<http://www.ibm.com/developerworks/cn/xml/wa-ajaxintro1.html>

Ajax 由 HTML、JavaScript™ 技术、DHTML 和 DOM 组成

编写应用程序时有两种基本的选择：

1、桌面应用程序（C/S架构）

2、web应用程序（B/S架构）

伴随着 Web 的强大而出现的是等待，等待服务器响应，等待屏幕刷新，等待请求返回和生成新的页面。

Ajax 尝试建立类桌面应用程序的功能和交互性，与不断更新的 Web 应用程序之间的桥梁。可以使用像桌面应用程序中常见的动态用户界面和漂亮的控件，不过是在 Web 应用程序中。

下面是 Ajax 应用程序所用到的基本技术：

1、HTML 用于建立 Web 表单并确定应用程序其他部分使用的字段。

2、JavaScript 代码是运行 Ajax 应用程序的核心代码，帮助改进与服务器应用程序的通信。

3、DHTML 或 Dynamic HTML，用于动态更新表单。我们将使用 div、span 和其他动态 HTML 元素来标记 HTML。

4、文档对象模型 DOM 用于（通过 JavaScript 代码）处理 HTML 结构和（某些情况下）服务器返回的 XML。

XMLHttpRequest对象,这是处理所有服务器通信的对象

创建：var xhr=new XMLHttpRequest();

通过 XMLHttpRequest 对象与服务器进行对话的是 JavaScript 技术。这不是一般的应用程序流，这恰恰是 Ajax 的强大功能的来源。

html表单《--》javascript 利用XMLHttpRequest《--》web服务器

#加入一些 JavaScript

得到 XMLHttpRequest 的句柄后，其他的 JavaScript 代码就非常简单了。事实上，我们将使用 JavaScript 代码完成非常基本的任务：

1. 获取表单数据：JavaScript 代码很容易从 HTML 表单中抽取数据并发送到服务器。

2． 修改表单上的数据：更新表单也很简单，从设置字段值到迅速替换图像。

3． 解析 HTML 和 XML：使用 JavaScript 代码操纵 DOM（请参阅 下一节），处理 HTML 表单服务器返回的 XML 数据的结构。

例:

//获得表单数据

var phone=doucument.getElementById("phone").value;

//通过服务器返回的数据修改表单

document.getElementById("order").value=response[0];

document.getElementById("address").value=response[1];

#获取 Request 对象

针对浏览器不同方法不同，IE又有两种版本

IE:

var xmlHttp=false;

try{

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

}catch(e){

try{

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

}catch(e2){

xmlHttp=false;

}

}

非IE浏览器

xmlHttp=new XMLHttpRequest();或者xmlHttp=new XMLHttpRequest object;

兼容代码：

var xmlHttp=false;

try{

xmlHttp=new XMLHttpRequest object;

}catch(e){

try{

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

}catch(e2){

xmlHttp=new ActiveObject("Microsoft.XMLHTTP")

}

}

}

#发出请求

接下来就是在所有 Ajax 应用程序中基本都雷同的流程：

1. 从 Web 表单中获取需要的数据。
2. 建立要连接的 URL。
3. 打开到服务器的连接。
4. 设置服务器在完成后要运行的函数(即回调函数)。
5. 发送请求。

//获取表单数

var city=document.getElementById("city").value;

var state=document.getElementById('state').value;

if(city==null || city=="") return;

if(state==null || state=="") return;

//建立链接的url

var url="/scripts/getZipCode.php?city="+escape(city)+"&state="+escape(state);

//打开链接到服务器

xmlHttp.open("GET",url,true);//设置为true为异步请求

//建立一个函数来处理服务器返回的数据

xmlHttp.onreadystatechange=statechange;

//发送请求

xmlHttp.send(null);//使用值 null 调用 send()。因为已经在请求 URL 中添加了要发送给服务器的数据（city 和 state），所以请求中不需要发送任何数据。这样就发出了请求，服务器按照您的要求工作。

onreadystatechange 属性可以告诉服务器在运行完成 后（可能要用五分钟或者五个小时）做什么

#处理响应

1、什么也不要做，找到xmlHttp.readyState属性的值为4

2、服务器将把响应填充到xmlHttp.responseText属性中

例：

function statechagne(){

if(xmlHttp.readyState==4 || xmlHttp.readyState=="complete"){

var response=xmlHttp.responseText;

document.getElementById("zipCode").value=response;

