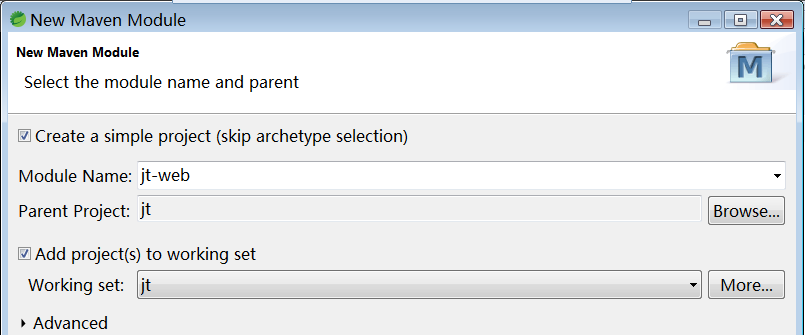
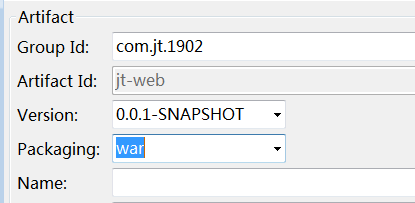
# 京淘前台项目创建

## 构建项目

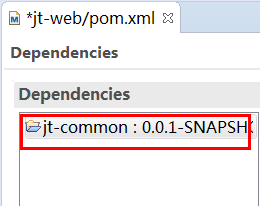
### 创建项目



### 选择打包类型



### 添加依赖



### 编辑pom.xml文件

在pom.xml文件中添加跳过测试类打包插件.目的以后项目打包.测试文件不会出现在jar/war包中

<build>

<plugins>

<!--跳过测试类打包 -->

<plugin>

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

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<skip>true</skip>

</configuration>

</plugin>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

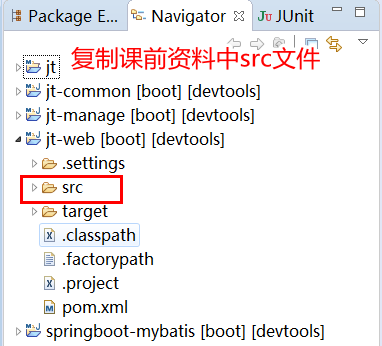
</plugin>

</plugins>

</build>

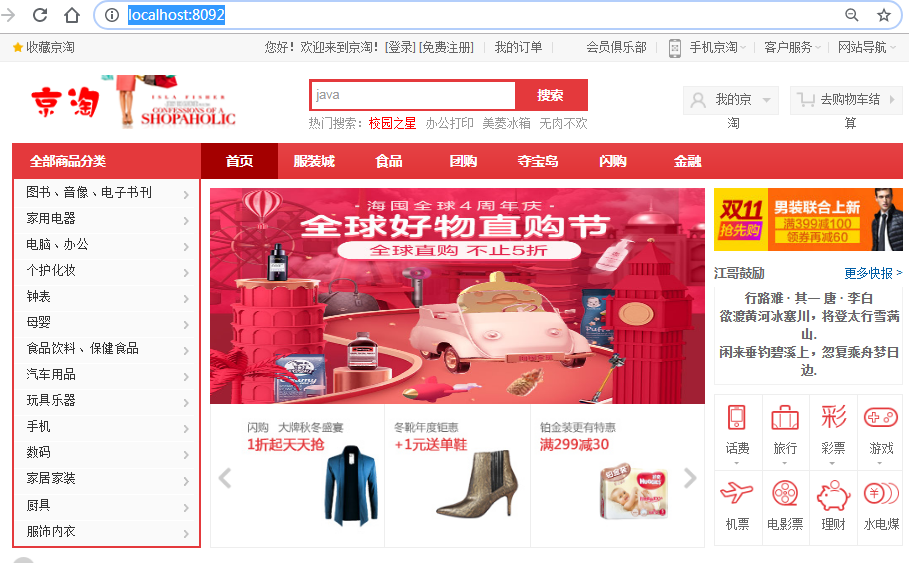
### 导入src文件

说明:将课前资料中的jt-web src文件导入



修改完成后,将redis.properties改为自己的redis链接.执行main方法启动即可.

### 页面启动效果



### 配置反向代理

#前台管理系统

server {

listen 80;

server\_name www.jt.com;

location / {

#代理路径

proxy\_pass http://127.0.0.1:8092;

}

}

配置完成之后,重启nginx服务器

### Http与https区别

https访问更加安全的请求协议规则.

秘钥机制:非对称加密方式. 公钥 私钥

摘要信息:校验信息完整性与安全性的. 对信息进行hash

认证机制:任何一家网站如果需要使用https必须备案.

