# JS部分

### javascript有哪几种数据类型

六种基本数据类型

* undefined
* null
* string
* boolean
* Number
* Object

### 说说js中的闭包和变量作用域？

一、变量的作用域

要理解闭包,首先必须理解Javascript特殊的变量作用域。

变量的作用域无非就是两种：全局变量和局部变量。

Javascript语言的特殊之处,就在于函数内部可以直接读取全局变量。

　　var n=999;

　　function f1(){

　　　　alert(n);

　　}

　　f1(); // 999

另一方面,在函数外部自然无法读取函数内的局部变量。

　　function f1(){

　　　　var n=999;

　　}

　　alert(n); // error

需要注意:函数内部声明变量的时候,一定要使用var命令。

如果不用的话,你实际上声明了一个全局变量！

　　function f1(){

　　　　n=999;

　　}

　　f1();

　　alert(n); // 999

---------------------------------------分割线---------------------------------------

二、如何从外部读取局部变量？

出于种种原因,我们有时候需要得到函数内的局部变量。

但是正常情况下,这是办不到的；只有通过变通方法才能实现。

那就是在函数的内部,再定义一个函数。

　　function f1(){

　　　　var n=999;

　　　　function f2(){

　　　　　　alert(n); // 999

　　　　}

　　}

在上面的代码中,函数f2就被包括在函数f1内部;

这时f1内部的所有局部变量,对f2都是可见的;

但是反过来就不行,f2内部的局部变量,对f1就是不可见的;

这就是Javascript语言特有的"链式作用域"结构（chain scope）;

子对象会一级一级地向上寻找所有父对象的变量。

所以,父对象的所有变量,对子对象都是可见的,反之则不成立。

既然f2可以读取f1中的局部变量,那么只要把f2作为返回值

,我们不就可以在f1外部读取它的内部变量了吗！

　　function f1(){

　　　　var n=999;

　　　　function f2(){

　　　　　　alert(n);

　　　　}

　　　　return f2;

　　}

　　var result=f1();

　　result(); // 999

---------------------------------------分割线---------------------------------------

三、闭包的概念

上一节代码中的f2函数,就是闭包。

各种专业文献上的"闭包"（closure）定义非常抽象,很难看懂。

我的理解是,闭包就是能够读取其他函数内部变量的函数。

由于在Javascript语言中,只有函数内部的子函数才能读取局部变量,

因此可以把闭包简单理解成"定义在一个函数内部的函数"。

所以,在本质上,闭包就是将函数内部和函数外部连接起来的一座桥梁。

---------------------------------------分割线---------------------------------------

四、闭包的用途

闭包可以用在许多地方。它的最大用处有两个:

一个是前面提到的可以读取函数内部的变量;

另一个就是让这些变量的值始终保持在内存中。

怎么来理解这句话呢？请看下面的代码。

　　function f1(){

　　　　var n=999;

　　　　nAdd=function(){n+=1}

　　　　function f2(){

　　　　　　alert(n);

　　　　}

　　　　return f2;

　　}

　　var result=f1();

　　result(); // 999

　　nAdd();

　　result(); // 1000

在这段代码中,result实际上就是闭包f2函数。

它一共运行了两次,第一次的值是999,第二次的值是1000。

这证明了,函数f1中的局部变量n一直保存在内存中,并没有在f1调用后被自动清除。

为什么会这样呢？原因就在于f1是f2的父函数,而f2被赋给了一个全局变量,

这导致f2始终在内存中,而f2的存在依赖于f1,因此f1也始终在内存中,

不会在调用结束后,被垃圾回收机制（garbage collection）回收。

这段代码中另一个值得注意的地方,就是"nAdd=function(){n+=1}"这一行,

1、首先在nAdd前面没有使用var关键字,因此nAdd是一个全局变量,而不是局部变量;

2、其次,nAdd的值是一个匿名函数（anonymous function）,而这个匿名函数本身也是一个闭包,

所以nAdd相当于是一个setter,可以在函数外部对函数内部的局部变量进行操作。

---------------------------------------分割线---------------------------------------

五、使用闭包的注意点

1、由于闭包会使得函数中的变量都被保存在内存中,内存消耗很大,所以不能滥用闭包,

否则会造成网页的性能问题,在IE中可能导致内存泄露。解决方法是,在退出函数之前,

将不使用的局部变量全部删除。

2、闭包会在父函数外部,改变父函数内部变量的值。

所以,如果你把父函数当作对象（object）使用,把闭包当作它的公用方法（Public Method）,

把内部变量当作它的私有属性（private value）,这时一定要小心,不要随便改变父函数内部变量的值。

---------------------------------------分割线---------------------------------------

六、思考题

如果你能理解下面两段代码的运行结果,应该就算理解闭包的运行机制了。（首先考虑作用域this的指向）

代码片段一。

　　var name = "The Window";

　　var object = {

　　　　name : "My Object",

　　　　getNameFunc : function(){

　　　　　　return function(){

　　　　　　　　return this.name;

　　　　　　};

　　　　}

　　};

　　alert(object.getNameFunc()());

代码片段二。

　　var name = "The Window";

　　var object = {

　　　　name : "My Object",

　　　　getNameFunc : function(){

　　　　　　var that = this;

　　　　　　return function(){

　　　　　　　　return that.name;

　　　　　　};

　　　　}

　　};

alert(object.getNameFunc()());

****说说js中的事件机制？****

一、事件机制(事件流)

浏览器中的事件流意味着页面上可有不仅一个,甚至多个元素响应同一个事件。

而这一个或多个元素响应事件发生的先后顺序在各个浏览器（主要针对IE和Netscape）上是不同的。

冒泡型事件（Dubbed Bubbling）

IE上的解决方案就是冒泡型事件（Dubbed Bubbling）,冒泡型事件的基本思想是,

事件按照从最特定的事件目标到最不特定的事件目标（document对象）的顺序触发。

捕获型事件（Event Capturing）

相对IE4.0,Netscape4.0则使用的是捕获型事件的解决方案。

这个事件触发的过程则正好和冒泡相反——在捕获型事件中,事件从最不精确的对象（document对象）开始触发,

然后到最精确的对象。

DOM 事件流

这个事件流则是W3C制定一个标准规范,它同时支持两种事件流模式,

不过是先发生捕获型事件流,再发生冒泡型事件流。

DOM事件流最独特的是,它支持文本节点也触发事件（IE中这不支持）,

不过说实话,我现在还看不出来让文本节点支持事件有什么作用。

最后要说的是,根据最近大家在开发的实践过程中的运用,

我们一般都采取冒泡型的事件流触发方式,这点我们的IE做的比较成功。

至于原因,我想你可以通过上面的解释可以看出,毕竟我们给元素触发事件,

肯定是希望从我们最希望先触发（从最精确的）的那个开始。

---------------------------------------分割线---------------------------------------

