# AJAX（Asynchronous Javascript And Xml）

## 传统请求及缺点

- 传统的请求都有哪些？
  - 直接在浏览器地址栏上输入URL。
  - 点击超链接
  - 提交form表单
  - 使用JS代码发送请求
    - window.open(url)
    - document.location.href = url
    - window.location.href = url
    - ....
- 传统请求存在的问题
  - 页面全部刷新导致了用户的体验较差。
  - 传统的请求导致用户的体验有空白期。（用户的体验是不连贯的）
  -

## AJAX概述

- AJAX不能称为一种技术，它是多种技术的综合产物。
- AJAX可以让浏览器发送一种特殊的请求，这种请求可以是：异步的。
- 什么是异步，什么是同步？
  - 假设有t1和t2线程，t1和t2线程并发，就是异步。
  - 假设有t1和t2线程，t2在执行的时候，必须等待t1线程执行到某个位置之后t2才能执行，那么t2在等t1，显然他们是排队的，排队的就是同步。
  - AJAX是可以发送异步请求的。也就是说，在同一个浏览器页面当中，可以发送多个ajax请求，这些ajax请求之间不需要等待，是并发的。
- AJAX代码属于WEB前端的JS代码。和后端的java没有关系，后端也可以是php语言，也可以是C语言。
- AJAX 应用程序可能使用 XML 来传输数据，但将数据作为纯文本或 JSON 文本传输也同样常见。
- AJAX可以更新网页的部分，而不需要重新加载整个页面。（页面局部刷新）
- AJAX可以做到在同一个网页中同时启动多个请求，类似于在同一个网页中启动“多线程”，一个“线程”一个“请求”。
-
-

## XMLHttpRequest对象

- XMLHttpRequest对象是AJAX的核心对象，发送请求以及接收服务器数据的返回，全靠它了。
- XMLHttpRequest对象，现代浏览器都是支持的，都内置了该对象。直接用即可。
- 创建XMLHttpRequest对象

    - `var xhr = new XMLHttpRequest();`

- XMLHttpRequest对象的方法

| 方法 | 描述 |
| --- | --- |
| abort() | 取消当前请求 |
| getAllResponseHeaders() | 返回头部信息 |
| getResponseHeader() | 返回特定的头部信息 |
| open(*method*, *url*, *async*, *user*, *psw*) | 规定请求method：请求类型 GET 或 POSTurl：文件位置async：true（异步）或 false（同步）user：可选的用户名称psw：可选的密码 |
| send() | 将请求发送到服务器，用于 GET 请求 |
| send(*string*) | 将请求发送到服务器，用于 POST 请求 |
| setRequestHeader() | 向要发送的报头添加标签/值对 |

- XMLHttpRequest对象的属性

| 属性 | 描述 |
| --- | --- |
| onreadystatechange | 定义当 readyState 属性发生变化时被调用的函数 |
| readyState | 保存 XMLHttpRequest 的状态。0：请求未初始化 1：服务器连接已建立 2：请求已收到 3：正在处理请求 4：请求已完成且响应已就绪 |
| responseText | 以字符串返回响应数据 |
| responseXML | 以 XML 数据返回响应数据 |
| status | 返回请求的状态号200: "OK"403: "Forbidden"404: "Not Found" |
| statusText | 返回状态文本（比如 "OK" 或 "Not Found"） |

# AJAX GET请求

- 发送AJAX get请求，前端代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>发送ajax get请求</title>
</head>
<body>
<script type="text/javascript">
    window.onload = function () {
        document.getElementById("btn").onclick = function () {
            //1. 创建AJAX核心对象
            var xhr = new XMLHttpRequest();
            //2. 注册回调函数
            xhr.onreadystatechange = function(){
```

