**AJAX编程**

目录

[目录 2](#_Toc493657288)

[第1章 AJAX编程 3](#_Toc493657289)

[1.1 异步 4](#_Toc493657290)

[1.2 XMLHttpRequest 4](#_Toc493657291)

[1.2.1 请求 5](#_Toc493657292)

[1.2.2 响应 5](#_Toc493657293)

[1.2.3 API详解 6](#_Toc493657294)

[1.3 XML 8](#_Toc493657295)

[1.3.1 语法规则 8](#_Toc493657296)

[1.4 JSON 9](#_Toc493657297)

[1.4.1 语法规则 9](#_Toc493657298)

[1.4.2 JSON解析 9](#_Toc493657299)

[1.5 兼容性 9](#_Toc493657300)

[第2章 模板引擎 10](#_Toc493657301)

[2.1 原理剖析 10](#_Toc493657302)

[2.2 流行模板引擎 10](#_Toc493657303)

[2.3 artTemplate 10](#_Toc493657304)

[第3章 同源&跨域 12](#_Toc493657305)

[3.1 同源 12](#_Toc493657306)

[3.2 跨域 12](#_Toc493657307)

[3.3 跨域解决方案 13](#_Toc493657308)

[3.4 JSONP 13](#_Toc493657309)

[第4章 封装AJAX工具函数 14](#_Toc493657310)

[第5章 jQuery中的Ajax 15](#_Toc493657311)

[5.1 XMLHttpRequest封装 15](#_Toc493657312)

[5.2 全局配置 16](#_Toc493657313)

[5.3 JSONP封装 16](#_Toc493657314)

[第6章 XMLHttpRequest 2.0 16](#_Toc493657315)

[6.1 设置超时 17](#_Toc493657316)

[6.2 FormData 17](#_Toc493657317)

[6.3 二进制 18](#_Toc493657318)

[6.4 上传进度 19](#_Toc493657319)

# AJAX编程

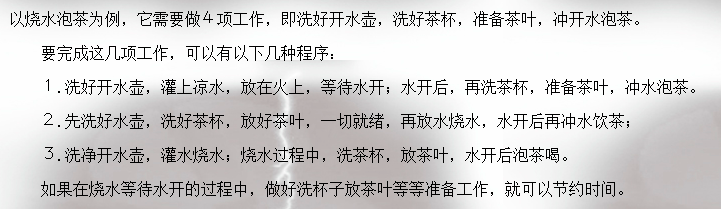
即 Asynchronous Javascript And XML，AJAX 不是一门的新的语言，而是对现有技术的综合利用。

