

# JavaScript程序设计 (下)

2016.12.4

isszym sysu.edu.cn

# 事件绑定

- 概述

- ✓ 当点击页面元素时会发生鼠标事件(event)，当输入字符时会产生按键事件，JavaScript可以捕捉到这些事件。
- ✓ 如果把事件和一段代码绑定(bind)在一起，可以实现在事件被触发时执行该段代码。与事件绑定的代码称为事件处理程序。
- ✓ 一些功能：
  - 通过捕捉事件，可以实现Ajax技术。
  - 可以通过捕捉提交事件进行合法性验证。
  - HTML5很多新增功能都要利用事件捕捉功能。

# ● 事件绑定方法

## 方法1：采用元素属性

```
<!DOCTYPE html>
<html>
<head>
  <title>Event binding</title>
  <script type = "text/javascript">
    function showMessage(){
      alert("Hello wolrd!");
    }
  </script>
</head>
<body>
  <input type="button" value="Click Me" onclick="alert(event.type)">
  <input type="button" value="Click Me" onclick="alert(this.value)">
  <input type="button" value="Hello" onclick="showMessage();">
  <input type="button" value="Click Me"
    onclick = "try {showMessage();}catch(ex){}">
</body>
</html>
```

Click

\* `event.type` 取到事件类型 Click, `this` 为被点击的对象

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
  function changetext(obj){
    obj.innerHTML="谢谢!";
  }
</script>
</head>
<body>
<div>
<p id="p1">请点击该文本</p>
<p id="p2">this is another paragraph</p>
</div>
<script type="text/javascript">
  var p1=document.getElementById("p1");
  var f1=function(){changetext(this)};
  p1.onclick= f1;
  p2.onclick= f1;
</script>
</body>
</html>
```

该语句要放在元素p1  
的后面，因为函数外  
的语句会顺序会执行。

## 方法2：采用addEventListener

```
var p1=document.getElementById("p1"); /* 执行要已定义元素p1 */  
var f1=function(){alert("hello");};  
p1.addEventListener("click",f1,false);/* false表示事件在冒泡阶段触发*/
```

addEventListener的参数：

第一个参数为要捕捉的事件

第二个参数为捕捉到事件后执行的函数

第三个参数false表示事件在冒泡阶段触发，为true则在捕捉阶段触发。

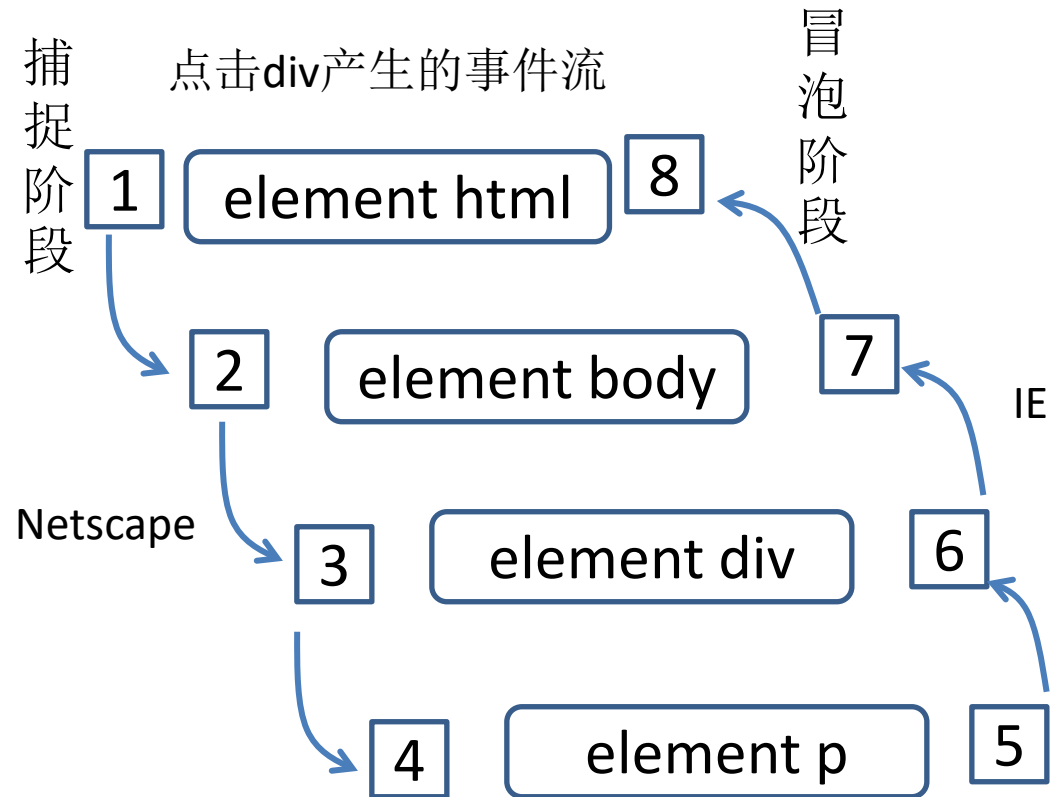
可以同时绑定多个事件处理程序：

```
p1.addEventListener("click",function(){alert("Hello!");});  
p1.addEventListener("click",function(event){  
    alert(event.type+": "  
    +event.clientX+", "+event.clientY  
    +event.screenX+", "+event.screenY);  
});
```

# 事件的捕捉阶段和冒泡阶段

- 概念

```
<html>
<head>
  <title>events</title>
</head>
<body id="body">
  <h1>addListener</h1>
  <div id="div">
    <p id="p1">
      This is a paragraph.
    </p>
  </div>
</body>
</html>
```



除了采用**addEventListener**绑定事件可以用于定义捕捉阶段和冒泡阶段触发事件，其它绑定事件的方法只在冒泡阶段触发事件。

```

<!DOCTYPE html>
<html id="html">
<head>
<meta charset="UTF-8">
<title>events</title>
</head>
<h1>addListener</h1>
<body id="body">
  <div id="div">
    <p id="p1">This is first paragraph.</p>
    <p id="p2">This is second paragraph.</p>
  </div>
</body>
<script>
  var p1 = document.getElementById("p1");
  var div = document.getElementById("div");
  var body = document.getElementById("body");
  var html = document.getElementById("html");
  p1.addEventListener("click", function() {alert("p1");}, false);
  body.addEventListener("click", function() {alert("body");}, false);
  div.addEventListener("click", function() {alert("div");}, false);
  html.addEventListener("click", function() {alert("html");}, false);
  html.addEventListener("click", function() {alert("html");}, true);
  body.addEventListener("click", function() {alert("body");}, true);
  div.addEventListener("click", function() {alert("div");}, true);
  p1.addEventListener("click", function() {alert("p1");}, true);
</script>
</html>

```

冒泡  
阶段

捕捉  
阶段

- 删除事件处理程序

```
btn.onclick = null;  
btn.removeEventListener("click",  
    function() {alert("Hello!");},  
    false);    //函数要完全一样才有效，否则，删除无效。
```

```
var handler = function(){alert(this.ad)};  
btn.addEventListener("click",handler, false);  
btn.removeEventListener("click",handler, false);  
    // 第三个参数与定义时也要一样(冒泡还是捕捉)
```

- 阻止事件传播

//阻止该事件传播

```
var btn = document.getElementById("myBtn"); // myBtn为一个a元素的id
```

```
btn.onclick = function(event){  
    event.stopPropagation();  
};
```

// 取消链接的默认行为，即不会导航到url。

```
var link = document.getElementById("myLink");  
link.onclick = function(event){  
    event.preventDefault();  
};
```



- 用程序产生事件

```
var p1=document.getElementById("p1");
var f1=function(){alert("hello");};
p1.addEventListener("click",f1,false);/* false表示事件在冒泡阶段触发*/
var ev = new MouseEvent('click', {
    cancelable: true,
    bubble: true,
    view: window });
p1.dispatchEvent(ev);
```

\* 早期版本的IE的事件见附录。

# 事件对象(event)

- 概述

[DOM事件](#)

- 在触发某个事件时，会产生一个事件对象event，这个对象包含所有与事件相关的信息。IE8.0及更早版本的事件对象为window.event。

```
<input type="button" value="Click Me" onclick="alert(event.type)">
```

- Javascript有六类事件：用户界面事件、焦点事件、鼠标事件、键盘事件、触摸屏事件、手势事件。

- 用户界面事件主要包括某个页面或图像加载完成事件load、加载中断事件abort、加载出错事件error、退出页面事件unload、页面文本被选中事件select和窗口或框架尺寸改变事件resize。

```
window.onload = function(){alert("Hello!")}  
<body onload = "alert('Hello!')">  
<img onload = "alert('Hello!')">
```

\* 如果希望html文件装载后立即发生则要使用document的事件DOMContentLoaded。

- 焦点事件主要包括获得焦点事件focus、失去焦点事件blur、通过冒泡方式得到焦点focusin和通过冒泡方式失去焦点focusout。

```
<input type= "text" onblur= "alert('blur')" onfocus="alert('focus')" >
```

- 鼠标事件主要包括单击click、双击dblclick、按下鼠标mousedown、放开鼠标mouseup、进入元素mouseenter、离开元素mouseout、在元素上方移动mousemove、悬浮在元素上方mouseover。

```
<div onclick="alert('click')"> hello </div>
```

- 鼠标可以相对于当前元素(event.offsetX, event.offsetY)、上级元素(event.x, event.y)、当前页面(event.pageX, event.pageY)、浏览器窗口(event.screenX, event.screenY)和客户区event.clientX, event.clientY进行定位。上级元素默认为BODY。
- 可以用event.button的取值判断哪些鼠标键被按下：0没按键,1按左键,2按右键,3按左右键,4按中间键,5按左键和中间键,6按右键和中间键,7按所有的键。
- 可以取到鼠标事件产生的元素：鼠标移动离开的对象event.fromElement、鼠标移动进入的对象event.toElement、触发事件(鼠标点击)的元素event.srcElement、冒泡事件中的最小范围元素event.target、当前事件的元素(同this) event.currentTarget。采用event.keyCode可以取到按键内码(扫描码), 用event.charCode取到按键字符(可显示), 用event.altKey、event.ctrlKey和event.shiftKey判断alt、ctrl或shift的键是否按下, 按下为true。
- 触摸屏事件的touchmove可以判断手指是否移动, 位置的获取与鼠标一样。姿势事件gesturestart可以用于判断是否两个手指同时触摸, 用event.scale取到两个手指之间的距离。具体的内容可以参见附录。

```

<!DOCTYPE html>
<html>
<body>
<p id="myBtn">第一段</p><p>第二段</p><p>第三段</p>
<script>
var btn = document.getElementById("myBtn");
var handler = function (event) {
    switch (event.type) {
        case "click": alert("Clicked"); break; //点击了myBtn之后显示Clicked
        case "mouseover":
            event.target.style.color = "red"; break; //这里的event.target就是this
        case "mouseout": event.target.style.color = "";
    }
};
btn.onclick = handler;
btn.onmouseover = handler;
btn.onmouseout = handler;
document.body.onclick = function (event) {
    alert(event.currentTarget === document.body); //点击了myBtn之后 //true
    alert(this === document.body); //true
    alert(event.target === event.currentTarget); //false
    alert(event.target === document.getElementById("myBtn")); //true
};
</script>
</body>
</html>

```

- event.currentTarget为本事件的实际定义者（上例是body）
- event.target为本事件实际作用的元素，（上例是myBtn）

## • 通过事件提交表单

```
<%@ page import="java.util.*" contentType="text/html; charset=utf-8"%>
<% request.setCharacterEncoding("utf-8");%>
<!DOCTYPE html><html><head>
<script type="text/javascript">
    function submit1() {
        var p1 = document.getElementById("p1");
        if (p1.value == "") { alert("姓名不能为空!"); return false;}
        document.forms["frm"].submit();           // 用document.frm.submit()也可以
        return false;                               // 返回false取消默认行为 (submit)
    }
</script>
</head><body>
    <form name="frm" method="post" action="post.jsp">
        <input id="p1" name="p1" type="text" value="Hello!" />
        <input id="p2" type="submit" value="showHTML" />
    </form>
    <script type="text/javascript">
        p2 = document.getElementById("p2");
        p2.addEventListener("click", submit1); // 也可以用"function(){submit();}"
    </script>
</body>
</html>
```

post.jsp

```
<%@ page language="java"
    import="java.util.*"
    contentType="text/html;
    charset=utf-8"%>
<% request.setCharacterEncoding("utf-8");%>
<%
    String p1=request.getParameter("p1");
    out.print("Get:"+p1);
%>
```

另外两种提交的定义方法:

```
<form onsubmit = "return submit1()" ...>
```

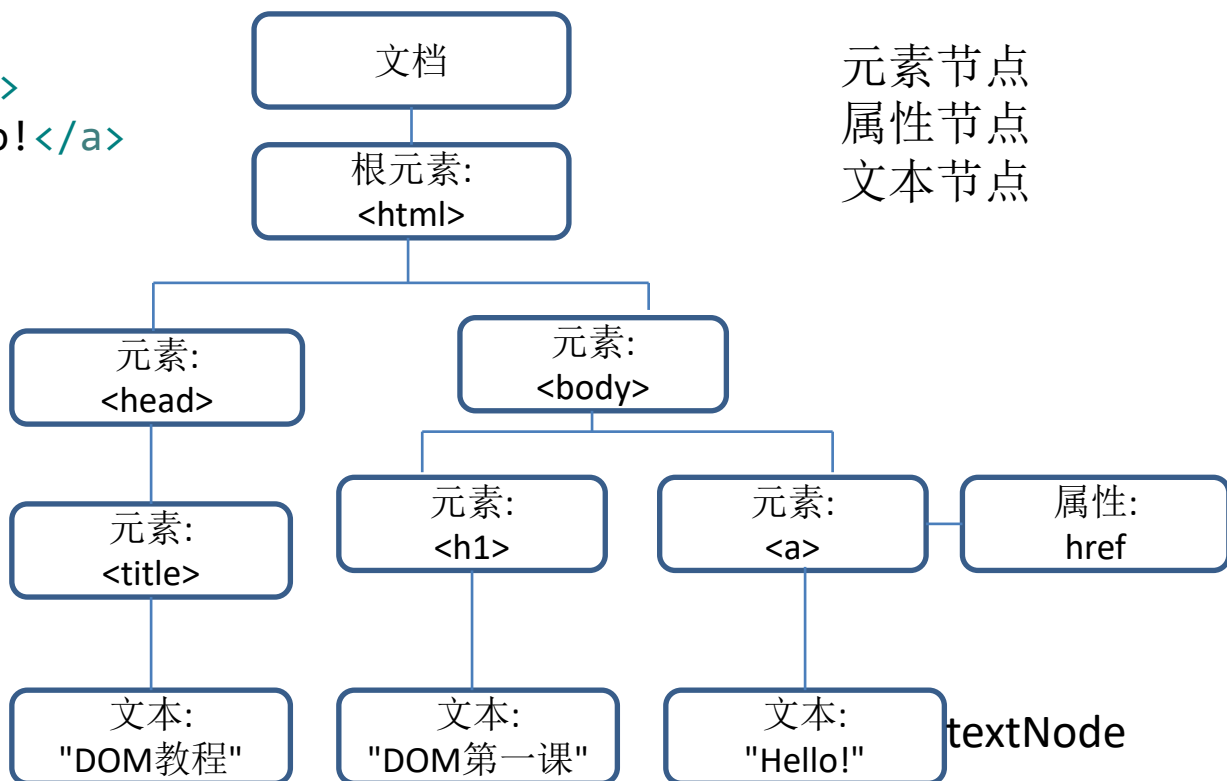
```
<input type="submit" onclick = "return submit1()" ...>
```

# 文档对象模型

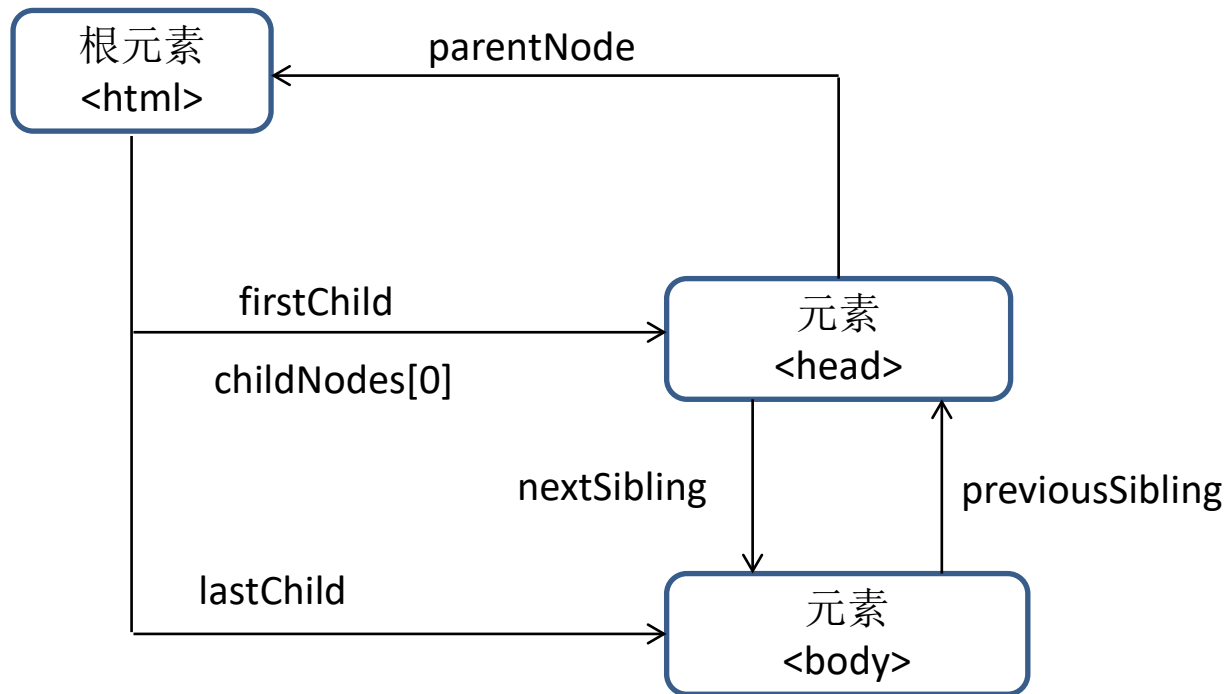
(Document Object Model, DOM)

- 文档树

```
<html>
  <head>
    <title>DOM 教程</title>
  </head>
  <body>
    <h1>DOM 第一课</h1>
    <a href="#" >Hello!</a>
  </body>
</html>
```



\* 每个节点都是一个对象



```
<div>  
  <span id="t1">  
    My text  
  </span>  
</div>
```

```
var s = document.getElementById("t1");  
alert(s.parentNode.nodeName);
```

```
//div
```

## • 节点的属性

:

nodeName	返回值：标签名(元素节点)或属性名(属性节点)或#text(文本节点)或#document(文档节点)
nodeType	取到节点类型：1-元素 2-属性 3-文本 8-注释 9-文档
nodeValue	取到节点值：文本(文本节点)，属性包含属性值(属性节点)，null(文档节点和元素节点)
innerHTML	取到节点（元素）的内容。 outerHTML，取到本元素(包含内容)。
	innerText 类似innerHTML，只是当成文本返回，只有IE支持。
	outerText 类似outerHTML，只是当成文本返回，只有IE支持。
childNodes	会返回所有子节点（数组）。children类似，只是不会返回TextNode。
attributes	取到节点（元素）的所有属性
className	取得节点属性class的值。
classList	获得节点的所有类名。可以用以下方法修改它： add(增加类)，remove（删除类） toggle（切换类），contains（是否包含一个类）(HTML 5支持)
data-	自定义属性名（HTML5支持）

\* 当元素内容为空格时，IE10之前的版本不会返回文本节点。为了统一所有浏览器的行为，html5为节点重新定义了一套遍历节点的属性：

firstElementChild, lastElementChild, nextElementSibling, previousElementSibling, childElementCount。



## • 直接获得元素的方法

除了通过从一个元素遍历可以取到所有元素（实际上还可以取到属性节点和文本节点），还可以通过**document**的一些方法直接得到所需的元素：

<code>getElementById()</code>	得到具有某个id值的元素。
<code>getElementsByTagName()</code>	返回包含带有指定标签名称的所有元素（数组）。
<code>getElementsByClassName()</code>	返回包含带有指定类名(可以是空格隔开的多个类名)的所有元素（数组）。只有html5支持
<code>querySelectorAll()</code>	取得与CSS选择器匹配的所有元素（数组）。只限html5
<code>querySelector()</code>	取得与CSS选择器匹配的第一个节点。只限html5

```
var s = document.querySelectorAll("a"); //返回文档中所有的a元素(数组)
document.write(s.length);
```

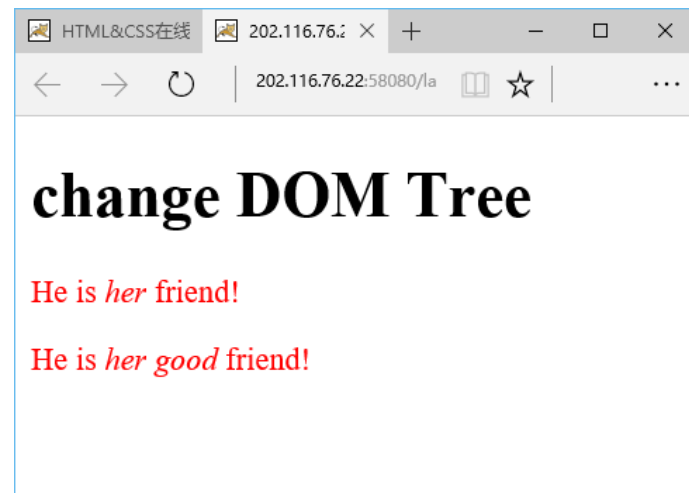
## • 修改文档树的方法

通过增删改子节点等方法可以直接改变文档树的结构，还可以修改节点属性（例如，`nodeValue`，`innerHTML`和`outerHTML`）或者通过修改元素属性和样式改变文档树。

### 例子1

```
<!DOCTYPE html><html><head>
<script type="text/javascript">
...//见下页
</script>
</head><body>
<h1>change DOM Tree</h1>
<div id="test"> </div>
</body></html>
```

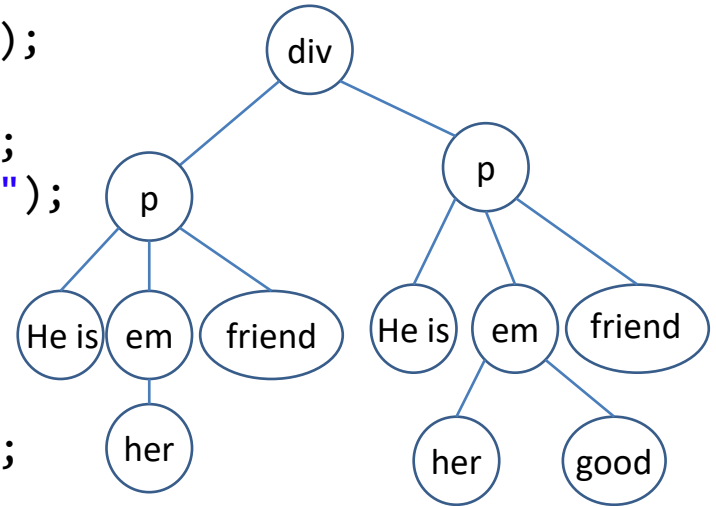
```
<div id="test">
  <p id="para1" style="color:red">He is <em> her </em> friend! </p>
  <p style="color:red">He is <em> her good</em> friend! </p>
</div>
```



```

window.onload= function(){
    var para1 = document.createElement("p");
    var txt1 = document.createTextNode("He is ");
    var em1 = document.createElement("em");
    var txt2 = document.createTextNode(" her ");
    var txt3 = document.createTextNode("friend!");
    para1.appendChild(txt1);
    em1.appendChild(txt2);
    para1.appendChild(txt3);
    para1.insertBefore(em1,txt3);
    var style=document.createAttribute("style");
    style.value="color:red";
    para1.setAttributeNode(style);
    var testdiv = document.getElementById("test");
    testdiv.appendChild(para1);
    var para2 = para1.cloneNode(true); // 参数: 是否克隆属性
    testdiv.insertBefore(para2,null);
    var txt4 = document.createTextNode(" ");
    para2.childNodes[1].appendChild(txt4);
    txt4.nodeValue = " good ";
    alert(para2.childNodes[1].childNodes[1].nodeValue);
    alert(testdiv.outerHTML);
}

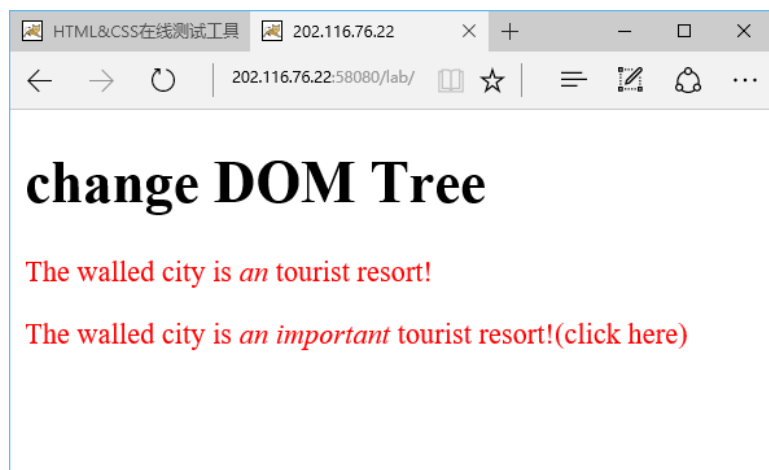
```



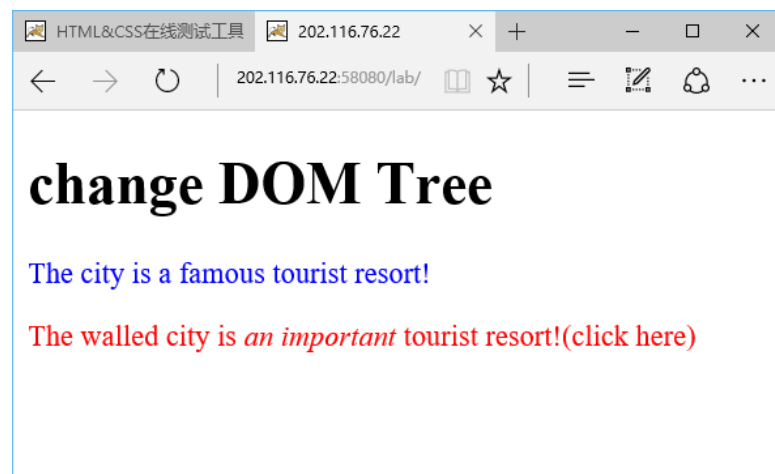
// 如果不采用window.load事件，而是直接执行，则需要把这段  
 // Javascript代码放在body元素的最后面，即插入到在</body>前面。

## 例子2

```
<!DOCTYPE html><html><head>
<script type="text/javascript">
...
</script>
</head><body>
<h1>change DOM Tree</h1>
<div id="test">
  <p id="para1" style="color:red">
    The walled city is <em>an</em>tourist resort!
  </p>
  <p id="para2" style="color:red">
    The walled city is <em>an important</em>tourist resort!(click here)
  </p>
</div>
</body></html>
```



点击第  
二行后  
→



```

function insertAfter(newElement,targetElement){
    var parent = targetElement.parentNode;
    if(parent.lastChild==targetElement){
        parent.appendChild(newElement);
    }else{
        parent.insertBefore(newElement, targetElement.nextSibling);
    }
}

function removeNode(){
    var para1 = document.getElementById("para1");
    para1.removeChild(para1.childNodes[1]);
    para1.style.color = "blue";
    var txt1 = document.createTextNode(" a famous ");
    insertAfter(txt1,para1.childNodes[0]);
    var txt2 = document.createTextNode("The city is ");
    para1.replaceChild(txt2,para1.childNodes[0]);
}

window.onload= function(){
    var testdiv = document.getElementById("test");
    var para2 = document.getElementById("para2");

    testdiv.childNodes[3].setAttribute("onclick","removeNode()");
    //para2.onclick=function(){removeNode()}; //功能同上
    alert(testdiv.outerHTML);
}

```

因为HTML的元素很多都是对象，所以除了采用DOM节点操作函数还可以采用这些对象的特定方法修改DOM树：

```
<html><head><script type="text/javascript">
function insRow()
{
    var x=document.getElementById('myTable').insertRow(0)
    var y=x.insertCell(0)
    var z=x.insertCell(1)
    y.innerHTML="NEW CELL1"
    z.innerHTML="NEW CELL2"
}
</script>
</head>
<body>
<table id="myTable" border="1">
<tr><td>Row1 cell1</td><td>Row1 cell2</td></tr>
<tr><td>Row2 cell1</td><td>Row2 cell2</td></tr>
<tr><td>Row3 cell1</td><td>Row3 cell2</td></tr>
</table>
<br />
<input type="button" onclick="insRow()" value="insert row">
</body></html>
```

\* 用Image img = new Image() 可以生成<img>元素

\* DOM对象见附录

## • 修改元素属性

通过属性节点的nodeValue可以获得或修改属性值，通过元素的方法getAttribute(name)和setAttribute(name,value)可以获得或设置属性值，还可以把属性名作为元素的数据域获得或修改属性值。

