**DOM操作——怎样添加、移除、移动、复制、创建和查找节点。**

（1）创建新节点

      createDocumentFragment()    //创建一个DOM片段

      createElement\_x\_x()   //创建一个具体的元素

      createTextNode()   //创建一个文本节点

（2）添加、移除、替换、插入

      appendChild()

      removeChild()

      replaceChild()

      insertBefore()

（3）查找

   getElementsByTagName()    //通过标签名称

   getElementsByName()    //通过元素的Name属性的值

   getElementById()    //通过元素Id，唯一性

getElementByClassName() //通过class查找

（4）复制节点

cloneNode() 方法，用于复制节点， 接受一个布尔值参数， true 表示深复制（复制节点及其所有子节点）， false 表示浅复制（复制节点本身，不复制子节点）

**谈谈 innerHTML、nodeValue 与  textContent  之间的区别**

innerHTML 属性读取或设置节点起始和结束标签中的 HTML  内容；  
nodeValue  属性读取或设置指定节点的文本内容，适用于文本类型的节点；  
textContent  属性读取或设置指定节点的文本内容，对于元素节点而言，会返回所包含的所有子节点中的文本内容的组合。

**DOM  操作中，如何获取元素的属性值？**

对于元素节点，获取其某属性的值有多种方式，如下所示：

1、element.attributes[下标].value  
2、element.attributes['属性名'].value  
3、element.getAttributeNode('属性名').value  
4、element.getAttribute('属性名')

**简述 window 对象除  document 以外的一些常用子对象，并描述其作用？**

window  对象有很多子对象，除了 document  以外，还有如下常用子对象：Ÿ

screen  对象：此对象包含有关客户端显示屏幕的信息，常用于获取屏幕的分辨率和色彩；ŸŸ  
history  对象：此对象包含用户（在浏览器窗口中）访问过的 URL；  
location  对象：此对象包含有关当前  URL  的信息，常用于获取和改变当前浏览的网址；ŸŸ  
navigator  对象：此对象包含有关浏览器的信息，常用于获取客户端浏览器和操作系统信息；  
event  对象：任何事件触发后将会产生一个 event  对象，该对象记录事件发生时的鼠标位置、键盘按键状态和触发对象等信息。

**JavaScript如何实现禁用浏览器后退按钮**

方法一：

<**script**>  
 **javascript**:**window**.**history**.forward(1);  
</**script**>

利用JS产生一个“前进”的动作，以抵消后退功能，这种方法应该是最简洁的，并且不需要考虑用户连点两次或多次“后退”的情况，缺点是当用户端禁用了JavaScript之后即失效。

方法二：

<**a HREF="logout.do" onclick="javascript**:**location**.replace(**this**.**href**); **event**.**returnValue**=**false**; **"**>  
 Logout (Back Disabled)  
</**a**>

用location.replace从一个页面转到另一个页面。这种方法的原理是，用新页面的URL替换当前的历史纪录，这样浏览历史记录中就只有一个页面，后退按钮永远不会变为可用。我想这可能正是许多人所寻求的方法，但这种方法仍旧不是任何情况下的最好方法。这种方法的缺点在于：简单地运用Response.Redirect将不再有效，这是因为每次用户从一个页面转到另一个页面，我们都必须用客户端代码清除location.history。另外还要注意，这种方法清除的是最后一个访问历史记录，而不是全部的访问记录。

方法三：

当键盘敲下后退键（Backspace）后   
1、禁止浏览器自动后退   
2、但不影响密码、单行文本、多行文本输入框等的回退操作

<**script**>  
 *//处理键盘事件 禁止后退键（Backspace）密码或单行、多行文本框除外* **function** *banBackSpace*(e){  
 **var** ev = e || **window**.**event**;*//获取event对象* **var** obj = ev.**target** || ev.**srcElement**;*//获取事件源* **var** t = obj.**type** || obj.getAttribute(**'type'**);*//获取事件源类型  
 //获取作为判断条件的事件类型* **var** vReadOnly = obj.getAttribute(**'readonly'**);  
 **var** vEnabled = obj.getAttribute(**'enabled'**);  
 *//处理null值情况* vReadOnly = (vReadOnly == **null**) ? **false** : vReadOnly;  
 vEnabled = (vEnabled == **null**) ? **true** : vEnabled;  
 *//当敲Backspace键时，事件源类型为密码或单行、多行文本的，  
 //并且readonly属性为true或enabled属性为false的，则退格键失效* **var** flag1=(ev.**keyCode** == 8 && (t==**"password"** || t==**"text"** || t==**"textarea"**)  
 && (vReadOnly==**true** || vEnabled!=**true**))?**true**:**false**;  
 *//当敲Backspace键时，事件源类型非密码或单行、多行文本的，则退格键失效* **var** flag2=(ev.**keyCode** == 8 && t != **"password"** && t != **"text"** && t != **"textarea"**)  
 ?**true**:**false**;  
 *//判断* **if**(flag2){  
 **return false**;  
 }  
 **if**(flag1){  
 **return false**;  
 }  
 }  
 *//禁止后退键 作用于Firefox、Opera* **document**.onkeypress=*banBackSpace*;  
 *//禁止后退键 作用于IE、Chrome* **document**.onkeydown=*banBackSpace*;  
</**script**>

以上方法都是针对“后退”按钮作出的反应，客户端浏览器需要打开JavaScript代码。

方法四：禁止缓存(看不懂无所谓，作了解即可)

<%php  
 response.setHeader("Cache-Control", "no-cache");  
  
 response.setHeader("Cache-Control", "no-store");  
  
 response.setDateHeader("Expires", 0);  
  
 response.setHeader("Pragma", "no-cache");  
%>

这种方法使用服务器端脚本，强制浏览器重新访问服务器下载页面，而不从缓存读取，结合struts jsp页面中的<logic>标签实现重新定向。

方法五：

<**script**>  
 **function** *logout*(){  
 **window**.close(**true**);  
 **window**.open(**"logout.do"**);  
 }  
</**script**>  
<**button onClick="***logout*()**"**>Logout</**button**>

 这种方法比较偷懒，关掉浏览器再重新开，经过我的测试在视觉上几乎感觉不出来延时，同时又保证了后退按钮不可用（新窗口浏览器后退按钮为灰色），看起来似乎是个好方法，但缺点也比较明显：

