Maven管理的工程结构：

。。。。。。

前台工程

contentSale-parent

contentSale-common

contentSale-manager(pom工程)聚合工程

contentSale -web(war包)

contentSale -service(jar包)

contentSale -mapper(jar包)

contentSale -pojo(jar包)

继承：

依赖：

**后台管理系统工程结构：**

contentSale -parent -- 管理依赖jar包的版本，全局，公司级别

|-- contentSale-common --- 通用组件、工具类

|-- contentSale -manage -- 后台系统

|--com. contentSale.manage.web

|--com. contentSale.manage.service

|--com. contentSale.manage.mapper

|--com. contentSale.manage.pojo

**运行：**

1. contentSale-parent，contentSale-common需要安装至本地仓库：

run as 🡪 maven install

2. 右键contentSale-manager,选择 run as 🡪 maven build，输入命令：

clean tomcat7:run

tomcat7指定使用tomcat7的插件。

3.若修改了contentSale-parent，contentSale-common的pom.xml文件，记得要重新将contentSale-parent，contentSale-common安装至本地。

## 整合的思路

### Dao层（Model层）

使用mybatis框架。创建SqlMapConfig.xml。

创建一个applicationContext-dao.xml

1. 配置数据源
2. 需要让spring容器管理SqlsessionFactory，单例存在。
3. 把mapper的代理对象放到spring容器中。使用扫描包的方式加载mapper的代理对象。

### Service层（Controller层）

1. 事务管理
2. 需要把service实现类对象放到spring容器中管理。

### 表现层（View层）

1. 配置注解驱动
2. 配置视图解析器
3. 需要扫描controller

### Web.xml

1. spring容器的配置
2. Springmvc前端控制器的配置
3. Post乱码过滤器

## 框架整合

需要把配置文件放到contentSale-manager-web工程下。因为此工程为war工程，其他的工程只是一个jar包。

# 2.测试整合结果

## 2.1需求

根据内容id查询内容信息。

## 2.2 Sql语句

SELECT \* from contents WHERE content\_id=2

## 2.3 Dao层

可以使用逆向工程生成的mapper文件。

**参考链接：**

[MyBatis框架的学习(七)——MyBatis逆向工程自动生成代码](http://blog.csdn.net/yerenyuan_pku/article/details/71909325)

## 2.4 Service层

接收商品id调用dao查询商品信息。返回商品pojo对象。

**package** com.contentsale.service;

**import** com.contentsale.pojo.Contents;

**public** **interface** ContentService {

Contents getContentById(Integer id);

}

**package** com.contentsale.service.impl;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** com.contentsale.mapper.ContentsMapper;

**import** com.contentsale.pojo.Contents;

**import** com.contentsale.pojo.ContentsExample;

**import** com.contentsale.pojo.ContentsExample.Criteria;

**import** com.contentsale.service.ContentService;

/\*\*

\* 内容管理Service

\* **@author** YQ

\*/

@Service

**public** **class** ContentServiceImpl **implements** ContentService {

@Autowired

**private** ContentsMapper contentsMapper;//注入接口代理对象

@Override

**public** Contents getContentById(Integer contentId) {

// Contents content = contentsMapper.selectByPrimaryKey(contentId);

ContentsExample example = **new** ContentsExample();

//添加查询条件,用户自定义查询条件

Criteria criteria = example.createCriteria();

criteria.andContentIdEqualTo(contentId);

List<Contents> list = contentsMapper.selectByExample(example);

**if**(list != **null** && list.size()>0) {

Contents content = list.get(0);

**return** content;

}

**return** **null**;

}

}

## 2.5 Controller层

接收页面请求商品id，调用service查询商品信息。直接返回一个json数据。需要使用@ResponseBody注解。

**package** com.contentsale.controller;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.ResponseBody;

**import** com.contentsale.pojo.Contents;

**import** com.contentsale.service.ContentService;

/\*\*

\* 内容管理C

\* **@author** YQ

\*

\*/

@Controller

**public** **class** ContentController {

@Autowired

**private** ContentService contentService;

@RequestMapping("/content/{contentId}")

@ResponseBody

**public** Contents getContentById(@PathVariable(value="contentId") Integer contentId) {

Contents content = contentService.getContentById(contentId);

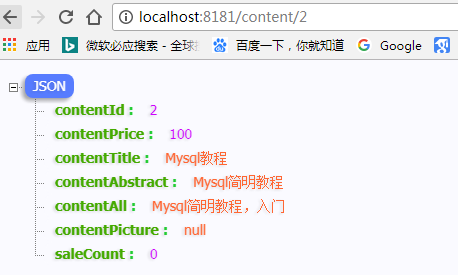
**return** content;

