　　前端是庞大的，包括HTML、CSS、Javascript、Image、Flash等等各种各样的资源。前端优化是复杂的，针对方方面面的资源都有不同的方式。那么，前端优化的目的是什么?

　　1. 从用户角度而言，优化能够让页面加载得更快、对用户的操作响应得更及时，能够给用户提供更为友好的体验。

　　2. 从服务商角度而言，优化能够减少页面请求数、或者减小请求所占带宽，能够节省可观的资源。

　　总之，恰当的优化不仅能够改善站点的用户体验并且能够节省相当的资源利用。

　　前端优化的途径有很多，按粒度大致可以分为两类，第一类是页面级别的优化，例如HTTP请求数、脚本的无阻塞加载、内联脚本的位置优化等;第二类则是代码级别的优化，例如Javascript中的DOM操作优化、CSS选择符优化、图片优化以及HTML结构优化等等。另外，本着提高投入产出比的目的，后文提到的各种优化策略大致按照投入产出比从大到小的顺序排列。

　　一、页面级优化

　　1. 减少HTTP请求数

　　这条策略基本上所有前端人都知道，而且也是最重要最有效的。都说要减少HTTP请求，那请求多了到底会怎么样呢?首先，每个请求都是有成本的，既包含时间成本也包含资源成本。一个完整的请求都需要经过DNS寻址、与服务器建立连接、发送数据、等待服务器响应、接收数据这样一个“漫长”而复杂的过程。时间成本就是用户需要看到或者“感受”到这个资源是必须要等待这个过程结束的，资源上由于每个请求都需要携带数据，因此每个请求都需要占用带宽。另外，由于浏览器进行并发请求的请求数是有上限的(具体参见此处)，因此请求数多了以后，浏览器需要分批进行请求，因此会增加用户的等待时间，会给用户造成站点速度慢这样一个印象，即使可能用户能看到的第一屏的资源都已经请求完了，但是浏览器的进度条会一直存在。

　　减少HTTP请求数的主要途径包括：

　　(1). 从设计实现层面简化页面

　　如果你的页面像百度首页一样简单，那么接下来的规则基本上都用不着了。保持页面简洁、减少资源的使用时最直接的。如果不是这样，你的页面需要华丽的皮肤，则继续阅读下面的内容。

　　(2). 合理设置HTTP缓存

　　缓存的力量是强大的，恰当的缓存设置可以大大的减少HTTP请求。以有啊首页为例，当浏览器没有缓存的时候访问一共会发出78个请求，共600多K数据(如图1.1)，而当第二次访问即浏览器已缓存之后访问则仅有10个请求，共20多K数据(如图1.2)。(这里需要说明的是，如果直接F5刷新页面的话效果是不一样的，这种情况下请求数还是一样，不过被缓存资源的请求服务器是304响应，只有Header没有Body，可以节省带宽)

　　怎样才算合理设置?原则很简单，能缓存越多越好，能缓存越久越好。例如，很少变化的图片资源可以直接通过HTTP Header中的Expires设置一个很长的过期头;变化不频繁而又可能会变的资源可以使用Last-Modifed来做请求验证。尽可能的让资源能够在缓存中待得更久。关于HTTP缓存的具体设置和原理此处就不再详述了，有兴趣的可以参考下列文章：

[HTTP1.1协议中关于缓存策略的描述](http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html)

[Fiddler HTTP Performance中关于缓存的介绍](http://msdn.microsoft.com/en-us/library/bb250442(VS.85).aspx)

　　(3). 资源合并与压缩

　　如果可以的话，尽可能的将外部的脚本、样式进行合并，多个合为一个。另外，CSS、Javascript、Image都可以用相应的工具进行压缩，压缩后往往能省下不少空间。

　　(4). CSS Sprites

　　合并CSS图片，减少请求数的又一个好办法。

　　(5). Inline Images

　　使用data: [URL scheme](http://tools.ietf.org/html/rfc2397)的方式将图片嵌入到页面或CSS中，如果不考虑资源管理上的问题的话，不失为一个好办法。如果是嵌入页面的话换来的是增大了页面的体积，而且无法利用浏览器缓存。使用在CSS中的图片则更为理想一些。

　　(6). Lazy Load Images

　　这条策略实际上并不一定能减少HTTP请求数，但是却能在某些条件下或者页面刚加载时减少HTTP请求数。对于图片而言，在页面刚加载的时候可以只加载第一屏，当用户继续往后滚屏的时候才加载后续的图片。这样一来，假如用户只对第一屏的内容感兴趣时，那剩余的图片请求就都节省了。[有啊首页](http://youa.baidu.com/)曾经的做法是在加载的时候把第一屏之后的图片地址缓存在Textarea标签中，待用户往下滚屏的时候才“惰性”加载。

　　2. 将外部脚本置底

　　前文有谈到，浏览器是可以并发请求的，这一特点使得其能够更快的加载资源，然而外链脚本在加载时却会阻塞其他资源，例如在脚本加载完成之前，它后面的图片、样式以及其他脚本都处于阻塞状态，直到脚本加载完成后才会开始加载。如果将脚本放在比较靠前的位置，则会影响整个页面的加载速度从而影响用户体验。解决这一问题的方法有很多，在[这里有比较详细的介绍](http://www.stevesouders.com/blog/2009/04/27/loading-scripts-without-blocking/)(这里是译文和[更详细的例子](http://varnow.org/?p=42))，而最简单可依赖的方法就是将脚本尽可能的往后挪，减少对并发下载的影响。

　　3. 异步执行inline脚本

　　inline脚本对性能的影响与外部脚本相比，是有过之而无不及。首页，与外部脚本一样，inline脚本在执行的时候一样会阻塞并发请求，除此之外，由于浏览器在页面处理方面是单线程的，当inline脚本在页面渲染之前执行时，页面的渲染工作则会被推迟。简而言之，inline脚本在执行的时候，页面处于空白状态。鉴于以上两点原因，建议将执行时间较长的inline脚本异步执行，异步的方式有很多种，例如使用script元素的defer属性(存在兼容性问题和其他一些问题，例如不能使用document.write)、使用setTimeout，此外，在HTML5中引入了[Web Workers](https://developer.mozilla.org/En/Using_web_workers)的机制，恰恰可以解决此类问题。

　　4. Lazy Load Javascript

　　随着Javascript框架的流行，越来越多的站点也使用起了框架。不过，一个框架往往包括了很多的功能实现，这些功能并不是每一个页面都需要的，如果下载了不需要的脚本则算得上是一种资源浪费-既浪费了带宽又浪费了执行花费的时间。目前的做法大概有两种，一种是为那些流量特别大的页面专门定制一个专用的mini版框架，另一种则是Lazy Load。YUI则使用了第二种方式，在YUI的实现中，最初只加载核心模块，其他模块可以等到需要使用的时候才加载。

　　5. 将CSS放在HEAD中

　　如果将CSS放在其他地方比如BODY中，则浏览器有可能还未下载和解析到CSS就已经开始渲染页面了，这就导致页面由无CSS状态跳转到CSS状态，用户体验比较糟糕。除此之外，有些浏览器会在CSS下载完成后才开始渲染页面，如果CSS放在靠下的位置则会导致浏览器将渲染时间推迟。

　　6. 异步请求Callback

　　在某些页面中可能存在这样一种需求，需要使用script标签来异步的请求数据。类似：

　　Javascript:

|  |
| --- |
| /\*Callback函数\*/   function myCallback(info){   //do something here   } |

　　HTML:

|  |
| --- |
| <script type="text/javascript" src="http://abc.com/cb"></script> |

　　cb返回的内容:

|  |
| --- |
| myCallback('Hello world!'); |

      像以上这种方式直接在页面上写<script>对页面的性能也是有影响的，即增加了页面首次加载的负担，推迟了DOMLoaded和window.onload事件的触发时机。如果时效性允许的话，可以考虑在DOMLoaded事件触发的时候加载，或者使用setTimeout方式来灵活的控制加载的时机。

　　7. 减少不必要的HTTP跳转

　　对于以目录形式访问的HTTP链接，很多人都会忽略链接最后是否带’/'，假如你的服务器对此是区别对待的话，那么你也需要注意，这其中很可能隐藏了301跳转，增加了多余请求。具体参见下图，其中第一个链接是以无’/'结尾的方式访问的，于是服务器有了一次跳转。

　　8. 避免重复的资源请求

　　这种情况主要是由于疏忽或页面由多个模块拼接而成，然后每个模块中请求了同样的资源时，会导致资源的重复请求

　　二、代码级优化

　　1. Javascript

　　(1). DOM

　　DOM操作应该是脚本中最耗性能的一类操作，例如增加、修改、删除DOM元素或者对DOM集合进行操作。如果脚本中包含了大量的DOM操作则需要注意以下几点：

　　a. HTML Collection

　　在脚本中document.images、document.forms、getElementsByTagName()返回的都是HTMLCollection类型的集合，在平时使用的时候大多将它作为数组来使用，因为它有length属性，也可以使用索引访问每一个元素。不过在访问性能上则比数组要差很多，原因是这个集合并不是一个静态的结果，它表示的仅仅是一个特定的查询，每次访问该集合时都会重新执行这个查询从而更新查询结果。所谓的“访问集合”包括读取集合的length属性、访问集合中的元素。

　　因此，当你需要遍历HTML Collection的时候，尽量将它转为数组后再访问，以提高性能。即使不转换为数组，也请尽可能少的访问它，例如在遍历的时候可以将length属性、成员保存到局部变量后再使用局部变量。

　　b. Reflow & Repaint

　　除了上面一点之外，DOM操作还需要考虑浏览器的Reflow和Repaint，因为这些都是需要消耗资源的，具体的可以参加以下文章：

[如何减少浏览器的repaint和reflow?](http://varnow.org/?p=232)

[Understanding Internet Explorer Rendering Behaviour](http://blog.dynatrace.com/2009/12/12/understanding-internet-explorer-rendering-behaviour/)

[Notes on HTML Reflow](http://www.mozilla.org/newlayout/doc/reflow.html)

　　(2). 慎用with

       with(obj){ p = 1}; 代码块的行为实际上是修改了代码块中的[执行环境](http://varnow.org/?p=171)，将obj放在了其作用域链的最前端，在with代码块中访问非局部变量是都是先从obj上开始查找，如果没有再依次按作用域链向上查找，因此使用with相当于增加了作用域链长度。而每次查找作用域链都是要消耗时间的，过长的作用域链会导致查找性能下降。

　　因此，除非你能肯定在with代码中只访问obj中的属性，否则慎用with，替代的可以使用局部变量缓存需要访问的属性。

　　(3). 避免使用eval和Function

　　每次 eval 或 Function 构造函数作用于字符串表示的源代码时，脚本引擎都需要将源代码转换成可执行代码。这是很消耗资源的操作 —— 通常比简单的函数调用慢100倍以上。

　　eval 函数效率特别低，由于事先无法知晓传给 eval 的字符串中的内容，eval在其上下文中解释要处理的代码，也就是说编译器无法优化上下文，因此只能有浏览器在运行时解释代码。这对性能影响很大。

　　Function 构造函数比eval略好，因为使用此代码不会影响周围代码;但其速度仍很慢。

　　此外，使用eval和Function也不利于Javascript压缩工具执行压缩。

　　(4). 减少作用域链查找

　　前文谈到了作用域链查找问题，这一点在循环中是尤其需要注意的问题。如果在循环中需要访问非本作用域下的变量时请在遍历之前用局部变量缓存该变量，并在遍历结束后再重写那个变量，这一点对全局变量尤其重要，因为全局变量处于作用域链的最顶端，访问时的查找次数是最多的。

　　低效率的写法：

|  |
| --- |
| //全局变量   var globalVar = 1;   function myCallback(info){      for( var i = 100000; i--;){         //每次访问globalVar都需要查找到作用域链最顶端，本例中需要访问100000次         globalVar += i;      }  } |

　　更高效的写法：

|  |
| --- |
| //全局变量   var globalVar = 1;   function myCallback(info){      //局部变量缓存全局变量      var localVar = globalVar;      for( var i = 100000; i--;){         //访问局部变量是最快的         localVar += i;      }      //本例中只需要访问2次全局变量      globalVar = localVar;   } |

　　此外，要减少作用域链查找还应该减少闭包的使用。

　　(5). 数据访问

　　Javascript中的数据访问包括直接量(字符串、正则表达式)、变量、对象属性以及数组，其中对直接量和局部变量的访问是最快的，对对象属性以及数组的访问需要更大的开销。当出现以下情况时，建议将数据放入局部变量：

　　a. 对任何对象属性的访问超过1次

　　b. 对任何数组成员的访问次数超过1次

　　另外，还应当尽可能的减少对对象以及数组深度查找。

　　(6). 字符串拼接

　　在Javascript中使用"+"号来拼接字符串效率是比较低的，因为每次运行都会开辟新的内存并生成新的字符串变量，然后将拼接结果赋值给新变量。与之相比更为高效的做法是使用数组的join方法，即将需要拼接的字符串放在数组中最后调用其join方法得到结果。不过由于使用数组也有一定的开销，因此当需要拼接的字符串较多的时候可以考虑用此方法。

　　关于Javascript优化的更详细介绍请参考：

[Write Efficient Javascript(PPT)](http://www.slideshare.net/nzakas/writing-efficient-javascript)

[Efficient JavaScript](http://dev.opera.com/articles/view/efficient-javascript/)

