and 6 of a
!(pattern)	"Not" prefix. Apply rule when URL does not match pattern.

### EVENT HANDLERS

onAbort	onMouseDown
onBlur	onMouseMove
onChange	onMouseOut
onClick	onMouseOver
onDblClick	onMouseUp
onDragDrop	onMove
onError	onReset
onFocus	onResize
onKeyDown	onSelect
onKeyPress	onSubmit
onKeyUp	onUnload
onLoad	

### FUNCTIONS AND METHODS

A method is a type of function, associated with an object. A normal function is not associated with an object.

Available free from  
AddedBytes.com

**DOM Collections**  
item

#### Range

collapse  
createContextualFragment  
moveEnd  
moveStart  
parentElement  
select  
setStartBefore

#### Style

getPropertyValue  
setProperty

#### Event

initEvent  
preventDefault  
stopPropagation

#### XMLSerializer

serializeToString

#### XMLHTTP

open  
send

#### XMLDOM

loadXML

#### DOMParser

parseFromString

[【返回】](#)



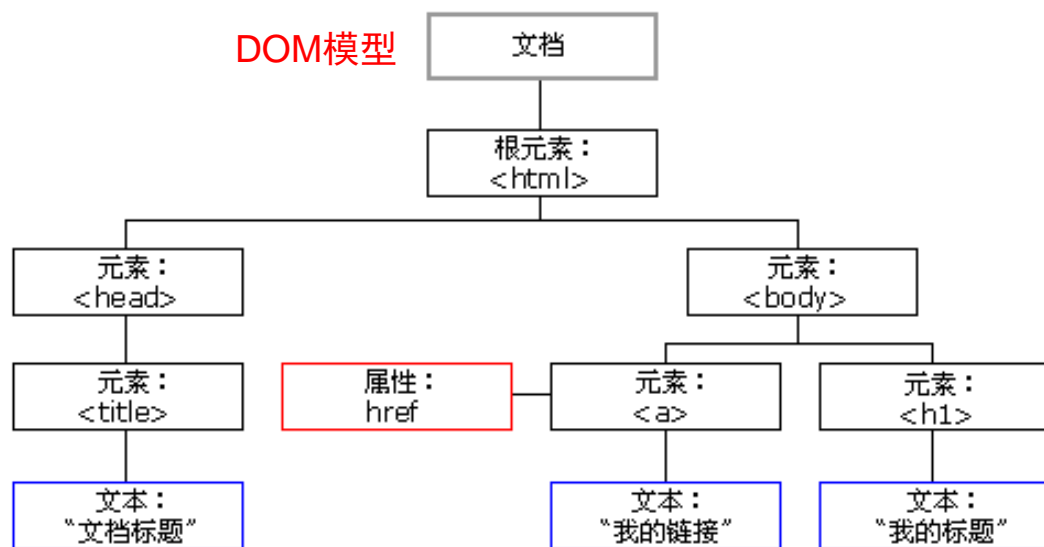
## 6.2 DOM操作

- DOM概念
- DOM查找
- DOM更新
- DOM添加
- DOM删除
- DOM事件

**[【返回】](#)**

# 1. DOM概念

- 当网页被加载时，浏览器会创建页面的文档对象模型DOM
- DOM: Document Object Model



## DOM说明

在 DOM 中, 每个东西都是节点:

- 文档本身就是一个文档对象
- 所有 HTML 元素都是元素节点
- 所有 HTML 属性都是属性节点
- 插入到 HTML 元素文本是文本节点 (包括空格、回车、换行等)
- 注释是注释节点

## DOM操作

注意：要在DOM加载完之后再执行相关js操作

■ DOM四大操作：查找、更新、添加、删除

■ 通过DOM，JavaScript 可创建动态的 HTML：

- 能够改变页面中的所有 HTML 元素
- 能够改变页面中的所有 HTML 属性
- 能够改变页面中的所有 CSS 样式
- 能够对页面中的所有事件做出反应

前提：首先找到相关元素

【[返回](#)】

## 2. DOM查找

### ■ 查找 DOM 节点方法:

#### ■ 返回指定 id 的第一个节点元素

```
document.getElementById("id值");
```



#### ■ 返回指定 标签名 的节点集合 (HTMLCollection集合)

```
document.getElementsByTagName("标签名")
```

也有length属性和下标访问

#### ■ 返回指定 类名 的节点集合 (HTMLCollection集合)

```
document.getElementsByClassName("样式类名")
```

类名不需要加'.'

示例：

```
<p>hello1</p>
<div id="main">
  <p>hello2</p>
  <p>hello3</p>
</div>
<script>
  var x=document.getElementsByTagName("p");
  document.write("长度: " + x.length+"<br>"); // 长度: 3
  document.write("p为: " + x[0].innerHTML+"<br>"); //hello1
</script>
```

修改一下结果如何： document.getElementById("main").getElementsByTagName("p");

## ■ 获取某节点下的其他相关节点:

```
var test = document.getElementById('demo');
```

```
// 获取test节点下所有直属儿子节点:
```

```
var children = test.children;
```

```
// 获取test节点下第一个、最后一个子节点:
```

```
var first = test.firstElementChild;
```

```
var last = test.lastElementChild;
```

```
// 获取test节点的父节点
```

```
var parent=test.parentElement;
```

```
// 获取test节点的上一个兄弟、下一个兄弟元素节点
```

```
var pre=test.previousElementSibling;
```

```
var next=test.nextElementSibling;
```

IE9+以上支持

这些新方法获得全部是元素节点 (OK)  
不包括文本节点(如空格、回车、换行等)、注释节点

## 示例：

```
<div id="test-div">
  <div class="c-red">
    <p id="test-p">JavaScript</p>
    <p>Java</p>
  </div>
  <div class="c-red c-green">
    <p>Python</p>
    <p>Ruby</p>
    <p>Swift</p>
  </div>
  <div class="c-green">
    <p>Scheme</p>
    <p>Haskell</p>
  </div>
</div>
```

问题1：选择<p>JavaScript</p>

答案：

```
var js = document.getElementById('test-p');
```

问题2：

选择<p>Python</p>,<p>Ruby</p>,<p>Swift</p>

答案：

```
var div1 =
  document.getElementsByClassName('c-red c-green');
var arr= div1[0].children; //获得第一个div的直属儿子
for (let i = 0; i < arr.length; i++) { //背景设置为黄色
  arr[i].style.backgroundColor = "yellow";
}
```

问题3：选择<p>Haskell</p>

答案：

```
var div2 = document.getElementsByClassName('c-
green');
var haskell=div2[1].lastElementChild //获取第二个div
haskell.style.backgroundColor = '#ff0000';
```

### 3. DOM 更新

- 修改 DOM 节点内容
- 修改 DOM 节点属性
- 修改 DOM 节点样式



## (1) 修改 DOM 节点内容

主要方法：修改 **innerHTML** 属性(注意大小写)，不但可以修改DOM节点文本内容，还可以直接修改DOM节点内部子树。

```
<p id="demo">Hello World!</p>
```

```
document.getElementById("demo").innerHTML = 'hello';
```

```
document.getElementById("demo").innerHTML = "<h3>JavaScript</h3>";
```

提示：放在等号左边是赋值，放在右边则是取值

或者：使用 **innerText** 属性（注意区别）

## (2) 修改 DOM 节点属性

修改 DOM 节点属性值：

`document.getElementById(id).属性名 = 属性值`

或者：`document.getElementById(id).setAttribute(属性名, 属性值)`

提示：放在等号左边是赋值，放在右边则是取值

补充：`getAttribute(属性名)`：获取属性值

```

```

```
<a id="link" href="#">网易</a>
```

```
<script>
```

```
    document.getElementById("img1").src="images/img2.jpg";
```

```
    document.getElementById("link").setAttribute("href","http://www.163.com");
```

```
</script>
```

### (3) 修改 DOM 节点样式

修改 DOM 节点的样式：使用style对象完成

`document.getElementById(id).style.样式属性 = 样式值(字符串)`

说明：**驼峰式命名** 例如：样式名为: font-size，对应的style属性名用: fontSize（去掉连接线,首字母变大写）

```
<p id="p1">Hello World!</p>
```

```
<p id="p2">Hello World!</p>
```

```
<script>
```

```
    document.getElementById("p1").style.color="blue";
```

```
    document.getElementById("p2").style.fontSize="20px";
```

```
</script>
```

引号不能掉，因为样式都是字符串

## 修改 DOM 节点样式类

修改 DOM 节点的样式类：使用class属性完成

```
document.getElementById(id).setAttribute("class","class名");
```

```
document.getElementById(id).className = "class名"
```

此处用'class'无效

```
<p id="p1">Hello World!</p>
```

```
<p id="p2">Hello World!</p>
```

```
<script>
```

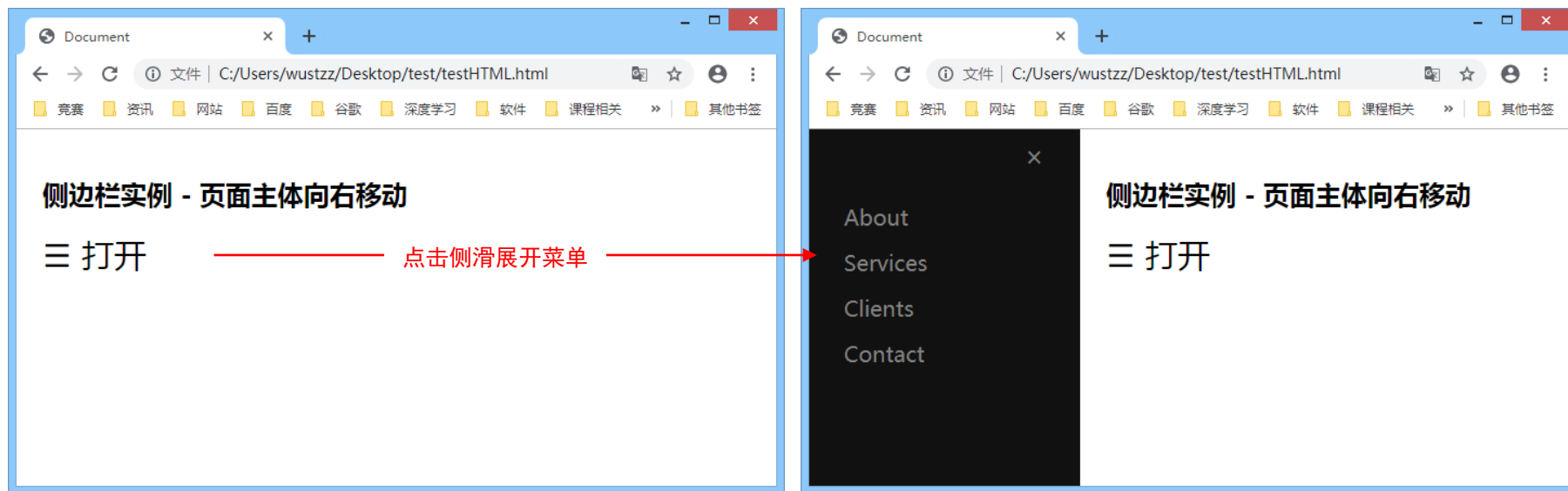
```
    document.getElementById("p1").className = "strong";
```

```
    document.getElementById("p2").setAttribute("class","normal");
```

```
</script>
```

```
.strong {  
    color: red;  
    font-size: 20px;  
}  
.normal {  
    color: black;  
    font-size: 16px;  
}
```