二、事件绑定

事件处理函数/监听函数

1、在DOM元素中直接绑定

2、在JavaScript代码中绑定

3、绑定事件监听函数

具体实践：

IE中attachEvent(”NAME\_OF\_EVENT\_HANDLER”, fnHandler)给元素绑定事件；

而在支持DOM事件流的浏览器里,则使用addEventListener(”NAME\_OF\_EVENT\_HANDLER”, fnHandler, isCapture);

前面我控制FIREFOX中触发捕获事件流,就是通过设置isCapture（ture：捕获；false：冒泡）做到的;

function addEvent(obj,type,handle){

try{ // Chrome、FireFox、Opera、Safari、IE9.0及其以上版本

obj.addEventListener(type,handle,false);

}catch(e){

try{ // IE8.0及其以下版本

obj.attachEvent('on' + type,handle);

}catch(e){ // 现代浏览器

obj['on' + type] = handle;

}

}

}

哪些事件是支持冒泡,那些不支持？

基本上只有onload,unload,focus,blur,submit和change事件是不支持冒泡的；

自然像keydownkeypress,keyup,click,dbclick,mousedown,mouseout,mouseover,mouseup,mousemove支持冒泡,

那就是“Mouse and Key Events”支持冒泡;

---------------------------------------分割线---------------------------------------

三、事件委托

document.onclick,这个示例把事件委托放到了document上,即点击document就直接触发我们相应的事件。

document.onclick = function(event){

var event = event || window.event; //IE doesn't pass in the event object

var target = event.target || event.srcElement; //IE uses srcElement as the target

switch(target.id){

case "help-btn":

openHelp();

break;

case "save-btn":

saveDocument();

break;

case "undo-btn":

undoChanges();

break;

//如果有其元素需要处理点击事件, 只需要在这里添加不同的case分支就行。

}

};

优点：

从“处理速度”、“新增元素事件处理”和“内存消耗”三方面比较了“事件委托”和“事件绑定”的对比,

可以很容易看出,“事件委托”在“处理速度”和“内存消耗”上,有得天独厚的优势。

所以,在Web编程的时候,尤其在构建大型系统的时候,应该尽量考虑使用“事件委托”。

但是,“事件委托”并不是万能的；它也有一些弊端。下面我们在论述一下它的弊端。

缺点：

使用“事件委托”时,并不是说把事件委托给的元素越靠近顶层就越好。

事件冒泡的过程也需要耗时,越靠近顶层,事件的”事件传播链”越长,也就越耗时。

---------------------------------------分割线---------------------------------------

四、阻止事件冒泡和阻止事件默认行为

a.阻止事件冒泡

function stopBubble(e) {

//如果提供了事件对象,则这是一个非IE浏览器

if ( e && e.stopPropagation ){

e.stopPropagation(); //因此它支持W3C的stopPropagation()方法

}else{

window.event.cancelBubble = true; //否则,我们需要使用IE的方式来取消事件冒泡

}

}

b.当按键后,不希望按键继续传递给如HTML文本框对象时,可以取消返回值.即停止事件默认行为;

//阻止浏览器的默认行为

function stopDefault(e) {

//如果提供了事件对象,则这是一个非IE浏览器

if (e && e.preventDefault ){

e.preventDefault(); //阻止默认浏览器动作(W3C)

}else{

window.event.returnValue = false; //IE中阻止函数器默认动作的方式

}

}

****说说http请求和ajax 以及ajax跨域？****

1、HTTP请求

超文本传输协议（HTTP）的设计目的是保证客户机与服务器之间的通信。

HTTP 的工作方式是客户机与服务器之间的请求-应答协议。

两种 HTTP 请求方法：GET 和 POST:

a.GET 方法

请注意,查询字符串（名称/值对）是在 GET 请求的 URL 中发送的：

/test/demo\_form.asp？name1=value1&name2=value2

有关 GET 请求的其他一些注释：

GET 请求可被缓存

GET 请求保留在浏览器历史记录中

GET 请求可被收藏为书签

GET 请求不应在处理敏感数据时使用

GET 请求有长度限制

GET 请求只应当用于取回数据

------------------------------------------------------------------------------------------

b.POST 方法

请注意,查询字符串（名称/值对）是在 POST 请求的 HTTP 消息主体中发送的：

POST /test/demo\_form.asp HTTP/1.1

Host: w3schools.com

name1=value1&name2=value2

有关 POST 请求的其他一些注释：

POST 请求不会被缓存

POST 请求不会保留在浏览器历史记录中

POST 不能被收藏为书签

POST 请求对数据长度没有要求

------------------------------------------------------------------------------------------

c、http状态码有那些？分别代表是什么意思？

100-199 用于指定客户端应相应的某些动作。

200-299 用于表示请求成功。

300-399 用于已经移动的文件并且常被包含在定位头信息中指定新的地址信息。

400-499 用于指出客户端的错误。

400 语义有误,当前请求无法被服务器理解。

401 当前请求需要用户验证

403 服务器已经理解请求,但是拒绝执行它。

500-599 用于支持服务器错误。

503 服务不可用

---------------------------------------分割线---------------------------------------

2、AJAX

AJAX (异步 JavaScript 和 XML) 是个新产生的术语,专为描述JavaScript的两项强大性能.

这两项性能在多年来一直被网络开发者所忽略,

直到最近Gmail, Google Suggest和Google Maps的横空出世才使人们开始意识到其重要性.

这两项被忽视的性能是:

无需重新装载整个页面便能向服务器发送请求.

对XML文档的解析和处理

------------------------------------------------------------------------------------------

1、步骤1 浏览器发送一个HTTP请求

为了用JavaScript向服务器发送一个HTTP请求, 需要一个具备这种功能的类实例.

这样的类首先由Internet Explorer以ActiveX对象引入, 被称为XMLHTTP.

后来Mozilla, Safari 和其他浏览器纷纷仿效, 提供了XMLHttpRequest类,它支持微软的ActiveX对象所提供的方法和属性.

A、创建一个跨浏览器的这样的类实例(对象), 可以应用如下代码:

if (window.XMLHttpRequest) {

http\_request = new XMLHttpRequest(); // Mozilla, Safari, ...