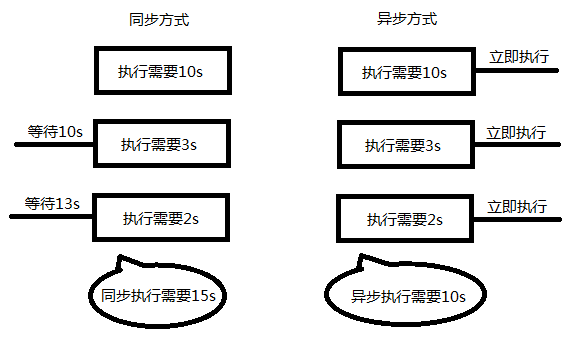
本质是在HTTP协议的基础上以异步的方式与服务器进行通信。

## 异步

指某段程序执行时不会阻塞其它程序执行，其表现形式为程序的执行顺序不依赖程序本身的书写顺序，相反则为同步。

其优势在于不阻塞程序的执行，从而提升整体执行效率。

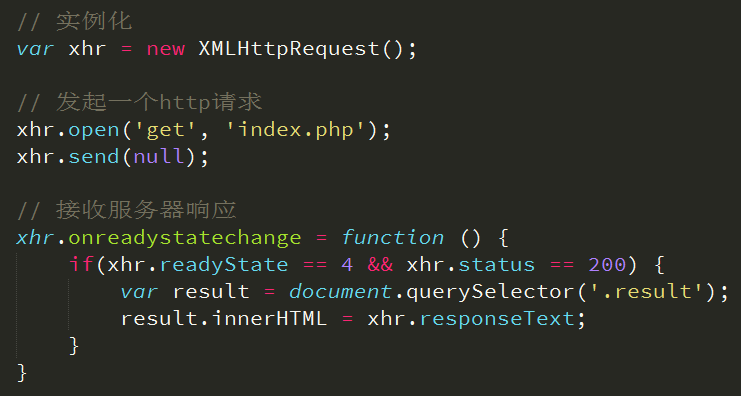




## XMLHttpRequest

浏览器内建对象，用于在后台与服务器通信(交换数据) ，由此我们便可实现对网页的部分更新，而不是刷新整个页面。

下面是一个简单的例子



由于XMLHttpRequest本质基于HTTP协议实现通信，所以结合HTTP协议和上面的例子我们分析得出如下结果：

### 请求

HTTP请求3个组成部分与XMLHttpRequest方法的对应关系

1、请求行



2、请求头



3、请求主体

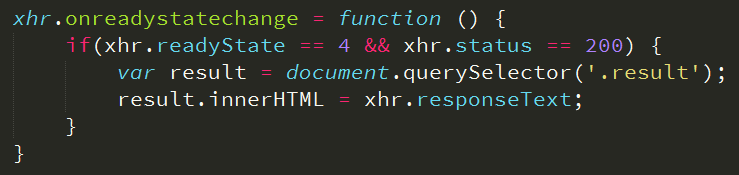


### 响应

HTTP响应是由服务端发出的，作为客户端更应关心的是响应的结果。

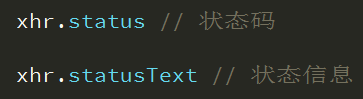
HTTP响应3个组成部分与XMLHttpRequest方法或属性的对应关系。

由于服务器做出响应需要时间（比如网速慢等原因），所以我们需要监听服务器响应的状态，然后才能进行处理。

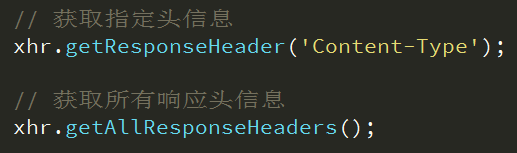


onreadystatechange是Javascript的事件的一种，其意义在于监听XMLHttpRequest的状态

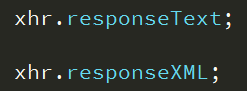
1、获取状态行（包括状态码&状态信息）



2、获取响应头



3、响应主体



### API详解

xhr.open() 发起请求，可以是get、post方式

xhr.setRequestHeader() 设置请求头

xhr.send() 发送请求主体get方式使用xhr.send(null)

xhr.onreadystatechange = function () {} 监听响应状态

xhr.readyState = 0时，UNSENT open尚未调用

xhr.readyState = 1时，OPENED open已调用

xhr.readyState = 2时，HEADERS\_RECEIVED 接收到头信息

xhr.readyState = 3时，LOADING 接收到响应主体

xhr.readyState = 4时，DONE 响应完成

不用记忆状态，只需要了解有状态变化这个概念

xhr.status表示响应码，如200

xhr.statusText表示响应信息，如OK

xhr.getAllResponseHeaders() 获取全部响应头信息

xhr.getResponseHeader('key') 获取指定头信息

xhr.responseText、xhr.responseXML都表示响应主体

**注：GET和POST请求方式的差异（面试题）**

1、GET没有请求主体，使用xhr.send(null)

2、GET可以通过在请求URL上添加请求参数

3、POST可以通过xhr.send('name=itcast&age=10')

4、POST需要设置



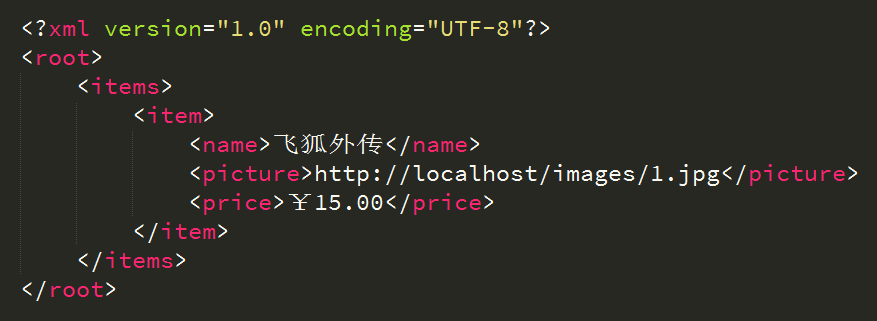
5、GET效率更好（应用多）

6、GET大小限制约4K，POST则没有限制

问题：如何获取复杂数据呢？

## XML

XML是一种标记语言，很类似HTML，其宗旨是用来传输数据，具有自我描述性。



### 语法规则

1、必须有一个根元素

2、不可有空格、不可以数字或.开头、大小写敏感

3、不可交叉嵌套

4、属性双引号（浏览器自动修正成双引号了）

5、特殊符号要使用实体

6、注释和HTML一样

虽然可以描述和传输复杂数据，但是其解析过于复杂并且体积较大，所以实现开发已经很少使用了。

## JSON

即 JavaScript Object Notation，另一种轻量级的文本数据交换格式，独立于语言。

### 语法规则

1、数据在名称/值对中

2、数据由逗号分隔(最后一个健/值对不能带逗号)

