# 跨域问题实现

## 用户页面通用跳转

### 业务需求

当用户点击登陆页面或者注册页面时.要跳转到指定的页面中去.

1. /user/register.html
2. /user/login.html

使用RestFul结构实现页面通用跳转

### 编辑Controller

@Controller

@RequestMapping("/user")

**public** **class** UserController {

@RequestMapping("/{moduleName}")

**public** String index(@PathVariable String moduleName){

**return** moduleName;

}

}

## 跨域和同域的区别

### 跨域测试

<script type="text/javascript">

$(**function**(){

$.get("http://manage.jt.com/test.json",**function**(data){

alert(data.name);

})

})

</script>

### 同源策略

说明:如果请求的**协议://域名:端口**都相同则是同源访问.可以访问数据.

除此之外的全部的访问都是跨域的,.禁止通信.



### 跨域实现

说明:利用script标签中src属性可以实现跨域.

步骤:

1. 利用src属性请求远程的js
2. 将返回值的JSON串进行特殊的处理.为返回值添加函数名称 hello(data)格式
3. 在客户端定义回调函数 function hello(data){…..}

页面编辑:

<script type="text/javascript">

/\*声明了一个函数 该函数需要被别人调用 \*/

**function** hello(data){

alert(data.name);

}

</script>

<script type="text/javascript" src="http://manage.jt.com/test.json"></script>

<script type="text/javascript" src="http://manage.jt.com/js/jquery-easyui-1.4.1/jquery.min.js"></script>

定义返回值类型:

hello({"id":"1","name":"tom"})

### 传统的JS中存在的问题

1. 如果函数的名称和JSON串的返回值不同则无法跨域.
2. 每次跨域请求都需要借用javaScript中的src属性.调用不直观.而且繁琐.

如何解决:

1. 发起请求时将函数名称当成参数传递即可

addUser?callback=hello.

后台接收callback参数最终封装为特殊格式的JSON串.hello(Data)

1. 将调用的过程抽取为**公共的方法**.以AJAX的形式进行封装.并且提供良好品参数传值的方式.

Jquery中的JSONP请求格式.

### JSONP介绍

JSONP(JSON with Padding)是[JSON](https://baike.baidu.com/item/JSON" \t "_blank)的一种“**使用模式**”，**可用于解决主流浏览器的跨域数据访问的问题**。由于同源策略，一般来说位于 server1.example.com 的网页无法与不是 server1.example.com的服务器沟通，而 HTML 的<script> 元素是一个例外。利用 <script> 元素的这个开放策略，网页可以得到从其他来源动态产生的 JSON 资料，而这种使用模式就是所谓的 JSONP。用 JSONP 抓到的资料并不是 JSON，而是任意的JavaScript，用 JavaScript 直译器执行而不是用 JSON 解析器解析。

总结:script利用src属性实现跨域请求就是JSONP.

## jQuery中JSONP使用

### 编辑页面JS

$.ajax({

url:"http://manage.jt.com/web/testJSONP",

type:"get",

dataType:"jsonp", //返回值的类型

//jsonp: "callback", //指定参数名称

//jsonpCallback: "hello", //指定回调函数名称

success:**function** (data){

alert(data.id);

//转化为字符串使用

//var obj = eval("("+data+")");

//alert(obj.name);

}

});

### 编辑服务端Controller

@Controller

@RequestMapping("/web")

**public** **class** WebJSONPController {

//http://manage.jt.com/web/testJSONP?callback=jQuery111102185225838437006\_1537844552339&\_=1537844552340

//@RequestMapping(value="/testJSONP",produces="text/html;charset=utf-8")

//@ResponseBody

**public** String jsonp(String callback) **throws** JsonProcessingException{

User user = **new** User();

user.setId(100);

user.setName("tomcat猫");

ObjectMapper objectMapper = **new** ObjectMapper();

String userJSON = objectMapper.writeValueAsString(user);

//因为返回值必须添加回调函数,否则无法解析. hello(data)

**return** callback+"(" + userJSON + ")";

}

//利用Spring中的JSONP返回

@RequestMapping(value="/testJSONP")

@ResponseBody

**public** MappingJacksonValue jsonpSuper(String callback){

User user = **new** User();

user.setId(100);

user.setName("tomcat猫");