}

else if (window.ActiveXObject)

{

http\_request = new ActiveXObject("Microsoft.XMLHTTP"); // IE

}

(上例对代码做了一定简化,这是为了解释如何创建XMLHTTP类实例. 实际的代码实例可参阅本篇步骤3.)

如果服务器的响应没有XML mime-type header,某些Mozilla浏览器可能无法正常工作.

为了解决这个问题, 如果服务器响应的header不是text/xml,可以调用其它方法修改该header.

http\_request = new XMLHttpRequest();

http\_request.overrideMimeType('text/xml');

------------------------------------------------------------------------------------------

B、决定当收到服务器的响应后,需要做什么.这需要告诉HTTP请求对象用哪一个JavaScript函数处理这个响应.

可以将对象的onreadystatechange属性设置为要使用的JavaScript的函数名,如下所示:

http\_request.onreadystatechange = nameOfTheFunction;

注意:在函数名后没有括号,也无需传递参数.另外还有一种方法,

你可以使用一个匿名函数来描述那些要对服务器返回的响应内容所进行的操作,如下所示:

http\_request.onreadystatechange = function(){

// do the thing

};

------------------------------------------------------------------------------------------

C、在定义了如何处理响应后,就要发送请求了.可以调用HTTP请求类的open()和send()方法, 如下所示:

http\_request.open('GET', 'http://www.example.org/some.file', true);

http\_request.send(null);

1、open()的第一个参数是HTTP请求方式 – GET, POST, HEAD 或任何服务器所支持的您想调用的方式；

按照HTTP规范,该参数要大写;否则,某些浏览器(如Firefox)可能无法处理请求；

有关HTTP请求方法的详细信息可参考http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

2、第二个参数是请求页面的URL.由于自身安全特性的限制,该页面不能为第三方域名的页面.

同时一定要保证在所有的页面中都使用准确的域名,否则调用open()会得到"permission denied"的错误提示.

一个常见的错误是访问站点时使用domain.tld,而当请求页面时,却使用www.domain.tld.

3、第三个参数设置请求是否为异步模式.如果是TRUE, JavaScript函数将继续执行,而不等待服务器响应.

------------------------------------------------------------------------------------------

这就是"AJAX"中的"A"：

1、如果第一个参数是"GET",通过open()第二个参数,传递参数,一般用于查询,jsonp的跨域就是这个原理,也只能解决get方式的请求；

2、如果第一个参数是"POST",send()方法的参数可以是任何想送给服务器的数据. 这时数据要以字符串的形式送给服务器,如下所示:

name=value&anothername=othervalue&so=on

------------------------------------------------------------------------------------------

2、步骤2 服务器收到请求后的响应

当发送请求时,要提供指定处理响应的JavaScript函数名.

http\_request.onreadystatechange = nameOfTheFunction;

我们来看看这个函数的功能是什么.首先函数会检查请求的状态.

如果状态值是4,就意味着一个完整的服务器响应已经收到了,您将可以处理该响应.

if (http\_request.readyState == 4) {

// everything is good, the response is received

} else {

// still not ready

}

readyState的取值如下:

0 (未初始化)

1 (正在装载)

2 (装载完毕)

3 (交互中)

4 (完成)

(Source)

接着,函数会检查HTTP服务器响应的状态值. 完整的状态取值可参见 W3C site. 我们着重看值为200 OK的响应:

if (http\_request.status == 200) {

// perfect!

} else {

// there was a problem with the request,

// for example the response may be a 404 (Not Found)

// or 500 (Internal Server Error)

response codes

}

在检查完请求的状态值和响应的HTTP状态值后, 您就可以处理从服务器得到的数据了.有两种方式可以得到这些数据:

http\_request.responseText – 以文本字符串的方式返回服务器的响应

http\_request.responseXML – 以XMLDocument对象方式返回响应.处理XMLDocument对象可以用JavaScript DOM函数

------------------------------------------------------------------------------------------

3、简单实例

我们现在将整个过程完整地做一次,发送一个简单的HTTP请求.

我们用JavaScript请求一个HTML文件, test.html, 文件的文本内容为"I'm a test.",

然后我们"alert()"test.html文件的内容.

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

var http\_request = false;

function makeRequest(url) {

http\_request = false;

if (window.XMLHttpRequest) { // Mozilla, Safari,...

http\_request = new XMLHttpRequest();

if (http\_request.overrideMimeType) {

http\_request.overrideMimeType('text/xml');

}

} else if (window.ActiveXObject) { // IE

try {

http\_request = new ActiveXObject("Msxml2.XMLHTTP");

}

catch (e) {

try {

http\_request = new ActiveXObject("Microsoft.XMLHTTP");

}catch(e){}

}

}

if (!http\_request) {

alert('Giving up :( Cannot create an XMLHTTP instance');

return false;

}

http\_request.onreadystatechange = alertContents;

http\_request.open('GET', url, true);

http\_request.send(null);

}

function alertContents() {

if (http\_request.readyState == 4) {

if (http\_request.status == 200) {

alert(http\_request.responseText);

}else{

alert('There was a problem with the request.');

}

}

}

</script>

<style>

.testbtn{ cursor: pointer; text-decoration: underline;}

</style>

<span class='testbtn' onclick="makeRequest('test.html')">Make a request</span>

本例中:

用户点击浏览器上的"请求"链接;

接着函数makeRequest()将被调用.其参数为HTML文件test.html在同一目录下;

这样就发起了一个请求.onreadystatechange的执行结果会被传送给alertContents();

alertContents()将检查服务器的响应是否成功地收到,如果是,就会"alert()"test.html文件的内容.