```
                if (this.readyState == 4) {
                    if (this.status == 200) {
                        // 通过XMLHttpRequest对象的responseText属性可以获取到服务器响应回来的内容。
                        // 并且不管服务器响应回来的是什么，都以普通文本的形势获取。（服务器可能响应回来：普通文本、XML、JSON、HTML...）
                        // innerHTML属性是javascript中的语法，和ajax的XMLHttpRequest对象无关。
                        // innerHTML可以设置元素内部的HTML代码。（innerHTML可以将后面的内容当做一段HTML代码解释并执行）
                        //document.getElementById("myspan").innerHTML = this.responseText
                        document.getElementById("mydiv").innerHTML = this.responseText
                        // innerText也不是AJAX中的，是javascript中的元素属性，和XMLHttpRequest无关。
                        // innerText也是设置元素中的内容，但是即使后面是一段HTML代码，也是将其看做一个普通字符串设置进去。
                        //document.getElementById("myspan").innerText = this.responseText
                    }else{
                        alert(this.status)
                    }
                }
            }
            //3. 开启通道
            xhr.open("GET", "/ajax/ajaxrequest2", true)
            //4. 发送请求
            xhr.send()
        }
    }
</script>
<button id="btn">发送ajax get请求</button>
<span id="myspan"></span>
<div id="mydiv"></div>
</body>
</html>
```

- 发送AJAX get请求，后端代码：

```
package com.bjpowernode.ajax.servlet;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;

/**
 * @program: 代码
 * @ClassName: AjaxRequest2Servlet
 * @version: 1.0
 * @description:
 * @author: bjpowernode
```

```
 * @create: 2022-05-13 10:46
 **/

@WebServlet("/ajaxrequest2")
public class AjaxRequest2Servlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        // 设置响应的内容类型以及字符集
        response.setContentType("text/html;charset=UTF-8");
        // 获取响应流
        PrintWriter out = response.getWriter();
        // 响应
        out.print("<font color='red'>用户名已存在！！！</font>");
    }
}
```

- AJAX get请求如何提交数据呢？
  - get请求提交数据是在“请求行”上提交，格式是：url?name=value&name=value&name=value....
  - 其实这个get请求提交数据的格式是HTTP协议中规定的，遵循协议即可。

# AJAX GET请求的缓存问题

- 对于低版本的IE浏览器来说，AJAX的get请求可能会走缓存。存在缓存问题。对于现代的浏览器来说，大部分浏览器都已经不存在AJAX get缓存问题了。
- 什么是AJAX GET请求缓存问题呢？
  - 在HTTP协议中是这样规定get请求的：get请求会被缓存起来。
  - 发送AJAX GET请求时，在同一个浏览器上，前后发送的AJAX请求路径一样的话，对于低版本的IE来说，第二次的AJAX GET请求会走缓存，不走服务器。
- POST请求在HTTP协议中规定的是：POST请求不会被浏览器缓存。
- GET请求缓存的优缺点：
  - 优点：直接从浏览器缓存中获取资源，不需要从服务器上重新加载资源，速度较快，用户体验好。
  - 缺点：无法实时获取最新的服务器资源。
- 浏览器什么时候会走缓存？
  - 第一：是一个GET请求
  - 第二：请求路径已经被浏览器缓存过了。第二次发送请求的时候，这个路径没有变化，会走浏览器缓存。
- 如果是低版本的IE浏览器，怎么解决AJAX GET请求的缓存问题呢？
  - 可以在请求路径url后面添加一个时间戳，这个时间戳是随时变化的。所以每一次发送的请求路径都是不一样的，这样就不会走浏览器的缓存问题了。
  - 可以采用时间戳："url?t=" + new Date().getTime()
  - 或者可以通过随机数："url?t=" + Math.random()
  - 也可以随机数+时间戳....

# AJAX POST请求

- AJAX POST请求和GET请求的代码区别在哪里？就是前端代码有区别。后端代码没有区别。