首先，关闭和重开的浏览器窗口大小可能不同，用户可以明显看出这一过程，并且在一定程度上影响操作。

其次，同上，这是一个JavaScript方法。

**body中的onload()函数和jQuery中的document.ready()有什么区别？**

onload()和document.ready()的区别有以下两点：

1、我们可以在页面中使用多个document.ready()，但只能使用一次onload()。

2、document.ready()函数在页面DOM元素加载完以后就会被调用，而onload()函数则要在所有的关联资源（包括图像、音频）加载完毕后才会调用。

**jQuery中有哪几种类型的选择器？**

有3种类型的选择器，如下：

1、基本选择器：直接根据id、css类名、元素名返回匹配的dom元素。

2、层次选择器：也叫做路径选择器，可以根据路径层次来选择相应的DOM元素。

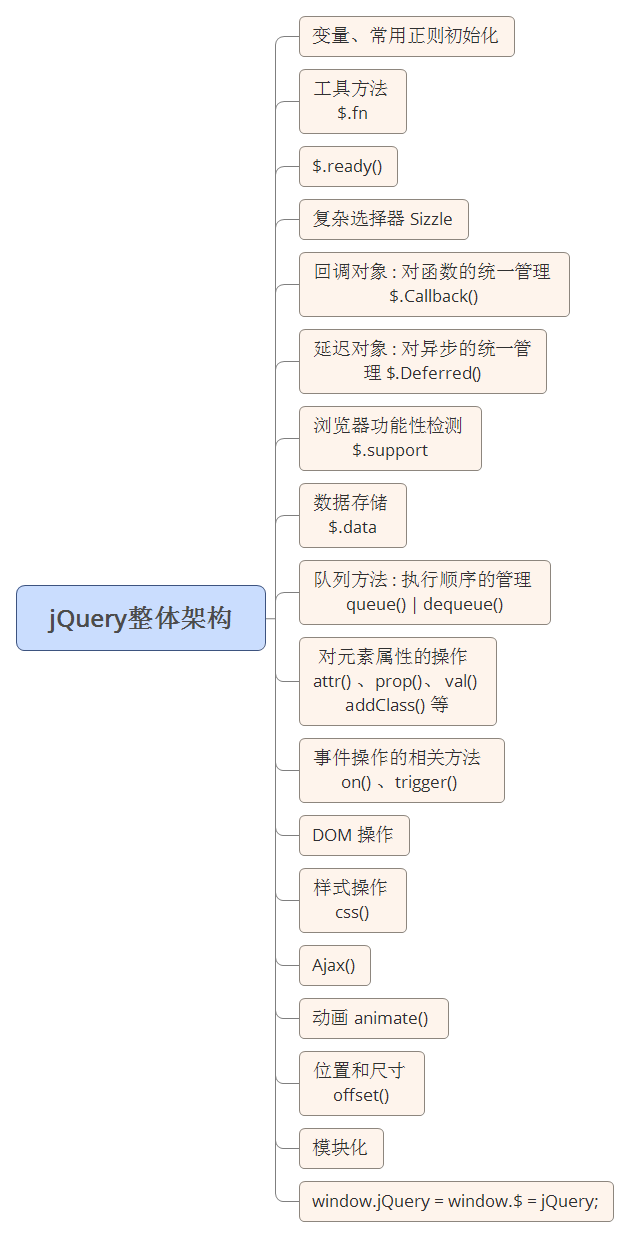
3、过滤选择器：在前面的基础上过滤相关条件，得到匹配的dom元素。

**当CDN上的jQuery文件不可用时，该怎么办？**

为了节省带宽和脚本引用的稳定性，我们会使用CDN上的jQuery文件，例如google的jquery cdn服务。但是如果这些CDN上的jQuery服务不可用，我们还可以通过以下代码来切换到本地服务器的jQuery版本：

<**script type="text/javascript" language="Javascript" src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.4.1.min.js "**></**script**>  
  
<**script type='text/javascript'**>*//<![CDATA[***if** (typeof jQuery == **'undefined'**) {  
  
 **document**.write(unescape(**"%3Cscript src='/Script/jquery-1.4.1.min.js' type='text/javascript' %3E%3C/script%3E"**));  
  
}*//]]>*</**script**>

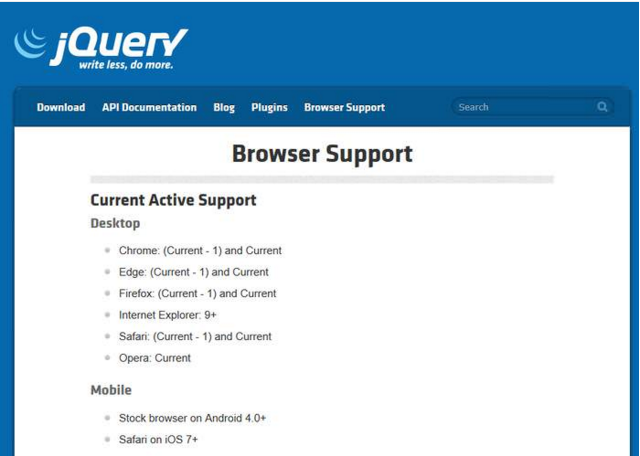
**Jquery源码整体架构你知道么？能说多少？**

****

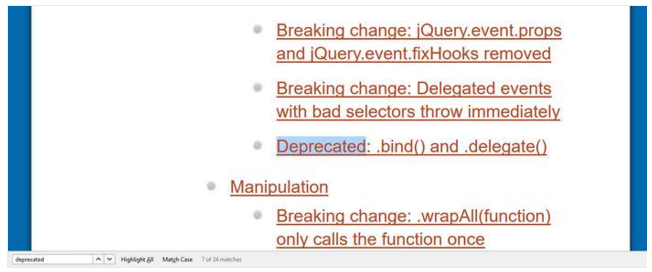
**面试题：jQuery3.0中的新特性有哪些？**

1. 移除旧的IE工作区

新的最终版最主要的目标是更加快速，更加时尚，因此，那些支持早于IE9版本的相关技术与工作区都被移除了。这意味着如果你想要或者需要支持IE6-8，你必须用回1.12版本，因为甚至是2.X版本都无法完整支持早于IE9的浏览器。



注意：在jQuery 3中还弃用了一些其它的功能。截止到2016年6月，升级指南还没有提供一个对于弃用功能的分组整理。因此，你如果想要了解弃用功能，可以通过ctrl+f键来进行搜索。



2. jQuery 3.0运行在Strict Mode下

大多数支持jQuery 3的浏览器都支持strict mode，而在本次更新中对此进行了规定。

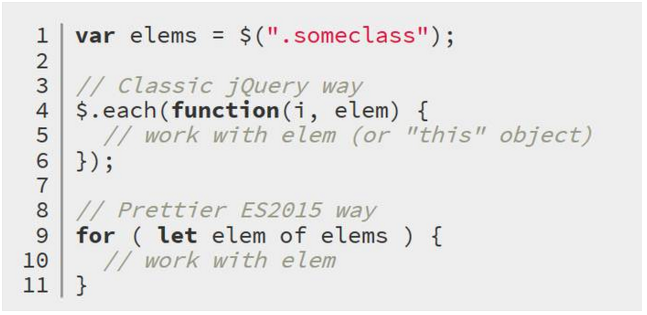
虽然jQuery 3是写在strict mode中的，但是你需要了解的是你的代码并不需要运行在strict mode中，因此如果你想要把以前的代码迁移到jQuery 3，你不需要对已存在的jQuery代码进行重写。Strict和non-strict mode的JavaScript代码可以和谐共存。

