**（1）适用：**

HTML和web，可广泛用于服务器，PC，笔记本电脑，平板电脑和手机。

是所有现代浏览器和HTML5的默认脚本语言。

实际写的时候是跟HTML写一起，或者通过HTML来调用的。

**（2）本质：**

脚本语言。

①轻量级编程语言；

②可插入HTML页面的编程代码；

③插入HTML页面后，可由所有的现代浏览器执行；

④易学。

**（3）<script>与</script>标签**

HTML的脚本必须位于<script>和</script>之中。

在这里，就是指JavaScript脚本必须位于<script>和</script>之中。

即：

<script>

JS脚本

</script>

**（4）JavaScript的作用**

①写入HTML用于输出（例如输出一行文字）；

②对事件作出反应（例如点击按钮后有变化）；

③其他（略）。

**（5）如何使用**

创建一个文本文档，然后后缀改为html，用记事本打开，然后就可以往里面写东西了。

写出来的是网页（html），以网页形式展现输出结果。

也可以使用微软的FrontPage2003软件，特点是代码加亮、所写即所见等，但缺点貌似是加载其他js文件时无法显示加载的内容。

也可以使用VS（我的VS2015可以），会代码加亮、自动缩进等，不过经常会有警告性报错，而且查看效果的话需要通过浏览器。

或者使用其他html编辑工具。

**（6）标签**

成对出现的<>（共2组）就是标签。

如<p>文本</p>就是一组标签。

标签可以加id，如<p id="abc">文本</p>，id用于之后寻找其使用。

寻找id使用getElementById("abc")这样的办法。

然后在这样的命令后面加对应的东西，如innerHTML（文本替换），value（得到值）。

如

<p id="abc">abc</p>

<button type="button" onclick="run()">点击</button>

<script>

function run()  
 {  
 document.getElementById("abc").innerHTML="ppp";

}

</script>

像这样的代码，在执行时，用ppp文本替换abc文本。

**（7）具体解释**：

第一行是输出文字abc，然后给他一个id（用来找她）。

第二行是输出一个按钮（button），按钮上面的文字是点击。

第三行到最后</script>是js脚本。这个脚本里面包含了一个函数（function）；

函数的效果是，在标签id为abc的地方（通过getElementById获得），用文本ppp替换之。

**（8）输出时间：**

<p id="time"></p>

<script>

var nowtime=new Date(); //获得时间

localtime=nowtime.toString(); //将时间转为字符串

document.getElementById("time").innerHTML=localtime; //在id为time的地方输出时间，注意，time需要在这行之前才可以

</script>

输出效果：

Sun Mar 27 2016 00:26:32 GMT+0800 (中国标准时间)

**（9）关于时间的其他函数：**

hour=nowtime.getHours(); //将小时赋给hour

min=nowtime.getMinutes(); //分钟

second=nowtime.getSeconds(); //秒

day=nowtime.getDate(); //天

mon=nowtime.getMonth(); //月

year=nowtime.getFullYear(); //年

datashow=year+"年"+mon+"月"+day+"日"+hour+"时"+min+"分"+second+"秒";

document.getElementById("time").innerHTML=datashow; //在id为time的地方输出时间，注意，time需要在这行之前才可以

输出效果：

2016年2月27日0时38分27秒

**（10）导入一个js文件**

使用

<script src="time.js"> //读取time.js这个JavaScript文件内容

</script>

然后可以将之前的关于time的js命令，放入time.js文件内（无需加上<script>标签）即可。

**（11）导入js文件的潜在问题**

假如有一个

<script src=”111.js”>

js脚本1

</script>

那么，由于导入了111.js的原因，“js脚本1”事实上是不会被执行的。

准确的说，包含导入js脚本功能的script标签，里面的脚本都不会执行（不能保证全部，但目前学到的都不会被执行，如果有可以执行的，其后会在这里修改说明）。

**（12）补充说明适应人群**

至少要有一点的编程概念，毫无编程概念的话，恐怕是有一些难度的。

由于时间很紧张，所以不能保证把基础概念写的很细，但总体的格式、思想都是相近的。我只有c++的一些编程基础，因此只要学过一点c++或者类似的编程，应该是都能搞得懂的。

**（13）出现一个弹框，要求输入文字，并且验证**

验证成功执行某个函数（显示文本），验证失败不执行。

ttt=prompt("enter"); //出现一个弹框，标题为enter，将输入的值赋给ttt

if(ttt=="aaa") //验证ttt的内容是不是字符串aaa

{

document.getElementById("show").innerHTML="输入正确！"; //如果是的话，在id为show的地方，用文本替换之

}

**（14）条件判断**

参考（13），if就是一个条件判断语句，括号里是条件，如果成立（条件为true），执行括号内的，不成立，执行else后面的括号内的，如果没有else则跳过这部分。

类似的还有while，for等，这些不仅仅包含条件，而且也是循环语句。具体略，可以参考C++的（我之前写过C++的帖子）。

**（15）事件处理程序**

在发生某种事件时，怎么处理。

先上代码：

<img src="01.png" onmouseover="src='02.png'" onmouseout="src='01.png'"> <!-- 这行的意思是，插入一个图片，初始是01.png，鼠标移动上去后，是02.png，移动走又变回01.png了-->

位置是在第一个标签之中，类型有（似乎对大小写并不敏感，即onMouseOver和onmouseover是一样的）

①按下时：onclick=”执行的语句”;

②鼠标移动到上面时：onMouseOver=”执行的语句”;

③鼠标离开：onMouseOut=”执行的语句”;

④点击后生效：onClick=”执行的语句”;（仅限鼠标左键）

⑤鼠标按下后生效：onMouseDown=”执行的语句”;（左右键都有用）

⑥增加条件判断（初始1#图片，第一次点击更换为2#图片，再次点击更换为1#图片，然后交替）

<img id="01png" src="01.png"onclick="png01()">

<script>

i=0; //需要在外面声明才行

function png01()

{

if(i==0){i=1;document.getElementById("01png").src="02.png";} //判断更改图片

else{i=0;document.getElementById("01png").src="01.png";} //判断更改图片

}

</script>

⑦鼠标在上面是一个鼠标，鼠标离开又是另外一个图片

<img id="01png" src="01.png" onmouseover="src='02.png'" onmouseout="src='01.png'">

⑧判断路径中，文件名是否有某个关键词；

function mouseover()

{

png=document.getElementById("01png");

if(png.src.match("01.png")) //如果路径中有关键词01，

{

png.src="02.png";

}

else

{

png.src="01.png";

}

}

**（16）注释**

html注释的语法是：

<!--这里面写注释-->

js注释的语法：

js标签之内（即<script>和</script>之间的部分），可以像c++那样用双斜线。

**（17）图片**

这里html内容：

①标签<img src=”图片链接”>

②限制高度和宽度（会导致图片变形）：width=””和height=””;

③响应，例如鼠标点击后做什么，参考（15）

④在文字之中插入图片：

<p>右边是一个美女图片<img id="01png" src="01.png" width="200" height="200" onmouseover="mouseover()" onmouseout="mouseover()"> 左边是美女图片</p>