}

}

通过 @PathVariable 可以将 URL 中占位符参数绑定到控制器处理方法的入参中：URL 中的 {xxx} 占位符可以通过@PathVariable("xxx") 绑定到操作方法的入参中。

**测试成功：**



# 3.系统需求分析

**默认为商品展示页面，要购买时跳转至登录页面。**

## 3.1 登录

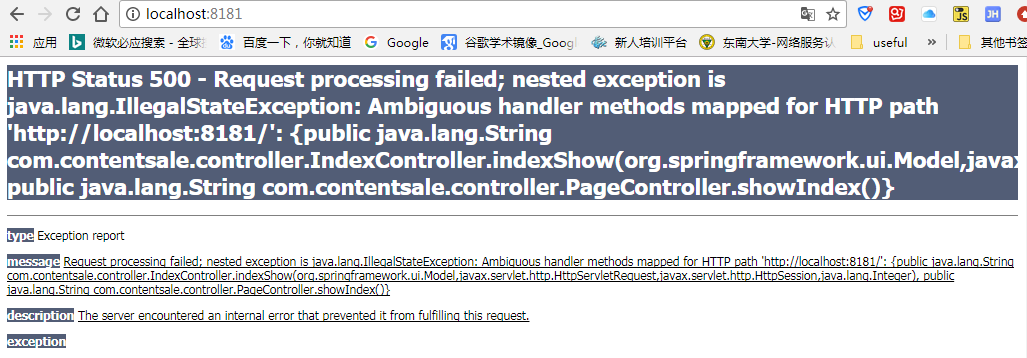
Error loading class

[org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer] for bean with name 'freemarkerConfig' defined in class path resource [spring/springmvc.xml]: problem with class file or dependent class; nested exception is java.lang.NoClassDefFoundError: org/springframework/ui/freemarker/FreeMarkerConfigurationFactory

解决：



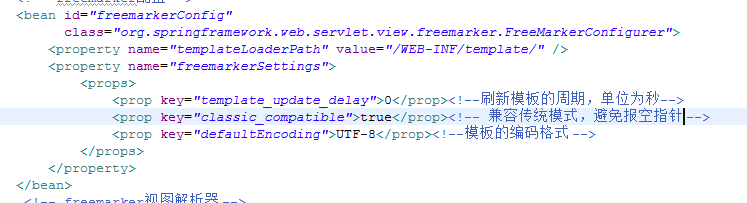
**BUG:**

原因：PageController中包含“/”的路由设置，重复报错。

**注释后报空指针错误：**



**Freemarker的classic\_compatible设置能够解决页面前端的报空错误：**



**登录成功后的页面：**



## 3.2 买家

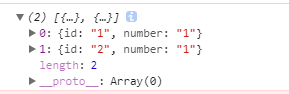
账号：buyer 密码：reyub

* 登录页面

**登陆后：**

* 内容展示界面：显示用户昵称，退出、财务、购物车三个功能按键
* 已购买商品做特殊标记，表明已购买。
* 可查看已购买内容，可查看未购买内容
* 已购买内容的购买按钮不可用，未购买内容按钮可用
* 已购买内容显示当时购买价格 和 购买数量
* 内容查看界面点击“购买”，弹窗确认是否购买，点击“确认”，内容加入购物车，购买过的内容可重复购买，此处的“购买”应指已添加到购物车的内容可再添加至购物车。
* 任何页面点击“购物车”，进入结算页面：
  + 结算页面点击“购买”🡪购买购物车内全部内容

