一、JSON

1、定义

JSON（JavaScript Object Notation, JS 对象简谱）是一种轻量级的**数据交换格式**。它基于 ECMAScript（欧洲计算机协会制定的 JS 规范）的一个子集，采用完全**独立于编程语言**的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成，并**有效地提升网络传输效率**。

2、格式

JSON 就是一种有格式的字符串。
任何支持的类型都可以通过 JSON 来表示，例如字符串、数字、对象、数组等。但是对象和数组是比较特殊且常用的两种类型。
规则如下:

- 映射用冒号（":"）表示。"名称" : 值，标准格式名称用双引号括起来;
- 并列的数据之间用逗号（","）分隔。"名称1" : 值1, "名称2" : 值2;
- 映射的集合（对象）用大括号（"{}"）表示。{"名称1" : 值1, "名称2": 值2}
- 并列数据的集合（数组）用方括号("[]")表示。示例如下:
  [
  {"名称1" : 值, "名称2" : 值2},
  {"名称1" : 值, "名称2" : 值2}
  ]
- 元素值可具有的类型：string，number，object，array，true，false，null。

二、在 JavaScript 中的 JSON

1、表示 JSON

在 webapp/jq_02/01.json.html，代码如下:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
        // 写 JSON，这个格式的字符串里面存一个员工数据 id 1 name zs age 18
        var json1 = '{"id":1,"name":"zs","age":18}'; // JSON
        // 写 JSON，这个格式的字符串里面存两个个员工数据 id 1 name zs age 18 id 2 name
ls age 19
        var json2 = '[{"id":1,"name":"zs","age":18},
{"id":2,"name":"ls","age":19}]';
        // 写 JSON，这个格式的字符串里面存一个员工数据 id 1 name zs age 18 部门 id 5
name 开发部
        var json3 = '{"id":1, "name":"zs", "age":18, "dept":{"id":5, "name":"开发
部"}}';
```

```
        var jsObj1 = {"id":1, "name":"zs", "age":18};   // JS 对象
        var jsObj2 = {id:1, name:"zs", age:18};         // JS 对象

        console.log(json1.name);    // undefined
        console.log(jsObj1.name);   // zs
        console.log(jsObj2.name);   // zs
    </script>
</head>
<body>

</body>
</html>
```

2、JSON 与 JS 对象转换

浏览器环境提供一个工具类名叫JSON，里面提供方法帮我们实现 JSON 与 JS 对象之间的转换。接着在上面页面的 script 标签中加入下面的代码：

```
//假设获取到服务器响应的数据是 JSON 格式，想获取到具体数据怎么？
// 有一种方式切割字符串，不可取的
// 另一种方式，JSON 是浏览器环境提供一个工具，里面提供方法实现两者的转换

// JSON 字符串转 JS 对象
console.log(JSON.parse(json1).age);
console.log(JSON.parse(json2));
console.log(JSON.parse(json3).dept.name);

// JS 对象转 JSON 字符串
console.log(JSON.stringify(jsObj2));

var json4 = "{'id':1,'name':'zs','age':18}"; // 错误格式的 JSON
console.log(JSON.parse(json4)); // 报错
```


三、在 Java 中的 JSON

1、表示 JSON

新建一个测试类 JsonTest，演示在 Java 中表示 JSON。

```
package cn.wolfcode.json;

public class JsonTest {
    @Test
    public void testJson() {
        String jsonStr = "{\"id\":1,\"name\":\"开发部\",\"sn\":\"DEV\"}";
    }
}
```

2、JSON 与 Java 对象转换

开发中一般都会使用第三方的一些 JSON 操作的依赖或者 JAR 包来来完成 Java 对象与 JSON 字符串之间的转换。
在 Java 中，转换 JSON 的依赖或者 JAR 有很多，这里单讲两种常用：

- Jackson：在 Spring MVC 中内置支持她，速度也挺快，稳定性比较好。
- Fastjson：阿里出品，号称是 Java 领域中转换 JSON 最快的一个插件，中文文档比较齐全。

3、Jackson

3.1、添加依赖

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.6</version>
</dependency>
```