　　2. CSS选择符

　　在大多数人的观念中，都觉得浏览器对CSS选择符的解析式从左往右进行的，例如

|  |
| --- |
| #toc A { color: #444; } |

　　这样一个选择符，如果是从右往左解析则效率会很高，因为第一个ID选择基本上就把查找的范围限定了，但实际上浏览器对选择符的解析是从右往左进行的。如上面的选择符，浏览器必须遍历查找每一个A标签的祖先节点，效率并不像之前想象的那样高。根据浏览器的这一行为特点，在写选择符的时候需要注意很多事项，有人已经一一列举了，[详情参考此处](https://developer.mozilla.org/en/Writing_Efficient_CSS)。

　　3. HTML

　　对HTML本身的优化现如今也越来越多的受人关注了，详情可以参见这篇[总结性文章](http://varnow.org/?p=249)。

　　4. Image压缩

　　图片压缩是个技术活，不过现如今这方面的工具也非常多，压缩之后往往能带来不错的效果，具体的压缩原理以及方法在《Even Faster Web Sites》第10章有很详细的介绍，有兴趣的可以去看看。

　　总结

　　本文从页面级以及代码级两个粒度对前端优化的各种方式做了一个总结，这些方法基本上都是前端开发人员在开发的过程中可以借鉴和实践的，除此之外，完整的前端优化还应该包括很多其他的途径，例如CDN、Gzip、多域名、无Cookie服务器等等，由于对于开发人员的可操作性并不强大，在此也就不多叙述了，详细的可以参考Yahoo和Google的这些“金科玉律”。1.Javascript简介

HTML是纯静态的的页面，而Javascript让页面有了动态的效果，比如;OA中模块的拖拉

所有的浏览器都会内置Javascript的解释器

1992年 Nombas公司开发出C减减的嵌入式脚本语言。这是最好的HTML页面的脚本语言。

Netscape为了扩展其浏览器的功能，开发了一套LiveScript，并与1995年与SUN公司联合把其改名为javascript,它的主要目的是处理一些输入的有效性验证，而之前这个工作是留给perl之类的服务器端语言完成，在以前使用电话线调制解调器(28.8kb/s)的时代，如此慢的与服务器交互，这绝对是一件很痛苦的事情。Javascript的出现，解决了这个问题，因为它的验证是基于客户端的。

微软公司早期版本的浏览器仅支持自己的vbscript，但后来不得不加入javascript

IE3中搭载Javascipt的克隆版本，命名为jscript

在一次技术会议中，sun，microsoft，netscape公司联合制定了ECMA-Script标准

在2005前，网页上提示框，广告越来越多，把javascipt滥用，使javascript背上了大量的罪名。

2005年，google公司的网上产品（google地图，gmail，google搜索建议）等使得ajax兴起，而javascript便是ajax最重要的元素之一

Javascript有三个部分组成

ECMAScript DOM BOM

WEB标准

网页主要有三部分组成

（结构HTML,XHTML，表现CSS，行为DOM，ECMA）

## 2.ECMA脚本

Javascript的语法

1. 区分大小写
2. 弱类型变量 var age=10 var name=”dd”
3. 每行结尾的分号可有可无，但建议还是加上
4. 注释与java相同

变量

变量是通过var关键字来声明的。（Variable）

变量的命名规则与java一致

注释有三种：// /\*\*/ <!-- --> 这个只能注释单行

### 2.1 javascript的Hello world

document.write()是写在文档的最前面

### 2.2 slice()、substring()、subtr

Slice和substring (2,5) 指的是从第3为开始，取（5-2）=3个数，其中slice的参数可以为负

Substr（2,5）指的是从第3为开始，取5个数。但ECMAscript 没有对该方法进行标准化，因此尽量少使用该方法

### 2.3 indexOf()和lastIndexOf(),isNan,typeOf

indexOf（”i”） //从前往后,i在第几位

indexOf（”i”,3）可选参数，从第几个字符开始往后找

lastIndexOf（”i”） //从后往前,i在第几位

lastIndexOf(”i”,3) //从后往前,i在第几位

如果没找到，则返回-1

String类型的变量，在Java中，用“”符号表示字符串，用’’表示单个字符。而在javascript中这两种都可以

**Nan（**not a number）

Alert(nan ==nan)返回false,因此不推荐使用nan本身，推荐函数isNan

Alert(isNanN(“ab”));//返回false

**typeof运算符**

var stmp = “test”;

alert(typeof stmp); //输出string

alert(type of 1);//输出number

此外：还有boolean,undefined,object(如果是引用类型或者null值,null值返回object，这其实是ecmascript的一个错误)

当声明的变量未初始化的时候，它的值就是undefined.当没有这个变量的时候，typeof 变

返回的值也是undefined。但是没声明的变量是不能参与计算的。

当函数无明确返回值时，返回的也是undefined

Function a(){

}

Alert(a() == undefined) //返回true

Alert(null == undefined)//返回true

### 2.4 数值计算

var mynum1 = 23.345;

var mynum2 = 45;

var mynum3 = -34;

var mynum4 = 9e5; //科学计数法 为 9\*10五次方

var fNumber = 123.456;

alert(fNumber.toExponential(1));//保留的小数点数 1.2e+2

alert(fNumber.toExponential(2));//1.23e+2

### .5 布尔值

var married = true;

alert("1." + typeof(married));//Boolean

married = "true";

alert("2." + typeof(married));//String

### .6 类型转换

转换成string类型有三种方式

var a = 3;

var b = a + "";

var c = a.toString();

var d = "student" + a;

toString()

var a=11;

document.write(a.toString(2) + "<br>");//转成2进制

document.write(a.toString(3) + "<br>");//转成3进制

如果不是数值，则转换报错

parseInt()

document.write(parseInt("1red6") + "<br>");//返回1，后面非数值的将全部忽略

document.write(parseInt("53.5") + "<br>");//返回53

document.write(parseInt("0xC") + "<br>"); //直接十进制转换12

document.write(parseInt("isaacshun@gmail.com") + "<br>");//NAN

document.write(parseInt("011",8) + "<br>"); 返回9

document.write(parseInt("011",10) + "<br>"); //指定为十进制 返回11

parseFloat()与parseInt()类似

### 2.7 数组

var aMap = new Array("China","USA","Britain");

**aMap[20]** = "Korea";

alert(aMap.length + " " + aMap[10] + " " + aMap[20]);//**aMap[10]返回undefined**

document.write(aMap.**join**("][") + "<br>"); //用“][”来连接

var sFruit = "apple,pear,peach,orange";

var aFruit = sFruit.**split**(",");

var aFruit = ["apple","pear","peach","orange"];

alert(aFruit.**reverse**().toString());

var aFruit = ["pear","apple","peach","orange"];

aFruit.**sort**();

var stack = new Array();

stack.**push**("red");

stack.push("green");

stack.push("blue");

document.write(stack.toString() + "<br>");

var vItem = stack.**pop**(); // blue

document.write(vItem + "<br>");

document.write(stack.toString()); // red green

### 2.8 if语句

//首先获取用户的一个输入，并用Number()强制转换为数字

var iNumber = **Number**(**prompt**("输入一个5到100之间的数字"**, ""**));//第二个参数,用于显示输入框的默认值

if(isNaN(iNumber)) //判断输入的是否是数字 NaN “Not a Number”

document.write("请确认你的输入正确");

else if(iNumber > 100 || iNumber < 5) //判断输入的数字范围

document.write("你输入的数字范围不在5和100之间");

else

document.write("你输入的数字是:" + iNumber);

### 2.9 switch

iWeek = parseInt(prompt("输入1到7之间的整数",""));

switch(iWeek){

case 1:

document.write("Monday");

break;

case 2:

document.write("Tuesday");

break;

case 3:

document.write("Wednesday");

break;

case 4:

document.write("Thursday");

break;

case 5:

document.write("Friday");

break;

case 6:

document.write("Saturday");

break;

case 7:

document.write("Sunday");

break;

default:

document.write("Error");

}

### 2.10 while语句

var i=iSum=0;

while(i<=100){

iSum += i;

i++;

}

alert(iSum);

do.while for break continue (与JAVA语法一致)

### 2.11 函数

function ArgsNum(){

return arguments.length;

}

document.write(ArgsNum("isaac",25) + "<br>");

document.write(ArgsNum() + "<br>");

document.write(ArgsNum(3) + "<br>");

**从这个例子可以看出，方法可以没有参数，也可以没有返回值，但是照样可以传入参数和返回值**。

### 2.12 Date对象

var myDate1 = new Date(); //运行代码前的时间

for(var i=0;i<3000000;i++);

var myDate2 = new Date(); //运行代码后的时间

document.write(myDate2);

var oMyDate = new Date();

var iYear = oMyDate.**getFullYear**();

var iMonth = oMyDate.**getMonth**() + 1; //月份是从0开始的

var iDate = oMyDate.**getDate**();

var iDay = oMyDate.**getDay**(); //0 为 星期日 1 为 星期一

function disDate(oDate, iDate){

var ms = oDate.**getTime**(); //换成毫秒数

ms -= iDate\*24\*60\*60\*1000; //计算相差的毫秒数

return new Date(ms); //返回新的时间对象

}

var oBeijing = new Date(2008,7,8);

var iNum = 100; //前100天

var oMyDate = disDate(oBeijing, iNum);

2.13 检测浏览器和操作系统

var sUserAgent = **navigator**.userAgent;

//检测Opera、KHTML

var isOpera = sUserAgent.indexOf("Opera") > -1;

var isKHTML = sUserAgent.indexOf("KHTML") > -1 || sUserAgent.indexOf("Konqueror") > -1 || sUserAgent.indexOf("AppleWebKit") > -1;

//检测IE、Mozilla

var isIE = sUserAgent.indexOf("compatible") > -1 && sUserAgent.indexOf("MSIE") > -1 && !isOpera;

var isMoz = sUserAgent.indexOf("Gecko") > -1 && !isKHTML;

//检测操作系统

var isWin = (navigator.platform == "Win32") || (navigator.platform == "Windows");

var isMac = (navigator.platform == "Mac68K") || (navigator.platform == "MacPPC") || (navigator.platform == "Macintosh");

var isUnix = (navigator.platform == "X11") && !isWin && !isMac;

if(isOpera) document.write("Opera ");

if(isKHTML) document.write("KHTML ");

if(isIE) document.write("IE ");

if(isMoz) document.write("Mozilla ");

if(isWin) document.write("Windows");

if(isMac) document.write("Mac");

if(isUnix) document.write("Unix");

* 1. Global对象

其实isNan，paraseInt等都是Global对象的方法

EncodeURI.因为有效的URI不能包含某些字符，如空格。这个方法就是用于将这个字符转换成UTF-8编码，使浏览器可以接受他们。

Var suil = “www.oseschool.com/pro file/a.html”;

Alert(encodeURI(suil));//www.oseschool.com/pro%20file/a.html

即将空格编码成%20

Eval方法

Eval(“alert(‘hi’)”);

当解析程序发现eval()时，它将把参数解析成真正的ECMA-script语句，然后插入该语句所在位置。

Global除了有内置方法外，还有很多内置的属性：如：undefined,nan,Array,String,Number，Date，RegExp等

* 1. Math对象

Max方法，min方法,ceil,floor,round,sqrt,random

Max(1,2,3);min(1.2,3.4);

想取到1~10的数据

Math.floor(Math.random()\*10+1)

2~9的数据

Math.floor(Math.random()\*9+2);

### 3.1 getElementsByTagName

function searchDOM(){

//**放在函数内，页面加载完成后才用<body>的onload加载，这时如果把alert这句改成用document.write则会把原内容覆盖掉，因为是后面才执行的**

var oLi = document.getElementsByTagName("li");

//输出长度、标签名称以及某项的文本节点值

alert(oLi.length + " " +oLi[0].**tagName** + " " + oLi[3].**childNodes**[0].**nodeValue**);

var oUl = document.getElementsByTagName("ul");

var oLi2 = oUl[1].getElementsByTagName("li");

alert(oLi2.length + " " +oLi2[0].tagName + " " + oLi2[1].childNodes[0].nodeValue);

}

<body onload="searchDOM()">

<ul>客户端语言

<li>HTML</li>

<li>JavaScript</li>

<li id="cssLi">CSS</li>

</ul>

<ul>服务器端语言

<li>ASP.NET</li>

<li>JSP</li>

<li>PHP</li>

</ul>

</body>

### 3.2 getElementById

var oLi = document.**getElementById**("cssLi");

oLi**.style .backgroundColor**="yellow"

//输出标签名称以及文本节点值

alert(oLi.tagName + " " + oLi.childNodes[0].nodeValue);

### 3.2 getElementsByName

alert(document.getElementsByName("a").length);

### 3.3 访问子节点

function myDOMInspector(){

var oUl = document.getElementById("myList"); //获取<ul>标记

var DOMString = "";

if(oUl.**hasChildNodes**()){ //判断是否有子节点

var oCh = oUl.childNodes;

for(var i=0;i<oCh.length;i++) //逐一查找

DOMString += oCh[i].**nodeName** + "\n";

}

alert(DOMString);

}

4访问父节点

**nodeName 如果为文本节点，则返回#text**

**tagName如果为文本节点，则返回undefined**

function myDOMInspector(){

var myItem = document.getElementById("myDearFood");

alert(myItem.**parentNode**.tagName);

}

function myDOMInspector(){

var myItem = document.getElementById("myDearFood");

var parentElm = myItem.parentNode;

while(**parentElm.className != "colorful" && parentElm != document.body)**

parentElm = parentElm.parentNode; //一路往上找

alert(parentElm.tagName);