**（18）超链接**

在使用<a href=”链接地址”></a> 即可，将触发链接的内容加入到<a>中间如：

如：

<ahref="http://baidu.com"target="\_blank"><!--加入target="\_blank"表示是新窗口，不加则是在之前的窗口-->

<imgid="01png"src="01.png"width="100"height="100"onmouseover="mouseover()"onmouseout="mouseover()">

</a>效果是当前页面更换为baidu.com（注意：如果使用外链，需要加入前缀http://

①如果是链接到本地（或者网站某个地址），则使用相对路径；

②链接到同一个页面的不同位置，参考链接：

<http://www.w3school.com.cn/tiy/t.asp?f=html_link_locations>

③如果图片不能存在的话，用文本替代：

<imgsrc="不存在的图片.png"alt="这里是不存在的图片">

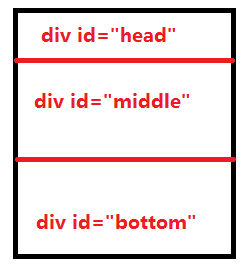
效果图：



**（19）使用div来布局**

<div>布局的好处是，提前把网站页面划分为若干个区域，然后设置好每个区域的大小。

如图片，先将页面分为3个区域。



然后在每个区域中再细分。

①一般在div中使用class属性：<div class=”某个样式”></div>

然后这个div则使用class的这个样式了。

②一个div实际上是一行（也就是说不同div之间是换行），但貌似可以通过class来让其位于不同的位置。

更多的内容，等之后学习再补充

**（20）事件触发的顺序**

假如文档中有多个脚本（例如自动执行的脚本），那么他们执行是有一定的顺序的（在ＨＴＭＬ文档之中）：

①先执行<head>内的<script>标签。因此，这里的js脚本通常是嵌入代码、指向JS文件，可以在这里定义其后要使用的函数；

②其次是执行<body>内的script标签；

③然后是执行触发的脚本内容，例如在button按钮上添加的onclick=””命令，就是在点击后触发的。

**（21）文档对象模型（DOM）**

文档对象模型（Document Ojbect Model），效果是允许脚本控制Web页面、窗口、文档。对于JS来说，他是制作动态页面的强有力工具。

DOM不是JavaScript语言的一个部分，而是内置于浏览器的一个API（接口），

为了操纵浏览器和文档，JavaScript使用分层的父对象和子对象（树形结构），这就称为DOM，表示一个Web文档的所有内容和组件。

DOM对象，有属性和方法。

①最顶层的是window对象，表示一整个浏览器接口，框架可以由window对象来表示。如alert就是window对象的方法

②document对象代表的是一个web文档或一个页面，web文档在浏览器窗口中显示，所以很明显document对象是window对象的子对象；

因此window.document和document都是指当前窗口；

如果使用多个窗口（或框架），就要使用多个window对象，每个window对象都有自己的document对象，因此使用其中某一个，要指出窗口名和对象名；

**（22）获得文档的信息**

①document.URL指明了文档的URL地址，这是一个简单的文本字段，不能改变它的属性（不懂，是指这个是一个string对象么？）；

如果要给用户一个不同的地址，要使用window.location对象；

②document.title列出了当前页面的标题，由HTML中的title标签定义。

如：

ti=document.title;

alert(ti);

效果是输出标题，注意，没有括号；

③document.referrer是用户所浏览的上一个页面的URL地址；

④document.lastModified是文档最后修改日期，这个日期从服务器传到页面（注：在360浏览器里，这个时间是变化的，在Firefox里，这个时间是固定的）但可能服务器上运行正常，需要验证；

⑤document.bgColor和document.fgColor是文档的背景和前景（文字）颜色，对应body下的BGCOLOR属性和TEXT属性；

例如：

<body bgcolor="green">

abc

<script>

ti=document.bgColor;

alert(ti);

</script>

</body>

输出是green，背景色变绿

⑥document.linkColor、document.alinkColor和document.vlinkColor是文档中链接的颜色。分别对应body标签中的LINK、ALINK和VLINK属性（但不知道这3个是什么意思）。

⑦document.cookie允许读取和设置一个文档的cookie。更多略。

**（23）history对象**

**一个可访问的属性：**

history.length，用户访问过的不同地址的数目。

如代码，表示当前访问过的地址的数目

<script>

alert(history.length);

</script>

初始是1，假如访问了一个其他网页，这里则弹出是2，如果访问更多，则相应增加。

**方法一：**

history.go(参数)

打开历史列表的一个网址，参数为正数或者负数。

history(-1)相当于后退一步。

**方法二：**

history.back()载入历史列表中的前一个网址，相当于按一下后退。

**方法三：**

history.forward()载入历史列表中的后一个网址，相当于按一下前进按钮。

**（24）AJAX**

传统情况下，JavaScript与服务器之间的通信方法只有一种——表单。

**1、前端/网页：**创建一个XMLHttpRequest对象，然后将其发送给Web服务器，并且无需一直等待请求，而是可以继续发送请求。

**2、服务器：**通过发送包含内容的文件（或服务器端应用程序的输出）作为相应。

**3、前端：**当接到服务器端的响应后，相应的JavaScript函数将被触发，以处理相关数据。

**4、DOM：**由于这项技术的目的在于实现更好的交互（否则接受到服务器信息就得刷新页面），因此脚本使用DOM来显示来自服务器的数据，从而无需再次刷新页面。

**其他：**可以同时处理多个请求（发送多个请求，服务器的速度不影响前端的）。

**AJAX的应用：**

主要用于创建Web程序——面向用户且基于Web的服务。

**使用XMLHttpRequest：（表示纯看概念很茫然啊）**

1、创建请求

创建一个XMLHttpRequest对象：

var ajaxreq = new XMLHttpRequest();

注：IE5/6除外（早版本的需要启用ActiveX控件），语法不同；

2、打开URL

ajaxreq.open("get", "filename");

可以使用GET或者POST命令（这里是get）；

由于GET需要使用将参数作为地址发送给服务器，因此如例子：

ajaxreq.open("get", "search.php?query=abc");

访问search.php网页，将abc作为查询参数发送给服务器。

问题：为什么是query？

3、发送请求

ajaxreq.send(null);

注：get方法用null，post方法用发送的数据。

4、等待响应：

XMLHttpRequest提供了专门的事件处理程序，用于处理服务器的返回信息。

ajaxreq.onreadystatechange = MyFunc;

请求（request）对象拥有一个名为readyState的属性，代表一个请求的当前状态。

当readyState属性发生变化时，该事件被触发。

ReadyState属性值的变化范围如下：

0 新的请求；

4 已结束的请求

所以事件处理程序会检测该属性，查看其值是否已变化为4。

由于可能发生错误，因此当请求成功时，status将被设置为200。但若发生错误，其值将被设置为对应的错误代码。

与错误相关的解释信息（成功时为OK），将被保存在statusText属性中。

5、解释响应数据

当readyState属性的值为4，而且请求已被响应时，可以通过另外两个属性进一步访问那些来自服务器的返回数据。

responseTeXt 用于访问以原始文本存在的响应信息；

responseXML 用于访问以XML对象形式存在的响应信息。

当数据格式不是XML时，只有文本属性可被访问。

这里的XML指的是.xml文件么？

如代码：

//AJAX库

var ajaxreq = false, ajaxCallback;

function ajaxRequest(filename)