```
// 4. 发送AJAX POST请求
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded") //
设置请求头的内容类型。模拟form表单提交数据。
// 获取表单中的数据
var username = document.getElementById("username").value;
var password = document.getElementById("password").value;
// send函数中的参数就是发送的数据，这个数据在“请求体”当中发送。
xhr.send("username="+username+"&password="+password)
```

- 实现一个案例：使用AJAX POST请求实现用户注册的时候，用户名是否可用。（验证用户名是否可以注册）实现步骤如下：
  - 在前端，用户输入用户名之后，失去焦点事件blur发生，然后发送AJAX POST请求，提交用户名
  - 在后端，接收到用户名，连接数据库，根据用户名去表中搜索
  - 如果用户名已存在
    - 后端响应消息：对不起，用户名已存在（在前端页面以红色字体展示）
  - 如果用户名不存在
    - 后端响应消息：用户名可以使用（在前端页面以绿色字体展示）
- 实现一个案例：用户点击按钮之后，发送AJAX请求，显示学生列表。
  - 在后端java程序中拼接HTML代码，然后将HTML代码直接响应到浏览器客户端。这种方式不好，不应该在java代码中编写HTML代码，能否在java程序中直接向前端响应数据？可以，可以在后端拼接JSON格式的字符串，或者XML格式的字符串，将这个字符串发送给前端，前端解析即可。

# 基于JSON的数据交换

- 在WEB前端中，如何将一个json格式的字符串转换成json对象

```
var jsonStr = "{\"username\" : \"zhangsan\", \"password\" : \"1233344\"}"
var jsonObj = JSON.parse(jsonStr)
console.log(jsonObj.username)
console.log(jsonObj.password)
```

- 在后端拼接JSON格式的字符串，响应给前端的浏览器

```
json.append("[");
while (rs.next()) {
    // 获取每个学生的信息
    String name = rs.getString("name");
    String age = rs.getString("age");
    String addr = rs.getString("addr");
    // 拼接json格式的字符串
    // {"name":"    王五    ","age":    20      ,"addr":"       北京大兴区
"},
    json.append("{\"name\":\"");
    json.append(name);
    json.append("\",\"age\":");
    json.append(age);
    json.append(",\"addr\":\"");
    json.append(addr);
    json.append("\"},");
}
```

```
jsonStr = json.substring(0, json.length() - 1) + "]";
```

- 拼接JSON格式的字符串太痛苦，可以使用阿里巴巴的fastjson组件，它可以将java对象转换成json格式的字符串

```
List<Student> studentList = new ArrayList<>();
while (rs.next()) {
    // 取出数据
    String name = rs.getString("name");
    int age = rs.getInt("age");
    String addr = rs.getString("addr");
    // 将以上数据封装成Student对象
    Student s = new Student(name, age, addr);
    // 将Student对象放到List集合
    studentList.add(s);
}
// 将List集合转换成json字符串
jsonStr = JSON.toJSONString(studentList);
```

注意：使用fastjson需要引入fastjson-1.2.2.jar

# 基于XML的数据交换

- 注意：如果服务器端响应XML的话，响应的内容类型需要写成：