3.2、API 使用

```
package cn.wolfcode.json;

public class JsonTest {
    @Test
    public void testJackson() throws Exception {
        Department department = new Department();
        department.setId(1L);
        department.setName("开发部");
        department.setSn("DEV");

        ObjectMapper objectMapper = new ObjectMapper();
        // Java 对象转 JSON
        System.out.println(objectMapper.writeValueAsString(department));
        // JSON 转 Java 对象
        System.out.println(objectMapper.readValue("{\"id\":1,\"name\":\"开发部
\",\"sn\":\"DEV\"}", Department.class));
    }
}
```


4、Fastjson

4.1、添加依赖

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.47</version>
</dependency>
```

4.2、API 使用

```
package cn.wolfcode.json;
```

```
public class JsonTest {
    @Test
    public void testFastjson() {
        Department department = new Department();
        department.setId(1L);
        department.setName("开发部");
        department.setSn("DEV");

        // Java 对象转 JSON
        System.out.println(JSON.toJSONString(department));
        // JSON 转 Java 对象
        System.out.println(JSON.parseObject("{\"id\":1,\"name\":\"开发部
\",\"sn\":\"DEV\"}", Department.class));
    }
}
```

四、Spring MVC 响应 JSON

1、使用 Servlet API 响应 JSON

比如响应类型这样的 JSON 数据 {"success":true,"msg":"2020-03-16 10:00"}。

1.1、新建 JsonResult 类

```
package cn.wolfcode.qo;

@Setter
@Getter
public class JsonResult {
    private boolean success;
    private String msg;
}
```

1.2、新建 JsonController 类

```
package cn.wolfcode.web.controller;

@Controller
public class JsonController {

    // 若形参有一个类型 response，方法返回值可以为 void
    @RequestMapping("/getTime")
    public void getTime(HttpServletResponse response) throws Exception {
        // 响应 HTML 格式内容回浏览器
        /*
        response.setContentType("text/html;charset=utf-8");
        PrintWriter writer = response.getWriter();
        writer.write("HTML 格式内容");
        writer.flush();*/

        // 响应 JSON 格式内容回浏览器
        response.setContentType("application/json;charset=utf-8");
        PrintWriter writer = response.getWriter();
```

```
        Date now = new Date();
        JsonResult jsonResult = new JsonResult();
        jsonResult.setSuccess(true);
        jsonResult.setMsg(now.toLocaleString());
        writer.write(JSON.toJSONString(jsonResult)); // 避免自己手动拼接字符串

        writer.flush();
    }
}
```

2、Spring MVC 响应 JSON 步骤

- 在 pom.xml 中添加 Jackson 依赖。
- 在 mvc.xml 配置 MVC 注解解析器。
- 定义一个类，里面提供对应属性封装数据。
- 在要响应 JSON 数据的控制器的处理方法上贴 @ResponseBody 注解，且方法返回类型为上面定义的类。
- 在处理方法中创建上面定义类 的对象，封装数据返回即可。

3、练习

3.1、练习一

响应类型这样的 JSON 数据 {"success":true,"msg":"2020-03-16 10:00"}。

在 JsonController 类追加一个处理方法，如下：

```
package cn.wolfcode.web.controller;

@Controller
public class JsonController {

    @RequestMapping("/getTime")
    @ResponseBody
    public JsonResult getTime() {
        Date now = new Date();
        JsonResult jsonResult = new JsonResult();
        jsonResult.setSuccess(true);
        jsonResult.setMsg(now.toLocaleString());
        return jsonResult;
    }
}
```


3.2、练习二

响应类型多个部门列表的 JSON 数据，类似这样 [{"id":1,"name":"开发部","sn":"DEV"},{"id":2,"name":"人事部","sn":"HR"}]。

在 JsonController 类追加一个处理方法，如下：

```
package cn.wolfcode.web.controller;

@Controller
public class JsonController {
```

```
    @Autowired
    private IDepartmentService departmentService;

