JavaScript及web前端开发

# 第10讲 ajax及node.js

唐大仕

# 相关网站

- ajax
  - https://w3school.com.cn/ajax/index.asp
  - https://w3school.com.cn/jquery/jquery_ajax_intro.asp
  - https://github.com/jakerella/jquery-mockjax
- node.js
  - 中文 http://nodejs.cn
  - 官网 https://nodejs.org
  - 参考 http://nodeapi.ucdok.com

# 课程进度

| | | |
|---|---|---|
| 11 综合应用 | | 10 ajax及node |
| 9 Bootstrap | | 8 Jquery |
| 5 HTML5 | 6 CSS3 | 7 JavaScript进阶 |
| 2 HTML | 3 CSS | 4 JavaScript基础 |
| 1 序论 | | |

# 本讲内容

- 前后端的数据交互
- Ajax
- 服务端及node.js

# 如何运行本节的示例

- 首先，网页要用http方式浏览
  - 在vscode中，可以这样：
    - 安装插件live server
    - 用vscode打开示例文件夹（打开文件夹是将它作为服务的“根目录"）
    - 在vscode中打开.html文件，右击，选Open with live server
  - 或者，起一个静态文件http服务（针对熟悉node的同学，可以使用这种方法）
    - 安装nodejs （到 nodejs.org下载）
    - 安装http-server （用命令npm install –g http-server）
    - 运行http-server –p 8080 这样就可以用http://localhost:8080/20-xxxx.html
- 其次，要运行ajax所能调用的服务（如果只要mock，则可以不运行这些服务）
  - 以Python Flask为例（针对熟悉python的同学）
    - 安装python
    - 装python里用的库，pip install flask  flask-cors
    - 运行我们写的flaskHello程序，用命令 python flaskHello.py
      - 其中ajax指向 http://localhost:5000/sqrt?num=56
  - 或者，以nodejs Express为例（针对熟悉nodejs的同学）
    - 安装nodejs （到 nodejs.org下载）
    - 安装express （用命令npm install express）
    - 运行我们写的expressHello程序，用命令node expressHello.js

# 前后端的通信

# HTTP及前后端的通信

- HTTP、HTTPS
- 在chrome中 F12, network

# form回顾

- form的写法
- action: 要提交的网址
- method: get或post
  - 差别在于：变量是否要显示在地址栏中
- input： 要提交的变量
  - 其name就是变量名，而id只是用于在客户端
- submit：提交按钮

# Ajax

# 什么是AJAX

- Ajax(Asynchronous JavaScript and Xml)
  - 几种技术，每种技术都有其独特用处，合在一起就成了一个功能强大的新技术。
- Ajax技术包括：
  - asynchronous:　使用XMLHttpRequest对象与Web服务器进行异步数据交换
  - JavaScript:　使用JavaScript操作DOM, 进行动态显示及交互
  - xml:　使用XML进行数据交换（现在多改用json了）
    - xml: <person><age>18</age><name>li ming</name></person>
    - json: { "age": 18,  "name":"li ming"}

# 在chrome中查看ajax请求

- F12， network中查看
- 如果只看ajax，可以选 XHR (xml http request)

# 应用示例

- 北大主页
  - 其新闻内容的呈现
  - homeJson.js中有：
    - $.ajax("dat/recentList.xml")
    - $.ajax("dat/topNews.json")