# 京淘前台商品信息展现

## 扩展后缀匹配

### 添加配置类

@Configuration

**public** **class** MvcConfigurer **implements** WebMvcConfigurer{

//开启匹配后缀型配置

@Override

**public** **void** configurePathMatch(PathMatchConfigurer configurer) {

configurer.setUseSuffixPatternMatch(**true**);

}

}

## 业务说明

### 查询商品规则

url地址: <http://www.jt.com/items/562379.html>

用户发起http请求.根据items/562379.html获取商品信息.并且要求页面展现.

实现步骤:

1. 编辑业务层Controller拦截用户请求562379商品查询.
2. Jt-web前台利用http协议发起请求访问后台jt-manage服务器.

http://manage.jt.com/xxxx/xxxx/xxx?id=562379

1. jt-manage后台接收前台请求.之后进行业务处理,之后将数据转化为**json**串返回.
2. Jt-web前台将后台返回的json串转化为对象之后页面展现.

## HttpClient

### HttpClient介绍

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 .6（2015-09-11）

### HttpClient入门案例

**public** **class** TestHttpClient {

/\*\*

\* 测试HttpClient

\* 1.实例化httpClient对象

\* 2.定义http请求路径 url/uri

\* 3.定义请求方式 get/post

\* 4.利用API发起http请求

\* 5.获取返回值以后判断状态信息 200

\* 6.获取响应数据.

\* **@throws** IOException

\* **@throws** ClientProtocolException

\*/

@Test

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

CloseableHttpClient client =

HttpClients.*createDefault*();

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

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

CloseableHttpResponse response =

client.execute(httpGet);

**if**(response.getStatusLine()

.getStatusCode() == 200) {

System.***out***.println("恭喜你请求成功!!!!!");

HttpEntity httpEntity = response.getEntity();

String result = EntityUtils.*toString*(httpEntity);

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

}**else** {

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

}

}

}

## SpringBoot整合httpClient

### 编辑配置类

@Configuration

@PropertySource(value="classpath:/properties/httpClient.properties")

**public** **class** HttpClientConfig {

@Value("${http.maxTotal}")

**private** Integer maxTotal; //最大连接数

@Value("${http.defaultMaxPerRoute}")

**private** Integer defaultMaxPerRoute; //最大并发链接数

@Value("${http.connectTimeout}")

**private** Integer connectTimeout; //创建链接的最大时间

@Value("${http.connectionRequestTimeout}")

**private** Integer connectionRequestTimeout; //链接获取超时时间

@Value("${http.socketTimeout}")

**private** Integer socketTimeout; //数据传输最长时间

@Value("${http.staleConnectionCheckEnabled}")

**private** **boolean** staleConnectionCheckEnabled; //提交时检查链接是否可用

//定义httpClient链接池

@Bean(name="httpClientConnectionManager")

**public** PoolingHttpClientConnectionManager getPoolingHttpClientConnectionManager() {

PoolingHttpClientConnectionManager manager = **new** PoolingHttpClientConnectionManager();

manager.setMaxTotal(maxTotal); //设定最大链接数

manager.setDefaultMaxPerRoute(defaultMaxPerRoute); //设定并发链接数

**return** manager;

}

//定义HttpClient

/\*\*

\* 实例化连接池，设置连接池管理器。

\* 这里需要以参数形式注入上面实例化的连接池管理器

**@Qualifier** 指定bean标签进行注入

\*/

@Bean(name = "httpClientBuilder")

**public** HttpClientBuilder getHttpClientBuilder(@Qualifier("httpClientConnectionManager")PoolingHttpClientConnectionManager httpClientConnectionManager){

//HttpClientBuilder中的构造方法被protected修饰，所以这里不能直接使用new来实例化一个HttpClientBuilder,可以使用HttpClientBuilder提供的静态方法create()来获取HttpClientBuilder对象

HttpClientBuilder httpClientBuilder = HttpClientBuilder.*create*();

httpClientBuilder.setConnectionManager(httpClientConnectionManager);

**return** httpClientBuilder;

}

/\*\*

\* 注入连接池，用于获取httpClient

\* **@param** httpClientBuilder

\* **@return**

\*/

@Bean

@Scope("prototype") //多例对象

**public** CloseableHttpClient getCloseableHttpClient(@Qualifier("httpClientBuilder") HttpClientBuilder httpClientBuilder){

**return** httpClientBuilder.build();

}

/\*\*

\* Builder是RequestConfig的一个内部类

\* 通过RequestConfig的custom方法来获取到一个Builder对象

\* 设置builder的连接信息

\* **@return**

\*/

@Bean(name = "builder")

**public** RequestConfig.Builder getBuilder(){

RequestConfig.Builder builder = RequestConfig.*custom*();

**return** builder.setConnectTimeout(connectTimeout)

.setConnectionRequestTimeout(connectionRequestTimeout)

.setSocketTimeout(socketTimeout)

.~~setStaleConnectionCheckEnabled~~(staleConnectionCheckEnabled);

}

/\*\*

\* 使用builder构建一个RequestConfig对象

\* **@param** builder

\* **@return**

\* 定义请求超时时间配置类

\*/

@Bean

**public** RequestConfig getRequestConfig(@Qualifier("builder") RequestConfig.Builder builder){

**return** builder.build();

}

}

### 编辑线程关闭方法

@Component //交给spring容器管理

**public** **class** HttpClientClose **extends** Thread{

@Autowired

**private** PoolingHttpClientConnectionManager manage;

**private** **volatile** **boolean** shutdown; //开关 volatitle表示多线程可变数据,一个线程修改,其他线程立即修改

**public** HttpClientClose() {

///System.out.println("执行构造方法,实例化对象");

//线程开启启动

**this**.start();

}

@Override

**public** **void** run() {

**try** {

//如果服务没有关闭,执行线程

**while**(!shutdown) {

**synchronized** (**this**) {

wait(5000); //等待5秒

//System.out.println("线程开始执行,关闭超时链接");

//关闭超时的链接

PoolStats stats = manage.getTotalStats();

**int** av = stats.getAvailable(); //获取可用的线程数量

**int** pend = stats.getPending(); //获取阻塞的线程数量

**int** lea = stats.getLeased(); //获取当前正在使用的链接数量

**int** max = stats.getMax();

//System.out.println("max/"+max+": av/"+av+": pend/"+pend+": lea/"+lea);

manage.closeExpiredConnections();

}

}

} **catch** (Exception e) {

e.printStackTrace();

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

}

**super**.run();

}

//关闭清理无效连接的线程

@PreDestroy //容器关闭时执行该方法.

**public** **void** shutdown() {

shutdown = **true**;

**synchronized** (**this**) {

//System.out.println("关闭全部链接!!");

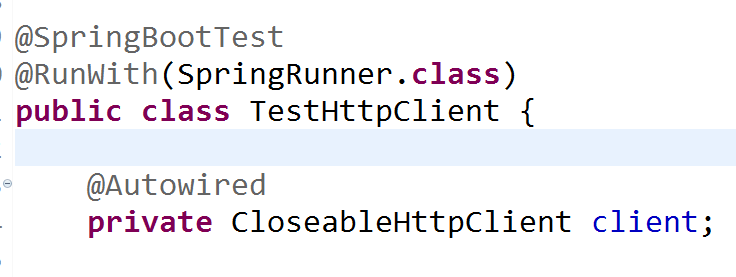
notifyAll(); //全部从等待中唤醒.执行关闭操作;

}

}

}

### SpringBoot测试类测试



## 实现httpCleint工具类编辑

### 工具类构建思路

1. 封装多种请求方式 get/post
2. 参数定义:
   1. url地址
   2. 封装用户提交参数 Map<String,String>
   3. 指定编码格式 设定默认值 utf-8
3. 返回字符串String

### 编辑工具API

@Service

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

//通过spring容器注入对象

@Autowired

**private** CloseableHttpClient httpClient;

@Autowired

**private** RequestConfig requestConfig;

/\*\*

\* 构建get请求工具类

\* **@param** url

\* **@param** params

\* **@param** charset

\* **@return**

\*

\* 目的:

\* 利用用户传递url,帮助用户发起http get请求.

\*

\* 是否有参数:

\* 没有参数: http://manage.jt.com/xxx

\* 有参数: http://manage.jt.com?key1=value1&key2=value2....

\* 利用httpClient发起请求返回结果即可.

\*/

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

//1.检验字符集编码格式

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

//表示用户没有指定字符类型 指定默认值

charset = "UTF-8";

}

//2.校验map集合是否为null

//Set<Map.Entry<K, V>> entrySet();

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

url = url + "?"; //http://manage.jt.com?

//有参数: ?key1=value1&key2=value2....

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

String key = entry.getKey();

String value = entry.getValue();

url = url+key+"="+value+"&";

}

//经过循环之后多&符

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

}

//定义请求对象

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

httpGet.setConfig(requestConfig); //设定超时时间

String result = **null**;

**try** {

CloseableHttpResponse httpResponse = httpClient.execute(httpGet);

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

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

}

} **catch** (Exception e) {

e.printStackTrace();