```
<div id="div1" class="hd">
  <img id="logo" />
  <a id="a1" href="http://www.sysu.edu.cn">中大</a>
</div>
var div = document.getElementById("div1");
var img = document.getElementById("logo");
var a = document.getElementById("a1");
//取得元素特性
alert(div.id);
alert(div.getAttribute("id"));
alert(div.attributes["id"].nodeValue);
alert(div.className);           //不用div.class是因为class是保留关键字
alert(div.getAttribute("class")); //注意这里是class而不是className
```

//设置元素特性

```
div.id = "div2";  
div.setAttribute("id", "div2");  
div.attributes["id"].nodeValue = "div2";  
img.src = "images/img1.gif";  
  
div.className = "ft";  
div.setAttribute("class", "ft");
```

//移除元素特性

```
div.removeAttribute("class");  
div.attributes.removeNamedItem("class");
```

```
a.href = "http://www.sina.com.cn";  
div.title = "header";  
a.innerHTML = "华工";
```

```
//重新设置超链接  
//title改为"header"  
// "中大"改为"华工"
```



## • 修改元素样式

如果想修改元素的样式，可以通过修改样式文件名、修改style属性值、修改原子样式的值和修改类名来实现。文档中的每个样式表定义(外部css文件和style元素)都可以取到和修改。

```
<p id="hello" class="c1">Hello World!</p>
<link rel = "stylesheet" type="text/css" id="css" href="firefox.css" />
<!--改变样式文件名-->
<span onclick="javascript:document.getElementById('css').href = 'ie.css'">
    点我改变样式
</span>
<script>
    document.getElementById("hello").style.color="blue"; //修改原子样式
    document.getElementById("hello").className="c2"; //改变类属性值
    var p1 = document.getElementById('hello');
    p1.setAttribute("style", "color:blue"); //style属性
    p1.setAttribute("class", "c2 c3"); //改变类属性值
    // 设置样式表（外部css文件、style元素）的全部内容
    document.styleSheets[0].cssText =
        document.styleSheets[0].cssText.replace(/red/g, "yellow");
    // p1.style.cssText = "color:blue;width:800px";
</script>
```

# AJAX技术

- 概述

传统的网页如果需要更新内容，必需重载整个网页。使用AJAX (Asynchronous Javascript And XML) 技术除了可以利用从 Web 服务器上获取的信息局部更新网页，还可以实现在保持当前网页的情况下提交数据给Web服务器。

2005年2月，Adaptive Path公司的Jesse James Garrett在文章“Ajax: A New Approach to Web Applications”中最早提出这个概念。这篇文章认为将XHTML、CSS、JavaScript、DOM和XMLHttpRequest混合使用来开发Web应用将会成为一种新的趋势。事实上，在Ajax这个概念出现之前就已经有了丰富的Ajax应用，例如，Google Maps和Google Suggest就是应用了XMLHttpRequest异步从服务器端来获取数据，实现了客户端无刷新的效果。

所有现代浏览器均支持 XMLHttpRequest 对象（IE5 和 IE6 使用 ActiveXObject）。

```
var xmlhttp;  
if (window.XMLHttpRequest) {  
    xmlhttp=new XMLHttpRequest();  
}else {  
    // code for IE6, IE5  
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");  
}
```

[参考](#)

- 用iframe实现AJAX技术

更早期用到的异步技术是采用一个隐藏的iframe实现的。通过改变iframe的src来生成http请求，当收到http响应时会触发onload事件，此时可以取到http响应的正文。

```
<!DOCTYPE html><html><head>                                javascript10.jsp
<script type="text/javascript">
  function LoadContent(src, iframeID) {
    var content = document.getElementById(iframeID);
    content.setAttribute('src', src);
  }
  function swipContent(src, to) {
    var bobo=document.getElementById(src);
    var content = bobo.contentDocument?bobo.contentDocument.body.innerHTML
      :bobo.Document.body.innerHTML;
    document.getElementById(to).innerHTML = content;
  }
</script></head>
<body>
  <a href="#" onclick="LoadContent('ajaxGet.jsp','icontent')">Link Button</a>
  <iframe id="icontent" width="600" height="0" scrolling="auto"
    frameborder="0" onload="swipContent('icontent','dest')"></iframe>
    <div id="dest" />
</body>
</html>
```

## • AJAX技术-GET

```
<html><head>
<script type="text/javascript">
    function get() {
        var xmlhttp = new XMLHttpRequest();           // 创建http请求
        xmlhttp.onreadystatechange = function () {    // 当http请求的状态变化时执行
            if (xmlhttp.readyState == 4) {           // 4-已收到http响应数据
                if (xmlhttp.status >= 200 && xmlhttp.status < 300
                    || xmlhttp.status == 304) {        // 200~299-OK 304-unmodified
                    alert(xmlhttp.responseText);      // http响应的正文
                    var oTest = document.getElementById("p4");
                    oTest.innerHTML = xmlhttp.responseText;
                } else {
                    alert("error");
                }
            }
        };
        // 打开http请求（open）的参数：get或post, url, 是否异步发送
        xmlhttp.open("get", "ajaxGet.jsp?p1=1&p2=5", true);
        xmlhttp.send(null);                          //发送http请求。get只能用null作为参数
    }
</script></head>
```

javascript7.jsp

```

<body>
  <form name="p3">
    <p id="p4">hello world!</p>
    <input id="p1" type="text" value="Hello!" />
    <input id="p2" type="button" value="showHTML" />
  </form>
  <script type="text/javascript">
    p2 = document.getElementById("p2");
    p2.addEventListener("click", get);
  </script>
</body>
</html>

```

## ajaxGet.jsp

```

<h1>DOM 1st Class </h1>
<p>Hello world!</p>
<% request.setCharacterEncoding("utf-8");%>
<% String p1=request.getParameter("p1");
    String p2=request.getParameter("p2");
    out.print("Get:"+p1+" "+p2);
%>

```

## • AJAX技术-POST

```
<script type="text/javascript">                                     javasrcipt8.jsp
    function post() {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function () {
            if (xmlhttp.readyState == 4) {
                if (xmlhttp.status >= 200 && xmlhttp.status < 300
                    || xmlhttp.status >= 304) {
                    alert(xmlhttp.responseText);
                    var oTest = document.getElementById("p4");
                    oTest.innerHTML = xmlhttp.responseText;
                } else {
                    alert("error");
                }
            }
        };
    };
    var v1 = document.getElementById("p1").value;
    var param = "p1=" + encodeURIComponent(v1) + "&p2=345"; //汉字需要编码
    xmlhttp.open("post", "ajaxPost.jsp", true);
    xmlhttp.setRequestHeader("Content-Type",
                            "application/x-www-form-urlencoded");
    xmlhttp.send(param);      // 没有参数就用null。
}
</script>
```

```

<form name="p3">
    <p id="p4">hello world!</p>
    <input id="p1" type="text" value="Hello!" />
    <input id="p2" type="button" value="showHTML" />
</form>
<script type="text/javascript">
    p2 = document.getElementById("p2");
    p2.addEventListener("click", post);
</script>
</body>
</html>

```

## ajaxPost.jsp

```

<%@ page language="java" import="java.util.*"
        contentType="text/html; charset=utf-8"%>
<% request.setCharacterEncoding("utf-8");%>
<% String p1=request.getParameter("p1");
    String p2=request.getParameter("p2");
    out.print("Get:"+p1+" "+p2);
%>

```

## • XMLHttpRequest对象

### xmlhttp的属性:

**responseText:** 作为响应主体被返回的文本  
**responseXML:** 如果响应的内容类型是"text/xml"或"application/xml",这里保存着响应数据。  
**status:** 响应的HTTP状态码  
**statusText:** HTTP状态说明。  
**readyState:** 请求或响应过程的当前活动阶段。

### xmlhttp.readyState

- 0: 未初始化。尚未调用open
- 1: 启动。已调用open, 未调用send
- 2: 发送。已调用send
- 3: 接收。已接收到部分响应数据
- 4: 完成。已接收到所有响应数据

### xmlhttp.status

- 200~299: 表示成功接收请求, 并已经完成处理。
- 304: 请求的资源没有更改, 可以直接使用缓存数据。可以取到responseText。
- 其它: 有错或未完成请求。

### 设置http头部信息

```
xmlhttp.setRequestHeader("Content-Type",  
    "application/x-www-form-urlencoded");
```

### 设定超时时间

```
xmlhttp.timeout=1000; //ms  
xmlhttp.ontimeout=function(){  
    alert("no reponse received in a second.")  
};
```



## • 串行化

如果采用AJAX技术实现post功能，就要对输入数据进行串行化。串行化就是把form中每个输入字段的值全部取到并形成如下key-value格式的数据：

$x1=v1\&x2=v2\&x3=v3\ldots$

如果x和v使用了汉字，则要使用函数encodeURIComponent()进行编码。采用ajax技术就可以把这个串行化的数据提交给后台处理。

下面是一个对表单(form)的串行化函数：

```
function serialize(form){
    var parts = [], field = null,i,len,j,optLen,option,optValue;
    for (i=0, len=form.elements.length; i < len; i++){
        field = form.elements[i];
        switch(field.type){
            ... // 见下页
        }//switch
    }//for
    return parts.join("&");
}//function
```

```
switch(field.type){
  case "select-one":
  case "select-multiple":
    if (field.name.length){
      for (j=0, optLen = field.options.length; j < optLen; j++){
        option = field.options[j];
        if (option.selected){
          optValue = "";
          if (option.hasAttribute){
            optValue = (option.hasAttribute("value")?
              option.value: option.text);
          }else {
            optValue = (option.attributes["value"].specified?
              option.value:option.text);
          } //if
          parts.push(encodeURIComponent(field.name) + "=" +
            encodeURIComponent(optValue));
        } //if
      } //for
    } //if
    break;
```

```

case undefined:           //字段集
case "file":              //文件输入
case "submit":            //提交按钮
case "reset":             //重置按钮
case "button":            //自定义按钮
    break;
case "radio":              //单选按钮
case "checkbox":             //复选框
    if (!field.checked){ break;}
/* 执行默认操作 */
default:                  //不包含没有名字的表单字段
    if (field.name.length){
        parts.push(encodeURIComponent(field.name)
                    + "=" + encodeURIComponent(field.value));
    }
}
} //switch

```


# 定时器

语句`setTimeout(f, t)`用于在`t`毫秒之后执行一次函数`f`。如果每个`t`毫秒都要执行一次`f`，可以在函数`f`中执行`setTimeout(f, t)`。如果`setTimeout(f, t)`返回的值为`id`，`id`可以用于停止定时器的执行`clearTimeout(id)`。也可以使用`setInterval(f, t)`实现周期性执行函数`f`的功能，利用其返回值`id`也可以停止该定时器`clearInterval(id)`。

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
    var i = 0;
    function hello() {
        i++;
        var ctrl = document.getElementById("timer");
        ctrl.innerHTML = i;
        window.setTimeout(hello, 1000);
    }
    var id=setTimeout(hello, 1000);
</script>
</head>
<body>
    timer:<span id="timer">0</span>
</body>
</html>
```

timer.html

\* 使用`setInterval(hello, 1000)`就可能周期时间内没有执行完毕而出现异常，采用`setTimeout`实现周期执行功能就不会出现这种情况。



`setInterval()` - 间隔指定的毫秒数不停地执行指定的代码。  
`setTimeout()` - 暂停指定的毫秒数后执行指定的代码

# Javascript对象

## （Browse Object Model， BOM）

- 概述

javascript可以通过一组对象获得浏览器属性或操作浏览器：

[参考](#)

- 通过window对象新建或访问当前浏览器窗口
- 通过document 对象访问每个载入浏览器的 HTML 文档（网页）
- 通过screen对象获得显示屏的特性
- 通过location对象得到和设置浏览器地址栏的内容
- 通过history对象操作浏览历史
- 通过navigator 对象获得浏览器本身的信息。
- 通过global对象可以使用全局变量和全局函数

Javascript的全局变量和函数都属于window对象。document、screen、location、history、navigator都属于window对象。对这些变量、函数和对象的访问可以省略window，例如，window.document可以写为document。

- window对象

window对象以用来确定浏览器窗口客户区大小，详细做法见附录。执行 `window.open("http://www.w3school.com.cn")` 可以打开一个窗口，还可以用参数来控制打开窗口的外观，例如，

```
window.open("http://www.w3school.com.cn", "_blank",  
            "toolbar=yes, location=yes, directories=no, status=no,  
            menubar=yes, scrollbars=yes, resizable=no,  
            copyhistory=yes, width=400, height=400")
```

- `_blank`表示在新窗口打开网页，`_self`表示在当前窗口打开网页，`name`表示在名称为`name`的窗口打开网页。
- `toolbar=yes, location=yes` 表示浏览器显示工具栏、地址栏
- `menubar=yes , scrollbars=yes`表示浏览器显示菜单和滚动条
- `directories=no, status=no, resizable=no`表示不显示收藏夹、状态栏，不可变化大小

**window**和**body**的**load**的事件是在整个页面的所有对象装载完毕后发生，如果希望**html**文件装载后立即发生则要使用**document**的事件**DOMContentLoaded**。

## • location对象

通过location.href可以取到当前浏览器地址栏中的地址(URL)，通过location.hostname和location.port还可以取到URL中的主机名和端口号。

## • document对象

document对象可以取到当前网页的内容。document.anchors[]、document.forms[]和document.images[]可以得到当前网页的所有a元素、所有form和所有img元素（均为对象数组）。document.referrer可以得到载入当前网页的网页的URL，即从哪个网页(URL)点击进入当前网页的。document.cookie可以取到浏览器的cookie。document.write(...)可以直接往网页中输出一个字符串。事件DOMContentLoaded只要页面加载完毕就发生，而事件window.onload需要等待所有页面对象加载完毕才发生。采用document的方法可以获得元素，见前面“获取元素的方法”。

```
var s = document.querySelectorAll("a");  
document.write(s.length);
```

```

document.ready = function (callback) {    //callback为要执行的函数
    if (document.addEventListener) {        //兼容FF,Google
        document.addEventListener('DOMContentLoaded', function () {
            document.removeEventListener('DOMContentLoaded',
                                           arguments.callee, false);

            callback();
        }, false)
    }
    else if (document.attachEvent) {        //兼容IE
        document.attachEvent('onreadystatechange', function () {
            if (document.readyState == "complete") {
                document.detachEvent("onreadystatechange",
                                       arguments.callee);

                callback();
            }
        })
    }
    else if (document.lastChild == document.body) {
        callback();
    }
}

