# 京淘项目实现商品展现

## 问题解答

### 跳过测试类打包

<!--跳过测试类打包 -->

<plugin>

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

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

<configuration>

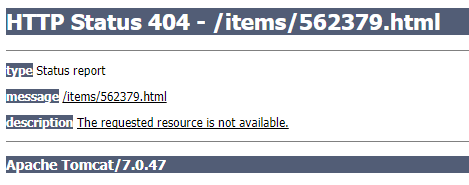
<skip>true</skip>

</configuration>

</plugin>

## 实现商品详情展现

### 页面分析



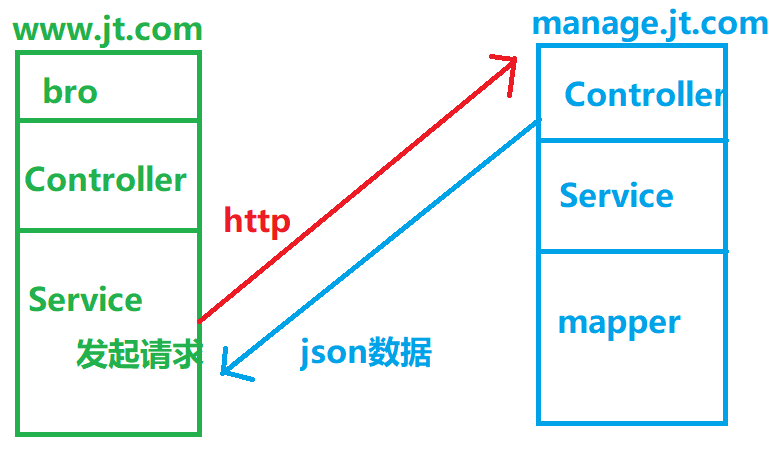
### 业务说明:

1. 接收用户请求.
2. 使用REST风格接收商品的ID号
3. 前台接收用户请求数据,之后通过HttpClient将请求发送给jt-manage
4. 之后jt-manage接收前台发起的请求,进行后端的业务层处理.处理完成后

返回数据给前台.

1. 前台接收返回数据后,将数据进行解析.最终为用户展现数据.

### 程序调用流程



### 编辑前台Controller

@Controller

@RequestMapping("/items")

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

@Autowired

**private** ItemService itemService;

//实现商品详情展现

@RequestMapping("/{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

@Service

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

@Autowired

**private** HttpClientService httpClient;

@Autowired

**private** ObjectMapper objectMapper;

@Override

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

//1.定义远程访问后台的url地址

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

//2.定义参数

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

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

//3发起请求,获取返回值结果json串

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

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

Item item = **null**;

**try** {

item =

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

} **catch** (Exception e) {

e.printStackTrace();

}

**return** item;

}

}

### 编辑后台Controller

//类中的所有的方法的返回值必须的为json

@RestController //@Controller + @ResponseBody

@RequestMapping("/web/item")

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

@Autowired

**private** ItemService itemService;

//http://manage.jt.com/web/item/findItemById?itemId=5623879

//表示返回的数据为json串

@RequestMapping("/findItemById")

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

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

}

}

### 编辑后台Service

public Item findItemById(Long itemId) {

return itemMapper.selectByPrimaryKey(itemId);

### }项目启动后台测试

**注意事项:必须保证数据库中有562379 的数据!!!**



### JSON转换异常



说明:

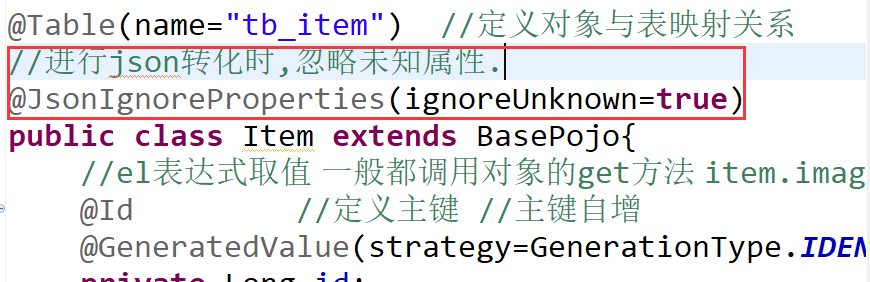
1. 从后台获取的JSON数据

{"created":1425821274000,"updated":1425821274000,"id":562379,"title":"三星 W999 黑色 电信3G手机 双卡双待双通","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"]}

1. 前台获取数据后,需要进行数据的转化

对象转化JSON时,调用的是对象的get方法.如果将json转化为对象时,调用对象的set方法.但是item对象中没有setImages方法,所以转化异常.

