武汉纺织大学

Web应用开发课程设计

**基于SSM的网站主页**

**学 院： 数学与计算机学院**

**班 级： 物联网11803**

**姓 名： 谭亿龙**

**学 号： 1804280301**

**指导老师： 聂刚**

**成 绩：**

**完成日期： 2020年12月15日**

目 录

[1 需求分析 1](#_Toc59046165)

[1.1 获取导航栏文字 1](#_Toc59046166)

[1.2获取轮播图片 1](#_Toc59046167)

[2 系统设计 1](#_Toc59046168)

[2.1导航栏表 1](#_Toc59046169)

[2.2 轮播图表 2](#_Toc59046170)

[2.3 UML类图（Class Diagram） 2](#_Toc59046171)

[2.4 UML时序图（Sequence Diagram） 3](#_Toc59046172)

[2.4.1 导航栏查询模块 3](#_Toc59046173)

[2.4.2 轮播图查询模块 4](#_Toc59046174)

[3 系统实现 5](#_Toc59046175)

[3.1 项目结构 5](#_Toc59046176)

[3.2 配置文件 5](#_Toc59046177)

[3.2.1 jdbc.properties文件 5](#_Toc59046178)

[3.2.2 log4j2.xml文件 6](#_Toc59046179)

[3.2.3 spring-mybatis.xml文件 7](#_Toc59046180)

[3.2.4 spring-mvc.xml文件 8](#_Toc59046181)

[3.2.5 mybatis.xml文件 9](#_Toc59046182)

[3.2.6 web.xml文件 9](#_Toc59046183)

[3.3 VO类Lunboimage.java和Catalog.Java 11](#_Toc59046184)

[3.4 DAO接口类ICatalogDAO.java和ILunboimageDAO.java 11](#_Toc59046185)

[3.5 接口映射文件Lunboimage.xml和Catalog.xml 12](#_Toc59046186)

[3.6 服务接口类ICatlogService.java和ILunboimageService.java 12](#_Toc59046187)

[3.7 服务接口实现类 CatlogServiceImpl.java和LunboimageServiceImpl.java 12](#_Toc59046188)

[3.8 控制器类CatlogController.java，LunboimageController.java 13](#_Toc59046189)

[4 系统测试 14](#_Toc59046190)

[4.1 目录获取模块 14](#_Toc59046191)

[4.2 轮播图获取模块 14](#_Toc59046192)

[5 系统总结 15](#_Toc59046193)

# 1 需求分析

中大型网站的导航栏通常有多级目录，并且其中的目录可能时常会有变化，同样的轮播图也是一个好看的主页所必不可少的元素，其中的播放的图片也是及时更新的所以我设计了一个网站的主页，实现在数据库中获取主页轮播图片一级图片的超链接和获取主页的导航栏文字一级二级导航栏的文字并显示到网页上。具体要求如下：

## 1.1 获取导航栏文字

进入网站主页之后，从数据库中查询一级，二级导航栏的文字内容以及链接。将其显示到网站上。使用bootstrap框架实现导航栏以及下拉款（dropdown），将其点击显示二级目录更改为鼠标移上去（鼠标的hover/focus事件）之后显示二级目录。实现这个功能还需要在数据库中存储导航栏文字，导航栏列号，导航栏的链接地址等信息。

## 1.2获取轮播图片

进入网站主页之后，从数据库中查询轮播图的信息，获取图片的地址和图片的对应超链接，显示到主页上。使用bootstrap框架实现轮播图，其中每个图片被a标签包含，其图片的地址（src）属性和超链接（href）属性都从数据库中查询。