```

## • history对象

执行history.back()、history.forward()和history.go(-2)可以回退一页、前进一页和回退两页。



## • cookie对象

下面的网页在加载之后立即检查是否存在username的cookie，如果存在，则取出其值，如果不存在，则要求输入值然后创建该cookie。

```
<!DOCTYPE html><html><head>                                cookie.html
<script type="text/javascript">
    function getCookie(c_name) {
        if (document.cookie.length > 0) {
            c_start = document.cookie.indexOf(c_name + "=");
            if (c_start != -1) {
                c_start = c_start + c_name.length + 1;
                c_end = document.cookie.indexOf(";", c_start);
                if (c_end == -1)
                    c_end = document.cookie.length;
                return unescape(document.cookie.substring(c_start, c_end));
            }
        }
        return "";
    }
    function setCookie(c_name, value, expiredays) {
        var exdate = new Date();
        exdate.setDate(exdate.getDate() + expiredays);
        document.cookie = c_name + "=" + escape(value) +
            ((expiredays == null) ? "" : ";expires=" + exdate.toGMTString());
    }
}
```

```
function checkCookie() {  
    var username = getCookie('username');  
    if (username != null && username != "") {  
        alert('Welcome again ' + username + '!')  
    }  
    else {  
        username = prompt('Please enter your name:', '')  
        if (username != null && username != "") {  
            setCookie('username', username, 365)  
        }  
    }  
}  
</script>  
</head>  
<body onload="checkCookie()">  
</body>  
</html>
```

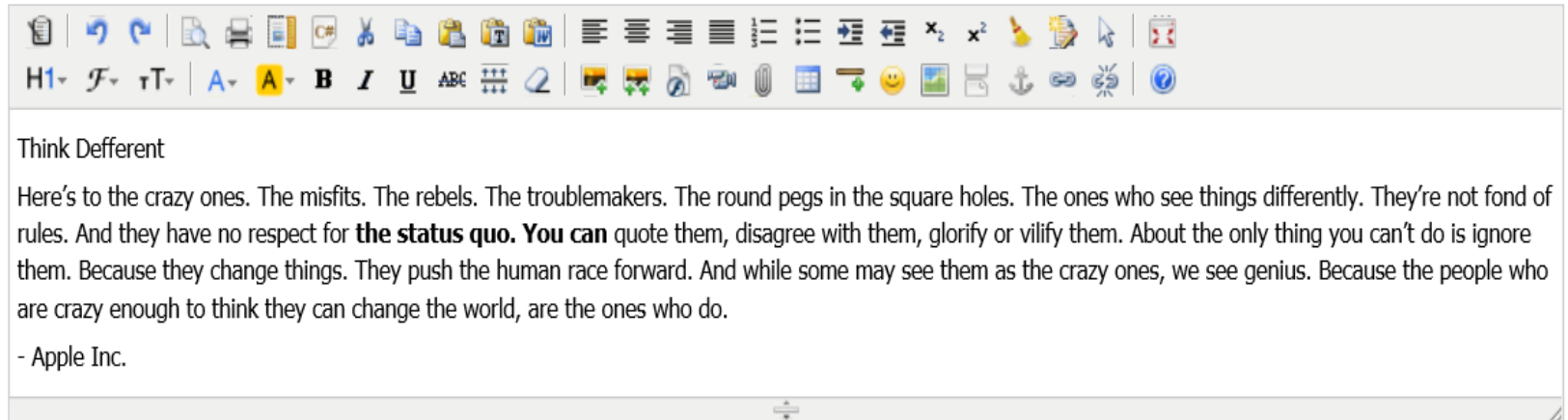
## • 其它对象

日期操作采用Date对象，数学函数采用Math对象，数值转换使用Number对象，RegExp对象用于操作正则表达式，文档中很多元素都属于DOM对象，具体内容见附录。

# 富文本编辑

`input`元素或者`textarea`元素只能输入文字，富文本编辑可以在输入框中直接设置文本颜色、字体大小、显示图片等。

有两种实现富文本编辑的方法，一种是通过`iframe`实现，另一种是通过`div`实现。



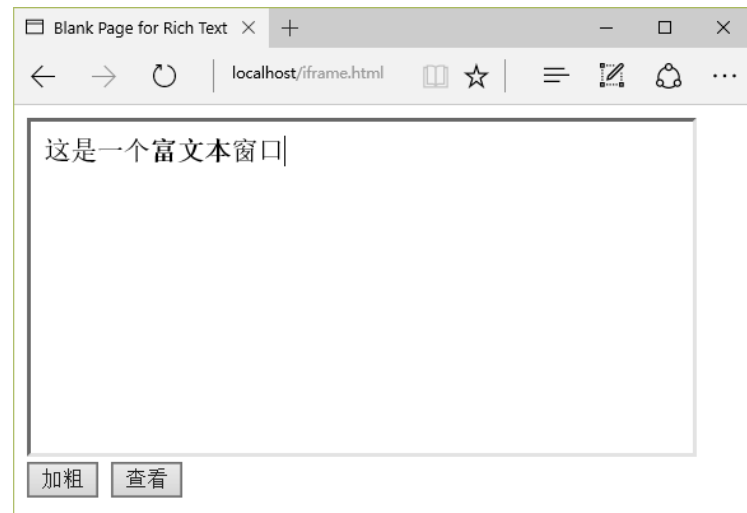
<http://kindeditor.net/demo.php>

```

<!DOCTYPE html>
<html><head> <title>Blank Page for Rich Text Editing</title></head>
<body>
  <iframe name="edit" style="height:200px;width:400px;">
  </iframe><br>
  <input type="button" value="加粗" onclick="bold()">
  <input type="button" value="代码" onclick="show()">
  <script>
    frames["edit"].document.designMode = "on";
    function bold() {
      frames["edit"].document.execCommand("bold", false, null);
    }
    function show() {
      var o = frames["edit"].document.body;
      alert(o.innerHTML);
    }
  </script>
</body>
</html>

```

选择一段文字，点击按钮“加粗”



```

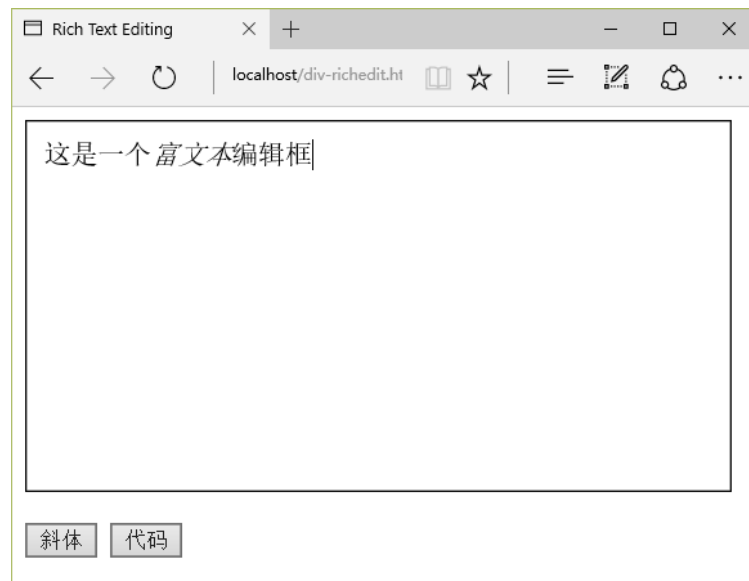
<!DOCTYPE html>
<html><head><title>Rich Text Editing</title></head>
<body>
  <div class="editable" id="richedit" contenteditable="true"
    style="padding:10px;width:400px;height:200px;border:solid 1px black">
  </div><br>
  <input type="button" value="斜体" onclick="italic()">
  <input type="button" value="代码" onclick="show()">
  <script>
    function italic() {
      document.execCommand("italic", false, null);
    }
    function show() {
      var o = document.getElementById("richedit");
      alert(o.innerHTML);
    }
  </script>
</body>
</html>

```

executeCommand的其他命令见附录。

获取选择的文档：

```
var selection = document.getSelection();
```



# 正则表达式

正则表达式可以用来在一个字符串中查找一个子串,，替换一个字符串等。

```
<html><head></head><body><script>
function prn(str) {
    document.write(str);
    document.write("<br>");
}
var sToMatch = "a bat, a Cat; a fAt caT"; // 要匹配的字符串
var reCat = new RegExp("at");           // 定义查找的子串at
prn(reCat.test(sToMatch));               // 返回true(找到子串).
prn(sToMatch.search(reCat));             // 返回3。如果未找到子串，则返回-1
reCat = new RegExp("at", "gi");          // 定义要查找或替换的子串：全部(g)，忽略大小写(i)
prn(sToMatch.replace(reCat, "ar"));      // 替换子串，返回a bar, a Car; a far car
prn(sToMatch.replace(reCat, function (sMatch) {
    if (sMatch == "At")                  // 子串At被替换为ast
        return "ast";
    else
        return "ar";                    // 其他子串，例如，at,aT，被替换为ar
}));                                     // 替换子串"At"和其它，返回a bar, a Car, a fast car
arrMatches = sToMatch.match(reCat);      // 匹配子串，返回一个数组at, at, At, aT
prn(arrMatches);
reCat = new RegExp(";|,");               // 用;或,划分子串
var arrWords = sToMatch.split(reCat);    // 返回一个数组: "a bat", "a Cat"和"a fAt caT"
prn(arrWords);
```

[参考](#)

```

// 除了用RegExp对象表示，正则表达式也可以用字面量表示：
prn(sToMatch.match(/cat/gi)); // 返回数组: "Cat","caT"
var reCat = /cat/gi; // 定义要查找或替换的子串: 全部(g), 忽略大小写(i)
prn(sToMatch.replace(reCat, "dog")); // a bat, a dog; a fAt dog
sToMatch = "ABCaACEFabCADDf";
var reCat = new RegExp("[A-Z]{3}", "g");
reCat.lastIndex = 0;
while ((result = reCat.exec(sToMatch)) != null) { //找出所有匹配
    var str = "str:" + result[0];
    var fIndex = reCat.firstIndex;
    var lIndex = reCat.lastIndex - result[0].length;
    str = str + " Index:" + lIndex; //str:ABC Index:0 str:ACE Index:4 str:ADD Index:11
    prn(str);
}
prn(sToMatch.replace("at", "ar")); // 返回a bar, a Cat; a fAt caT。 替换第一个匹配子串
//替换第一个匹配子串(大小写敏感)。如果未找到子串，则返回-1
prn(sToMatch.search("aT")); // 返回21。
// 返回一个数组，包含两个子串:"a bat, a Cat"和"a fAt caT"
var arrSen = sToMatch.split(";");
</script>
</body>
</html>

```

```

<!DOCTYPE HTML><html> <head>    <title>reg1.html</title> </head>
<body><script>
    function prn(str) {
        document.write(str);
        document.write("<br>");
    }
    var sToMatch = "aaaabbbbbbbbaaabbbaabbbb123aaaaabbbb123"; // 要匹配的字符串
    var re1 = /. *bbb/g;      //无限匹配, 尽可能往后匹配, .*包含尽可能多的字符
    var re2 = /. *?bbb/g;     //加?表示懒惰匹配, 每次匹配到1个子串则返回, g表示要反复进行
    //var re3 = /. *+bbb/g;   //无限匹配, 含义同re1, Javascript不可用
    prn(re1.test(sToMatch));   // 是否匹配
    arrMatches = sToMatch.match(re1); // 无匹配项返回null, 否则返回数组
    prn(arrMatches);          // 返回aaaabbbbbbbbaaabbbaabbbb123aaaaabbbb
    arrMatches = sToMatch.match(re2);
    prn(arrMatches);          //返回aaaabbbb,bbb,baaabbbaabbbb,b123aaaaabbbb
    sToMatch = "dasdasblackredOUOI";
    prn(sToMatch.match(/(red|black|green)/g)); //返回black,red
    sToMatch = "Important word is the last one.";
    prn(sToMatch.match(/^(.+?)\b/g)); //懒惰匹配,返回Important (找出第一个单词)
    prn(sToMatch.match(/(\w+)\.$/g)); //返回one. (找出最后一个单词)
    prn(sToMatch.match(/\b(\S+?)\b/g)); //返回Important,word,is,the,last,one (找出所有单词)
    prn(sToMatch.match(/(\w+)/g));      //返回结果同上
    sToMatch = "This is a string using badword1 and anotherbadword.";
    prn(sToMatch.replace(/badword1|anotherbadword/g, "*****")); //返回
    prn(sToMatch.replace(/badword1|anotherbadword/g,
    function (sMatch) { return sMatch.replace(/./g, "*"); })); //返回

```



```

sToMatch = "#1234567890";
prn(sToMatch.match(/#\d+/g));           //返回#1234567890 未捕捉
prn(RegExp.$1);                         //返回""
prn(sToMatch.match(/#(\d+)/g));         //返回#1234567890 捕捉
prn(RegExp.$1);                         //返回1234567890
sToMatch = "1234 5678";
prn(sToMatch.replace(/(\d{4}) (\d{4})/g, "$2 $1")); //返回5678 1234
prn(RegExp.$1);                         //返回1234
prn(RegExp.$2);                         //返回5678
sToMatch = "dogdog";
prn(sToMatch.match(/(dog)\1/));         //\1代表第一个反向引用。返回dogdog,dog
prn(RegExp.$1);                         //返回dog
prn(RegExp.$2);                         //返回""
sToMatch = "#1234567890";
prn(sToMatch.match(/#(?:\d+)/g));       //返回#1234567890 取消捕捉(?:)
prn(RegExp.$1);                         //返回""
sToMatch = "<b>this would be bold</b>";
prn(sToMatch.replace(/<(?:.|\s)*?>/g, "")); //返回this would be bold 取消捕捉,同/<?:.*?>/
prn("<br>==多行模式:/m==");
sToMatch = "First second\nthird fourth\nfifth sixth";
prn(sToMatch.match(/(\w+)$/g));         //返回sixth
prn(sToMatch.match(/(\w+)$/gm));        //返回每一行最后词: second,fourth,sixth
prn("<br>==前瞻==");
var sToMatch1 = "bedroom";
var sToMatch2 = "bedding";

```

```

prn(sToMatch1.match(/(\bed(=?room))/)); //前瞻。IE有错
prn(sToMatch2.match(/(\bed(=?room))/)); //前瞻。IE有错
prn(sToMatch1.match(/(\bed(?!room))/)); //后瞻。IE有错
prn(sToMatch2.match(/(\bed(?!room))/)); //后瞻。IE有错
prn(RegExp.$1);
prn("<br>==RegExp==");
sToMatch = "bbq is a short for barbecue";
var reB = /b/g;
prn(reB.global); // true. g
prn(reB.ignoreCase); // false. i
prn(reB.multiline); // false. m
prn(reB.source); // b
reB.exec(sToMatch);
prn(reB.lastIndex); // 1
reB.exec(sToMatch);
prn(reB.lastIndex); // 2
reB.exec(sToMatch);
prn(reB.lastIndex); // 20
reB.exec(sToMatch);
prn(reB.lastIndex); // 23
reB.exec(sToMatch);
prn(reB.lastIndex); // 0
re1 = /(s)hort/g;
re1.test(sToMatch);
prn(RegExp.input); // "bbq is a short for barbecue"要匹配的字符串 简写为RegExp["$ _"]
prn(RegExp.leftContext); // "bbq is a " 匹配子串左边的字串 简写为RegExp["$ `"]
prn(RegExp.rightContext); // " for barbecue" 匹配子串右边的字串 简写为RegExp["$ '"]
prn(RegExp.lastMatch); // "short" 最后匹配的字符串 简写为RegExp["$ &"]
prn(RegExp.lastParen); // "s" 最后匹配的分组 简写为RegExp["$ +"]
</script></body> </html>