解决方案:



## 作业

实现商品详情展现

### 编辑前台Controller

@Controller

@RequestMapping("/items")

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

@Autowired

**private** ItemService itemService;

//实现商品详情展现

@RequestMapping("/{itemId}")

**public** String findItemById

(@PathVariable Long itemId,Model model) {

Item item = itemService.findItemById(itemId);

ItemDesc itemDesc = itemService.findItemDesc(itemId);

model.addAttribute("item", item);

model.addAttribute("itemDesc", itemDesc);

**return** "item";

}

}

### 编辑前台Service

@Override

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

//1.定义远程访问后台的url地址

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

//2.定义参数

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

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

//3发起请求,获取返回值结果json串

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

//System.out.println(result);

ItemDesc itemDesc = **null**;

**try** {

//对象转化json get json转化为对象 set方法

itemDesc =

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

} **catch** (Exception e) {

e.printStackTrace();

}

**return** itemDesc;

}

### 编辑后台Controller

@RequestMapping("/findItemDescById")

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

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

}

### 编辑后台Service

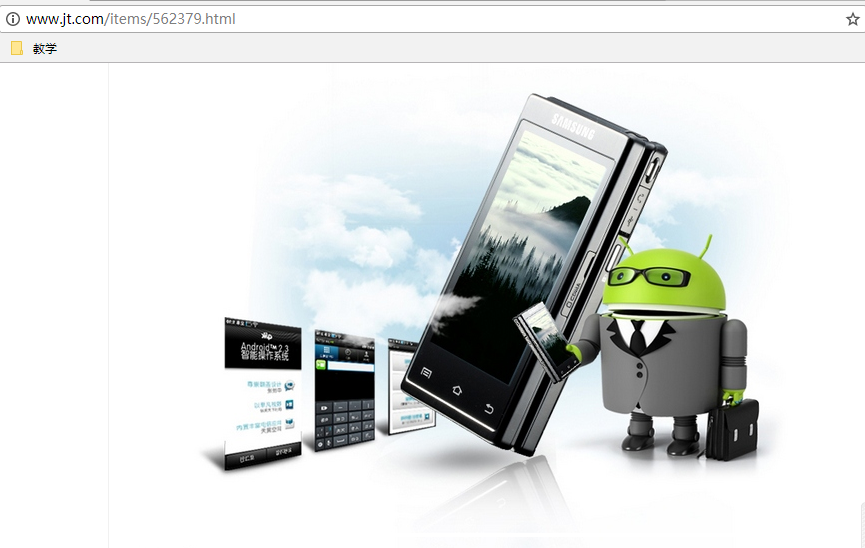
@Override

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

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

}

### 页面效果



# SpringBoot

## SpringBoot介绍

### SSM框架使用的缺点

1. 三大框架使用时,配置比较繁琐
2. 框架使用时,jar包文件依赖复杂.初学者不清楚依赖关系.
3. Spring/SpringMVC/mybatis,需要人为的干预配置.同时依赖web容器.

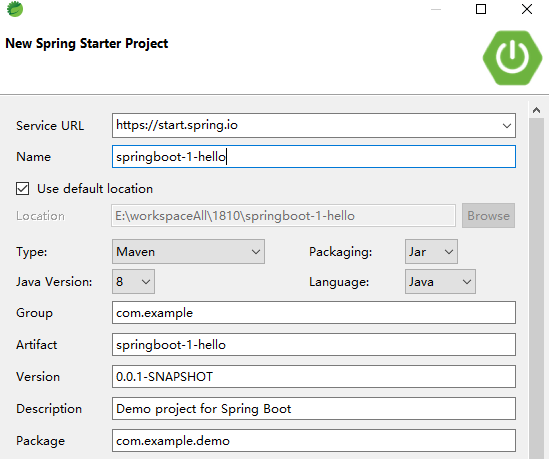
### SpringBoot

说明:SpringBoot是spring最新的框架一站式的解决方案.其中内部将SSM框架再次封装.并且内部实现了自动的依赖.如果引入第三方框架或者jar包.则会自动的添加对应的依赖.引入特别的简单.