这里有一些例外：某些版本的ASP.NET，因为strict mode的缘故，无法与jQuery 3兼容，如果你的代码涉及到了ASP.NET，你可以查看[关于strict mode的细节](http://jquery.com/upgrade-guide/3.0/#breaking-change-jquery-3-0-runs-in-strict-mode)。

3. 引进for...of循环

jQuery 3支持[for...of](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Statements/for...of)语句，这是由[ECMAScript 6](http://www.hongkiat.com/blog/ecmascript-6/)中新引进的一种for循环语句。这提供了对[Arrays](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)，[Maps](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map)和[Sets](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set)这样的可迭代对象一种更直接的遍历方法。

在jQuery中，for...of循环可以取代以前的[$.each(...)](http://api.jquery.com/jquery.each/)语法，并且更容易通过jQuery的元素[集合](https://learn.jquery.com/using-jquery-core/jquery-object/)进行循环。

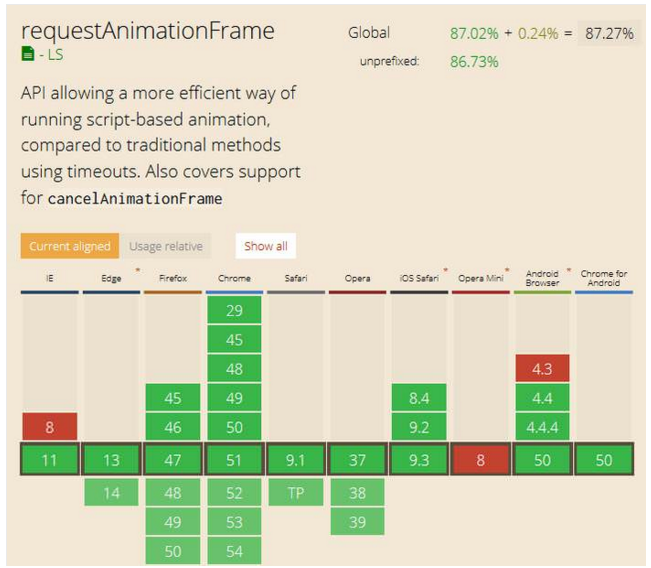


注意：for...of只能在支持ECMAScript 6的环境下或者使用Babel这样的JavaScript编译器下工作。

#### 4. 动画方面采用新的API

jQuery 3使用requestAnimationFrame() API来执行动画，使动画运行得更加顺畅、快速。新的API只用于支持它的浏览器，对于那些更老的浏览器（如IE9）jQuery使用先前的API来作为显示动画的后备方案。

RequestAnimationFrame已经推出一段时间了，如果你对如何使用它以及什么时候使用它感兴趣，CSS Tricks上有一篇很好的[文章](https://css-tricks.com/using-requestanimationframe/)可供你阅读。



#### 5. 对包含特殊含义的字符串提供转义的新方法

新的jQuery.escapeSelector()提供了对在CSS中存在特殊含义的字符串或字符进行转义的方法，该方法可使这些字符串或字符能够继续用于jQuery选择器中，而无需对那些无法进行正确理解的JavaScript解释器进行转义。

这个[示例](http://jquery.com/upgrade-guide/3.0/#feature-new-method-jquery-escapeselector)可以让你更好的了解这个全新的方法：

举个例子，页面中某个ID为“abc.def”的元素由于选择器将其解析为ID为“abc”且包含一个名为“def”的类的对象，而无法被$( "#abc.def" )选定。但是它可以由 $( "#" + $.escapeSelector( "abc.def" ) )来进行选定。

我无法确定发生这种情况的频率，但是假如你碰到过这种类似的情况，这无疑给了你一个解决该问题的简单方法。

#### 6. 类操作方法支持SVG

不幸的是，jQuery 3现在还无法完整的支持SVG，但是对于[操作CSS类名称](https://api.jquery.com/category/manipulation/class-attribute/)的jQuery方法，如[.addClass()](https://api.jquery.com/addclass/)和[.hasClass()](https://api.jquery.com/hasClass/)现在可以将SVG文档作为目标。这意味着你可以修改（添加、删除、切换）或者查找SVG（可缩放矢量图形）下的jQuery类，然后使用CSS的类样式。（相关阅读：[Styling Scalable Vector Graphic (SVG) with CSS](http://www.hongkiat.com/blog/scalable-vector-graphic-css-styling/)）

#### 7. 延迟对象现在与JS Promises兼容

[JavaScript Promises](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise)，用于异步计算的对象，现在在ECMAScript 6中已经被标准化；它们的运行状况和特性的详细说明参见[Promises/A+](https://promisesaplus.com/)标准。



在jQuery 3中，[延迟对象](https://api.jquery.com/category/deferred-object/)已经与新的Promises/A+标准相兼容，延迟对象已经成为让创建回调队列成为可能的可链对象(chainable objects)。

新的特性改变了异步回调函数执行过程，Promises允许开发者编写在逻辑上与同步代码更接近的异步代码。

查看升级指南的[代码示例](http://jquery.com/upgrade-guide/3.0/#callback-exit)，或者查看关于JavaScript Promises基础的[Scotch.io教程](https://scotch.io/tutorials/understanding-javascript-promises-pt-i-background-basics)可以让你了解更多的信息。

#### 8. jQuery.when()对于多种参数的不同解读

[$.when()](https://api.jquery.com/jquery.when/)提供了执行回调函数的一种方法，它成为了jQuery 1.5之后的一部分。这个方法非常灵活，它适用于零参数，也适用于一个或者多个作为参数的对象。

jQuery 3改变了当包含[$.then()](https://api.jquery.com/deferred.then/)方法时对$.when()中参数的解读方法，你可以对$.when()进行额外的参数回调。

在jQuery 3中，如果你在$.when()中的then()方法内增加一个输入参数，该参数将会被解释成一个可与Promise兼容的["thenable"](https://promisesaplus.com/#point-7)。

这意味着 $.when方法能够接收更加宽广的输入范围，比如原生的[ES6 Promises](https://www.promisejs.org/)和[Bluebird Promises](http://bluebirdjs.com/docs/getting-started.html)，这也使得编写更复杂的异步回调成为了可能。

#### 9. 新的显示/隐藏逻辑

为了增加响应式设计的兼容性，与元素显示和隐藏的代码在jQuery 3中进行了更新。

从现在开始，[.show()](http://api.jquery.com/show/)、[.hide()](http://api.jquery.com/hide/)和[.toggle()](http://api.jquery.com/toggle/)方法将会以内联样式为重点取代以前的计算样式，这种方式能够更尊重样式列表的改变。

新的代码尽可能的尊重样式列表中所呈现的数值，这意味着CSS规则可根据事件（如设备重新定位和窗口尺寸调整）进行动态改变。

这个说明文件申明了最重要的结果就是：

“作为结果，断开连接的元素除非指定了内联display: none，否则将不再认定为是隐藏的。因此，在jQuery 3.0中，.toggle()不再对断开连接元素与已连接元素进行区分。”