```
response.setContentType("text/xml;charset=UTF-8");
```

- xml和JSON都是常用的数据交换格式
  - XML体积大，解析麻烦。较少用。
  - JSON体积小，解析简单，较常用。
- 基于XML的数据交换，前端代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>使用XML完成数据交换</title>
</head>
<body>
<script type="text/javascript">
    window.onload = function(){
        document.getElementById("btn").onclick = function(){
            // 1.创建XMLHTTPRequest对象
            var xhr = new XMLHttpRequest();
            // 2.注册回调函数
            xhr.onreadystatechange = function () {
                if (this.readyState == 4) {
                    if (this.status == 200) {
                        // 服务器端响应了一个XML字符串，这里怎么接收呢？
                        // 使用XMLHTTPRequest对象的responseXML属性，接收返回之后，可以自动封装成document对象（文档对象）
                        var xmlDoc = this.responseXML
                        //console.log(xmlDoc)
                        // 获取所有的<student>元素，返回了多个对象，应该是数组。
```

```
                        var students =
xmlDoc.getElementsByTagName("student")
                        //console.log(students[0].nodeName)
                        var html = "";
                        for (var i = 0; i < students.length; i++) {
                            var student = students[i]
                            // 获取<student>元素下的所有子元素
                            html += "<tr>"
                            html += "<td>"+(i+1)+"</td>"
                            var nameOrAge = student.childNodes
                            for (var j = 0; j < nameOrAge.length; j++) {
                                var node = nameOrAge[j]
                                if (node.nodeName == "name") {
                                    //console.log("name = " +
node.textContent)
                                    html += "<td>"+node.textContent+"</td>"
                                }
                                if (node.nodeName == "age") {
                                    //console.log("age = " +
node.textContent)
                                    html += "<td>"+node.textContent+"</td>"
                                }
                            }
                            html += "</tr>"
                        }
                        document.getElementById("stutbody").innerHTML = html
                    }else{
                        alert(this.status)
                    }
                }
            }
            // 3.开启通道
            xhr.open("GET", "/ajax/ajaxrequest6?t=" + new Date().getTime(),
true)
            // 4.发送请求
            xhr.send()
        }
    }
</script>
<button id="btn">显示学生列表</button>
<table width="500px" border="1px">
    <thead>
    <tr>
        <th>序号</th>
        <th>姓名</th>
        <th>年龄</th>
    </tr>
    </thead>
    <tbody id="stutbody">
    <!--<tr>
        <td>1</td>
        <td>zhangsan</td>
        <td>20</td>
    </tr>
    <tr>
        <td>2</td>
        <td>lisi</td>
        <td>22</td>
```

```
    </tr>-->
    </tbody>
</table>
</body>
</html>
```

- 基于XML的数据交换，后端java程序：

```
package com.bjpowernode.ajax.servlet;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;

/**
 * @program: 代码
 * @ClassName: AjaxRequest6Servlet
 * @version: 1.0
 * @description: 服务器端返回XML字符串
 * @author: bjpowernode
 * @create: 2022-05-15 11:48
 **/
@WebServlet("/ajaxrequest6")
public class AjaxRequest6Servlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        // 注意：响应的内容类型是XML。
        response.setContentType("text/xml;charset=UTF-8");
        PrintWriter out = response.getWriter();

        /*
        <students>
            <student>
                <name>zhangsan</name>
                <age>20</age>
            </student>
            <student>
                <name>lisi</name>
                <age>22</age>
            </student>
        </students>
         */

        StringBuilder xml = new StringBuilder();
        xml.append("<students>");
        xml.append("<student>");
        xml.append("<name>zhangsan</name>");
        xml.append("<age>20</age>");
        xml.append("</student>");
        xml.append("<student>");
```

```
        xml.append("<name>lisi</name>");
        xml.append("<age>22</age>");
        xml.append("</student>");
        xml.append("</students>");

        out.print(xml);
    }
}
```

# AJAX乱码问题

- 测试内容：
  - 发送ajax get请求
    - 发送数据到服务器，服务器获取的数据是否乱码？
    - 服务器响应给前端的中文，会不会乱码？
  - 发送ajax post请求
    - 发送数据到服务器，服务器获取的数据是否乱码？
    - 服务器响应给前端的中文，会不会乱码？
- 包括还要测试tomcat服务器的版本：
  - tomcat10和tomcat9都要进行测试。
- 测试结果：
  - 对于tomcat10来说，关于字符集，我们程序员不需要干涉，不会出现乱码。
  - 对于tomcat9来说呢？
    - 响应中文的时候，会出现乱码，怎么解决？