```

## 规则定义:

[...] 位于括号之内的任意字符  
[^...] 不在括号之中的任意字符  
. 除了换行符之外的任意字符,等价于`[^\n]`  
\w 任何单字字符,等价于`[a-zA-Z0-9]`  
\W 任何非单字字符,等价于`[^a-zA-Z0-9]`  
\s 任何空白符,等价于`[\t\n\r\f\v]`  
\S 任何非空白符,等价于`[^\t\n\r\f\v]`  
\d 任何数字,等价于`[0-9]`  
\D 除了数字之外的任何字符,等价于`[^0-9]`  
[b] 一个退格直接量(特例)

{n, m} 匹配前一项至少n次,但是不能超过m次  
{n, } 匹配前一项n次,或者多次  
{n} 匹配前一项恰好n次  
? 匹配前一项0次或1次,等价于 {0, 1},即可选的.  
+ 匹配前一项1次或多次,等价于 {1,}  
\* 匹配前一项0次或多次,等价于 {0,}

| 选择.匹配的要么是该符号左边的子表达式,要么它右边的子表达式  
(...) 分组.将几个项目分为一个单元.这个单元可由 \*、+、? 和|等符号使用,而且还可以记住和这个组匹配的字符以供此后引用使用  
\n 和第n个分组所匹配的字符相匹配.分组是括号中的子表达式(可能是嵌套的).分组号是从左到右计数的左括号数  
^ 匹配的是字符的开头,在多行检索中,匹配的是一行的开头  
\$ 匹配的是字符的结尾,在多行检索中,匹配的是一行的结尾  
\b 匹配的是一个词语的边界.简而言之就是位于字符\w 和 \w之间的位置(注意:[\b]匹配的是退格符)  
\B 匹配的是非词语的边界的字符  
i 执行大小写不敏感的匹配  
g 执行一个全局的匹配,简而言之,就是找到所有的匹配,而不是在找到第一个之后就停止了

\* RegExp采用引号时的斜杠\要用双斜杠\\, 引号采用\", 例如RegExp("\"\\(\\w\\)\"", "g")找出所有的 "(字母数字串)" 模式。

\f	换页符
\n	换行符
\r	回车
\t	制表符
\v	垂直制表符
\/	一个 / 直接量
\\	一个 \ 直接量
\.	一个 . 直接量
\*	一个 * 直接量
\+	一个 + 直接量
\?	一个 ? 直接量
\	一个   直接量
\(	一个 ( 直接量
\)	一个 ) 直接量
\[	一个 [ 直接量
\]	一个 ] 直接量
\{	一个 { 直接量
\}	一个 } 直接量
\XXX	由十进制数 XXX 指定的ASCII码字符
\Xnn	由十六进制数 nn 指定的ASCII码字符
\cX	控制字符^X. 例如, \cl等价于
\t, \cJ	等价于 \n

**非贪婪重复**: 在匹配量词后加上?, 表示最小匹配, 即匹配尽可能少的字符。

**贪婪重复**只要符合匹配模式就会尽可能多地包含字符。

```
var content="abc37&*&defdasabc48*&n+_+def";
var reCat = new RegExp(/abc(.|\n)+?def/g);
var all = "", ind=0;
reCat.lastIndex = 0;
while ((result = reCat.exec(content)) != null) { //匹配两次
    var res = "" + result[0];
    var str = "str:" + res;
    var lIndex = reCat.lastIndex - res.length;
    str = str + " Index:" + lIndex;
    all = all + content.substring(ind,lIndex) + " " + res;
    document.write(str);
}
```

//去掉正则表达式的?只会匹配一次, res包含整个字符串

[replace](#)

# 元素大小与scroll

`clientHeight`, `clientWidth`: 元素的内容加上padding的大小。

`offsetHeight`, `offsetWidth`: 元素除了margin之外的大小（包含滚动条）。

`offsetLeft`, `offsetTop`: 元素的边框（包含滚动条）与父元素之间的距离。

`scrollHeight`, `scrollWidth`: 在滚动窗口内或者无滚动时元素内容的总高度或宽度。

`scrollLeft`, `scrollTop`: 因为滚动隐藏在内容区域左侧（或顶部）的像素数。设置它们会造成滚动而隐藏一部分内容。

元素`document.documentElement`和`document.body`(IE)的`scrollHeight`和`clientHeight`的最大者可以用来取得文档总高度。

`scrollIntoView()`: 所有html元素都有这个方法，如果参数为`true`（默认），则滚动元素让元素的顶部与窗口对齐，如果参数为`false`，则会尽量多地显示元素，超出时底部会对齐。

# 性能问题

由于闭包需要占用额外空间，而且使用了闭包的對象不能被銷毀，跨區域執行還會帶來性能損失，一般最好避免出現閉包。解決方法就是不使用外部引用，可以採用參數帶入所需要的引用。

JavaScript程序可以分成多個script元素，可以放在html可以放在任何地方。瀏覽器是從前往後執行這些片段，而且一般不會並行執行，也就是要等前面的片段執行完畢才會繼續處理後面的網頁，這會導致阻塞或減慢顯示網頁的速度。採用的方法是把片段放置在body元素的末尾。

# 附录

# 附录1、BOM对象

## location 对象

- location 对象用于获得当前页面的地址 (URL)，并把浏览器重定向到新的页面。通过location.href可以返回当前页面的url，设置它可以转到 相应页面。

### Location 对象属性

属性	描述
<a href="#">hash</a>	设置或返回从井号 (#) 开始的 URL（锚）。
<a href="#">host</a>	设置或返回主机名和当前 URL 的端口号。
<a href="#">hostname</a>	设置或返回当前 URL 的主机名。
<a href="#">href</a>	设置或返回完整的 URL。
<a href="#">pathname</a>	设置或返回当前 URL 的路径部分。
<a href="#">port</a>	设置或返回当前 URL 的端口号。
<a href="#">protocol</a>	设置或返回当前 URL 的协议。
<a href="#">search</a>	设置或返回从问号 (?) 开始的 URL（查询部分）。

### Location 对象方法

属性	描述
<a href="#">assign()</a>	加载新的文档。
<a href="#">reload()</a>	重新加载当前文档。
<a href="#">replace()</a>	用新的文档替换当前文档。



# history对象

- 通过history对象可以访问浏览器历史。

## History 对象属性

属性

[length](#)

描述

返回浏览器历史列表中的 URL 数量。

## History 对象方法

方法

[back\(\)](#)

[forward\(\)](#)

[go\(\)](#)

描述

加载 history 列表中的前一个 URL。

加载 history 列表中的下一个 URL。

加载 history 列表中的某个具体页面。history.go(-1)与 history.back()作用相同。

# window对象

window 对象表示浏览器窗口。所有 JavaScript 全局对象、函数以及变量均自动成为 window 对象的成员。全局变量是 window 对象的属性。全局函数是 window 对象的方法。HTML DOM 的 document 也是 window 对象的属性之一。

确定浏览器窗口客户区的尺寸（不包括工具栏和滚动条）。

```
var w=window.innerWidth || document.documentElement.clientWidth ||  
    document.body.clientWidth;
```

```
var h=window.innerHeight || document.documentElement.clientHeight ||  
    document.body.clientHeight;
```

Internet Explorer 8、7、6、5

对浏览器窗口的操作：

window.open() - 打开新窗口

window.close() - 关闭当前窗口

window.moveTo() - 移动当前窗口

window.resizeTo() - 调整当前窗口的尺寸

属性	描述
<a href="#"><u>closed</u></a>	返回窗口是否已被关闭。
<a href="#"><u>defaultStatus</u></a>	设置或返回窗口状态栏中的默认文本。
<a href="#"><u>document</u></a>	对 Document 对象的只读引用。请参阅 <a href="#"><u>Document 对象</u></a> 。
<a href="#"><u>history</u></a>	对 History 对象的只读引用。请参阅 <a href="#"><u>History 对象</u></a> 。
<a href="#"><u>innerheight</u></a>	返回窗口的文档显示区的高度。
<a href="#"><u>innerwidth</u></a>	返回窗口的文档显示区的宽度。
<a href="#"><u>location</u></a>	用于窗口或框架的 Location 对象。请参阅 <a href="#"><u>Location 对象</u></a> 。
<a href="#"><u>Navigator</u></a>	对 Navigator 对象的只读引用。请参阅 <a href="#"><u>Navigator 对象</u></a> 。
<a href="#"><u>outerheight</u></a>	返回窗口的外部高度。
<a href="#"><u>outerwidth</u></a>	返回窗口的外部宽度。
<a href="#"><u>Screen</u></a>	对 Screen 对象的只读引用。请参阅 <a href="#"><u>Screen 对象</u></a> 。
<a href="#"><u>self</u></a>	返回对当前窗口的引用。等价于 Window 属性。
<a href="#"><u>status</u></a>	设置窗口状态栏的文本。
<a href="#"><u>top</u></a>	返回最顶层的先辈窗口。

事件: onload

方法	描述
<a href="#"><u>alert()</u></a>	显示带有一段消息和一个确认按钮的警告框。
<a href="#"><u>blur()</u></a>	把键盘焦点从顶层窗口移开。
<a href="#"><u>clearInterval()</u></a>	取消由 setInterval() 设置的 timeout。
<a href="#"><u>clearTimeout()</u></a>	取消由 setTimeout() 方法设置的 timeout。
<a href="#"><u>close()</u></a>	关闭浏览器窗口。
<a href="#"><u>confirm()</u></a>	显示带有一段消息以及确认按钮和取消按钮的对话框。
<a href="#"><u>createPopup()</u></a>	创建一个 pop-up 窗口。
<a href="#"><u>focus()</u></a>	把键盘焦点给予一个窗口。
<a href="#"><u>moveBy()</u></a>	可相对窗口的当前坐标把它移动指定的像素。
<a href="#"><u>moveTo()</u></a>	把窗口的左上角移动到一个指定的坐标。
<a href="#"><u>open()</u></a>	打开一个新的浏览器窗口或查找一个已命名的窗口。
<a href="#"><u>print()</u></a>	打印当前窗口的内容。
<a href="#"><u>prompt()</u></a>	显示可提示用户输入的对话框。
<a href="#"><u>resizeBy()</u></a>	按照指定的像素调整窗口的大小。
<a href="#"><u>resizeTo()</u></a>	把窗口的大小调整到指定的宽度和高度。
<a href="#"><u>scrollBy()</u></a>	按照指定的像素值来滚动内容。
<a href="#"><u>scrollTo()</u></a>	把内容滚动到指定的坐标。
<a href="#"><u>setInterval()</u></a>	按照指定的周期（以毫秒计）来调用函数或计算表达式。
<a href="#"><u>setTimeout()</u></a>	在指定的毫秒数后调用函数或计算表达式。

# document对象

每个载入浏览器的 HTML 文档都会成为 Document 对象。  
Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。

## document 对象的集合

集合	描述	IE	F	O	W3C	属性	描述	IE	F	O	W3C
<a href="#">all[]</a>	提供对文档中所有 HTML 元素的访问。	4	1	9	No	body	提供对 <body> 元素的直接访问。				
<a href="#">anchors[]</a>	返回对文档中所有 Anchor 对象的引用。	4	1	9	Yes		对于定义了框架集文档，该属性引用最外层的 <frameset>。				
applets	返回对文档中所有 Applet 对象的引用。	-	-	-	-	<a href="#">cookie</a>	设置或返回与当前文档有关的所有 cookie。	4	1	9	Yes
<a href="#">forms[]</a>	返回对文档中所有 Form 对象引用。	4	1	9	Yes	<a href="#">domain</a>	返回当前文档的域名。	4	1	9	Yes
<a href="#">images[]</a>	返回对文档中所有 Image 对象引用。	4	1	9	Yes	<a href="#">lastModified</a>	返回文档被最后修改的日期和时间。	4	1	No	No
<a href="#">links[]</a>	返回对文档中所有 Area 和 Link 对象引用。	4	1	9	Yes	<a href="#">referrer</a>	返回载入当前文档的文档的 URL。	4	1	9	Yes
						<a href="#">title</a>	返回当前文档的标题。	4	1	9	Yes
						<a href="#">URL</a>	返回当前文档的 URL。	4	1	9	Yes

## document 对象的方法

方法	描述
<a href="#">close()</a>	关闭用 <code>document.open()</code> 方法打开的输出流，并显示选定的数据。
<a href="#">getElementById()</a>	返回对拥有指定 id 的第一个对象的引用。
<a href="#">getElementsByName()</a>	返回带有指定名称的对象集合。
<a href="#">getElementsByTagName()</a>	返回带有指定标签名的对象集合。
<a href="#">getElementsByClassName()</a>	返回带有指定类名的对象集合
<a href="#">open()</a>	打开一个流，以收集来自任何 <code>document.write()</code> 或 <code>document.writeln()</code> 方法的输出。
<a href="#">write()</a>	向文档写 HTML 表达式 或 JavaScript 代码。
<a href="#">writeln()</a>	等同于 <code>write()</code> 方法，不同的是在每个表达式之后写一个换行符。

事件：

`window.onload`和`document.ready`的不同：一个要所有页面对象加载完毕加载完毕才发生，一个只要页面加载完毕就发生。  
<http://www.cnblogs.com/a546558309/p/3478344.html>

## html5

属性	描述
<code>readyState</code>	取值"loading"和"complete"，表示正在加载文档和文档加载完毕。
<code>head</code>	返回文档的head元素。 <code>document.body</code> 返回body元素。
<code>charset</code>	文档中实际使用的字符集
<code>documentElement</code>	把document当成元素对待

## scrollIntoView()

方法	描述
<code>querySelectorAll()</code>	返回与CSS选择符匹配的所有节点。
<code>querySelector()</code>	返回与CSS选择符匹配的第一个节点。
<code>matchSelector()</code>	判断元素与CSS选择符是否匹配。很多浏览器不支持
<code>hasFocus()</code>	判断文档是否获得了焦点