## 示例：侧边栏菜单



```
<div id="mySidenav" class="sidenav">
  <a href="javascript:void(0)" class="closebtn" onclick="closeNav()">&times;</a>
  <a href="#">About</a>
  <a href="#">Services</a>
  <a href="#">Clients</a>
  <a href="#">Contact</a>
</div>

<div id="main">
  <h2>侧边栏实例 - 页面主体向右移动</h2>
  <span style="font-size:30px;cursor:pointer" onclick="openNav()">&#9776; 打开
</span>
</div>
```

```
.sidenav {  
    height: 100%;  
    width: 0px; /*初始不可见*/  
    background-color: #111;  
    padding-top: 60px;  
    overflow-x: hidden; /*x轴溢出处理*/  
    position: fixed;  
    top: 0px;  
    left: 0px;  
    z-index: 999;  
    transition: 0.5s;  
}  
.sidenav a {  
    text-decoration: none;  
    display: block;  
    padding: 8px 8px 8px 32px;  
    font-size: 20px;  
    color: #818181;  
    transition: 0.3s;  
}
```

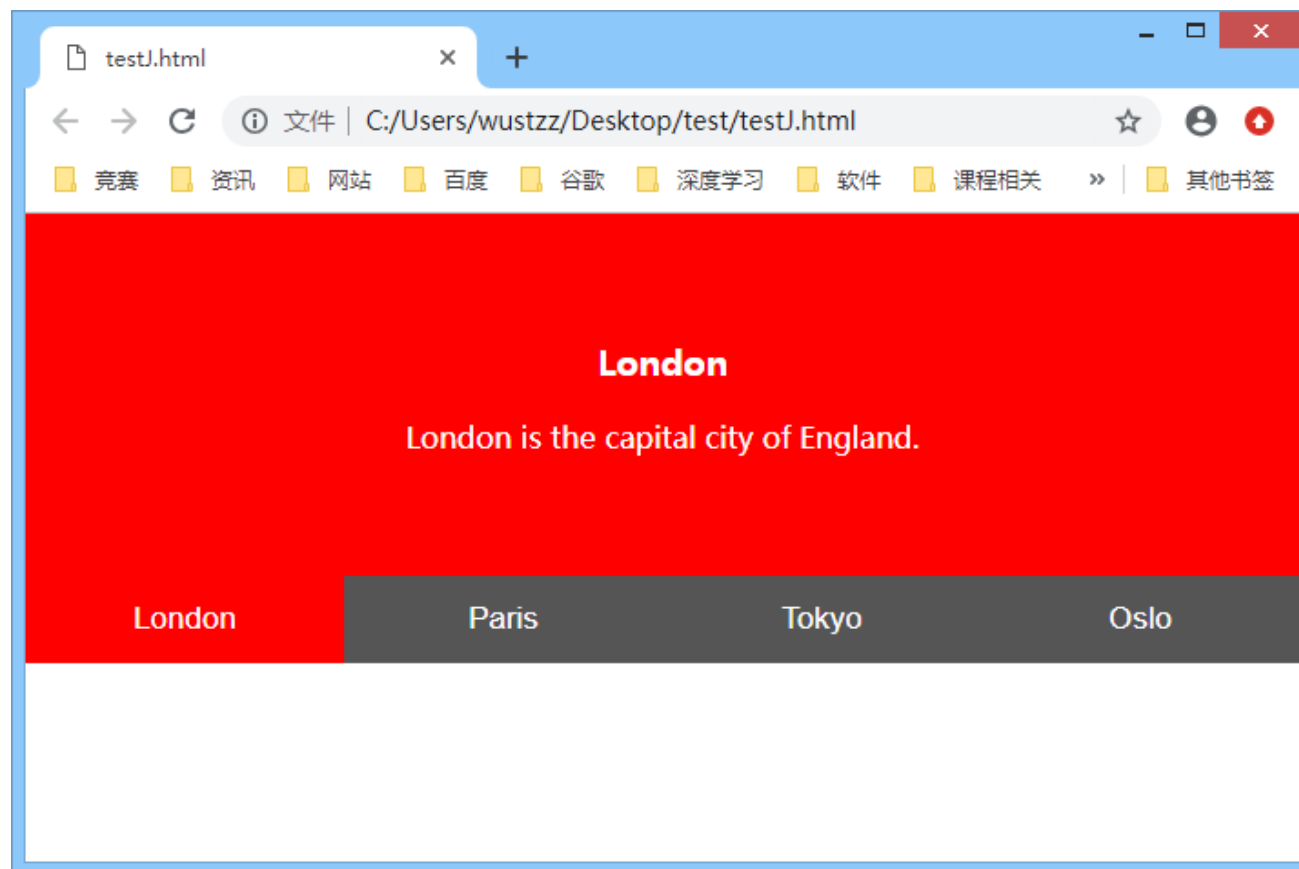
```
.sidenav a:hover {  
    color: #f1f1f1;  
}  
.sidenav .closebtn {  
    position: absolute;  
    top: 0px;  
    right: 25px;  
    font-size: 25px;  
}  
#main {  
    transition: margin-left .5s;  
    padding: 16px;  
}
```

```
<script>
    function openNav() {
        document.getElementById("mySidenav").style.width = "250px";
        document.getElementById("main").style.marginLeft = "250px";
    }

    function closeNav() {
        document.getElementById("mySidenav").style.width = "0";
        document.getElementById("main").style.marginLeft = "0";
    }
</script>
```



## 课后阅读：选项卡效果



```
<div class="container">
  <div class="content">
    <div id="London" class="tabcontent">
      <h3>London</h3>
      <p>London is the capital city of England.</p>
    </div>
    <div id="Paris" class="tabcontent">
      <h3>Paris</h3>
      <p>Paris is the capital of France.</p>
    </div>
    <div id="Tokyo" class="tabcontent">
      <h3>Tokyo</h3>
      <p>Tokyo is the capital of Japan.</p>
    </div>
    <div id="Oslo" class="tabcontent">
      <h3>Oslo</h3>
      <p>Oslo is the capital of Norway.</p>
    </div>
  </div>
  <div class="bottom">
    <button class="tablink" onclick="openCity('London', this, 'red')">London</button>
    <button class="tablink" onclick="openCity('Paris', this, 'green')" id="default">Paris</button>
    <button class="tablink" onclick="openCity('Tokyo', this, 'blue')">Tokyo</button>
    <button class="tablink" onclick="openCity('Oslo', this, 'orange')">Oslo</button>
  </div>
</div>
```

```
body {  
    margin: 0;  
    padding: 0;  
}  
  
.tablink {  
    background-color: #555;  
    color: white;  
    float: left;  
    border: none;  
    outline: none;  
    cursor: pointer;  
    padding: 14px 16px;  
    font-size: 17px;  
    width: 25%;  
}  
  
.tablink:hover {  
    background-color: #777;  
}
```

```
.tabcontent {  
    height: auto;  
    color: white;  
    display: none;  
    padding: 50px;  
    text-align: center;  
}  
  
#London {  
    background-color: red;  
}  
#Paris {  
    background-color: green;  
}  
#Tokyo {  
    background-color: blue;  
}  
#Oslo {  
    background-color: orange;  
}
```

```
<script>
    function openCity(cityName, elmnt, color) {
        var tabcontent = document.getElementsByClassName("tabcontent");
        for (var i = 0; i < tabcontent.length; i++) {
            tabcontent[i].style.display = "none";
        }
        var tablinks = document.getElementsByClassName("tablink");
        for (i = 0; i < tablinks.length; i++) {
            tablinks[i].style.backgroundColor = "";
        }
        document.getElementById(cityName).style.display = "block";
        elmnt.style.backgroundColor = color;
    }
    // 触发 id="defaultOpen" click 事件(初始选中一项)
    document.getElementById("default").click();
</script>
```

[【返回】](#)

## 4. DOM添加

说明：

- 如果 DOM 节点是空的，例如：<div></div>，那么，直接使用：  
**节点.innerHTML** = '<span>child</span>' 相当于“插入”了新的 DOM 节点。
- 如果 DOM 节点不是空的，那就不能这么做，因为innerHTML会直接替换掉原来的所有子节点。

两个办法可以插入新的节点：

1. **节点.appendChild(newnode)**：向节点添加最后一个新子节点newnode
2. **节点.insertBefore(newnode,existingnode)**：在节点的existingnode子节点前插入一个新的子节点newnode。

## 示例：移动 DOM 节点

```
<p id="js">JavaScript</p>  
<div id="list">  
  <p id="java">Java</p>  
  <p id="python">Python</p>  
  <p id="scheme">Scheme</p>  
</div>
```

要求：把<p id="js">JavaScript</p>  
添加到<div id="list">的最后一项

```
var js = document.getElementById('js');  
var list = document.getElementById('list');
```

**list.appendChild(js);** → 从原先的位置删除，再插入到新的位置

## 示例：从零创建一个新的节点，然后插入到指定位置

- (1) 创建新元素：`document.createElement()`
- (2) 找到父级元素：通过id等
- (3) 在指定位置插入元素：`父级元素.appendChild()` / `insertBefore()`

```
<ul id="menu">
```

```
  <li><a href="#">a1</a></li>
```

```
  <li><a href="#">a2</a></li>
```

```
</ul>
```

```
<script>
```

```
  var item = document.createElement("li");
```

```
  item.innerHTML="<a href='#>a3</a>";
```

```
  document.getElementById("menu").appendChild(item);//尾部添加
```

```
</script>
```