# 使用ajax有什么好处

- 局部刷新
  - 实时获取数据
- 异步操作
  - 更好的用户体验
- 前后端任务分离
  - 更清楚的逻辑

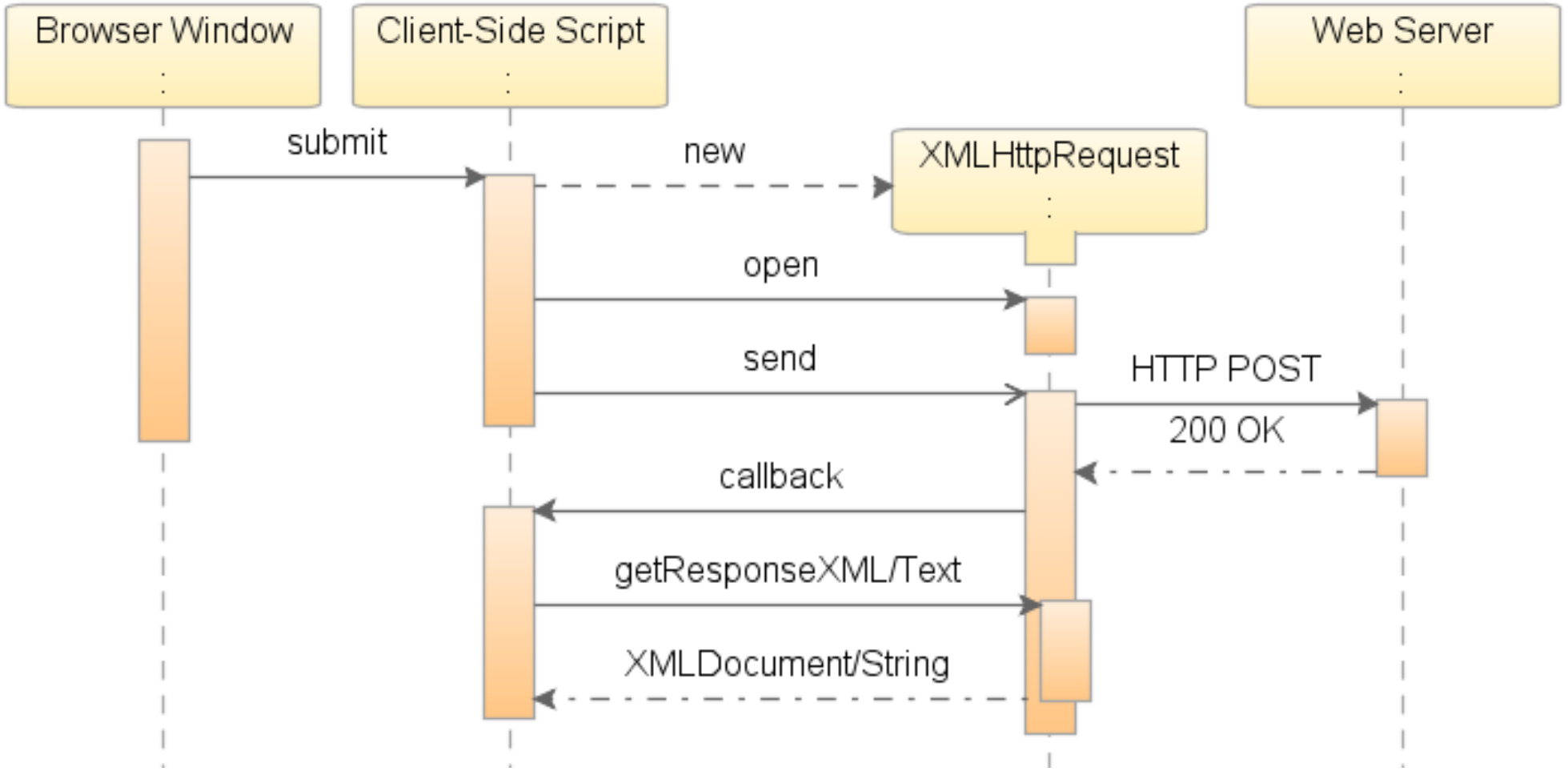

- 现在ajax已成为与服务端交互的主流技术

# 简单的ajax (了解)

- 获取参数、ajax取得数据、更新界面
  - var xmlHttp = new XMLHttpRequest();
  - xmlHttp.open("GET", url, false);  //这里false表示“同步”
  - xmlHttp.send(null);
  - var result = xmlHttp.responseText;

  - document.getElementById("txtHint").innerHTML= result;

# XMLHttpRequest请求流程



# 更一般地

- 一般使用异步调用
  - AJAX发起请求和接收响应的一般过程
    - 创建新的 XMLHttpRequest 对象
    - open()方法建立连接
    - onreadystatechange事件设置回调函数
      - responseText或responseXML属性接收响应
      - 在回调函数中进行界面的更新
    - send()方法发送请求
- 示例
  - ajax-10.htm
- XMLHttpRequest详细介绍
  - https://www.w3school.com.cn/xmldom/dom_http.asp

# XMLHttpRequest 简介(了解)

- XMLHttpRequest对象的属性和事件

| 属性/事件 | 说明 |
| --- | --- |
| ReadyState属性 | XMLHttpRequest对象远程通讯状态标志 |
| onreadystatechange事件 | readyState值发生改变，XMLHttpRequest对象都会激发一个readystatechange事件 |
| responseText属性 | 包含客户端接收到的HTTP响应的文本内容 |
| responseXML属性 | 此属性用于当接收到完整的HTTP响应时(readyState为4)描述XML响应；此时，Content-Type头部指定MIME(媒体)类型为text/xml，application/xml或以+xml结尾 |
| status属性 | 这个status属性描述了HTTP状态代码，而且其类型为short |
| statusText属性 | 这个statusText属性描述了HTTP状态代码文本 |

