天天商城\_day02

1. **课程计划**
2. SSM框架整合
3. mybatis逆向工程
4. 商品列表
5. 商品列表分页处理
6. **SSM框架整合**
   1. **后台系统所用的技术**

框架：SpringMVC + Spring + Mybatis

前端：EasyUI

数据库：MySQL

* 1. **创建数据库**

1、安装mysql数据库

2、在mysql中创建一个shops数据库

3、导入数据库脚本ttshop.sql。

注意：对于电商项目，sql语句基本都是单表查询,分布式

* 1. **Mybatis逆向工程**

**执行逆向工程**

使用官方网站的mapper自动生成工具mybatis-generator-core-1.3.2来生成po类和mapper映射文件。

* 1. **整合思路**

### 1.Dao层：

mybatis整合spring，通过spring管理SqlSessionFactory、mapper代理对象。需要mybatis和spring的整合包。

|  |  |
| --- | --- |
| 整合内容 | 对应工程 |
| Pojo | tt-mangaer-pojo |
| Mapper映射文件 | tt-mangaer-dao |
| Mapper接口 | tt-mangaer-dao |
| sqlmapConfig.xml | tt-manager-web |
| applicationContext-dao.xml | tt-manager-web |

### 2.Service层：

所有的实现类都放到spring容器中管理。由spring创建数据库连接池，并有spring管理实务。

|  |  |
| --- | --- |
| 整合内容 | 对应工程 |
| Service接口及实现类 | tt-mangaer-service |
| applicationContext-service.xml | tt-manager-web |
| applicationContext-trans.xml | tt-manager-web |

### 3.表现层：

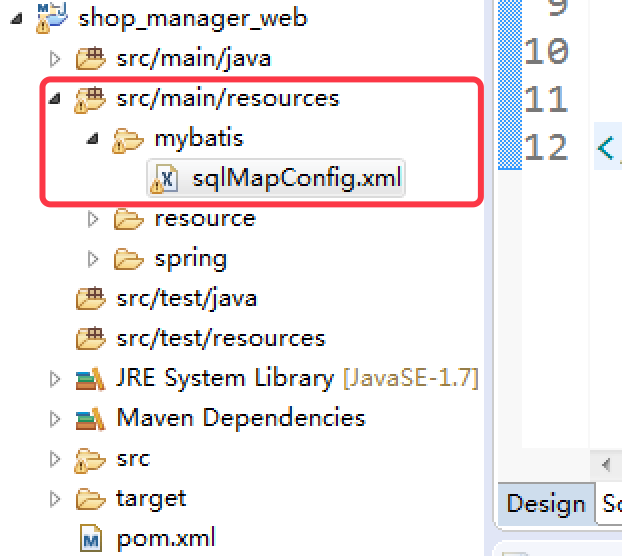
Springmvc整合spring框架，由springmvc管理controller。

|  |  |
| --- | --- |
| 整合内容 | 对应工程 |
| springmvc.xml | tt-manager-web |
| Controller | tt-manager-web |

* 1. **Dao整合**
     1. **创建sqlMapConfig.xml配置文件**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  <!-- 配置Mybatis的分页插件 -->  <plugins>  <plugin interceptor=*"com.github.pagehelper.PageHelper"*>  <property name=*"dialect"* value=*"mysql"* />  </plugin>  </plugins>  </configuration> |

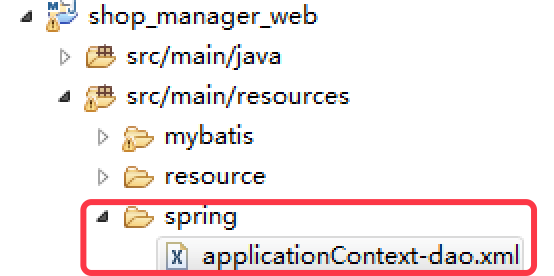
将该文件放到tt-manager-web中.