}

}

#链接web表单

启动一个ajax过程

<form>

city:<input type="text" name="" onchange="callserver()">

state:<input type="text" name="" onchange="callserver()">

zip code:<input type="text" name="" onchange="callserver">

</form>

当用户在 city 或 state 字段中输入新的值时，callServer() 方法就被触发，于是 Ajax 开始运行了。有点儿明白怎么回事了吧？好，就是如此！

第二部分:使用javascript和ajax发出异步请求

传统的web应用程序：请求\响应模型来从服务器得到html页面，这需要等待

有了ajax和XMLHttpRequest对象，就可以不用等待

Web 2.0消除了这种看得见的往复交互，例子如google maps

使得交互成为可能，需要发出请求和接受响应，但正是如此造成了web交互的缓慢，

笨拙的web交互感受；我们需要一种方法使发送的请求和接收的响应只包含需要的数据而不是整个 HTML 页面。惟一需要获得整个新 HTML 页面的时候就是希望用户看到新页面的时候。

奇迹关键:XMLHttpRequest对象，属于javascript

将要用到该对象的很少的属性和方法：

1、open()建立到服务器的新请求

2、send()向服务器发送请求

3、abort()退出当前请求

4、readyState属性，提供当前HTML的就绪状态

5、responseText属性，服务器返回的请求响应文本

创建具有错误处理能力的XMLHttpRequest

var request=false;

try{

response=new XMLHttpRequest();

}catch(e){

request=false;

}

if(!request){

alert("初始化XMLHttpRequest错误");

}

增加对IE的支持

var request=false;

try{

response=new XMLHttpRequest();

}catch(tryIE){

try{

request=new ActiveObject("Msxml2.XMLHTTP");

}catch(otherIE){

try{

request=new ActiveObject("Microsoft.XMLHTTP");

}catch(failed){

request=false;

}

}

}

if(!request){

alert("初始化XMLHttpRequest错误");

}

将 XMLHttpRequest 创建代码移动到方法中

functon getXMLHttpRequest(){

var request=false;

try{

request=new XMLHttpRequest();

}catch(tryIE){

try{

request=new ActiveObject("Msxml2.XMLHTTP");

}catch(otherIE){

try{

request=new ActiveObject("Microsoft.XMLHTTP");

}catch(failed){

request=false;

}

}

}

if(!request){

alert("初始化XMLHttpRequest错误");

return;

}

return request;

}

建议编写静态的代码，让用户尽可能早地发现问题。

XMLHttpRequest 惟一的目的是让您发送请求和接收响应。其他一切都是 JavaScript、CSS或页面中其他代码的工作：改变用户界面、切换图像、解释服务器返回的数据

设置服务器URL：多数应用程序都会结合一些静态数据和用户处理的表单中的数据来钩子啊URL

如：

var phone=document.getElementById("phone").value;

var url="/zipcode/look.php?phone="+escape(phone)+"&sid="+Math.random();

//注意：Ajax 代码是沙箱型的，因而只能连接到同一个域，实际上 URL 中不需要域名。

escape()方法用于转义不能用明文正确发送的任何字符，比如，电话号码中的空格将被转换成字符 %20，从而能够在 URL 中传递这些字符。

可以根据需要加入任意多个参数，使用&

#打开请求

request.open("GET",url,true);

XMLHttpRequest对象的open()方法有5个参数，分别为：

1、request-type:发送请求的类型，但也可以发送head请求

2、url：要连接的url

3、asynch:如果希望使用异步链接则为true，否则为false,参数可选默认为true

4、username:如果需要身份验证，则可以在此指定用户名，可选参数没有默认值

5、password：如果需要身份验证，则可以在此指定口令，可选参数没有默认值

而异步请求不 等待服务器响应。发送请求后应用程序继续运行。用户仍然可以在 Web 表单中输入数据，甚至离开表单。没有旋转的皮球或者沙漏，应用程序也没有明显的冻结。服务器悄悄地响应请求，完成后告诉原来的请求者工作已经结束（具体的办法很快就会看到）。结果是，应用程序感觉不那么迟钝或者缓慢，而是响应迅速、交互性强，感觉快多了。这仅仅是 Web 2.0 的一部分，但它是很重要的一部分。所有老套的 GUI 组件和 Web 设计范型都不能克服缓慢、同步的请求/响应模型。

