# 京淘前台实现商品展现

## HttpClient介绍

### 概念

**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就是操作java的http请求协议工具包.

### 加载jar包

<!-- httpclient -->

<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** testGet() **throws** ClientProtocolException, IOException{

//1.定义请求对象

CloseableHttpClient httpClient =

HttpClients.*createDefault*();

//2.定义请求网站

String url = "https://www.baidu.com";

//3.定义请求对象

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

//4.发起请求

CloseableHttpResponse response =

httpClient.execute(get);

//5.判断请求是否正确

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

//6.获取返回值结果

String result =

EntityUtils.*toString*(response.getEntity());

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

}

}

## Spring整合HttpClient

### 编辑pro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

spring导入配置文件

<list>

<value>classpath:/properties/jdbc.properties</value>

<value>classpath:/properties/redis.properties</value>

<value>classpath:/properties/httpclient.properties</value>

</list>

### Spring整合HttpClient

<!-- 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"* />

### 编辑工具类

@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;

/\*\*

\* 工具类开发的步骤:

\* 需求:

\* 根据url发起请求,最终获取响应的结果

\* 设计

\* 参数设计:

\* 1.String url 用户发起请求的地址

\* 2.Map数据类型 map<String,String>

\* 3.String charset 设定字符集编码

\*

\* GET请求参数传递规则:

\* 用户传参: id=1,name=tom,age=18

\* get传参 : url?id=1&name=tom&age=18

\* url = url + "?";

//遍历map集合获取参数

//url?id=1&name=tom&age=18

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

url = url + entry.getKey() + "=" + entry.getValue() + "&";

}

url = url.substring(0,url.length()-1);

\*/

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

String result = **null**;

//1.判断字符集编码是否为空 如果为空则给定默认值utf-8

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

charset = "UTF-8";

}

//2.判断用户是否需要传递参数

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

**try** {

URIBuilder uriBuilder = **new** URIBuilder(url);

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

uriBuilder.addParameter(entry.getKey(), entry.getValue());

}

//url?id=1&name=tom

url = uriBuilder.build().toString();

} **catch** (Exception e) {

e.printStackTrace();

}

}

//3.定义参数提交对象

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

//4.为请求设定超时时间

get.setConfig(requestConfig);

//5.通过httpClient发送请求

**try** {

CloseableHttpResponse response =

httpClient.execute(get);

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

//表示程序调用成功

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

}**else**{

System.***out***.println("调用异常:状态信息:"+response.getStatusLine().getStatusCode());

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

}

} **catch** (Exception e) {

e.printStackTrace();

//异常信息的处理

}

**return** result;

}

**public** String doGet(String url){

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

}

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

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

}

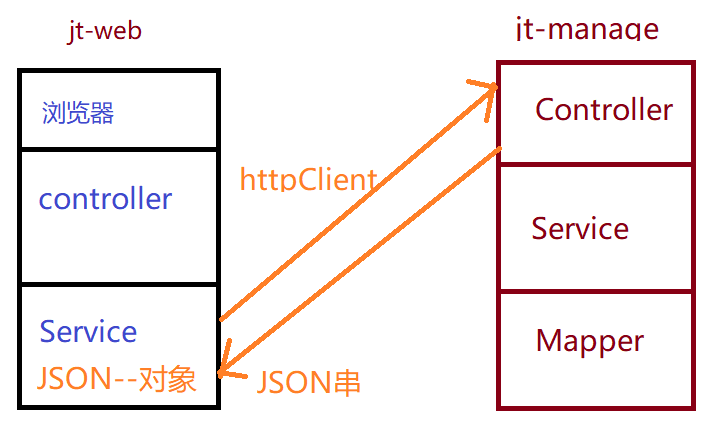
**public** String doGet(String url,String charset){

**return** doGet(url, **null**, charset);

}

}

### httpClient调用过程



### 编辑前台Controller

@Controller

@RequestMapping("/items")

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

@Autowired

**private** ItemService itemService;

@RequestMapping(value="/{itemId}")

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

Item item = itemService.findItemById(itemId);

model.addAttribute("item", item);

**return** "item";

}

}

### 编辑前台Service

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

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

@Autowired

**private** HttpClientService httpClient;

@Override

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

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

//获取后台返回的json数据

String result = httpClient.doGet(url);

Item item = **null**;

**try** {

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

} **catch** (Exception e) {

e.printStackTrace();