购买是异步请求,请求的json数据：



* 1. 结算页面点击“退出”🡪返回进入结算页面之前的页面
* 财务页面：列表方式列出所有已购买内容，列表最后计算总金额。
* 登录退出。

内容查看页面：

product.isBuy —— 标记该内容买家是否购买过

## 3.3 卖家

账号：seller 密码：relles

登陆后：

* 显示用户昵称，有“退出”和“发布”两个功能按钮
* 查看购买情况：

已出售内容上有数字表明已出售数量。

* 展示界面可删除**未出售**内容
* 查看内容页面，展示标题、摘要文字、价格、全文和“编辑”按钮

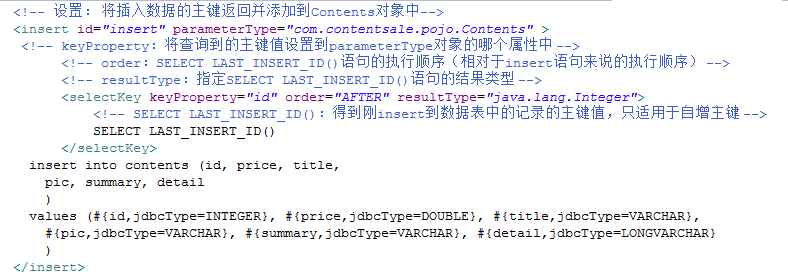
1. 发布内容：

* 内容展示界面点击“发布”，进入发布界面；
* 点击查看按钮即回到查看界面

1. 内容编辑

* 查看界面点击编辑，进入编辑页面。
* 编辑页面可修改内容全部细节，提交后回到查看界面。

**ssm配置mybatis实现自动获取自增id号至目标对象属性中：**



# 问题：

* 拦截器设置，一些页面登录拦截
* settleAccount页面点击“退出”按钮会出现404：

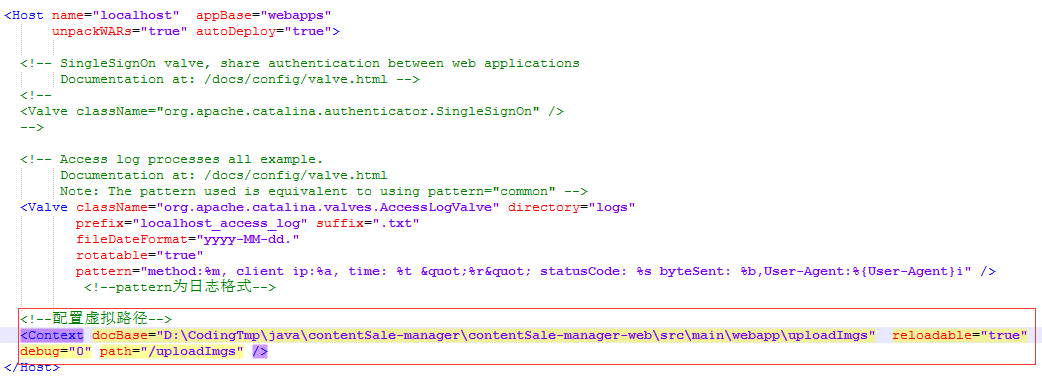


* 编辑页面提交后，详细内容未更新。。。。待更正
* 卖家的查看内容页面：详细信息未展示
* 本文图片上传失败，需要配置网站虚拟路径？？？

**网站根路径：**

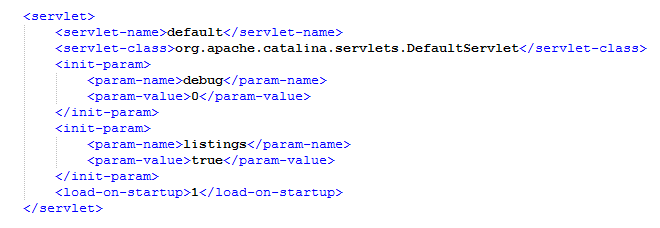
D:\CodingTmp\java\contentSale-manager\contentSale-manager-web\src\main\webapp\

tomcat的server.xml的host标签中添加虚拟路径配置语句：



docBase后的路径为文件夹绝对路径，path后的路径为对应映射的网站虚拟路径。

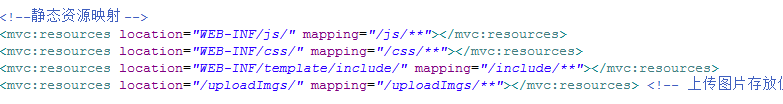
web.xml文件中listings的false改为true：可访问该目录，不会发生404错误



以上为单独的tomcat中的解决方案。

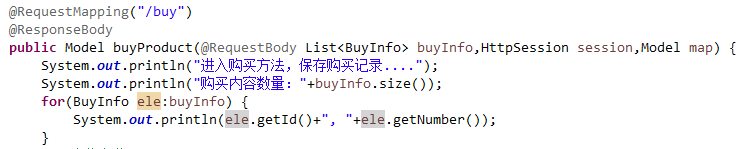
**maven tomcat plugin插件中的配置：**

springmvc.xml中添加静态资源映射路径解决：

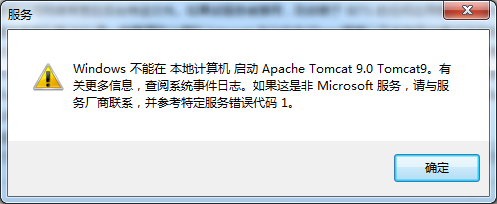


* org.springframework.beans.BeanInstantiationException: Failed to instantiate [java.util.List]: Specified class is an interface

Controller中对应参数要加@RequestBody注解，不然接收不到：



* tomcat 9安装失败：



* 不用保存在数据库中的对象怎么处理——新建一个对象