# 2 系统设计

## 2.1导航栏表

实现导航栏的获取需要一张导航栏表，其字段和内容如下：

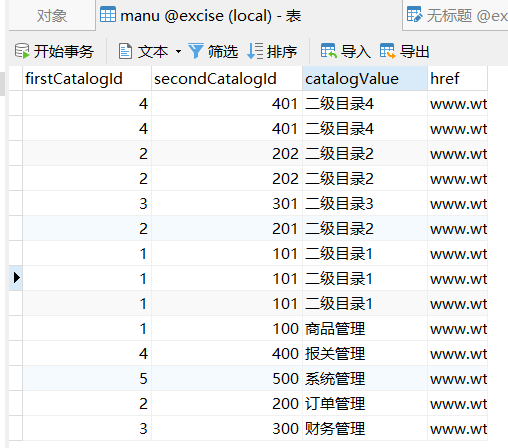


图2-1 导航栏表图

## 2.2 轮播图表

实现轮播图信息的获取需要一张轮播图表，其字段和内容如下

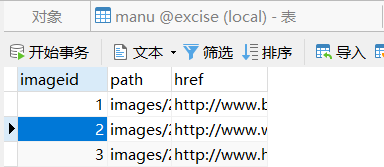


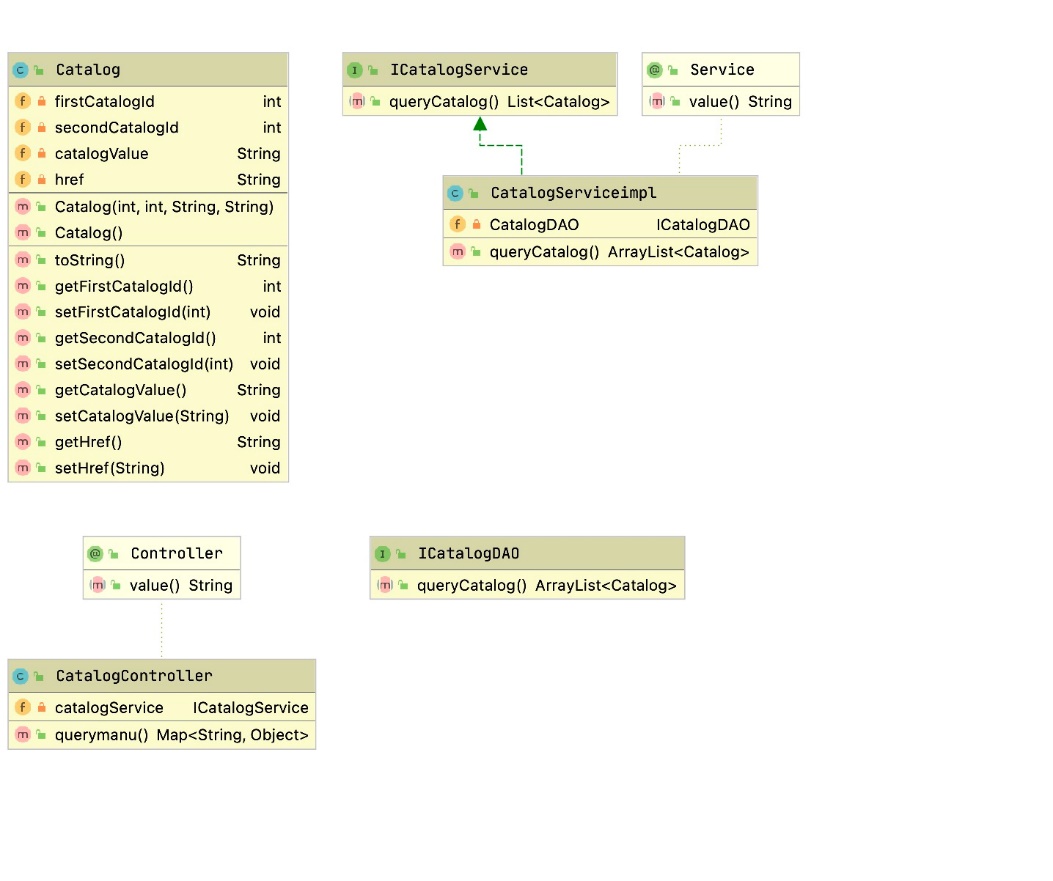
图3-1 轮播图表图

## 2.3 UML类图（Class Diagram）

对于导航栏查询模块，共设计如下6个类。

* VO实体类Catalog：与数据库结构进行映射的类。主要由属性，setter, getter方法组成，VO类中的属性与表中的字段相对应，每一个VO类的对象都表示表中的每一条记录
* DAO接口ICatalogDAO：主要定义操作的接口，定义一系列数据库的原子性操作，例如增删改查（通常称为CRUD）等。
* Service接口ICatalogService：主要定义基本操作业务的接口
* Service实现类CatalogServiceimpl：Service接口的真实实现类，实现业务逻辑操作。