# XMLHttpRequest 简介(了解)

- ReadyState属性详解

| ReadyState属性值 | 说明 |
| --- | --- |
| 0 | 描述一种"未初始化"状态；此时，已经创建一个XMLHttpRequest对象，但是还没有初始化 |
| 1 | 描述一种"发送"状态；此时，代码已经调用了XMLHttpRequest open()方法并且XMLHttpRequest已经准备好把一个请求发送到服务器 |
| 2 | 描述一种"发送"状态；此时，已经通过send()方法把一个请求发送到服务器端，但是还没有收到一个响应 |
| 3 | 描述一种"正在接收"状态；此时，已经接收到HTTP响应头部信息，但是消息体部分还没有完全接收结束 |
| 4 | 描述一种"已加载"状态；此时，响应已经被完全接收 |

# XMLHttpRequest 简介(了解)

- XMLHttpRequest对象的方法

| 方法 | 说明 |
| --- | --- |
| abort()方法 | 使用这个abort()方法来暂停与一XMLHttpRequest对象相联系的HTTP请求，从而把该对象复位到未初始化状态 |
| open()方法 | 调用open(DOMString method，DOMString uri，boolean async，DOMString username，DOMString password)方法初始化一个XMLHttpRequest对象 |
| send()方法 | 调用open()方法准备好一个请求之后，你需要把该请求发送到服务器 |
| setRequestHeader()方法 | 该setRequestHeader(DOMString header，DOMString value)方法用来设置请求的头部信息 |
| getResponseHeader()方法 | getResponseHeader(DOMString header，value)方法用于检索响应的头部值 |
| getAllResponseHeaders()方法 | 该getAllResponseHeaders()方法以一个字符串形式返回所有的响应头部（每一个头部占单独的一行） |

# jQuery.ajax

# jQuery Ajax应用

- 详细用法 （掌握）
  - $.ajax( )
- 简单用法（了解）
  - $.get( )
  - $.post( )
  - $.getJSON( )
  - $(….).load( )

# $.ajax( )　（重点掌握）

- $.ajax(options)
  - options (选项) : AJAX 请求设置，是key/value方式的json对象
  - 常见选项设置如下:
    - type --- 请求方式,默认为GET
    - url --- 请求资源的URL
    - data --- 请求参数信息的json对象（key/value）
    - success --- 回应成功的回调函数
    - error --- 回应失败的回调函数

# $.ajax( )示例

- 示例

```
$.ajax(
    {
        type:'POST',
        url:'demo.jsp',
        data:{qq:'zhangsan'},
        success:function(msg){
            $('#result').empty().append(msg);
        },
        error:function(xhr){alert(xhr.statusText);}
    }
);
```

# $.ajax( )示例

```
$.ajax({ url:  “action.action",
        data: "name=John&location=Boston",
        dataType:  “json" ,
        success: function(data){
                //成功后执行
        }
});
```

# $.get( )

- $.get(url,[data],[callback])
  - url 请求资源的URL
  - data 请求参数信息 key/value
  - callback 载入成功时回调函数

```
$.get('demo.jsp',{qq:'wangwu'},function(msg){
    $('#result').empty().append(msg);
});
```

# $.post( )

- $.post(url,[data],[callback])
  - url 请求资源的URL
  - data 请求参数信息 key/value
  - callback 发送成功时回调函数

```
$.post('demo.jsp',{qq:'wangwu'},function(msg){
    $('#result').empty().append(msg);
});
```

# 返回数据的格式

- 服务器返回数据常用的格式
  - 字符串
    - 普通字符串（如一个数值及一段文字）
    - 带一定格式（如用逗号分开的多个单词）
    - 带html标记的数据（可直接显示到DOM中）
  - XML
    - <books> <book title="C#程序设计" author="唐大仕"></book> </books>
  - json
    - [ {title: "C#程序设计", author:"唐大仕"}, {.....} ]

# 返回数据的处理

- 服务器返回数据常用的处理方式
  - 带html标记的数据——用innerHTML处理
    - 或者直接用 $(选择器).load( url, data );
  - 格式化的数据——用split()函数处理
  - XML——用XML-DOM方法处理（现在较少用）
    - 或者用$(xmldata).find("...")
  - json——用 JSON.parse(...) 解析成json对象
    - 或者直接用 $.getJSON(url, data, function(json){.....} )

# $.ajax中处理json

- 一是当做文本，用JSON.parse（）
  - success: function(data){　var obj = JSON.parse(data);　}
  - 新的jquery中一般已经解析好了成为json对象了（因为它已经知道格式为Content-Type: application/json），就不需要用JSON.parse()
- 一是指定dataType
  - dataType:'json',
  - success: function(data){ data已解析为json对象了 }
- 更简单地，使用
  - $.getJSON( url, data, function(json){... } );

# 案例分析

- 网站的“加载更多”分页
  - https://help.geogebra.org/
  - 搜索
    - $('.filter-search'). $('.filter-search').on('change'
  - 翻译
    - $.ajax({type: "POST", url: "https://accounts.geogebra.org/api/translate.php",
  - 加载更多
  - 提交信息

# 直接用fetch (了解)

- 现代浏览器中，可以不用jQuery，直接用 fetch()函数
  - fetch()函数得到的一是Promise对象，它可以用then()方法进一步处理
  - 参考 https://developer.mozilla.org/zh-CN/docs/Web/API/Fetch_API/Using_Fetch
  - 在访问www.pku.edu.cn 时，可以在F12的控制台运行以下代码

```
fetch('https://www.pku.edu.cn/dat/topNews.json')
    .then(function(response) {
        console.log("得到响应", response)
        return response.json();
    })
    .then(function(myJson) {
        console.log("得到json对象", myJson);
    });
```

# 异步的三种实现方式（了解）

- 回调函数（callback ）
  - xmlHttp.onreadystatechange=function(){....}
  - xmlHttp.send(null);
- 使用Promise.then
  - fetch(url)　//fetch的结果是一个Promise对象
  - 　.then(function(response) {return response.json();　})
  - 　.then(function(myJson) {console.log(myJson); });
- 使用async/await 关键字
  - async function foo() {return Promise.resolve("hello");}　//使用async来表明它是异步函数
  - s = await foo();　　//foo()的结果是Promise对象，但await可以取得其中的结果
  - 示例：demo-await.js

# 使用Axios

- 网站
  - http://www.axios-js.com/
  - https://github.com/axios/axios
- 特点
  - 支持node端和浏览器端
  - 支持 Promise异步操作，不用传统callback方式
  - 丰富的配置项，支持拦截器等高级配置

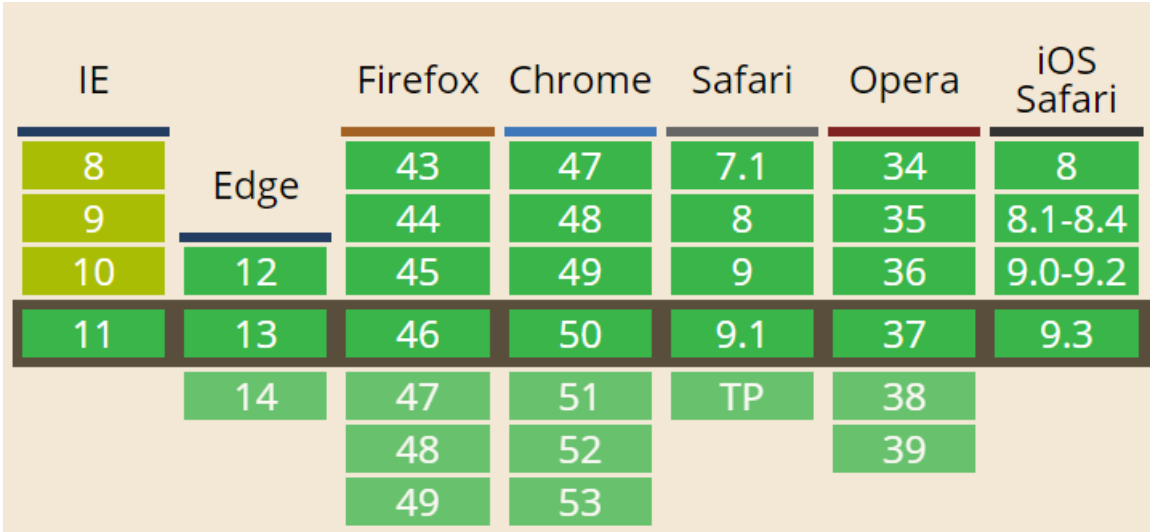
# ajax应用

# ajax不能访问文件

- ajax不支持file://
  - 为了安全考虑
- 解决方法: 使用http服务器
  - 一个简单的http静态服务器
    - 如 Mini-WebServer
    - http://www.downg.com/soft/4247.html
  - Windows自带IIS
  - node中的http-server　(安装node, 安装npm install -g http-server)
  - python中的编程 flask
  - vscode 使用 live server

# ajax的跨域问题

- 一般情况下，ajax不能跨域访问
  - 不能访问其他服务器上的链接
- 解决办法
  - 服务端要进行设置
    - http头中加入 Access-Control-Allow-Origin "*"
  - 客户端要求浏览器版本较新
    - 更多信息请参考
    - http://enable-cors.org
  - 采用jsonp（针对低版本）
    - <script src=...>



# 使用mock

- 使用mock可以模拟产生ajax数据
  - 这样，客户端开发不受服务端开发的影响
- 常用的框架
  - mockjax https://github.com/jakerella/jquery-mockjax
    - 修改$.ajax函数，使得可以用mock数据。缺点：不兼容新的jQuery版本
    - 可以将任意网址mock，网页可以以文件方式访问
    - 引用： https://cdn.bootcss.com/jquery-mockjax/1.6.2/jquery.mockjax.js
  - mock.js http://mockjs.com/推荐
    - 可生成随机数据，但不支持文件访问

# mockjax （了解）

- $.mockjax({ url: ,  responseText: {a:b,c:d} } )
- $.mockjax({ url: ,  response: function(){this.responseText= {a:b,c:d} } )

# mock.js

- <script src="http://mockjs.com/dist/mock.js"></script>

```
//调用mock方法模拟数据
    Mock.mock(
        'http://foo.com', {
            "userName" : '@name',     //模拟名称
            "age|1-100": 100,         //模拟年龄(1-100)
            "color"    : "@color",    //模拟色值
            "date"     : "@date('yyyy-MM-dd')",  //模拟时间
                        "address":{
                                    "url"      : "@url()",     //模拟url
                                    "email"    : "@email()",     //模拟email
                                    "content"  : "@cparagraph()" //模拟文本
                        }
        }
    );
```

# 使用第三方提供的数据服务

- 使用第三方提供的数据服务
  - 聚合数据 https://www.juhe.cn/
  - 如 https://www.wilddog.com/
  - https://github.com/public-apis/public-apis
  - https://github.com/n0shake/Public-APIs
  - 还有百度、新浪等各种网站提供的api

# 实际案例

- 案例
  - 北大主页
  - 北大门户
  - 幕课后台
  - 电子商务
  - 一些应用

# 应用示例

- 动态加载数据
  - 分页
  - 树、卡片、表格
  - 三级联动
  - 瀑布式加载

# 服务端技术简介

# 服务端

- 服务端（后端）
  - 基本的http服务
  - 动态地执行程序
    - 获取客户端传来的数据
    - 进行运算处理（如数据库查询）
    - 向客户端传送数据

# HTTP服务器

- http服务器很多，常见的
  - apache （开源、跨平台）
  - windows平台上的 IIS (Internet Information Services)
  - 基于各种技术开发的http服务器
    - 基于node.js 的 http-server
    - 基于java的 Tomcat
    - 基于php的 ApmServer，XMPP, WAMP

# 服务端技术

- 服务端技术很多，常见的有
  - java技术： javaEE， 使用java语言
  - 微软技术：asp.net，使用C#语言
  - 脚本技术：php, perl, asp等
  - 一些新的语言：python, ruby，go，node.js

# asp.net服务端程序

```
<%@Page language="C#" %>
<%
string q = Request["q"];
double m = Math.Sqrt(Double.Parse(q));
Response.Write(m);
%>
```

# javaEE中的Servlet

```
<servlet>
  <servlet-name>zipCodeServlet</servlet-name>
  <servlet-class>com.newer.ajax.ZipCodeServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>zipCodeServlet</servlet-name>
  <url-pattern>/zipCodeServlet</url-pattern>
</servlet-mapping>
```

servlet路径

```
 * 5. 发送请求。
 */
function callServer() {
    // 从表单中取得城市及状态数据
    var city = document.getElementB
    // 仅当城市的值正确填写后才续继处
    if ((city == null) || (city ==
    // 建立远程连接的WEB对应的URL地址
    var url = "zipCodeServlet?city=
    // 打开一个对服务器的连接
    xmlHttp.open("GET", url, true);
    // 设置一个函数，当从服务器读取数
    xmlHttp.onreadystatechange = up
    // 开始发送HTTP请求
    xmlHttp.send(null);
}
```

发送请求到servlet

```
class ZipCodeServlet extends HttpServlet {

tected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
// 设置客户机对数据不进行缓存
response.setHeader("Cache-Control", "no-store");
response.setHeader("Pragrma", "no-cache");
response.setDateHeader("Expires", 0);
// 取得请求参数值
String city = new String(
        request.getParameter("city").getBytes(
        "ISO-8859-1"), "UTF-8");
// 调用服务器业务方法
String zipCode = ZipCode.getZipCode(city);

PrintWrite                    Writer();
out.println(zipCode);
out.close();
}
```

servlet响应结果

# Ajax+Struts

```
function callServer() {
    // 从表单中取得城市及状态数据
    var city = document.getElementById("
    // 仅当城市的值正确填写后才续继处理
    if ((city == n                    = ""))
    // 建立远程连接                          址
    var url = "zipCode.do?city="+encodeU
    // 打开一个对服务器的连接
    xmlHttp.open("GET", url, true);
    // 设置一个函数，当从服务器读取数据后返
    xmlHttp.onreadystatechange = updateF
    // 开始发送HTTP请求
    xmlHttp.send(null);
}
```

发送请求到struts

```
<action-mappings>
  <action path="/zipCode"
          type="com.newer.ajax.ZipCodeAction">
  </action>
</action-mappings>
```

struts配置

```
public class ZipCodeAction extends Action {
    public ActionForward execute(ActionMapping mapping,
                                 ActionForm form,
                                 HttpServletRequest request,
                                 HttpServletResponse response)
            throws Exception {
        // 设置客户机对数据不进行缓存
        response.setHeader("Cache-Control", "no-store");
        response.setHeader("Pragma", "no-cache");
        response.setDateHeader("Expires", 0);
        // 取得请求参数值
        String city = new String(request.getParameter("city").getBytes(
                "ISO-885
        // 调用服务器业务
        String zipCode = ZipCode.getZipCode(city);
        PrintWriter out = response.getWriter();
        out.println(zipCode);
        out.close();
        return null;
    }
}
```

struts响应结果

# 数据库技术

- 数据库管理系统(DBMS)
- 常见的数据库
  - Oracle 企业级数据库
  - Sql Server 微软数据库
  - MySql 开源的数据库
  - MongoDB、Redis 新型的数据库
  - Sqlite, Access 嵌入型、桌面型数据库

# node.js 简介

# node.js

- http://nodejs.org
- Node.js 是服务端的JavaScript
  - 一个基于 Chrome V8 引擎的 JavaScript 运行环境。
  - Node.js 使用了一个事件驱动、非阻塞式 I/O 的模型，使其轻量又高效。

# 命令行方式使用

- 命令行方式
  - 又称：交互式解释器(REPL, Read-Eval-Print-Loop)
  - 直接输入JavaScript表达式，或语句、函数
  - 多行使用 Ctrl+回车
  - 使用console.log( )来显示信息
  - .help 帮助 （注意前面的点）
  - .exit 退出 （注意前面的点，或者按Ctrl+C两次）
- 直接运行js
  - node xxx.js

# node.js与网页中的js的区别

- 语言基本一样
  - 但node.js支持更新的一些语言特性，如require语句等 ES5 ES6
- 运行环境不同
  - node.js中不存在window，document等对象
  - node.js中也可以有console.log()
  - node.js可以存取本地文件、数据库
    - 示例：demo-readfile.js
  - 可以访问别的server上的api

# npm

- npm
  - Node.js 的包管理器 npm
  - 是全球最大的开源库生态系统
- 使用npm来安装各种软件
  - npm install express

# 基于node.js的应用

- 服务端可以基于node.js来写各种应用
- 也可借助一定的框架
  - 如express是快速、开放、极简的 web 开发框架
  - 参考 http://www.expressjs.com.cn/
- 其中，有路由功能，可以直接获得参数
  - 示例： expressHello.js

# 类似地，Flask

- Flask是用python来运行的
- 其中有路由/参数获取/模板展示等功能

  - 示例： flaskHello.py

# 小结

- form （复习）
  - method, action, input, name
- ajax （掌握）
  - $.ajax({
    - url:'http://aa.com/bb', type:'GET', data:{xxx:xxx},
    - success:function(data){...}, error:function(){...} )
- mock.js （了解，会调用即可）
  - Mock.mock(url: '...', {'aa':'@name', 'age|18-20':18} } )
- node js (了解)
- express / flask 初步 (了解)

作者：唐大仕
北京大学信息科学技术学院。