3、花括号保存对象方括号保存数组

4、使用双引号

### JSON解析

JSON数据在不同语言进行传输时，类型为字符串，不同的语言各自也都对应有解析方法，需要解析完成后才能读取

**1、Javascript 解析方法**

eavl()、JSON对象 JSON.parse()、JSON.stringify()；

JSON兼容处理json2.js

**2、PHP解析方法**

json\_encode()、json\_decode()

总结：JSON体积小、解析方便且高效，在实际开发成为首选。

## 兼容性

IE5、IE6中使用 ActiveObject("Microsoft.XMLHTTP")

如下图



关于IE的兼容方面，了解即可。

# 模板引擎

## 原理剖析

其本质是利用正则表达式，替换模板当中预先定义好的标签。

## 流行模板引擎

**ArtTemplate：**https://github.com/aui/artTemplate

**velocity.js：**https://github.com/shepherdwind/velocity.js/

**Handlebars：**http://handlebarsjs.com/

## artTemplate

引入template-web.js后可以使用模板引擎语法。

调用**template('字符串模板ID', 对象格式数据)**，会返回将数据添加至模板字符串的html字符串。

1、简洁语法

a) <% 与  %> 符号包裹起来的语句则为模板的逻辑表达式

b) <%= content %>为输出表达式

c) <% if(条件) { %>

<tr>

<td><%= 输出内容 %></td>

.....

<% } %>

d) <% for(var i=0; i<数组.length; i++) { %>

<tr>

<td><%= 输出内容 %></td>

</tr>

<% } %>

2、标准语法

a) {{输出内容}}

b) {{if(条件)}}

{{输出内容}}

{{/if}}

c) {{each 数组}}

<tr>

<td>{{$index}}{{$value}}</td>

</tr>

{{/each}}

c) {{each 数组 as val key}}

<tr>

<td>{{key }}{{val }}</td>

</tr>

{{/each}}

以上语法可满足基本需求，细节语法可[参照文档](https://aui.github.io/art-template/)。

# 同源&跨域

## 同源

同源策略是浏览器的一种安全策略，所谓同源是指，域名，协议，端口完全相同。

## 跨域

不同源则跨域

例如http://www.example.com/

|  |  |  |
| --- | --- | --- |
| http://api.example.com/detail.html | 不同源 | 域名不同 |
| https//www.example.com/detail.html | 不同源 | 协议不同 |
| http://www.example.com:8080/detail.html | 不同源 | 端口不同 |
| http://api.example.com:8080/detail.html | 不同源 | 域名、端口不同 |
| https://api.example.com/detail.html | 不同源 | 协议、域名不同 |
| https://www.example.com:8080/detail.html | 不同源 | 端口、协议不同 |
| http://www.example.com/detail/index.html | 同源 | 只是目录不同 |

## 跨域解决方案

1、document.domain + iframe

2、window.name + iframe

3、location.hash + iframe

4、window.postMessage()

**参考资料**

**http://rickgray.me/2015/09/03/solutions-to-cross-domain-in-browser.html**

## JSONP

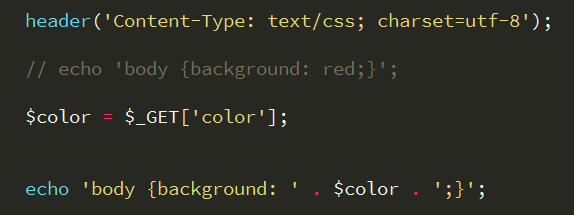
JSON with Padding，是一种解决跨域的方案。

**1、原理剖析**

其本质是利用了<script src=""></script>标签具有可跨域的特性，由服务端返回一个预先定义好的Javascript函数的调用，并且将服务器数据以该函数参数的形式传递过来，此方法需要前后端配合完成。

1. 文件后缀并不能代表什么，关建我们要看服务器返回的内容，比如css.php

这时返回的内容为 echo ‘body {background:color}’我们可以通过content-type:text/css来指定浏览器该如何解析返回的内容。



1. 同样我们也可以js文件以.php结尾，比如js.php 这时我们可以指定Content-Type: text/javascript可告知浏览器要以javascript来执行我们返回的内容，如果返回的是一个javascript的一个函数调用，那么在这个函数调用过程中可以将跨域请求来的数据以“实参”的传递过来，并且这个实参一般是JSON格式的。



# 封装AJAX工具函数

为了提升我们的开发效率，我们自已将XMLHttpRequest封装成一个函数，同时让同工具函数支持JSONP跨域。

# jQuery中的Ajax

## XMLHttpRequest封装

jQuery为我们提供了更强大的Ajax封装

1、$.ajax({}) 可配置方式，发起Ajax请求

常用配置参数：

a) url 请求地址

b) type 请求方式

c) timeout 请求超时

d) dataType 约束服务器返回数据格式

e) contentType 设置Content-Type请求头

f) data 发送请求数据

g) beforeSend: function () {} 请求发起前调用

h) success 成功响应后调用

i) error 错误响应时调用?