      ```
      response.setContentType("text/html;charset=UTF-8");
      ```

    - 发送ajax post请求的时候，发送给服务器的数据，服务器接收之后乱码，怎么解决？

      ```
      request.setCharacterEncoding("UTF-8");
      ```

# AJAX的异步与同步

- 什么是异步？什么是同步？
  - ajax请求1和ajax请求2，同时并发，谁也不用等谁，这就是异步。（a不等b，b也不等a）
  - 如果ajax请求1在发送的时候需要等待ajax请求2结束之后才能发送，那么这就是同步。（a等待b，或者b等待a，只要发生等待，就是同步。）
- 异步和同步在代码上如何实现？

```
// 假设这个是ajax请求1
// 如果第三个参数是false：这个就表示“ajax请求1”不支持异步，也就是说ajax请求1发送之后，会影响其他ajax请求的发送，只有当我这个请求结束之后，你们其他的ajax请求才能发送。
// false表示，不支持异步。我这个请求发了之后，你们其他的请求都要靠边站。都等着。你们别动呢，等我结束了你们再说。
xhr1.open("请求方式", "URL", false)
xhr1.send()

// 假设这个是ajax请求2
// 如果第三个参数是true：这个就表示“ajax请求2”支持异步请求，也就是说ajax请求2发送之后，不影响其他ajax请求的发送。
xhr2.open("请求方式", "URL", true)
xhr2.send()
```

- 什么情况下用同步？（大部分情况下我们都是使用ajax异步方式，同步很少用。）
  - 举一个例子
    - 用户注册
      - 用户名需要发送ajax请求进行校验
      - 邮箱地址也需要发送ajax请求校验
      - 其他的也可能需要发送ajax请求。。。
      - 并且最终注册按钮的时候，也是发送ajax请求进行注册。
      - 那么显然，注册的Ajax请求和校验的ajax请求不能异步，必须等待所有的校验ajax请求结束之后，注册的ajax请求才能发。

# AJAX代码封装

- AJAX请求相关的代码都是类似的，有很多重复的代码，这些重复的代码能不能不写，能不能封装一个工具类。要发送ajax请求的话，就直接调用这个工具类中的相关函数即可。
- 接下来，手动封装一个工具类，这个工具类我们可以把它看做是一个JS的库。我们把这个JS库起一个名字，叫做jQuery。（我这里封装的jQuery只是一个前端的库，和后端的java没有关系，只是为了方便web前端代码的编写，提高WEB前端的开发效率）
- 手动开发jQuery，源代码

```
function jQuery(selector){
    if (typeof selector == "string") {
        if (selector.charAt(0) == "#") {
            domObj = document.getElementById(selector.substring(1))
            return new jQuery()
        }
    }
    if (typeof selector == "function") {
        window.onload = selector
    }
    this.html = function(htmlStr){
        domObj.innerHTML = htmlStr
    }
    this.click = function(fun){
        domObj.onclick = fun
    }
    this.focus = function (fun){
        domObj.onfocus = fun
    }
    this.blur = function(fun) {
```

```
        domObj.onblur = fun
    }
    this.change = function (fun){
        domObj.onchange = fun
    }
    this.val = function(v){
        if (v == undefined) {
            return domObj.value
        }else{
            domObj.value = v
        }
    }

    // 静态的方法，发送ajax请求
    /**
     * 分析：使用ajax函数发送ajax请求的时候，需要程序员给我们传过来什么？
     *       请求的方式(type)：GET/POST
     *       请求的URL(url)：url
     *       请求时提交的数据(data)：data
     *       请求时发送异步请求还是同步请求(async)：true表示异步，false表示同步。
     */
    jQuery.ajax = function(jsonArgs){
        // 1.
        var xhr = new XMLHttpRequest();
        // 2.
        xhr.onreadystatechange = function(){
            if (this.readyState == 4) {
                if (this.status == 200) {
                    // 我们这个工具类在封装的时候，先不考虑那么多，假设服务器返回的都是
json格式的字符串。
                    var jsonObj = JSON.parse(this.responseText)
                    // 调用函数
                    jsonArgs.success(jsonObj)
                }
            }
        }

        if (jsonArgs.type.toUpperCase() == "POST") {
            // 3.
            xhr.open("POST", jsonArgs.url, jsonArgs.async)
            // 4.
            xhr.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded")
            xhr.send(jsonArgs.data)
        }

        if (jsonArgs.type.toUpperCase() == "GET") {
            xhr.open("GET", jsonArgs.url + "?" + jsonArgs.data,
jsonArgs.async)
            xhr.send()
        }

    }
}
$ = jQuery

// 这里有个细节，执行这个目的是为了让静态方法ajax生效。
new jQuery()
```