    @RequestMapping("/depts")
    @ResponseBody
    public List<Department> getDepts(){
        return departmentService.listAll();
    }
}
```

五、AJAX 概述

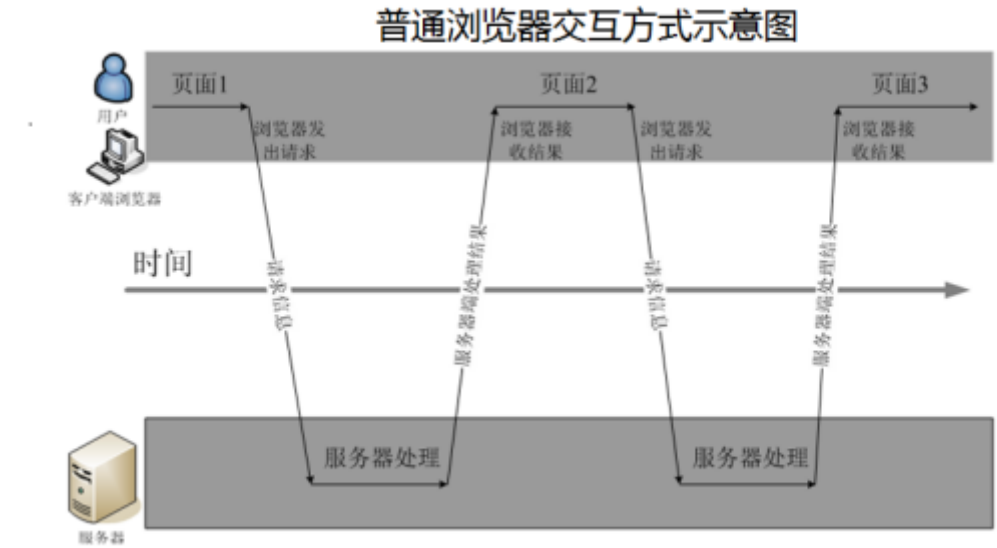
1、AJAX介绍

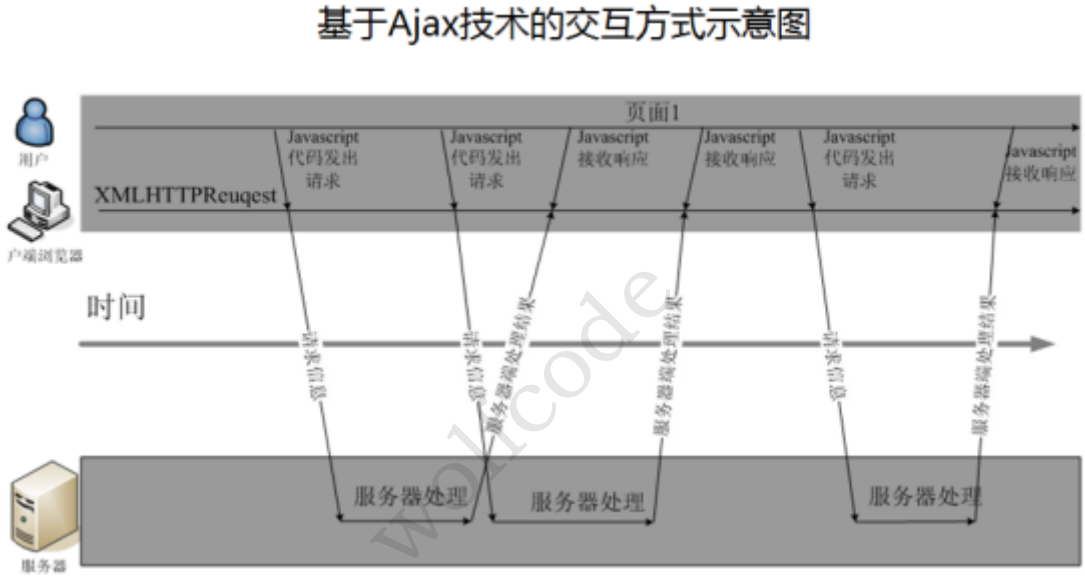
- AJAX 不是一项具体的技术，而是几门技术的综合应用。Javascript、XHTML和CSS、DOM、XML和XMLHttpRequest。
- AJAX 核心只不过是要在 Javascript 中调用一个叫 XMLHttpRequest 类，这个类可以与 Web 服务器使用 HTTP 协议进行交互，程序不通过浏览器发出请求，而是用这个特殊的 JavaScript 对象发送请求和接收响应。
- XMLHttpRequest 对象在网络上的俗称为 AJAX 对象。

2、AJAX 特点

浏览器中显示一个页面后，这个页面以后一直不改变，所有的操作请求都由这个网页中的 Javascript 代码发出，所有的结果都由 Javascript 代码接受并增加到这个页面上，浏览器窗口中显示的网页始终都是初始的那个网页。
增强用户体验：可以在用户浏览网页的同时与服务器进行**异步**交互和实现网页内容的**局部更新**。

同步和异步交互：
同步：提交请求 ---> 等待服务器处理 ---> 处理完毕返回 这个期间客户端浏览器不能干任何事。同步是指：发送方发出数据后，等接收方发回响应以后才发下一个数据包的通 讯方式。