---------------------------------------分割线---------------------------------------

3、Ajax的跨域

概念：只要协议、域名、端口有任何一个不同,都被当作是不同的域。

URL 说明 是否允许通信

http://www.a.com/a.js http://www.a.com/b.js 同一域名下 允许

http://www.a.com/lab/a.js http://www.a.com/script/b.js 同一域名下不同文件夹 允许

http://www.a.com:8000/a.js http://www.a.com/b.js 同一域名,不同端口 不允许

http://www.a.com/a.js https://www.a.com/b.js 同一域名,不同协议 不允许

http://www.a.com/a.js http://70.32.92.74/b.js 域名和域名对应ip 不允许

http://www.a.com/a.js http://script.a.com/b.js 主域相同,子域不同 不允许

http://www.a.com/a.js http://a.com/b.js 同一域名,不同二级域名（同上） 不允许

http://www.cnblogs.com/a.js http://www.a.com/b.js 不同域名 不允许

对于端口和协议的不同,只能通过后台来解决。

------------------------------------------------------------------------------------------

A、跨域资源共享（CORS）

1、CORS（Cross-Origin Resource Sharing）跨域资源共享,定义了必须在访问跨域资源时,

浏览器与服务器应该如何沟通。

2、CORS背后的基本思想就是使用自定义的HTTP头部让浏览器与服务器进行沟通,

从而决定请求或响应是应该成功还是失败。

<script type="text/javascript">

var xhr = new XMLHttpRequest();

xhr.open("￼GET", "/trigkit4",true);

xhr.send();

</script>

以上的trigkit4是相对路径,如果我们要使用CORS,相关Ajax代码可能如下所示：

<script type="text/javascript">

var xhr = new XMLHttpRequest();

xhr.open("￼GET", "http://segmentfault.com/u/trigkit4/",true);

xhr.send();

</script>

代码与之前的区别就在于相对路径换成了其他域的绝对路径,也就是你要跨域访问的接口地址。

服务器端对于CORS的支持,主要就是通过设置Access-Control-Allow-Origin来进行的。

如果浏览器检测到相应的设置,就可以允许Ajax进行跨域的访问。

------------------------------------------------------------------------------------------

要解决跨域的问题,我们可以使用以下几种方法：

B、通过jsonp跨域

现在问题来了？什么是jsonp？

1、维基百科的定义是：JSONP（JSON with Padding）是资料格式 JSON 的一种“使用模式”,

可以让网页从别的网域要资料。

2、JSONP也叫填充式JSON,是应用JSON的一种新方法,只不过是被包含在函数调用中的JSON,

例如: callback({"name","trigkit4"});

JSONP由两部分组成：回调函数和数据：

回调函数是当响应到来时应该在页面中调用的函数,

而数据就是传入回调函数中的JSON数据。

在js中,我们直接用XMLHttpRequest请求不同域上的数据时,是不可以的。

但是,在页面上引入不同域上的js脚本文件却是可以的,jsonp正是利用这个特性来实现的。例如：

<script type="text/javascript">

function dosomething(jsondata){

//处理获得的json数据

}

</script>

<script src="http://example.com/data.php？callback=dosomething"></script>

js文件载入成功后会执行我们在url参数中指定的函数,并且会把我们需要的json数据作为参数传入。

所以jsonp是需要服务器端的页面进行相应的配合的。

<？php

$callback = $\_GET['callback'];//得到回调函数名

$data = array('a','b','c');//要返回的数据

echo $callback.'('.json\_encode($data).')';//输出

？>

最终,输出结果为：dosomething(['a','b','c']);

如果你的页面使用jquery,那么通过它封装的方法就能很方便的来进行jsonp操作了。

<script type="text/javascript">

$.getJSON('http://example.com/data.php？callback=？,function(jsondata)'){

//处理获得的json数据

});

</script>

jquery会自动生成一个全局函数来替换callback=？中的问号,之后获取到数据后又会自动销毁,

实际上就是起一个临时代理函数的作用。

$.getJSON方法会自动判断是否跨域：

1、不跨域的话,就调用普通的ajax方法；

2、跨域的话,则会以异步加载js文件的形式来调用jsonp的回调函数。

3、并且在请求完毕后可以通过调用callback的方式回传结果。

JSONP的优点是：

1、它不像XMLHttpRequest对象实现的Ajax请求那样受到同源策略的限制；

2、它的兼容性更好,在更加古老的浏览器中都可以运行,不需要XMLHttpRequest或ActiveX的支持；

JSONP的缺点则是：

1、它只支持GET请求而不支持POST等其它类型的HTTP请求；

2、它只支持跨域HTTP请求这种情况,不能解决不同域的两个页面之间如何进行JavaScript调用的问题。

CORS和JSONP对比：

CORS与JSONP相比,无疑更为先进、方便和可靠。

1、JSONP只能实现GET请求,而CORS支持所有类型的HTTP请求。

2、使用CORS,开发者可以使用普通的XMLHttpRequest发起请求和获得数据,比起JSONP有更好的错误处理。

3、JSONP主要被老的浏览器支持,它们往往不支持CORS,而绝大多数现代浏览器都已经支持了CORS）。

------------------------------------------------------------------------------------------

C、通过修改document.domain来跨子域

浏览器都有一个同源策略：

1、第一个限制就是第一种方法中我们说的不能通过ajax的方法去请求不同源中的文档。

2、第二个限制是浏览器中不同域的框架之间是不能进行js的交互操作的。

不同的框架之间是可以获取window对象的,但却无法获取相应的属性和方法。

有一个页面,它的地址是http://www.example.com/a.html ,

在这个页面里面有一个iframe,它的src是http://example.com/b.html,

这个页面与它里面的iframe框架是不同域的,所以我们是无法通过在页面中书写js代码来获取iframe中的东西的：

<script type="text/javascript">

function test(){

var iframe = document.getElementById('￼ifame');

var win = document.contentWindow;

//可以获取到iframe里的window对象,但该window对象的属性和方法几乎是不可用的

var doc = win.document;//这里获取不到iframe里的document对象

var name = win.name;//这里同样获取不到window对象的name属性

}

</script>

<iframe id = "iframe" src="http://example.com/b.html" onload = "test()"></iframe>

这个时候,document.domain就可以派上用场了:

1、只要设置http://www.example.com/a.html和http://example.com/b.html

这两个页面的document.domain为相同的域名就可以了。

2、但要注意的是：document.domain的设置是有限制的,

我们只能把document.domain设置成自身或更高一级的父域,且主域必须相同。

1.在页面 http://www.example.com/a.html 中设置document.domain:

<iframe id = "iframe" src="http://example.com/b.html" onload = "test()"></iframe>

<script type="text/javascript">

document.domain = 'example.com';//设置成主域

function test(){

//contentWindow可取得子窗口的window对象

alert(document.getElementById('￼iframe').contentWindow);

}

</script>

2.在页面 http://example.com/b.html 中也设置document.domain:

<script type="text/javascript">

//在iframe载入这个页面也设置document.domain,使之与主页面的document.domain相同

document.domain = 'example.com';

</script>

\*\*修改document.domain的方法只适用于不同子域的框架间的交互\*\*

------------------------------------------------------------------------------------------

D、使用window.name来进行跨域

window对象有个name属性,该属性有个特征：

即在一个窗口(window)的生命周期内,窗口载入的所有的页面都是共享一个window.name的,

每个页面对window.name都有读写的权限,window.name是持久存在一个窗口载入过的所有页面中的

------------------------------------------------------------------------------------------