如果你想要更好的了解关于全新的显示/隐藏逻辑，这个[Github discussion](https://github.com/jquery/jquery/issues/2854)会对你有帮助。jQuery开发者还发布了[Google文档表](https://docs.google.com/spreadsheets/d/1UaISjcS3UMxVJ7eSBIXtK-jqF8Grl67w640peCqlkoc/edit?pref=2&pli=1#gid=0)来展示在不同用例下新规则的运转。

#### 10. 对于防止XSS攻击的额外保护

jQuery 3增加一个额外的安全层用户用于防备跨站点脚本(XSS)攻击，它需要开发者指定[$.ajax()](http://api.jquery.com/jquery.ajax/)和[$.get()](https://api.jquery.com/jquery.get/)方法中的选项为dataType: "script"。

换言之，如果你想要执行跨站点的脚本请求，你必须在这些方法中作出这样的声明。



新的改变对于当“远程站点传递非脚本内容之后又提供有恶意脚本的服务内容”的情况是非常有效的。这个变化不会影响$.getScript()方法，因为它明确地设置了dataType: "script"选项。

**什么是AJAX**

什么是ajax：

AJAX是“Asynchronous JavaScript and XML”的缩写。他是指一种创建交互式网页应用的网页开发技术。  
Ajax包含下列技术：  
基于web标准（standards-basedpresentation）XHTML+CSS的表示；  
使用 DOM（Document ObjectModel）进行动态显示及交互；  
使用 XML 和 XSLT 进行数据交换及相关操作；  
使用 XMLHttpRequest 进行异步数据查询、检索；  
使用 JavaScript 将所有的东西绑定在一起。

**为什么要用ajax：**

Ajax应用程序的优势在于：

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

**AJAX最大的特点是什么。**

Ajax可以实现动态不刷新（局部刷新）。就是能在不更新整个页面的前提下维护数据。这使得Web应用程序更为迅捷地回应用户动作，并避免了在网络上发送那些没有改变过的信息。

**请介绍一下XMLhttprequest对象。**

Ajax的核心是JavaScript对象XmlHttpRequest。该对象在Internet Explorer 5中首次引入，它是一种支持异步请求的技术。简而言之，XmlHttpRequest使您可以使用JavaScript向服务器提出请求并处理响应，而不阻塞用户。通过XMLHttpRequest对象，Web开发人员可以在页面加载以后进行页面的局部更新。

**AJAX技术体系的组成部分有哪些。**

HTML，css，dom，xml，xmlHttpRequest，javascript  
**AJAX应用和传统Web应用有什么不同。**

在传统的Javascript编程中，如果想得到服务器端数据库或文件上的信息，或者发送客户端信息到服务器，需要建立一个HTML form然后GET或者POST数据到服务器端。用户需要点击”Submit”按钮来发送或者接受数据信息，然后等待服务器响应请求，页面重新加载。因为服务器每次都会返回一个新的页面， 所以传统的web应用有可能很慢而且用户交互不友好。使用AJAX技术， 就可以使Javascript通过XMLHttpRequest对象直接与服务器进行交互。通过HTTP Request， 一个web页面可以发送一个请求到web服务器并且接受web服务器返回的信息(不用重新加载页面)，展示给用户的还是通一个页面，用户感觉页面刷新，也看不到到Javascript后台进行的发送请求和接受响应。  
**AJAX请求总共有多少种CALLBACK。**

Ajax请求总共有八种Callback

onSuccess  
onFailure  
onUninitialized  
onLoading  
onLoaded  
onInteractive  
onComplete  
onException

**Ajax和**[**JavaScript**](http://lib.csdn.net/base/javascript)**的区别。**

javascript是一种在浏览器端执行的脚本语言，Ajax是一种创建交互式网页应用的开发技术 ，它是利用了一系列相关的技术其中就包括javascript。  
Javascript是由网景公司开发的一种脚本语言，它和sun公司的java语言是没有任何关系的，它们相似的名称只是一种行销策略。  
在一般的web开发中，javascript是在浏览器端执行的，我们可以用javascript控制浏览器的行为和内容。

**介绍一下XMLHttpRequest对象的常用方法和属性。**  open(“method”,”URL”)方法， 建立对服务器的调用，第一个参数是HTTP请求，方式可以为GET，POST或任何服务器所支持的您想调用的方式。第二个参数是请求页面的URL。  
    send()方法，发送具体请求  
    abort()方法，停止当前请求  
    readyState属性   请求的状态 有5个可取值0=未初始化 ，1=正在加载，2=以加载，3=交互中，4=完成  
    responseText 属性  服务器的响应，表示为一个串  
    reponseXML 属性 服务器的响应，表示为XML  
    status   服务器的HTTP状态码，200对应ok  400对应not found

**AJAX都有哪些优点和缺点？**优点：

1、最大的一点是页面无刷新，用户的体验非常好。  
2、使用异步方式与服务器通信，具有更加迅速的响应能力。  
3、可以把以前一些服务器负担的工作转嫁到客户端，利用客户端闲置的能力来处理，减轻服务器和带宽的负担，节约空间和宽带租用成本。并且减轻服务器的负担，ajax的原则是“按需取数据”，可以最大程度的减少冗余请求，和响应对服务器造成的负担。  
4、基于标准化的并被广泛支持的技术，不需要下载插件或者小程序。

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。  
5、不容易调试。

**AJAX基础实现步骤**

<**script**>  
 *//当页面加载完毕后，执行以下代码* **window**.onload = **function**(){  
 **document**.getElementById(**"ok"**).**onclick** = **function**(){  
 *//1 创建XMLHttpRequest对象* **var** xhr = *ajaxFunction*();  
 */\*  
 \* 2 客户端与服务器端，建立连接  
 \*  
 \* open(method, url, asynch)  
 \* \* method:请求类型，类似 “GET”或”POST”的字符串  
 \* \* url:请求路径  
 \* \* asynch:表示请求是否要异步传输，默认值为true(异步)。  
 \*/* xhr.open(**"get"**,**"../testServlet?timeStamp="**+**new** Date().getTime()+**"&c=9"**,**true**);  
 */\*  
 \* 3 客户端向服务器端发送请求  
 \*  
 \* send()方法  
 \* \* 如果请求类型是GET方式的话，send()方法发送的请求数据，服务器端接收不到.这个步骤是不能省略的！  
 \*/* xhr.send(**"a=7&b=8"**); *//send(null);  
 /\*  
 \* 4 服务器端响应数据  
 \*  
 \* readyState 属性表示Ajax请求的当前状态。它的值用数字代表。  
 0 代表未初始化。 还没有调用 open 方法  
 1 代表正在加载。 open 方法已被调用，但 send 方法还没有被调用  
 2 代表已加载完毕。send 已被调用。请求已经开始  
 3 代表交互中。服务器正在发送响应  
 4 代表完成。响应发送完毕  
  
 常用状态码及其含义：  
 404 没找到页面(not found)  
 403 禁止访问(forbidden)  
 500 内部服务器出错(internal service error)  
 200 一切正常(ok)  
 304 没有被修改(not modified)(服务器返回304状态，表示源文件没有被修改 )  
 \*/* xhr.onreadystatechange = **function**(){  
 *// alert(xhr.readyState);  
 // alert(xhr.status);* **if**(xhr.**readyState**==4){  
 **if**(xhr.**status**==200||xhr.**status**==304){  
 **var** data = xhr.**responseText**;  
  
 alert(data);  
 }  
 }  
 }  
 }  
 }  
  
 **function** *ajaxFunction*(){  
 **var** xmlHttp;  
 **try**{ *// Firefox, Opera 8.0+, Safari* xmlHttp=**new** XMLHttpRequest();  
 }  
 **catch** (e){  
 **try**{*// Internet Explorer* xmlHttp=**new** ActiveXObject(**"Msxml2.XMLHTTP"**);  
 }  
 **catch** (e){  
 **try**{  
 xmlHttp=**new** ActiveXObject(**"Microsoft.XMLHTTP"**);  
 }  
 **catch** (e){}  
 }  
 }  
 **return** xmlHttp;  
 }  
</**script**>

**JQuery中$.ajax()方法参数详解**

url: 要求为String类型的参数，（默认为当前页地址）发送请求的地址。

type: 要求为String类型的参数，请求方式（post或get）默认为get。注意其他http请求方法，例如put和delete也可以使用，但仅部分浏览器支持。

timeout: 要求为Number类型的参数，设置请求超时时间（毫秒）。此设置将覆盖$.ajaxSetup()方法的全局设置。

async：要求为Boolean类型的参数，默认设置为true，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为false。注意，同步请求将锁住浏览器，用户其他操作必须等待请求完成才可以执行。

cache：要求为Boolean类型的参数，默认为true（当dataType为script时，默认为false）。设置为false将不会从浏览器缓存中加载请求信息。

data: 要求为Object或String类型的参数，发送到服务器的数据。如果已经不是字符串，将自动转换为字符串格式。get请求中将附加在url后。防止这种自动转换，可以查看processData选项。对象必须为key/value格式，例如{foo1:"bar1",foo2:"bar2"}转换为&foo1=bar1&foo2=bar2。如果是数组，JQuery将自动为不同值对应同一个名称。例如{foo:["bar1","bar2"]}转换为&foo=bar1&foo=bar2。

dataType: 要求为String类型的参数，预期服务器返回的数据类型。如果不指定，JQuery将自动根据http包mime信息返回responseXML或responseText，并作为回调函数参数传递。

可用的类型如下：

       xml：返回XML文档，可用JQuery处理。

       html：返回纯文本HTML信息；包含的script标签会在插入DOM时执行。

       script：返回纯文本JavaScript代码。不会自动缓存结果。除非设置了cache参数。

注意：在远程请求时（不在同一个域下），所有post请求都将转为get请求。

       json：返回JSON数据。

       jsonp：JSONP格式。使用SONP形式调用函数时，例如myurl?callback=?，JQuery将自动替换后一个“?”为正确的函数名，以执行回调函数。

       text：返回纯文本字符串。

beforeSend：要求为Function类型的参数，发送请求前可以修改XMLHttpRequest对象的函数，例如添加自定义HTTP头。在beforeSend中如果返回false可以取消本次ajax请求。

complete：要求为Function类型的参数，请求完成后调用的回调函数（请求成功或失败时均调用）。

success：要求为Function类型的参数，请求成功后调用的回调函数，有两个参数。

         (1)由服务器返回，并根据dataType参数进行处理后的数据。

         (2)描述状态的字符串。

error：要求为Function类型的参数，请求失败时被调用的函数。该函数有3个参数，即XMLHttpRequest对象、错误信息、捕获的错误对象(可选)。

contentType：要求为String类型的参数，当发送信息至服务器时，内容编码类型默认为"application/x-www-form-urlencoded"。该默认值适合大多数应用场合。

dataFilter：要求为Function类型的参数，给Ajax返回的原始数据进行预处理的函数。提供data和type两个参数。data是Ajax返回的原始数据，type是调用jQuery.ajax时提供的dataType参数。函数返回的值将由jQuery进一步处理。

global：要求为Boolean类型的参数，默认为true。表示是否触发全局ajax事件。设置为false将不会触发全局ajax事件，ajaxStart或ajaxStop可用于控制各种ajax事件。

ifModified：要求为Boolean类型的参数，默认为false。仅在服务器数据改变时获取新数据。服务器数据改变判断的依据是Last-Modified头信息。默认值是false，即忽略头信息。

jsonp：要求为String类型的参数，在一个jsonp请求中重写回调函数的名字。该值用来替代在"callback=?"这种GET或POST请求中URL参数里的"callback"部分。

username：要求为String类型的参数，用于响应HTTP访问认证请求的用户名。

password：要求为String类型的参数，用于响应HTTP访问认证请求的密码。

processData：要求为Boolean类型的参数，默认为true。默认情况下，发送的数据将被转换为对象（从技术角度来讲并非字符串）以配合默认内容类型"application/x-www-form-urlencoded"。如果要发送DOM树信息或者其他不希望转换的信息，请设置为false。

scriptCharset：要求为String类型的参数，只有当请求时dataType为"jsonp"或者"script"，并且type是GET时才会用于强制修改字符集(charset)。通常在本地和远程的内容编码不同时使用。

案例代码：

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 <**script**>  
 $(**function**(){  
 $(**'#send'**).click(**function**(){  
 $.ajax({  
 **type**: **"GET"**,  
 **url**: **"test.json"**,  
 **data**: {**username**:$(**"#username"**).val(), **content**:$(**"#content"**).val()},  
 **dataType**: **"json"**,  
 success: **function**(data){  
 $(**'#resText'**).empty(); *//清空resText里面的所有内容* **var** html = **''**;  
 $.each(data, **function**(commentIndex, comment){  
 html += **'<div class="comment"><h6>'** + comment[**'username'**]  
 + **':</h6><p class="para"'** + comment[**'content'**]  
 + **'</p></div>'**;  
 });  
 $(**'#resText'**).**html**(html);  
 }  
 });  
 });  
 });  
 </**script**>  
</**head**>  
<**body**>  
  
</**body**>  
</**html**>

**Javascript中callee和caller的作用？**

1. caller

caller返回一个函数的引用，这个函数调用了当前的函数。  
使用这个属性要注意:  
1 这个属性只有当函数在执行时才有用  
2 如果在[JavaScript](http://lib.csdn.net/base/javascript)程序中，函数是由顶层调用的，则返回null

举个栗子：

<**script**>  
 **var** *a* = **function**() {  
 alert(*a*.**caller**);  
 }  
 **var** *b* = **function**() {  
 *a*();  
 }  
 *b*();  
</**script**>

上面的代码中，b调用了a，那么a.caller返回的是b的引用，结果如下:

<**script**>  
 **var** *b* = **function**() {  
 a();  
 }  
</**script**>

如果直接调用a(即a在任何函数中被调用，也就是顶层调用),返回null:

<**script**>  
 **var** *a* = **function**() {  
 alert(*a*.**caller**);  
 }  
 **var** *b* = **function**() {  
 *a*();  
 }  
 *//b();   
 a*();  
</**script**>