#发送请求

send()只有一个参数，就是要发送的内容；但一般（80%情况）在URL阶段已经将

数据存储在URL里了，因而一般不需要使用send发送数据，如果

需要发送安全信息或XML，可以使用send()方法，如果不需要通过send传递数据，只需要

传递null作为该方法的参数即可，即request.send(null);

#指定回调方法

属性：onreadystatechange

指定一个回调函数，允许服务器反向调用web页面中的代码，他给了服务器一定的控制权

request.onreadystatechange=updatePage;

该属性要在send()之前设置，且必需

#回调和Ajax

将onreadystatechange属性设置为要运行的函数名，当服务器处理完请求后就会自动调用

该函数，也不需要担心该函数的任何参数

如：

function updatePage(){

alert("服务器处理完成");

}

#HTTP就绪状态 readyState

五种就绪状态

值 描述

0 请求没有发出-在调用open之前

1 请求已经建立但还没有发出-在调用send之前

2 请求已经发出正在处理之中-这里通常可以从响应得到内容头部

3 求已经处理，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 响应已完成，可以访问服务器响应并使用它

基于此，回调方法中必须：

function updatePage(){

if(request.readyState==4 || request.readyState=="complete"){

//执行代码

}

}

#HTTP状态 404 401 403等

服务器履行了请求（就绪状态是4）但是没有返回客户拒预期的数据，

因此除了就绪状态外，还需要检查HTTP状态，我们期望的状态码是200，

他表示一切顺利，如果就绪状态是4，而且状态码是200就可以处理服务器

的数据了，而且这些数据应该就是要求的数据（而不是错误或者其他有问题的信息）。因此还要在回调方法中增加状态检查，

检查HTTP状态码：

function updatePage(){

if(request.readyState==4 || request.readyState=="complete"){

if(request.status==200){

//处理代码

}

}

}

为了增加更健壮的错误处理并尽量避免过于复杂，可以增加一两个状态码检查;

function updatePage(){

if(request.readyState==4 || request.readyState=="complete"){

if(request.status==200){

//执行代码

}else if(request.status==404){

alert("url不存在");

}else{

alert("错误码为:"+request.status);

}

}

}

#读取响应文本

返回的数据保存在XMLHttpRequest对象的responseText属性中

function updatePage() {

if (request.readyState == 4) {

if (request.status == 200) {

var response = request.responseText.split("|");

document.getElementById("order").value = response[0];

document.getElementById("address").innerHTML =

response[1].replace(/\n/g, "

");

} else

alert("status is " + request.status);

}

}

还要介绍 XMLHttpRequest 的另一个重要属性 responseXML。如果服务器选择使用 XML 响应则该属性包含（也许您已经猜到）XML 响应。处理 XML 响应和处理普通文本有很大不同，涉及到解析、文档对象模型（DOM）和其他一些问题。

###第三部分 Ajax中的高级请求和响应

全面理解 HTTP 的状态代码、就绪状态和 XMLHttpRequest 对象

HTTP就绪状态 readyState

五种就绪状态

值 描述

0 请求没有发出-在调用open之前

1 请求已经建立但还没有发出-在调用send之前

2 请求已经发出正在处理之中-这里通常可以从响应得到内容头部

3 求已经处理，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 响应已完成，可以访问服务器响应并使用它

隐秘就绪状态:

readyState==0:表示未初始化状态，一旦对请求对象调用open()方法后这个属性就被设置为1；

在大部分 Ajax 编程的真实情况中，这种就绪状态的唯一用法就是使用相同的 XMLHttpRequest 对象在多个函数之间生成多个请求。在这种（不常见的）情况中，您可能会在生成新请求之前希望确保请求对象是处于未初始化状态（readyState == 0）。这实际上是要确保另外一个函数没有同时使用这个对象。

如果您必须使用多个函数，最好是为每个函数都创建并使用一个函数，而不是在多个函数之间共享相同的对象。

function(){

alert(xhr.readyState);//每次就绪状态放生变坏时，都会触发执行该函数，因此输出0,1,2,3,4（不同的浏览器可能有某些值不存在）

}

这段代码就是onreadystatechange意义的一个确切展示

#每次请求的就绪状态发生变化时#就调用 updatePage()，然后我们就可以看到一个警告了。

#不同的浏览器可能会不包含某些就绪码，但是他们的含义都相同

#查看请求的响应文本

与就绪状态类似，responseText属性的值在整个请求的声明周期内也会发生变化，可以使用如下代码查看：

function statechange(){

alert(xhr.responseText);