E、使用HTML5的window.postMessage方法跨域

window.postMessage(message,targetOrigin) 方法是html5新引进的特性；

可以使用它来向其它的window对象发送消息,无论这个window对象是属于同源或不同源,

目前IE8+、FireFox、Chrome、Opera等浏览器都已经支持window.postMessage方法。

---------------------------------------分割线---------------------------------------

4、优缺点

\* 优点

1. 通过异步模式，提升了用户体验

2. 优化了浏览器和服务器之间的传输，减少不必要的数据往返，减少了带宽占用

3. Ajax在客户端运行，承担了一部分本来由服务器承担的工作，减少了大用户量下的服务器负载。

\* 缺点

1、ajax不支持浏览器back按钮。

2、安全问题 AJAX暴露了与服务器交互的细节。

3、对搜索引擎的支持比较弱。

4、破坏了程序的异常机制。

****如何给js数组去重？****

以下是数组去重的三种方法：

Array.prototype.unique1 = function () {

var n = []; //一个新的临时数组

for (var i = 0; i < this.length; i++) //遍历当前数组

{

//如果当前数组的第i已经保存进了临时数组，那么跳过，

//否则把当前项push到临时数组里面

if (n.indexOf(this[i]) == -1) n.push(this[i]);

}

return n;

}

-------------------------------------------------------------------------------------------------

Array.prototype.unique2 = function()

{

var n = {},r=[]; //n为hash表，r为临时数组

for(var i = 0; i < this.length; i++) //遍历当前数组

{

if (!n[this[i]]) //如果hash表中没有当前项

{

n[this[i]] = true; //存入hash表

r.push(this[i]); //把当前数组的当前项push到临时数组里面

}

}

return r;

}

-------------------------------------------------------------------------------------------------

Array.prototype.unique3 = function()

{

var n = [this[0]]; //结果数组

for(var i = 1; i < this.length; i++) //从第二项开始遍历

{

//如果当前数组的第i项在当前数组中第一次出现的位置不是i，

//那么表示第i项是重复的，忽略掉。否则存入结果数组

if (this.indexOf(this[i]) == i) n.push(this[i]);

}

return n;

}

### sql注入原理

就是通过把SQL命令插入到Web表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。

总的来说有以下几点：

1.永远不要信任用户的输入，要对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双"-"进行转换等。

2.永远不要使用动态拼装SQL，可以使用参数化的SQL或者直接使用存储过程进行数据查询存取。

3.永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。

4.不要把机密信息明文存放，请加密或者hash掉密码和敏感的信息。

XSS原理及防范

Xss(cross-site scripting)攻击指的是攻击者往Web页面里插入恶意html标签或者javascript代码。比如：攻击者在论坛中放一个 看似安全的链接，骗取用户点击后，窃取cookie中的用户私密信息；或者攻击者在论坛中加一个恶意表单， 当用户提交表单的时候，却把信息传送到攻击者的服务器中，而不是用户原本以为的信任站点。

XSS防范方法

1.代码里对用户输入的地方和变量都需要仔细检查长度和对”<”,”>”,”;”,”’”等字符做过滤；其次任何内容写到页面之前都必须加以encode，避免不小心把html tag 弄出来。这一个层面做好，至少可以堵住超过一半的XSS 攻击。   
2.避免直接在cookie 中泄露用户隐私，例如email、密码等等。 3.通过使cookie 和系统ip 绑定来降低cookie 泄露后的危险。这样攻击者得到的cookie 没有实际价值，不可能拿来重放。   
4.尽量采用POST 而非GET 提交表单

XSS与CSRF有什么区别吗？

XSS是获取信息，不需要提前知道其他用户页面的代码和数据包。CSRF是代替用户完成指定的动作，需要知道其他用户页面的代码和数据包。

要完成一次CSRF攻击，受害者必须依次完成两个步骤：

　　1.登录受信任网站A，并在本地生成Cookie。 　　2.在不登出A的情况下，访问危险网站B。

CSRF的防御

1.服务端的CSRF方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数。 2.使用验证码

****说说JS的对象,类和对象的继承？****

一、对象的定义,及构造函数 ：Javascript 面向对象编程（一）：封装

a、Javascript是一种基于对象（object-based）的语言,

遇到的所有东西几乎都是对象;

b、但是,它又不是一种真正的面向对象编程（OOP）语言,

因为它的语法中没有class（类）;

1、生成对象的原始模式

假定我们把猫看成一个对象,它有"名字"和"颜色"两个属性。

　　var Cat = {

　　　　name : '',

　　　　color : ''

　　}

现在,我们需要根据这个原型对象的规格（schema）,生成两个实例对象。

　　var cat1 = {}; // 创建一个空对象

　　　　cat1.name = "大毛"; // 按照原型对象的属性赋值

　　　　cat1.color = "黄色";

　　var cat2 = {};

　　　　cat2.name = "二毛";

　　　　cat2.color = "黑色";

好了,这就是最简单的封装了,把两个属性封装在一个对象里面。

但是有两个缺点:

一是如果多生成几个实例,写起来就非常麻烦；

二是实例与原型之间,没有任何办法,可以看出有什么联系。

------------------------------------------------------------------------------------------

2、原始模式的改进

我们可以写一个函数,解决代码重复的问题。

　　function Cat(name,color){

　　　　return {

　　　　　　name:name,

　　　　　　color:color

　　　　}

　　}

然后生成实例对象,就等于是在调用函数：

　　var cat1 = Cat("大毛","黄色");

　　var cat2 = Cat("二毛","黑色");

这种方法的问题依然是,cat1和cat2之间没有内在的联系,不能反映出它们是同一个原型对象的实例。

------------------------------------------------------------------------------------------

3、构造函数模式

为了解决从原型对象生成实例的问题,Javascript提供了一个构造函数（Constructor）模式。

所谓"构造函数",其实就是一个普通函数,但是内部使用了this变量。

对构造函数使用new运算符,就能生成实例,并且this变量会绑定在实例对象上。

比如,猫的原型对象现在可以这样写,

　　function Cat(name,color){

　　　　this.name=name;

　　　　this.color=color;

　　}

我们现在就可以生成实例对象了。

　　var cat1 = new Cat("大毛","黄色");

　　var cat2 = new Cat("二毛","黑色");

　　alert(cat1.name); // 大毛

　　alert(cat1.color); // 黄色

这时cat1和cat2会自动含有一个constructor属性,指向它们的构造函数。

　　alert(cat1.constructor == Cat); //true

　　alert(cat2.constructor == Cat); //true

Javascript还提供了一个instanceof运算符,验证原型对象与实例对象之间的关系。

　　alert(cat1 instanceof Cat); //true

　　alert(cat2 instanceof Cat); //true

------------------------------------------------------------------------------------------