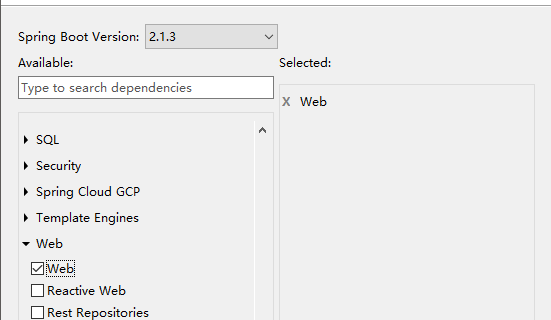
核心:使用springBoot程序,程序员无需再次关注配置文件.将更多的时间放在业务中.5分钟

## SpringBoot入门案例

### 创建项目



2.选择依赖



### Jar包依赖

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

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

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.3.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>springboot-1-hello</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>springboot-1-hello</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!--springBoot动态的引入springMVC全部的配置 -->

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--引入测试类 -->

<dependency>

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

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!--引入springBoot项目启动项 内置tomcat -->

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

### 编辑主启动类

@SpringBootApplication

**public** **class** Springboot1HelloApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(Springboot1HelloApplication.**class**, args);

}

}

主启动类说明:

以后写代码,一定要在主启动类的包下或者子包中编辑,因为包扫描时按照主启动类的路径扫描.

### 编辑Controller

@RestController

**public** **class** HelloController {

@RequestMapping("/hello")

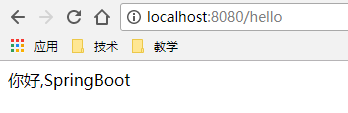
**public** String hello() {

**return** "你好,SpringBoot";

}

}

### 测试案例



## SpringBoot中的配置文件

### 修改端口/发布路径

server:

port: 8091 #定义项目端口

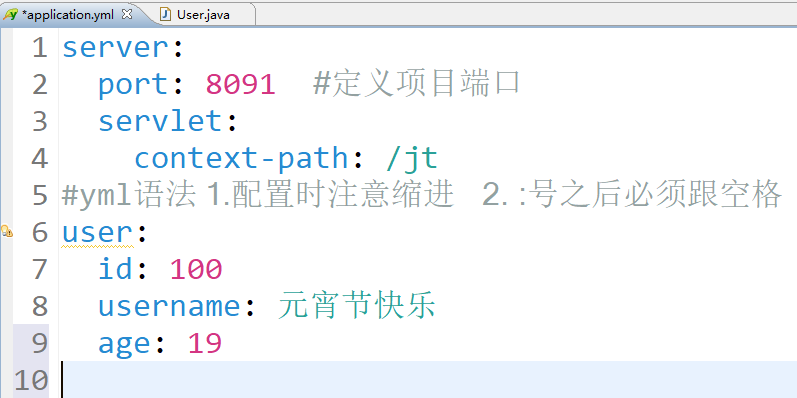
servlet:

context-path: /jt

#yml语法 1.配置时注意缩进 2. :号之后必须跟空格

### 为对象赋值

1. 编辑yml文件



1. 编辑对象

@Component //表示将user对象交给spring容器管理

//@ConfigurationProperties(prefix="user")

**public** **class** User {

@Value("${user.id}")

**private** Integer id;

@Value("${user.username}")

**private** String username;

@Value("${user.age}")

**private** Integer age;

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** Integer getAge() {

**return** age;

}

**public** **void** setAge(Integer age) {

**this**.age = age;

}

@Override

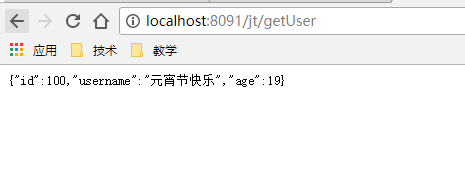
**public** String toString() {

**return** "User [id=" + id + ", username=" + username + ", age=" + age + "]";

}

}

1. 页面效果



### 批量属性赋值

1. 引入jar包

<!--添加属性注入依赖 -->

<dependency>

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

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

1. 编辑User对象



作用:可以实现对象属性的批量赋值.

### 通过指定pro文件为属性赋值

1. 编辑pro文件

dog.id=101

dog.name=tomcat

2.编辑Dog对象

@Component

//导入指定的properties配置文件

@PropertySource("classpath:/properties/dog.properties")

@ConfigurationProperties(prefix="dog")

**public** **class** Dog {

**private** Integer id;

**private** String name;

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

### 页面效果展现



### 环境切换

说明: 实际开发中,可能需要多个环境测试.