//其他代码

}

#深入了解HTTP状态代码

401：未经授权

403：禁止

404：未找到

200：一切正常

如果脚本需要认证，而请求却没有提供有效的证书，那么服务器就会返回诸如 403 或 401 之类的错误代码。然而，由于服务器对请求进行了应答，因此就绪状态就被设置为 4（即使应答并不是请求所期望的也是如此）。最终，用户没有获得有效数据，当 JavaScript 试图使用不存在的服务器数据时就可能会出现严重的错误。

#重定向和重新路由，在 HTTP 状态代码中，这是 300 系列的状态代码

301：永久移动

302：找到，请求被重新定向到另外一个 URL/URI 上

305：使用代理（请求必须使用一个代理来访问所请求的资源）

结果是您的请求无法重定向到其他服务器上，而不会产生安全性错误。在这些情况中，您根本就不会得到状态代码。通常在调试控制台中都会产生一个 JavaScript 错误。因此，在对状态代码进行充分的考虑之后，您就可以完全忽略重定向代码的问题了。

检查有效状态码

function updatePage(){

if(xhr.readyState==4 || xhr.readyState=="complete"){

if(xhr.status==200){

//正常代码

}else if(xhr.status==404){

alert("文件为未找到");

}esle if(xhr.status==403){

alert("请求被禁止");

}else{

alert("REQUEST ERROR STATUS:"+xhr.status);

}

}

}

#其他请求类型

生成head请求，其实很简单，xhr.open("HEAD",url,true);

服务器并不会像对 GET 或 POST 请求一样返回一个真正的响应。相反，服务器只会返回资源的 头（header），这包括响应中内容最后修改的时间、请求资源是否存在和很多其他有用信息。您可以在服务器处理并返回资源之前使用这些信息来了解有关资源的信息。

输出从head请求中获得的响应头内容

function statechange(){

if(xhr.readyState==4 || xhr.readyState="complete"){

alert(xhr.getAllResponseHeaders());

}

}

函数getAllResponseHeaders()可以获得响应头,无论对于get、post还是head请求方法

#有用的 HEAD 请求

您会发现 HEAD 请求非常有用的一个领域是用来查看内容的长度或内容的类型。这样可以确定是否需要发回大量数据来处理请求，和服务器是否试图返回二进制数据，而不是 HTML、文本或 XML（在 JavaScript 中，这 3 种类型的数据都比二进制数据更容易处理）。

在这些情况中，您只使用了适当的头名，并将其传递给 XMLHttpRequest 对象的 getResponseHeader() 方法。因此要获取响应的长度，只需要调用 request.getResponseHeader("Content-Length");。要获取内容类型，请使用 request.getResponseHeader("Content-Type");。

在很多应用程序中，生成 HEAD 请求并没有增加任何功能，甚至可能会导致请求速度变慢（通过强制生成一个 HEAD 请求来获取有关响应的数据，然后在使用一个 GET 或 POST 请求来真正获取响应）。然而，在出现您不确定有关脚本或服务器端组件的情况时，使用 HEAD 请求可以获取一些基本的数据，而不需要对响应数据真正进行处理，也不需要大量的带宽来发送响应。