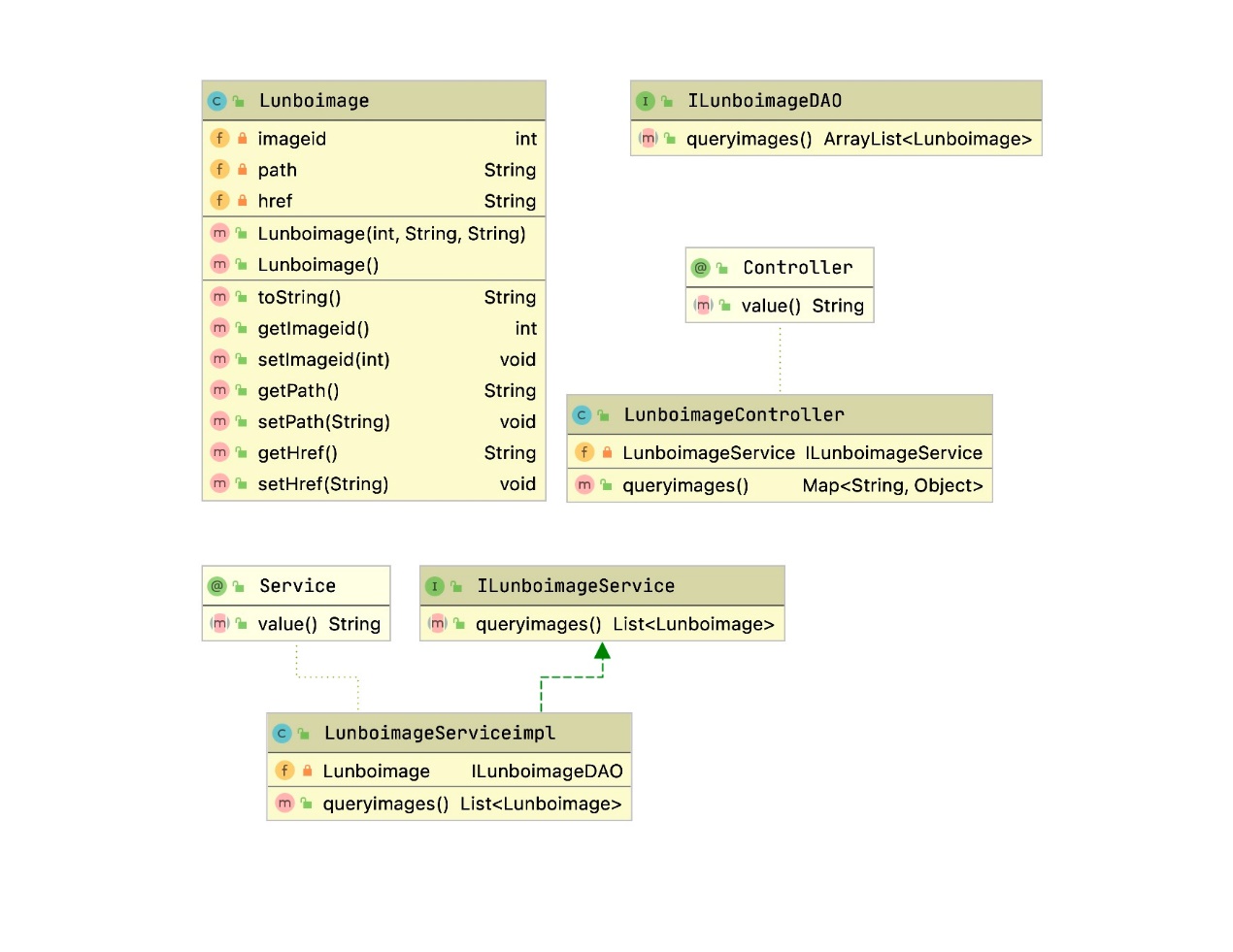
类图如下：



对于轮播图查询模块，共设计如下6个类。

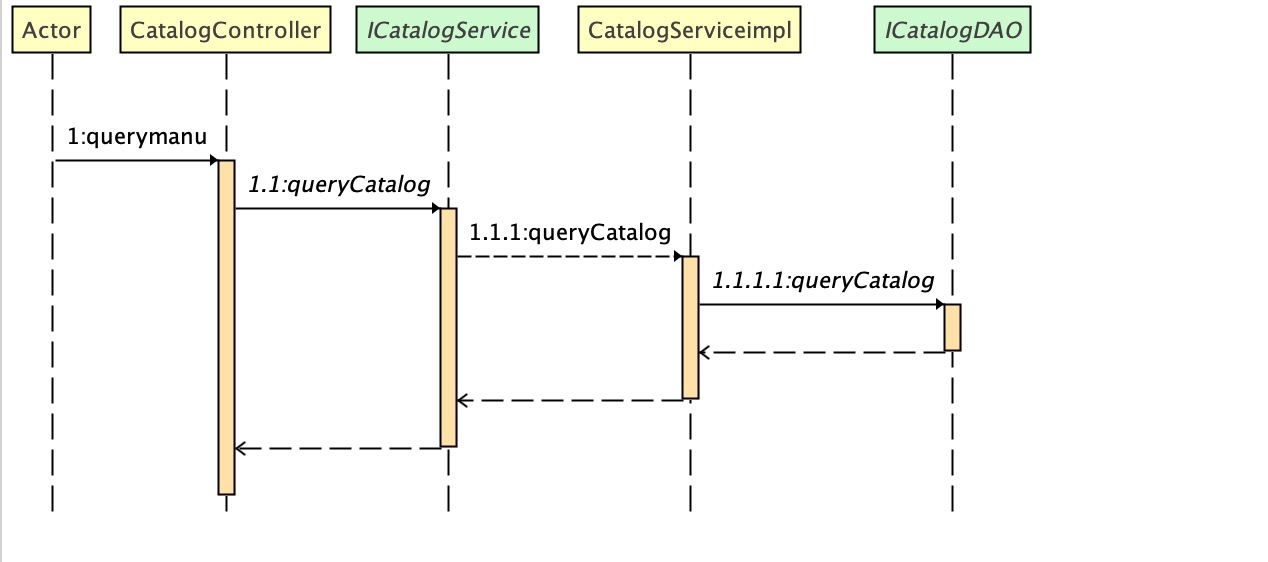
* VO实体类Lunboimage：与数据库结构进行映射的类。主要由属性，setter, getter方法组成，VO类中的属性与表中的字段相对应，每一个VO类的对象都表示表中的每一条记录
* DAO接口ILunboimageDAO：主要定义操作的接口，定义一系列数据库的原子性操作，例如增删改查（通常称为CRUD）等。
* Service接口ILunboimageService：主要定义基本操作业务的接口
* Service实现类LunboimageServiceimpl：Service接口的真实实现类，实现业务逻辑操作。