1. 编辑yml配置文件

#定义用户默认的链接环境

spring:

profiles:

active: prod

--- #定义环境分割线

server:

port: 8091 #定义项目端口

servlet:

context-path: /jt

#yml语法 1.配置时注意缩进 2. :号之后必须跟空格

user:

id: 100

username: 元宵节快乐

age: 19

#定义环境的名称

spring:

profiles: prod

--- #环境的分割线

server:

port: 8095 #定义项目端口

servlet:

context-path: /

#yml语法 1.配置时注意缩进 2. :号之后必须跟空格

user:

id: 1000

username: 元宵节快乐

age: 1900

#定义环境的名称

spring:

profiles: dev

通过选择不同的环境,测试代码是否正常.

### 添加tomcat热部署

<!--支持热部署 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>

<version>1.2.8.RELEASE</version>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

修改文件后测试文件是否自动加载

## Lombok介绍

### 介绍

一般使用POJO对象时,需要自己定义GET/SET/toString…..,使用lombok可以自动的额生成POJO对象中的全部的方法.

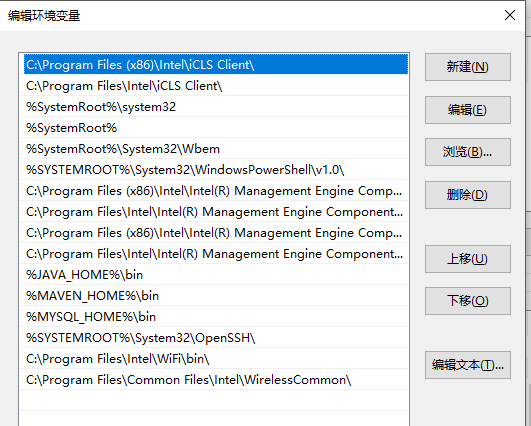
### JDK安装问题

1. 版本统一

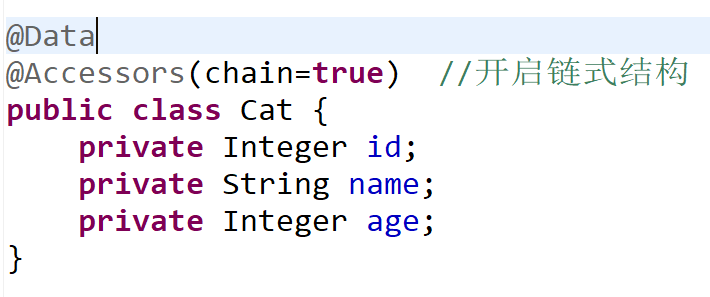
Java版本 64位

STS 64位

1. JDK安装问题
2. 定义java\_home
3. 将java\_home的path配置放置到第一个



### LomBok使用



## SpringBoot整合Mybatis

1. 添加jar包文件

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.jt.1810</groupId>

<artifactId>springboot-2-mybatis</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

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

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.3.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!--springBoot动态的引入springMVC全部的配置 -->

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--引入测试类 -->

<dependency>

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

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!--添加属性注入依赖 -->

<dependency>

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

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

<!--支持热部署 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>

<version>1.2.8.RELEASE</version>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

<!--引入插件lombok 自动的set/get/构造方法插件 -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<!--引入数据库驱动 -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<!--引入druid数据源 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.1.12</version>

</dependency>

<!--spring整合mybatis 暂时 -->

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

<build>

<plugins>

<!--引入springBoot项目启动项 内置tomcat -->

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

### 编辑主启动类

//springBoot的主启动类

@SpringBootApplication

@MapperScan("com.jt.mapper") //为指定的包路径创建代理对象

**public** **class** SpringBoot\_mybatis {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBoot\_mybatis.**class**, args);

}

}

### 编辑yml文件

server:

port: 8091

servlet:

context-path: /

spring:

datasource:

#引入druid数据源

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/jtdb?serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true

username: root

password: root

mybatis:

type-aliases-package: com.jt.pojo

mapper-locations: classpath:/mybatis/mappers/\*.xml

#开启驼峰映射

configuration:

map-underscore-to-camel-case: true

### 编辑mapper接口文件

1.添加mapper接口

public interface UserMapper {

List<User> findAll();

}

1. 编辑mapper映射文件

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

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

<!--返回值需要添加别名包 -->

<select id=*"findAll"* resultType=*"User"*>

select \* from user

</select>

</mapper>

### 代码测试

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** TestMybatis {

//该测试类,相当于直接获取了spring容器对象

@Autowired

**private** UserMapper userMapper;

@Test

**public** **void** testFind() {

List<User> userList = userMapper.findAll();

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

}

}

测试结果:

