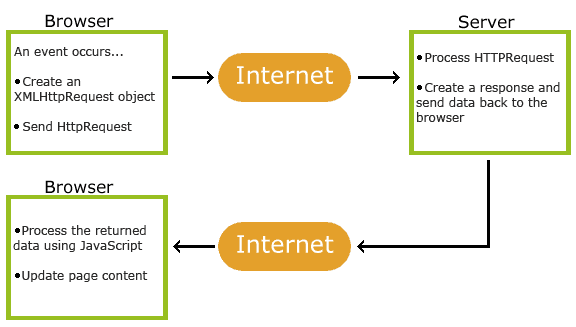
AJAX的工作原理

# AJAX工作原理示意图

XHR：XMLHttpRequest



# XMLHttpRequest属性和方法介绍

## 创建XMLHttpRequest对象

var xhr = new XMLHttpRequest();

## XMLHttpRequest属性

### xhr.onreadystatechange：指定事件；

### xhr.readyState

### xhr.status

## XMLHttpRequest方法

### xhr.open(method,url,true)

### xhr.send(string)

# 创建 XMLHttpRequest 对象

## XMLHttpRequest对象

**XMLHttpRequest** 是 AJAX 的基础。所有现代浏览器均支持 **XMLHttpRequest 对象**（IE5 和 IE6 使用 **ActiveXObject**）。

**XMLHttpRequest** 用于**在后台与服务器交换数据**。可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

## 创建XMLHttpRequest对象

所有现代浏览器（IE7+、Firefox、Chrome、Safari 以及 Opera）均内建 XMLHttpRequest 对象。

创建 XMLHttpRequest 对象的语法：

**variable=new XMLHttpRequest();**

老版本的 Internet Explorer （IE5 和 IE6）使用 ActiveX 对象：

**variable=new ActiveXObject("Microsoft.XMLHTTP");**

为了应对所有的现代浏览器，包括 IE5 和 IE6，请检查浏览器是否支持 XMLHttpRequest 对象。**如果支持，则创建 XMLHttpRequest 对象；如果不支持，则创建 ActiveXObject** ：

var xmlhttp;

if (window.XMLHttpRequest) {

// IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码

xmlhttp=new XMLHttpRequest();

}else{

// IE6, IE5 浏览器执行代码

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

}

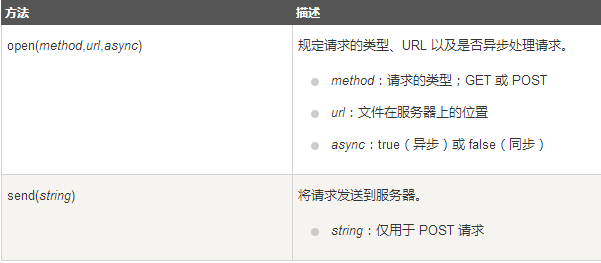
# XHR请求：AJAX-向服务器发送请求

**XMLHttpRequest 对象**用于和服务器交换数据。

## 向服务器发送请求

如需将**请求**发送到服务器，我们使用 XMLHttpRequest 对象的 **open() 和 send()** 方法：





## Method：GET 还是 POST？

与 POST 相比，**GET 更简单也更快**，并且在大部分情况下都能用。

然而，在以下情况中，请使用 POST 请求：

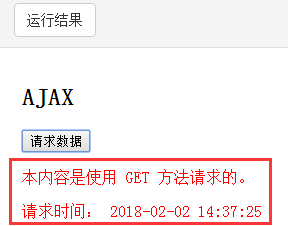
1. **无法使用缓存文件**（更新服务器上的文件或数据库）
2. **向服务器发送大量数据**（POST 没有数据量限制）

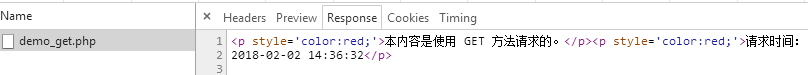
C．**发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠**

## GET请求

1. 一个简单的 GET 请求：







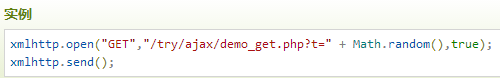
请求返回的数据：

<p style='color:red;'>本内容是使用 GET 方法请求的。</p><p style='color:red;'>请求时间：

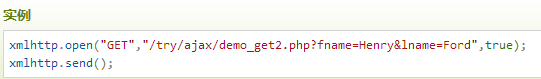
2018-02-02 14:37:25</p>

1. 在上面的例子中，您可能得到的是缓存的结果。

为了避免这种情况，请向 URL 添加一个唯一的 ID：



C．如果您希望通过 GET 方法发送信息，请向 URL 添加信息：



## POST请求

1. 一个简单的POST请求：



1. 如果需要**像 HTML 表单那样 POST 数据**，请使用 **setRequestHeader()** 来添加 HTTP 头。然后在 **send() 方法**中规定您希望发送的数据：



xmlhttp.open("POST","/try/ajax/demo\_post2.php",true); xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");

xmlhttp.send("fname=Henry&lname=Ford");

方法：setRequestHeader(header,value)



# open()方法参数介绍：

## url—获取服务器上的文件

open() 方法的 *url* 参数是服务器上文件的地址：

xmlhttp.open("GET","ajax\_test.html",true);

该文件可以是**任何类型的文件**，比如 .txt 和 .xml，或者**服务器脚本文件**，比如 .asp 和 .php （在传回响应之前，能够在服务器上执行任务）。

## 异步 - True 或 False？

AJAX 指的是**异步 JavaScript 和 XML（Asynchronous JavaScript and XML）**。**XMLHttpRequest 对象**如果要用于 AJAX 的话，其 open() 方法的 **async 参数**必须设置为 true：

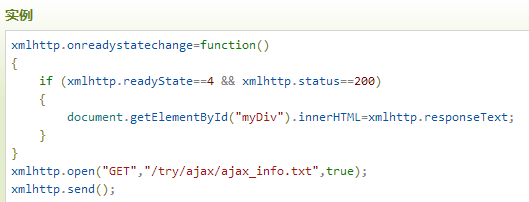
xmlhttp.open("GET","ajax\_test.html",true);

对于 web 开发人员来说，发送异步请求是一个巨大的进步。很多在服务器执行的任务都相当费时。AJAX 出现之前，这可能会引起应用程序挂起或停止。

通过 AJAX，JavaScript 无需等待服务器的响应，而是：

* **在等待服务器响应时**执行其他脚本
* **当响应就绪后**对响应进行处理
  1. async = true:

当使用 **async=true** 时，请规定**在响应处于 onreadystatechange 事件中的就绪状态时**执行的函数：



xmlhttp.onreadystatechange = function() {

if (xmlhttp.readyState==4 && xmlhttp.status==200) {

document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

}

}

xmlhttp.open("GET","/try/ajax/ajax\_info.txt",true);

xmlhttp.send();

* 1. async = false:

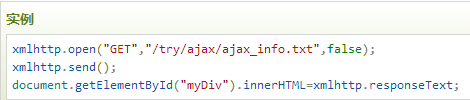
如需使用 **async=false**，请将 open() 方法中的第三个参数改为 false：

xmlhttp.open(&quot;GET&quot;,&quot;test1.txt&quot;,false);

我们不推荐使用 async=false，但是对于一些小型的请求，也是可以的。

请记住，JavaScript 会等到服务器响应就绪才继续执行。如果服务器繁忙或缓慢，应用程序会挂起或停止。

注意：当您使用 **async=false** 时，**请不要编写 onreadystatechange 函数** - 把代码放到 send() 语句后面即可：

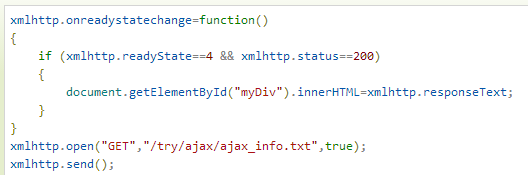


xmlhttp.open("GET","/try/ajax/ajax\_info.txt",false);