输出结果:

null

1. callee

callee放回正在执行的函数本身的引用，它是arguments的一个属性  
使用callee时要注意:  
1 这个属性只有在函数执行时才有效  
2 它有一个length属性，可以用来获得形参的个数，因此可以用来比较形参和实参个数是否一致，即比较arguments.length是否等于arguments.callee.length  
3 它可以用来递归匿名函数。

举个栗子：

<**script**>  
 **var** *a* = **function**() {  
 alert(arguments.**callee**);  
 }  
 **var** *b* = **function**() {  
 *a*();  
 }  
 *b*();  
</**script**>

a在b中被调用，但是它返回了a本身的引用，结果如下:

<**script**>  
 **var** *a* = **function**() {  
 alert(arguments.**callee**);  
 }  
</**script**>

**一次完整的HTTP事务是怎样的一个过程？**

当我们在浏览器的地址栏输入 [www.linux178.com](http://www.linux178.com/) ，然后回车，回车这一瞬间到看到页面到底发生了什么呢？

域名解析 --> 发起TCP的3次握手 --> 建立TCP连接后发起http请求 --> 服务器响应http请求，浏览器得到html代码 --> 浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） --> 浏览器对页面进行渲染呈现给用户

以下就是上面过程的一一分析，我们就以Chrome浏览器为例：

1. 域名解析

首先Chrome浏览器会解析 www.linux178.com 这个域名（准确的叫法应该是主机名）对应的IP地址。怎么解析到对应的IP地址？

1. Chrome浏览器会首先搜索浏览器自身的DNS缓存（缓存时间比较短，大概只有1分钟，且只能容纳1000条缓存），看自身的缓存中是否有www.linux178.com 对应的条目，而且没有过期，如果有且没有过期则解析到此结束。 注：我们怎么查看Chrome自身的缓存？可以使用 chrome://net-internals/#dns 来进行查看
2. 如果浏览器自身的缓存里面没有找到对应的条目，那么Chrome会搜索操作系统自身的DNS缓存,如果找到且没有过期则停止搜索解析到此结束. 注：怎么查看操作系统自身的DNS缓存，以Windows系统为例，可以在命令行下使用 ipconfig /displaydns 来进行查看
3. 如果在Windows系统的DNS缓存也没有找到，那么尝试读取hosts文件（位于C:\Windows\System32\drivers\etc），看看这里面有没有该域名对应的IP地址，如果有则解析成功。
4. 如果在hosts文件中也没有找到对应的条目，浏览器就会发起一个DNS的系统调用，就会向本地配置的首选DNS服务器（一般是电信运营商提供的，也可以使用像Google提供的DNS服务器）发起域名解析请求（通过的是UDP协议向DNS的53端口发起请求，这个请求是递归的请求，也就是运营商的DNS服务器必须得提供给我们该域名的IP地址），运营商的DNS服务器首先查找自身的缓存，找到对应的条目，且没有过期，则解析成功。如果没有找到对应的条目，则有运营商的DNS代我们的浏览器发起迭代DNS解析请求，它首先是会找根域的DNS的IP地址（这个DNS服务器都内置13台根域的DNS的IP地址），找打根域的DNS地址，就会向其发起请求（请问www.linux178.com这个域名的IP地址是多少啊？），根域发现这是一个顶级域com域的一个域名，于是就告诉运营商的DNS我不知道这个域名的IP地址，但是我知道com域的IP地址，你去找它去，于是运营商的DNS就得到了com域的IP地址，又向com域的IP地址发起了请求（请问www.linux178.com这个域名的IP地址是多少?）,com域这台服务器告诉运营商的DNS我不知道www.linux178.com这个域名的IP地址，但是我知道linux178.com这个域的DNS地址，你去找它去，于是运营商的DNS又向linux178.com这个域名的DNS地址（这个一般就是由域名注册商提供的，像万网，新网等）发起请求（请问www.linux178.com这个域名的IP地址是多少？），这个时候linux178.com域的DNS服务器一查，诶，果真在我这里，于是就把找到的结果发送给运营商的DNS服务器，这个时候运营商的DNS服务器就拿到了www.linux178.com这个域名对应的IP地址，并返回给Windows系统内核，内核又把结果返回给浏览器，终于浏览器拿到了www.linux178.com对应的IP地址，该进行下一步的动作了。 注：一般情况下是不会进行以下步骤的 如果经过以上的4个步骤，还没有解析成功，那么会进行如下步骤
5. 操作系统就会查找NetBIOS name Cache（NetBIOS名称缓存，就存在客户端电脑中的），那这个缓存有什么东西呢？凡是最近一段时间内和我成功通讯的计算机的计算机名和Ip地址，就都会存在这个缓存里面。什么情况下该步能解析成功呢？就是该名称正好是几分钟前和我成功通信过，那么这一步就可以成功解析。
6. 如果第5步也没有成功，那会查询WINS 服务器（是NETBIOS名称和IP地址对应的服务器）
7. 如果第6步也没有查询成功，那么客户端就要进行广播查找
8. 如果第7步也没有成功，那么客户端就读取LMHOSTS文件（和HOSTS文件同一个目录下，写法也一样） 如果第八步还没有解析成功，那么就宣告这次解析失败，那就无法跟目标计算机进行通信。只要这八步中有一步可以解析成功，那就可以成功和目标计算机进行通信。
9. 发起TCP的3次握手

拿到域名对应的IP地址之后，User-Agent（一般是指浏览器）会以一个随机端口（1024 < 端口 < 65535）向服务器的WEB程序（常用的有httpd,nginx等）80端口发起TCP的连接请求。这个连接请求（原始的http请求经过TCP/IP4层模型的层层封包）到达服务器端后（这中间通过各种路由设备，局域网内除外），进入到网卡，然后是进入到内核的TCP/IP协议栈（用于识别该连接请求，解封包，一层一层的剥开），还有可能要经过Netfilter防火墙（属于内核的模块）的过滤，最终到达WEB程序（本文就以Nginx为例），最终建立了TCP/IP的连接。

1） Client首先发送一个连接试探，ACK=0 表示确认号无效，SYN = 1 表示这是一个连接请求或连接接受报文，同时表示这个数据报不能携带数据，seq = x 表示Client自己的初始序号（seq = 0 就代表这是第0号包），这时候Client进入syn\_sent状态，表示客户端等待服务器的回复

2） Server监听到连接请求报文后，如同意建立连接，则向Client发送确认。TCP报文首部中的SYN 和 ACK都置1 ，ack = x + 1表示期望收到对方下一个报文段的第一个数据字节序号是x+1，同时表明x为止的所有数据都已正确收到（ack=1其实是ack=0+1,也就是期望客户端的第1个包），seq = y 表示Server 自己的初始序号（seq=0就代表这是服务器这边发出的第0号包）。这时服务器进入syn\_rcvd，表示服务器已经收到Client的连接请求，等待client的确认。

3） Client收到确认后还需再次发送确认，同时携带要发送给Server的数据。ACK 置1 表示确认号ack= y + 1 有效（代表期望收到服务器的第1个包），Client自己的序号seq= x + 1（表示这就是我的第1个包，相对于第0个包来说的），一旦收到Client的确认之后，这个TCP连接就进入Established状态，就可以发起http请求了。

TCP为什么需要三次握手呢？

举个例子：

假设一个老外在故宫里面迷路了，看到了小明，于是就有下面的对话：

老外： Excuse me，Can you Speak English?