MappingJacksonValue value = **new** MappingJacksonValue(user);

value.setJsonpFunction(callback);

**return** value;

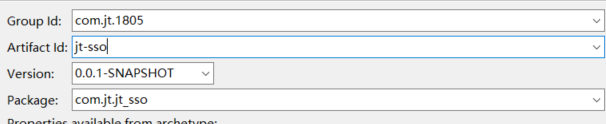
}

}

# 构建jt-sso项目

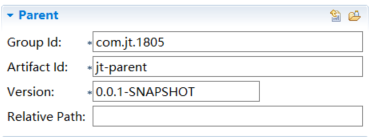
## 构建项目

### 创建项目

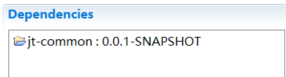


### 添加继承和依赖

1.添加继承



2.添加依赖



### 添加tomcat插件

1. 添加tomcat插件

<build>

<plugins>

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<port>8093</port>

<!--项目的发布路径 -->

<path>/</path>

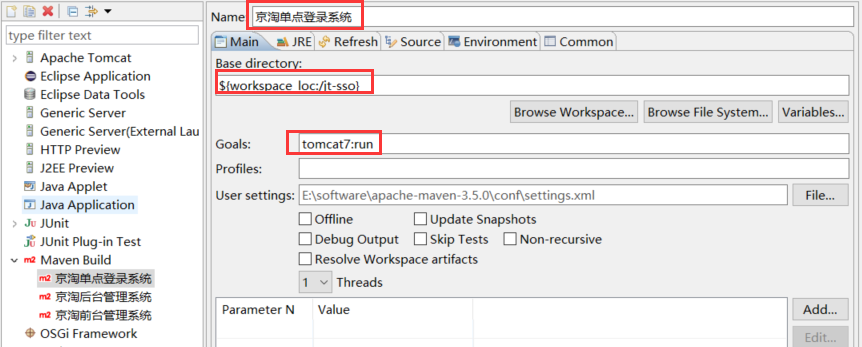
</configuration>

</plugin>

</plugins>

</build>

2.添加tomcat插件



### 修改Nginx配置文件

#单点登录管理系统

server {

listen 80;

server\_name sso.jt.com;

location / {

#实现服务器代理

proxy\_pass http://localhost:8093;

proxy\_connect\_timeout 3;

proxy\_read\_timeout 3;

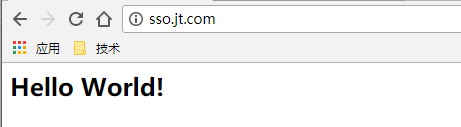
proxy\_send\_timeout 3;

}

}

修改Hosts文件 127.0.0.1 sso.jt.com

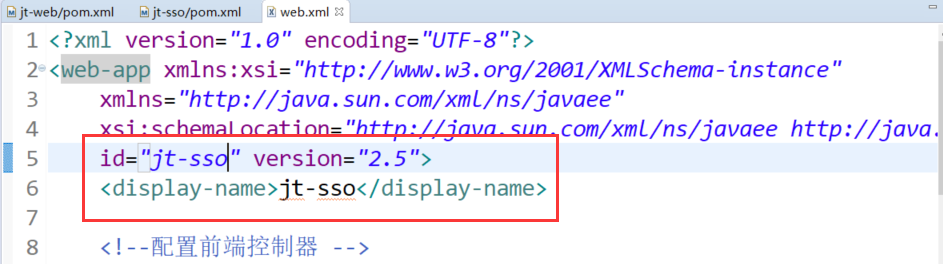
### 效果测试



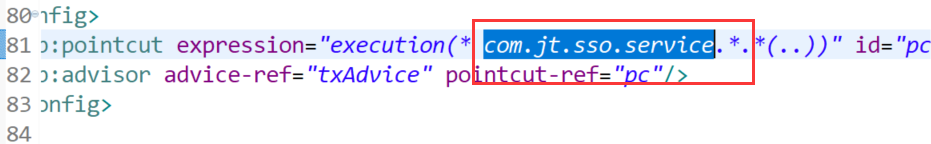
## 添加配置文件

### 导入web.xml

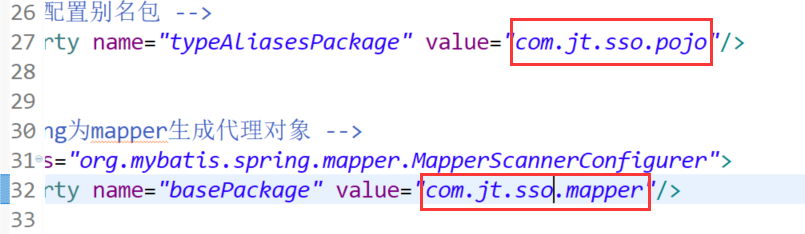
说明:将jt-manage后台中的web.xml导入sso中,因为前天的web.xml中拦截路径.html结尾的请求



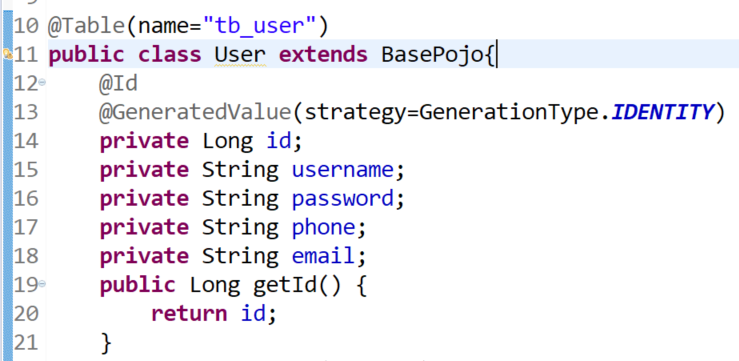
### 修改Spring配置文件



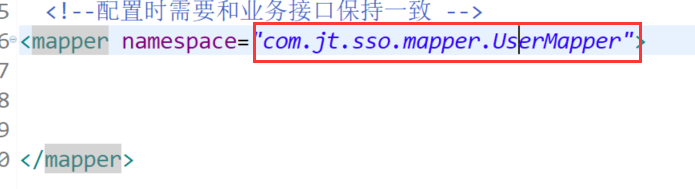
### 修改Mybatis配置文件



### 创建POJO对象

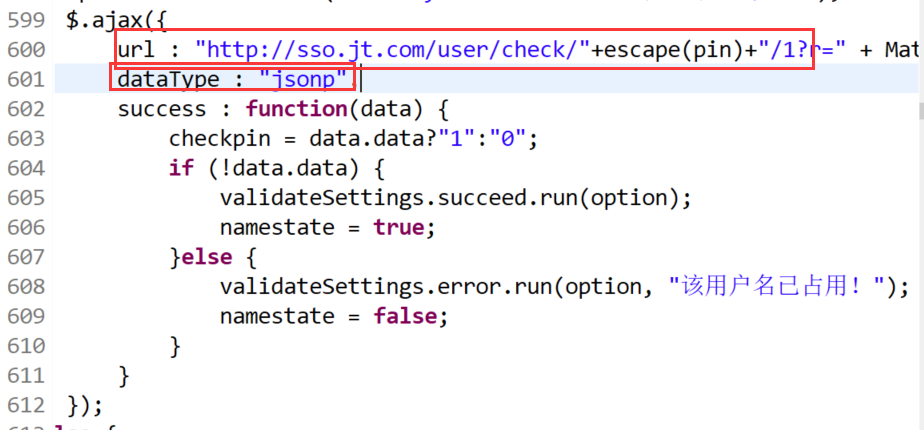


### 修改mapper映射文件的路径



## 用户信息校验

### 分析页面JS



### 接口文档规范

|  |  |
| --- | --- |
| 请求方法 | GET |
| URL | http://sso.jt.com/user/check/{param}/{type} |
| 参数 | 格式如：chenchen/1  其中chenchen是校验的数据  Type为类型，可选参数1 username、2 phone、3 email |
| 示例 | http://sso.jt.com/user/check/chenchen/1 |
| 返回值 | {  status: 200 //200 成功，201 没有查到  msg: “OK” //返回信息消息  data: false //返回数据true用户已存在，false用户不存在，可以  } |
|  |  |

### 编辑后台Controller

@Controller

@RequestMapping("/user")

**public** **class** UserController {

@Autowired

**private** UserService userService;

//JSONP格式返回 param 表示校验参数 type表示校验类型

@RequestMapping("/check/{param}/{type}")