}

<body onload="myDOMInspector()">

<div class="colorful">

<ul>

<li>糖醋排骨</li>

<li>圆笼粉蒸肉</li>

<li>泡菜鱼</li>

<li id="myDearFood">板栗烧鸡</li>

<li>麻婆豆腐</li>

</ul>

</div>

</body>

访问兄弟节点

function myDOMInspector(){

var myItem = document.getElementById("myDearFood");

//访问兄弟节点

var nextListItem = myItem.**nextSibling**;

var preListItem = myItem.**previousSibling**;

alert(nextListItem.tagName +" "+ preListItem.tagName);

}

**在Firefox中不支持，但是IE中却是支持的。**

### 3.6 第一个 最后一个 子节点

function nextSib(node){

var tempLast = node.parentNode.**lastChild**;

//判断是否是最后一个节点，如果是则返回null

if(node == tempLast)

return null;

var tempObj = node.nextSibling;

//逐一搜索后面的兄弟节点，直到发现元素节点为止

while(tempObj.nodeType!=1 && tempObj.nextSibling!=null)

tempObj = tempObj.nextSibling;

//三目运算符，**如果是元素节点**则返回节点本身，否则返回null

return (**tempObj.nodeType==1**)?tempObj:null;

}

function prevSib(node){

var tempFirst = node.parentNode.**firstChild**;

//判断是否是第一个节点，如果是则返回null

if(node == tempFirst)

return null;

var tempObj = node.previousSibling;

//逐一搜索前面的兄弟节点，直到发现元素节点为止

while(tempObj.nodeType!=1 && tempObj.previousSibling!=null)

tempObj = tempObj.previousSibling;

return (tempObj.nodeType==1)?tempObj:null;

}

function myDOMInspector(){

var myItem = document.getElementById("myDearFood");

//获取后一个元素兄弟节点

var nextListItem = nextSib(myItem);

//获取前一个元素兄弟节点

var preListItem = prevSib(myItem);

alert("后一项:" + ((nextListItem!=null)?nextListItem.firstChild.nodeValue:null) + " 前一项:" + ((preListItem!=null)?preListItem.firstChild.nodeValue:null) );

}

**nodeType**

|  |  |
| --- | --- |
| 元素element | 1 |
| 属性attr | 2 |
| 文本text | 3 |
| 注释comments | 8 |
| 文档document | 9 |

function showAttr(){

var btnShowAttr=document.getElementById("btnShowAttr"); //演示按钮，有很多属性

var attrs=btnShowAttr.**attributes**;

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

var attr=attrs[i];

alert('nodeType:'+**attr.nodeType); //attribute 的nodeType=2**

alert('attr:'+attr);

alert('attr.name:'+attr.name+'='+attr.value);

}

function showDocument(){

alert('nodeType:'+**document.nodeType); //9**

alert('nodeName:'+document.nodeName);

alert(document);

}

### 3.7 getAttribute setAttribute

function myDOMInspector(){

//获取图片

var myImg = document.getElementsByTagName("img")[0];

//获取图片title属性

alert(myImg.**getAttribute**("title"));

}

function changePic(){

//获取图片

var myImg = document.getElementsByTagName("img")[0];

//设置图片src和title属性

myImg.**setAttribute**("src","02.jpg");

myImg.setAttribute("title","紫荆公寓");

}

### 3.8创建新节点

function createP(){

var op = document.**createElement**("p");

var otext = document.**createTextNode**("HHHHH");

op.**appendChild**(otext);

op.setAttribute("style","text-align:center");

**document.body.appendChild(**op);

//创建完节点，就马上会影响到下面的操作，比如P的数量就会多1个

}

### 3.9删除节点

需要注意的是标签之间的父子关系！！！

function deleteP(){

var oP = document.getElementsByTagName("p")[0];

oP.parentNode.**removeChild**(oP); //删除节点

}

### 3.10替换节点

function replaceP(){

var oOldP = document.getElementsByTagName("p")[0];

var oNewP = document.createElement("p"); //新建节点

var oText = document.createTextNode("这是一个感人肺腑的故事");

oNewP.appendChild(oText);

oOldP.parentNode.**replaceChild**(oNewP,oOldP); //替换节点

}

### 3.11插入节点

function insertP(){

var oOldP = document.getElementsByTagName("p")[0];

var oNewP = document.createElement("p"); //新建节点

var oText = document.createTextNode("这是一个感人肺腑的故事");

oNewP.appendChild(oText);

oOldP.parentNode.**insertBefore**(oNewP,oOldP); //插入节点

}

没有insertAfter，但是可以自己写一个

function insertAfter(newElement, targetElement){

var oParent = targetElement.parentNode; //首先找到目标元素的父元素

if(oParent.lastChild == targetElement) //如果目标元素已经是最后一个子元素了

oParent.appendChild(newElement); //则直接用appendChild()加到子元素列表的最后

else //否则用insertBefore()插入到目标元素的下一个兄弟元素之前

oParent.insertBefore(newElement,targetElement.nextSibling);

}

function insertP(){

var oOldP = document.getElementById("myTarget");

var oNewP = document.createElement("p"); //新建节点

var oText = document.createTextNode("这是一个感人肺腑的故事");

oNewP.appendChild(oText);

insertAfter(oNewP,oOldP); //插入节点

}

其实这个也是通过**insertBefore原理来实现的**

### ****3.12 cloneNode(deepBoolean)****

复制并返回当前节点的复制节点，这个复制得到的节点是一个孤立的节点，不在document树中。复制原来节点的属性值，包括ID属性，所以在把这个新节点加到document之前，一定要修改ID属性，以便使它保持唯一。当然如果ID的唯一性不重要可以不做处理。

这个方法支持一个布尔参数，当deepBoolean设置true时，复制 当前节点的所有子节点，包括该节点内的文本。

<p id=”mypara”>11111</p>

p=document.getElementById("mypara")

pclone = p.cloneNode(true);

p.parentNode.appendChild(pclone

### 3.12文档碎片

function insertPs(){

var aColors = ["red","green","blue","magenta","yellow","chocolate","black","aquamarine","lime","fuchsia","brass","azure","brown","bronze","deeppink","aliceblue","gray","copper","coral","feldspar","orange","orchid","pink","plum","quartz","purple"];

var oFragment = document.**createDocumentFragment**(); //创建文档碎片

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

var oP = document.createElement("p");

var oText = document.createTextNode(aColors[i]);

oP.appendChild(oText);

oFragment.appendChild(oP); //将节点先添加到碎片中

}

document.body.appendChild(oFragment); //最后一次性添加到页面

}

### 3.13 innerHTML

function myDOMInnerHTML(){

var myDiv = document.getElementById("myTest");

alert(myDiv.innerHTML); //直接显示innerHTML的内容

//修改innerHTML，可直接添加代码

myDiv.innerHTML = "<img src='01.jpg' title='情人坡'>";

}

**innerHTML可同时显示没有的代码**

### 3.14 换皮肤

<style type="text/css">

.myUL1{

color:#0000FF;

font-family:Arial;

font-weight:bold;

}

.myUL2{

color:#FF0000;

font-family:Georgia, "Times New Roman", Times, serif;

}

</style>

<script language="javascript">

function check(){

var oMy = document.getElementsByTagName("ul")[0];

oMy.className =(oMy.className=="myUL1"? "myUL2":"myUL1"); //修改CSS类

}

</script>

</head>

<body>

<ul onclick="check()" class="myUL1">

<li>HTML</li>

<li>JavaScript</li>

<li>CSS</li>

</ul>

</body>

## 4.表格与表单

### 4.1 动态添加行

<script language="javascript">

window.onload=function(){

var oTr = document.getElementById("member").**insertRow**(2); //插入一行

var aText = new Array();

aText[0] = document.createTextNode("fresheggs");

aText[1] = document.createTextNode("W610");

aText[2] = document.createTextNode("Nov 5th");

aText[3] = document.createTextNode("Scorpio");

aText[4] = document.createTextNode("1038818");

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

**var oTd = oTr.insertCell(i);**

**oTd.appendChild(aText[i]);**

}

}

</script>

### 4.2修改单元格内容

<script language="javascript">

window.onload=function(){

var oTable = document.getElementById("member");

**oTable.rows[3].cells[4].innerHTML** = "lost"; //修改单元格内容

}

</script>

### 4.3 动态删除

parentElement是IE dom，

parentNode是标准DOM

<script language="javascript">

window.onload=function(){

var oTable = document.getElementById("member");

**oTable.deleteRow(2);** //删除一行，后面的行号自动补齐//指从table中的第2行开始进行删除

**oTable.rows[2].deleteCell(1);** //删除一个单元格，后面的也自动补齐

}

</script>

<script language="javascript">

function myDelete(){

var oTable = document.getElementById("member");

//删除该行

this.parentNode.parentNode.parentNode.removeChild(this.parentNode.parentNode);

}

window.onload=function(){

var oTable = document.getElementById("member");

var oTd;

//动态添加delete链接

for(var i=1;i<oTable.rows.length;i++){

oTd = oTable.rows[i].insertCell(5);

oTd.innerHTML = "<a href='#'>delete</a>";

oTd.firstChild.onclick = myDelete; //添加删除事件

}

}

</script>

### 4.4动态删除列

<script language="javascript">

function deleteColumn(oTable,iNum){

//自定义删除列函数，即每行删除相应单元格

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

oTable.rows[i].deleteCell(iNum);

}

window.onload=function(){

var oTable = document.getElementById("member");

deleteColumn(oTable,2);

}

</script>

### 4.5 控制textarea的字符个数

<script language="javascript">

function LessThan(oTextArea){

//返回文本框字符个数是否符号要求的boolean值

**return oTextArea.value.length < oTextArea.getAttribute("maxlength");**

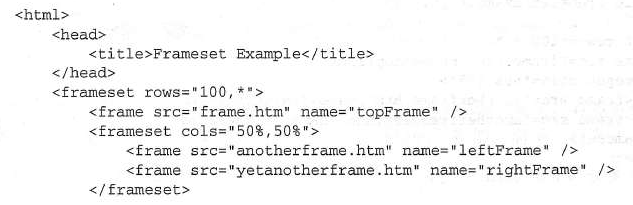
}

</script>

## 5.BOM模型

浏览器对象模型，可以对浏览器窗口进行访问和操作，使用BOM,开发者可以移动窗口，改变状态栏中的文本等与页面内容不相关的操作

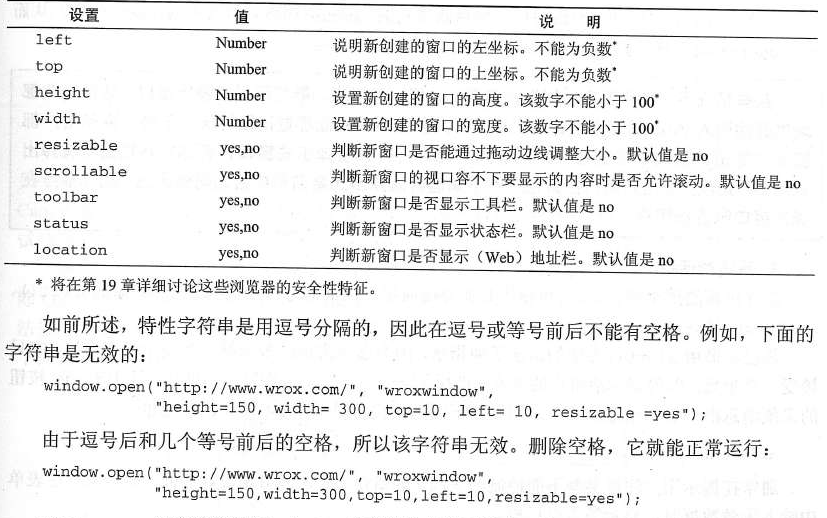
### Window对象



这里可以用window.frames[0]或者用windows.frames[“topFrame”]来引用框架，也可以用topl来代替window属性。Top.frames[0] 。window对象可以忽略

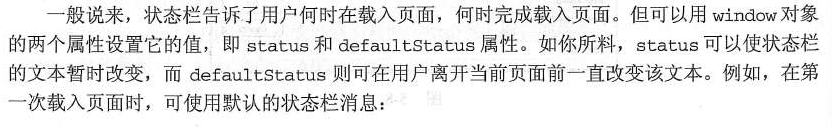
提供的方法有moveto,moveBy,resizeTo,resizeBy等方法。但尽量避免使用它们，因为会对用户浏览产生影响

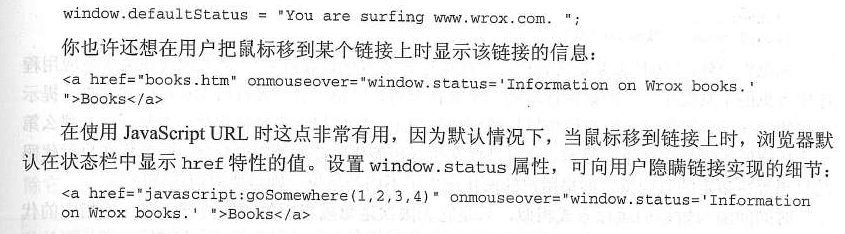
### Open方法



除了open方法，还有alert,comfirm,prompt方法

### 状态栏





### Settimeout与setInterval

Settimeout

下面的代码都是在1秒钟后显示一条警告

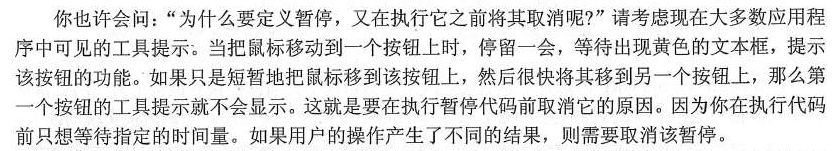
Settimeout(“alert(‘aa’),1000”);