{

try

{

//Firefox / IE7 / Others

ajaxreq = new XMLHttpRequest();

}

catch(error)

{

try

{

ajaxreq = new ActiveXObject("Microsoft.XMLHTTP");

}

catch(error)

{

return false;

}

}

ajaxreq.open("GET", filename);

ajaxreq.onreadystatechange = ajaxResponse;

ajaxreq.send(null); //发送请求

}

//事件处理程序

function ajaxResponse()

{

if(ajaxreq.readyState != 4) return ;

if(ajaxreq.status = 200)

{

//事件正常，请求成功

if(ajaxCallback)ajaxCallback();

}

else

{

alert("Request failed" + ajaxreq.statusText);

}

return true; //为什么return true？只要readyState等于4就一定成功？

}

**ajaxRequest()函数：**

实现了处理、创建、发送XMLHttpRequest对象所需的全部操作。

该函数首先会创建XMLHttpRequest对象。此时，对于不同的浏览器需要使用不同的命令。如果使用了错误的命令就一定会出现错误。

所以需要使用try和catch语句。

先试着使用标准方式，出错了使用另一种方式（这里是针对IE5和IE6的），如果还不行，那么返回一个false值给ajaxreq，然后通过条件判断，告诉用户浏览器不支持AJAX功能。

**ajaxResponse()函数：**

ajaxResponse()函数是onreadystatechange事件的处理程序。

该函数首先检查readyState属性，查看其值是否为4。如果不是，什么都不做（return）

如果是4，那么检查属性值是否为200（是200说明成功），是的话，执行保存于变量ajaxCallback中的处理函数。

如果不是200，说明出错了，alert弹框提示出错信息。

**使用AJAX库：**

①将库文件保存为ajax.js，并使其与HTML和脚本位于同一目录中。

②利用<script src="ajax.js">导入HTML中。

③为请求结束事件创建一个处理函数，并将变量ajaxCallback的值设定为该函数。

④调用ajaxRequest()函数，其参数为服务器端应用程序（或文件）的文件名。（本库只支持GET，因此不考虑POST，也无需指定使用方式）。

⑤如果请求操作正常结束，那么将调用刚才在ajaxCallback中指定的函数。同时全局变量ajaxreq将会把服务器的响应数据分别保存在responseXML和responseText属性中。

**使用AJAX库编写AJAX测验程序：**

在创建了一个可重用的AJAX库后，就可以用他来创建JS程序。

JS代码例子如下；

var qn = 0; //qn是问题编号

function getQuestions()

{

qn = 0; //点击开始按钮后，重置问题

obj = document.getElementById("question"); //obj是question的（HTML）

//之所以设置下面这行，是因为在异步加载成功前显示这里

obj.firstChild.nodeValue="(please wait)"; //obj的所有子节点中的第一个子节点的值是please wait。作用是将其写到<span id="question">...</span>中

ajaxCallback = nextQuestion; //将函数赋给回调函数（ajaxCallback），在ajax里调用

ajaxRequest("questions.xml"); //调用ajax库加载questions.xml文件

}

//作用是读取、并输出问题

function nextQuestion()

{

questions = ajaxreq.responseXML.getElementsByTagName("q"); //questions被赋值为AJAX返回的XML文件中标签为q的（在XML中）

obj = document.getElementById("question"); //obj为标签是question（在AJAX.html中）

if( qn < questions.length) //如果qn小于questions的长度（即未到结尾）

{

q = questions[qn].firstChild.nodeValue; //q是读取的值

obj.firstChild.nodeValue = q; //将其输入到HTML中显示

}

else

{

obj.firstChild.nodeValue = "(no more questions)"; //如果没有更多问题了，那么提示用户

}

}

function checkAnswer()

{

answers = ajaxreq.responseXML.getElementsByTagName("a"); //answers是一个数组（因为有多个<a>标签）

a = answers[qn].firstChild.nodeValue; //firstChild.nodeValue指所有子节点（childNodes）中的第一个子节点的文本值

answerfield = document.getElementById("answer"); //HTML输入位置的内容（id是answer）的值

if(a == answerfield.value) //如果读取的值（XML中的），和输入的值（HTML用户输入的）一样

{

alert("Correct!"); //弹框提示正确

}

else

{

alert("Incorrect. The correct answer is: " + a); //弹框提示错误，并告知正确的答案

}

qn += 1; //问题编号+1

answerfield.value=""; //重置答案

nextQuestion(); //调用函数

}

obj = document.getElementById("startq"); //ojb是startq按钮

obj.onclick = getQuestions; //点击后调用函数，不用()表示不执行，只有当点击后才执行

ans = document.getElementById("submit"); //ans是提交按钮

ans.onclick = checkAnswer; //点击后调用函数

另有一XML文件：

<?xml version="1.0" ?>

<questions>

<q>a</q>

<a>1</a>

<q>b</q>

<a>2</a>

<q>c</q>

<a>3</a>

<q>d</q>

<a>4</a>

<q>e</q>

<a>5</a>

</questions>

最后是html文件：

<html>

<head>

<title>AJAX测试</title>

<script src="ajax.js"></script>

</head>

<body>

<h1>AJAX的测试例子</h1>

<form>

<p>

<b>问题：</b>

<span id="question">...</span>

</p>

<p>

<b>答案：</b>

<input type="text" name="answer" id="answer">

<input type="button" value="Submit" id="submit">

</p>

<input type="button" value="Start the Quiz" id="startq">

</form>

<script src="quiz.js"></script>

</body>

</html>

效果是：

①点击Start the Quiz按钮后，读取XML文件，将读取到的问题输出在html上。

②答案框可以用于输入答案，按submit按钮提交答案。

③提交后进行验证，如果和预留答案相符，提示，不符，提示正确答案。

④然后变更到下一个问题。

**（25）name**

name的作用和id类似。

但还有一些其他用处：

* 用途1: 作为可与服务器交互数据的HTML元素的服务器端的标示，比如input、select、textarea、和button等。我们可以在服务器端根据其Name通过Request.Params取得元素提交的值。
* 用途2: HTML元素Input type='radio'分组，我们知道radio button控件在同一个分组类，check操作是mutex的，同一时间只能选中一个radio，这个分组就是根据相同的Name属性来实现的。
* 用途3: 建立页面中的锚点，我们知道<a href="URL">link</a>是获得一个页面超级链接，如果不用href属性，而改用Name，如：<a name="PageBottom"></a>，我们就获得了一个页面锚点。
* 用途4: 作为对象的Identity，如Applet、Object、Embed等元素。比如在Applet对象实例中，我们将使用其Name来引用该对象。
* 用途5: 在IMG元素和MAP元素之间关联的时候，如果要定义IMG的热点区域，需要使用其属性usemap，使usemap="#name"(被关联的MAP元素的Name)。
* 用途6: 某些特定元素的属性，如attribute，meta和param。例如为Object定义参数<PARAM NAME = "appletParameter" VALUE = "value">或Meta中<META NAME = "Author" CONTENT = "Dave Raggett">。