* + 1. **Spring整合Mybatis,创建applicationContext-dao.xml**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-4.0.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util*  *http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- Spring整合MyBatis的配置 -->    <!-- 加载db.properties配置文件 -->  <context:property-placeholder location=*"classpath:resource/\*.properties"* />    <!-- 配置德鲁伊数据源 -->  <bean name=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*>  <property name=*"driverClassName"* value=*"${db.driver}"*/>  <property name=*"url"* value=*"${db.url}"*/>  <property name=*"username"* value=*"${db.username}"*/>  <property name=*"password"* value=*"${db.password}"*/>  </bean>    <!-- 创建SqlSessionFactory -->  <bean name=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <!-- 设置数据源 -->  <property name=*"dataSource"* ref=*"dataSource"*/>  <!-- 加载sqlMapConfig.xml文件 -->  <property name=*"configLocation"* value=*"classpath:mybatis/sqlMapConfig.xml"*/>  </bean>    <!-- 创建Mapper的代理对象,负责生成Mapper对象 -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"com.syc.mapper"*/>  </bean>    </beans> |

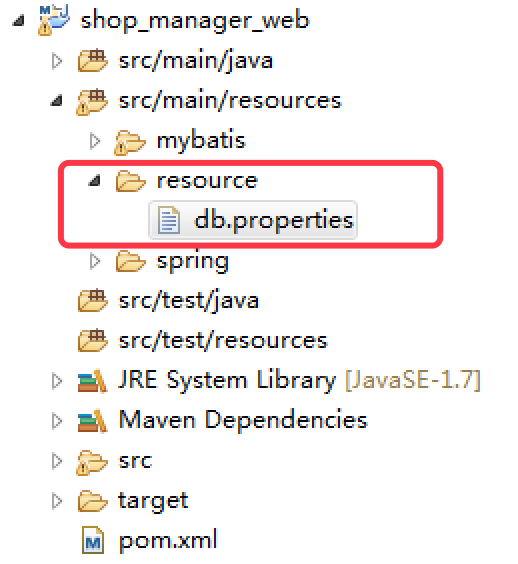
该文件存放在tt-manager-web项目下



### 创建db.properties

|  |
| --- |
| db.driver=com.mysql.jdbc.Driver  db.url=jdbc:mysql://localhost:3306/shop?characterEncoding=utf-8  db.username=root  db.password=syc |

该文件存放在tt-manager-web项目下



备注：

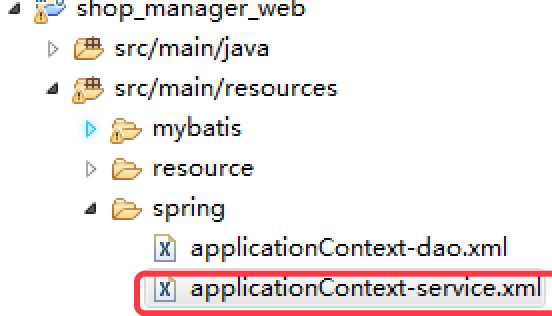
Druid是目前最好的数据库连接池，在功能、性能、扩展性方面，都超过其他数据库连接池，包括DBCP、C3P0、BoneCP、Proxool、JBoss DataSource。

Druid已经在阿里巴巴部署了超过600个应用，经过多年多生产环境大规模部署的严苛考验。

* 1. **Service整合**
     1. **管理Service类,创建applicationContext-service.xml**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-4.0.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util*  *http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- Spring负责创建service层的对象 -->  <context:component-scan base-package=*"com.syc.service"*/>  </beans> |