异步：请求通过事件触发 ---> 服务器处理（这时浏览器仍然可以作其他事情）---> 处理完毕。异步是指：发送方发出数据后，不等接收方发回响应，接着发送下个数据包的通 讯方式。





3、AJAX缺陷

- AJAX 大量使用了 Javascript 和 AJAX 引擎，而这个取决于浏览器的支持。IE5.0 及以上、Mozilla1.0、NetScape7 及以上版本才支持，Mozilla 虽然也支持 AJAX，但是提供 XMLHttpRequest 的方式不一样。所以，使用 AJAX 的程序必须测试针对各个浏览器的兼容性。
- AJAX 更新页面内容的时候并没有刷新整个页面，因此，网页的后退功能是失效的；有的用户还经常搞不清楚现在的数据是旧的还是已经更新过的。这个就需要在明显位置提醒用户“数据已更新”。
- 对流媒体的支持没有 Flash、Java Applet 好。
- AJAX 不支持跨域访问，更多参见。

六、AJAX的简单入门-获取服务端的时间（不讲）

1、项目准备

使用之前的集成的项目

2、思路

- 编写页面，页面有个按钮；
- 给按钮绑定一个点击事件处理函数；
- 触发点击事件发送 AJAX 请求到控制器；
- 控制器响应时间给客户端 JSON 数据，标准格式，取值方便；
- 客户端接收响应，显示时间。

3、代码实现

- 创建 AJAX 对象（发送请求和接收响应）；
- 给 AJAX 对象设置 HTTP 请求方式，URL 和是否异步；
- 给 AJAX 对象设置状态监听函数（回调函数），当 AJAX 对象的 readyState 状态改变会执行此函数；
- 发送请求。