小明： yes 。

老外： OK,I want ...

在问路之前，老外先问小明是否会说英语，小明回答是的，这时老外才开始问路

两个计算机通信是靠协议（目前流行的TCP/IP协议）来实现,如果2个计算机使用的协议不一样，那是不能进行通信的，所以这个3次握手就相当于试探一下对方是否遵循TCP/IP协议，协商完成后就可以进行通信了，当然这样理解不是那么准确。

为什么HTTP协议要基于TCP来实现？

目前在Internet中所有的传输都是通过TCP/IP进行的，HTTP协议作为TCP/IP模型中应用层的协议也不例外，TCP是一个端到端的可靠的面向连接的协议，所以HTTP基于传输层TCP协议不用担心数据的传输的各种问题。

1. 建立TCP连接后发起http请求

经过TCP 3次握手之后，浏览器发起了http的请求，使用的http的方法 GET 方法

1. 服务器端响应http请求，浏览器得到html代码

在这里介绍一下常见其他状态码含义：

1xx: 信息性状态码

100, 101

2xx: 成功状态码

200：OK

3xx: 重定向状态码

301: 永久重定向, Location响应首部的值仍为当前URL，因此为隐藏重定向; 302: 临时重定向，显式重定向, Location响应首部的值为新的URL

304：Not Modified 未修改，比如本地缓存的资源文件和服务器上比较时，发现并没有修改，服务器返回一个304状态码， 告诉浏览器，你不用请求该资源，直接使用本地的资源即可。

4xx: 客户端错误状态码

404: Not Found 请求的URL资源并不存在

5xx: 服务器端错误状态码

500: Internal Server Error 服务器内部错误

502: Bad Gateway 前面代理服务器联系不到后端的服务器时出现

504：Gateway Timeout 这个是代理能联系到后端的服务器，但是后端的服务器在规定的时间内没有给代理服务器响应

1. 浏览器解析html代码，并请求html代码中的资源

浏览器拿到index.html文件后，就开始解析其中的html代码，遇到js/css/image等静态资源时，就向服务器端去请求下载（会使用多线程下载，每个浏览器的线程数不一样），这个时候就用上keep-alive特性了，建立一次HTTP连接，可以请求多个资源，下载资源的顺序就是按照代码里的顺序，但是由于每个资源大小不一样，而浏览器又多线程请求请求资源，显示的顺序并不一定是代码里面的顺序。

浏览器在请求静态资源时（在未过期的情况下），向服务器端发起一个http请求（询问自从上一次修改时间到现在有没有对资源进行修改），如果服务器端返回304状态码（告诉浏览器服务器端没有修改），那么浏览器会直接读取本地的该资源的缓存文件。

1. 浏览器对页面进行渲染呈现给用户

最后，浏览器利用自己内部的工作机制，把请求到的静态资源和html代码进行渲染，渲染之后呈现给用户。

### 自此一次完整的HTTP事务宣告完成.

**如何实现浏览器内多个标签页之间的通信？都有哪些方式？**

本题主要考察数据存储的知识，数据存储有本地和服务器存储两种方式，对于[前端开发](http://lib.csdn.net/base/javascript)来讲，只需要讲解用本地存储的方式来解决就好。当然也能知道服务器端的方式更好。本题的难易程度一般，只要能够说出思路就可以，至少说两种解决方法。

方法一：使用localStorage

使用localStorage.setItem(key,value);添加内容，使用storage事件监听添加、修改、删除的动作

<**script**>  
 **window**.addEventListener(**"storage"**,**function**(event){  
 $(**"#name"**).val(event.key+**"="**+event.**newValue**);  
 });  
</**script**>

方法二：使用cookie+setInterval

<!DOCTYPE **html**>  
<**html lang="en"**>  
 <**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 </**head**>  
 <**body**>  
 <**input id="name"**>  
 <**input type="button" id="btnOK" value="发送"**>  
 <**script**>  
 $(**function**(){  
 $(**"#btnOK"**).click(**function**(){  
 **varname**=$(**"#name"**).val();  
 **document**.**cookie**=**"name="**+name;  
 });  
 });  
 *//获取Cookie天的内容* **function** *getKey*(key) {  
 **return JSON**.parse(**"{\""**+ **document**.**cookie**.replace(/;\s+/gim,**"\",\""**).replace(/=/gim, **"\":\""**) +**"\"}"**)[key];  
 }  
 *//每隔1秒获取Cookie的内容* setInterval(**function**(){  
 **console**.log(*getKey*(**"name"**));  
 },1000);  
 </**script**>  
 </**body**>  
</**html**>

**Html事件绑定有几种方式？**

 直接在DOM里绑定事件：<div onclick=”test()”></div> 这种方式称为原生事件或者属性事件

 在JS里通过onclick绑定：xxx.onclick = test

 Dom标准通过事件添加进行绑定：addEventListener("click",test, false) //第三个参数为是否支持事件捕捉

IE事件：attachEvent("onclick",test)

**AJAX跨域请求问题**

[JavaScript](http://lib.csdn.net/base/javascript)是一种在Web开发中经常使用的前端动态脚本技术。在JavaScript中，有一个很重要的安全性限制，被称为“Same-Origin Policy”（同源策略）。这一策略对于JavaScript代码能够访问的页面内容做了很重要的限制，即JavaScript只能访问与包含它的文档在同一域下的内容。

JavaScript这个安全策略在进行多iframe或多窗口编程、以及Ajax编程时显得尤为重要。根据这个策略，在baidu.com下的页面中包含的JavaScript代码，不能访问在google.com域名下的页面内容；甚至不同的子域名之间的页面也不能通过JavaScript代码互相访问。对于Ajax的影响在于，通过XMLHttpRequest实现的Ajax请求，不能向不同的域提交请求，例如，在abc.example.com下的页面，不能向def.example.com提交Ajax请求，等等。

然而，当进行一些比较深入的前端编程的时候，不可避免地需要进行跨域操作，这时候“同源策略”就显得过于苛刻。那么在讲解之前，我们先看看跨域的实例举例。我们先回顾一下域名地址的组成：

http:// www . google : 8080 / script/jquery.js

　　　　  http:// （协议号）

               www  （子域名）

             google （主域名）

               8080 （端口号）

script/jquery.js （请求的地址）

当协议、子域名、主域名、端口号中任意一各不相同时，都算不同的“域”；不同的域之间相互请求资源，就叫“跨域”。下面我们介绍三种AJAX跨域请求问题的解决方案。

1. “笨方法”——代理

比如在北京（[www.beijing.com/sever.php](http://www.beijing.com/sever.php)）和上海（[www.shanghai.com/sever.php](http://www.shanghai.com/sever.php)）各有一个服务器，北京的后端（[www.beijing.com/sever.php](http://www.beijing.com/sever.php)）直接访问上海的服务，然后把获取的响应值返回给前端。也就是北京的服务在后台做了一个代理，前端只需要访问北京的服务器也就相当与访问了上海的服务器。这种代理属于后台的技术，所以不展开叙述。