- 使用以上库，怎么用?

```
<script type="text/javascript" src="/ajax/js/jQuery-1.0.0.js"></script>
<script type="text/javascript">
    $(function(){
        $("#btn1").click(function(){
            $.ajax({
                type : "POST",
                url : "/ajax/ajaxrequest11",
                data : "username=" + $("#username").val(),
                async : true,
                success : function(json){
                    $("#div1").html(json.uname)
                }
            })
        })
    })
</script>
```

# AJAX实现省市联动

- 什么是省市联动?
  - 在网页上，选择对应的省份之后，动态的关联出该省份对应的市。选择对应的市之后，动态的关联出该市对应的区。（首先要清楚需求）
- 进行数据库表的设计
  - ```
    t_area （区域表）
    id(PK-自增)      code       name         pcode
    ------------------------------------------------
    1                001        河北省         null
    2                002        河南省         null
    3                003        石家庄         001
    4                004        邯郸           001
    5                005        郑州           002
    6                006        洛阳           002
    7                007        丛台区         004

    将全国所有的省、市、区、县等信息都存储到一张表当中。
    采用的存储方式实际上是code pcode形势。
    ```
- 建表t_area，模拟好数据。
- 首先实现第一个功能:
  - 页面加载完毕之后，先把省份全部展现出来。

# AJAX跨域问题
    ```

# 跨域

- 跨域是指从一个域名的网页去请求另一个域名的资源。比如从百度(https://baidu.com)页面去请求京东(https://www.jd.com)的资源。
- 通过超链接或者form表单提交或者window.location.href的方式进行跨域是不存在问题的（**大家可以编写程序测试一下**）。但在一个域名的网页中的一段js代码发送ajax请求去访问另一个域名中的资源，由于同源策略的存在导致无法跨域访问，那么ajax就存在这种跨域问题。
- 同源策略是指一段脚本只能读取来自同一来源的窗口和文档的属性，同源就是协议、域名和端口都相同。
- 同源策略有什么用？如果你刚刚在网银输入账号密码，查看了自己还有1万块钱，紧接着访问一些不规矩的网站，这个网站可以访问刚刚的网银站点，并且获取账号密码，那后果可想而知。所以，从安全的角度来讲，同源策略是有利于保护网站信息的。
- 有一些情况下，我们是需要使用ajax进行跨域访问的。比如某公司的A页面(a.bjpowernode.com)有可能需要获取B页面(b.bjpowernode.com)。

# 同源还是不同源

- 区分同源和不同源的三要素
-
  - 协议
  - 域名
  - 端口
- 协议一致，域名一致，端口号一致，三个要素都一致，才是同源，其它一律都是不同源

| URL1 | URL2 | 是否同源 | 描述 |
| --- | --- | --- | --- |
| http://localhost:8080/a/index.html | http://localhost:8080/a/first | 同源 | 协议 域名 端口一致 |
| http://localhost:8080/a/index.html | http://localhost:8080/b/first | 同源 | 协议 域名 端口一致 |
| http://www.myweb.com:8080/a.js | https://www.myweb.com:8080/b.js | 不同源 | 协议不同 |
| http://www.myweb.com:8080/a.js | http://www.myweb.com:8081/b.js | 不同源 | 端口不同 |
| http://www.myweb.com/a.js | http://www.myweb2.com/b.js | 不同源 | 域名不同 |
| http://www.myweb.com/a.js | http://crm.myweb.com/b.js | 不同源 | 子域名不同 |

# 复现AJAX跨域问题

# AJAX跨域解决方案

## 方案1：设置响应头

- 核心原理：跨域访问的资源允许你跨域访问。
- 实现：