@ResponseBody

**public** MappingJacksonValue checkUser(

@PathVariable String param,

@PathVariable **int** type,

String callback){

//校验用户名信息是否存在

**boolean** flag = userService.findCheckUser(param,type);

MappingJacksonValue jacksonValue =

**new** MappingJacksonValue(SysResult.*oK*(flag));

jacksonValue.setJsonpFunction(callback);

**return** jacksonValue;

}

}

### 编辑Service

/\*\*

\* 思考:

\* admin/1 表示校验用户名为admin的数据

\* sql: select count(\*) from tb\_user where username = "admin"

\* Type为类型，可选参数1 username、2 phone、3 email

\*/

@Override

**public** **boolean** findCheckUser(String param, **int** type) {

String cloumn = **null**;

**switch** (type) {

**case** 1:

cloumn = "username"; **break**;

**case** 2:

cloumn = "phone"; **break**;

**case** 3:

cloumn = "email"; **break**;

}

**int** count = userMapper.findCheckUser(param,cloumn);

//如果返回true表示用户已经存在

**return** count == 0 ? **false** : **true**;

}

### 编辑Mapper接口/映射文件

**int** findCheckUser(@Param("param")String param,

@Param("cloumn")String cloumn);

编辑映射文件

<mapper namespace="com.jt.sso.mapper.UserMapper">

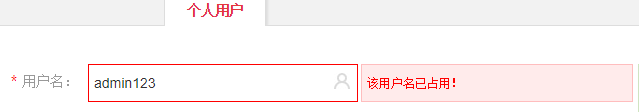
<select id="findCheckUser" resultType="int">

select count(\*) from tb\_user where ${cloumn} = #{param}

</select>

</mapper>

### 页面效果



## HttpClient学习

### HttpClient介绍

HttpClient 是Apache Jakarta Common 下的子项目，**可以用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包**，并且它支持 HTTP 协议最新的版本和建议。

HTTP 协议可能是现在 Internet 上使用得最多、最重要的协议了，越来越多的 Java 应用程序需要直接通过 HTTP 协议来访问网络资源。虽然在 JDK 的 java net包中已经提供了访问 HTTP 协议的基本功能，但是对于大部分应用程序来说，JDK 库本身提供的功能还不够丰富和灵活。HttpClient 是 Apache Jakarta Common 下的子项目，用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包，并且它支持 HTTP 协议最新的版本和建议。HttpClient 已经应用在很多的项目中，比如 Apache Jakarta 上很著名的另外两个开源项目 Cactus 和 [HTMLUnit](https://baike.baidu.com/item/HTMLUnit" \t "_blank) 都使用了 HttpClient。现在HttpClient最新版本为 HttpClient 4.5 (GA) （2015-09-11）

总结:HTTPClient是java发起Http请求协议的工具包

### 导入jar包

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpclient</artifactId>

<version>${httpclient.version}</version>

</dependency>

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpmime</artifactId>

<version>4.3.1</version>

</dependency>

### HttpClient入门案例

@Test

**public** **void** test01() **throws** ClientProtocolException, IOException{

//1.获取HttpClient对象

CloseableHttpClient httpClient = HttpClients.*createDefault*();

//2.定义网址

String url = "https://item.jd.com/188550.html";

//3.定义请求方式 GET/POST

HttpGet httpGet = **new** HttpGet(url);

HttpPost httpPost = **new** HttpPost(url);

//4.发送请求

CloseableHttpResponse httpResponse = httpClient.execute(httpPost);

//5.判断返回值数据是否正确

**if**(httpResponse.getStatusLine().getStatusCode() == 200){

System.***out***.println("表示请求有效的");

//获取返回值实体内容

String result = EntityUtils.*toString*(httpResponse.getEntity());

System.***out***.println(result);