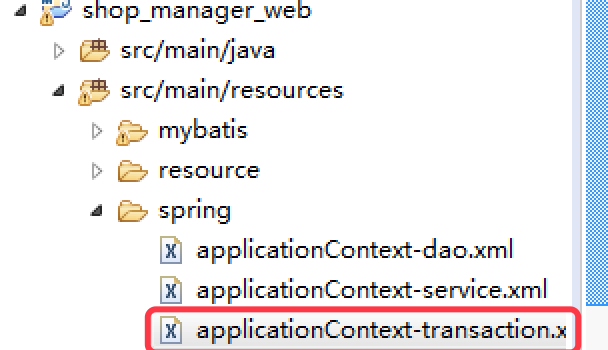
该文件存放在tt-manager-web项目下



* + 1. **进行事务管理,创建applicationContext-transaction.xml**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-4.0.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util*  *http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- Spring负责管理事务 -->    <!-- 创建事务管理器 -->  <bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <property name=*"dataSource"* ref=*"dataSource"*/>  </bean>  <!-- 配置事务的切面 -->  <tx:advice id=*"adviceId"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <tx:method name=*"add\*"* propagation=*"REQUIRED"* />  <tx:method name=*"insert\*"* propagation=*"REQUIRED"* />  <tx:method name=*"save\*"* propagation=*"REQUIRED"* />  <tx:method name=*"create\*"* propagation=*"REQUIRED"* />  <tx:method name=*"update\*"* propagation=*"REQUIRED"* />  <tx:method name=*"delete\*"* propagation=*"REQUIRED"* />  <tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  </tx:attributes>  </tx:advice>  <!-- 配置切入点表达式 -->  <aop:config>  <aop:advisor advice-ref=*"adviceId"*  pointcut=*"execution(\* com.syc.service..\*.\*(..))"* />  </aop:config>  </beans> |

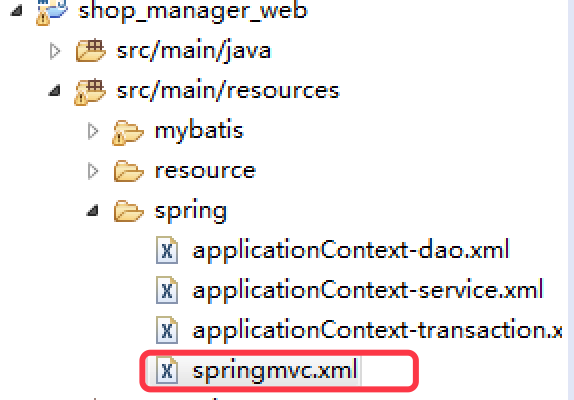
该文件存放在tt-manager-web项目中



* 1. **表现层整合**
     1. **创建springmvc.xml文件**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>  <!-- 配置要扫描的组件包 -->  <context:component-scan base-package=*"com.syc.controller"* />  <!-- 开启注解的扫描功能 -->  <mvc:annotation-driven />  <!-- 配置视图解析器 -->  <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/jsp/"*/>  <property name=*"suffix"* value=*".jsp"*/>  </bean>  <!-- 优雅REST风格的资源URL不希望带 .html 或 .do 等后缀.  由于早期的Spring MVC不能很好地处理静态资源,  所以在web.xml中配置DispatcherServlet的请求映射,  往往使用 \*.do 、 \*.xhtml等方式。这就决定了请求URL必须是一个带后缀的URL,  而无法采用真正的REST风格的URL。  如果将DispatcherServlet请求映射配置为"/"，则Spring MVC将捕获Web容器所有的请求，  包括静态资源的请求，Spring MVC会将它们当成一个普通请求处理，因此找不到对应处理器将导致错误。  如何让Spring框架能够捕获所有URL的请求，同时又将静态资源的请求转由Web容器处理，  是可将DispatcherServlet的请求映射配置为"/"的前提。  由于REST是Spring3.0最重要的功能之一，所以Spring团队很看重静态资源处理这项任务，  给出了堪称经典的两种解决方案。 -->  <!-- 资源映射,在spring3.0.4出现的,主要是用来进行静态资源的访问 -->  <mvc:resources location=*"/WEB-INF/js/"* mapping=*"/js/\*\*"* />  <mvc:resources location=*"/WEB-INF/css/"* mapping=*"/css/\*\*"* />  </beans> |