```
<script>
    window.onload = function () {
        document.querySelector('#getTime').onclick = function () {
            // 发送请求
            // 创建 XMLHttpRequest 对象
            var ajax = new XMLHttpRequest();
            // 设置请求路径 ,第一个参数是请求方式，第二个参数请求路径，第三个是否异步
            ajax.open('get', '/getTime.do', true);
            // 设置监听服务响应函数
            ajax.onreadystatechange = function () {
                console.log(1);
                var state = ajax.readyState;
                var status = ajax.status;
                if(state == 4 && status == 200){ // 服务成功响应
                    // 获取响应数据
                    var data = ajax.responseText;
                    console.log(data);
                    var obj = JSON.parse(data);
                    console.log(obj);
                    document.querySelector('#time').innerHTML = obj.msg;
                }
            };
            // 发送请求
            ajax.send();
        }
    }
</script>
</head>
<body>
<span id="time" style="color: green"></span>
<button id="getTime">获取系统时间</button>
```

```
@RequestMapping("/getTime")
public void getTime(HttpServletResponse response) throws IOException {
    // 响应数据类型是 JSON
    response.setContentType("application/json;charset=utf-8");
    PrintWriter writer = response.getWriter();
    Date now = new Date();
    String jsonStr = "{\"msg\" : \"" + now.toLocaleString() + "\"}";
    writer.write(jsonStr);
    writer.flush();
}
```


七、jQuery 中 AJAX API

1、jQuery.ajax([options])

async Boolean

(默认: true) 默认设置下，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false。注意，同步请求将锁住浏览器，用户其它操作必须等待请求完成才可以执行。

contentType String

(默认: "application/x-www-form-urlencoded") 发送信息至服务器时内容编码类型。默认值适合大多数情况。如果你明确地传递了一个content-type给 $.ajax() 那么他必定会发送给服务器（即使没有数据要发送）

data Object,String

发送到服务器的数据。将自动转换为请求字符串格式。GET 请求中将附加在 URL 后。查看 processData 选项说明以禁止此自动转换。必须为 Key/Value 格式。如果为数组，jQuery 将自动为不同值对应同一个名称。如 {foo:["bar1", "bar2"]} 转换为 '&foo=bar1&foo=bar2'。

dataType String

预期服务器返回的数据类型。如果不指定，jQuery 将自动根据 HTTP 包 MIME 信息来智能判断，比如XML MIME类型就被识别为XML。在1.4中，JSON就会生成一个JavaScript对象，而script则会执行这个脚本。随后服务器端返回的数据会根据这个值解析后，传递给回调函数。可用值:

"xml": 返回 XML 文档，可用 jQuery 处理。

"html": 返回纯文本 HTML 信息；包含的script标签会在插入dom时执行。

"script": 返回纯文本 JavaScript 代码。不会自动缓存结果。除非设置了"cache"参数。'''注意：'''在远程请求时(不在同一个域下)，所有POST请求都将转为GET请求。(因为将使用DOM的script标签来加载)

"json": 返回 JSON 数据 。

"jsonp": JSONP 格式。使用 JSONP 形式调用函数时，如 "myurl?callback=?" jQuery 将自动替换 ? 为正确的函数名，以执行回调函数。

"text": 返回纯文本字符串

success Function

请求成功后的回调函数。参数：由服务器返回，并根据dataType参数进行处理后的数据；描述状态的字符串。 Ajax 事件。

```
function (data, textStatus) {
    // data 可能是 xmlDoc, jsonObj, html, text, 等等...
    this; // 调用本次AJAX请求时传递的options参数
}
```


type String

(默认: "GET") 请求方式 ("POST" 或 "GET")， 默认为 "GET"。注意：其它 HTTP 请求方法，如 PUT 和 DELETE 也可以使用，但仅部分浏览器支持。

url String

(默认: 当前页地址) 发送请求的地址。

2、jQuery.get(url, [data], [callback], [type])

jQuery.get(url, *[data]*, *[callback]*, *[type]*) 返回值:XMLHttpRequest

概述

通过远程 HTTP GET 请求载入信息。

这是一个简单的 GET 请求功能以取代复杂 $.ajax 。请求成功时可调用回调函数。如果需要在出错时执行函数，请使用 $.ajax。

参数

url	String
待载入页面的URL地址	
data *(可选)*	Map
待发送 Key/value 参数。	
callback *(可选)*	Function
载入成功时回调函数。	
type *(可选)*	String
返回内容格式，xml, html, script, json, text, _default。	

3、jQuery.post(url, [data], [callback], [type])

jQuery.post(url, *[data]*, *[callback]*, *[type]*) 返回值:XMLHttpRequest

概述

通过远程 HTTP POST 请求载入信息。

这是一个简单的 POST 请求功能以取代复杂 $.ajax 。请求成功时可调用回调函数。如果需要在出错时执行函数，请使用 $.ajax。

参数

url	String
发送请求地址。	
data *(可选)*	Map
待发送 Key/value 参数。	
callback *(可选)*	Function
发送成功时回调函数。	
type *(可选)*	String
返回内容格式，xml, html, script, json, text, _default。	

八、练习 - GET 请求检查用户名是否存在

1、前端 JS 代码

新建 webapp/jq_02/02.check_username.html，使用 jQuery 发送 AJAX 请求。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="/static/jQuery-1.11/jquery-1.11.3.min.js"></script>
    <script>
```

```
        $(function () {
            $('#username').blur(function () {
                // console.log(this); // this 事件源
                var username = $(this).val(); // 获取用户输入的用户名
                // 拼接参数字符串
                var paramString = 'username=' + username;
                $.get('/checkUsername.do', paramString, function (data) {
                    $('#result').html(data.msg)
                        .css('color', data.success ? 'green' : 'red');
                })
            });
        });
    </script>
</head>
<body>
    <span id="result"></span><br/>
    用户名:<input type="text" name="username" id="username">
</body>
</html>
```

2、后端 Java 代码

在 JsonController 类追加一个处理方法，使用 Spring MVC 响应 JSON 字符串。

```
package cn.wolfcode.web.controller;

@Controller
public class JsonController {

    // 若存在 {"success":false,"msg":"用户名已注册"}
    // 若不存在 {"success":true,"msg":"恭喜您可以入坑"}
    @RequestMapping("/checkUsername")
    @ResponseBody
    public JsonResult checkUsername(String username){
        // 数据库查询，模拟一下 lony
        JsonResult jsonResult = new JsonResult();
        if("lony".equals(username)){
            jsonResult.setSuccess(false);
            jsonResult.setMsg("用户名已注册");
        }else {
            jsonResult.setSuccess(true);
            jsonResult.setMsg("恭喜您可以入坑");
        }
        return jsonResult;
    }
}
```