}

}

## Spring整合HttpClient

### 编辑配置文件

#从连接池中获取到连接的最长时间

http.request.connectionRequestTimeout=500

#5000

http.request.connectTimeout=5000

#数据传输的最长时间

http.request.socketTimeout=30000

#提交请求前测试连接是否可用

http.request.staleConnectionCheckEnabled=true

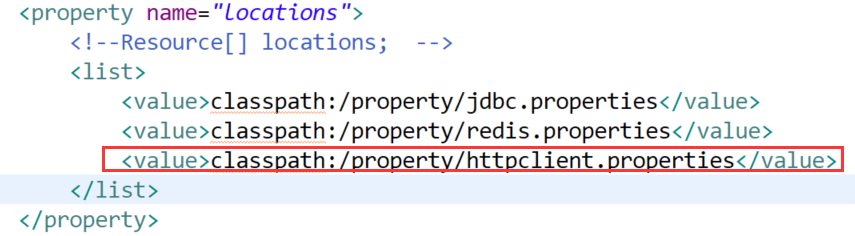
#设置连接总数

http.pool.maxTotal=200

#设置每个地址的并发数

http.pool.defaultMaxPerRoute=100

添加配置文件



### 编辑HttpClient配置文件

<!-- 定义httpclient连接池 -->

<bean id=*"httpClientConnectionManager"* class=*"org.apache.http.impl.conn.PoolingHttpClientConnectionManager"* destroy-method=*"close"*>

<!-- 设置连接总数 -->

<property name=*"maxTotal"* value=*"${http.pool.maxTotal}"*></property>

<!-- 设置每个地址的并发数 -->

<property name=*"defaultMaxPerRoute"* value=*"${http.pool.defaultMaxPerRoute}"*></property>

</bean>

<!-- 定义 HttpClient工厂，这里使用HttpClientBuilder构建-->

<bean id=*"httpClientBuilder"* class=*"org.apache.http.impl.client.HttpClientBuilder"* factory-method=*"create"*>

<property name=*"connectionManager"* ref=*"httpClientConnectionManager"*></property>

</bean>

<!-- 得到httpClient的实例 -->

<bean id=*"httpClient"* factory-bean=*"httpClientBuilder"* factory-method=*"build"*/>

<!-- 定期清理无效的连接 -->

<bean class=*"com.jt.common.util.IdleConnectionEvictor"* destroy-method=*"shutdown"*>

<constructor-arg index=*"0"* ref=*"httpClientConnectionManager"* />

<!-- 间隔一分钟清理一次 -->

<constructor-arg index=*"1"* value=*"60000"* />

</bean>

<!-- 定义requestConfig的工厂 -->

<bean id=*"requestConfigBuilder"* class=*"org.apache.http.client.config.RequestConfig.Builder"*>

<!-- 从连接池中获取到连接的最长时间 -->

<property name=*"connectionRequestTimeout"* value=*"${http.request.connectionRequestTimeout}"*/>

<!-- 创建连接的最长时间 -->

<property name=*"connectTimeout"* value=*"${http.request.connectTimeout}"*/>

<!-- 数据传输的最长时间 -->

<property name=*"socketTimeout"* value=*"${http.request.socketTimeout}"*/>

<!-- 提交请求前测试连接是否可用 -->

<property name=*"staleConnectionCheckEnabled"* value=*"${http.request.staleConnectionCheckEnabled}"*/>

</bean>

<!-- 得到requestConfig实例 -->

<bean id=*"requestConfig"* factory-bean=*"requestConfigBuilder"* factory-method=*"build"* />

### 编辑HttpClient工具API-POST

@Service

**public** **class** HttpClientService {

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(HttpClientService.**class**);

@Autowired(required=**false**)

**private** CloseableHttpClient httpClient;

@Autowired(required=**false**)

**private** RequestConfig requestConfig;

/\*\*

\* 1.实现HttpClientPost方法

\* 思考:

\* 1.需要设定url参数

\* 2.Map<String,String> 使用Map数据结构实现参数封装

\* 3.设定字符集编码 utf-8

\*

\* 难点:POST如何传递参数?????

\* Post请求将参数转化为二进制字节流信息进行数据传输.