问题1：如要插入到a1之前  
问题2：如要插入到a2之前  
答案见下页

```
<ul id="menu">
  <li><a href="#">a1</a></li>
  <li><a href="#">a2</a></li>
</ul>
<script>
  var item = document.createElement("li");
  item.innerHTML="<a href='#>a3</a>";
  // 如要插入到a1之前:
  var menu=document.getElementById("menu");
  menu.insertBefore( item, menu.firstChild );
  // 如要插入到a2之前:
  // var menu=document.getElementById("menu");
  // menu.insertBefore( item, menu.children[1] );
</script>
```

[【返回】](#)



## 5. DOM删除

- 要删除一个节点，首先要获得该节点本身以及它的父节点(parentElement)，然后，调用父节点的 `removeChild` 把自己删掉。

删除节点过程如下：

// 拿到待删除节点：

```
var self = document.getElementById('to-be-removed');
```

// 拿到父节点：

```
var parent = self.parentElement;
```

// 删除：

```
var removed = parent.removeChild(self);
```

```
removed === self; // true 说明待删除节点虽不在文档树中，但还在内存里
```

## DOM删除注意：

- 删除多个节点时，要注意children属性时刻都在变化

```
<div id="parent">
  <p>First</p>
  <p>Second</p>
</div>
<script>
  var parent = document.getElementById('parent');
  parent.removeChild(parent.children[0]);
  parent.removeChild(parent.children[1]); // 报错，想想什么原因
</script>
```

## 课后练习：

学 生 名 单

学号	姓名	院系	操作
2017001	张三	计算机学院	<input type="button" value="删除"/>
2017002	张三	计算机学院	<input type="button" value="删除"/>
2017003	张三	计算机学院	<input type="button" value="删除"/>
2017004	张三	计算机学院	<input type="button" value="删除"/>

问题1：tr 如何自动生成条纹背景色(原来是手工添加 class="alt")

问题2：如何删除当前行？

(每行添加<td><button onclick="deleteRow(this)">删除</button></td>)

## ■ 问题1代码实现:

页面加载完成后触发窗口的onload事件

```
window.onload = function () {  
    //找到所有 tr 行  
    var trs = document.getElementById("table的id").getElementsByTagName("tr");  
    //从第2行开始, 偶数行添加 class="alt"  
    for (var i = 1; i < trs.length; i++) {  
        if (i % 2 == 0) {  
            trs[i].className = "alt";  
        }  
    }  
}
```

## ■ 问题2代码实现：2种方法

```
function deleteRow(obj) {  
    //获取当前tr对象;  
    var tr = obj.parentElement.parentElement;  
    //通过removeChild来删除:  
    tr.parentElement.removeChild(tr);  
}
```

```
function deleteRow(obj) {  
    var tr = obj.parentElement.parentElement; //获得当前行  
    var index = tr.rowIndex; //获得当前行序号  
    var table=document.getElementById("table的id值");  
    table.deleteRow(index);  
}
```

补充：js的table对象本身提供有  
deleteRow(序号)和insertRow(序号)方法

**【[返回](#)】**

## 6. DOM 事件(部分)



常用事件	说明
<a href="#"><u>onload</u></a>	页面或图像加载完成后立即发生
<a href="#"><u>onclick</u></a>	用户点击 HTML 元素
<a href="#"><u>onblur</u></a>	元素失去焦点
<a href="#"><u>onfocus</u></a>	元素获得焦点
<a href="#"><u>onchange</u></a>	当输入域的内容改变并失去焦点时发生
<a href="#"><u>onmouseover</u> / <u>onmouseout</u></a>	鼠标被移到某元素之上/鼠标从某元素移开
<a href="#"><u>onkeyup</u></a>	在键盘按键被松开时发生
<a href="#"><u>onsubmit</u></a>	表单提交事件

【[返回](#)】

## (1) onload事件

```
<script>
function load(){
    alert("页面已加载！");
}
</script>
<body onload="load()">
    <h1>Hello World!</h1>
</body>
```

用匿名函数实现：常用

```
<script>
    window.onload=function(){
        alert("页面已加载！");
    }
</script>
```

备注：onload也可以用在img上

[【返回】](#)

## (2) onclick事件

```
<script>  
function changetext(x){  
    x.innerHTML="hello!";  
}  
</script>  
<h2 onclick="changetext(this)">点击换文本!</h2>
```

简单嵌入写法(代码少情况):

```
<h2 onclick=" this.innerHTML='hello!' ">点击换文本!</h2>
```




## onclick事件(续): 用匿名函数实现

```
<h2 id="test" onclick="changetext(this)">点击换文本!</h2>  
<script>  
  document.getElementById("test").onclick = function() {  
    this.innerHTML="hello!";  
  }  
</script>
```

[【返回】](#)

### (3) onblur事件

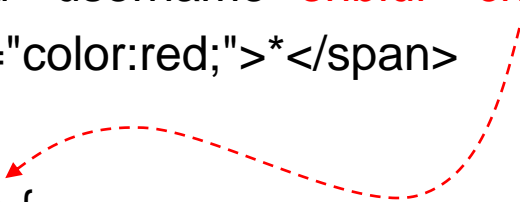
```
用户名: <input type="text" id="username" onblur="upperCase()"><br>
年龄: <input type="text" id="age" onblur="alert(this.value)">
<script>
    function upperCase() {
        var x=document.getElementById("username");
        x.value=x.value.toUpperCase();
    }
</script>
```



## 示例：必填验证

 必填,不能为空

```
<input type="text" id="username" onblur="check(this)"/>
<span id="tip" style="color:red;">*</span>
<script>
    function check(x) {
        var t = document.getElementById("tip");
        if( x.value.length==0 ){
            t.innerHTML = "必填,不能为空";
        }
        else {
            t.innerHTML = "OK";
        }
    }
</script>
```



## 示例：使用正则式验证

```
<script>
function checkUsername() {
    var u = document.getElementById("username");
    var t = document.getElementById("tip");
    var reg = /^[a-z]{6,10}$/;
    if( !reg.test(u.value) ){
        t.innerHTML="用户名为6-10个小写字母";
        //u.focus();    //获得焦点，重新输入
    }
    else{
        t.innerHTML="OK";
    }
}
</script>
```

正则式(有关内容详见本章附录)

## 补充：关于value属性

- 对于text、password、hidden以及select元素，可以直接调用value属性获得用户输入值。
- 对于file，value属性是文件名(含路径)
- 对于radio和checkbox，value属性始终返回的是HTML预设的value值，要需要获得的实际是选中项值，应先用checked属性判断。
- 设置值和获取值类似，对于text、password、hidden以及select，直接设置value属性，对于radio和checkbox，设置checked属性为true或false即可。

[【返回】](#)

## (4) onfocus事件

```
<input type="text" id="username" placeholder="请输入用户名" onfocus="setStyle(this)"  
onblur="clsStyle(this)">
```

```
<br>
```

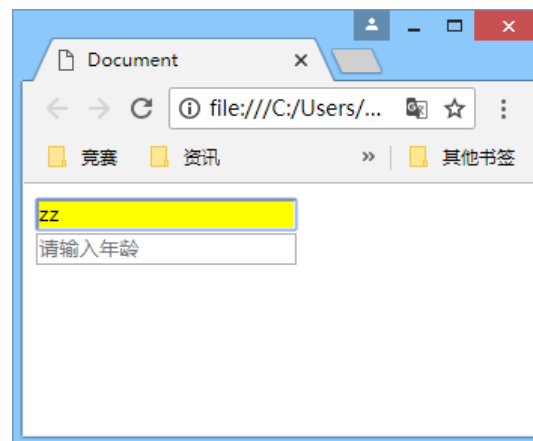
```
<input type="text" id="age" placeholder="请输入年龄" onfocus="setStyle(this)"  
onblur="clsStyle(this)">
```

```
<script>
```

```
function setStyle(x) {  
    x.style.background = "yellow";  
}
```

```
function clsStyle(x) {  
    x.style.background = "white";  
}
```

```
</script>
```



【[返回](#)】

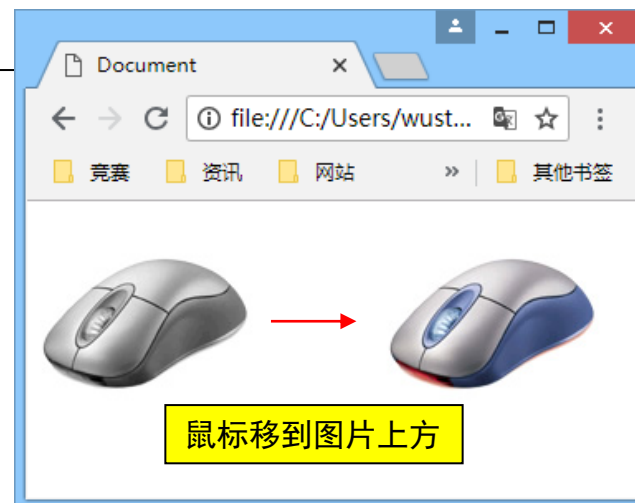
## (5) onchange事件

```
<select name="here"           或用 window.open(this.options[this.selectedIndex].value)  
    onchange="window.location=this.options[this.selectedIndex].value">  
    <option value=""> 请选择 </option>  
    <option value="http://www.163.com"> 网易 </option>  
    <option value="http://www.sina.com">新浪 </option>  
    <option value="http://www.sohu.com">搜狐 </option>  
</select>
```

[【返回】](#)

## (6) onmouseover/onmouseout事件:

```
<a href="#">  
      
</a>
```





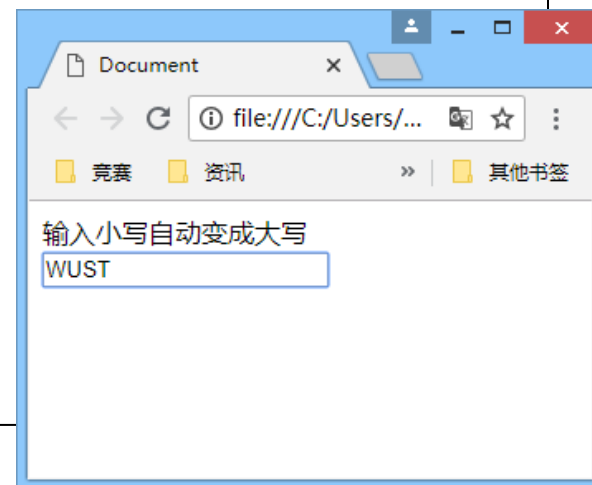
## 示例

```
  
<script>  
    function mouseover(x) {  
        x.width="200";  
        x.height="200";  
        x.style.cursor="pointer";    }  
    function mouseout(x) {  
        x.width='100';  
        x.height='100'; }  
</script>
```