九、练习 - POST 请求用户登录操作

1、前端 JS 代码

新建 webapp/jq_02/03.login.html，使用 jQuery 发送 AJAX 请求。

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>登录</title>
    <script src="/static/jQuery-1.11/jquery-1.11.3.min.js"></script>
    <script>
        $(function () {
            $('#login').click(function () {
                // 发送 AJAX
                // 获取到用户输入的用户名和密码
                var $usernameInput = $('#username');
                var $passwordInput =$('#password');

                var u = $usernameInput.val();
                var p = $passwordInput.val();

                var param = {username : u, password : p}; // 构成 JS 对象，作为下面
请求的参数

                console.log(param);
                $.post('/loginJson.do', param, function (data) {
                    console.log(data);
                    if(data.success){ // 登录成功
                        // 通过 JS 代码发送发送求
                        // 修改地址栏的地址，并发送请求到 http://www.baidu.com
                        location.href = 'http://www.baidu.com';
                    }else{ // 登录失败
                        $('#result').html(data.msg)
                            .css('color', 'red');
                    }
                });
            });
        });
    </script>
</head>
<body>
    <span id="result"></span><br/>
    用户名:<input type="text" name="username" id="username">
    密码:<input type="text" name="password" id="password">
    <button id="login">登录</button>
</body>
</html>
```


2、后端 Java 代码

在 JsonController 类追加一个处理方法，使用 Spring MVC 响应 JSON 字符串。

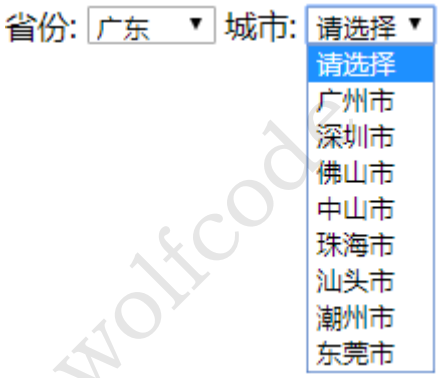
```
package cn.wolfcode.web.controller;

@Controller
public class JsonController {

    // 若登录成功 {"success":true,"msg":"登录成功"}
    // 若登录失败 {"success":false,"msg":"登录失败 "}
    @RequestMapping("/loginJson")
    @ResponseBody
```

```
    public JsonResult checkUsername(String username, String password){
        // 模拟，假设数据存只存在用户 lony 密码 123
        JsonResult jsonResult = new JsonResult();
        if("lony".equals(username) && "123".equals(password)){
            jsonResult.setSuccess(true);
            jsonResult.setMsg("登录成功");
        }else {
            jsonResult.setSuccess(false);
            jsonResult.setMsg("登录失败");
        }
        return jsonResult;
    }
}
```

十、练习 - 二级联动

1、二级联动实际应用

比如用户注册填写地址信息的会使用二级联动。购物时，选择所在地区也会使用二级联动。



而三级联动只多一个级而已，做法也是一样的。

2、二级联动的实现思路

- 页面加载完，省份下拉框从后台获取省份数据；
- 将后台获取的响应数据，渲染到省份下拉框中；
- 给省份下框绑定值改变的事件，值发生改变之后，把选择的省份 id 传给后台；
- 将后台获取的响应数据，渲染到城市下拉框中。

3、代码实现

3.1、编写页面

新建 webapp/jq_02/04.province_city.html，引入好 jQuery，提供省份和城市下拉框。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>二级联动</title>
    <script src="/static/jQuery-1.11/jquery-1.11.3.min.js"></script>
</head>
<body>
```

```
省份:<select id="p">
    <option value="-1">请选择</option>
</select>
城市:<select id="c">
    <option value="-1">请选择</option>
</select>
</body>
</html>
```