\* 一般form表单提交试用POST提交

\* 回顾: get http://addUser?id=1&name=tom

\*/

**public** String doPost(String url,Map<String,String> params,

String charset){

String result = **null**;

//1.判断字符集编码是否为null 如果参数为空则默认为UTF-8

**if**(StringUtils.*isEmpty*(charset)){

charset = "UTF-8";

}

//2.获取请求对象的实体

HttpPost httpPost = **new** HttpPost(url);

httpPost.setConfig(requestConfig);

**try** {

//3.判断用户是否传递参数

**if**(params != **null**){

//将用户传入的数据Map封装到List集合中

List<NameValuePair> parameters = **new** ArrayList<>();

**for** (Map.Entry<String, String> entry: params.entrySet()) {

BasicNameValuePair nameValuePair =

**new** BasicNameValuePair(entry.getKey(), entry.getValue());

parameters.add(nameValuePair);

}

//实现参数封装

UrlEncodedFormEntity formEntity =

**new** UrlEncodedFormEntity(parameters, charset);

//将参数封装为formEntity进行数据传输

httpPost.setEntity(formEntity);

}

//4.发起Post请求

CloseableHttpResponse httpResponse =

httpClient.execute(httpPost);

//5.判断请求结果是否正确

**if**(httpResponse.getStatusLine().getStatusCode() == 200){

//6.获取服务端回传数据

result = EntityUtils.*toString*(httpResponse.getEntity(),charset);

}

} **catch** (Exception e) {

e.printStackTrace();

}

**return** result;

}

**public** String doPost(String url){

**return** doPost(url, **null**, **null**);

}

**public** String doPost(String url,Map<String,String> params){

**return** doPost(url, params, **null**);

}

}

编辑完工具类后将项目打包

## 实现用户注册

### 页面JS



### 查看接口文档

|  |  |
| --- | --- |
|  | POST |
| URL | http://sso.jt.com/user/register |
| 参数 | username 用户名  password 密码  phone 手机号  email 邮箱 |
| 示例 | <http://sso.jt.com/user/register>   |  |  | | --- | --- | | **username：** | chenchen | | **password：** | 123456 | | **phone：** | 13579003045 | | **email：** | chenchen@163.com | |
| 返回值 | {  status: 200 //200 成功，201 没有查到  msg: “OK” //返回信息消息  data:username //返回数据username值  } |
|  |  |

### 编辑前台Controller

//http://www.jt.com/service/user/doRegister

@RequestMapping("/doRegister")

@ResponseBody

**public** SysResult saveUser(User user){

**try** {

userService.saveUser(user);

**return** SysResult.*oK*();

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"新增用户失败");

}

### 编辑前台Service

/\*\*

\* 问题:

\* 因为前台项目只负责数据的展现,不负责数据更新操作

\*

\* 如何将数据传给sso单点登录系统???

\* HTTP协议 GET/POST

\* HTTPClient技术:java代码中发起Http请求的

\*

\*/

@Override

**public** **void** saveUser(User user) {

String url = "http://sso.jt.com/user/register";

Map<String,String> params = **new** HashMap<>();

params.put("username", user.getUsername());

params.put("password", user.getPassword());

params.put("phone", user.getPhone());

params.put("email", user.getEmail());

//前台通过httpClient将数据进行远程传输.如果程序在后台执行错误!!

String result = httpClient.doPost(url, params);

**try** {

//检测返回值结果是否正确

SysResult sysResult = ***objectMapper***.readValue(result, SysResult.**class**);

**if**(sysResult.getStatus() != 200){

//表示程序有错

**throw** **new** RuntimeException();

}

} **catch** (Exception e) {

e.printStackTrace();

**throw** **new** RuntimeException();

}

}

### 编辑后台Controller

//编辑sso后台用户新增业务

@RequestMapping("/register")

@ResponseBody

**public** SysResult saveUser(User user){

**try** {

userService.saveUser(user);

**return** SysResult.*oK*();

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201, "新增失败");

}

### 编辑后台Service

//编辑sso新增业务 1.数据不全

@Override

**public** **void** saveUser(User user) {

String md5Pass = DigestUtils.*md5Hex*(user.getPassword());