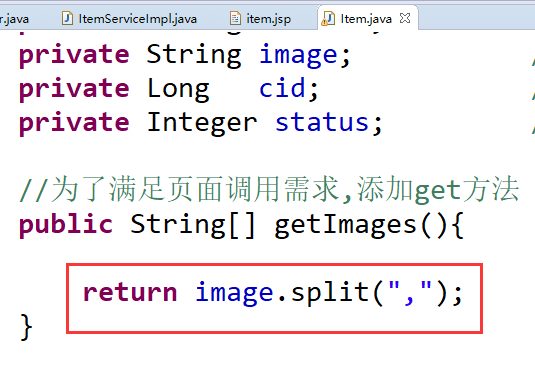
}

**return** item;

}

}

### 编辑ITEM对象



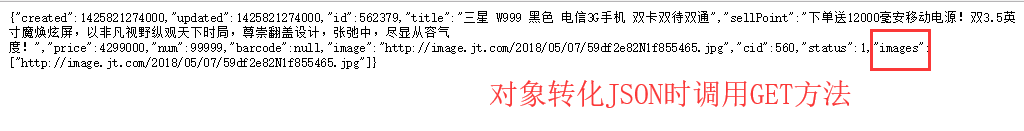
### 关于JSON解析异常

1. 报错

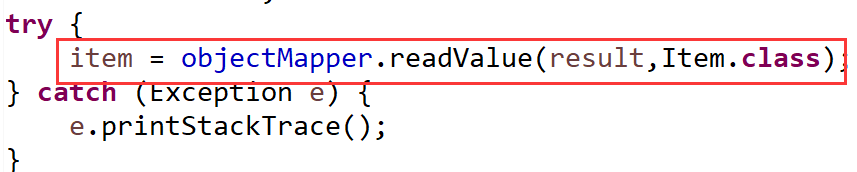
com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException: Unrecognized field "images" (class com.jt.common.po.Item), not marked as ignorable (11 known properties: "barcode", "num", "created", "price", "status", "title", "updated", "image", "id", "sellPoint", "cid"])

at [Source: {"created":1425821274000,"updated":1425821274000,"id":562379,"title":"三星 W999 黑色 电信3G手机 双卡双��双通","sellPoint":"下单送12000毫安移动电源！双3.5英寸魔焕炫屏，以非凡视野纵观天下时局，尊崇翻盖设计，张弛中，尽显从容气度！","price":4299000,"num":99999,"barcode":null,"image":"http://image.jt.com/2018/05/07/59df2e82N1f855465.jpg","cid":560,"status":1,"images":["http://image.jt.com/2018/05/07/59df2e82N1f855465.jpg"]}; line: 1, column: 305] (through reference chain: com.jt.common.po.Item["images"])

1. 后台JSON串

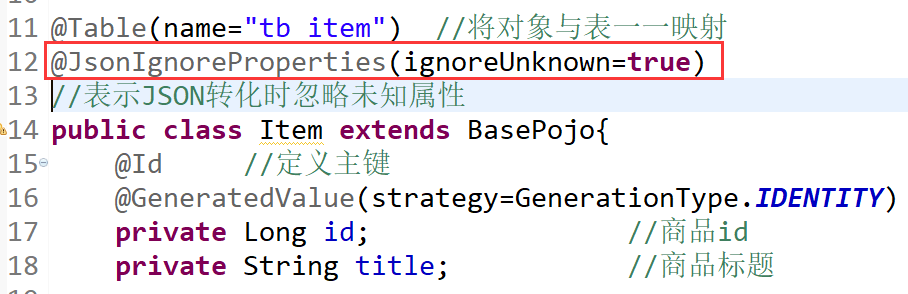


1. 前台数据解析



将JSON串转化为对象性,根据json数据,调用对象的set方法.

1. 解决方案:



### 编辑后台Controller

//修改名称的原因是在spring容器中 id一定不能重复,否则

//容器将不能正常启动 Map<id,Object>

@Controller

@RequestMapping("/web/item")

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

//http://manage.jt.com/web/item/findItemById/"+itemId

@Autowired

**private** ItemService itemService;

@RequestMapping("/findItemById/{itemId}")

@ResponseBody

**public** Item findItemByItemId(@PathVariable Long itemId){

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

}

}

### 编辑后台Service

@Override

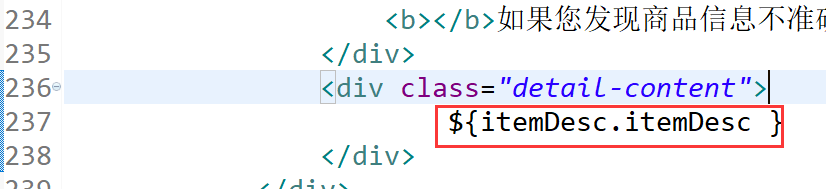
**public** Item findItemByItemId(Long itemId) {

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

}

## 实现商品详情展现

### 页面分析



### 编辑Controller

@RequestMapping(value="/{itemId}")

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

Item item = itemService.findItemById(itemId);

model.addAttribute("item", item);

//实现商品详情展现

ItemDesc itemDesc = itemService.findItemDescById(itemId);

model.addAttribute("itemDesc", itemDesc);

**return** "item";

}

### 编辑前台Service

@Override

**public** ItemDesc findItemDescById(Long itemId) {

String url = "http://manage.jt.com/web/item/findItemDescById/"+itemId;

//itemDesc的JSON串

String result = httpClient.doGet(url);

ItemDesc itemDesc = **null**;

**try** {

itemDesc =

objectMapper.readValue(result, ItemDesc.**class**);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** itemDesc;

}

### 编辑后台Controller

//根据id查询商品详情

@RequestMapping("/findItemDescById/{itemId}")

@ResponseBody

**public** ItemDesc findItemDescById(@PathVariable Long itemId){

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

}

### 编辑后台Service

@Override

**public** ItemDesc findItemDescById(Long itemId) {

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

}

### 页面效果展现