3.2、编写 JS 代码

使用 jQuery 发送 AJAX 请求获取省份和城市数据，注意发送时机。在上面引入 jQuery 的后面追加如下代码：

```
<script>
    $(function () {
        var $p = $('#p');
        var $c = $('#c');

        // 页面加载完，发送 AJAX 请求获取省份数据
        $.get('/provinces.do', function (data) {
            // 遍历省份数组
            data.forEach(function (value) {
                $p.append('<option value="' + value.id + '">' + value.name +
'</option>');
            });
        });

        // 给省份下拉框绑定值改变事件处理函数，当省份下拉框选项改变了，就发送请求获取这个省份
对应城市数据，拿到数据再使用 DOM 显示城市下拉框中
        $p.change(function () {

            var pid = $(this).val(); // 获取被选中省份下拉框的 option 的 value 属性
值，即省份 id 值

            // 清楚旧有子元素
            $c.empty();
            $c.append('<option value="-1">请选择</option>');

            if(pid >= 1) {
                $.get('/cities.do', 'pid=' + pid, function (data) {
                    // 遍历城市数组
                    data.forEach(function (value) {
                        console.log(value);
                        $c.append('<option value="' + value.id + '">' +
value.name + '</option>');
                    });
                });
            }
        });
    });
</script>
```

3.3、编写后台获取省份数据和城市数据

拷贝准备的 Province 类 和 City 类，用来封装数据转 JSON 使用，避免自己手动拼接 JSON。

```java
package cn.wolfcode.domain;

// 省份
public class Province {

    private Long id;
    private String name;

    public Province() {
    }

    public Province(Long id, String name) {
        this.id = id;
        this.name = name;
    }

    public Long getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    // 获取所有的省份
    public static List<Province> getAllProvince() {
        List<Province> provinces = new ArrayList<Province>();
        provinces.add(new Province(1L, "湖南"));
        provinces.add(new Province(2L, "广东"));
        provinces.add(new Province(3L, "湖北"));
        return provinces;
    }
    public String toString() {
        return "Province [id=" + id + ", name=" + name + "]";
    }

}
```

```java
package cn.wolfcode.domain;

// 城市
public class City {
    private Long id;
    private String name;

    public City() {
    }

    public City(Long id, String name) {
        this.id = id;
        this.name = name;
    }

    /**
     * 根据省份 id 查询省份中的城市!
```

```
     * @return
     */
    public static List<City> getCityByProvinceId(Long pid) {

        List<City> citys = new ArrayList<City>();

        if (pid == 1) {
            citys = Arrays.asList(
                    new City(11L,"长沙市"),
                    new City(12L,"株洲市"),
                    new City(13L,"湘潭市"),
                    new City(14L,"衡阳市"),
                    new City(15L,"邵阳市"),
                    new City(16L,"岳阳市"),
                    new City(17L,"常德市"),
                    new City(18L,"张家界市")
            );
        } else if (pid == 2) {
            citys = Arrays.asList(
                    new City(21L,"广州市"),
                    new City(22L,"深圳市"),
                    new City(23L,"佛山市"),
                    new City(24L,"中山市"),
                    new City(25L,"珠海市"),
                    new City(26L,"汕头市"),
                    new City(27L,"潮州市"),
                    new City(28L,"东莞市")
            );
        } else if (pid == 3) {
            citys = Arrays.asList(
                    new City(31L,"孝感市"),
                    new City(32L,"黄冈市"),
                    new City(33L,"咸宁市"),
                    new City(34L,"恩施州"),
                    new City(35L,"鄂州市"),
                    new City(36L,"武汉市"),
                    new City(37L,"荆门市"),
                    new City(38L,"襄阳市")
            );
        }
        return citys;
    }

    public Long getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return "City [id=" + id + ", name=" + name + "]";
    }
}
```


在 JsonController 类追加两个个处理方法，使用 Spring MVC 响应省份和城市数据，是 JSON 格式。

```java
package cn.wolfcode.web.controller;

@Controller
public class JsonController {

    // 获取省份 JSON 数据
    @RequestMapping("/provinces")
    @ResponseBody
    public List<Province> getPovinces(){
        List<Province> provinces = Province.getAllProvince();
        return provinces;
    }

    // 获取对应省份的城市 JSON 数据
    @RequestMapping("/cities")
    @ResponseBody
    public List<City> getPovinces(Long pid){
        return City.getCityByProvinceId(pid);
    }
}
```