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

}

**return** result;

}

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

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

}

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

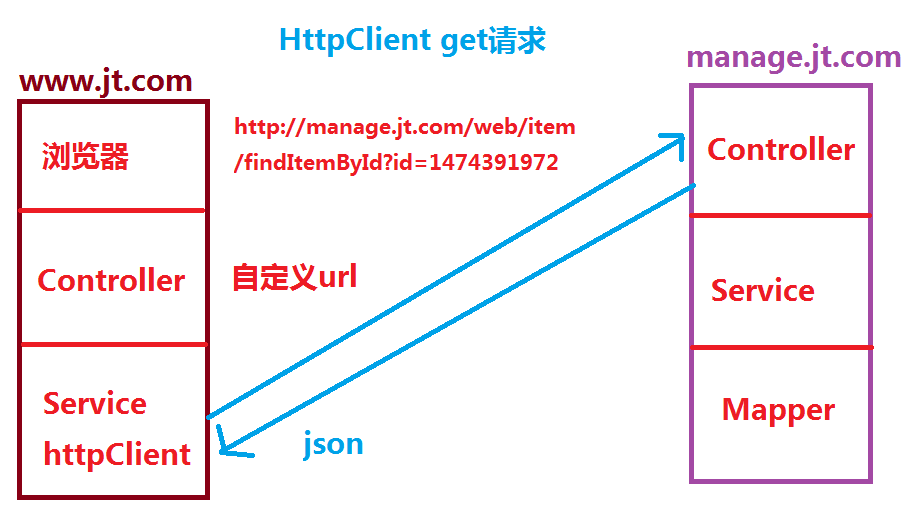
**return** doGet(url, params, **null**);

}

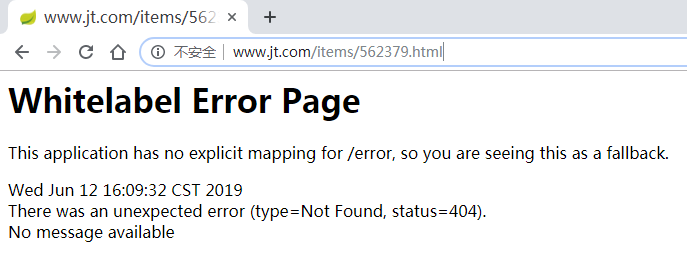
}

## 商品详情展现

### httpClient调用流程



### 页面url分析



页面分析



### 编辑后台Controller

//后台接收用户请求获取商品信息

//url:manage.jt.com/web/item/findItemById?id=562379

@RestController

@RequestMapping("/web/item")

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

@Autowired

**private** ItemService itemService;

//查询item对象

@RequestMapping("/findItemById")

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

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

}

//http://manage.jt.com/web/item/findItemById

//?id=562379

//查询itemDesc对象

@RequestMapping("/findItemDescById")

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

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

}

}

### 编辑后台Service

@Override

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

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

}

@Override

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

**return** itemMapper.selectById(id);

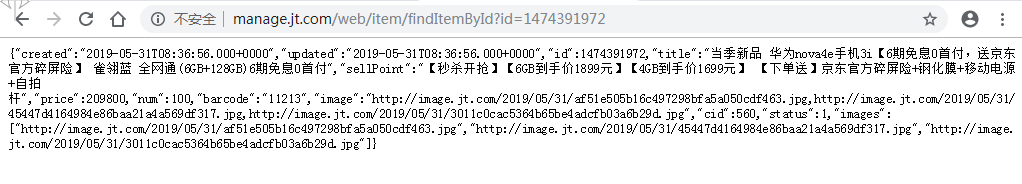
}

### 页面效果

利用url:

<http://manage.jt.com/web/item/findItemById?id=1474391972>

展现商品数据.



### 编辑前台Controller

@Controller

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

@Autowired

**private** ItemService itemService;

/\*\*

\* 根据商品id查询后台服务器数据

\* 业务步骤:

\* 1.在前台service中实现httpClient调用

\* 2.后台根据itemId查询数据库返回对象的json串

\* 3.将json转化为item对象

\* 4.将item对象保存到request域中

\* 5.返回页面逻辑名称 item

\*/

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

**public** String findItemById(@PathVariable Long itemId,

Model model) {

Item item = itemService.findItemById(itemId);

ItemDesc itemDesc = itemService.findItemDescById(itemId);

model.addAttribute("item", item);

model.addAttribute("itemDesc", itemDesc);

**return** "item";

}

}

### 编辑前台Service

@Service

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

@Autowired

**private** HttpClientService httpClient;

@Override

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

String url =

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

//为了满足get请求需求 定义id=xxx

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

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

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

Item item = ObjectMapperUtil

.*toObject*(result, Item.**class**);

**return** item;

}

@Override

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

String url =

"http://manage.jt.com/web/item/findItemDescById";

//为了满足get请求需求 定义id=xxx

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

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

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

ItemDesc itemDesc = ObjectMapperUtil

.*toObject*(result, ItemDesc.**class**);

**return** itemDesc;

}

}

### 页面效果