# screen对象

Screen 对象包含有关客户端显示屏幕的信息。

## Screen 对象属性

属性	描述
<a href="#">availHeight</a>	返回显示屏幕的高度 (除 Windows 任务栏之外)。
<a href="#">availWidth</a>	返回显示屏幕的宽度 (除 Windows 任务栏之外)。
<a href="#">bufferDepth</a>	设置或返回调色板的比特深度。
<a href="#">colorDepth</a>	返回目标设备或缓冲器上的调色板的比特深度。
<a href="#">deviceXDPI</a>	返回显示屏幕的每英寸水平点数。
<a href="#">deviceYDPI</a>	返回显示屏幕的每英寸垂直点数。
<a href="#">fontSmoothingEnabled</a>	返回用户是否在显示控制面板中启用了字体平滑。
<a href="#">height</a>	返回显示屏幕的高度。
<a href="#">logicalXDPI</a>	返回显示屏幕每英寸的水平方向的常规点数。
<a href="#">logicalYDPI</a>	返回显示屏幕每英寸的垂直方向的常规点数。
<a href="#">pixelDepth</a>	返回显示屏幕的颜色分辨率（比特每像素）。
<a href="#">updateInterval</a>	设置或返回屏幕的刷新率。
<a href="#">width</a>	返回显示器屏幕的宽度。

# Navigator 对象

Navigator 对象包含有关浏览器的信息。

## Navigator 对象属性

属性	描述
<a href="#">appName</a>	返回浏览器的代码名。
<a href="#">appMinorVersion</a>	返回浏览器的次级版本。
<a href="#">appName</a>	返回浏览器的名称。
<a href="#">appVersion</a>	返回浏览器的平台和版本信息。
<a href="#">browserLanguage</a>	返回当前浏览器的语言。
<a href="#">cookieEnabled</a>	返回指明浏览器中是否启用 <b>cookie</b> 的布尔值。
<a href="#">cpuClass</a>	返回浏览器系统的 <b>CPU</b> 等级。
<a href="#">onLine</a>	返回指明系统是否处于脱机模式的布尔值。
<a href="#">platform</a>	返回运行浏览器的操作系统平台。
<a href="#">systemLanguage</a>	返回 OS 使用的默认语言。
<a href="#">userAgent</a>	返回由客户机发送服务器的 <b>user-agent</b> 头部的值。
<a href="#">userLanguage</a>	返回 OS 的自然语言设置。

## Navigator 对象方法

方法	描述
<a href="#">javaEnabled()</a>	规定浏览器是否启用 <b>Java</b> 。
<a href="#">taintEnabled()</a>	规定浏览器是否启用数据污点 ( <b>data tainting</b> )。

# Global对象

## 函数

[decodeURI\(\)](#)

[decodeURIComponent\(\)](#)

[encodeURI\(\)](#)

[encodeURIComponent\(\)](#)

[escape\(\)](#)

[eval\(\)](#)

[getClass\(\)](#)

[isFinite\(\)](#)

[isNaN\(\)](#)

[Number\(\)](#)

[parseFloat\(\)](#)

[parseInt\(\)](#)

[String\(\)](#)

[unescape\(\)](#)

## 描述

解码某个编码的 URI。

解码一个编码的 URI 组件。

把字符串编码为 URI。

把字符串编码为 URI 组件。

对字符串进行编码。

计算 JavaScript 字符串，并把它作为脚本代码来执行。

返回一个 `JavaScript` 的 `JavaClass`。

检查某个值是否为有穷大的数。

检查某个值是否是数字。

把对象的值转换为数字。

解析一个字符串并返回一个浮点数。

解析一个字符串并返回一个整数。

把对象的值转换为字符串。

对由 `escape()` 编码的字符串进行解码。

## 属性

[Infinity](#)

[java](#)

[NaN](#)

[Packages](#)

[undefined](#)

## 描述

代表正的无穷大的数值。

代表 `java.*` 包层级的一个 `JavaPackage`。

指示某个值是不是数字值。

根 `JavaPackage` 对象。

指示未定义的值。



# 附录2、DOM对象

[Document](#)

[Anchor](#)

[Area](#)

[Base](#)

[Body](#)

[Button](#)

[Canvas](#)

[Event](#)

[Form](#)

[Frame](#)

[Frameset](#)

[IFrame](#)

[Image](#)

[Input Button](#)

[Input Checkbox](#)

[Input File](#)

[Input Hidden](#)

[Input Password](#)

[Input Radio](#)

[Input Reset](#)

[Input Submit](#)

[Input Text](#)

[Link](#)

[Meta](#)

[Object](#)

[Option](#)

[Select](#)

[Style](#)

[Table](#)

[TableCell](#)

[TableRow](#)

[Textarea](#)

```
var img1 = new Image(); // HTML5 Constructor
img1.src = 'image1.png';
img1.alt = 'alt';
document.body.appendChild(img1);
var img2 = document.createElement('img');
img2.src = 'image2.jpg';
img2.alt = 'alt text';
document.body.appendChild(img2);
// using first image in the document
alert(document.images[0].src);
```

[参考1](#)   [参考2](#)

\*TextArea要用.value改变值，  
不能采用InnerHTML？

# 附录3、其它对象

- Date对象
- Math对象
- cookie对象
- Number对象
- String对象
- RegExp对象
- DOM对象

# Date对象

```
var now=new Date()  
document.write(now.getMonth())
```

## Date 对象方法

方法	描述
<a href="#">Date()</a>	返回当日的日期和时间。
<a href="#">getDate()</a>	从 Date 对象返回一个月中的某一天 (1 ~ 31)。
<a href="#">getDay()</a>	从 Date 对象返回一周中的某一天 (0 ~ 6)。
<a href="#">getMonth()</a>	从 Date 对象返回月份 (0 ~ 11)。
<a href="#">getFullYear()</a>	从 Date 对象以四位数字返回年份。
<a href="#">getYear()</a>	请使用 <a href="#">getFullYear()</a> 方法代替。
<a href="#">getHours()</a>	返回 Date 对象的小时 (0 ~ 23)。
<a href="#">getMinutes()</a>	返回 Date 对象的分钟 (0 ~ 59)。
<a href="#">getSeconds()</a>	返回 Date 对象的秒数 (0 ~ 59)。
<a href="#">getMilliseconds()</a>	返回 Date 对象的毫秒(0 ~ 999)。
<a href="#">getTime()</a>	返回 1970 年 1 月 1 日至今的毫秒数。
<a href="#">getTimezoneOffset()</a>	返回本地时间与格林威治标准时间 (GMT) 的分钟差。
<a href="#">getUTCDate()</a>	根据世界时从 Date 对象返回月中的一天 (1 ~ 31)。
<a href="#">getUTCDay()</a>	根据世界时从 Date 对象返回周中的一天 (0 ~ 6)。
<a href="#">getUTCMonth()</a>	根据世界时从 Date 对象返回月份 (0 ~ 11)。
<a href="#">getUTCFullYear()</a>	根据世界时从 Date 对象返回四位数的年份。
<a href="#">getUTCHours()</a>	根据世界时返回 Date 对象的小时 (0 ~ 23)。
<a href="#">getUTCMinutes()</a>	根据世界时返回 Date 对象的分钟 (0 ~ 59)。
<a href="#">getUTCSeconds()</a>	根据世界时返回 Date 对象的秒钟 (0 ~ 59)。
<a href="#">getUTCMilliseconds()</a>	根据世界时返回 Date 对象的毫秒(0 ~ 999)。
<a href="#">parse()</a>	返回1970年1月1日午夜到指定日期（字符串）的毫秒数。

方法	描述
<a href="#">setDate()</a>	设置 Date 对象中月的某一天 (1 ~ 31)。
<a href="#">setMonth()</a>	设置 Date 对象中月份 (0 ~ 11)。
<a href="#">setFullYear()</a>	设置 Date 对象中的年份（四位数字）。
<a href="#">setYear()</a>	请使用 <a href="#">setFullYear()</a> 方法代替。
<a href="#">setHours()</a>	设置 Date 对象中的小时 (0 ~ 23)。
<a href="#">setMinutes()</a>	设置 Date 对象中的分钟 (0 ~ 59)。
<a href="#">setSeconds()</a>	设置 Date 对象中的秒钟 (0 ~ 59)。
<a href="#">setMilliseconds()</a>	设置 Date 对象中的毫秒 (0 ~ 999)。
<a href="#">setTime()</a>	以毫秒设置 Date 对象。
<a href="#">setUTCDate()</a>	根据世界时设置 Date 对象中月份的一天 (1 ~ 31)。
<a href="#">setUTCMonth()</a>	根据世界时设置 Date 对象中的月份 (0 ~ 11)。
<a href="#">setUTCFullYear()</a>	根据世界时设置 Date 对象中的年份（四位数字）。
<a href="#">setUTCHours()</a>	根据世界时设置 Date 对象中的小时 (0 ~ 23)。
<a href="#">setUTCMinutes()</a>	根据世界时设置 Date 对象中的分钟 (0 ~ 59)。
<a href="#">setUTCSeconds()</a>	根据世界时设置 Date 对象中的秒钟 (0 ~ 59)。
<a href="#">setUTCMilliseconds()</a>	根据世界时设置 Date 对象中的毫秒 (0 ~ 999)。
<a href="#">toSource()</a>	返回该对象的源代码。
<a href="#">toString()</a>	把 Date 对象转换为字符串。
<a href="#">toTimeString()</a>	把 Date 对象的时间部分转换为字符串。
<a href="#">toDateString()</a>	把 Date 对象的日期部分转换为字符串。
<a href="#">toGMTString()</a>	请使用 <a href="#">toUTCString()</a> 方法代替。
<a href="#">toUTCString()</a>	根据世界时，把 Date 对象转换为字符串。
<a href="#">toLocaleString()</a>	根据本地时间格式，把 Date 对象转换为字符串。
<a href="#">toLocaleTimeString()</a>	根据本地时间格式，把 Date 对象的时间部分转换为字符串。
<a href="#">toLocaleDateString()</a>	根据本地时间格式，把 Date 对象的日期部分转换为字符串。
<a href="#">UTC()</a>	根据世界时返回 1970 年 1 月 1 日到指定日期的毫秒数。
<a href="#">valueOf()</a>	返回 Date 对象的原始值。

# Math对象

```
var pi_value=Math.PI;  
var sqrt_value=Math.sqrt(15);
```

## Math 对象属性

属性	描述
<a href="#"><u>E</u></a>	返回算术常量 <b>e</b> ，即自然对数的底数（约等于 <b>2.718</b> ）。
<a href="#"><u>LN2</u></a>	返回 <b>2</b> 的自然对数（约等于 <b>0.693</b> ）。
<a href="#"><u>LN10</u></a>	返回 <b>10</b> 的自然对数（约等于 <b>2.302</b> ）。
<a href="#"><u>LOG2E</u></a>	返回以 <b>2</b> 为底的 <b>e</b> 的对数（约等于 <b>1.414</b> ）。
<a href="#"><u>LOG10E</u></a>	返回以 <b>10</b> 为底的 <b>e</b> 的对数（约等于 <b>0.434</b> ）。
<a href="#"><u>PI</u></a>	返回圆周率（约等于 <b>3.14159</b> ）。
<a href="#"><u>SQRT1_2</u></a>	返回返回 <b>2</b> 的平方根的倒数（约等于 <b>0.707</b> ）。
<a href="#"><u>SQRT2</u></a>	返回 <b>2</b> 的平方根（约等于 <b>1.414</b> ）。

## Math 对象方法

方法	描述
<a href="#"><u>abs(x)</u></a>	返回数的绝对值。
<a href="#"><u>acos(x)</u></a>	返回数的反余弦值。
<a href="#"><u>asin(x)</u></a>	返回数的反正弦值。
<a href="#"><u>atan(x)</u></a>	以介于 $-\pi/2$ 与 $\pi/2$ 弧度之间的数值来返回 <b>x</b> 的反正切值。
<a href="#"><u>atan2(y,x)</u></a>	返回从 <b>x</b> 轴到点 ( <b>x,y</b> ) 的角度（介于 $-\pi/2$ 与 $\pi/2$ 弧度之间）。
<a href="#"><u>ceil(x)</u></a>	对数进行上舍入。
<a href="#"><u>cos(x)</u></a>	返回数的余弦。
<a href="#"><u>exp(x)</u></a>	返回 <b>e</b> 的指数。
<a href="#"><u>floor(x)</u></a>	对数进行下舍入。
<a href="#"><u>log(x)</u></a>	返回数的自然对数（底为 <b>e</b> ）。
<a href="#"><u>max(x,y)</u></a>	返回 <b>x</b> 和 <b>y</b> 中的最高值。
<a href="#"><u>min(x,y)</u></a>	返回 <b>x</b> 和 <b>y</b> 中的最低值。
<a href="#"><u>pow(x,y)</u></a>	返回 <b>x</b> 的 <b>y</b> 次幂。
<a href="#"><u>random()</u></a>	返回 <b>0 ~ 1</b> 之间的随机数。
<a href="#"><u>round(x)</u></a>	把数四舍五入为最接近的整数。
<a href="#"><u>sin(x)</u></a>	返回数的正弦。
<a href="#"><u>sqrt(x)</u></a>	返回数的平方根。
<a href="#"><u>tan(x)</u></a>	返回角的正切。
<a href="#"><u>toSource()</u></a>	返回该对象的源代码。
<a href="#"><u>valueOf()</u></a>	返回 <b>Math</b> 对象的原始值。

# RegExp对象

直接量: /pattern/attributes

## attributes

修饰符	描述
i	执行对大小写不敏感的匹配。
g	执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。
m	执行多行匹配。

## 方括号

方括号用于查找某个范围内的字符:

表达式	描述
[abc]	查找方括号之间的任何字符。
[^abc]	查找任何不在方括号之间的字符。
[0-9]	查找任何从 0 至 9 的数字。
[a-z]	查找任何从小写 a 到小写 z 的字符。
[A-Z]	查找任何从大写 A 到大写 Z 的字符。
[A-z]	查找任何从大写 A 到小写 z 的字符。
[adgk]	查找给定集合内的任何字符。
[^adgk]	查找给定集合外的任何字符。
(red blue green)	查找任何指定的选项。

## 元字符

元字符（Metacharacter）是拥有特殊含义的字符:

元字符	描述
.	查找单个字符，除了换行和行结束符。
\w	查找单词字符。
\W	查找非单词字符。
\d	查找数字。
\D	查找非数字字符。
\s	查找空白字符。
\S	查找非空白字符。
\b	匹配单词边界。
\B	匹配非单词边界。
\0	查找 NUL 字符。
\n	查找换行符。
\f	查找换页符。
\r	查找回车符。
\t	查找制表符。
\v	查找垂直制表符。
\xxx	查找以八进制数 xxx 规定的字符。
\xdd	查找以十六进制数 dd 规定的字符。
\uxxxx	查找以十六进制数 xxxx 规定的 Unicode 字符。