user.setPassword(md5Pass);//将密码进行加密

user.setEmail(user.getPhone());//暂时代替,否则入库报错

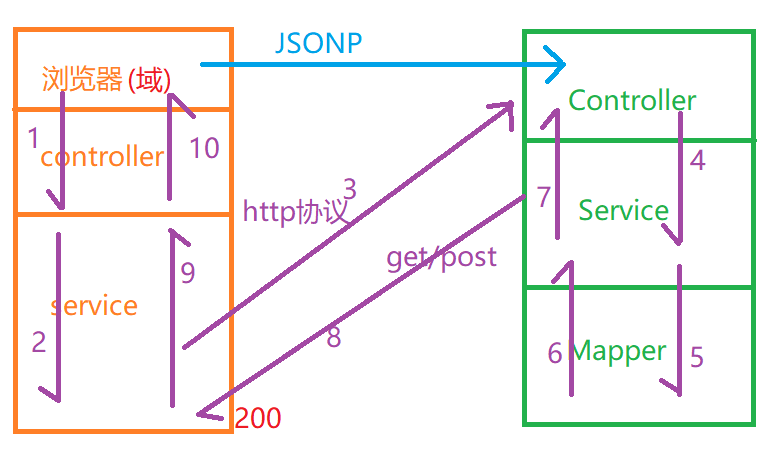
user.setCreated(**new** Date());

user.setUpdated(user.getCreated());

userMapper.insert(user);

}

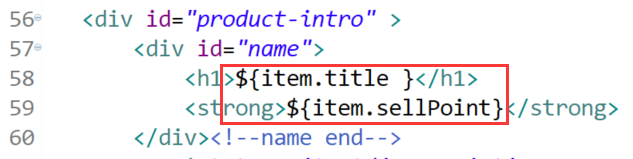
### HttpClient调用流程



## 实现商品信息回显

### 页面分析

说明:根据el表达式获取页面数据



### 编辑前台Controller

@Controller

@RequestMapping("/items")

**public** **class** ItemController {

@Autowired

**private** ItemService itemService;

@RequestMapping("/{itemId}")

**public** String findItemById(@PathVariable Long itemId,Model model){

//获取远程商品信息

Item item = itemService.findItemById(itemId);

model.addAttribute("item", item);

**return** "item";

}

} @Controller

@RequestMapping("/items")

**public** **class** ItemController {

@Autowired

**private** ItemService itemService;

@RequestMapping("/{itemId}")

**public** String findItemById(@PathVariable Long itemId,Model model){

//获取远程商品信息

Item item = itemService.findItemById(itemId);

model.addAttribute("item", item);

**return** "item";

}

}

### 编辑前台Service

@Service

**public** **class** ItemServiceImpl **implements** ItemService {

@Autowired

**private** HttpClientService httpClient;

**private** **static** **final** ObjectMapper ***objectMapper*** = **new** ObjectMapper();

//前台获取后台的商品信息

@Override

**public** Item findItemById(Long itemId) {

String url = "http://manage.jt.com/web/item/findItemById";

Map<String, String> params = **new** HashMap<String, String>();

params.put("itemId", itemId+"");

Item item = **null**;

//返回的Item的JSON

String resultJSON = httpClient.doPost(url, params);

**try** {

item = ***objectMapper***.readValue(resultJSON, Item.**class**);

} **catch** (Exception e) {

e.printStackTrace();

**throw** **new** RuntimeException();

}

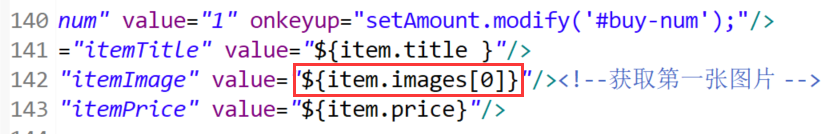
**return** item;

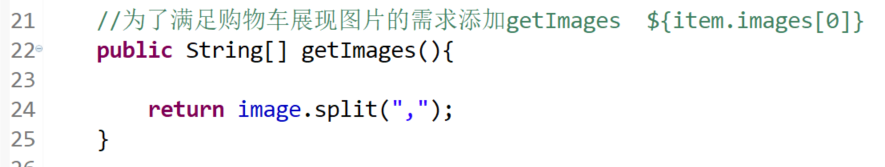
}

}

### 编辑前台POJO对象

说明:为了后期添加购物车实现首张图片回显.添加以下操作





### 编辑后台Controller

@Controller

@RequestMapping("/web/item")

**public** **class** WebItemController {

@Autowired

**private** ItemService itemService;

//编辑manage后台实现商品信息查询

@RequestMapping("/findItemById")

@ResponseBody

**public** Item findItemById(Long itemId){

**return** itemService.findItemById(itemId);

}

}

### 编辑后台Service

@Override

**public** Item findItemById(Long itemId) {

**return** itemMapper.selectByPrimaryKey(itemId);

}

### 页面效果展现