xmlhttp.send(); document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

## onreadystatechange

异步请求时：需要指定，在响应处于 **onreadystatechange** 事件中的**就绪状态**时执行的函数，其值就是一个function。



理解意思：就是当发送异步请求后，就会执行其他操作了，当此请求获得响应之后，就会执行该函数function。一般，该函数的定义在open和send方法之前。对于同步请求，onreadystatechange就没有意义了，发送同步请求之后，就会一直等待，直到获得响应，才会继续执行。所以，获得响应后的操作需要定义在open和send方法之后。

# XHR响应：AJAX-服务器响应

如需获得来自服务器的响应，请使用 **XMLHttpRequest 对象**的 r**esponseText 或 responseXML 属性**。



## responseText属性

如果来自服务器的响应**并非 XML**，请使用 **responseText 属性**。

responseText 属性返回**字符串形式**的响应，因此您可以这样使用：

document.getElementById("myDiv").innerHTML=**xmlhttp.responseText**;

## responseXML 属性

如果来自服务器的响应是 XML，而且需要作为 XML 对象进行解析，请使用 responseXML 属性：



# AJAX - onreadystatechange 事件

## onreadystatechange 事件介绍：

当请求被发送到服务器时，我们需要执行一些基于响应的任务。

**XMLHttpRequest对象具有readyState 属性，**这个属性存有 **XMLHttpRequest** 的状态信息。每当 **readyState值改变**时，就会**触发 onreadystatechange指定的事件**。

下面是 XMLHttpRequest 对象的三个重要的属性：

xmlHttpRequest.**onreadystatechange**=function(){…}

xmlHttpRequest.**readyState**==4

xmlHttpRequest.**status** == 200

**xmlhttp.readyState==4 && xmlhttp.status==200**的解释：**请求完成并且成功返回**



## xmlhttp.readyState的值及解释：

0：请求未初始化（还没有调用 open()）。

1：请求已经建立（服务器连接已建立），但是还没有发送（还没有调用 send()）。

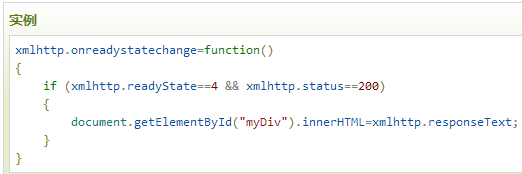
2：请求已发送，正在处理中（通常现在可以从响应中获取内容头）。

3：**请求处理中**；通常响应中已有部分数据可用了，但是服务器还没有完成响应的生成。

4：**响应已完成**；您可以获取并使用服务器的响应了。

在 onreadystatechange 事件中，我们规定**当服务器响应已做好被处理的准备时**所执行的任务。

当 **readyState** **等于 4 且状态status为 200** 时，表示响应已就绪：



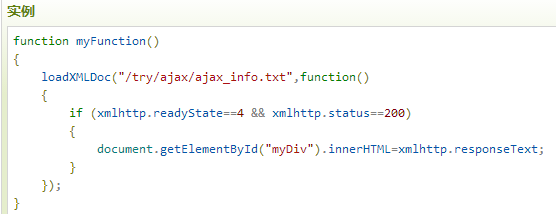
**注意：** onreadystatechange 事件被触发 5 次（0 - 4），对应着 readyState 的每个变化。

## 使用回调函数

**回调函数**是一种以**参数形式**传递给**另一个函数**的**函数**。

如果您的网站上存在**多个 AJAX 任务**，那么您应该为**创建 XMLHttpRequest 对象**编写一个**标准的函数**，并为每个 AJAX 任务调用该函数。

该函数调用应该包含 URL 以及发生 onreadystatechange 事件时执行的任务（每次调用可能不尽相同）：



<!DOCTYPE html>

<html>

<head>

<script>

var xmlhttp;