类图如下：

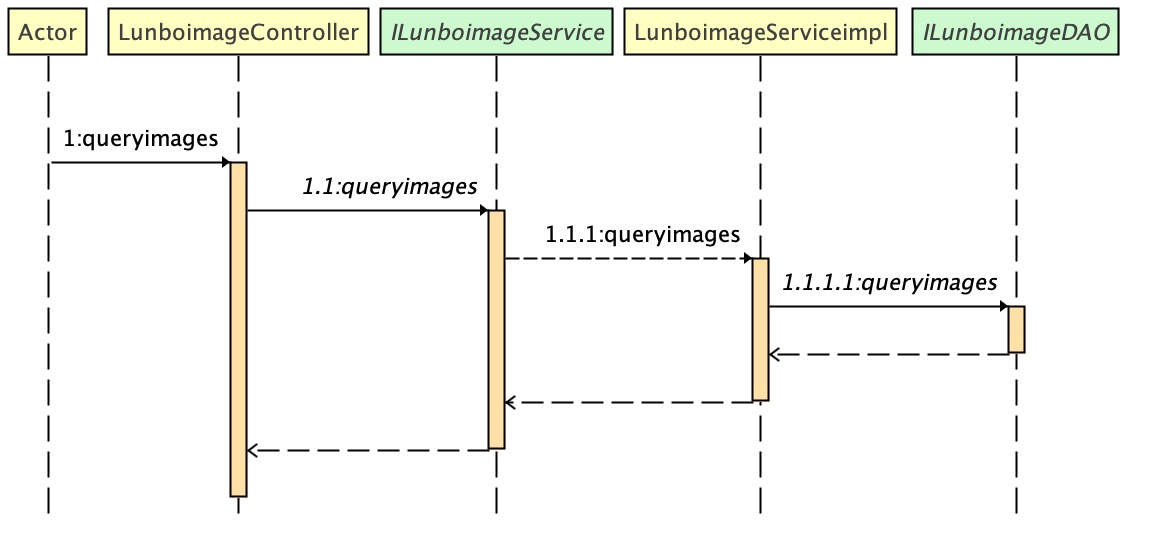


## 2.4 UML时序图（Sequence Diagram）

### 2.4.1 导航栏查询模块

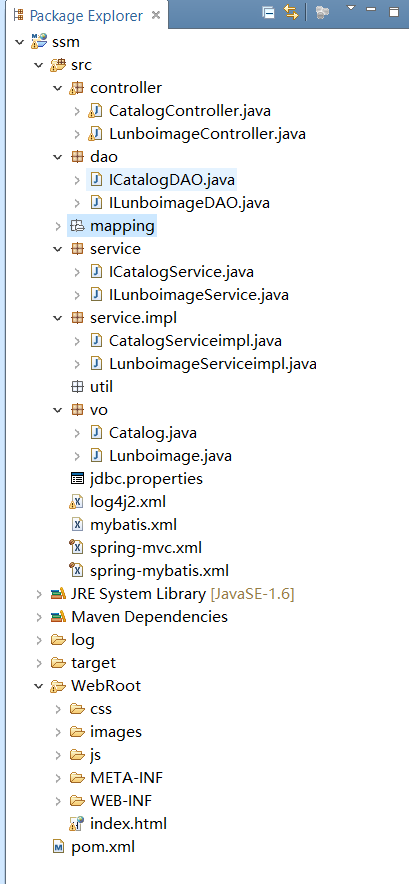


### 2.4.2 轮播图查询模块



# 3 系统实现

## 3.1 项目结构



## 3.2 配置文件

配置文件存放在src目录下

### 3.2.1 jdbc.properties文件

该配置文件主要是以文件形式保存数据库的驱动类名称，连接数据库的URL地址，访问数据库的用户名及对应的密码，程序运行时会读取该文件相关信息，避免硬编码，当相关信息发生变化时，只需修改配置文件而不用修改源代码，增加程序的可扩展型。

driver=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost:3306/excise?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=UTC

username=root

password=123456

#定义初始连接数

initialSize=0

#定义最大连接数

maxActive=20

#定义最小空闲

minIdle=1

#定义最长等待时间

maxWait=60000

#验证连接是否可用，使用的SQL语句

validationQuery =SELECT 1

### 3.2.2 log4j2.xml文件

该文件是log4j2的配置文件

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

<!--日志级别以及优先级排序: OFF > FATAL > ERROR > WARN > INFO > DEBUG > TRACE > ALL -->

<!-- status配置Log4j2启动和加载配置文件时的日志输出登记 -->

<Configuration status="info">

<!--appenders:定义输出内容,输出格式,输出方式,日志保存策略等,常用其下三种标签[console,File,RollingFile] -->

<!--Appender可以理解为日志的输出目的地 -->

<Appenders>

<!--console :控制台输出的配置 -->

<Console name="Console" target="SYSTEM\_OUT">

<!--

%d 输出日志时间点的日期或时间

%t 输出产生该日志事件的线程名

%p 输出优先级，即DEBUG,INFO,WARN,ERROR,FATAL

%F 输出日志消息产生时所在的文件名称

%c 输出所属的类目,通常就是所在类的全名

%L 输出代码中的行号

-->

<PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} %-5p [%c,%L]- %msg%n" />

</Console>

<RollingFile name="RollingFile" filename="log/ZycTest.log"

filepattern="${logPath}/%d{YYYYMMddHHmmss}-fargo.log">

<PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} %-5p [%c,%L] - %msg%n" />