#第四部分 利用DOM进行WEB响应

WEB程序员做什么？控制页面的标记结构

标记就绪以后才能对其进行操作或设计样式。

标记只为您的页面提供组织、框架

浏览器将这些文本组织转化为对象，即文档对象

文本标记的优点：

1、速度快

2、传输容易

3、统一格式

4、开发容易，标记和样式分离

文本标记的缺点：

1、浏览器很难将文本标记显示为图像等

2、

浏览器接受页面工作原理：

浏览器接受该页面并将之转换为树形结构

每一个标记都可以看小矩形，这是一个对象。通过使用对象来表示HTML文档的一部分

可以很容易的更改、应用样式、允许javascript访问文档

DOM对象类型和属性:如HTML中的元素用Element对象类型来表示，

文档中的文本用text类型来表示，属性用Attribute类型表示等

DOM:标记中的任何内容都必须转换为某种类型的对象

浏览器将页面转换为对象来表示，对象的表示是DOM树，它是一种规范

DOM定义了对象的属性和类型，从而允许浏览器解释标记

标记的每一部分都由一个对象表示，但他不是换一个任意的对象，它是特定了类型

的对象，一个DOM节点

基本节点类型：元素节点、文本节点和属性节点

#缺少5

#第六部分 建立基于DOM的web应用程序

在不刷新页面的情况下使用DOM改变页网页页面

DOM可以移动网页中的任何东西而不需要提交表单，因此足以和Ajax媲美

DOM中最关键的时刻document对象，它代表整个网页，在DOM中一切都是节点

#第七部分 在请求和响应中使用XML

了解什么时候适合设么时候不适合使用xml

XMLHttpRequest只不过是为客户机代码发送HTTP请求的一种方式，与xml没有太大的实质性关系，当然它可以发送和接受xml数据；XMLHttpRequest对象更多的是关于在不重新爱在页面的情况下发出请求，而不会太多的涉及XML甚至HTTP

无论使用 Ajax 还是普通的表单 POST，甚至超链接，打交道的都是 HTTP。

虽然 XML 可以通过 HTTP 发送，但 XML 是一种数据格式而不是传输协议。

#使用XML

1、以XML格式从网页向服务器发送请求

2、以XML格式在网页中从服务器接受请求

XML 不是一种简洁、快速和节省空间的格式，但是，和普通文本相比，XML 通常总会占用更多的空间，速度也更慢，因为需要在消息中增加 XML 所需要的标签和语义

#从客户机到服务器的XML

发送名/值对：

firstname=larry;

lastname=gullahorn;

street=9018 heatherhorn drive;

city=texas;

zipCodde=75080;

传统的方法：使用普通文本发送名/值对：构造URL

url="/scripts/saveaddress.php?firstname="+escape(firstname)+"&lastname="+escape(lastname)+"&street="+escape(street)+"&city="+escape(city)+"&state="+escape(state)+"&zipcode="+escape(zipcode);

xhr.open("GET",url,true);

xhr.onreadystatechange=statechange;

xhr.send(null);

采用XML格式：

1、将数据转换为XML

<address>

<firstname>larry</firstname>

<lastname>gullahorn</lastname>

<street>9018 heatherhorn drive</street>

<city>texas</city>

<zipCodde>=75080</zipCodde>

</address>

2、确定服务器是否能接受该格式的xml

1、保证向其发送xml的脚本能够接受xml数据格式

2、保证脚本认可发送数据所采用的特定xml格式和结构

3、向服务器发送XML

var firstname=larry;

var lastname=gullahorn;

var street=9018 heatherhorn drive;

var city=texas;

var zipCodde=75080;

var xmlString="<address>"+

"<firstname>"+escape(firstname)+"</firstname>"+

"<lastname>"+escape(lastname)+"</lastname>"+

"<street>"+escape(state)+"</street>"+

"<city>"+escape(city)+"</city>"+

"<zipCodde>"+escape(zipCode)+"</zipCodde>"+

"</address>";

url="/scripts/saveaddress.php";

xhr.open("POST",url,true);

xhr.setRequestHeader("Content-Type","text/xml");//告诉服务器要发送的是XML而不是一般的名值对

xhr.onreadystatechange=statechange;

xhr.send(smlString);

1、这里的另一个问题前面已经提到，即必须手工创建 XML。使用 DOM 不是一种好的选择，因为没有简单易行的办法将 DOM 对象转化成在请求中发送的字符串。

2、除了复杂性的问题之外，和普通文本以及名/值对相比，在请求中使用 XML 实际上没有多少好处（如果有的话）。要注意，本文坚持使用前面用名/值对发送的同一些数据（请参阅清单 1）来用 XML 发送。我没有提什么数据能用 XML 但是不能用普通文本发送，这是因为实际上没有任何东西可用 XML 而不能用普通文本发送。

因此，除非特殊情况，在请求中最好使用普通文本；

#第八部分 在请求和响应中使用XML

这篇文章主要探讨在大多数情况下确实是 好主意的一种做法：

向客户机返回 XML 响应。

需要向服务器发送 XML 的所有理由都差不多可以归入以下两个基本的类别中：

1、服务器仅 接受 XML 请求。在这类情况下，您别无选择。上一期文章中介绍的基础知识应已使您掌握了发送此类请求所必需的工具。

2、您正在调用一个仅接受 XML 或 SOAP 请求的远程 API。这实际上就是上一种情况的特例，但值得单独拿出来提一下。如果您希望在一个异步请求中使用来自 Google 或 Amazon 的 API，就会有一些特殊的考虑事项。在下一期的文章中，我将介绍这些考虑事项，还会给出一些向 API 发送此类请求的示例