该文件存放在tt-manager-web目录下



* + 1. **配置web.xml文件**

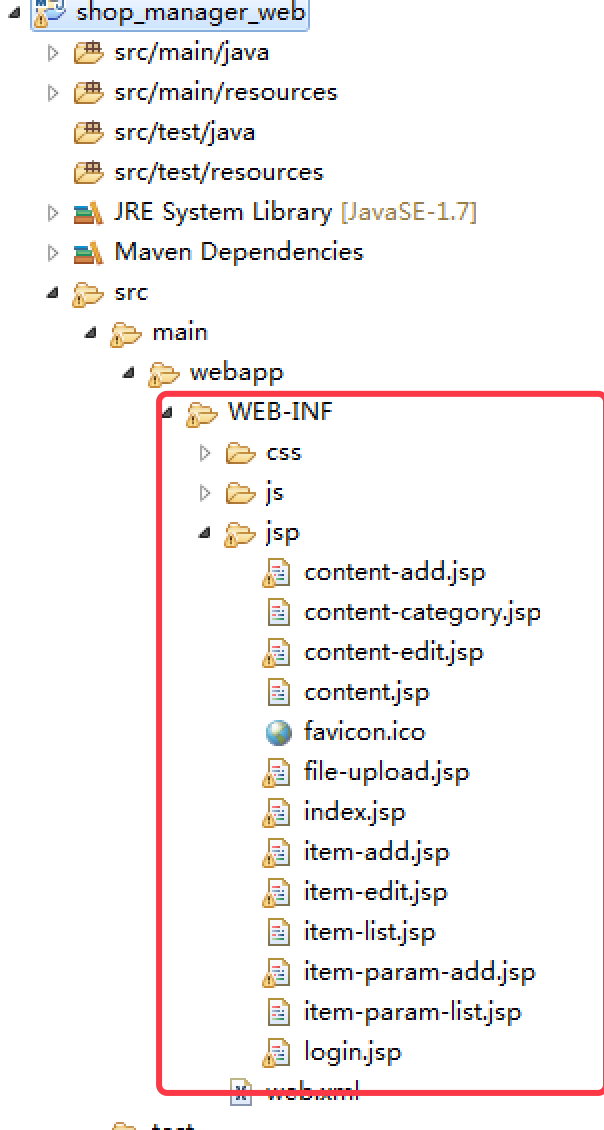
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  version=*"2.5"*>  <display-name>shop\_manager\_web</display-name>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  <!-- ContextLoaderListener监听器的作用就是启动Web容器时,  自动装配ApplicationContext的配置信息.因为它实现了ServletContextListener这个接口,  在web.xml配置这个监听器,启动容器时,就会默认执行它实现的方法.  在ContextLoaderListener中关联了ContextLoader这个类,  所以整个加载配置过程由ContextLoader来完成 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>    <!-- 配置要加载的applicationContext.xml文件路径 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext-\*.xml</param-value>  </context-param>  <!-- 配置SpringMVC的前端控制器 -->  <servlet>  <servlet-name>SpringMVC</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/springmvc.xml</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>SpringMVC</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>    <!-- 配置Spring-Web提供的过滤器,解决Post请求时的中文乱码问题 -->  <filter>  <filter-name>EncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>EncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  </web-app> |

* + 1. **整合静态页面**

静态页面位置：第二天（三大框架整合，后台系统搭建）\01.参考资料\后台管理系统静态页面

**使用方法:**

把静态页面添加到tt-manager-web工程中的WEB-INF下：



由于在web.xml中定义的url拦截形式为“/”表示拦截所有的url请求，包括静态资源例如css、js等。所以需要在springmvc.xml中添加资源映射标签：

|  |
| --- |
| <mvc:resources location=*"/WEB-INF/js/"* mapping=*"/js/\*\*"* />  <mvc:resources location=*"/WEB-INF/css/"* mapping=*"/css/\*\*"* /> |

* 1. **修改tt-manager-dao的pom文件**

在pom文件中添加如下内容：

|  |
| --- |
| <!-- 如果不添加此节点,mybatis的mapper.xml文件都会被漏掉? -->  <build>  <resources>  <resource>  <directory>src/main/java</directory>  <includes>  <include>\*\*/\*.properties</include>  <include>\*\*/\*.xml</include>  </includes>  <filtering>false</filtering>  </resource>  </resources>  </build> |