**【[返回](#)】**

## (7) onkeyup事件:

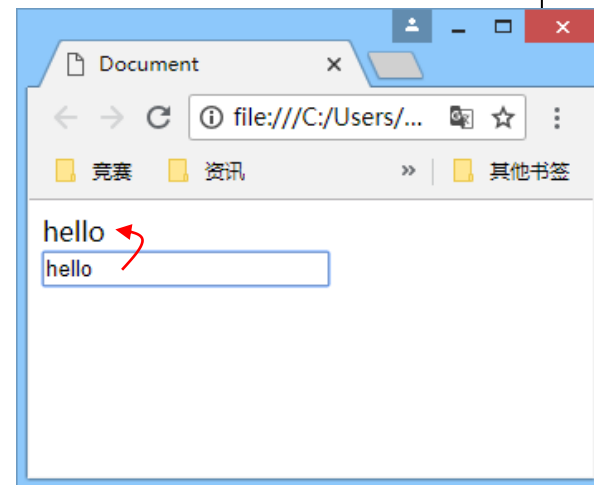
```
<div>输入小写自动变成大写</div>  
<input id="source" type="text" onkeyup="upper()" />  
  
<script>  
    function upper() {  
        var s = document.getElementById("source");  
        s.value=s.value.toUpperCase();  
    }  
</script>
```



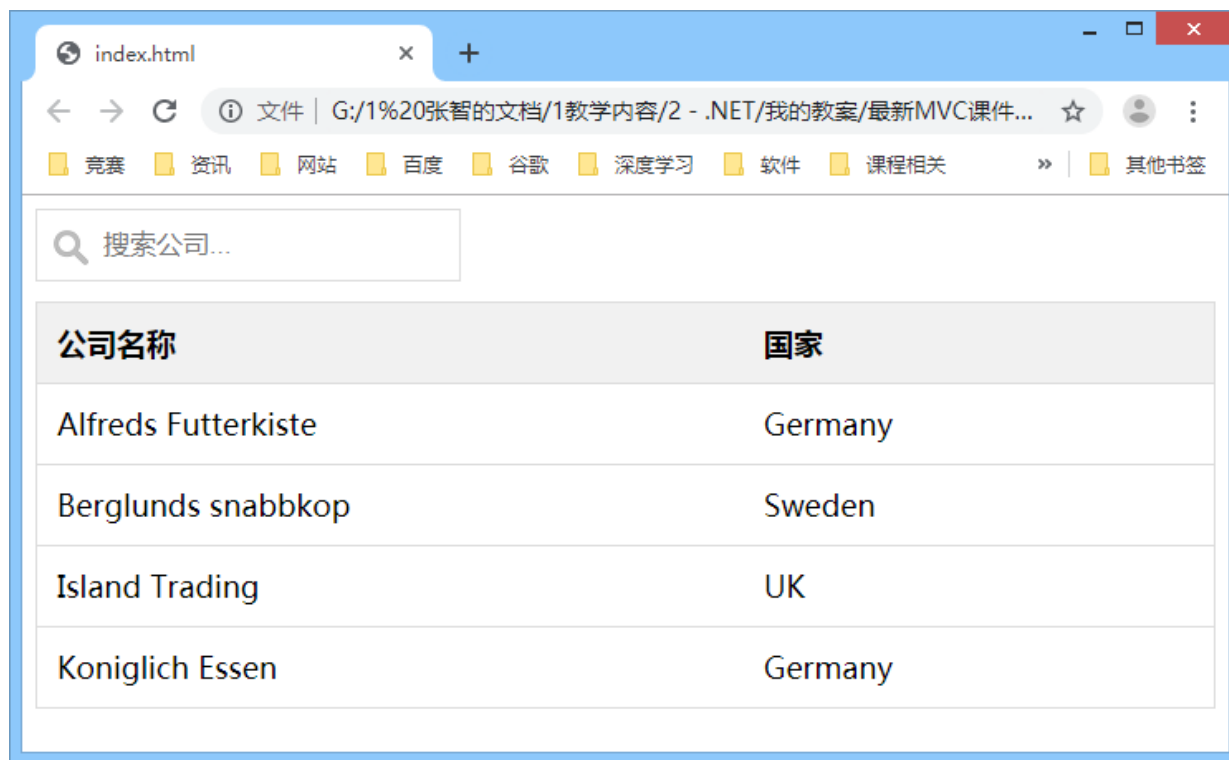
## 示例：将文本框输入内容即时显示

```
<div id="target">显示输入的内容</div>
<input id="source" type="text" onkeyup="dataBind()" />

<script>
    function dataBind(){
        var s=document.getElementById("source");
        var t= document.getElementById("target");
        t.innerHTML=s.value;
    }
</script>
```



## 课后练习：表格数据搜索



## 前端HTML代码：

样式部分略

```
<input type="text" id="myInput" onkeyup="myFunction()" placeholder="搜索公司...">
<table id="myTable">
  <tr class="header">
    <th style="width:60%;">公司名称</th>
    <th style="width:40%;">国家</th>
  </tr>
  <tr>
    <td>Alfreds Futterkiste</td>
    <td>Germany</td>
  </tr>
  <tr>
    <td>Berghlunds snabbkop</td>
    <td>Sweden</td>
  </tr>
  <tr>
    <td>Island Trading</td>
    <td>UK</td>
  </tr>
  <tr>
    <td>Koniglich Essen</td>
    <td>Germany</td>
  </tr>
</table>
```

## JS代码:

```
function myFunction() {  
    var input = document.getElementById("myInput");  
    var filter = input.value.toUpperCase();  
    var table = document.getElementById("myTable");  
    var tr = table.getElementsByTagName("tr");  
    // 循环表格每一行，查找匹配项  
    for (var i = 0; i < tr.length; i++) {  
        var td = tr[i].getElementsByTagName("td")[0]; //获得第一列（公司名称）  
        if (td) {  
            if (td.innerHTML.toUpperCase().indexOf(filter) > -1) {  
                tr[i].style.display = "";  
            } else {  
                tr[i].style.display = "none";  
            }  
        }  
    }  
}
```

[【返回】](#)

## (8) onsubmit事件



- 支持该事件的标签: `<form>`
- onsubmit 事件发生在单击表单的“提交”按钮时。常使用该事件来验证表单的有效性。在事件处理程序中返回 `false` 值可以阻止表单提交。

## 示例1

```
<script>
```

```
function check() {
```

```
    var psd1= document.getElementById("psd1");
```

```
    var psd2= document.getElementById("psd2");
```

```
    if( psd1.value != psd2.value ) { alert("密码不一致"); return false; }
```

```
    else return true;
```

JS中字符串比较用==或!=

```
}
```

```
</script>
```



```
<form name=f1 onsubmit="return check()" action="/Home/Index">
```

```
    输入密码<input type="password" id="psd1"><br/>
```

```
    确认密码<input type="password" id="psd2">
```

```
    <input type="submit" value="提交">
```

```
</form>
```

单击提交按钮后，如果check()函数返回值为true，则提交跳转到action。否则不提交。



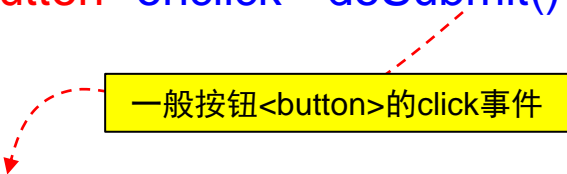
## 示例2

正则式含义：以大写字母开头，后接字母、数字或下划线，总长度[5,8]

```
<script>
function check() {
    var psd = document.getElementById("psd1");
    var reg = /^[A-Z]{1}\w{4,7}$/;
    if ( reg.test(psd.value) ) return true;
    else {    err.innerHTML = "密码以大写字母开头...";
             psd.focus();      return false;    }
}
</script>
<form name=f1 onsubmit="return check()" action="/Home/Index">
    输入密码<input type="text" id="psd1">
    <input type="submit" value="提交"><br/>
    <span id="err" style="color:red;"></span>
</form>
```

## 补充：使用一般按钮进行提交

```
<form id="test-form">
  <input type="text" name="test">
  <button type="button" onclick="doSubmit()">Submit</button>
</form>
<script>
function doSubmit() {
  var f = document.getElementById('test-form');
  // ...
  // 提交form:
  f.submit();
}
</script>
```