从服务器接受XML

两种方法处理一个来自服务器的XML响应：

1、作为碰巧格式化的xml纯文本

2、作为一个xml文档，由一个DOM Document对象表示

#将xml当成xml

使用responseXML属性

function statechange(){

if(xhr.readyState==4 || xhr.readyState=="complete){

if(xhr.status==200){

var xmlDoc=xhr.responseXML;

//其他代码

}

}

}

进一步处理XML数据（记住xml也是一种DOM）：

获取所有的show元素：

var showElements=xmlDoc.getElementsByTagName("show");

遍历xml数据：

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

var title=showElements[i].childNodes[0].value;

var rating=showElements[i].childNodes[1].value;

//处理title和rating的代码

}

返回xml的PHP脚本

<?php

// Connect to a MySQL database

$conn = @mysql\_connect("mysql.myhost.com", "username", "secret-password");

if (!conn)

die("Error connecting to database: " . mysql\_error());

if (!mysql\_select\_db("television", $conn))

die("Error selecting TV database: " . mysql\_error());

// Get ratings for all TV shows in database

$select = 'SELECT title, rating';

$from = ' FROM ratings';

$queryResult = @mysql\_query($select . $from);

if (!$queryResult)

die("Error retrieving ratings for TV shows.');

// Let the client know we're sending back XML

header("Content-Type: text/xml");

echo "<?xml version=\"1.0\" encoding=\"utf-8\"?>";

echo "<ratings>";

while ($row = mysql\_fetch\_array($queryResult)) {

$title = $row['title'];

$rating = $row['rating'];

echo "<show>

echo "<title>" . $title . "</title>";

echo "<rating>" . $rating . "</rating>";

echo "</show>";

}

echo "</ratings>";

mysql\_close($conn);

?>

#解释xml的其他可选方法

值得一提。那就是 JSON

大体上，可以用 JSON 做的事，用 DOM 都可以完成，反之亦然

###第九部分 使用google ajax search api

在异步应用程序中使用公共 API

例如，通过一个用于 Google 搜索引擎的公共 API 可以发出搜索请求，但是实际上是由 Google 的代码搜索 Google 的数据，然后将结果返回给您的程序。

##使用Google Ajax Search API的准备工作

#从Google获取开发者秘钥（本文着重讨论Google的Ajax Search API）

###使用JSON进行数据传输

以原生方式操作JavaScript对象

在大多数异步应用程序场合，总是应该首先考虑使用名称/值对，除非有某种限制迫使您转向XML，否则用不着考虑使用别的数据格式

1、如果服务器对接受的数据有限制，那么就要使用xml格式；

2、在大多数情况下，当需要向应用程序/客户端发送多个名称/值对时，就应该考虑使用XML格式；

3、XML格式通常更适合用来向客户端浏览器做出响应，而不是从Ajax应用程序发出请求

JSON示例：

{"firstname":"brett","lastname":"mclaughlin","email":"xxx@xx.com"}

##值和数组

{ "people": [

{ "firstName": "Brett", "lastName":"McLaughlin", "email": "brett@newInstance.com" },

{ "firstName": "Jason", "lastName":"Hunter", "email": "jason@servlets.com" },

{ "firstName": "Elliotte", "lastName":"Harold", "email": "elharo@macfaq.com" }

]}

右如：

{ "programmers": [

{ "firstName": "Brett", "lastName":"McLaughlin", "email": "brett@newInstance.com" },

{ "firstName": "Jason", "lastName":"Hunter", "email": "jason@servlets.com" },

{ "firstName": "Elliotte", "lastName":"Harold", "email": "elharo@macfaq.com" }

],

"authors": [

{ "firstName": "Isaac", "lastName": "Asimov", "genre": "science fiction" },

{ "firstName": "Tad", "lastName": "Williams", "genre": "fantasy" },

{ "firstName": "Frank", "lastName": "Peretti", "genre": "christian fiction" }

],

"musicians": [

{ "firstName": "Eric", "lastName": "Clapton", "instrument": "guitar" },

{ "firstName": "Sergei", "lastName": "Rachmaninoff", "instrument": "piano" }

]

}

##在JavaScript中使用JSON

json是javascrpt的原生格式，因此js操作json不需要任何特殊的API和工具包

var people=

{ "programmers": [

{ "firstName": "Brett", "lastName":"McLaughlin", "email": "brett@newInstance.com" },

{ "firstName": "Jason", "lastName":"Hunter", "email": "jason@servlets.com" },

{ "firstName": "Elliotte", "lastName":"Harold", "email": "elharo@macfaq.com" }

],

"authors": [

{ "firstName": "Isaac", "lastName": "Asimov", "genre": "science fiction" },

{ "firstName": "Tad", "lastName": "Williams", "genre": "fantasy" },

{ "firstName": "Frank", "lastName": "Peretti", "genre": "christian fiction" }

],

"musicians": [

{ "firstName": "Eric", "lastName": "Clapton", "instrument": "guitar" },

{ "firstName": "Sergei", "lastName": "Rachmaninoff", "instrument": "piano" }

]

}

访问数据：

people.programmers[0].lastname;

修改数据：

people.programmers[0].lastname="haha";

转换回字符串：

String new newJSONtext=people.toJSONString();

##重要的是：可以将任何JavaScript对象转换为JSON文本，如：

String newObjectInJSON=myobject.toJSONString();

###第十一部分 服务器端的JSON

在服务器端脚本和程序中用JSON进行响应和回复

##将JSON发给服务器

#通过GET以名称/值对发送JSON

\*将JSON数据发给服务器的最简单方法是将其转换为文本，然后以名称/值对的形式

进行发送

var people = { "programmers": [ { "firstName": "Brett", "lastName":"McLaughlin",

"email": "brett@newInstance.com" }, { "firstName": "Jason", "lastName":"Hunter",

"email": "jason@servlets.com" }, { "firstName": "Elliotte", "lastName":"Harold",

"email": "elharo@macfaq.com" } ], "authors": [ { "firstName": "Isaac",

"lastName": "Asimov", "genre": "science fiction" }, { "firstName": "Tad",

"lastName": "Williams", "genre": "fantasy" }, { "firstName": "Frank",

"lastName": "Peretti", "genre": "christian fiction" } ], "musicians": [

{ "firstName": "Eric", "lastName": "Clapton", "instrument": "guitar" },

{ "firstName": "Sergei", "lastName": "Rachmaninoff", "instrument": "piano" } ] }

var url="organization.php?people="+people.toJSONString():

xhr.open("GET",url,true);

xhr.onreadystatechange=statechange;

xhr.send(null);

但却存在一个问题：在 JSON 数据中会有空格和各种字符，Web 浏览器往往要尝试对其继续编译。要确保这些字符不会在服务器上（或者在将数据发送给服务器的过程中）引起混乱，需要在 JavaScript escape() 函数中做如下添加：

url="organization.php?people="+escape(people.toJSONString());

该函数(escape可以处理空格、斜线和其他任何可能影响浏览器的内容，并将它们转换成 Web 可用字符（比如，空格会被转换成 %20，浏览器并不会将其视为空格处理，而是不做更改，将其直接传递到服务器）。之后，服务器会（通常自动）再把它们转换回它们传输后的本来 “面目”。

这种发送的缺陷（也是GET的缺陷）：

1、数据量大

2、安全性

##使用POST请求发送JSON数据

var url = "organizePeople.php?timeStamp=" + new Date().getTime();

request.open("POST", url, true);

request.onreadystatechange = updatePage;

request.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

request.send(people.toJSONString())

Content-Type 头被设置为让服务器预知它能得到何种数据。在这种情况下，即为 application/x-www-form-urlencoded，它让服务器知道现在发送的是文本，正如它从常规的 HTML 表单中得到的一样。

另一个简单提示是 URL 的末尾追加了时间。这就确保了请求不会在它第一次被发送后即缓存，而是会在此方法每次被调用后重新创建和重发；此 URL 会由于时间戳的不同而稍微有些不同。这种技巧常被用于确保到脚本的 POST 每次都会实际生成新请求且 Web 服务器不会尝试缓存来自服务器的响应。

\*JSON只是文本

##在服务器上解释JSON

处理JSON的两步骤：

1、针对服务端程序语言找到JSON解析器/工具箱/帮助其API

2、使用JSOn解析器等取得来自客户端的请求数据，并将数据转换为脚本所能理解的东西

#寻找JSON解析器

#使用JSON解析器