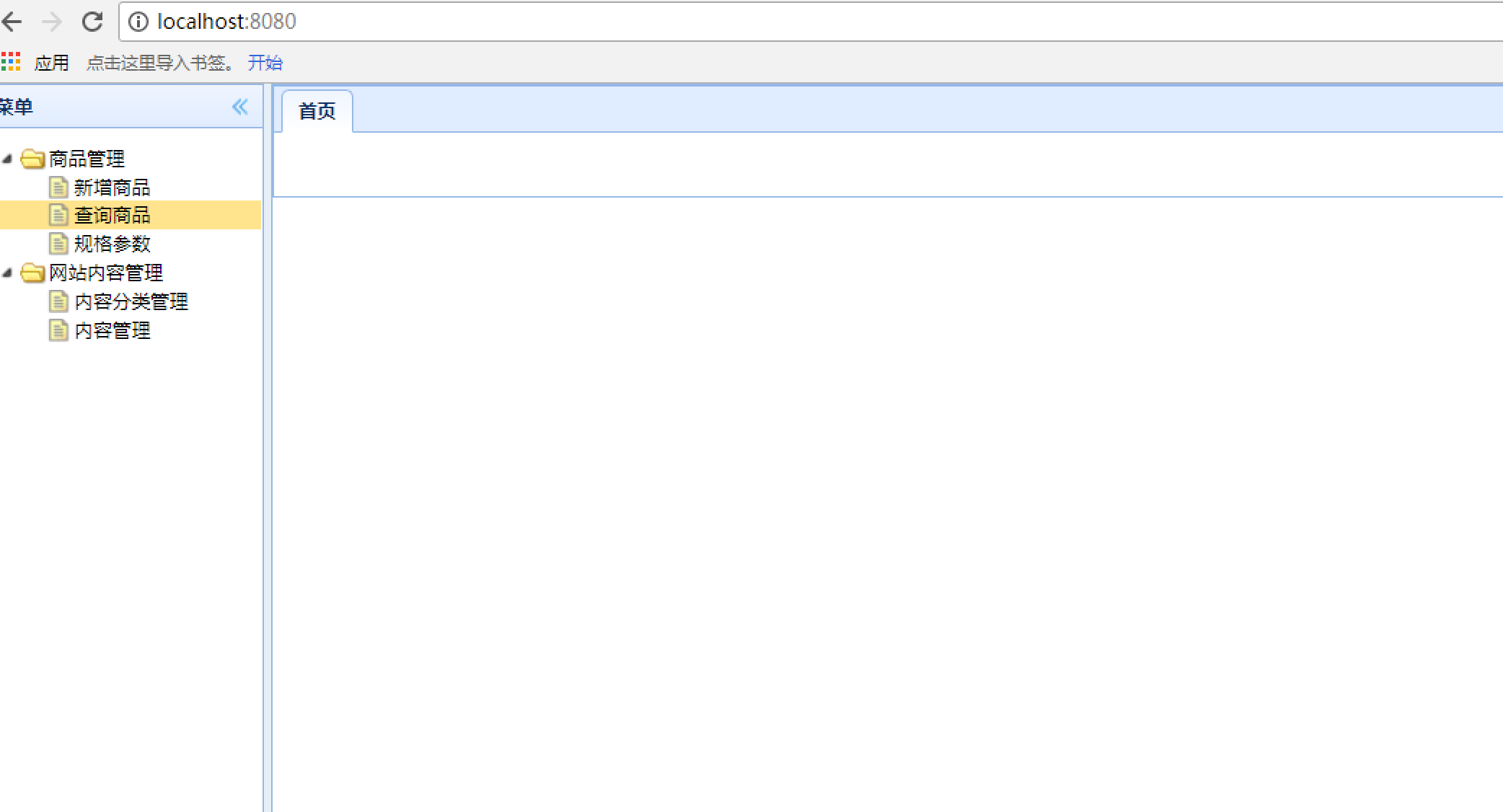
* 1. **整合测试**

根据商品id查询商品信息。

### 2.9.1 创建PageController控制器,跳转到首页

|  |
| --- |
| package com.syc.controller;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestMapping;  /\*  \* 分页的控制器  \*/  @Controller  public class PageController {  //显示首页  @RequestMapping("/")  public String toIndex(){  return "index";  }    //调到第几页  @RequestMapping("{page}")  public String goToPage(@PathVariable String page){  return page;  }  } |

### 2.9.2 在浏览器中输入<http://localhost:8080/>,进入首页



**对应的jsp为**：/WEB-INF/jsp/index.jsp

1. **商品列表查询**
   1. **商品列表页面**

**请求的url：**/item/list

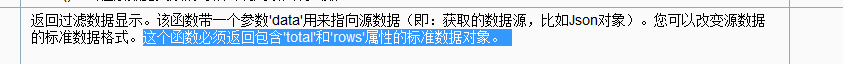
**请求的参数：datagrid给controller传递的参数**

page=1&rows=30 currentPage PageSize

**响应的json数据格式：**

Easyui中datagrid控件要求的数据格式为：Map,封装一个实体类

{total:”2”,rows:[{“id”:”1”,”name”,”张三”},{“id”:”2”,”name”,”李四”}]}



* 1. **响应的json数据格式EasyUIResult**

|  |
| --- |
| **package** com.syc.common.pojo;  **import** java.util.List;  **public** **class** EasyUIDataGridResult {  **private** **long** total;//总记录数  **private** List<?> rows;//行数  **public** **long** getTotal() {  **return** total;  }  **public** **void** setTotal(**long** total) {  **this**.total = total;  }  **public** List<?> getRows() {  **return** rows;  }  **public** **void** setRows(List<?> rows) {  **this**.rows = rows;  }  } |