Settimeout(function(){alert(‘aa’);},1000);

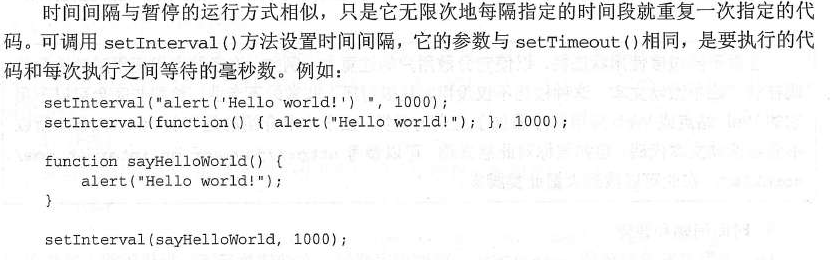
如果要还未执行的暂停，可调用clearTimeOut()方法

Var si = Settimeout(function(){alert(‘aa’);},1000);

clearTimeout(si);



setInterval



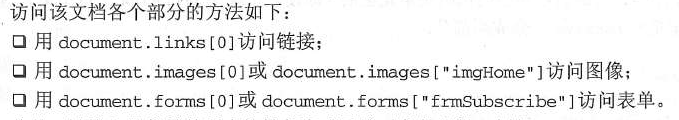
### History

向后一页window.history.go(-1); 等于history.back();

向前一页window.history.go(1); 等于 history.forword();

### Document

LastModified,title,URL属性都是比较常用



### Location对象

### Navigator对象

### Screen对象

## 6.事件

### 6.1 冒泡型事件

<script language="javascript">

function add(sText){

var oDiv = document.getElementById("display");

oDiv.innerHTML += sText; //输出点击顺序

}

</script>

</head>

<body onclick="add('body<br>');">

<div onclick="add('div<br>');">

<p onclick="add('p<br>');">Click Me</p>

</div>

<div id="display"></div>

</body>

**先执行最里面的p，再往外执行**

### 6.2 监听函数

<script language="javascript">

**window.onload** = function(){

var oP = document.getElementById("myP"); //找到对象

**oP.onclick** = function(){ //设置事件监听函数

alert('我被点击了');

}

}

</script>

</head>

<body>

<div>

<p id="myP">Click Me</p>

</div>

</body>

**Function a(){}**

**oP.onclick** = a

这样会先把a函数加载到缓存，不是最佳方案

**Var A = Function(){}**

**oP.onclick** = a

这样只有在onclick事件发生的时候，加载该函数

**若以上的监听函数，在onclick的时候，需要执行多个函数，那就只能用以下的方法**：

IE标准：

<script language="javascript">

function fnClick(){

alert("我被点击了");

oP.**detachEvent**("onclick",fnClick); //点击了一次后删除监听函数

}

**var oP;//声明在函数外，这样就可以两个函数一起使用**

window.onload = function(){

oP = document.getElementById("myP"); //找到对象

oP.**attachEvent**("onclick",fnClick); //添加监听函数

}

</script>

</head>

<body>

<div>

<p id="myP">Click Me</p>

</div>

</body>

**也可以添加多个监听器**

oP.attachEvent("onclick",fnClick1); //添加监听函数1

oP.attachEvent("onclick",fnClick2); //添加监听函数2

执行顺序为fnClick2-> fnClick1

**但是以上的监听器均为IE中的标准，而符合标准DOM（firefox）的监听器如下**

oP.addEventListener("click",fnClick1,false); //添加监听函数1

oP.addEventListener("click",fnClick2,false); //添加监听函数2

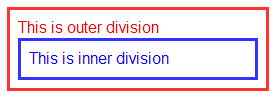
**因此这种方式在Firefox中支持，而在IE中不支持**

**为了兼容性，可这样写**

if (el.addEventListener)...{  
el.addEventListener('click', KindDisableMenu, false);  
} else if (el.attachEvent)...{

 el.attachEvent('onclick', KindDisableMenu);  
}

第三个参数为useCapture



而useCapture這個參數就是在控制這時候兩個click事件的先後順序。如果是false，那就會使用bubbling，他是從內而外的流程，所以會先執行藍色元素的click事件再執行紅色元素的click事件，如果是true，那就是capture，和bubbling相反是由外而內

### 6.3 事件的类型 event.type

IE浏览器中事件对象是window对象的一个属性event

Op.onlick=function(){ var o = window.event}

而标准DOM中规定event对象必须作为唯一的参数传给事件处理函数

Op.onclick=function(oevent){

}

因此为了兼容两种浏览器，通常采用下面的方法

Op.onclick=function(o){

If(window.event)//**假如不等于空，则为IE浏览器**

O = window.event;

}

<script language="javascript">

function handle(oEvent){

var oDiv = document.getElementById("display");

**if(window.event) oEvent = window.event;** //处理兼容性，获得事件对象

if(**oEvent.type** == "click") //检测事件名称

oDiv.innerHTML += "你点击了我&nbsp&nbsp;";

else if( oEvent.type == "mouseover")

oDiv.innerHTML += "你移动到我上方了&nbsp&nbsp;";

}

window.onload = function(){

var oImg = document.getElementsByTagName("img")[0];

**oImg.onclick = handle;**

**oImg.onmouseover = handle;**

}

</script>

**还有很多鼠标事件**

window.onload = function(){

var oImg = document.getElementsByTagName("img")[0];

oImg.onmousedown = handle; //将鼠标事件除了mousemove外都监听

oImg.onmouseup = handle;

oImg.onmouseover = handle;

oImg.onmouseout = handle;

oImg.onclick = handle;

oImg.ondblclick = handle;

}

### 6.4 事件的激活元素 event.srcElement 或者 target

<script language="javascript">

function handle(oEvent){

if(window.event) oEvent = window.event; //处理兼容性，获得事件对象

var oTarget;

**if(oEvent.srcElement) //处理兼容性，获取事件目标**

**oTarget = oEvent.srcElement; //IE支持的写法**

**else**

**oTarget = oEvent.target; //Firefox支持的写法**

alert(oTarget.tagName); //弹出目标的标记名称

}

window.onload = function(){

var oImg = document.getElementsByTagName("img")[0];

oImg.onclick = handle;

}

</script>

**event.button**

<script language="javascript">

function TestClick(oEvent){

var oDiv = document.getElementById("display");

if(window.event)

oEvent = window.event;

oDiv.innerHTML += **oEvent.button**; //输出button的值

}

document.onmousedown = TestClick;

window.onload = TestClick; //测试未按下任何键

</script>

</head>

<body>

<div id="display"></div>

</body>

在IE/Opera中，是window.event，而在Firefox中，是event  
而事件的对象，在IE中是window.event.srcElement，在Firefox中是event.target，而在Opera中则两者都支持。

在 IE 里面   
左键是 window.event.button = **1**   
右键是 window.event.button = **2**  
中键是 window.event.button = **4**  
没有按键动作的时候 window.event.button = **0**  
  
在 Firefox 里面   
左键是 event.button = **0**   
右键是 event.button = **2**  
中键是 event.button = **1**   
没有按键动作的时候 event.button = **0**  
  
在 Opera 7.23/7.54 里面   
鼠标左键是 window.event.button = **1**  
没有按键动作的时候 window.event.button = **1**   
右键和中键无法获取

**键盘事件**

window.onload = function(){

var oTextArea = document.getElementsByTagName("textarea")[0];

**oTextArea.onkeydown = handle; //监听所有键盘事件**

**oTextArea.onkeyup = handle;**

**oTextArea.onkeypress = handle;**

}

**e.keyCode;**

onkeypress是在用户按下并放开任何**字母数字键**时发生。系统按钮（例如，箭头键和功能键, Shift、Ctrl、Alt、F1、F2）无法得到识别。   
onkeyup 是在用户放开任何先前按下的键盘键时发生。   
onkeydown 是在用户按下**任何键盘键**（包括系统按钮，如箭头键和功能键）时发生

**屏蔽鼠标右键**

第一种方式：

<script language="javascript">

function block(oEvent){

if(window.event)

oEvent = window.event;

if(oEvent.button == 2)

alert("鼠标右键不可用");

}

document.**onmousedown** = block;

</script>

第二种方式：

<script language="javascript">

function block(oEvent){

if(window.event){

oEvent = window.event;

oEvent.returnValue = false; //取消默认事件 支持IE

}else

oEvent.preventDefault(); //取消默认事件 支持Firefox

}

document.**oncontextmenu** = block;

</script>

**6.8伸缩的菜单**

<script language="javascript">

function change(){

//通过父元素li，找到兄弟元素ul

var oSecondDiv = **this**.parentNode.getElementsByTagName("ul")[0];//**这里的this就是下面的OA**

//CSS交替更换来实现显、隐

if(oSecondDiv.className == "myHide")

oSecondDiv.className = "myShow";

else

oSecondDiv.className = "myHide";

}

window.onload = function(){

var oUl = document.getElementById("listUL");

var aLi = oUl.**childNodes**; //子元素

var oA;

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

//如果子元素为li，且这个li有子菜单ul

if(**aLi[i].tagName == "LI"** && aLi[i].getElementsByTagName("ul").length){

oA = aLi[i].firstChild; //找到超链接

oA.onclick = change; //动态添加点击函数

}

}

}

</script>

**tagName出来的都是大写**

**总结**

**《Event详解》**

DOM常用属性

tagName

nodeValue

nodeName

nodeType

parentNode

childNodes

firstChild

lastChild

nextSibling (IE)

previousSibling (IE)

attributes

innerHTML

style

className

方法

getElementById

getElementsByName

getElementsByTagName

hasChildNodes()

getAttribute

setAttribute

createElement

createTextNode

appendChild

removeChild

replaceChild

insertBefore

cloneNode

createDocumentFragment

detachEvent

attachEvent（IE) addEventListener(DOM)

event属性

type

keyCode

srcElement(IE) target(DOM)

button

鼠标事件

onclick

onmouseover

onmousedown

onmouseup

onmouseout

ondblclick

键盘事件

onkeydown

onkeyup

onkeypress

window事件

onload

document事件

oncontextmenu

write

onmousedown

**7.JavaScipt优化和调试**

7.1 错误和异常

拼写错误、访问不存在的变量，括号不匹配，等号与赋值

声明变量时，要记住局部变量和全局变量的区别

Function square(num){

Total = num\*num;

Return total;

}

Var total = 50;

Var number = Square(20);

Alert(total);

这些代码将不可避免地导致全局变量total的值发生变化。

Function square(num){

**Var** Total = num\*num;

Return total;

}

### 7.2 错误处理

onerror

<head>

<title>onerror</title>

<script language="javascript">

window.onerror = function(){

alert("出错啦！");

return true; //屏蔽系统事件

}

</script>

</head>

<body onload="nonExistent()">

</body>

Try..catch

<script language="javascript">

try{

alert("this is an example");

alert(fresheggs);

} catch(exception){

var sError = "";

for(var i in exception)

sError += i + ":" + exception[i] + "\n";

alert(sError);

}

</script>

### 7.3 调试器

IE-工具-Intenet选项-高级->禁用调试，显示脚本错误

Firefox错误控制台

Microsoft script debugger

Venkman firefox的插件

6.4 JavaScript优化

1.提高JavaScript下载时间。将JavaScript写到同一行

2.尽量使用内置函数（因为内置函数是通过C语言编译到浏览器中的）

**.实例**

1 图片查看器

<html>

<head>

<title></title>

<script>

function showPic(obj){

var h = obj.getAttribute("href");

document.getElementById("image").setAttribute("src",h);

}

</script>

</head>

<body>

<h1>Snapshots</h1>

<ul>

<li><a href="photo/01.jpg" title="a" onclick="showPic(this);**return false;**">01</a></li>

<li><a href="photo/02.jpg" title="b" onclick="showPic(this);return false;">02</a></li>

<li><a href="photo/03.jpg" title="c" onclick="showPic(this);return false;">03</a></li>

<li><a href="photo/04.jpg" title="d" onclick="showPic(this);return false;">04</a></li>

<li><a href="photo/05.jpg" title="e" onclick="showPic(this);return false;">05</a></li>

</ul>

<img id="image" src="photo/01.jpg" alt="my image"></img>

</body>

</html>

Return false指的是把默认的noclick事件取消

给其加上css

<style>

body{

color:#333;

background:#ccc;

margin:1em 10%;

}

a{

text-decoration:none;

padding:10px;

color:#c60;

}

a:link, a:visited{

color: #A62020; background-color: #ecd8db; text-decoration: none; padding:4px 10px 4px 10px;

border-top:1px solid #EEE;

border-left:1px solid #EEE;

border-bottom:1px solid #717171;

border-right:1px solid #717171;

}

a:hover{

color:#821818; background-color:#e2c4c9; padding:5px 8px 3px 12px;

border-top:1px solid #717171;

border-left:1px solid #717171;

border-bottom:1px solid #EEE;

border-right:1px solid #EEE;}

ul{

margin:0px;

padding:0px;

}

li{

list-style-type:none;

display:inline;

}

img{

margin:10px 0px;

}

</style>

当时现在有个缺陷，就是onclick的事件直接写在了HTML上，**分离**

先给ul加上个属性id<ul id=”img\_ul”>

window.onload = prepareGalley;

function prepareGalley(){

var img\_ul = document.getElementById("img\_ul");

var links = img\_ul.getElementsByTagName("a");

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

links[i].onclick = function(){

showPic(this);

return false;

}

}

}

有一个问题，如果onload的函数有多个怎么办？

window.onload = prepareGalley1;

window.onload = prepareGalley2;

显然，这样第一个函数就会被第二个函数覆盖。

可以这样写

Window.onload = function(){

prepareGalley1();

prepareGalley2();

}

还有一个比这个更NB的写法，由Simon Willison写的

function addLoadEvent(func){

var oldonload = window.onload;

if(typeof window.onload !='function'){

window.onload = func;

}else{

window.onload = function(){

oldonload();

func();

}

}

}

addLoadEvent(prepareGalley1);

addLoadEvent(prepareGalley2);

编写一个方法 求一个字符串的字节长度

new function(s){

if(!arguments.length||!s) return null;

if(""==s) return 0;

var l=0;

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

if(s.charCodeAt(i)>255) l+=2; else l++;

}

alert(l);

}("hello world!");

如何控制alert中的换行

alert("hello\nworld");

解释document.getElementById("ElementID").style.fontSize="1.5em"

设置id为ElementID的元素的字体大小为1.5个相对单位

Em为相对长度单位。相对于当前对象内文本的字体尺寸。

如当前对行内文本的字体尺寸未被人为设置，则相对于浏览器的默认字体尺寸。

1em=16px

按照格式 xxxx年xx月xx日xx时xx分xx秒动态显示时间 要求不足10的补0

<script type="text/javascript" language="javascript">

function tick(){

var d=new Date();

var t=function(a){return a<10?"0"+a:a;}

Clock.innerHTML=d.getFullYear()+"年"+t(d.getMonth()+1)+"月"+t(d.getDate())+"日"+t(d.getHours())+"时"

+t(d.getMinutes())+"分"+t(d.getSeconds())+"秒";

window.setTimeout("tick()",1000);

}

window.onload=tick;

</script>

<body>

<div id="Clock"></div>

</body>

编写一个方法 去掉一个数组的重复元素

Array.prototype.strip=function(){

if(this.length<2) return [this[0]]||[];

var arr=[];

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

arr.push(this.splice(i--,1));//将本数组中第一个元素取出放入到数组arr中

for(var j=0;j<this.length;j++){

if(this[j]==arr[arr.length-1]){

this.splice(j--,1);//删除本数组中与数组arr中最后一个元素相同的元素

}

}

}

return arr;

}

var arr=["abc",85,"abc",85,8,8,1,2,5,4,7,8];

alert(arr.strip());

说出3条以上ff和ie的脚本兼容问题

IE有children，FF没有；

IE有parentElement，FF没有；

IE有 innerText,outerText,outerHTML，FF没有；

IE有数据岛，FF没有；

FF有

HTMLElement,HTMLDivElement,XMLDocument,DocumentFragment,Node,Event,Element 等等，IE没有；

IE跟FF创建HttpRequest实例的方法不一样

DIV中border、margin和padding的区别和用法

边框属性(border)用来设定一个元素的边线

外边距属性(margin)是用来设置一个元素所占空间的边缘到相邻元素之间的距离

内边距属性(padding)是用来设置元素内容到元素边界的距离

为Array写一个indexof方法

Array.prototype.indexOf = function(e){

for(var i=0,j; j=this[i]; i++){

if(j==e){return i;}

}

return -1;

}

Array.prototype.lastIndexOf = function(e){

for(var i=this.length-1,j; j=this[i]; i--){

if(j==e){return i;}

}

return -1;

}

var arr=[1,2,3,4,5];

alert(arr.indexOf(5));

克隆

浅复制(影子克隆):只复制对象的基本类型,对象类型,仍属于原来的引用

深复制(深度克隆):不紧复制对象的基本类,同时也复制原对象中的对象.就是说完全是新对象产生的

Object.prototype.Clone = function(){

var objClone;

if( this.constructor == Object )

objClone = new this.constructor();

else

objClone = new this.constructor(this.valueOf());

for ( var key in this ){

if ( objClone[key] != this[key] ){

if ( typeof(this[key]) == 'object' ){

objClone[key] = this[key].Clone();

}else{

objClone[key] = this[key];

}

}

}

objClone.toString = this.toString;

objClone.valueOf = this.valueOf;

return objClone;

}

兼容 IE 和 FF 的换行 CSS 推荐样式

word-wrap:break-word; overflow:hidden;

word-wrap是控制换行的。使用break-word时，是将强制换行。中文没有任何问题，英文语句也没问题。但是对于长串的英文，就不起作用。

word-break是控制是否断词的。

normal是默认情况，英文单词不被拆开。

break-all，是断开单词。在单词到边界时，下个字母自动到下一行。主要解决了长串英文的问题。

keep-all，是指Chinese, Japanese, and Korean不断词。即只用此时，不用word-wrap，中文就不会换行了。（英文语句正常。）

手型Cursor的兼容IE和FF写法

cursor:pointer

<img>元素的alt和title有什么异同？

alt作为图片的替代文字出现，title是图片的解释文字

图片存在

只有alt 图片的解释文字

只有title 图片的解释文字

两者都有 图片的解释文字

两者都没有 图片既没有替代文字，也没有解释文字

图片不存在

只有alt 图片既有替代文字，又有解释文字

只有title 图片没有替代文字，只有解释文字

两者都有 图片既有替代文字，又有解释文字

两者都没有 图片既没有替代文字，也没有解释文字

当然不同的浏览器处理方式也会不一样

border-color-left、marin-left、-moz-viewport改写成JavaScript格式

border-color-left:borderLeftColor

marin-left:marinLeft

-moz-viewport:MozViewport

用css、html编写一个两列布局的网页，要求右侧宽度为200px，左侧自动扩展。

CSS:

#right{

　　 float:right;

　　 width:200px;

}

#left{

　　 marin-right:200px;

}

HTML:

<body>

　　 <div id=”right”>…</div>

　　 <div id=”left”>…</div>

</body>

Linux题目：批量删除当前目录下后缀名为.c的文件，如a.c、b.c

rm \*.c

如何提高网页的运行速度

内容与形式分离，模块化开发，优化CSS

减少页面文档大小

尽量减少图片的使用或注意图片的大小，优化图片：格式、质量、图片长宽标志

减少响应的次数，用Ajax

网址后面加一个“/”

按要求写一个简单的ajax示例

<body>

<div id="load">数据正在加载......</div>

<script type="text/javascript">

var Browser={/\*\*Browser对象用于检测浏览器，其中用到了IE的条件编译\*/

isFF:window.navigator.appName.toUpperCase().indexOf("NETSCAPE")!=-1?true:false,

isOpera:window.navigator.appName.toUpperCase().indexOf("OPERA")!=-1?true:false

};

Function.prototype.bind=function(object){

var \_this=this;

return function(){

\_this.apply(object,arguments);

}

}

function HttpRequest(){

this.async=true;

this.cache=false;

this.xmlhttp=function(){

if(Browser.isFF&&window.XMLHttpRequest){

try{

return new XMLHttpRequest();

}catch(e){}

}else if(Browser.isIE&&window.ActiveXObject){

var Version = ["Msxml2.XMLHTTP.6.0","Msxml2.XMLHTTP.5.0","Msxml2.XMLHTTP.4.0","Msxml2.XMLHTTP.3.0",+

"Msxml2.XMLHTTP.2.6","Msxml2.XMLHTTP","Microsoft.XMLHTTP.1.0","Microsoft.XMLHTTP.1","Microsoft.XMLHTTP"];

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

try{

return new ActiveXObject(Version[i]);

}catch(e){}

}

}

}()||false;

}HttpRequest.prototype={

send:function(object,url,callback){

if(!this.xmlhttp) return;

this.xmlhttp.open(object?"post":"get",url,!!this.async);

if(object) this.xmlhttp.setRequestHeader("content-type","application/x-www-form-urlencoded");

if(!this.cache){

this.xmlhttp.setRequestHeader("No-Cache","1");

this.xmlhttp.setRequestHeader("Pragma","no-cache");

this.xmlhttp.setRequestHeader("Cache-Control","no-cache");

this.xmlhttp.setRequestHeader("Expire","0");

this.xmlhttp.setRequestHeader("Last-Modified","Wed, 1 Jan 1997 00:00:00 GMT");

this.xmlhttp.setRequestHeader("If-Modified-Since","-1");

}

if(!this.callback) this.callback=callback;

if(!this.async){

if(typeof(this.callback)=="string"){

eval(this.callback);

}else if(typeof(this.callback)=="function"){

this.callback(this.xmlhttp);

}

}else{

this.xmlhttp.onreadystatechange=function(){

if(this.xmlhttp.readyState==4){

if(this.xmlhttp.status==0||this.xmlhttp.status==200){

if(typeof(this.callback)=="string"){

eval(this.callback);

}else if(typeof(this.callback)=="function"){

this.callback(this.xmlhttp);

}

}

}

}.bind(this);

}

this.xmlhttp.send(object);

},abort:function(){

if(this.xmlhttp&&this.xmlhttp.abort) this.xmlhttp.abort();

}

}; //ajax类定义结束

new HttpRequest().send(null,"http://bbs.51js.com/index.php",function(r){

document.getElementById("load").innerHTML=r.responseText.match(/<img.\*?(?:\/)?>/img).join(""); });

</script>

</body>

**IE6、IE7、IE8、Firefox兼容性CSS HACK代码+示例**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

整理关于IE6、IE7、IE8、Firefox兼容性CSS HACK问题

1.区别IE和非IE浏览器CSS HACK代码

#divcss5{

background:blue; /\*非IE 背景藍色\*/

background:red \9; /\*IE6、IE7、IE8背景紅色\*/

}

2.区别IE6,IE7,IE8,FF CSS HACK

【区别符号】：「\9」、「\*」、「\_」

【示例】：

#divcss5{

background:blue; /\*Firefox 背景变蓝色\*/

background:red \9; /\*IE8 背景变红色\*/

\*background:black; /\*IE7 背景变黑色\*/

\_background:orange; /\*IE6 背景变橘色\*/

}

【说明】：因为IE系列浏览器可读「\9」，而IE6和IE7可读「\*」(米字号)，另外IE6可辨识「\_」(底线)，因此可以依照顺序写下来，就会让浏览器正确的读取到自己看得懂得CSS语法，所以就可以有效区分IE各版本和非IE浏览器(像是Firefox、Opera、Google Chrome、Safari等)。

3.区别IE6、IE7、Firefox (EXP 1)

【区别符号】：「\*」、「\_」

【示例】：

#divcss5{

background:blue; /\*Firefox背景变蓝色\*/

\*background:black; /\*IE7 背景变黑色\*/

\_background:orange; /\*IE6 背景变橘色\*/

}

【说明】：IE7和IE6可读「\*」(米字号)，IE6又可以读「\_」(底线)，但是IE7却无法读取「\_」，至于Firefox(非IE浏览器)则完全无法辨识「\*」和「\_」，因此就可以透过这样的差异性来区分IE6、IE7、Firefox

4.区别IE6、IE7、Firefox (EXP 2)

【区别符号】：「\*」、「!important」

【示例】：

#divcss5{

background:blue; /\*Firefox 背景变蓝色\*/

\*background:green !important; /\*IE7 背景变绿色\*/

\*background:orange; /\*IE6 背景变橘色\*/

}

【说明】：IE7可以辨识「\*」和「!important」，但是IE6只可以辨识「\*」，却无法辨识「!important」，至于Firefox可以读取「!important」但不能辨识「\*」因此可以透过这样的差异来有效区隔IE6、IE7、Firefox。

5.区别IE7、Firefox

【区别符号】：「\*」、「!important」

【示例】：

#divcss5{

background:blue; /\*Firefox 背景变蓝色\*/

\*background:green !important; /\*IE7 背景变绿色\*/

}

【说明】：因为Firefox可以辨识「!important」但却无法辨识「\*」，而IE7则可以同时看懂「\*」、「!important」，因此可以两个辨识符号来区隔IE7和Firefox。

6.区别IE6、IE7 (EXP 1)

【区别符号】：「\*」、「\_」

【示例】：

#tip {

\*background:black; /\*IE7 背景变黑色\*/

\_background:orange; /\*IE6 背景变橘色\*/

}

【说明】：IE7和IE6都可以辨识「\*」(米字号)，但IE6可以辨识「\_」(底线)，IE7却无法辨识，透过IE7无法读取「\_」的特性就能轻鬆区隔IE6和IE7之间的差异。

7.区别IE6、IE7 (EXP 2)

【区别符号】：「!important」

【示例】：

#divcss5{

background:black !important; /\*IE7 背景变黑色\*/

background:orange; /\*IE6 背景变橘色\*/

}

【说明】：因为IE7可读取「!important;」但IE6却不行，而CSS的读取步骤是从上到下，因此IE6读取时因无法辨识「!important」而直接跳到下一行读取CSS，所以背景色会呈现橘色。

8.区别IE6、Firefox

【区别符号】：「\_」

【示例】：

#divcss5{

background:black; /\*Firefox 背景变黑色\*/

\_background:orange; /\*IE6 背景变橘色\*/

}

【说明】：因为IE6可以辨识「\_」(底线)，但是Firefox却不行，因此可以透过这样的差异来区隔Firefox和IE6，有效达成CSS hack。

以上包括了IE6\IE8\IE7\火狐浏览器兼容问题及解决方法。

[HTML&& CSS]

1. Doctype? 严格模式与混杂模式-如何触发这两种模式，区分它们有何意义?

Doctype声明位于文档中的最前面的位置，处于标签之前。此标签可告知浏览器文档使用哪种 HTML 或 XHTML 规范。该标签可声明三种DTD 类型，分别表示严格版本、过渡版本以及基于框架的 HTML 文档。

当浏览器厂商开始创建与标准兼容的浏览器时，他们希望确保向后兼容性。为了实现这一点，他们创建了两种呈现模式：标准模式和混杂模式（quirks mode）。在标准模式中，浏览器根据规范呈现页面；在混杂模式中，页面以一种比较宽松的向后兼容的方式显示。混杂模式通常模拟老式浏览器（比如Microsoft IE 4和Netscape Navigator 4）的行为以防止老站点无法工作。

浏览器根据DOCTYPE是否存在以及使用的哪种DTD来选择要使用的呈现方法。如果XHTML文档包含形式完整的DOCTYPE，那么它一般以标准模式呈现。对于HTML 4.01文档，包含严格DTD的DOCTYPE常常导致页面以标准模式呈现。包含过渡DTD和URI的DOCTYPE也导致页面以标准模式呈现，但是有过渡DTD而没有URI会导致页面以混杂模式呈现。DOCTYPE不存在或形式不正确会导致HTML和XHTML文档以混杂模式呈现。

2. 行内元素有哪些？块级元素有哪些？CSS的盒模型？

行内元素有：a b span I bem img input select strong

块级元素有：div ul ol lidl dt dd h1 h2 h3 h4…p

盒模型：margin borderpadding width

3. CSS引入的方式有哪些? link和@import的区别是?

1. 使用 LINK标签

将样式规则写在.css的样式文件中，再以<link>标签引入。

<link rel=stylesheet type="text/css" href="example.css">

2. 使用@import引入

跟link方法很像，但必须放在<STYLE>...</STYLE> 中

<STYLE TYPE="text/css">

<!--

　　@import url(css/example.css);

-->

</STYLE>

3. 使用STYLE标签

将样式规则写在<STYLE>...</STYLE>标签之中。

<STYLE TYPE="text/css">

<!--

body {color: #666;background: #f0f0f0;font-size: 12px;}

td,p {color:#c00;font-size: 12px;}

-->

</STYLE>

4. 使用STYLE属性

将STYLE属性直接加在个别的元件标签里，<元件(标签) STYLE="性质(属性)1: 设定值1; 性质(属性)2: 设定值2; ...}

5. 使用<span></span>标记引入样式

<span style="font:12px/20px #000000;">cnwebshow.com</span>

两者区别：加载顺序的差别。当一个页面被加载的时候，link引用的CSS会同时被加载，而@import引用的CSS会等到页面全部被下载完再被加载。@import可以在css中再次引入其他样式表，比如可以创建一个主样式表，在主样式表中再引入其他的样式表，如：

main.css

———————-

@import“sub1.css”;

@import“sub2.css”;

这样做有一个缺点，会对网站服务器产生过多的HTTP请求，以前是一个文件，而现在却是两个或更多文件了，服务器的压力增大，浏览量大的网站还是谨慎使用。

4. CSS选择符有哪些？哪些属性可以继承？优先级算法如何计算？内联和important哪个优先级高？

通配选择符\* { sRules }

类型选择符E { sRules }

td { font-size:14px; width:120px; }

属性选择符

E [ attr ] { sRules }

E [ attr = value ] { sRules }

E [ attr ~= value ] { sRules }

E [ attr |= value ] { sRules }

h[title] { color: blue; }/\* 所有具有title属性的h对象 \*/

span[class=demo] { color: red; }

div[speed="fast"][dorun="no"] { color: red; }

a[rel~="copyright"] { color:black; }

包含选择符E1 E2 { sRules }

table td { font-size:14px; }

子对象选择符E1 > E2 { sRules }

div ul>li p { font-size:14px; }

ID选择符#ID { sRules }

类选择符E.className { sRules }

选择符分组

E1 , E2 , E3 { sRules }

伪类及伪对象选择符

E : Pseudo-Classes { sRules }

( Pseudo-Classes )[:link :hover :active :visited :focus :first-child :first :left :right :lang]

E : Pseudo-Elements { sRules }

( Pseudo-Elements )[:first-letter :first-line :before :after]

可以继承的有：font-size font-family color

不可继承的一般有：border padding margin background-color width height等

============================

关于CSS specificity

CSS 的specificity 特性或称非凡性，它是衡量一个衡量CSS值优先级的一个标准，既然作为标准，就具有一套相关的判定规定及计算方式，specificity用一个四位的数 字串(CSS2是三位)来表示，更像四个级别，值从左到右，左面的最大，一级大于一级，数位之间没有进制，级别之间不可超越。

在多个选择符应用于同一个元素上那么Specificity值高的最终获得优先级。

使用!important可以改变优先级别为最高，其次是style对象，然后是id> class >tag ，另外在同级样式按照申明的顺序后出现的样式具有高优先级。

5. 前端页面由哪三层构成，分别是什么？作用是什么？

网页分成三个层次，即：结构层、表示层、行为层。

网页的结构层（structurallayer）由HTML 或XHTML 之类的标记语言负责创建。标签，也就是那些出现在尖括号里的单词，对网页内容的语义含义做出了描述，但这些标签不包含任何关于如何显示有关内容的信息。例如，P标签表达了这样一种语义：“这是一个文本段。”

网页的表示层（presentationlayer）由CSS 负责创建。CSS 对“如何显示有关内容”的问题做出了回答。

网页的行为层（behaviorlayer）负责回答“内容应该如何对事件做出反应”这一问题。这是Javascript 语言和DOM 主宰的领域。

6. css的基本语句构成是？

选择器{属性1:值1;属性2:值2;……}

7. 你做的页面在哪些流览器测试过？这些浏览器的内核分别是什么?经常遇到的浏览器的兼容性有哪些？怎么会出现？解决方法是什么？

IE内核浏览器：360，傲游，搜狗，世界之窗，腾讯TT

非IE内核浏览器：firefoxopera safari chrome

8. 如何居中一个浮动元素?

设置容器的浮动方式为相对定位，然后确定容器的宽高，比如宽500 高 300 的层，然后设置层的外边距。

div{Width:500px;height:300px;Margin: -150px 0 0-250px;position:relative;left:50%;top:50%;}

9. 有没有关注HTML5和CSS3?如有请简单说一些您对它们的了解情况！

HTML5标签的改变：<header>,<footer>, <dialog>, <aside>, <figure>, <section> 等

IE9以上开始支持

CSS3实现圆角，阴影，对文字加特效，增加了更多的CSS选择器。

10. 如果让你来制作一个访问量很高的大型网站，你会如何来管理所有CSS文件、JS与图片？

11. 你对前端界面工程师这个职位是怎么样理解的？它的前景会怎么样？

以下为Web前端开发笔试题集锦之Javascript篇，移步HTML/CSS篇

1, 判断字符串是否是这样组成的，第一个必须是字母，后面可以是字母、数字、下划线，总长度为5-20

var reg = /^[a-zA-Z][a-zA-Z\_0-9]{4,19}$/;

reg.test("a1a\_\_a1a\_\_a1a\_\_a1a\_\_");

2，截取字符串abcdefg的efg

var str = "abcdefg";

if (/efg/.test(str)) {

var efg = str.substr(str.indexOf("efg"), 3);

alert(efg);

}

3，判断一个字符串中出现次数最多的字符，统计这个次数

//将字符串的字符保存在一个hash table中，key是字符，value是这个字符出现的次数

var str = "abcdefgaddda";

var obj = {};

for (var i = 0, l = str.length; i &lt; l; i++) {

var key = str[i];

if (!obj[key]) {

obj[key] = 1;

} else {

obj[key]++;

}

}

/\*遍历这个hash table，获取value最大的key和value\*/

var max = -1;

var max\_key = "";

var key;

for (key in obj) {

if (max < obj[key]) {

max = obj[key];

max\_key = key;

}

}

alert("max:"+max+" max\_key:"+max\_key);

4，IE与FF脚本兼容性问题

(1) window.event：

表示当前的事件对象，IE有这个对象，FF没有，FF通过给事件处理函数传递事件对象

(2) 获取事件源

IE用srcElement获取事件源，而FF用target获取事件源

(3) 添加，去除事件

IE：element.attachEvent(“onclick”, function) element.detachEvent(“onclick”, function)

FF：element.addEventListener(“click”, function, true) element.removeEventListener(“click”, function, true)

(4) 获取标签的自定义属性

IE：div1.value或div1[“value”]

FF：可用div1.getAttribute(“value”)

(5) document.getElementByName()和document.all[name]

IE；document.getElementByName()和document.all[name]均不能获取div元素

FF：可以

(6) input.type的属性

IE：input.type只读

FF：input.type可读写

(7) innerText textContent outerHTML

IE：支持innerText, outerHTML

FF：支持textContent

(8) 是否可用id代替HTML元素

IE：可以用id来代替HTML元素

FF：不可以

这里只列出了常见的，还有不少，更多的介绍可以参看JavaScript在IE浏览器和Firefox浏览器中的差异总结

5，规避javascript多人开发函数重名问题

(1) 可以开发前规定命名规范，根据不同开发人员开发的功能在函数前加前缀

(2) 将每个开发人员的函数封装到类中，调用的时候就调用类的函数，即使函数重名只要类名不重复就ok

6，javascript面向对象中继承实现

javascript面向对象中的继承实现一般都使用到了构造函数和Prototype原型链，简单的代码如下：

function Animal(name) {

this.name = name;

}

Animal.prototype.getName = function() {alert(this.name)}

function Dog() {};

Dog.prototype = new Animal("Buddy");

Dog.prototype.constructor = Dog;

var dog = new Dog();

7，FF下面实现outerHTML

FF不支持outerHTML，要实现outerHTML还需要特殊处理

思路如下：

在页面中添加一个新的元素A，克隆一份需要获取outerHTML的元素，将这个元素append到新的A中，然后获取A的innerHTML就可以了。

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />

<title>获取outerHMTL</title>

<style>

div{ background:#0000FF;width:100px;height:100px;}

span{ background:#00FF00;width:100px;height:100px;}

p{ background:#FF0000;width:100px;height:100px;}

</style>

</head>

<body>

<div id="a"><span>SPAN</span>DIV</div>

<span>SPAN</span>

<p>P</p>

<script type="text/javascript">

function getOuterHTML(id){

var el = document.getElementById(id);

var newNode = document.createElement("div");

document.appendChild(newNode);

var clone = el.cloneNode(true);

newNode.appendChild(clone);

alert(newNode.innerHTML);

document.removeChild(newNode);

}

getOuterHTML("a");

</script>

</body>

</html>

8，编写一个方法 求一个字符串的字节长度

假设：

一个英文字符占用一个字节，一个中文字符占用两个字节

function GetBytes(str){

var len = str.length;

var bytes = len;

for(var i=0; i<len; i++){

if (str.charCodeAt(i) > 255) bytes++;

}

return bytes;

}

alert(GetBytes("你好,as"));

9，编写一个方法 去掉一个数组的重复元素

var arr = [1 ,1 ,2, 3, 3, 2, 1];

Array.prototype.unique = function(){

var ret = [];

var o = {};

var len = this.length;

for (var i=0; i<len; i++){

var v = this[i];

if (!o[v]){

o[v] = 1;

ret.push(v);

}

}

return ret;

};

alert(arr.unique());

10，写出3个使用this的典型应用

（1）在html元素事件属性中使用，如

<input type=”button” onclick=”showInfo(this);” value=”点击一下”/>

（2）构造函数

function Animal(name, color) {

this.name = name;

this.color = color;

}

（3）

<input type="button" id="text" value="点击一下" />

<script type="text/javascript">

var btn = document.getElementById("text");

btn.onclick = function() {

alert(this.value); //此处的this是按钮元素

}

</script>

（4）CSS expression表达式中使用this关键字

<table width="100px" height="100px">

<tr>

<td>

<div style="width:expression(this.parentNode.width);">div element</div>

</td>

</tr>

</table>

12，如何显示/隐藏一个DOM元素？

el.style.display = "";

el.style.display = "none";

el是要操作的DOM元素

13，JavaScript中如何检测一个变量是一个String类型？请写出函数实现

String类型有两种生成方式：

(1)Var str = “hello world”;

(2)Var str2 = new String(“hello world”);

function IsString(str){

return (typeof str == "string" || str.constructor == String);

}

var str = "";

alert(IsString(1));

alert(IsString(str));

alert(IsString(new String(str)));

14，网页中实现一个计算当年还剩多少时间的倒数计时程序，要求网页上实时动态显示“××年还剩××天××时××分××秒”

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

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

<title>倒计时</title>

</head>

<body>

<input type="text" value="" id="input" size="1000"/>

<script type="text/javascript">

function counter() {

var date = new Date();

var year = date.getFullYear();

var date2 = new Date(year, 12, 31, 23, 59, 59);

var time = (date2 - date)/1000;

var day =Math.floor(time/(24\*60\*60))

var hour = Math.floor(time%(24\*60\*60)/(60\*60))

var minute = Math.floor(time%(24\*60\*60)%(60\*60)/60);

var second = Math.floor(time%(24\*60\*60)%(60\*60)%60);

var str = year + "年还剩"+day+"天"+hour+"时"+minute+"分"+second+"秒";

document.getElementById("input").value = str;

}

window.setInterval("counter()", 1000);

</script>

</body>

</html>

15，补充代码，鼠标单击Button1后将Button1移动到Button2的后面<div> <input type=”button” id =”button1″ value=”1″ onclick=”???”> <input type=”button” id =”button2″ value=”2″ /”> </div>

<div>

<input type="button" id ="button1" value="1" onclick="moveBtn(this);">

<input type="button" id ="button2" value="2" />

</div>

<script type="text/javascript">

function moveBtn(obj) {

var clone = obj.cloneNode(true);

var parent = obj.parentNode;

parent.appendChild(clone);

parent.removeChild(obj);

}

</script>

16，JavaScript有哪几种数据类型

简单：Number，Boolean，String，Null，Undefined

复合：Object，Array，Function

17，下面css标签在JavaScript中调用应如何拼写，border-left-color，-moz-viewport

borderLeftColor

mozViewport

18，JavaScript中如何对一个对象进行深度clone

function cloneObject(o) {

if(!o || 'object' !== typeof o) {

return o;

}

var c = 'function' === typeof o.pop ? [] : {};

var p, v;

for(p in o) {

if(o.hasOwnProperty(p)) {

v = o[p];

if(v && 'object' === typeof v) {

c[p] = Ext.ux.clone(v);

}

else {

c[p] = v;

}

}

}

return c;

};

19，如何控制alert中的换行

\n alert(“p\np”);

20，请实现，鼠标点击页面中的任意标签，alert该标签的名称．（注意兼容性）

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />

<title>鼠标点击页面中的任意标签，alert该标签的名称</title>

<style>

div{ background:#0000FF;width:100px;height:100px;}

span{ background:#00FF00;width:100px;height:100px;}

p{ background:#FF0000;width:100px;height:100px;}

</style>

<script type="text/javascript">

document.onclick = function(evt){

var e = window.event || evt;

var tag = e["target"] || e["srcElement"];

alert(tag.tagName);

};

</script>

</head>

<body>

<div id="div"><span>SPAN</span>DIV</div>

<span>SPAN</span>

<p>P</p>

</body>

</html>

21，请编写一个JavaScript函数 parseQueryString，它的用途是把URL参数解析为一个对象，如：

var url = “http://witmax.cn/index.php?key0=0&key1=1&key2=2″;

function parseQueryString(url){

var params = {};

var arr = url.split("?");

if (arr.length <= 1)

return params;

arr = arr[1].split("&");

for(var i=0, l=arr.length; i<l; i++){

var a = arr[i].split("=");

params[a[0]] = a[1];

}

return params;

}

var url = "http://witmax.cn/index.php?key0=0&key1=1&key2=2";

var ps = parseQueryString(url);

alert(ps["key1"]);

22，ajax是什么? ajax的交互模型? 同步和异步的区别? 如何解决跨域问题?

Ajax是多种技术组合起来的一种浏览器和服务器交互技术，基本思想是允许一个互联网浏览器向一个远程页面/服务做异步的http调用，并且用收到的数据来更新一个当前web页面而不必刷新整个页面。该技术能够改进客户端的体验。包含的技术：

XHTML：对应W3C的XHTML规范，目前是XHTML1.0。

CSS：对应W3C的CSS规范，目前是CSS2.0

DOM：这里的DOM主要是指HTML DOM，XML DOM包括在下面的XML中

JavaScript：对应于ECMA的ECMAScript规范

XML：对应W3C的XML DOM、XSLT、XPath等等规范

XMLHttpRequest：对应WhatWG的Web Applications1.0规范（http://whatwg.org/specs/web-apps/current-work/）

AJAX交互模型

同步：脚本会停留并等待服务器发送回复然后再继续

异步：脚本允许页面继续其进程并处理可能的回复

跨域问题简单的理解就是因为JS同源策略的限制，a.com域名下的JS无法操作b.com或c.a.com下的对象，具体场景如下：

PS：（1）如果是端口或者协议造成的跨域问题前端是无能为力的

(2) 在跨域问题上，域仅仅通过URL的首部来识别而不会尝试判断相同的IP地址对应的域或者两个域是否对应一个IP

前端对于跨域的解决办法：

(1) document.domain+iframe

(2) 动态创建script标签

23，什么是闭包？下面这个ul，如何点击每一列的时候alert其index?

<ul id=”test”>

<li>这是第一条</li>

<li>这是第二条</li>

<li>这是第三条</li>

</ul>

内部函数被定义它的函数的外部区域调用的时候就产生了闭包。

(function A() {

var index = 0;

var ul = document.getElementById("test");

var obj = {};

for (var i = 0, l = ul.childNodes.length; i < l; i++) {

if (ul.childNodes[i].nodeName.toLowerCase() == "li") {

var li = ul.childNodes[i];

li.onclick = function() {

index++;

alert(index);

}

}

}

})();

24，请给出异步加载js方案，不少于两种

默认情况javascript是同步加载的，也就是javascript的加载时阻塞的，后面的元素要等待javascript加载完毕后才能进行再加载，对于一些意义不是很大的javascript，如果放在页头会导致加载很慢的话，是会严重影响用户体验的。

异步加载方式：

(1) defer，只支持IE

(2) async：

(3) 创建script，插入到DOM中，加载完毕后callBack，见代码：

function loadScript(url, callback){

var script = document.createElement("script")

script.type = "text/javascript";

if (script.readyState){ //IE

script.onreadystatechange = function(){

if (script.readyState == "loaded" ||

script.readyState == "complete"){

script.onreadystatechange = null;

callback();

}

};

} else { //Others: Firefox, Safari, Chrome, and Opera

script.onload = function(){

callback();

};

}

script.src = url;

document.body.appendChild(script);

}

25，请设计一套方案，用于确保页面中JS加载完全。

var n = document.createElement("script");

n.type = "text/javascript";

//以上省略部分代码

//ie支持script的readystatechange属性(IE support the readystatechange event for script and css nodes)

if(ua.ie){

n.onreadystatechange = function(){

var rs = this.readyState;

if('loaded' === rs || 'complete'===rs){

n.onreadystatechange = null;

f(id,url); //回调函数

}

};

//省略部分代码

//safari 3.x supports the load event for script nodes(DOM2)

n.addEventListener('load',function(){

f(id,url);

});

//firefox and opera support onload(but not dom2 in ff) handlers for

//script nodes. opera, but no ff, support the onload event for link

//nodes.

}else{

n.onload = function(){

f(id,url);

};

}

26，js中如何定义class,如何扩展prototype?

Ele.className = “\*\*\*”; //\*\*\*在css中定义，形式如下：.\*\*\* {…}

A.prototype.B = C;

A是某个构造函数的名字

B是这个构造函数的属性

C是想要定义的属性的值

27，如何添加html元素的事件,有几种方法.

（1） 为HTML元素的事件属性赋值

（2） 在JS中使用ele.on\*\*\* = function() {…}

（3） 使用DOM2的添加事件的方法 addEventListener或attachEvent

28，documen.write和 innerHTML的区别

document.write只能重绘整个页面

innerHTML可以重绘页面的一部分

29，多浏览器检测通过什么？

（1） navigator.userAgent

（2） 不同浏览器的特性，如addEventListener

30，js的基础对象有那些, window和document的常用的方法和属性列出来

String,Number,Boolean

Window:

方法：setInterval,setTimeout,clearInterval,clearTimeout,alert,confirm,open

属性：name,parent,screenLeft,screenTop,self,top,status

Document

方法：createElement,execCommand,getElementById,getElementsByName,getElementByTagName,write,writeln

属性：cookie,doctype,domain,documentElement,readyState,URL,

31，前端开发的优化问题

（1） 减少http请求次数：css spirit,data uri

（2） JS，CSS源码压缩

（3） 前端模板 JS+数据，减少由于HTML标签导致的带宽浪费，前端用变量保存AJAX请求结果，每次操作本地变量，不用请求，减少请求次数

（4） 用innerHTML代替DOM操作，减少DOM操作次数，优化javascript性能

（5） 用setTimeout来避免页面失去响应

（6） 用hash-table来优化查找

（7） 当需要设置的样式很多时设置className而不是直接操作style

（8） 少用全局变量

（9） 缓存DOM节点查找的结果

（10） 避免使用CSS Expression

（11） 图片预载

（12） 避免在页面的主体布局中使用table，table要等其中的内容完全下载之后才会显示出来，显示比div+css布局慢

32，如何控制网页在网络传输过程中的数据量

启用GZIP压缩

保持良好的编程习惯，避免重复的CSS，JavaScript代码，多余的HTML标签和属性

33，Flash、Ajax各自的优缺点，在使用中如何取舍？

Ajax的优势

（1） 可搜索型

（2） 开放性

（3） 费用

（4） 易用性

（5） 易于开发

Flash的优势

（1） 多媒体处理

（2） 兼容性

（3） 矢量图形 比SVG，Canvas优势大很多

（4） 客户端资源调度，比如麦克风，摄像头

4用脚本写去除字符串的前后空格

String.prototype.trim = function(mode)

{//前后去空格

if (mode==’left’) {

return ((this.charAt(0) == ” “ && this.length > 0) ? this.slice(1).trim(’left’) : this);

} else

if (mode == ’right’) {

return ((this.charAt(this.length - 1) == ” “ && this.length > 0) ? this.slice(0, this.length - 1).trim(’right’) : this);

} else {

return this.trim(’left’).trim(’right’);

}

};

5算出字符串中出现次数最多的字符是什么，出现了多少次

<script type=”text/javascript”>

//<![CDATA[var str ="adadfdfseffserfefsefseeffffftsdg"; //命名一个变量放置给出的字符串

var maxLength = 0; //命名一个变量放置字母出现的最高次数并初始化为0

var result = ''; //命名一个变量放置结果输入

while( str != '' ){ //循环迭代开始，并判断字符串是否为空

oldStr = str; //将原始的字符串变量赋值给新变量

getStr = str.substr(0,1); //用字符串的substr的方法得到第一个字符（首字母）

eval("str = str.replace(/"+getStr+"/g,'')"); //详细如补充

if( oldStr.length-str.length > maxLength )

{ //判断原始的字符串的长度减去替代后字符串长度是否大于之前出现的最大的字符串长度

maxLength = oldStr.length-str.length; //两字符串长度相减得到最大的字符串长度

result = getStr + "=" + maxLength //返回最大的字符串结果（字母、出现次数）

}

}

alert(result) //弹出结果

//]]>

</script>

6写出3中使用this的典型应用

在HTML元素事件属性中inline方式使用this关键字：

7.最后一个问题是，如何制作一个combo选项，就是可以输入可以下拉菜单选择。

这个网上有很多成品，主要是通过绝对定位和相对定位结合的方式，改造原有的select 标签和input标签的方法。

附上javascript正则表达式的基本知识：

g: 全局匹配

i: 忽略大小写

^ 匹配一个输入或一行的开头，/^a/匹配”an A”，而不匹配”An a”

$ 匹配一个输入或一行的结尾，/a$/匹配”An a”，而不匹配”an A”

\* 匹配前面元字符0次或多次，/ba\*/将匹配b,ba,baa,baaa

+ 匹配前面元字符1次或多次，/ba\*/将匹配ba,baa,baaa

? 匹配前面元字符0次或1次，/ba\*/将匹配b,ba

(x) 匹配x保存x在名为$1…$9的变量中

x|y 匹配x或y

{n} 精确匹配n次

{n,} 匹配n次以上

{n,m} 匹配n-m次

[xyz] 字符集(character set)，匹配这个集合中的任一一个字符(或元字符)

[^xyz] 不匹配这个集合中的任何一个字符

[b] 匹配一个退格符

b 匹配一个单词的边界

B 匹配一个单词的非边界

cX 这儿，X是一个控制符，/cM/匹配Ctrl-M

d 匹配一个字数字符，/d/ = /[0-9]/

D 匹配一个非字数字符，/D/ = /[^0-9]/

n 匹配一个换行符

r 匹配一个回车符

s 匹配一个空白字符，包括n,r,f,t,v等

S 匹配一个非空白字符，等于/[^nfrtv]/

t 匹配一个制表符

v 匹配一个重直制表符

w 匹配一个可以组成单词的字符(alphanumeric，这是我的意译，含数字)，包括下划线，如[w]匹配”$5.98″中的5，等于[a-zA-Z0-9]

W 匹配一个不可以组成单词的字符，如[W]匹配”$5.98″中的$，等于[^a-zA-Z0-9

举例：验证email

var myReg = /^[\_a-z0-9]+@([\_a-z0-9]+.)+[a-z0-9]{2,3}$/;

if(myReg.test(strEmail)) return true;