**function loadXMLDoc(url,cfunc)//定义一个函数，参数为url和回调函数，当请求url成功后，执行回调函数**

**{**

**if (window.XMLHttpRequest)**

**{// IE7+, Firefox, Chrome, Opera, Safari 代码**

**Xmlhttp = new XMLHttpRequest();**

**}**

**else**

**{// IE6, IE5 代码**

**xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");**

**}**

**xmlhttp.onreadystatechange = cfunc;**

**xmlhttp.open("GET",url,true);**

**xmlhttp.send();**

**}**

function myFunction()

{

**loadXMLDoc("/try/ajax/ajax\_info.txt",function()**

**{**

**if (xmlhttp.readyState==4 && xmlhttp.status==200)**

**{**

**document.getElementById("myDiv").innerHTML=xmlhttp.responseText;**

**}**

**});//函数以参数的形式传递给loadXMLDoc函数。**

}

</script>

</head>

<body>

<div id="myDiv"><h2>使用 AJAX 修改文本内容</h2></div>

<button type="button" **onclick="myFunction()"**>修改内容</button>

</body>

</html>

# 附录：

## 服务器常用的状态码及其对应的含义如下：

**200：服务器响应正常**。

304：**该资源在上次请求之后没有任何修改**（这通常用于浏览器的缓存机制，使用GET请求时尤其需要注意）。

400：无法找到请求的资源。

401：访问资源的权限不够。

403：没有权限访问资源。

**404：需要访问的资源不存在，Not Found。**

405：需要访问的资源被禁止。

407：访问的资源需要代理身份验证。

414：请求的URL太长。

**500：服务器内部错误。**

## 更多的状态码：

100——客户必须继续发出请求

101——客户要求服务器根据请求转换HTTP协议版本

200——交易成功，OK

201——提示知道新文件的URL

202——接受和处理、但处理未完成

203——返回信息不确定或不完整

204——请求收到，但返回信息为空

205——服务器完成了请求，用户代理必须复位当前已经浏览过的文件

206——服务器已经完成了部分用户的GET请求

300——请求的资源可在多处得到

301——删除请求数据

302——在其他地址发现了请求数据

303——建议客户访问其他URL或访问方式

304——客户端已经执行了GET，但文件未变化

305——请求的资源必须从服务器指定的地址得到

306——前一版本HTTP中使用的代码，现行版本中不再使用

307——申明请求的资源临时性删除

400——错误请求，如语法错误

401——请求授权失败

402——保留有效ChargeTo头响应

403——请求不允许

404——没有发现文件、查询或URl

405——用户在Request-Line字段定义的方法不允许

406——根据用户发送的Accept拖，请求资源不可访问

407——类似401，用户必须首先在代理服务器上得到授权

408——客户端没有在用户指定的时间内完成请求

409——对当前资源状态，请求不能完成

410——服务器上不再有此资源且无进一步的参考地址

411——服务器拒绝用户定义的Content-Length属性请求

412——一个或多个请求头字段在当前请求中错误

413——请求的资源大于服务器允许的大小

414——请求的资源URL长于服务器允许的长度

415——请求资源不支持请求项目格式

416——请求中包含Range请求头字段，在当前请求资源范围内没有range指示值，请求也不包含If-Range请求头字段

417——服务器不满足请求Expect头字段指定的期望值，如果是代理服务器，可能是下一级服务器不能满足请求

500——服务器产生内部错误

501——服务器不支持请求的函数

502——服务器暂时不可用，有时是为了防止发生系统过载

503——服务器过载或暂停维修

504——关口过载，服务器使用另一个关口或服务来响应用户，等待时间设定值较长

505——服务器不支持或拒绝支请求头中指定的HTTP版本

**合起来**

**1xx:信息响应类**，表示接收到请求并且继续处理

2xx:处理成功响应类，表示动作被成功接收、理解和接受

**3xx:重定向响应类**，为了完成指定的动作，必须接受进一步处理

**4xx:客户端错误**，客户请求包含语法错误或者是不能正确执行

**5xx:服务端错误**，服务器不能正确执行一个正确的请求