**（26）验证输入内容是不是空，是不是数字**

field为检查所内容（input）的id

function checkIsNumber(field) //检查输入的文字是不是数字，这的field是id的值

{

with(field)

{

if(value==null||value=="")

{

document.getElementById(field).innerText="<b>请输入内容</b>";

}

else {

if(value>0)

{

return false;

}

else {

document.getElementById(field).innerText="<b>请输入数字</b>";

}

}

}

return true;

}

**（27）下拉列表**

<form>

<select name="cars">

<option value="volvo">Volvo</option>

<option value="saab">Saab</option>

<option value="fiat">Fiat</option>

<option value="audi">Audi</option>

</select>

</form>

如果带默认选择（即默认不是第一个），加上下面红色部分（即默认选择这个）

<select name="cars">

<option value="1">1</option>

<option value="2">2</option>

<option value="3" selected="selected">3</option>

<option value="4">4</option>

</select>

**（28）div、span、ul的区别**

①div是一个块，占用的位置是一行，不同div之间自带换行。

例如：fff<div>www</div><div>ppp</div>

其显示是：

fff

www

ppp

②span是行内小块，是不带换行的。

例如：fff<span>www</span><span>ppp</span>

其显示是：fffwwwppp

③ul目前已知是自带缩进的。

**（29）把json转化为字符串**

JSON.stringify(store)

即可（注意JSON要大写），store是json对象

**（30）对一个数组赋值**

①假如我们有一个不定长度的数组abc

②我们想要将该数组逐值赋给另一个数组def；

③因此，机智的我们想到了使用for循环；

for(var i = 0; i < abc.length; i++)

{

def[i] = abc[i];

}

④然而，这样是无法赋值的（原因在于def之前未定义，因此不能直接把他当做一个数组）；

⑤那么，在for循环之间加入var def = new Array()即可

代码：

var def = new Array()

for(var i = 0; i < abc.length; i++)

{

def[i] = abc[i];

}

**（31）截取部分字符串**

**x = 字符串.substr(起始字符编号, 返回值的长度);**

假如我们有一个字符串abc。

①我们想要获得第1个字符，这么写：x = "abc".substr(0,1);

②假如我们想要获得第1，2个字符，这么写：x = "abc".substr(0,2);

③假如我们想要获得第2，3个字符，这么写：x = "abc".substr(1,2);

④假如我们想要获得第2个字符之后所有字符，这么写：x = "abc".substr(1);

⑤假设我们要获得第5个字符（字符串长度至少需要5），这么写：x= 字符串. substr(4,1);

即：

①第一个参数决定起始字符，需要是编号（第一个字符编号0），不能使用字符"a"，来假设从字母开始；

②第二个参数决定返回的字符串的长度，如果是剩下所有，就省略不写。

**（32）遍历一个对象的方法**

在javascript，都是一个对象（object），比如函数、比如变量。

比如var m=1; 此时m是一个对象；

var n = function(){ alert("1"): } 此时n也是一个对象（函数对象）；

对象可以理解是一个类，有变量有函数。

假如我们想要知道一个对象的所有函数、变量。那么就需要遍历他。

假设这个对象是obj，遍历他的方法是：

for ( var i in obj)

{

console.log(i);

}

那么在浏览器的调试器之中，就会出现我们想知道的东西。

比如说，ojb是这样的：

var obj = new Object; //注意，这个代码不能缺少

obj.a = 1;

obj.b = 2;

obj.c = function()

{

alert("1");

}

那么for in遍历他的结果是：

a

b

c

undefined

这告诉我们，obj有三个方法，分别是a、b、c（即ojb.a等）；

**（33）定时执行函数与延迟执行函数**

假设目标函数为：

function test(){

console.log("1");

}

定时执行函数：

window.setInterval(test, 2000); //2000毫秒执行一次

window.setTimeout(test, 2000); //延迟2秒执行函数

移除定时器：

前提是定时器赋值给某个变量，假如是A

清除setTimeoue的：clearTimeout(A)

清除setInterval的：clearInterval(A)

**（34）将输入框数字，每3位用逗号分隔**

输入框的值为this.value

var val = this.value.toString();  
var length = val.length;  
for (var i = 1; i < length / 3; i++) {  
 var temp = val.slice(0, -4 \* i + 1);  
 val = val.replace(temp, temp + ',');  
}  
this.value = val;

<**input** type="text"   
 onKeypress="return (/[\d]/.test(String.fromCharCode(event.**keyCode**)))"  
 onfocus="this.**value**=this.**value**.replace(/,/gi,'')" />

onKeypress只允许输入数字，如果允许输入句号的话，在[\d]改为[\d.]即可

onfocus是将上面分隔用的逗号，删除掉，只显示数字内容

**（35）查看对象是否有某个属性（转）**

来源：

http://www.cnblogs.com/snandy/archive/2011/03/04/1970162.html

内容：

两种方式，但稍有区别

**1，in 运算符**

|  |  |
| --- | --- |
| 1  2  3 | var obj = {name:'jack'};  alert('name' in obj); // --> true  alert('toString' in obj); // --> true |

可看到无论是name，还是原形链上的toString，都能检测到返回true。

**2，hasOwnProperty 方法**

|  |  |
| --- | --- |
| 1  2  3 | var obj = {name:'jack'};  obj.hasOwnProperty('name'); // --> true  obj.hasOwnProperty('toString'); // --> false |

原型链上继承过来的属性无法通过hasOwnProperty检测到，返回false。  
  
需注意的是，虽然in能检测到原型链的属性，但for in通常却不行。

当然重写原型后for in在IE9/Firefox/Safari/Chrome/Opera下是可见的。见：[for in的缺陷](http://www.cnblogs.com/snandy/archive/2011/03/03/1970132.html)

<http://www.cnblogs.com/snandy/archive/2011/03/03/1970132.html>

**（36）合并两个数组**

①将一个对象放到数组里，是arr.push(obj)

②将两个数组合并起来是：

var arr3 = arr1.concat(arr2);

返回值是合并好的数组

**（37）只需要前端知识，就能理解的ajax教程**

①新人学web前端时，遇见的第一个困难就是ajax

【1】到底什么是ajax？

【2】ajax时发生了什么事情？

【3】为什么ajax时要有url？

【4】为什么ajax时，有时候会有一个对象（object），有时候没有，有时候url后面有?或者&或者=或者其他什么？

【5】我怎么知道服务器（后端）返回什么内容？我如何处理？

对于以上问题，我不深入，只用最简单的话讲清楚什么是ajax，并且让你会用ajax。

②**ajax是什么？**

用通俗的话一步一步来解释：

【1】我决定去你家拜访，首先得知道你家在哪吧。你家的地址就是**http的url**；

【2】我到你家来做客，这是进行**http访问**（访问你家）；

【3】你给我端茶倒水，这是你给我返回了一些资源，通常是一个**网页（html）**；

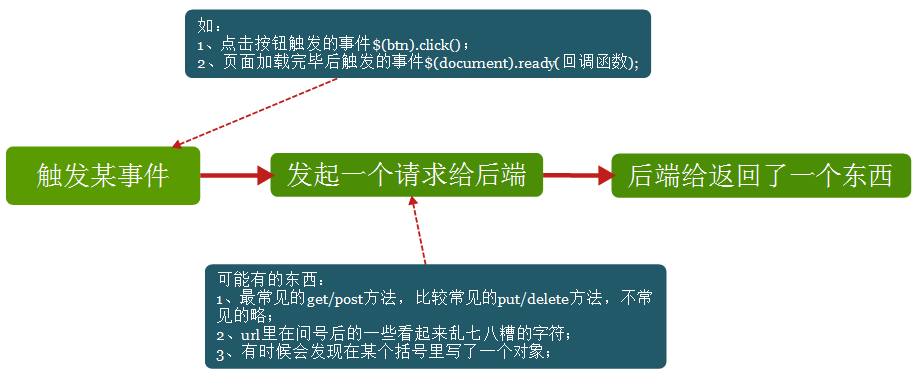
【4】你在你家客厅摆放了一些吃的，并且我知道这些东西我可以吃。我知道可以吃，然后我就告诉你我想吃点东西，这是我再次发起请求；

【5】你让我不用动，然后给我拿了一点糕点过来。此时，我依然坐在你家的客厅沙发上，我只告诉你我要吃东西，你就给我了。

【4】和【5】这两步结合起来，这就是**ajax**；

ajax的实质是，在原有页面的基础上，发生某个js事件，然后执行了一个请求，服务器给予了一个回应。

如图：



如果说的再简单一点，就是我（网页）问服务器（后端）要东西，然后服务器给我了。

至于怎么要，下面说。

③**ajax时发生了什么事情？**

首先请看上图。

【1】触发某事件，实际上这里是指触发ajax事件。例如jquery里get请求事件：

$.get(*URL*,*callback*);

或者是一个post请求事件

$.post(URL,data,callback);

【2】发起请求后，就是执行以上事件。

这个事件必须的参数是URL，可选参数是data（get没有）和callback（回调函数，请求完成后执行的）；

关键是URL

**理解URL：**

大部分新人之所以不明白ajax，主要是不明白这个url。

按照普通人的理解，url就是一个地址，比如一个文件a.txt他的url可以是C:\a.txt

当我访问c:\a.txt时，就访问的是这个文本文件。

这样理解当然也不算错误，但不全面。

按照我的解释，你可以把url理解为一个**暗号**，和**服务器（后端）**预先约定好的暗号。

注意，这里的url，通常指的是例如：

https://www.zhihu.com/topic/19626558/hot

中的

/topic/19626558/hot

为什么这么说呢？很简单。

你发送的url，事实上在后端看来，只是一个字符串，在未进行特殊处理的情况下，他并不能直接访问文件。

就比如上面我写的那个知乎的url，可能存在一个文件名是hot这样的文件么？显然并不可能。

事实上，不存在一个根目录下的topic文件夹，也不存在一个名为19626558的文件夹。

但为什么时候我们可以通过访问这个url来访问一个网页呢？事实上，是因为在http服务器进行处理过的了。

这里我简单的说一下其流程：

【1】我们前端和后端约定了一个url（暗号），这个暗号以'/topic'开头，但这个暗号表示什么意思呢？

【2】当后端遇见一个以这样暗号为开头的，他就知道，我是在访问一个话题类的页面，

但目前为止，后端还不知道话题是什么，自然也没法给我们返回内容；

如果我们不是以/topic为开头，而且以其他字符串为开头，自然也会有对应的代码来处理我们的请求（包括无效的字符串，也是有办法处理的）

【3】然后我们继续发送了下一部分'/19626558'，这个实际上就是话题的id。

当后端获取到这个话题的id后，他会去数据库里进行查询。

如果这是一个有效的话题id，那么会返回给我们一大堆跟这个id相关的内容；

假如后端发现这个id是无效的，那么会告诉我们



注意这个页面是预制好的。

【5】然后后端会查看url（暗号）的下一部分'/hot '。服务器发现是'/hot'，于是他按照热门程度，将内容排序好后返回给我们，避免我们看到的都是没人回复的帖子（那也太浪费感情了）。

如果下一部分是'/newest '呢，那就以时间发表顺序来返回内容给我们（毕竟有时候我们只想看看最新的）；

那么假如什么都没有呢？是个空字符串的话，该怎么办？后端也有预先准备的办法，那就是默认给我们发送'/hot'的内容，并且将我们的url设置为'/hot'结尾的。

如果是其他结尾的，那么就可能返回一个404的页面（就像上面那样）

【6】按道理来说，讲解就到这里结束了。但是为了防止理解的过于片面，我假设一种情况。

比如说有一个调皮的程序员，他写了这么一段代码，假如你访问的url是一个/topic/19626558/joke（显然知乎并不会这么做），

那么他会将你转到一个小游戏的页面（他显然跟topic毫无关系，跟我们要访问的url也毫无关系）。

因此要搞清楚的是，**url**，只是我们约定好的一种形式，我通常会照着这个通俗的做法去做，但是我也可以任性的做些别的，比如/a/b/c页面却是/a/b页面的父页面（通俗做法显然是相反的）；/a/b页面和/c页面是平级页面。

所以我说与其将url理解为一个地址，不如理解为一个暗号，一个你与后端程序员约定好的暗号。

你告诉他，当我说这个暗号的时候，我可能给你发送些什么东西（比如id，密码之类），你要给我返回些什么东西，比如一个页面、一个表格、一段动画这样。

④**为什么要有url？**

就像上面说的那样，他是一个暗号，你们必须有同样的暗号时，他才知道你想做什么；

例如/game时是想访问游戏页面，/news时是想访问新闻页面，/login是想登陆，/logout是想登出。

至于究竟是想访问页面呢，还是想登陆登出呢，或者是想获取一段数据呢？这要根据url（约定的暗号）来决定。

⑤**为什么ajax时，有时候会有一个对象（object），有时候没有，有时候url后面有?或者&或者=或者其他什么？**

这个问题事实上就涉及到了http请求的方法了。

我们通常用的http请求方法有两种，分别是get和post。

他们的区别简单的来说，就是一个会附带一个对象（post），另外一个是不带的（get）。

先说get和post的区别：

【1】get，如字面意思，请求，获取。

我（前端）告诉你（后端），我想获取什么东西（根据暗号事先约定好的），你当我说这个暗号（发送http请求）的时候，就返回给我；

【2】post，如字面意思，邮寄、发送。

假如我在玩游戏，我要下线/退出了，因此我要存档的对吧。你可以把存档这个行为，理解为post。我存档时，肯定要把我的经验、等级、装备、任务进度之类之类的内容，存到文件中去（在后端，一般是存到数据库之中）；

但后端怎么知道我的这些数据呢？一般就是通过post了，例如我发送这样一个对象：

{lv:100, name:"水哥", exp:987654321, atk:12345, def:12345, gold:10000}

后端就会获取到这个对象，然后他就会把这个对象解析出来，存到数据库之中（比如MySQL）。

get一般在无敏感数据、数据量小的时候使用；

post一般在有敏感数据（比如密码），数据量大的时候使用；

因此，**“为什么有时候会有这个对象”**，这个问题就解释清楚啦。

另外一个，**url里为什么有时候会有很长串的字符呢？**

比如说：

/friend?id=123&online=true#abc

我将其分为三部分：

【1】问号以前的：/friend

【2】#号以前的：id=123&online=true

【3】#号以后的：abc

第一部分属于url，即我们约定的暗号，称为pathname

第二部分和第三部分都是请求部分，但他们有所区别：

仔细看第二部分，他们以&分隔，分别为id=123与online=true，这是典型的KV结构（即key和value）。而&符号通常表示and。因此我们可以将其理解为一个对象：{id:123, online:true} ——具体怎么转换是后端处理的事情，前端只需要这么理解就可以了。

第三部分是一个字符串abc，我们将其称为哈希地址。

他们有什么用呢？很简单。

【1】假设我在玩网页游戏，我需要获取在线的好友列表，因此以/friend开头，表示我要获取好友列表。

【2】假设我的id是123，因此我发送id=123（表示要找我的好友），但我的好友里面有在线的，也有非在线的，服务器怎么知道我要获取哪种呢？那么就通过检查online的值，发现其值为true，于是就知道我找的是在线好友了。

【3】另外还有一个哈希地址。我可以这么解释，我需要查看的是好友名称为abc的人的资料。因此，服务器返回的是一个在线的好友列表，以及好友名称为abc的人的全部资料。

（ps：具体怎么解释，全看约定）

⑥**我怎么知道服务器（后端）返回什么内容？我如何处理？**

还记不记得之前jquery的两个方法，他们都有一个回调函数？

这个回调函数的参数呢，就是返回的内容。

例如：

$.get('/abc', function(item){

//do something

})

假设后端返回一个这样的对象：

{ id:1, name:"abc"}

那么回调函数的参数item的值，就是{ id:1, name:"abc"}

至于怎么知道的，想要深入的话可以去看源代码，只是了解ajax的话，知道这个事实就行了。

那么这个函数什么时候执行呢？

是在服务器返回内容之后，浏览器接受到的时候执行的。

**Notes：**

【1】假设我在0s的时候发送了这个get请求，

【2】在1s的时候，服务器接受到这个请求，他用了0.1s来处理，在1.1s的时候，给我返回以上那个对象；

【3】又过了0.5s，我接受到了这个返回对象（此时是1.6s），因此这个回调函数执行的时候，是在我发送get请求后1.6s执行的。

**为什么要特别强调这个过程，原因在于：**

当我们用ajax来给一个变量赋值时，我们不能在ajax外对赋值结束后的这个变量进行处理。

例如：

var id = null;

$.get('/abc', function(item){

id = item.id;

})

console.log(id);

你觉得console.log出来的id的值是什么？

答案：是null

原因在于，console.log(id)这段代码，在上面第【1】步执行结束后执行的。因此他的执行时间，实际上可能是第0.01s（第0s发送的get请求）

而等他获取值的时候，已经在第1.6s了，因此，第0.01时他的值是初始值（null）

这是新人很容易犯的一个错误，务必提醒自己，多多注意。

⑦补充：

对于ajax，纯前端是无法使用的，因为ajax的实质就是前后端交互，建议大家看一看我的nodejs的博客，架设一个最简单的nodejs服务器，来进行一次简单的前后端交互，对于提高自己的ajax水平会有很大的帮助。

**（38）阻止默认右键弹窗**

①在点击事件的处理函数之中，默认的右键弹窗的出现是在 **鼠标弹起时** 出现的

②事件名为：

document.oncontextmenu = function(evt){

//这里是函数

return true;

}

当返回值为true时，将弹出右键菜单；

当返回值为false时，右键菜单将不会被弹出；

注意：这是设置后永久生效的（即当返回false时，右键点击其他地方也不会出现鼠标右键的下拉菜单）；

③只对当前区域有效：

原理：

当前区域的mouseup事件时，设置return false；然后设置延迟执行函数（setTimeout），延迟0.01秒执行一个函数，将返回值再设置为true即可。

例如：

$("#test").mouseup(function(){

document.oncontextmenu = function(evt){

return false;

}

setTimeout(function(){

ocument.oncontextmenu = function(evt){

return true;

}

},10}

});

这样的话，就类似于起到只点id为test的区域时不会弹出右键菜单，但点击其他区域时会弹出。

**（39）汉字和英文字母长度计算**

for (var i = 0; i < text.**length**; i++) {  
 if (text[i].match(/[^\x00-\xff]/ig) != null)  
 length += 2;  
 else  
 length += 1;  
}

英文字母为1，中文汉字为2

**（40）几种弹窗**

**①alert**

常见，不解释

**②confirm(“弹窗内容”)**

弹出一个对话框，有“确认”和“取消”两种选择，返回值为true或者false；

例如：



**③prompt(“弹窗内容”)**

带输入框的弹窗，返回值是输入框的内容（点确定），null（点取消）

**（41）字符串匹配**

str.search(/(字符串1|字符串2|……)/) > -1

因为匹配不到的时候值为-1，因此假如能匹配到，那么值就应该大于1，以上代码可以作为if的判断条件使用

**（42）如何让用户创建一个对象的实例时，相当于自动new一个实例**

内容来源于：

<http://www.cnblogs.com/TomXu/archive/2012/02/21/2352994.html>

假如有构造函数：

function test(a, b) {  
 if (!(this instanceof test)) { //假如this不是test的实例  
 return new test(a,b);  
 }  
 this.**a** = a;  
 this.**b** = b;  
}  
var m = test(1,2);  
console.log(m);

这个的关键在于this

①当用户像下面那样直接调用一个test(1,2)时，相当于window.test(1,2)，此时，this是window，而window必然不是test的实例；

②当用户是通过new来创建一个test的实例时new test(1,2)；此时this指向的是当前test函数，因此必然是test的实例；

当处于情况①时，if的最终结果是true，因此触发new test(a,b)，并将其返回给变量，相当于用户var m = new test(1,2);

当处于情况②时，可能是用户自己new了一个test，或者是触发了new test()这个方法，无论是哪个，if的最终结果是false，因此跳过判断语句，相当于用户正常new了一个test的实例。因此不用怕用户忘记写new这个方法。

**（43）深度克隆函数**

var objectClone = function (obj, preventName) {  
 if ((typeof obj) == 'object') { //假如是个对象  
 var res = (!obj.sort) ? {} : []; //如果有sort说明是数组，否则是对象，并赋res为对应的空数组/对象  
 for (var i in obj) { //for in循环遍历复制，对数组和对象都有效  
 if (i != preventName) //如果i不是禁止复制的方法名  
 res[i] = objectClone(obj[i], preventName); //递归执行  
 }  
 return res;  
 }  
 return obj; //否则返回复制好的对象（如果不是对象，则因为按值传递，因此返回的还是一个赋值后的）  
}

可以深度、递归复制一个对象、数组、普通类型的变量。

返回值是复制好的内容

注；

①函数可以浅克隆；

②数组和对象必须深克隆；

**（44）图片预览FileReader**

①核心部分为：

<**script**>  
 var result = document.getElementById("result");  
 var input = document.getElementById("file\_input");  
 function readFile() { //传一个dom给该函数  
 var file = this.files[0]; //上传后，抓取图片数据（在files属性中）  
 if (!/image\/\w+/.test(file.**type**)) { //必须type是图片才可以  
 alert("文件必须为图片！");  
 return false;  
 }  
 var reader = new FileReader(); //创建一个新的FileReader对象  
 reader.readAsDataURL(file); //然后用其方法来读取图片  
 reader.**onload** = function (e) { //读取完毕时，调用此函数  
 result.innerHTML = '<img src="' + this.**result** + '" alt=""/>' //将图片加到该dom之中（该dom只能有这个图片一个内容）  
 }  
 }  
 if (typeof FileReader === 'undefined') { //如果不支持则提示  
 result.innerHTML = "抱歉，你的浏览器不支持 FileReader";  
 input.setAttribute('disabled', 'disabled');  
 } else {  
 input.addEventListener('change', readFile, false); //监视change事件，调用readFile函数，在冒泡阶段执行  
 }  
</**script**>

解释：

判断能否使用，不能预览则报错；

能预览，则监视文件上传的输入框，触发change事件时，则调用预览函数。

读取图片，将图片以img形式显示。

②完整代码（依赖jquery）

<!DOCTYPE HTML>  
<**html**>  
<**head**>  
 <**script** type="text/javascript" src="jq.js"></**script**>  
</**head**>  
<**body**>  
<**p**>  
 <**label**>请选择一个图像文件：</**label**>  
 <**input** type="file" id="file\_input"/>  
</**p**>  
<**div** id="result">  
 <**img** src="./1.jpg"/>  
</**div**>  
<**script**>  
 var result = document.getElementById("result");  
 var input = document.getElementById("file\_input");  
 function readFile() { //传一个dom给该函数  
 var file = this.files[0]; //上传后，抓取图片数据（在files属性中）  
 if (!/image\/\w+/.test(file.**type**)) { //必须type是图片才可以  
 alert("文件必须为图片！");  
 return false;  
 }  
 var reader = new FileReader(); //创建一个新的FileReader对象  
 reader.readAsDataURL(file); //然后用其方法来读取图片  
 reader.**onload** = function (e) { //读取完毕时，调用此函数  
 result.innerHTML = '<img src="' + this.**result** + '" alt=""/>' //将图片加到该dom之中（该dom只能有这个图片一个内容）  
 }  
 }  
 if (typeof FileReader === 'undefined') { //如果不支持则提示  
 result.innerHTML = "抱歉，你的浏览器不支持 FileReader";  
 input.setAttribute('disabled', 'disabled');  
 } else {  
 input.addEventListener('change', readFile, false); //监视change事件，调用readFile函数，在冒泡阶段执行  
 }  
</**script**>  
<**style**>  
</**style**>  
</**body**>  
</**html**>

**（45）开启摄像头**

支持浏览器：chrome，360安全浏览器

代码如下：

<**html**>  
<**head**>  
 <**meta** charset="UTF-8">  
 <**title**>Document</**title**>  
</**head**>  
<**body**>  
<**video** width="640" height="480" autoplay></**video**>  
  
<**script**>  
 var video = **document**.querySelector('video');  
  
 **navigator**.webkitGetUserMedia({  
 **video**: true  
 }, success, error);  
  
 function success(stream) {  
 video.**src** = **window**.webkitURL.createObjectURL(stream);  
 video.play();  
 }  
  
 function error(err) {  
 alert('video error: ' + err)  
 }  
</**script**>  
</**body**>  
</**html**>

需要点击允许开启摄像头。

进阶内容如下：

<http://www.cnblogs.com/vajoy/p/5656439.html#!comments>

**step1. 调用摄像头**

**step2. 捕获 video 帧画面**

**step3. 对捕获的两个帧画面执行差异混合**

**step4. 上报异常图片**

**（46）替代jQuery常用功能的原生JS代码**

①addEventListener

|  |  |
| --- | --- |
| 命令 | addEventListener |
| 效果 | 添加响应事件 |
| 兼容性 | IE9及以上  <http://caniuse.com/#search=addEventListener> |
| 注： | 同一个回调函数，多次添加不会重复触发 |

范例：

<**div** id="test">  
 abc  
</**div**>  
<**script**>  
 var node = document.getElementById("test");  
 console.log(node);  
 node.addEventListener("click", function () {  
 alert("click!");  
 })  
</**script**>

②DOMContentLoaded

|  |  |
| --- | --- |
| 命令 | DOMContentLoaded |
| 效果 | 页面加载完成后触发本事件 |
| 兼容性 | IE9及以上  http://caniuse.com/#search=DOMContentLoaded |

范例：

<**script**>  
 document.addEventListener("DOMContentLoaded", function () {  
 var node = document.getElementById("test");  
 console.log(node);  
 })  
</**script**>  
<**div** id="test">  
 abcd  
</**div**>

注：

假如不加这个事件，那么输出结果是null

③querySelector、querySelectorAll 选择器

|  |  |
| --- | --- |
| 命令 | querySelector（单个，遇见的第一个）；  querySelectorAll（全部符合的） |
| 效果 | DOM元素选择器，类似jQuery的选择器 |
| 兼容性 | IE9及以上，IE8只有部分支持  <http://caniuse.com/#search=DOMContentLoaded> |
| 关于IE8 | IE8的支持，被限制在CSS2.1和CSS3的一小部分。另外，在IOS8.x里会有bug（id然后加元素会失效）。  具体可以参见上面url里的说明 |

范例：

<**body**>  
<**script**>  
 document.addEventListener("DOMContentLoaded", function () {  
 var node = document.querySelector('.container').querySelector('.test');  
 console.log(node);  
 var node = document.querySelector('body').querySelectorAll('.test');  
 console.log(node);  
 })  
</**script**>  
<**div** class="test">  
 abcd  
</**div**>  
<**div** class="container">  
 <**div** class="test">  
 defg  
 </**div**>  
</**div**>  
</**body**>

注：

【1】可以选择某一个DOM元素下的子元素（如第二个）；

【2】不加All的是选择第一个，返回结果是单个DOM元素；加了All返回的是一个数组（即使只有一个）；

④removeEventListener 移除事件

|  |  |
| --- | --- |
| 命令 | removeEventListener |
| 效果 | 移除响应事件 |
| 兼容性 | 没有查到，但推测和addEventListener相同 |
| 注： | 同一个回调函数，多次添加不会重复触发；  删除已删除事件也不会报错。 |

<**script**>  
 document.addEventListener("DOMContentLoaded", function () {  
 var event = function () {  
 alert("event happened!");  
 }  
 var node = document.querySelector('.test');  
 var add = document.querySelector('.add');  
 var remove = document.querySelector('.remove');  
 add.addEventListener("click", function () {  
 node.addEventListener("click", event);  
 })  
 remove.addEventListener("click", function () {  
 node.removeEventListener("click", event);  
 })  
 })  
</**script**>  
<**div** class="test">  
 事件触发器（点击触发）  
</**div**>  
<**button** class="add">添加事件</**button**>  
<**button** class="remove">移除事件</**button**>

⑤classList 类列表

|  |  |
| --- | --- |
| 命令 | DOM.classList |
| 效果 | 类列表 |
| 兼容性 | IE10及以上  http://caniuse.com/#search=classList |
| 注： | 结果是一个对象，但可以像数组那样获取第N个类名； |
| 常用方法 | 单独获取所有类名（以字符串形式）是classList.value  添加类名是classList.add(类名)  移除类名是classList.remove(类名)  查看是否存在某个类classList.contains(类名)  切换是否显示某个类classList.toggle(类名) |

范例：

<**script**>  
 document.addEventListener("DOMContentLoaded", function () {  
 var node = document.querySelector('.test');  
 var list = node.**classList**;  
 console.log(list);  
 var add = document.querySelector('.add');  
 var remove = document.querySelector('.remove');  
 add.addEventListener("click", function () {  
 list.add("anotherClass")  
 })  
 remove.addEventListener("click", function () {  
 list.remove("anotherClass")  
 })  
 })  
</**script**>  
<**div** class="test">  
 事件触发器（点击触发）  
</**div**>  
<**button** class="add">添加</**button**>  
<**button** class="remove">移除</**button**>

⑥textContent和innerHTML

|  |  |
| --- | --- |
| 命令 | textContent、innerHTML |
| 效果 | 前者查看文本内容；  后者查看html内容 |
| 兼容性 | 前者IE9及以上；  后者IE8及以上  http://caniuse.com/#search=textContent  http://caniuse.com/#search=innerHTML |

范例：

<**script**>  
 document.addEventListener("DOMContentLoaded", function () {  
 var node = document.querySelector('.test');  
 var add = document.querySelector('.add');  
 var remove = document.querySelector('.remove');  
 add.addEventListener("click", function () {  
 console.log(node.**textContent**);  
 })  
 remove.addEventListener("click", function () {  
 console.log(node.innerHTML);  
 })  
 })  
</**script**>  
<**div** class="test">  
 <**span**>事件触发器（点击触发）</**span**>  
</**div**>  
<**button** class="add">查看textContent内容</**button**>  
<**button** class="remove">查看innerHTML内容</**button**>

注：

可以直接对其属性进行修改；但修改textContent属性，如果内容有html标签，不会被解释为html标签；

但通过innerHTML修改的html标签会被解释。

**（47）apply和call**

这个之前没研究过，今天简单的看了一下。

|  |  |  |
| --- | --- | --- |
| 命令 | 简单介绍 | 例子 |
| apply | 将某个函数提供给另外一个对象使用，类似继承 | 提供的方法.apply(获取方法的对象, 参数); |
| call | 改变函数中this指向的对象 | 方法.call(被this指向的目标, 传给方法的参数). |

共同点：两个都是函数对象的方法。

**call的说明：（略微修改自他人的博客，博客链接下附）**

<!doctype html>  
<**html**>  
<**head**>  
</**head**>  
<**body**>  
<**input** type="text" id="idTxt" value="input text">  
<**script** type="text/javascript">  
 var value = "global var";  
 function mFunc() {  
 this.**value** = "member var";  
 }  
 function gFunc() {  
 alert(this.**value**);  
 }  
 window.gFunc(); // 全局的值  
 gFunc.call(window); // 全局的值，this指向参数  
 gFunc.call(new mFunc()); // this指向mFunc  
 gFunc.call(document.getElementById('idTxt')); // this指向这个dom结点  
</**script**>  
  
<**script** language="javascript">  
 var func = new function () {  
 this.**a** = "func";  
 }  
  
 var func2 = function (x, y) {  
 var a = "func3";  
 alert(this.**a**);  
 alert(x);  
 alert(y);  
 }  
  
 func2.call(func, "func2", 'func3'); //this指向func这个函数，x是传递的参数  
</**script**>  
</**body**>  
</**html**>

关于用call来进行继承（同样改自某博客的例子）

<!doctype html>  
<**html**>  
<**head**>  
</**head**>  
<**body**>  
<**script** type="text/javascript">  
 function baseA() // base Class A  
 {  
 this.**member** = "baseA member";  
 this.showSelfA = function()  
 {  
 window.alert(this.member);  
 }  
 }  
  
 function baseB() // base Class B  
 {  
 this.**member** = "baseB member";  
 this.showSelfB = function()  
 {  
 window.alert(this.member);  
 }  
 }  
  
 function extendAB() // Inherit Class from A and B  
 {  
 baseA.call(this); // call for A  
 baseB.call(this); // call for B  
 // this指向的是extentAB这个方法，这里相当于将上面两个方法都执行了一遍，  
 // 由于后调用的baseB，而两个都有member属性，因此后调用的baseB的member覆盖了baseA的member，  
 // 但showSelfA和showSelfB没有冲突，因此两个方法都在  
 // 这种写法相当于形成了一个继承的形式。  
 }  
  
 window.onload = function()  
 {  
 var extend = new extendAB();  
 extend.showSelfA(); // show A  
 extend.showSelfB(); // show B  
 }  
</**script**>  
</**body**>  
</**html**>

博客链接：

<http://blog.csdn.net/sunboy_2050/article/details/6592082>

apply可以参照这个（我之前写的）：

apply是函数对象（function Function(){}）的prototype方法中的一个方法

可以通过**console.dir(Function)**来打印全部方法

又因为所有函数都继承于Function函数，因此，所有函数都继承了这个方法。

apply的功能是，函数借用。将函数借用给一个对象，帮助他实现函数所定义的逻辑的功能。

如例子：

function ab(x, y) {  
 this.**x** = x;  
 this.**y** = y;  
}  
ab.**prototype**.move = function (x, y) {  
 this.**x** += x;  
 this.**y** += y  
}  
var c = new ab(1, 1);  
**console**.log(c);  
c.move(2, 2)  
**console**.log(c);  
**p** = {**x**: 0, **y**: 0, **z**: 0}  
**console**.log(**p**);  
c.move.apply(**p**, [5, 5]) //在这步，ab的实例c将move方法提供给了p，参数是数组 **console**.log(**p**);

输出结果为：

ab {x: 1, y: 1}

ab {x: 3, y: 3}

Object {x: 0, y: 0, z: 0}

Object {x: 5, y: 5, z: 0}

注：如果c中没有y，那么在执行后，会显示y的值为NaN