j) complete 响应完成时调用（包括成功和失败）

2、$.get() 以GET方式发起Ajax请求

3、$.post() 以POST方式发起Ajax请求

4、$.getJSON 约束服务器返回数据格式为json。

5、$.getScript约束服务器返回数据格式为javascript

6 、$('selector').load异步加载DOM

## 全局配置

1、ajaxSetup全局配置ajax方法

2、jQuery中提供了精细控制ajax请求及响应步骤的方法，具体操作是将整个请求和响应划分成6个阶段，当执行到某个阶段时即会调用一个回调函数。

a) ajaxStart 阶段

b) ajaxSend阶段

c) ajaxSuccess阶段

d) ajaxError阶段

e) ajaxStop阶段

f) ajaxComplete阶段

## JSONP封装

jQuery 的$.ajax() 方法当中集成了JSONP的实现，可以非常方便的实现跨域数据的访问。

使用步骤：

dataType: 'jsonp' 设置dataType值为jsonp即开启跨域访问

jsonp 可以指定服务端接收的参数的“key”值，默认为callback

jsonpCallback 可以指定相应的回调函数，默认自动生成

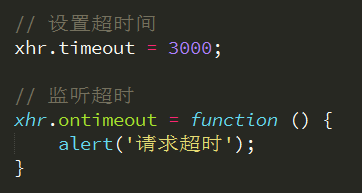
# XMLHttpRequest 2.0

技术总是在实践中不断更新的，XMLHttpRequest也不例外。

## 设置超时

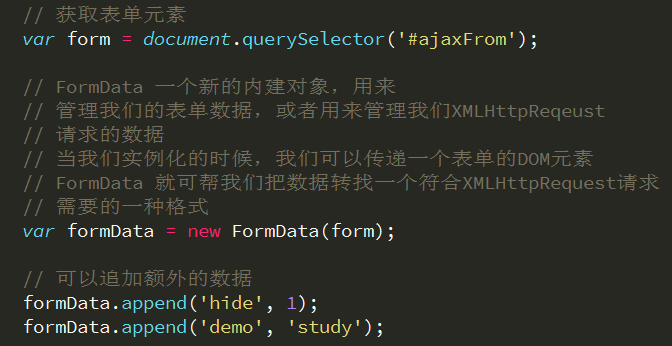
1. 设置超时 xhr.timeout
2. 监听超时事件 xhr.ontimeout = function () {// code}

当请求超时，此事件就会被触发



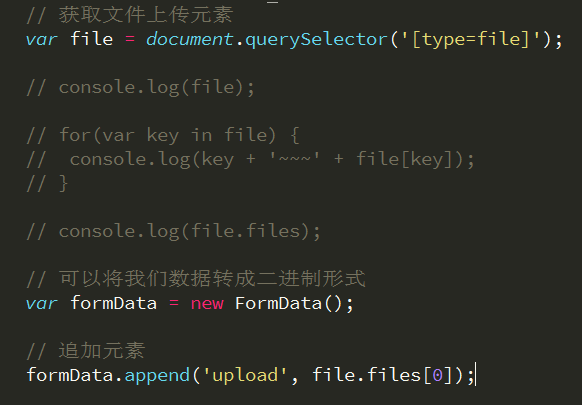
## FormData

1. 提供了一个新的内建对象，可用于管理表单数据
2. 首先要获取一个表单元素form
3. 然后在实例化时 new FormData(form)，将表单元素form传进去
4. 会返回一个对象，此对象可以直接做为xhr.send(formData)的参数
5. 此时我们的数据就是以二进制形式传递了
6. 注意我们这里只能以post形式传递，浏览器会自动为我们设置一个合适的请求头



## 二进制

1. 我们上传文件是以二进制形式传递的
2. 我们可以通过表单<input type=”file”>获取到一个文件对象
3. 然后file.files[0]可以获取文件信息
4. 然后再利用var formData = new FormData() 实例化
5. 然后再利用formData.append(‘upload’, file.files[0])将文件转成二进制
6. 最后将 formData 做为xhr.send(formData)的参数



## 上传进度

1. 利用XMLHttpRequest我们可以实现文件的上传
2. 并且我们可以通过xhr.upload.onprogress = function (ev) {// code}，监听上传的进度
3. 这时我们上传的进度信息会保存在事件对象ev里
4. ev.loaded 表示已上传的大小，ev.total表示文件整体的大小
5. var percent = ev.loaded / ev.total