1. JSONP

JSONP跨域GET请求是一个常用的解决方案，下面我们来看一下JSONP跨域是如何实现的，并且探讨下JSONP跨域的原理。

利用在页面中创建<script>节点的方法向不同域提交HTTP请求的方法称为JSONP，这项技术可以解决跨域提交Ajax请求的问题。JSONP的工作原理如下所述：

假设在http://example1.com/index.[PHP](http://lib.csdn.net/base/php)这个页面中向http://example2.com/getinfo.php提交GET请求，我们可以将下面的JavaScript代码放在http://example1.com/index.php这个页面中来实现：

**var *eleScript***= **document**.createElement(**"script"**);  
***eleScript***.**type** = **"text/javascript"**;  
***eleScript***.**src** = **"http://example2.com/getinfo.php"**;  
**document**.getElementsByTagName(**"HEAD"**)[0].appendChild(***eleScript***);

当GET请求从http://example2.com/getinfo.php返回时，可以返回一段JavaScript代码，这段代码会自动执行，可以用来负责调用http://example1.com/index.php页面中的一个callback函数。

JSONP的优点是：它不像XMLHttpRequest对象实现的Ajax请求那样受到同源策略的限制；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；并且在请求完毕后可以通过调用callback的方式回传结果。

JSONP的缺点则是：它只支持GET请求而不支持POST等其它类型的HTTP请求；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。

再看下面这个栗子：

<**script**>  
 **var *qsData*** = {**'searchWord'**:$(**"#searchWord"**).attr(**"value"**),**'currentUserId'**:  
 $(**"#currentUserId"**).attr(**"value"**),**'conditionBean.pageSize'**:$(**"#pageSize"**).attr(**"value"**)};  
 $.ajax({  
 **async**:**false**,  
 **url**: http:*//跨域的dns/document!searchJSONResult.action,* type: **"GET"**,  
 dataType: **'jsonp'**,  
 jsonp: **'jsoncallback'**,  
 data: ***qsData***,  
 timeout: 5000,  
 beforeSend: *function*(){  
 *//jsonp 方式此方法不被触发.原因可能是dataType如果指定为jsonp的话,就已经不是ajax事件了* },  
 success: *function* (json) {*//客户端jquery预先定义好的callback函数,成功获取跨域服务器上的json数据后,会动态执行这个callback函数* **if**(json.actionErrors.**length**!=0){  
 alert(json.actionErrors);  
 }  
 genDynamicContent(***qsData***,type,json);  
 },  
 complete: *function*(XMLHttpRequest, textStatus){  
 $.unblockUI({ **fadeOut**: 10 });  
 },  
 error: *function*(xhr){  
 *//jsonp 方式此方法不被触发.原因可能是dataType如果指定为jsonp的话,就已经不是ajax事件了  
 //请求出错处理* alert(**"请求出错(请检查相关度网络状况.)"**);  
 }  
 });  
</**script**>

有时候可能也会看到这种写法：

<**script**>  
 $.getJSON(**"http://跨域的dns/document!searchJSONResult.action?name1="**+value1+**"&jsoncallback=?"**,  
 **function**(json) {  
 **if** (json.属性名 == 值) {  
 *// 执行代码* }  
 });  
</**script**>

这种方式其实是上例$.ajax({..}) api的一种高级封装，有些$.ajax api底层的参数就被封装而不可见了。

这样，[jQuery](http://lib.csdn.net/base/jquery)就会拼装成如下的url get请求：

http://跨域的dns/document!searchJSONResult.action?&jsoncallback=jsonp1236827957501&\_=1236828192549&searchWord=

%E7%94%A8%E4%BE%8B&currentUserId=5351&conditionBean.pageSize=15

JSONP原理介绍

JSONP的最基本的原理是：动态添加一个<script>标签，而script标签的src属性是没有跨域的限制的。这样说来，这种跨域方式其实与ajax XmlHttpRequest协议无关了。

这样其实"jQuery AJAX跨域问题"就成了个伪命题，jquery $.ajax方法名有误导人之嫌。

如果设为dataType: 'jsonp'，这个$.ajax方法就和ajax XmlHttpRequest没什么关系了，取而代之的则是JSONP协议。JSONP是一个非官方的协议，它允许在服务器端集成Script tags返回至客户端，通过javascript callback的形式实现跨域访问。

JSONP即JSON with Padding。由于同源策略的限制，XmlHttpRequest只允许请求当前源（域名、协议、端口）的资源。如果要进行跨域请求， 我们可以通过使用html的script标记来进行跨域请求，并在响应中返回要执行的script代码，其中可以直接使用JSON传递javascript对象。 这种跨域的通讯方式称为JSONP。

jsonCallback 函数jsonp1236827957501(....)：是浏览器客户端注册的，获取跨域服务器上的json数据后，回调的函数

Jsonp的执行过程如下：

1.首先在客户端注册一个callback (如:'jsoncallback'), 然后把callback的名字(如:jsonp1236827957501)传给服务器。注意：服务端得到callback的数值后，要用jsonp1236827957501(......)把将要输出的json内容包括起来，此时，服务器生成 json 数据才能被客户端正确接收。

2.然后以 javascript 语法的方式，生成一个function， function 名字就是传递上来的参数 3.'jsoncallback'的值 jsonp1236827957501 .

4.最后将 json 数据直接以入参的方式，放置到 function 中，这样就生成了一段 js 语法的文档，返回给客户端。

5.客户端浏览器，解析script标签，并执行返回的 javascript 文档，此时javascript文档数据，作为参数， 传入到了客户端预先定义好的 callback 函数(如上例中jquery $.ajax()方法封装的的success: function (json))里。

可以说jsonp的方式原理上和<script src="http://跨域/...xx.js"></script>是一致的(qq空间就是大量采用这种方式来实现跨域数据交换的)。JSONP是一种脚本注入(Script Injection)行为，所以有一定的安全隐患。

那jquery为什么不支持post方式跨域呢？

虽然采用post+动态生成iframe是可以达到post跨域的目的,但这样做是一个比较极端的方式，不建议采用。

也可以说get方式的跨域是合法的，post方式从安全角度上，被认为是不合法的，万不得已还是不要剑走偏锋。

1. XHR2（推荐方法,较新方法）

“XHR2” 全称 “XMLHttpRequest Level2” 是HTML5提供的方法，对跨域访问提供了很好的支持，并且还有一些新的功能。

需注意： IE10一下的版本都不支持此方法

使用方法：

只需要在服务器端头部加上下面两句代码：

　 header( "Access-Control-Allow-Origin:\*" );

　 header( "Access-Control-Allow-Methods:POST,GET" );