## 量词

量词	描述
<a href="#"><u>n+</u></a>	匹配任何包含至少一个 <b>n</b> 的字符串。
<a href="#"><u>n*</u></a>	匹配任何包含零个或多个 <b>n</b> 的字符串。
<a href="#"><u>n?</u></a>	匹配任何包含零个或一个 <b>n</b> 的字符串。
<a href="#"><u>n{X}</u></a>	匹配包含 <b>X</b> 个 <b>n</b> 的序列的字符串。
<a href="#"><u>n{X,Y}</u></a>	匹配包含 <b>X</b> 或 <b>Y</b> 个 <b>n</b> 的序列的字符串。
<a href="#"><u>n{X,}</u></a>	匹配包含至少 <b>X</b> 个 <b>n</b> 的序列的字符串。
<a href="#"><u>n\$</u></a>	匹配任何结尾为 <b>n</b> 的字符串。
<a href="#"><u>^n</u></a>	匹配任何开头为 <b>n</b> 的字符串。
<a href="#"><u>?=n</u></a>	匹配任何其后紧接指定字符串 <b>n</b> 的字符串。
<a href="#"><u>?!n</u></a>	匹配任何其后没有紧接指定字符串 <b>n</b> 的字符串。

## RegExp 对象属性

属性	描述
<a href="#"><u>global</u></a>	RegExp 对象是否具有标志 <b>g</b> 。
<a href="#"><u>ignoreCase</u></a>	RegExp 对象是否具有标志 <b>i</b> 。
<a href="#"><u>lastIndex</u></a>	一个整数，标示开始下一次匹配的字符位置。
<a href="#"><u>multiline</u></a>	RegExp 对象是否具有标志 <b>m</b> 。
<a href="#"><u>source</u></a>	正则表达式的源文本。

## RegExp 对象方法

方法	描述
<a href="#"><u>compile</u></a>	编译正则表达式。
<a href="#"><u>exec</u></a>	检索字符串中指定的值。返回找到的值，并确定其位置。
<a href="#"><u>test</u></a>	检索字符串中指定的值。返回 <b>true</b> 或 <b>false</b> 。

## 支持正则表达式的 **String** 对象的方法

方法	描述
<a href="#"><u>search</u></a>	检索与正则表达式相匹配的值。
<a href="#"><u>match</u></a>	找到一个或多个正则表达式的匹配。
<a href="#"><u>replace</u></a>	替换与正则表达式匹配的子串。
<a href="#"><u>split</u></a>	把字符串分割为字符串数组。

# Number 对象

Number 对象是原始数值的包装对象。

## Number 对象属性

属性

[constructor](#)

[MAX\\_VALUE](#)

[MIN\\_VALUE](#)

[NaN](#)

[NEGATIVE\\_INFINITY](#)

[POSITIVE\\_INFINITY](#)

prototype

描述

返回对创建此对象的 Number 函数的引用。

可表示的最大的数。

可表示的最小的数。

非数字值。

负无穷大，溢出时返回该值。

正无穷大，溢出时返回该值。

使您有能力向对象添加属性和方法。

## Number 对象方法

方法

[toString](#)

[toLocaleString](#)

[toFixed](#)

[toExponential](#)

[toPrecision](#)

[valueOf](#)

描述

把数字转换为字符串，使用指定的基数。

把数字转换为字符串，使用本地数字格式顺序。

把数字转换为字符串，结果的小数点后有指定位数的数字。

把对象的值转换为指数计数法。

把数字格式化为指定的长度。

返回一个 Number 对象的基本数字值。

# String 对象

String 对象用于处理文本（字符串）。`var s1=new String("abc")`

## String 对象属性

属性	描述
constructor	对创建该对象的函数的引用
<a href="#">length</a>	字符串的长度
prototype	允许您向对象添加属性和方法

## String 对象方法

方法	描述
<a href="#">anchor()</a>	创建 HTML 锚。
<a href="#">big()</a>	用大号字体显示字符串。
<a href="#">blink()</a>	显示闪动字符串。
<a href="#">bold()</a>	使用粗体显示字符串。
<a href="#">charAt()</a>	返回在指定位置的字符。
<a href="#">charCodeAt()</a>	返回在指定的位置的字符的 Unicode 编码。
<a href="#">concat()</a>	连接字符串。
<a href="#">fixed()</a>	以打字机文本显示字符串。
<a href="#">fontcolor()</a>	使用指定的颜色来显示字符串。
<a href="#">fontsize()</a>	使用指定的尺寸来显示字符串。
<a href="#">fromCharCode()</a>	从字符编码创建一个字符串。
<a href="#">indexOf()</a>	检索字符串。

方法	描述
<a href="#">italics()</a>	使用斜体显示字符串。
<a href="#">lastIndexOf()</a>	从后向前搜索字符串。
<a href="#">link()</a>	将字符串显示为链接。
<a href="#">localeCompare()</a>	用本地特定的顺序来比较两个字符串。
<a href="#">match()</a>	找到一个或多个正则表达式的匹配。
<a href="#">replace()</a>	替换与正则表达式匹配的子串。
<a href="#">search()</a>	检索与正则表达式相匹配的值。
<a href="#">slice()</a>	提取字符串的片断，并在新的字符串中返回被提取的部分。
<a href="#">small()</a>	使用小字号来显示字符串。
<a href="#">split()</a>	把字符串分割为字符串数组。
<a href="#">strike()</a>	使用删除线来显示字符串。
<a href="#">sub()</a>	把字符串显示为下标。
<a href="#">substr()</a>	从起始索引号提取字符串中指定数目的字符。
<a href="#">substring()</a>	提取字符串中两个指定的索引号之间的字符。
<a href="#">sup()</a>	把字符串显示为上标。
<a href="#">toLocaleLowerCase()</a>	把字符串转换为小写。
<a href="#">toLocaleUpperCase()</a>	把字符串转换为大写。
<a href="#">toLowerCase()</a>	把字符串转换为小写。
<a href="#">toUpperCase()</a>	把字符串转换为大写。
<a href="#">toSource()</a>	代表对象的源代码。
<a href="#">toString()</a>	返回字符串。
<a href="#">valueOf()</a>	返回某个字符串对象的原始值。



# 附录4、富文本编辑框命令

下表列出了那些被支持最多的命令（第二个参数表示是否显示对话框）：

命令值	(第三个参数)	说明
backcolor	颜色字符串	设置文档的背景颜色
bold	null	将选择的文本转换为粗体
copy	null	将选择的文本复制到剪贴板
createlink	URL字符串	将选择的文本转换成一个链接，指向指定的URL
cut	null	将选择的文本剪切到剪贴板
delete	null	删除选择的文本
fontname	字体名称	将选择的文本修改为指定字体
fontsize	1~7	将选择的文本修改为指定字体大小
forecolor	颜色字符串	将选择的文本修改为指定的颜色
formatblock	要包围当前文本块的HTML标签；如<h1>	使用指定的HTML标签来格式化选择的文本块
indent	null	缩进文本
inserthorizontalrule	null	在插入字符处插入一个<hr>元素
insertimage	图像的URL	在插入字符处插入一个图像
insertorderedlist	null	在插入字符处插入一个<ol>元素
insertunorderedlist	null	在插入字符处插入一个<ul>元素
insertparagraph	null	在插入字符处插入一个<p>元素
italic	null	将选择的文本转换成斜体
justifycenter	null	将插入光标所在文本块居中对齐
justifyleft	null	将插入光标所在文本块左对齐
outdent	null	凸排文本（减少缩进）
paste	null	将剪贴板中的文本粘贴到选择的文本
removeformat	null	移除文本块的块级格式。这是撤销formatblock命令的操作
selectall	null	选择文档中的所有文本
underline	null	为选择的文本添加下划线
unlink	null	移除文本的链接。这是撤销createlink命令的操作

[参考1](#) [参考2](#)

# 附录5、元素的其它操作

<code>focus()</code>	让输入类元素得焦点 htm5。
<code>scrollIntoView()</code>	滚动页面让元素可见；
<code>contains()</code>	是否为后代元素；
<code>insertAdjacentHTML()</code>	在元素的开始标签和结束标签的前后插入元素。
<code>compareDocumentPosition()</code>	比较位置：1-无关 2-居前 3-居后 8-包含 16-被包含。

# 附录6、DOM事件和方法

## • 鼠标事件

属性	描述	DOM
<a href="#">onclick</a>	当用户点击某个对象时调用的事件句柄。	2
<a href="#">oncontextmenu</a>	在用户点击鼠标右键打开上下文菜单时触发	
<a href="#">ondblclick</a>	当用户双击某个对象时调用的事件句柄。	2
<a href="#">onmousedown</a>	鼠标按钮被按下。	2
<a href="#">onmouseenter</a>	当鼠标指针移动到元素上时触发。	2
<a href="#">onmouseleave</a>	当鼠标指针移出元素时触发	2
<a href="#">onmousemove</a>	鼠标被移动。	2
<a href="#">onmouseover</a>	鼠标移到某元素之上。	2
<a href="#">onmouseout</a>	鼠标从某元素移开。	2
<a href="#">onmouseup</a>	鼠标按键被松开。	2

## • 键盘事件

属性	描述	DOM
<a href="#">onkeydown</a>	某个键盘按键被按下。	2
<a href="#">onkeypress</a>	某个键盘按键被按下并松开。	2
<a href="#">onkeyup</a>	某个键盘按键被松开。	2

DOM: 指明使用的 DOM 属性级别。

- 框架/对象（Frame/Object）事件

属性	描述	DOM
<a href="#">onabort</a>	图像的加载被中断。( <object>)	2
<a href="#">onbeforeunload</a>	该事件在即将离开页面（刷新或关闭）时触发	2
<a href="#">onerror</a>	在加载文档或图像时发生错误。( <object>, <body>和 <frameset>)	
<a href="#">onhashchange</a>	该事件在当前 URL 的锚部分发生修改时触发。	
<a href="#">onload</a>	一张页面或一幅图像完成加载。	2
<a href="#">onpageshow</a>	该事件在用户访问页面时触发	
<a href="#">onpagehide</a>	该事件在用户离开当前网页跳转到另外一个页面时触发	
<a href="#">onresize</a>	窗口或框架被重新调整大小。	2
<a href="#">onscroll</a>	当文档被滚动时发生的事件。	2
<a href="#">onunload</a>	用户退出页面。( <body> 和 <frameset>)	2

- 表单事件

属性	描述	DOM
<a href="#">onblur</a>	元素失去焦点时触发	2
<a href="#">onchange</a>	该事件在表单元素的内容改变时触发( <input>, <keygen>, <select>, 和 <textarea>)	2
<a href="#">onfocus</a>	元素获取焦点时触发	2
<a href="#">onfocusin</a>	元素即将获取焦点时触发	2
<a href="#">onfocusout</a>	元素即将失去焦点时触发	2
<a href="#">oninput</a>	元素获取用户输入时触发	3
<a href="#">onreset</a>	表单重置时触发	2
<a href="#">onsearch</a>	用户向搜索域输入文本时触发 ( <input="search">)	
<a href="#">onselect</a>	用户选取文本时触发 ( <input> 和 <textarea>)	2
<a href="#">onsubmit</a>	表单提交时触发	2

- 剪贴板事件

属性	描述	DOM
<a href="#">oncopy</a>	该事件在用户拷贝元素内容时触发	
<a href="#">oncut</a>	该事件在用户剪切元素内容时触发	
<a href="#">onpaste</a>	该事件在用户粘贴元素内容时触发	

- 拖动事件

事件	描述	DOM
<a href="#">ondrag</a>	该事件在元素正在拖动时触发	
<a href="#">ondragend</a>	该事件在用户完成元素的拖动时触发	
<a href="#">ondragenter</a>	该事件在拖动的元素进入放置目标时触发	
<a href="#">ondragleave</a>	该事件在拖动元素离开放置目标时触发	
<a href="#">ondragover</a>	该事件在拖动元素在放置目标上时触发	
<a href="#">ondragstart</a>	该事件在用户开始拖动元素时触发	
<a href="#">ondrop</a>	该事件在拖动元素放置在目标区域时触发	

- 打印事件

属性	描述	DOM
<a href="#">onafterprint</a>	该事件在页面已经开始打印，或者打印窗口已经关闭时触发	
<a href="#">onbeforeprint</a>	该事件在页面即将开始打印时触发	

## • 多媒体（Media）事件

事件	描述
<a href="#">onabort</a>	事件在视频/音频（audio/video）终止加载时触发。
<a href="#">oncanplay</a>	事件在用户可以开始播放视频/音频（audio/video）时触发。
<a href="#">oncanplaythrough</a>	事件在视频/音频（audio/video）可以正常播放且无需停顿和缓冲时触发。
<a href="#">ondurationchange</a>	事件在视频/音频（audio/video）的时长发生变化时触发。
<a href="#">onemptied</a>	当期播放列表为空时触发
<a href="#">onended</a>	事件在视频/音频（audio/video）播放结束时触发。
<a href="#">onerror</a>	事件在视频/音频（audio/video）数据加载期间发生错误时触发。
<a href="#">onloadeddata</a>	事件在浏览器加载视频/音频（audio/video）当前帧时触发触发。
<a href="#">onloadedmetadata</a>	事件在指定视频/音频（audio/video）的元数据加载后触发。
<a href="#">onloadstart</a>	事件在浏览器开始寻找指定视频/音频（audio/video）触发。
<a href="#">onpause</a>	事件在视频/音频（audio/video）暂停时触发。
<a href="#">onplay</a>	事件在视频/音频（audio/video）开始播放时触发。
<a href="#">onplaying</a>	事件在视频/音频（audio/video）暂停或者在缓冲后准备重新开始播放时触发。
<a href="#">onprogress</a>	事件在浏览器下载指定的视频/音频（audio/video）时触发。
<a href="#">onratechange</a>	事件在视频/音频（audio/video）的播放速度发送改变时触发。
<a href="#">onseeked</a>	事件在用户重新定位视频/音频（audio/video）的播放位置后触发。
<a href="#">onseeking</a>	事件在用户开始重新定位视频/音频（audio/video）时触发。
<a href="#">onstalled</a>	事件在浏览器获取媒体数据，但媒体数据不可用时触发。
<a href="#">onsuspend</a>	事件在浏览器读取媒体数据中止时触发。
<a href="#">ontimeupdate</a>	事件在当前的播放位置发送改变时触发。
<a href="#">onvolumechange</a>	事件在音量发生改变时触发。
<a href="#">onwaiting</a>	事件在视频由于要播放下一帧而需要缓冲时触发。

## • 动画事件

事件	描述	DOM
<a href="#">animationend</a>	该事件在 CSS 动画结束播放时触发	
<a href="#">animationiteration</a>	该事件在 CSS 动画重复播放时触发	
<a href="#">animationstart</a>	该事件在 CSS 动画开始播放时触发	

## • 过渡事件

事件	描述	DOM
<a href="#">transitionend</a>	该事件在 CSS 完成过渡后触发。	

## • 其他事件

事件	描述	DOM
onmessage	该事件通过或者从对象(WebSocket, Web Worker, Event Source 或者子 frame 或父窗口)接收到消息时触发	
onmousewheel	<b>已废弃。</b> 使用 <a href="#">onwheel</a> 事件替代	
<a href="#">ononline</a>	该事件在浏览器开始在线工作时触发。	
<a href="#">onoffline</a>	该事件在浏览器开始离线工作时触发。	
onpopstate	该事件在窗口的浏览历史（history 对象）发生改变时触发。	
<a href="#">onshow</a>	该事件当 <menu> 元素在上下文菜单显示时触发	
onstorage	该事件在 Web Storage(HTML 5 Web 存储)更新时触发	
<a href="#">ontoggle</a>	该事件在用户打开或关闭 <details> 元素时触发	
<a href="#">onwheel</a>	该事件在鼠标滚轮在元素上下滚动时触发	