4、构造函数模式的问题

构造函数方法很好用,但是存在一个浪费内存的问题。

请看,我们现在为Cat对象添加一个不变的属性"type"（种类）,再添加一个方法eat（吃老鼠）,原型对象Cat就变成了下面这样：

　　function Cat(name,color){

　　　　this.name = name;

　　　　this.color = color;

　　　　this.type = "猫科动物";

　　　　this.eat = function(){alert("吃老鼠");};

　　}

还是采用同样的方法,生成实例：

　　var cat1 = new Cat("大毛","黄色");

　　var cat2 = new Cat ("二毛","黑色");

　　alert(cat1.type); // 猫科动物

　　cat1.eat(); // 吃老鼠

表面上好像没什么问题,但是实际上这样做,有一个很大的弊端。

那就是对于每一个实例对象,type属性和eat()方法都是一模一样的内容,每一次生成一个实例,

都必须为重复的内容,多占用一些内存。这样既不环保,也缺乏效率。

　　 alert(cat1.eat == cat2.eat); //false

能不能让type属性和eat()方法在内存中只生成一次,然后所有实例都指向那个内存地址呢？回答是可以的。

------------------------------------------------------------------------------------------

5、Prototype模式

Javascript规定,每一个构造函数都有一个prototype属性,指向另一个对象。

这个对象的所有属性和方法,都会被构造函数的实例继承。

这意味着,我们可以把那些不变的属性和方法,直接定义在prototype对象上。

　　function Cat(name,color){

　　　　this.name = name;

　　　　this.color = color;

　　}

　　Cat.prototype.type = "猫科动物";

　　Cat.prototype.eat = function(){alert("吃老鼠")};

然后,生成实例。

　　var cat1 = new Cat("大毛","黄色");

　　var cat2 = new Cat("二毛","黑色");

　　alert(cat1.type); // 猫科动物

　　cat1.eat(); // 吃老鼠

这时所有实例的type属性和eat()方法,其实都是同一个内存地址,指向prototype对象,因此就提高了运行效率。

　　alert(cat1.eat == cat2.eat); //true

------------------------------------------------------------------------------------------

6、Prototype模式的验证方法

为了配合prototype属性,Javascript定义了一些辅助方法,帮助我们使用它;

6.1 isPrototypeOf()

这个方法用来判断,某个proptotype对象和某个实例之间的关系。

　　alert(Cat.prototype.isPrototypeOf(cat1)); //true

　　alert(Cat.prototype.isPrototypeOf(cat2)); //true

6.2 hasOwnProperty()

每个实例对象都有一个hasOwnProperty()方法,

用来判断某一个属性到底是本地属性,还是继承自prototype对象的属性。

　　alert(cat1.hasOwnProperty("name")); // true

　　alert(cat1.hasOwnProperty("type")); // false

6.3 in运算符

in运算符可以用来判断,某个实例是否含有某个属性,不管是不是本地属性。

　　alert("name" in cat1); // true

　　alert("type" in cat1); // true

in运算符还可以用来遍历某个"对象"的所有属性。

　　for(var prop in cat1) {

alert("cat1["+prop+"]="+cat1[prop]);

}

---------------------------------------分割线---------------------------------------

二、对象构造函数的继承 Javascript面向对象编程（二）：构造函数的继承

今天要介绍的是,对象之间的"继承"的五种方法:

现在有一个"动物"对象的构造函数:

　　function Animal(){

　　　　this.species = "动物";

　　}

还有一个"猫"对象的构造函数:

　　function Cat(name,color){

　　　　this.name = name;

　　　　this.color = color;

　　}

怎样才能使"猫"继承"动物"呢？

------------------------------------------------------------------------------------------

1、构造函数绑定

第一种方法也是最简单的方法,使用call（子对象,单个参数）或apply（子对象,数组参数）方法,

将父对象的构造函数绑定在子对象上,即在子对象构造函数中加一行：

　　function Cat(name,color){

　　　　Animal.apply(this, arguments);

　　　　this.name = name;

　　　　this.color = color;

　　}

　　var cat1 = new Cat("大毛","黄色");

　　alert(cat1.species); // 动物

------------------------------------------------------------------------------------------

2、prototype模式

第二种方法更常见,使用prototype属性,

如果"猫"的prototype对象,指向一个Animal的实例,那么所有"猫"的实例,就能继承Animal了。

　　Cat.prototype = new Animal();

　　Cat.prototype.constructor = Cat;

　　var cat1 = new Cat("大毛","黄色");

　　alert(cat1.species); // 动物

代码的第一行,我们将Cat的prototype对象指向一个Animal的实例。

　　Cat.prototype = new Animal();

它相当于完全删除了prototype 对象原先的值,然后赋予一个新值。但是,第二行又是什么意思呢？

　　Cat.prototype.constructor = Cat;

原来,任何一个prototype对象都有一个constructor属性,指向它的构造函数;

如果没有"Cat.prototype = new Animal();"这一行,Cat.prototype.constructor是指向Cat的;

加了这一行以后,Cat.prototype.constructor指向Animal;

　　alert(Cat.prototype.constructor == Animal); //true

更重要的是,每一个实例也有一个constructor属性,默认调用prototype对象的constructor属性。

　　alert(cat1.constructor == Cat.prototype.constructor); // true

因此,在运行"Cat.prototype = new Animal();"这一行之后,cat1.constructor也指向Animal！

　　alert(cat1.constructor == Animal); // true

这显然会导致继承链的紊乱（cat1明明是用构造函数Cat生成的）,因此我们必须手动纠正,

将Cat.prototype对象的constructor值改为Cat。这就是第二行的意思;

很重要的一点,编程时务必要遵守。下文都遵循这一点,即如果替换了prototype对象,

　　o.prototype = {};

那么,下一步必然是为新的prototype对象加上constructor属性,并将这个属性指回原来的构造函数。

　　o.prototype.constructor = o;

------------------------------------------------------------------------------------------

3、直接继承prototype

第三种方法是对第二种方法的改进。

由于Animal对象中,不变的属性都可以直接写入Animal.prototype。

所以,我们也可以让Cat()跳过 Animal(),直接继承Animal.prototype。

现在,我们先将Animal对象改写：

　　function Animal(){}

　　Animal.prototype.species = "动物";