该类存放在tt-commons项目中

* 1. **分页处理**
     1. **Mybatis分页插件 - PageHelper说明**

如果你也在用Mybatis，建议尝试该分页插件，这个一定是最方便使用的分页插件。

该插件目前支持Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL六种数据库分页。

* + 1. **使用方法**

第一步：在Mybatis配置xml中配置拦截器插件:

<plugins>

<!-- com.github.pagehelper为PageHelper类所在包名 -->

<plugin interceptor="com.github.pagehelper.PageHelper">

<!-- 设置数据库类型 Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL六种数据库-->

<property name="dialect" value="mysql"/>

</plugin>

</plugins>

第二步：在代码中使用

1、设置分页信息：

//获取第1页，10条内容，默认查询总数count

PageHelper.startPage(1, 10);

//紧跟着的第一个select方法会被分页

List<Country> list = countryMapper.selectIf(1);

2、取分页信息

//分页后，实际返回的结果list类型是Page<E>，如果想取出分页信息，需要强制转换为Page<E>，

Page<Country> listCountry = (Page<Country>)list;

listCountry.getTotal();

1. 取分页信息的第二种方法

//获取第1页，10条内容，默认查询总数count

PageHelper.startPage(1, 10);

List<Country> list = countryMapper.selectAll();

//用PageInfo对结果进行包装

PageInfo page = new PageInfo(list);

//测试PageInfo全部属性

//PageInfo包含了非常全面的分页属性

assertEquals(1, page.getPageNum());

assertEquals(10, page.getPageSize());

assertEquals(1, page.getStartRow());

assertEquals(10, page.getEndRow());

assertEquals(183, page.getTotal());

assertEquals(19, page.getPages());

assertEquals(1, page.getFirstPage());

assertEquals(8, page.getLastPage());

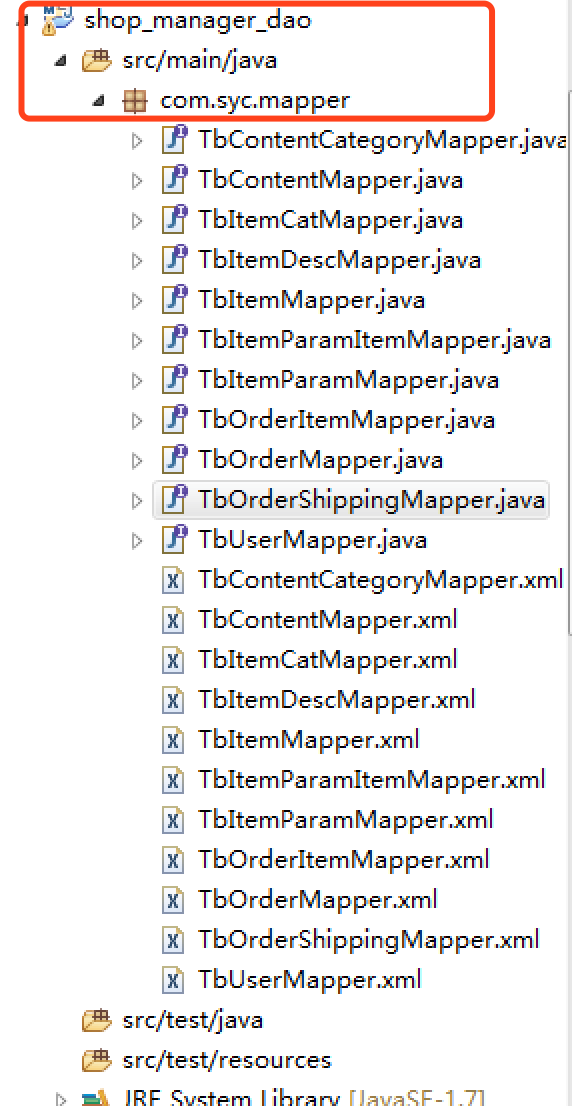
assertEquals(true, page.isFirstPage());

assertEquals(false, page.isLastPage());