一般按钮<button>的click事件

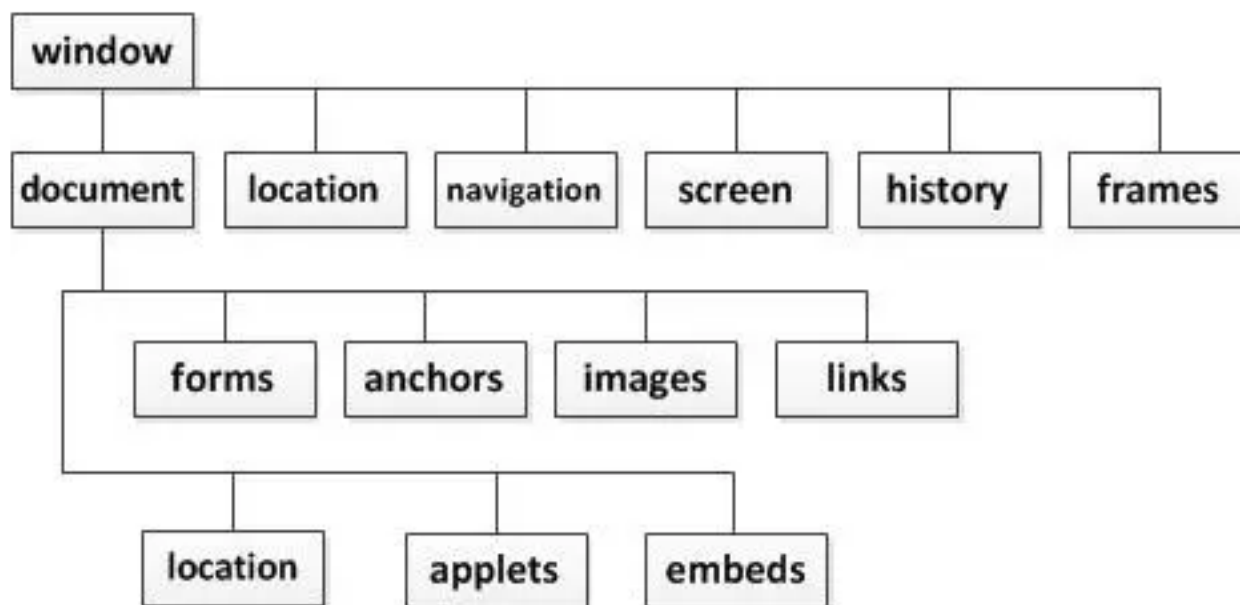
## 课后练习：

利用JavaScript检查用户注册信息是否正确，在以下情况不满足时报错并阻止提交表单：

- 用户名必须是3-10位英文字母或数字；
- 密码必须是6-20位；
- 两次输入密码必须一致。

**【[返回](#)】**

## 6.3 浏览器对象



## 1. window对象

- window对象是**最顶层对象**，不但充当全局作用域，而且表示浏览器窗口。
- 主要属性：
  - window.innerWidth：浏览器窗口的内部宽度
  - window.innerHeight：浏览器窗口的内部高度

内部宽高是指除去菜单栏、工具栏、边框等占位元素后，用于**显示网页的净宽高**

- window.outerWidth：浏览器窗口的整个宽度
- window.outerHeight：浏览器窗口的整个高度

## window对象常用方法

### ■ 常用函数：

- alert()：警告框
- confirm()：确认对话框
- open()：打开浏览器窗口
- close()：关闭浏览器窗口

### ■ 计时方法：

- setInterval()：间隔指定的毫秒数不停的执行指定的代码
- clearInterval()：用于停止setInterval()方法执行的函数代码
- setTimeout()：暂停指定的毫秒数后执行一次指定的代码
- clearTimeout()：用于停止执行setTimeout()方法的函数代码

## 示例1:

试一试：将按钮换成a标签

```
<input type="button" onclick="exit_confirm()" value="退出" />
```

```
<script>
```

```
function exit_confirm() {
```

```
    var r = confirm("确认退出? ");
```

```
    if ( r===true ) {
```

```
        window.close();
```

```
    }
```

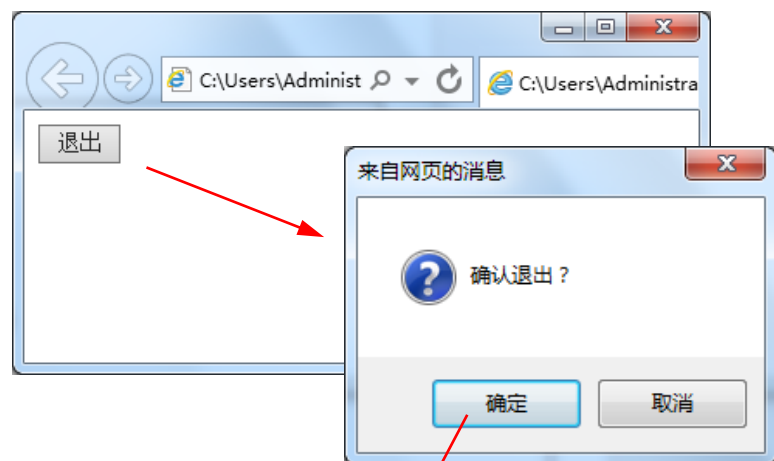
```
}
```

```
</script>
```

window可省略

==: 只比较值

===: 即比较值还比较类型

点击"确认"按钮返回值 true  
点击"取消"按钮返回值 false

## 示例2:

```
<a onclick="window.open('http://www.163.com');">
```

在新标签页中打开

```
</a>
```

```
<a onclick="window.open('http://www.163.com', '_self');">
```

在当前页打开

```
</a>
```

```
<a onclick="window.open('http://www.163.com', '_blank','width=800,height=600');">
```

打开新窗口

```
</a>
```



## 示例3：页面时钟

试一试：添加开始和停止按钮

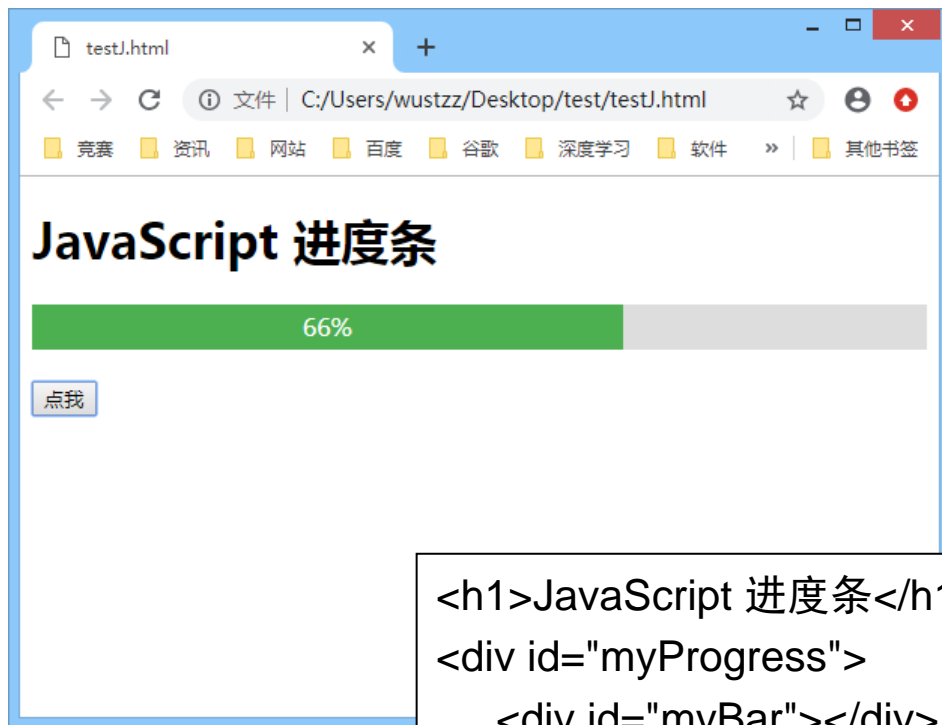
```
<span id="showtime"></span>
<script>
    var mytimer = function () {
        var d = new Date(); //获得系统当前日期时间
        document.getElementById("showtime").innerHTML =
            d.toLocaleString();
    };
    setInterval(mytimer, 1000);
</script>
```

此处直接用函数变量，不要用mytimer()

setInterval(): 按照指定的周期（以毫秒计）来调用函数

如何停止执行？ var id = setInterval(mytimer, 1000);  
clearInterval(id);

## 示例4：进度条



```
<h1>JavaScript 进度条</h1>
<div id="myProgress">
  <div id="myBar"></div>
</div>
<br>
<button onclick="move()">点我</button>
```

```
<style>
  #myProgress {
    width: 100%;
    background-color: #ddd;
  }
  #myBar {
    width: 1%;
    height: 30px;
    background-color: #4CAF50;
    text-align: center;
    line-height: 30px;
    color: white;
  }
</style>
```

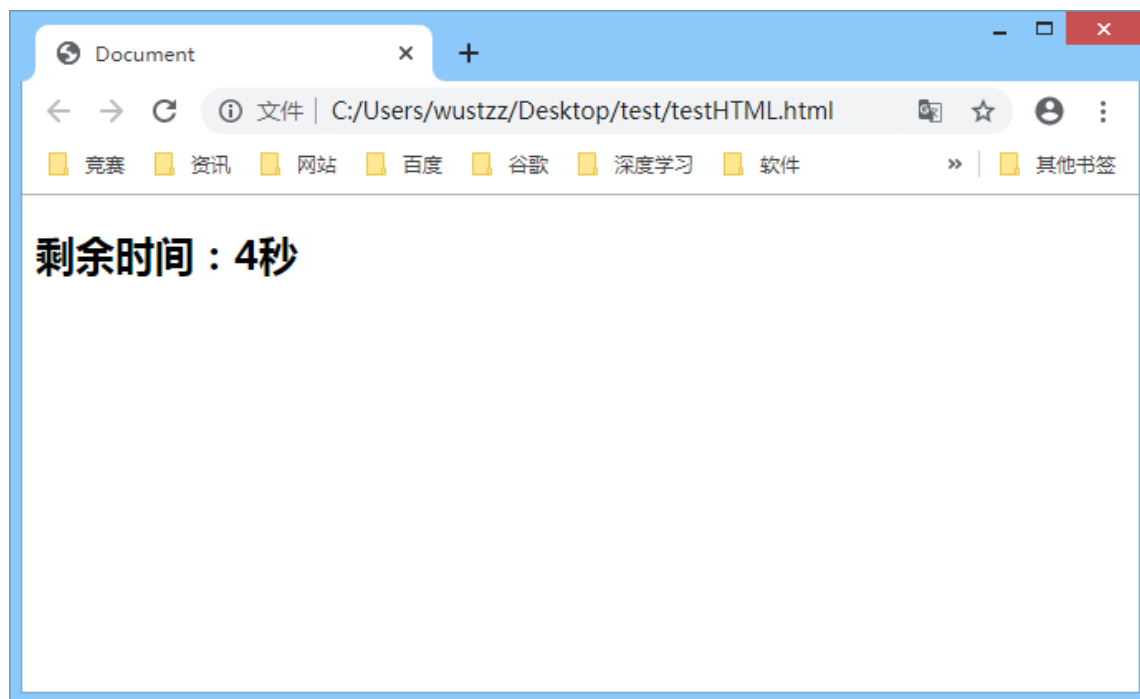
```
function move() {  
    var elem = document.getElementById("myBar");  
    var width = 1;  
    var id = setInterval(frame, 10); //每隔0.01秒执行一次frame函数  
  
    function frame() { // 该函数是嵌套函数，也可改成一般函数  
        if (width >= 100) {  
            clearInterval(id);  
        } else {  
            width++;  
            elem.style.width = width + '%';  
            elem.innerHTML = width + '%';  
        }  
    }  
}
```