**Date 对象属性**

FF: Firefox, IE: Internet Explorer

属性 描述 FF IE

constructor 返回对创建此对象的 Date 函数的引用。 1 4

prototype 使您有能力向对象添加属性和方法。 1 4

Date 对象方法

FF: Firefox, IE: Internet Explorer

方法 描述 FF IE

Date() 返回当日的日期和时间。 1 3

getDate() 从 Date 对象返回一个月中的某一天 (1 ~ 31)。 1 3

getDay() 从 Date 对象返回一周中的某一天 (0 ~ 6)。 1 3

getMonth() 从 Date 对象返回月份 (0 ~ 11)。 1 3

getFullYear() 从 Date 对象以四位数字返回年份。 1 4

getYear() 请使用 getFullYear() 方法代替。 1 3

getHours() 返回 Date 对象的小时 (0 ~ 23)。 1 3

getMinutes() 返回 Date 对象的分钟 (0 ~ 59)。 1 3

getSeconds() 返回 Date 对象的秒数 (0 ~ 59)。 1 3

getMilliseconds() 返回 Date 对象的毫秒(0 ~ 999)。 1 4

getTime() 返回 1970 年 1 月 1 日至今的毫秒数。 1 3

getTimezoneOffset() 返回本地时间与格林威治标准时间 (GMT) 的分钟差。 1 3

getUTCDate() 根据世界时从 Date 对象返回月中的一天 (1 ~ 31)。 1 4

getUTCDay() 根据世界时从 Date 对象返回周中的一天 (0 ~ 6)。 1 4

getUTCMonth() 根据世界时从 Date 对象返回月份 (0 ~ 11)。 1 4

getUTCFullYear() 根据世界时从 Date 对象返回四位数的年份。 1 4

getUTCHours() 根据世界时返回 Date 对象的小时 (0 ~ 23)。 1 4

getUTCMinutes() 根据世界时返回 Date 对象的分钟 (0 ~ 59)。 1 4

getUTCSeconds() 根据世界时返回 Date 对象的秒钟 (0 ~ 59)。 1 4

getUTCMilliseconds() 根据世界时返回 Date 对象的毫秒(0 ~ 999)。 1 4

parse() 返回1970年1月1日午夜到指定日期（字符串）的毫秒数。 1 3

setDate() 设置 Date 对象中月的某一天 (1 ~ 31)。 1 3

setMonth() 设置 Date 对象中月份 (0 ~ 11)。 1 3

setFullYear() 设置 Date 对象中的年份（四位数字）。 1 4

setYear() 请使用 setFullYear() 方法代替。 1 3

setHours() 设置 Date 对象中的小时 (0 ~ 23)。 1 3

setMinutes() 设置 Date 对象中的分钟 (0 ~ 59)。 1 3

setSeconds() 设置 Date 对象中的秒钟 (0 ~ 59)。 1 3

setMilliseconds() 设置 Date 对象中的毫秒 (0 ~ 999)。 1 4

setTime() 以毫秒设置 Date 对象。 1 3

setUTCDate() 根据世界时设置 Date 对象中月份的一天 (1 ~ 31)。 1 4

setUTCMonth() 根据世界时设置 Date 对象中的月份 (0 ~ 11)。 1 4

setUTCFullYear() 根据世界时设置 Date 对象中的年份（四位数字）。 1 4

setUTCHours() 根据世界时设置 Date 对象中的小时 (0 ~ 23)。 1 4

setUTCMinutes() 根据世界时设置 Date 对象中的分钟 (0 ~ 59)。 1 4

setUTCSeconds() 根据世界时设置 Date 对象中的秒钟 (0 ~ 59)。 1 4

setUTCMilliseconds() 根据世界时设置 Date 对象中的毫秒 (0 ~ 999)。 1 4

toSource() 返回该对象的源代码。 1 -

toString() 把 Date 对象转换为字符串。 1 4

toTimeString() 把 Date 对象的时间部分转换为字符串。 1 4

toDateString() 把 Date 对象的日期部分转换为字符串。 1 4

toGMTString() 请使用 toUTCString() 方法代替。 1 3

toUTCString() 根据世界时，把 Date 对象转换为字符串。 1 4

toLocaleString() 根据本地时间格式，把 Date 对象转换为字符串。 1 3

toLocaleTimeString() 根据本地时间格式，把 Date 对象的时间部分转换为字符串。 1 3

toLocaleDateString() 根据本地时间格式，把 Date 对象的日期部分转换为字符串。 1 3

UTC() 根据世界时返回 1997 年 1 月 1 日 到指定日期的毫秒数。 1 3

valueOf() 返回 Date 对象的原始值。 1 4

Array 对象属性

FF: Firefox, IE: Internet Explorer

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 描述 | FF | IE |
| [constructor](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_constructor_array.asp.htm) | 返回对创建此对象的数组函数的引用。 | 1 | 4 |
| index |  | 1 | 4 |
| input |  | 1 | 4 |
| [length](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_length_array.asp.htm) | 设置或返回数组中元素的数目。 | 1 | 4 |
| [prototype](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_prototype_array.asp.htm) | 使您有能力向对象添加属性和方法。 | 1 | 4 |

Array 对象方法

FF: Firefox, IE: Internet Explorer

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 描述 | FF | IE |
| [concat()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_concat_array.asp.htm) | 连接两个或更多的数组，并返回结果。 | 1 | 4 |
| [join()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_join.asp.htm) | 把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。 | 1 | 4 |
| [pop()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_pop.asp.htm) | 删除并返回数组的最后一个元素 | 1 | 5.5 |
| [push()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_push.asp.htm) | 向数组的末尾添加一个或更多元素，并返回新的长度。 | 1 | 5.5 |
| [reverse()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_reverse.asp.htm) | 颠倒数组中元素的顺序。 | 1 | 4 |
| [shift()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_shift.asp.htm) | 删除并返回数组的第一个元素 | 1 | 5.5 |
| [slice()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_slice_array.asp.htm) | 从某个已有的数组返回选定的元素 | 1 | 4 |
| [sort()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_sort.asp.htm) | 对数组的元素进行排序 | 1 | 4 |
| [splice()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_splice.asp.htm) | 删除元素，并向数组添加新元素。 | 1 | 5.5 |
| [toSource()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_tosource_array.asp.htm) | 返回该对象的源代码。 | 1 | - |
| [toString()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_toString_array.asp.htm) | 把数组转换为字符串，并返回结果。 | 1 | 4 |
| [toLocaleString()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_toLocaleString_array.asp.htm) | 把数组转换为本地数组，并返回结果。 | 1 | 4 |
| [unshift()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_unshift.asp.htm) | 向数组的开头添加一个或更多元素，并返回新的长度。 | 1 | 6 |
| [valueOf()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/jsref_valueof_array.asp.htm) | 返回数组对象的原始值 |  |  |

jQuery AJAX 请求

|  |  |
| --- | --- |
| 请求 | 描述 |
| $(selector).load(url,data,callback) | 把远程数据加载到被选的元素中 |
| $.ajax(options) | 把远程数据加载到 XMLHttpRequest 对象中 |
| $.get(url,data,callback,type) | 使用 HTTP GET 来加载远程数据 |
| $.post(url,data,callback,type) | 使用 HTTP POST 来加载远程数据 |
| $.getJSON(url,data,callback) | 使用 HTTP GET 来加载远程 JSON 数据 |
| $.getScript(url,callback) | 加载并执行远程的 JavaScript 文件 |

(selector) [jQuery 元素选择器语法](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/jquery/jquery_selectors.asp.htm)

(url) 被加载的数据的 URL（地址）

(data) 发送到服务器的数据的键/值对象

(callback) 当数据被加载时，所执行的函数

(type) 被返回的数据的类型 (html,xml,json,jasonp,script,text)

(options) 完整 AJAX 请求的所有键/值对选项

AJAX - 浏览器支持

AJAX 的要点是 XMLHttpRequest 对象。

不同的浏览器创建 XMLHttpRequest 对象的方法是有差异的。

IE 浏览器使用 ActiveXObject，而其他的浏览器使用名为 XMLHttpRequest 的 JavaScript 内建对象。

如需针对不同的浏览器来创建此对象，我们要使用一条 "try and catch" 语句。您可以在我们的 JavaScript 教程中阅读更多有关 [try 和 catch 语句](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/js/js_try_catch.asp.htm) 的内容。

让我们用这段创建 XMLHttpRequest 对象的 JavaScript 来更新一下我们的 "testAjax.htm" 文件：

<html>

<body>

<script type="text/javascript">

function ajaxFunction()

{

var xmlHttp;

try

{

// Firefox, Opera 8.0+, Safari

xmlHttp=new XMLHttpRequest();

}

catch (e)

{

// Internet Explorer

try

{

xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");

}

catch (e)

{

try

{

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

}

catch (e)

{

alert("您的浏览器不支持AJAX！");

return false;

}

}

}

}

</script>

<form name="myForm">

用户: <input type="text" name="username" />

时间: <input type="text" name="time" />

</form></body>

</html>

例子解释：

首先声明一个保存 XMLHttpRequest 对象的 xmlHttp 变量。

然后使用 XMLHttp=new XMLHttpRequest() 来创建此对象。这条语句针对 Firefox、Opera 以及 Safari 浏览器。假如失败，则尝试针对 Internet Explorer 6.0+ 的 xmlHttp=new ActiveXObject("Msxml2.XMLHTTP")，假如也不成功，则尝试针对 Internet Explorer 5.5+ 的 xmlHttp=new ActiveXObject("Microsoft.XMLHTTP")。

假如这三种方法都不起作用，那么这个用户所使用的浏览器已经太过时了，他或她会看到一个声明此浏览器不支持 AJAX 的提示。

注释：上面这些浏览器定制的代码很长，也很复杂。不过，每当您希望创建 XMLHttpRequest 对象时，这些代码就能派上用场，因此您可以在任何需要使用的时间拷贝粘贴这些代码。上面这些代码兼容所有的主流浏览器：Internet Explorer、Opera、Firefox 以及 Safari。

下一节为您展示如何使用 XMLHttpRequest 对象与服务器进行通信。

AJAX - 更多有关 XMLHttpRequest 对象的知识

在向服务器发送数据之前，我们有必要解释一下 XMLHttpRequest 对象的三个重要的属性。

onreadystatechange 属性

onreadystatechange 属性存有处理服务器响应的函数。下面的代码定义一个空的函数，可同时对 onreadystatechange 属性进行设置：

xmlHttp.onreadystatechange=function()

{

// 我们需要在这里写一些代码

}

readyState 属性

readyState 属性存有服务器响应的状态信息。每当 readyState 改变时，onreadystatechange 函数就会被执行。

这是 readyState 属性可能的值：

|  |  |
| --- | --- |
| 状态 | 描述 |
| 0 | 请求未初始化（在调用 open() 之前） |
| 1 | 请求已提出（调用 send() 之前） |
| 2 | 请求已发送（这里通常可以从响应得到内容头部） |
| 3 | 请求处理中（响应中通常有部分数据可用，但是服务器还没有完成响应） |
| 4 | 请求已完成（可以访问服务器响应并使用它） |

我们要向这个 onreadystatechange 函数添加一条 If 语句，来测试我们的响应是否已完成（意味着可获得数据）：

xmlHttp.onreadystatechange=function()

{

if(xmlHttp.readyState==4)

{

// 从服务器的response获得数据

}

}

responseText 属性

可以通过 responseText 属性来取回由服务器返回的数据。

在我们的代码中，我们将把时间文本框的值设置为等于 responseText：

xmlHttp.onreadystatechange=function()

{

if(xmlHttp.readyState==4)

{

document.myForm.time.value=xmlHttp.responseText;

}

}

下一节，我们会介绍如何向服务器请求数据！

AJAX - 向服务器发送一个请求

要想把请求发送到服务器，我们就需要使用 open() 方法和 send() 方法。

open() 方法需要三个参数。第一个参数定义发送请求所使用的方法（GET 还是 POST）。第二个参数规定服务器端脚本的 URL。第三个参数规定应当对请求进行异步地处理。

send() 方法可将请求送往服务器。如果我们假设 HTML 文件和 ASP 文件位于相同的目录，那么代码是这样的：

xmlHttp.open("GET","time.asp",true);

xmlHttp.send(null);

现在，我们必须决定何时执行 AJAX 函数。当用户在用户名文本框中键入某些内容时，我们会令函数“在幕后”执行。

<html>

<body>

<script type="text/javascript">

function ajaxFunction()

{

var xmlHttp;

try

{

// Firefox, Opera 8.0+, Safari

xmlHttp=new XMLHttpRequest();

}

catch (e)

{

// Internet Explorer

try

{

xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");

}

catch (e)

{

try

{

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

}

catch (e)

{

alert("您的浏览器不支持AJAX！");

return false;

}

}

}

xmlHttp.onreadystatechange=function()

{

if(xmlHttp.readyState==4)

{

document.myForm.time.value=xmlHttp.responseText;

}

}

xmlHttp.open("GET","time.asp",true);

xmlHttp.send(null);

}

</script>

<form name="myForm">

用户: <input type="text" name="username" onkeyup="ajaxFunction();" />

时间: <input type="text" name="time" />

</form>

</body>

</html>

通过微软的 XML 解析器来加载 XML

微软的 XML 解析器内建于 Internet Explorer 5 以及更高的版本中。

下面的 JavaScript 片段把一个 XML 文档载入解析器中：

var xmlDoc=new ActiveXObject("Microsoft.XMLDOM");

xmlDoc.async="false";

xmlDoc.load("note.xml");

例子解释：

上面代码的第一个行创建一个空的微软 XML 文档对象。

第二行关闭异步加载，这样确保在文档完全加载之前解析器不会继续脚本的执行。

第三行告知解析器加载名为 "note.xml" 的 XML 文档。

下面的 JavaScript 片段把字符串 txt 载入解析器：

var xmlDoc=new ActiveXObject("Microsoft.XMLDOM");

xmlDoc.async="false";

xmlDoc.loadXML(txt);

注释：[loadXML()](mk:@MSITStore:F:\学习\1W3CSchool.chm::/www.w3school.com.cn/xmldom/met_document_loadxml.asp.htm) 方法用于加载字符串（文本），load() 用于加载文件。

在 Firefox 及其他浏览器中的 XML 解析器

下面的 JavaScript 片段把 XML 文档 ("note.xml") 载入解析器：

var xmlDoc=document.implementation.createDocument("","",null);

xmlDoc.async="false";

xmlDoc.load("note.xml");

例子解释：

上面代码的第一个行创建一个空的 XML 文档对象。

第二行关闭异步加载，这样确保在文档完全加载之前解析器不会继续脚本的执行。

第三行告知解析器加载名为 "note.xml" 的 XML 文档。

下面的 JavaScript 片段把字符串 txt 载入解析器：

var parser=new DOMParser();

var doc=parser.parseFromString(txt,"text/xml");

例子解释：

上面代码的第一个行创建一个空的微软 XML 文档对象。

第二行告知解析器载入名为 txt 的字符串。

注释：Internet