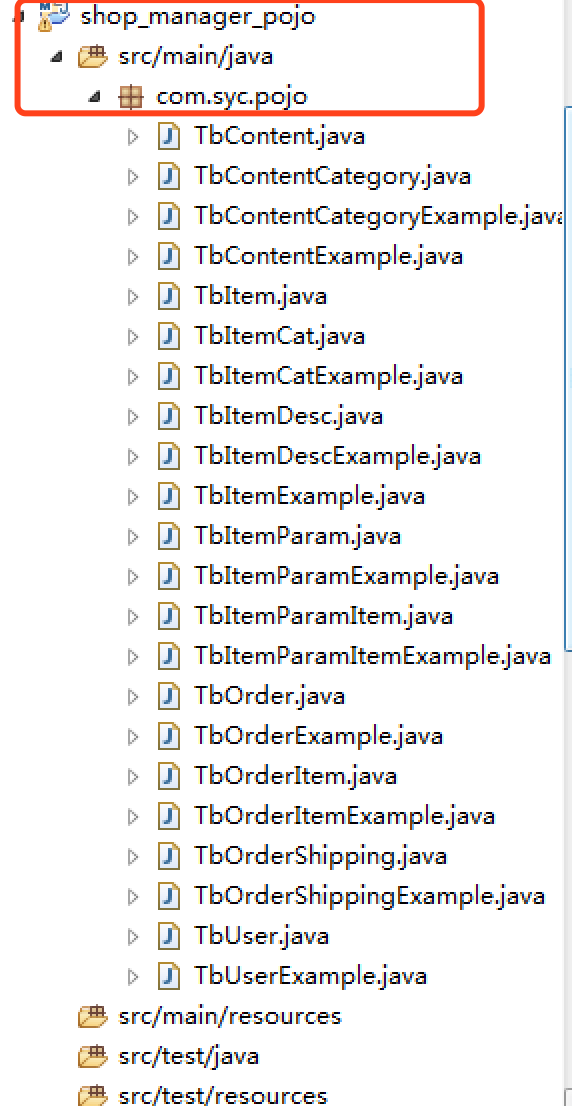
assertEquals(false, page.isHasPreviousPage());

assertEquals(true, page.isHasNextPage());

* 1. **创建Mapper,使用逆向工程生成的mapper文件。**



## POJO项目,tt-manager-pojo项目



* 1. **创建Service类**

|  |
| --- |
| package com.syc.service.impl;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import com.github.pagehelper.PageHelper;  import com.github.pagehelper.PageInfo;  import com.syc.common.pojo.EasyUIDataGridResult;  import com.syc.mapper.TbItemMapper;  import com.syc.pojo.TbItem;  import com.syc.pojo.TbItemExample;  import com.syc.service.ItemService;  @Service  public class ItemServiceImpl implements ItemService {  @Autowired  private TbItemMapper itemMapper;    @Override  public EasyUIDataGridResult findItemsByPage(Integer pageNum, Integer pageSize) {  EasyUIDataGridResult result=new EasyUIDataGridResult();    System.out.println("Num="+pageNum+",Size="+pageSize);    //开始分页  PageHelper.startPage(pageNum, pageSize);    TbItemExample example=new TbItemExample();    List<TbItem> items = itemMapper.selectByExample(example);    //设置行数  result.setRows(items);    //对Page<E>结果进行包装  PageInfo<TbItem> info=new PageInfo<>(items);  long total = info.getTotal();    //设置总数量  result.setTotal(total);    return result;  }  } |

* 1. **Controller**

|  |
| --- |
| package com.syc.controller;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.ResponseBody;  import com.syc.common.pojo.EasyUIDataGridResult;  import com.syc.service.ItemService;  /\*  \* 商品条目控制器  \*/  @Controller  @RequestMapping("/item")  public class ItemController {  @Autowired  private ItemService itemService;  @RequestMapping("/list")  @ResponseBody//RestFul风格,以JSON格式返回.  public EasyUIDataGridResult findItems(Integer page, Integer rows) {    return itemService.findItemsByPage(page, rows);  }  } |