此处如要用frame(), 则需加引号即: "frame()"

试一下

## 补充练习

- 要求：5秒倒计时，跳转到学校官网。



## 参考答案

```
<h2>剩余时间: <span id="clock">5</span> 秒</h2>
<script>
    var t = 5;
    var x = document.getElementById("clock");
    function fun() {
        t--;
        x.innerHTML = t;
        if (t <= 0) {
            clearInterval(id);
            window.location = "http://www.wust.edu.cn";
        }
    }
    var id = setInterval(fun, 1000);
</script>
```

## 2. navigator对象

- navigator对象表示浏览器的信息。
- 主要属性：
  - navigator.appName: 浏览器名称
  - navigator.appVersion: 浏览器版本
  - navigator.language: 浏览器设置的语言
  - navigator.platform: 操作系统类型
  - navigator.userAgent: 浏览器设定的User-Agent字符串

### 3. screen对象

- screen对象表示屏幕的信息。
- 主要属性：
  - screen.width: 屏幕宽度，以像素为单位
  - screen.height: 屏幕高度，以像素为单位
  - screen.colorDepth: 返回颜色位数，如8、16、24

## 4. location对象

- location对象表示当前页面的URL信息。

- URL例如：`http://www.abc.com:8080/path/index.html?a=1&b=2#TOP`

- 主要属性：针对上面URL

- **location.href**：设置或返回当前完整的URL：如上URL所示
- **location.protocol**：设置或返回当前 URL 的协议：'http:'
- **location.host**：设置或返回当前 URL 的主机名：'www.abc.com'
- **location.port**：设置或返回当前 URL 的端口：'8080'
- **location.pathname**：设置或返回当前 URL 的路径名：'/path/index.html'
- **location.search**：设置或返回当前 URL 的查询部分(? 之后)：'?a=1&b=2'
- **location.hash**：返回 URL 的锚部分(#号开始)：'#TOP'



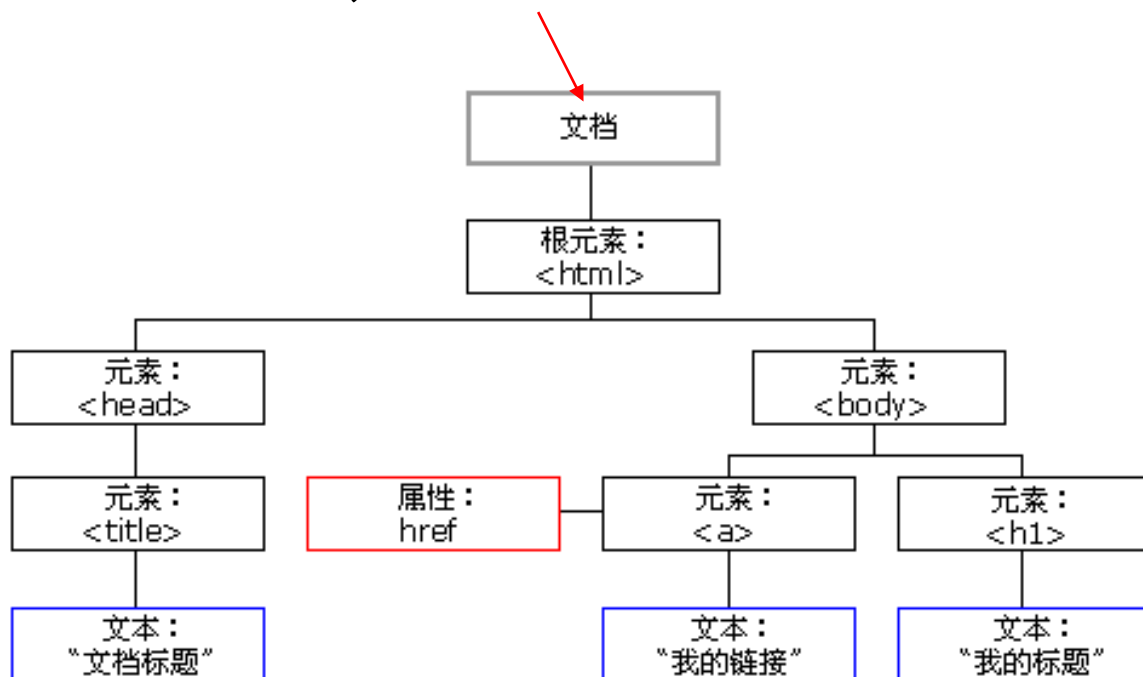
## 5. history对象

遗留对象，已不推荐使用

- history对象保存了浏览器的历史记录。
- 主要属性：
  - location.length：返回历史列表中的网址数
- 主要方法：
  - history.back()：与在浏览器点击后退按钮相同
  - history.forward()：与在浏览器点击向前按钮相同
  - history.go(number|URL)：进入历史中的某个页面
    - history.go(-1)：等同于 history.back()
    - history.go(1)：等同于 history.forward()

## 6. document对象

- **document对象表示当前页面。** 由于HTML在浏览器中以DOM形式表示为树形结构，document对象就是**DOM树的根节点**。



[【返回】](#)

## 6.4 JSON对象

- JSON: JavaScript Object Notation
- JSON 是轻量级的数据交换格式 (纯文本)
- JSON 比 XML 更小、更快, 更易解析

一个天气预报返回json数据示例: 武汉今天和未来5天天气信息

<https://api.openweathermap.org/data/2.5/forecast?q=Wuhan,cn&appid=800f49846586c3ba6e7052cfc89af16c>

## JSON语法

- JSON语法是JS对象表示语法的子集，有如下特点：

- ① 数据在键值对中(键名即属性名必须加双引号)
- ② 数据由逗号分隔
- ③ 花括号保存对象
- ④ 方括号保存数组

- JSON 可通过 JavaScript 进行解析

JSON 值可以是：数字、字符串、逻辑值、数组(在中括号中)、对象(在大括号中)、null

注意JSON不能存储Date对象，如果需要则用字符串表示

## json对象示例

```
var jsonObj = {
```

```
  "name": "wust",
```

```
  "url": "www.wust.edu.cn",
```

```
  "age": 120
```

```
};
```

访问：与js对象访问相同

```
jsonObj.url 或 jsonObj["url"]    // "www.wust.edu.cn"
```

```
jsonObj.url = "... "    //修改
```

注：js对象的key值一般不用双引号引着

JSON 值可以是：数字、字符串、逻辑值、数组(在中括号中)、对象(在大括号中)、null

//json对象遍历（注：x是key值）

```
for( var x in jsonObj ) {
```

```
    document.write( x + ":" + jsonObj[x] + "<br>" )
```

```
}
```

## json嵌套示例

```
var jsonObj = {  
    "name": "datacode",  
    "code": {  
        "cs": 100,  
        "se": 101,  
        "ne": 102  
    }  
};
```

嵌套

遍历嵌套：

```
for(var key in jsonObj.code) {  
    document.write( jsonObj.code[key] )  
}
```

访问：

`jsonObj.code.cs` 或 `jsonObj.code["cs"]`    //100

`jsonObj.code.cs="1000"`    //修改

## json值为数组示例

```
var jsonObj = {  
    "name": "网站",  
    "num": 3,  
    "sites": ["Google", "Sina", "Taobao"]  
};
```

→ [ ]: 是js数组

访问:

```
jsonObj.sites[0]    // "Google"
```

```
jsonObj.sites[0] ="Wust"    //修改
```

遍历数组:

```
for( var i=0; i < jsonObj.sites.length; i++ )  
    document.write( jsonObj.sites[i] )
```

## json值为json数组示例

```
var jsonObj = {  
    "sites": [ { "name": "wust",    "url": "www.wust.edu.cn" },  
               { "name": "google", "url": "www.google.com" },  
               { "name": "weibo",   "url": "www.weibo.com" } ]  
};
```

数组元素是json对象

访问：如

```
jsonObj.sites[0].url    // "www.wust.edu.cn"
```

```
jsonObj.sites[0].url="..."    //修改
```

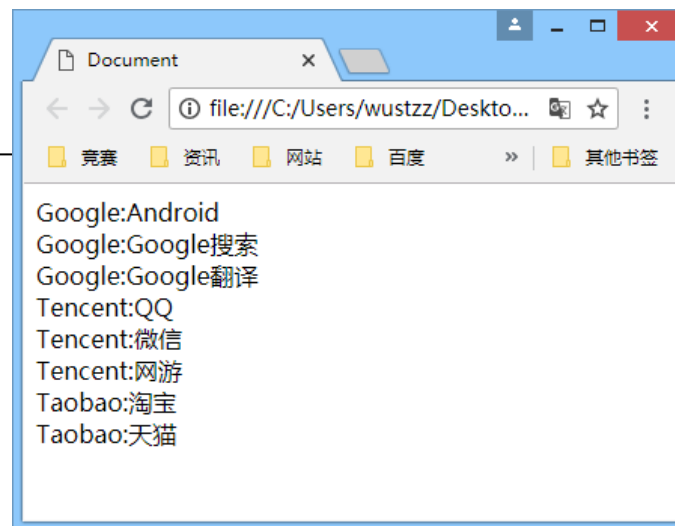
遍历数组元素：

```
for ( var i=0; i < jsonObj.sites.length; i++ )  
    document.write( jsonObj.sites[i].url )
```



## 练习题：如何遍历info值

```
myjsonObj = {  
    "name": "网站",  
    "num": 3,  
    "sites": [ { "name": "Google", "info": ["Android", "Google搜索", "Google翻译"] },  
                { "name": "Tencent", "info": ["QQ", "微信", "网游"] },  
                { "name": "Taobao", "info": ["淘宝", "天猫"] }  
            ]  
}
```



## 参考答案：

```
for ( var i = 0; i < myjsonObj.sites.length; i++ )  
    for( var j = 0; j < myjsonObj.sites[i].info.length; j++ ) {  
        document.write( myjsonObj.sites[i].name + ":" )  
        document.write( myjsonObj.sites[i].info[j] + "<br>" );  
    }
```

## 课堂练习

- 根据天气预报返回的json数据，获取今天或未来所有12:00:00的气温值。

参考代码:

先将json数据保存为weather.js文件, 并用 var weather= ... 接收

```
<div id="show"></div>
<script src="weather.js"></script>
<script>
    var show=document.getElementById("show");
    show.innerHTML="未来5天12点气温: <br>";

    var list= weather.list; // list字段存放天气信息, 该值为json数组
    for(var i=0;i<list.length;i++){
        var d = new Date(list[i].dt_txt); // dt_txt字段存放的是期时间串
        if( d.getHours()==12 ){
            var temp = list[i].main.temp; //main字段是一个json对象, 其temp为温度值
            show.innerHTML+=d.toLocaleDateString()+" 12:00温度: " + temp + "<br>";
        }
    }
</script>
```

未来5天12点气温:

2020/9/14 12:00温度: 296.29

2020/9/15 12:00温度: 294.85

2020/9/16 12:00温度: 293.92

2020/9/17 12:00温度: 291.18

2020/9/18 12:00温度: 295.62

## JSON.parse()用法



称为：json反序列化

- JSON.parse() 方法将JSON格式字符串转换为**JS 对象**(属性名没有双引号)。
- 解析前要确保数据是标准的JSON格式，否则会解析出错。

```
var jstr = '{"name":"wust", "url":"www.wust.edu.cn", "age":120}'
```

json串

```
var obj = JSON.parse(jstr);
```

转化为js对象

```
// 结果 obj= { name:"wust", url:"www.wust.edu.cn", age:120 }
```

json数组串

```
var jarrstr = '[{"id":101,"name":"计算机科学"}, {"id":102,"name":"软件工程"}]'
```

```
var arr = JSON.parse(jarrstr);
```

转化为js对象数组

```
// 结果 arr = [ {id:101,name:"计算机科学"}, {id:102,name:"软件工程"} ]
```

## 注意：json日期数据处理

json不能存储Date对象，需要用字符串表示

```
var jstr = '{ "name" : "wust", "initDate" : "1898-03-16"}';  
var obj = JSON.parse(jstr); //转换为js对象  
obj.initDate = new Date(obj.initDate); //还原为Date 类型  
console.log( obj.initDate.toLocaleDateString() ) //控制台显示: '1898/3/16'
```

## JSON.parse()高级用法

JSON.parse(text, reviver):

reviver: 可选，一个转换结果的函数， 将为对象的每个成员调用此函数。

```
var jstr = '{"name":"wust", "url":"www.wust.edu.cn", "birthday":"1898-11-21"}';
var jsonObj = JSON.parse(jstr, function (key, value) {
    if ( key == "birthday" ) {
        var diff = new Date() - new Date(value);    //两个日期差：相隔毫秒数
        var year = parseInt(diff/1000/60/60/24/365); //相差的年数
        return year;
    } else {
        return value;    // 其他值原样不动返回
    }
});
console.log(jsonObj.birthday);
```

本例功能：将birthday字段转换为距今多少年

试一试：将name值变成大写

## JSON.stringify()用法



称为：json序列化

- 将json对象转为串（注：也可把js对象、数组转换成字符串）

一般用法：将js对象或json对象转为json串

```
var obj = { name:"wust", url:"www.wust.edu.cn", age:120 } ; //js对象
```

```
var jstr = JSON.stringify(obj);
```

```
//jstr = '{ "name":"wust", "url":"www.wust.edu.cn", "age":120 }'
```

将一般数组转为字符串：

```
var arr = [ "Google", "Taobao", "Facebook" ];
```

```
var str = JSON.stringify(arr); // str = '[ "Google", "Taobao", "Facebook" ]'
```

[【返回】](#)



## 附录1：RegExp 对象

- 字符串是用得较多的一种数据结构，比如判断一个字符串是否是合法的Email地址，虽然可编程提取@前后子串来处理，但这样做不但麻烦，而且代码难以复用。
- 正则表达式是一种用来匹配字符串的强有力的武器。
- **设计思想：**用一种描述性语言来给字符串定义规则，凡是符合规则的字符串，就认为它“匹配”了，否则就是不合法的。

## 正则式创建：



### 方法1：常用

```
var 变量 = /正则表达式/;
```

前后'/'不能少

### 方法2：

```
var 变量 = new RegExp('正则式');
```

字符串( / 变引号 )

例如：

```
var reg = /\d/; // 元字符 \d 含义：0-9任意数字
```

```
var reg = new RegExp('\\d');
```

注意：由于是字符串  
所以此处两个 \\ 实际上是一个 \ (转义符)

## 正则式示例：

正则式含义：匹配三个连续的数字

```
var reg = /\d\d\d/;  
var s = "a123b456";  
var flag = reg.test(s);           // flag=true  
var s_new = s.match(reg);         // 匹配成功, s_new=['123']
```

由于本例不是全局匹配  
所以结果不是['123','456']

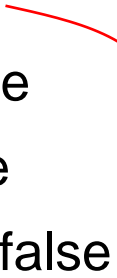
### 函数说明：

RegExp.test(str): 用于检测字符串str是否符合正则式规则，符合则为真，否则为假  
String.math(reg): 找到与正则式reg的匹配串，匹配结果存放在数组



## 正则式示例：

```
var reg = /^[a-z]{6,10}$/;  
reg.test("abc")      // false  
reg.test("abcdef")   // true  
reg.test("abcdefghijk") //false
```



正则式含义：以小写字母开头和结尾，中间5-9个小写字母

其中： / / 为模式固定格式

[a-z]：表示任意小写字母，{6,10}表示：6-10个

^表示串的开头，\$表示串的结束

阅读：

```
var re = /^d{3}-d{3,8}$/;  
re.test('010-12345'); // true  
re.test('010-1234x'); // false  
re.test('010 12345'); // false
```

## 使用正则式切分字符串

问题提出：

无法识别连续的空格

'a b c'.split(' '); //结果：['a', 'b', "", "", 'c'] 数组长度=5

1个空格

3个空格

1个空格

会得到2个空串

使用正则式：

'a b c'.split(**\s+**); //结果：['a', 'b', 'c'] 数组长度=3

没有空串

\s：匹配任何空白字符  
+：一个或多个

## 使用正则式切分字符串练习：

```
1、 'a,b, c d'.split( _____ );    // ['a', 'b', 'c', 'd']
2、 'a,b;; c d'.split( _____ );      // ['a', 'b', 'c', 'd']
3、 var names = ' Harry Trump ;Fred Barney; Helen Rigby ; Bill Abel ;Chris Hand ';
   var re = _____ ; //去掉名字间的空格和分号
   var list = names.trim().split(re);
   //list=["Harry Trump", "Fred Barney", "Helen Rigby", "Bill Abel", "Chris Hand"]
```

答案： 1、 /[\s\,]+/      2、 /[\s\,\;]+/      3、 /\s\*\;\s\*/

## 使用正则式匹配字符串

在字符串检索指定值，或找到正则式匹配，将匹配的结果存放在数组

```
var str="abcd abCd";
```

```
var patt=/[^a-c]/g;
```

含义：对不在字符范围 [a-c] 内的字符进行全局搜索

```
var arr=str.match(patt); // arr: ['d',' ','C','d'] 第二个为空格串 arr.length=4
```

改一下：

```
var patt=/[^a-c]/gi;
```

g:全局匹配 i: 忽略大小写

```
var arr=str.match(patt); // arr: ['d',' ','d'] arr.length=3
```

```
var str="This is an\n antzone good";
```

```
var reg=/an$/;
```

含义：以an作为结尾(\$表示结尾)

```
var arr=str.match(reg) // arr: null 没有匹配
```

改一下：

```
var reg=/an$/m;
```

m: 多行匹配

```
var arr=str.match(reg) // arr: ["an"]
```



## 使用正则式分组功能提取子串

如果正则式中定义了组，就可以使用RegExp.exec()提取出子串。

**示例：提取出号码串如 '027-12345' 格式的区号和号码**

```
var re = /(\d{3})\d{4})-(\d{3,8})$/;    //定义了两个组，两组间用 '-' 连接  
re.exec('027-12345');    // ['027-12345', '027', '12345', ...]  
re.exec('0714-1234567');    // ['0714-1234567', '0714', '1234567', ...]  
re.exec('027 12345');    // null 匹配很严格  
re.exec('%027-12345%');    // null 匹配很严格
```

函数说明：

RegExp.exec(str)：匹配str成功后，返回一个数组，该数组第一个元素是正则表达式匹配到的整个字符串，后面的元素是与各个分组匹配成功的子串(如果有分组的话)，该数组还有一些其他元素(略)。如匹配失败时返回null。

## 正则捕获

正则式中如果有分组，那么分组匹配的内容，会被保存到RegExp.\$1-RegExp.\$99中，这种现象叫做正则捕获。

示例：

```
var re = /(\d{3})\d{4})-(\d{3,8})$/;    //定义了两个组，两组间用 '-' 连接  
re.exec('027-12345');    // ['027-12345', '027', '12345']  
console.log( RegExp.$1 );    // '027'  
console.log( RegExp.$2 );    // '12345'
```

## 补充: exec() 没有分组情况示例

在非全局模式下, exec()和match()函数是一样的, 只能够在字符串中匹配一次

```
var str = "This is your book. Here you are.";
var reg = /you/;
reg.exec(str); // ['you',....] 只匹配一次, 即使添加全局模式g
```

全局模式下, 可多次执行exec()来搜索一个匹配的字符串(不是自动滴), 正则式本身会更新lastIndex属性。也可通过设置lastIndex属性指定开始查找的位置

```
var str = "the name 123 and 456";
var reg = /\d{3}/g;
console.log(reg.exec(str)); // ["123",...] 此时 reg.lastIndex = 9
console.log(reg.exec(str)); // ["456",...] 此时 reg.lastIndex = 15
console.log(reg.exec(str)); //null
```

如果此处添加: reg.lastIndex = 15; 则第一次exec后结果为["456", ...]

## 正则式修饰符：

修饰符	描述
<b>g</b>	全局匹配（查找所有匹配而非在找到第一个匹配后停止）
<b>i</b>	对大小写不敏感的匹配(忽略大小写)。
<b>m</b>	<b>多行</b> 匹配。

基本语法：

```
var re=new RegExp('正则式', '修饰符');
```

简洁方式：

```
var re=/正则式/修饰符;
```

## 修饰符示例

```
var str="abcd abCd";
```

```
var patt=/^a-c/g;
```

含义：对不在字符范围 [a-c] 内的字符进行全局搜索

```
var arr=str.match(patt); // arr: ['d',' ','C','d'] arr.length=4
```

改一下：

```
var patt=/^a-c/gi;
```

g:全局匹配 i: 忽略大小写

```
var arr=str.match(patt); // arr: ['d',' ','d'] arr.length=3
```

```
var str="This is an\n antzone good";
```

```
var reg=/an$/;
```

含义：以an作为结尾(\$表示结尾)

```
var arr=str.match(reg) // arr: null 没有匹配
```

改一下：

```
var reg=/an$/m;
```

m: 多行匹配

```
var arr=str.match(reg) // arr: ["an"]
```

## 一些常用的正则式：

数字: `"^[0-9]*$"`

n位的数字: `"^\d{n}$"`

至少n位的数字: `"^\d{n,}$"`

m~n位的数字: `"^\d{m,n}$"`

零和非零开头的数字: `"^(0|[1-9][0-9]*)$"`

有两位小数的正实数: `"^[0-9]+(\.[0-9]{2})?$"`

有1~3位小数的正实数: `"^[0-9]+(\.[0-9]{1,3})?$"`

非零的正整数: `"^\+[1-9][0-9]*$"`

非零的负整数: `"^\-[1-9][0-9]*$"`

长度为3的字符: `"^.{3}$"`

由26个英文字母组成的字符串: `"^[A-Za-z]+$"`

由26个小写英文字母组成的字符串: `"^[a-z]+$"`

由数字和26个英文字母组成的字符串: `"^[A-Za-z0-9]+$"`

由数字、26个英文字母或者下划线组成的字符串: `"^\\w+$"`

验证用户密码: `"^[a-zA-Z]\\w{5,17}$"` 正确格式为: 以字母开头, 长度在6~18之间, 只能包含字符、数字和下划线

验证是否含有`^%&',;=?$\"`等字符: `"[^%&',;=?$\\x22]+"`

只能输入汉字: `"^[\\u4e00-\\u9fa5]{0,}$"`

JS变量命名规则: `[a-zA-Z_\\$][0-9a-zA-Z_\\$]*` 可以匹配由字母或下划线、\$开头, 后接任意个由一个数字、字母或者下划线、\$组成的字符串

验证Email地址: `"^\\w+([-+.]\\w+)*@\\w+([-.]\\w+)*\\.\\w+([-.]\\w+)*$"`

验证InternetURL: `"^http://([\\w-]+\\.)+[\\w-]+(/[\\w-./?%&=]*)?$"`

验证电话号码: `"^((\\d{3,4}-)|\\d{3,4}-)?\\d{7,8}$"` 正确格式为: "XXX-XXXXXXX"、"XXXX-XXXXXXX"、"XXX-XXXXXXX"、"XXXXXXX"和"XXXXXXX"

验证身份证号(15位或18位数字): `"^\\d{15}|\\d{18}$"`

验证手机号: `/^1[345678]\\d{9}$/`

验证一年的12个月: `"^(0?[1-9]|1[0-2])$"` 正确格式为: "01"~"09"和"1"~"12"

验证一个月的31天: `"^((0?[1-9])|((1|2)[0-9])|30|31)$"` 正确格式为;"01"~"09"和"1"~"31"

## 正则式元字符：

元字符	描述
<b>[xyz]</b>	字符集合。 <b>匹配所包含的任意一个字符</b> 。例如， '[abc]' 可以匹配 "plain_c" 中的 'a'和'c'。
<b>[^xyz]</b>	非字符集合。匹配未包含的任意字符。例如， '[^abc]' 可以匹配 "plain" 中的'p'、'l'、'i'、'n'。
<b>[a-z]</b>	<b>字符范围</b> 。匹配指定范围内的任意字符。例如： '[a-z]' 匹配 'a' 到 'z' 范围内的任意小写字母字符 '[A-Z]'匹配 'A' 到 'Z' 范围内的任意大写字母字符 '[0-9]' 匹配'0' 到 '9' 范围内的任意数字字符
<b>[^a-z]</b>	非字符范围。匹配任何不在指定范围内的任意字符。例如： '[^a-z]' 可以匹配任何不在 'a' 到 'z' 范围内的任意字符。
<b>x y</b>	<b>匹配 x 或 y</b> 。例如： 'z food' 能匹配 "z" 或 "food"。'(z f)ood' 则匹配 "zood" 或 "food"
<b>()</b>	<b>分组，将括号里面的字符作为整体进行匹配</b> 。例如： '(J j)ava(S s)cript' 可匹配'JavaScript'、'Javascript'、'javaScript'或者'javascript'



元字符	描述
<b>.</b> (点号)	匹配除换行符 ( <code>\n</code> 、 <code>\r</code> ) 之外的 <b>任何单个字符</b> 。 要匹配包括 <code>\n</code> 在内的任何字符, 请使用像 <code>"(.\\n)"</code> 的模式。
<b>\w</b>	<b>匹配字母、数字、下划线</b> 。等价于 <code>'[A-Za-z0-9_]'</code>
<b>\W</b>	匹配非字母、数字、下划线。等价于 <code>'[^A-Za-z0-9_]'</code>
<b>\d</b>	<b>匹配一个数字字符</b> 。等价于 <code>[0-9]</code>
<b>\D</b>	匹配一个非数字字符。等价于 <code>[^0-9]</code>
<b>\s</b>	<b>匹配任何空白字符</b> , 包括空格、制表符、换页符等等。等价于 <code>[\f\n\r\t\v]</code> (注: <code>\f</code> 前面有一个空格字符) 其中: <code>\f</code> 换页符、 <code>\n</code> 换行符、 <code>\r</code> 回车符、 <code>\t</code> 制表符、 <code>\v</code> 垂直制表符
<b>\S</b>	匹配任何非空白字符。等价于 <code>[^\f\n\r\t\v]</code> 。
<b>\b</b>	匹配一个单词边界, 也就是指单词和空格间的位置。例如: ' <code>er\b</code> ' 可以匹配 "never" 中的 'er', 但不能匹配 "verb" 中的 'er'。
<b>\B</b>	匹配非单词边界。' <code>er\B</code> ' 能匹配 "verb" 中的 'er', 但不能匹配 "never" 中的 'er'。

元字符	描述
<code>\0</code>	匹配 NULL 字符。
<code>\n</code>	匹配换行符。
<code>\f</code>	匹配换页符。
<code>\r</code>	匹配回车符。
<code>\t</code>	匹配制表符。
<code>\v</code>	匹配垂直制表符。
<code>\xn</code>	匹配 <code>n</code> ，其中 <code>n</code> 为十六进制值。十六进制转义值必须为确定的两个数字长。例如： <code>\x41</code> 匹配 "A"。 <code>\x041</code> 则等价于 <code>\x04</code> & "1"
<code>\uxxxx</code>	匹配以十六进制数 <code>xxxx</code> 规定的 Unicode 字符。

注意：'n' 匹配字符 "n"      \n' 匹配一个换行符

其他特殊字符：\' 匹配一个'    \' 匹配一个\'    \. 匹配一个.    \\* 匹配一个\*

\+ 匹配一个+    \? 匹配一个?    \| 匹配一个|    \(、\)、\[、\]、\{、\}

元字符	描述
<b>+</b>	匹配前面的子表达式 <b>一次或多次</b> ，等价于 $\{1,\}$ 。例如： 'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。 'a+' 能匹配 "candy" 中的 "a"，"caaaaaaandy" 中所有的 "a"。
<b>*</b>	匹配前面的子表达式 <b>零次或多次</b> ，等价于 $\{0,\}$ 。例如： 'zo*' 能匹配 "z" 以及 "zoo"。 'bo*' 能匹配 "A ghost boooood" 中的 "boooo"，"A bird warbled" 中的 "b"，但是不匹配 "A goat grunted"
<b>?</b>	匹配前面的子表达式 <b>零次或一次</b> ，等价于 $\{0,1\}$ 。例如： 'do(es)?' 可以匹配 "do" 或 "does" 。 'e?le?' 可匹配 "angel" 中的 "el"，"angle" 中的 "le"。
<b>{n}</b>	n 是一个非负整数， <b>匹配确定的 n 次</b> 。例如： 'a{2}' 不匹配 "candy," 中的 "a"，可匹配 "caandy," 中的两个 "a"， 可匹配 "caaandy." 中的前两个 "a"。
<b>{n,}</b>	n 是一个非负整数。 <b>至少匹配n 次</b> 。例如： 'a{2,}' 不匹配 "candy" 中的 "a"，但是匹配 "caandy" 和 "caaaaaaandy" 中所有的 "a"。

元字符	描述
<code>{n,m}</code>	m 和 n 均为非负整数，其中 $n \leq m$ 。最少匹配 n 次且最多匹配 m 次。请注意在逗号和两个数之间不能有空格。例如： '/a{1,3}' 不匹配 "cndy"，匹配 "candy," 中的 "a"，匹配 "caandy," 中的两个 "a"，匹配 "caaaaaaandy" 中的前面三个 "a"。
<code>n\$</code>	匹配任何结尾为 n 的字符串。例如：'\d\$'表示必须以数字结束
<code>^n</code>	匹配任何开头为 n 的字符串。例如：'^\d'表示必须以数字开头 例如：'js'也可以匹配'jsp'，'^js\$'就变成了整行匹配，就只能匹配 'js'
<code>?</code>	当该字符紧跟在任何一个其他限制符 (*, +, ?, {n}, {n,}, {n,m}) 后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。 例如：对于字符串 "oooo", 'o+?' 将匹配单个 "o"，而 'o+' 将匹配所有 'o'。

注意：和前面 ? 区别

元字符	描述
(?=n)	匹配任何其后紧接指定字符串 n 的字符串。例如： "Windows(?=95 98 NT 2000)"能匹配"Windows2000"中的"Windows"，但不能匹配"Windows3.1"中的"Windows"。
(?!n)	匹配任何其后没有紧接指定字符串 n 的字符串。例如： "Windows(?!95 98 NT 2000)"能匹配"Windows3.1"中的"Windows"，但不能匹配"Windows2000"中的"Windows"。
(?<=n)	与(?=n)类似，只是方向相反。例如： 例如，"(?<=95 98 NT 2000)Windows"能匹配"2000Windows"中的"Windows"，但不能匹配"3.1Windows"中的"Windows"。
(?<!n)	与(?!n)类似，只是方向相反。例如： "(?<!95 98 NT 2000)Windows"能匹配"3.1Windows"中的"Windows"，但不能匹配"2000Windows"中的"Windows"。

【完】