```
response.setHeader("Access-Control-Allow-Origin",
"http://localhost:8080"); // 允许某个
response.setHeader("Access-Control-Allow-Origin", "*"); // 允许所有
```

### 方案2：jsonp

- jsonp：json with padding（带填充的json【学完之后再理解这个什么意思！！！】）
- jsonp不是一个真正的ajax请求。只不过可以完成ajax的局部刷新效果。可以说jsonp是一种类ajax请求的机制。
- jsonp不是ajax请求，但是可以完成局部刷新的效果，并且可以解决跨域问题。
- 注意：jsonp解决跨域的时候，只支持GET请求。不支持post请求。

### 方案3：jQuery封装的jsonp

- 牛人们写的jQuery库，已经对jsonp进行了封装。大家可以直接拿来用。
- 用之前需要引入jQuery库的js文件。（这里的jQuery库咱们就不再封装了，咱们直接用jQuery写好的jsonp方式。）
- jQuery中的jsonp其实就是我们方案2的高度封装，底层原理完全相同。
- 核心代码

```
$.ajax({
    type : "GET",
    url : "跨域的url",
    dataType : "jsonp", // 指定数据类型
    jsonp : "fun", // 指定参数名（不设置的时候，默认是："callback"）
    jsonpCallback : "sayHello" // 指定回调函数的名字
                               // （不设置的时候，jQuery会自动生成一个随机的回调函数，
                               //并且这个回调函数还会自动调用success的回调函数。）
})
```

### 方案4：代理机制（httpclient）

- 使用Java程序怎么去发送get/post请求呢？【GET和POST请求就是HTTP请求。】
  - 第一种方案：使用JDK内置的API（java.net.URL.....），这些API是可以发送HTTP请求的。
  - 第二种方案：使用第三方的开源组件，比如：apache的httpclient组件。（httpclient组件是开源免费的，可以直接用）
- 在java程序中，使用httpclient组件可以发送http请求。
  - 对于httpclient组件的代码，大家目前可以不进行深入的研究，可以从网上直接搜。然后粘贴过来，改一改，看看能不能完成发送get和post请求。
  - 使用httpclient组件，需要先将这个组件相关的jar包引入到项目当中。

### 方案5：nginx反向代理

- nginx反向代理中也是使用了这种代理机制来完成AJAX的跨域，实现起来非常简单，只要修改一个nginx的配置即可。以后大家学习nginx之后再说吧。！！！！

# AJAX实现搜索联想 自动补全

- 什么是搜索联想？自动补全？

    - 百度是一个很典型的代表。在百度的搜索框中输入相关信息的时候，会有搜索联想以及自动补全。
    - 搜索联想和自动补全：实际上是为了方便用户的使用。让用户的体验更好。
    - 搜索联想：当用户输入一些单词之后，自动联想出用户要搜索的信息，给一个提示。
    - 自动补全：当联想出一些内容之后，用户点击某个联想的单词，然后将这个单词自动补全到搜索框当中。
    - 搜索联想和自动补全功能，因为是页面局部刷新效果，所以需要使用ajax请求来完成。
- 搜索联想，自动补全功能的核心实现原理？
    - 当键盘事件发生之后，比如：keyup：键弹起事件。
    - 发送ajax请求，请求中提交用户输入的搜索内容，例如：北京（发送ajax请求，携带“北京”两个字）
    - 后端接收到ajax请求，接收到“北京”两个字，执行select语句进行模糊查询。返回查询结果。
    - 将查询结果封装成json格式的字符串，将json格式的字符串响应到前端。
    - 前端接收到json格式的字符串之后，解析这个json字符串，动态展示页面。

# 附录：HTTP状态信息

## 1xx: 信息

| 消息: | 描述: |
| --- | --- |
| 100 Continue | 服务器仅接收到部分请求，但是一旦服务器并没有拒绝该请求，客户端应该继续发送其余的请求。 |
| 101 Switching Protocols | 服务器转换协议：服务器将遵从客户的请求转换到另外一种协议。 |

## 2xx: 成功

| 消息: | 描述: |
| --- | --- |