然后,将Cat的prototype对象,然后指向Animal的prototype对象,这样就完成了继承。

　　Cat.prototype = Animal.prototype;

　　Cat.prototype.constructor = Cat;

　　var cat1 = new Cat("大毛","黄色");

　　alert(cat1.species); // 动物

与前一种方法相比,这样做的优点是效率比较高（不用执行和建立Animal的实例了）,比较省内存。

缺点是 Cat.prototype和Animal.prototype现在指向了同一个对象,

那么任何对Cat.prototype的修改,都会反映到Animal.prototype。

所以,上面这一段代码其实是有问题的。请看第二行

　　Cat.prototype.constructor = Cat;

这一句实际上把Animal.prototype对象的constructor属性也改掉了,所有请看第四种方法！！！！

　　alert(Animal.prototype.constructor); // Cat

------------------------------------------------------------------------------------------

4、利用空对象作为中介

由于"直接继承prototype"存在上述的缺点,所以就有第四种方法,利用一个空对象作为中介。

　　var F = function(){};

　　F.prototype = Animal.prototype;

　　Cat.prototype = new F();

　　Cat.prototype.constructor = Cat;

F是空对象,所以几乎不占内存。这时,修改Cat的prototype对象,就不会影响到Animal的prototype对象。

　　alert(Animal.prototype.constructor); // Animal

我们将上面的方法,封装成一个函数,便于使用。

　　function extend(Child, Parent) {

　　　　var F = function(){};

　　　　F.prototype = Parent.prototype;

　　　　Child.prototype = new F();

　　　　Child.prototype.constructor = Child;

　　　　Child.uber = Parent.prototype;

　　}

使用的时候,方法如下

　　extend(Cat,Animal);

　　var cat1 = new Cat("大毛","黄色");

　　alert(cat1.species); // 动物

这个extend函数,就是YUI库如何实现继承的方法。

另外,说明一点,函数体最后一行：

　　Child.uber = Parent.prototype;

意思是为子对象设一个uber属性,这个属性直接指向父对象的prototype属性,uber是一个德语词,意思是"向上"、"上一层"。

这等于在子对象上打开一条通道,可以直接调用父对象的方法。这一行放在这里,只是为了实现继承的完备性,纯属备用性质。

------------------------------------------------------------------------------------------

5、拷贝继承

上面是采用prototype对象,实现继承。我们也可以换一种思路,纯粹采用"拷贝"方法实现继承。

简单说,如果把父对象的所有属性和方法,拷贝进子对象,不也能够实现继承吗？这样我们就有了第五种方法。

首先,还是把Animal的所有不变属性,都放到它的prototype对象上。

　　function Animal(){}

　　Animal.prototype.species = "动物";

然后,再写一个函数,实现属性拷贝的目的。

　　function extend2(Child, Parent) {

　　　　var p = Parent.prototype;

　　　　var c = Child.prototype;

　　　　for (var i in p) {

　　　　　　c[i] = p[i];

}

　　　　return c;

　　}

这个函数的作用,就是将父对象的prototype对象中的属性,一一拷贝给Child对象的prototype对象

（但这里for in 属于浅拷贝,因为for in 会遍历原型链上的属性） 。

使用的时候,这样写：

　　extend2(Cat, Animal);

　　var cat1 = new Cat("大毛","黄色");

alert(cat1.species); // 动物

---------------------------------------分割线---------------------------------------

三、非构造函数的继承 Javascript面向对象编程（三）：非构造函数的继承

第一部分介绍了"封装"；

第二部分介绍了使用构造函数实现"继承"。

第三部分介绍不使用构造函数实现"继承"。

1、什么是"非构造函数"的继承？

比如,现在有一个对象,叫做"中国人"。

　　var Chinese = {

　　　　nation:'中国'

　　};

还有一个对象,叫做"医生"。

　　var Doctor ={

　　　　career:'医生'

　　}

请问怎样才能让"医生"去继承"中国人",也就是说,我怎样才能生成一个"中国医生"的对象？

这里要注意,这两个对象都是普通对象,不是构造函数,无法使用构造函数方法实现"继承"。

------------------------------------------------------------------------------------------

2、object()方法 返回new原型对象的方法 （创建jquery对象就是用这个原理,但多了一步判断）

json格式的发明人Douglas Crockford,提出了一个object()函数,可以做到这一点。

　　function object(o) {

　　　　function F() {}

　　　　F.prototype = o;

　　　　return new F();

　　}

这个object()函数,其实只做一件事,就是把子对象的prototype属性,指向父对象,从而使得子对象与父对象连在一起。

使用的时候,第一步先在父对象的基础上,生成子对象：

　　var Doctor = object(Chinese);

然后,再加上子对象本身的属性：

　　Doctor.career = '医生';

这时,子对象已经继承了父对象的属性了。

　　alert(Doctor.nation); //中国

------------------------------------------------------------------------------------------

3、object.create()方法

a.在父对象：

var Chinese={"nation":"中国"};

b.建立子对象：

var Doctor= Object.create(Chinese);

c.再加上子对象本身的属性：

Doctor.career = '医生';

这时,子对象已经继承了父对象的属性了。

console.log(Doctor.nation + Doctor.career);//中国医生

------------------------------------------------------------------------------------------

4、浅拷贝

除了使用"prototype链"以外,还有另一种思路：把父对象的属性,全部拷贝给子对象,也能实现继承。

下面这个函数,就是在做拷贝：

　　function extendCopy(p) {

　　　　var c = {};

　　　　for (var i in p) {

　　　　　　c[i] = p[i];

　　　　}

　　　　return c;

　　}

使用的时候,这样写：

　　var Doctor = extendCopy(Chinese);

　　Doctor.career = '医生';

　　alert(Doctor.nation); // 中国

但是,这样的拷贝有一个问题。那就是,如果父对象的属性等于数组或另一个对象,

那么实际上,子对象获得的只是一个内存地址,而不是真正拷贝,因此存在父对象被篡改的可能。

请看,现在给Chinese添加一个"出生地"属性,它的值是一个数组。

　　Chinese.birthPlaces = ['北京','上海','香港'];

通过extendCopy()函数,Doctor继承了Chinese。

　　var Doctor = extendCopy(Chinese);

然后,我们为Doctor的"出生地"添加一个城市：

　　Doctor.birthPlaces.push('厦门');