## • 常用事件

onclick	元素被点击
onabort	图像加载被中断
onblur	元素失去焦点
onchange	用户改变域的内容
onerror	当加载文档或图像时发生某个错误
onfocus	元素获得焦点
onload	某个页面或图像被完成加载
onreset	重置按钮被点击
onresize	窗口或框架被调整尺寸
onselect	文本被选定
onsubmit	提交按钮被点击
onunload	用户退出页面
onpropertychange	元素属性改变 (event.propertyName设置或返回元素的变化了的属性的名称)

## •鼠标事件

- 事件列举:

```
<div onclick="alert('click')" > hello </div>
```

click	单击	dblclick	双击	mousedown	按下	mouseup	放开
mouseenter	进入	mouseout	离开	mousemove	移动	mouseover	悬浮

- 事件属性:

event.screenX, event.screenY	鼠标在窗口的坐标
event.clientX,event.clientY	鼠标在客户区的坐标
event.offsetX, event.offsetY	鼠标在点击对象上的坐标
event.pageX,event.pageY	鼠标在页面中的坐标
event.x,event.y	相对于上级元素的坐标(默认为BODY)
event.button	按下的鼠标键: 0 没按键 1 按左键 2 按右键 3 按左右键 4 按中间键 5 按左键和中间键 6 按右键和中间键 7 按所有的键
event.fromElement	鼠标移动离开的对象
event.toElement	鼠标移动进入的对象
event.srcElement	触发事件(鼠标点击)的元素
event.target	冒泡事件中的最小范围元素
event.currentTarget	当前事件的元素 (同this)

## ● 键盘事件

keydown  
keyup, keypress  
event.keyCode  
event.charCode  
event.altKey, event.ctrlKey, event.shiftKey:

按下任意键  
按下字符键，长按则连续触发  
按键内码（扫描码）  
按键字符(可显示)  
alt, ctrl或shift的键是否按下，按下为true

## ● 触摸屏事件

touchstart  
touchend  
touchmove  
touchenter  
touchcancel  
touchleave  
touches  
targetTouches  
changeTouches  
orientationchange  
devicemotion

开始触摸  
结束触摸。触摸位置的参数与鼠标事件一样。  
手指移动。  
手指进入目标的有效区域  
中断触摸或手指离开有效区域  
中断还在触摸但是离开了有效区域  
跟踪Touch对象的数组  
特定于时间目标的Touch对象数组  
自从上次触摸以来发生了什么改变的Touch对象数组  
方向：0-肖像模式 90-左旋90度 90-右旋90度  
设备正在移动

## ● 手势事件

gesturestart  
gesturechange  
gestureend  
event.scale  
event.orientation

两个手指均触摸时发生  
至少有一个手指位置发生变化。  
至少有一个手指从触摸屏移开。  
两指间距离,从1开始，值越大距离越大  
手指变化的旋转角度，正(负)值为正(逆)时针旋转

```
<head>
<script type="text/javascript">
    function bind(fn, context) {
        return function () {
            return fn.apply(context, arguments);
        };
    }
</script>
</head>
<body>
<p id="p1">请点击该文本</p>
<p id="p2">this is another paragraph</p>
<script type="text/javascript">
var handler = {
    message: "Event handled",
    handleClick: function (event) {
        alert(this.message);
    }
};
var btn = document.getElementById("p1");
btn.onclick = bind(handler.handleClick, handler);
</script>
</body>
```

# 附录7、事件对象(event)

## • 常量

静态变量	描述	DOM
CAPTURING-PHASE	当前事件阶段为捕获阶段(3)	1
AT-TARGET	当前事件是目标阶段,在评估目标事件(1)	2
BUBBLING-PHASE	当前的事件为冒泡阶段 (2)	3

## • 属性

属性	描述	DOM
<a href="#">bubbles</a>	返回布尔值，指示事件是否是起泡事件类型。	2
<a href="#">cancelable</a>	返回布尔值，指示事件是否可拥可取消的默认动作。	2
<a href="#">currentTarget</a>	返回其事件监听器触发该事件的元素。	2
eventPhase	返回事件传播的当前阶段。	2
<a href="#">target</a>	返回触发此事件的元素（事件的目标节点）。	2
<a href="#">timeStamp</a>	返回事件生成的日期和时间。	2
<a href="#">type</a>	返回当前 Event 对象表示的事件的名称。	2

## • 方法

方法	描述	DOM
initEvent()	初始化新创建的 Event 对象的属性。	2
preventDefault()	通知浏览器不要执行与事件关联的默认动作。	2
stopPropagation()	不再派发事件。	2

# 附录8、 鼠标/键盘事件对象

属性	描述	DOM
<a href="#">altKey</a>	返回当事件被触发时, "ALT" 是否被按下。	2
<a href="#">button</a>	返回当事件被触发时, 哪个鼠标按钮被点击。	2
<a href="#">clientX</a>	返回当事件被触发时, 鼠标指针的水平坐标。	2
<a href="#">clientY</a>	返回当事件被触发时, 鼠标指针的垂直坐标。	2
<a href="#">ctrlKey</a>	返回当事件被触发时, "CTRL" 键是否被按下。	2
<a href="#">Location</a>	返回按键在设备上的位置	3
<a href="#">charCode</a>	返回onkeypress事件触发键值的字母代码。	2
<a href="#">key</a>	在按下按键时返回按键的标识符。	3
<a href="#">keyCode</a>	返回onkeypress事件触发的键的值的字符代码, 或者 onkeydown 或 onkeyup 事件的键的代码。	2
<a href="#">which</a>	返回onkeypress事件触发的键的值的字符代码, 或者 onkeydown 或 onkeyup 事件的键的代码。	2
<a href="#">metaKey</a>	返回当事件被触发时, "meta" 键是否被按下。	2
<a href="#">relatedTarget</a>	返回与事件的目标节点相关的节点。	2
<a href="#">screenX</a>	返回当某个事件被触发时, 鼠标指针的水平坐标。	2
<a href="#">screenY</a>	返回当某个事件被触发时, 鼠标指针的垂直坐标。	2
<a href="#">shiftKey</a>	返回当事件被触发时, "SHIFT" 键是否被按下。	2

方法	描述	W3C
<code>initMouseEvent()</code>	初始化鼠标事件对象的值	2
<code>initKeyboardEvent()</code>	初始化键盘事件对象的值	3

# 附录9、其它对象的方法

- 目标事件对象的方法

方法	描述	DOM
addEventListener()	允许在目标事件中注册监听事件(IE8 = attachEvent())	2
dispatchEvent()	允许发送事件到监听器上 (IE8 = fireEvent())	2
removeEventListener()	运行一次注册在事件目标上的监听事件(IE8 = detachEvent())	2

- 事件监听对象的方法

方法	描述	DOM
handleEvent()	把任意对象注册为事件处理程序	2

- 文档事件对象的方法

方法	描述	DOM
createEvent()		2

# 附录10、IE8.0及更早版本的事件

```
var p1=document.getElementById("p1");
var f1=function(){alert("hello");};
if(p1.addEventListener){
    p1.addEventListener("click", function(event){
        alert(event.type+":"+event.clientX+","+event.clientY+"
            +event.screenX+","+event.screenY);
    });
}
else{
    p1.attachEvent("onclick", function(){ var event = window.event;
        alert(event.type+":"+event.clientX+","+event.clientY
            +event.screenX+","+event.screenY);
    });
}
```

p1.detachEvent("onclick", function(){ ... }); //删除事件处理程序

window.event.returnValue = false // IE9之前的版本用于取消默认行为

window.event.cancelBubble = true; // IE9之前的版本用于阻止事件传播：



可以做一个跨浏览器平台的事件绑定函数：

```
var addMyEvent = function(obj, ev, fn){  
    if(obj.addEventListener){  
        obj.addEventListener(ev, fn, false);  
    }else if(obj.attachEvent){  
        obj.attachEvent("on"+ev, fn);  
    }else{  
        obj["on"+ev] = fn;  
    }  
}
```

# 附录11、自定义属性的特征

自定义属性默认是可枚举的，可以通过defineProperty重新定义进行改变。

```
var person1={name:"Nicholas"};
alert(person1.propertyIsEnumerable("name"));           //true
Object.defineProperty(person1,"name",{enumerable:false});
alert(person1.propertyIsEnumerable("name"));           //false
Object.defineProperty(person1,"name",{enumerable:true});
alert(person1.propertyIsEnumerable("name"));           //true
```

如果定义为不可配置的，则不能改变或删除对象的属性。通过定义为不可写入的，则不能改变属性值。设置禁止扩展后将不允许增加新属性。如果要求只能读写属性值，不可增删属性和改变属性特征，则要设置为被封印。如果不允许属性进行任何更改，其只能读出属性值，则可冻结对象。冻结对象也是不可扩展和被封印的。

```
Object.defineProperty(person1,"name",{configurable:false});
Object.defineProperty(person1,"name",{writable:false});
Object.preventExtensions(person1);
alert(Object.isExtensible(person1)); //false
Object.seal(person1);
alert(Object.isSealed(person1));      //true
Object.freeze(person1);
alert(Object.isFrozen(person1));      //true
```

通过描述子得到属性特征:

```
var person1={name:"Nicholas"};
var desc = Object.getOwnPropertyDescriptor(person1,"name");
alert(desc.enumerable);
alert(desc.configurable);
alert(desc.writable);
alert(desc.value);
```

采用defineProperty定义一个属性:

```
var person1 = {};
Object.defineProperty(person1,"name", {
    value:"Nicholas",
    enumerable:true,
    configurable:true,
    writable:true
})
alert(person1.name);    // Nicholas
```

采用defineProperties可以一次定义多个属性，还可以定义访问器属性(get和set函数)：

```
var person2 = {};  
Object.defineProperties(person2,{  
  _name:{  
    value:"Nicholas",  
    enumerable:true,  
    configurable:true,  
    writable:true  
  },  
  name:{  
    get:function(){  
      return this._name;  
    },  
    set:function(val){  
      this._name = val;  
    },  
    enumerable: true,  
    configurable: true  
  }  
})  
person2.name="David";  
alert(person2.name);
```

数据属性和访问器属性

# 附录12、通过原型继承

JavaScript的对象的原型属性默认都是指向Object.prototype。Object.prototype的原型属性为空。当查找对象属性时，先到本对象的自有属性中查找，然后沿着原型链进行查找。

```
var book={  
    title:"My Road",  
};
```

// 上面定义与下面定义相同，Object.create的第一个参数赋值给对象的原型属性  
// 第二个参数用于定义其它属性

```
var book=Object.create(Object.prototype,{  
    title:{  
        configurable:true,  
        enumerable:true,  
        value:"My Road...",  
        writable:true  
    }  
})  
alert(book.title);
```

```

// var person1={ name:"Greg", sayName: function(){ alert(this.name);}}; //与下面定义等同
var person1=Object.create(Object.prototype,{ // 改为null, 下面的toString()将出错
    name:{
        configurable:true,
        enumerable:true,
        value:"Greg",
        writable:true
    },
    sayName:{
        configurable:true,
        enumerable:true,
        value:function(){alert(this.name);},
        writable:true
    }
});
var person2=Object.create(person1,{
    //constructor:person2,
    name:{
        configurable:true,
        enumerable:true,
        value:"Nicholas",
        writable:true
    }
});
person1.sayName(); //Greg
person2.sayName(); //Nicholas
alert(person2.toString());//[object Object].

```

如果只继承原型的属性，则可以使用原型赋值： `Person.prototype=Parent.prototype`;  
如果Person要使用Parent的构造函数，可以对Parent使用call的方法。

```
function Person(name){
    this.name = name;
}
Person.prototype.sayName=function(){alert(this.name)};
function Person1(name){
    Person.call(this,name);
}
Person1.prototype = Person.prototype;           // 可以用Object.create创建
var person1 = new Person1("Greg");
person1.sayName();                               // Greg
alert(person1.hasOwnProperty("Name"));           // true
```

利用call可以直接调用双亲中的方法：

```
function Person1(name){
    Person.call(this,name);
    Person.prototype.sayName.call(this,name);
}
```

# 附录13、私有成员

利用立调函数形成私有成员的方法（利用了闭包）：

```
var person =  
  (function(){  
    var age = 25; // 私有成员，一般采用下划线命名_age  
    return {  
      name: "Nicholas",  
      getAge: function(){  
        return age;  
      },  
      growOlder: function(){  
        age++;  
      }  
    }  
  })()  
;  
console.log(person.name); // Nicholas  
console.log(person.getAge()); // 25  
person.age = 125; // 不能访问age  
console.log(person.getAge()); // 25  
person.growOlder();  
console.log(person.getAge()); // 26
```



也可以先定义好所有的内部数据和函数，然后再定义要返回对象（暴露模块模式）。

```
var person =
    (function(){
        var age = 25;
        function getAge(){
            return age;
        };
        function growOlder(){
            age++;
        };
        return {
            name:"Nicholas",
            getAge:getAge,
            growOlder:growOlder
        };
    })();
console.log(person.name);           // Nicholas
console.log(person.getAge());       // 25
person.age = 125;
console.log(person.getAge());       // 25
person.growOlder();
console.log(person.getAge());       // 26
```

利用构造函数形成私有成员：

```
function Person(name){  
    var age = 25;  
    this.name = name;  
    this.getAge=function(){  
        return age;  
    }  
    this.growOlder=function(){  
        age++;  
    }  
}  
var person1 = new Person("Nicholas");  
var person2 = new Person("Greg");  
  
console.log(person1.name);           // Nicholas  
console.log(person2.name);           // Greg  
console.log(person1.getAge());        // 25  
person1.age = 125;  
console.log(person1.getAge());        // 25  
person1.growOlder();  
console.log(person1.getAge());        // 26  
console.log(person2.getAge());        // 25
```

利用原型共享函数，也共享了私有变量：

```
var Person = (function(){
    var age = 25;
    function InnerPerson(name){
        this.name = name;
    };
    InnerPerson.prototype.getAge=function(){
        return age;
    };
    InnerPerson.prototype.growOlder=function(){
        age++;
    };
    return InnerPerson;
})();
var person1 = new Person("Nicholas");
var person2 = new Person("Greg");
console.log(person1.name);           // Nicholas
console.log(person2.name);           // Greg
person1.age = 125;
console.log(person1.getAge());        // 25
person1.growOlder();
console.log(person1.getAge());        // 26
console.log(person2.getAge());        // 26
```