| 200 OK | 请求成功（其后是对GET和POST请求的应答文档。） |
| 201 Created | 请求被创建完成，同时新的资源被创建。 |
| 202 Accepted | 供处理的请求已被接受，但是处理未完成。 |
| 203 Non-authoritative Information | 文档已经正常地返回，但一些应答头可能不正确，因为使用的是文档的拷贝。 |
| 204 No Content | 没有新文档。浏览器应该继续显示原来的文档。如果用户定期地刷新页面，而Servlet可以确定用户文档足够新，这个状态代码是很有用的。 |
| 205 Reset Content | 没有新文档。但浏览器应该重置它所显示的内容。用来强制浏览器清除表单输入内容。 |
| 206 Partial Content | 客户发送了一个带有Range头的GET请求，服务器完成了它。 |

## 3xx: 重定向

| 消息: | 描述: |
| --- | --- |
| 300 Multiple Choices | 多重选择。链接列表。用户可以选择某链接到达目的地。最多允许五个地址。 |
| 301 Moved Permanently | 所请求的页面已经转移至新的url。 |
| 302 Found | 所请求的页面已经临时转移至新的url。 |
| 303 See Other | 所请求的页面可在别的url下被找到。 |
| 304 Not Modified | 未按预期修改文档。客户端有缓冲的文档并发出了一个条件性的请求（一般是提供If-Modified-Since头表示客户只想比指定日期更新的文档）。服务器告诉客户，原来缓冲的文档还可以继续使用。 |
| 305 Use Proxy | 客户请求的文档应该通过Location头所指明的代理服务器提取。 |
| 306 *Unused* | 此代码被用于前一版本。目前已不再使用，但是代码依然被保留。 |
| 307 Temporary Redirect | 被请求的页面已经临时移至新的url。 |

## 4xx: 客户端错误

| 消息: | 描述: |
| --- | --- |
| 400 Bad Request | 服务器未能理解请求。 |
| 401 Unauthorized | 被请求的页面需要用户名和密码。 |
| 402 Payment Required | 此代码尚无法使用。 |
| 403 Forbidden | 对被请求页面的访问被禁止。 |
| 404 Not Found | 服务器无法找到被请求的页面。 |
| 405 Method Not Allowed | 请求中指定的方法不被允许。 |
| 406 Not Acceptable | 服务器生成的响应无法被客户端所接受。 |
| 407 Proxy Authentication Required | 用户必须首先使用代理服务器进行验证，这样请求才会被处理。 |
| 408 Request Timeout | 请求超出了服务器的等待时间。 |
| 409 Conflict | 由于冲突，请求无法被完成。 |
| 410 Gone | 被请求的页面不可用。 |
| 411 Length Required | "Content-Length" 未被定义。如果无此内容，服务器不会接受请求。 |
| 412 Precondition Failed | 请求中的前提条件被服务器评估为失败。 |
| 413 Request Entity Too Large | 由于所请求的实体的太大，服务器不会接受请求。 |
| 414 Request-url Too Long | 由于url太长，服务器不会接受请求。当post请求被转换为带有很长的查询信息的get请求时，就会发生这种情况。 |
| 415 Unsupported Media Type | 由于媒介类型不被支持，服务器不会接受请求。 |
| 416 | 服务器不能满足客户在请求中指定的Range头。 |
| 417 Expectation Failed | |

## 5xx: 服务器错误

| 消息: | 描述: |
| --- | --- |
| 500 Internal Server Error | 请求未完成。服务器遇到不可预知的情况。 |
| 501 Not Implemented | 请求未完成。服务器不支持所请求的功能。 |
| 502 Bad Gateway | 请求未完成。服务器从上游服务器收到一个无效的响应。 |
| 503 Service Unavailable | 请求未完成。服务器临时过载或当机。 |
| 504 Gateway Timeout | 网关超时。 |
| 505 HTTP Version Not Supported | 服务器不支持请求中指明的HTTP协议版本。 |