发生了什么事？Chinese的"出生地"也被改掉了！

　　alert(Doctor.birthPlaces); //北京, 上海, 香港, 厦门

　　alert(Chinese.birthPlaces); //北京, 上海, 香港, 厦门

所以,extendCopy()只是拷贝基本类型的数据,我们把这种拷贝叫做"浅拷贝"。这是早期jQuery实现继承的方式。

------------------------------------------------------------------------------------------

5、深拷贝

所谓"深拷贝",就是能够实现真正意义上的数组和对象的拷贝。它的实现并不难,只要递归调用"浅拷贝"就行了。

　　function deepCopy(p, c) {

　　　　var c = c || {};

for (var i in p) {

if(p.hasOwnProperty(i)){ //不去继承p原型链上的属性

if (typeof p[i] === 'object') {

　　　　　　　　c[i] = (p[i].constructor === Array) ？ [] : {};

　　　　　　　　deepCopy(p[i], c[i]);

　　　　　　} else {

　　　　　　　　　c[i] = p[i];

　　　　　　}

}

　　　　}

　　　　return c;

　　}

使用的时候这样写：

　　var Doctor = deepCopy(Chinese);

现在,给父对象加一个属性,值为数组。然后,在子对象上修改这个属性：

　　Chinese.birthPlaces = ['北京','上海','香港'];

　　Doctor.birthPlaces.push('厦门');

这时,父对象就不会受到影响了。

　　alert(Doctor.birthPlaces); //北京, 上海, 香港, 厦门

　　alert(Chinese.birthPlaces); //北京, 上海, 香港

目前,jQuery库使用的就是这种继承方法。

### **栈和堆的区别？**

栈区（stack）— 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。

堆区（heap） — 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收。

堆（数据结构）：堆可以被看成是一棵树，如：堆排序；

栈（数据结构）：一种先进后出的数据结构。

### **说说你对Promise的理解**

依照 Promise/A+ 的定义，Promise 有四种状态：

pending: 初始状态, 非 fulfilled 或 rejected.

fulfilled: 成功的操作.

rejected: 失败的操作.

settled: Promise已被fulfilled或rejected，且不是pending

另外， fulfilled 与 rejected 一起合称 settled。

Promise 对象用来进行延迟(deferred) 和异步(asynchronous ) 计算。

Promise 的构造函数

构造一个 Promise，最基本的用法如下：

var promise = new Promise(function(resolve, reject) {

if (...) { // succeed

resolve(result);

} else { // fails

reject(Error(errMessage));

}

});

Promise 实例拥有 then 方法（具有 then 方法的对象，通常被称为 thenable）。它的使用方法如下：

promise.then(onFulfilled, onRejected)

接收两个函数作为参数，一个在 fulfilled 的时候被调用，一个在 rejected 的时候被调用，接收参数就是 future，onFulfilled 对应 resolve, onRejected 对应 reject。

### 谈谈This对象的理解

* this总是指向函数的直接调用者（而非间接调用者）；
* 如果有new关键字，this指向new出来的那个对象；
* 在事件中，this指向触发这个事件的对象，特殊的是，IE中的attachEvent中的this总是指向全局对象Window；

### **函数内部arguments变量有哪些特性,有哪些属性,如何将它转换为数组**

* arguments所有函数中都包含的一个局部变量，是一个类数组对象，对应函数调用时的实参。如果函数定义同名参数会在调用时覆盖默认对象
* arguments[index]分别对应函数调用时的实参，并且通过arguments修改实参时会同时修改实参
* arguments.length为实参的个数（Function.length表示形参长度）
* arguments.callee为当前正在执行的函数本身，使用这个属性进行递归调用时需注意this的变化
* arguments.caller为调用当前函数的函数（已被遗弃）
* 转换为数组：var args = Array.prototype.slice.call(arguments, 0);

### **offsetWidth/offsetHeight,clientWidth/clientHeight与scrollWidth/scrollHeight的区别**

* offsetWidth/offsetHeight返回值包含****content + padding + border****，效果与e.getBoundingClientRect()相同
* clientWidth/clientHeight返回值只包含****content + padding****，如果有滚动条，也****不包含滚动条****
* scrollWidth/scrollHeight返回值包含****content + padding + 溢出内容的尺寸****

****JSON的了解？XML和JSON的区别？****

了解： JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。 它是基于JavaScript的一个子集。数据格式简单, 易于读写, 占用带宽小； {'age':'12', 'name':'back'} JSON.parse('str') //转换string为JSON格式 JSON.stringfy('JSON') //转换JSON为string格式

\*区别：

(1).数据体积方面。

JSON相对于XML来讲，数据的体积小，传递的速度更快些。

(2).数据交互方面。

JSON与JavaScript的交互更加方便，更容易解析处理，更好的数据交互。

(3).数据描述方面。

JSON对数据的描述性比XML较差。

(4).传输速度方面。

JSON的速度要远远快于XML。

****哪些地方会出现css阻塞,哪些地方会出现js阻塞？****

js的阻塞特性：

所有浏览器在下载JS的时候,会阻止一切其他活动,比如其他资源的下载,内容的呈现等等。

直到JS下载、解析、执行完毕后才开始继续并行下载其他资源并呈现内容。

为了提高用户体验,新一代浏览器都支持并行下载JS,但是JS下载仍然会阻塞其它资源的下载（例如.图片,css文件等）。

\*由于浏览器为了防止出现JS修改DOM树,需要重新构建DOM树的情况,所以就会阻塞其他的下载和呈现。

\*嵌入JS会阻塞所有内容的呈现,而外部JS只会阻塞其后内容的显示,2种方式都会阻塞其后资源的下载。

也就是说外部样式不会阻塞外部脚本的加载,但会阻塞外部脚本的执行。

CSS怎么会阻塞加载了？

CSS本来是可以并行下载的,在什么情况下会出现阻塞加载了(在测试观察中,IE6下CSS都是阻塞加载）

\*当CSS后面跟着嵌入的JS的时候,该CSS就会出现阻塞后面资源下载的情况;

\*而当把嵌入JS放到CSS前面,就不会出现阻塞的情况了。

根本原因：

\*因为浏览器会维持html中css和js的顺序,样式表必须在嵌入的JS执行前先加载、解析完。

\*而嵌入的JS会阻塞后面的资源加载,所以就会出现上面CSS阻塞下载的情况。

嵌入JS应该放在什么位置？

1、放在底部,虽然放在底部照样会阻塞所有呈现,但不会阻塞资源下载。

2、如果嵌入JS放在head中,请把嵌入JS放在CSS头部。

3、使用defer（只支持IE）

4、不要在嵌入的JS中调用运行时间较长的函数,如果一定要用,可以用`setTimeout`来调用