<Policies>

<SizeBasedTriggeringPolicy size="100 MB" />

</Policies>

<DefaultRolloverStrategy max="20" />

</RollingFile>

</Appenders>

<!--定义logger，只有定义了logger并引入的appender，appender才会生效 -->

<Loggers>

<!-- Logger节点用来单独指定日志的形式，可以为通过name属性设置指定包下的class指定不同的日志级别等 ,

可以设置Logger的additivity="false"只在自定义的Appender中进行输出 -->

<logger name="dao" level="debug" additivity="false">

<appender-ref ref="Console" />

</logger>

<!-- Root节点用来指定项目的根日志，如果没有单独指定Logger，那么就会默认使用该Root日志输出 -->

<Root level="info">

<AppenderRef ref="Console" />

<AppenderRef ref="RollingFile" />

</Root>

</Loggers>

</Configuration>

### 3.2.3 spring-mybatis.xml文件

这个文件主要用来完成spring和mybatis的整合的。主要的就是自动扫描，自动注入，配置数据库。

<?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: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/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd">

<!-- 扫描类包，将标注Spring注解的类自动转化Bean，同时完成Bean的注入 -->

<context:component-scan base-package="service" />

<!-- 引入jdbc配置文件 -->

<bean id="propertyConfigurer"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="location" value="classpath:jdbc.properties" />

</bean>

<!-- 数据库连接池 -->

<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"

destroy-method="close">

<property name="driverClassName" value="${driver}" />

<property name="url" value="${url}" />

<property name="username" value="${username}" />

<property name="password" value="${password}" />

<!-- 初始化连接大小 -->

<property name="initialSize" value="${initialSize}"></property>

<!-- 连接池最大数量 -->

<property name="maxActive" value="${maxActive}"></property>

<!-- 连接池最小空闲 -->

<property name="minIdle" value="${minIdle}"></property>

<!-- 获取连接最大等待时间 -->

<property name="maxWait" value="${maxWait}"></property>

<!-- 验证连接是否可用，使用的SQL语句 -->

<property name="validationQuery" value="SELECT 1"></property>

</bean>

<!-- spring和MyBatis整合，不需要mybatis的配置映射文件 -->

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="configLocation" value="classpath:mybatis.xml"></property>

<!-- 自动扫描mapping.xml文件 -->

<property name="mapperLocations" value="classpath:mapping/\*.xml"></property>

</bean>

<!-- DAO接口所在包名，Spring会自动查找其下的类 -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<property name="basePackage" value="dao" />

<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"></property>

</bean>

<!-- (事务管理)transaction manager-->

<bean id="transactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource" />

</bean>

</beans>

### 3.2.4 spring-mvc.xml文件

Spring的配置文件，完成了自动注册DefaultAnnotationHandlerMapping与AnnotationMethodHandlerAdapter 两个bean，和包扫描自动注入bean。

<?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:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

<mvc:annotation-driven />

<!-- 扫描类包，将标注Spring注解的类自动转化Bean，同时完成Bean的注入 -->

<context:component-scan base-package="controller" />

<context:component-scan base-package="service" />

</beans>

### 3.2.5 mybatis.xml文件

这是一个可选配置，spring-mybatis.xml文件中已经完成了mybatis的基本配置，其中指定了mybatis的日志实现为log4j2。

<?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>

<settings>

<setting name="logImpl" value="LOG4J2" />

</settings>

</configuration>

### 3.2.6 web.xml文件

Web项目的必备文件，主要定义了sevlet，过滤器，监听器和应用的hello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\_3\_0.xsd"

id="WebApp\_ID" version="3.0">

<display-name>ssm</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<!-- Spring和mybatis的配置文件 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring-mybatis.xml</param-value>

</context-param>

<!-- log4j2的配置文件 -->

<context-param>

<param-name>log4jConfigLocation</param-name>

<param-value>classpath:log4j2.xml</param-value>

</context-param>

<!-- 编码过滤器 -->

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<async-supported>true</async-supported>

<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>

<!-- Spring监听器 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 防止Spring内存溢出监听器 -->

<listener>

<listener-class>org.springframework.web.util.IntrospectorCleanupListener</listener-class>

</listener>

<!—定义dispatcher Servlet -->

<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-mvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

<async-supported>true</async-supported>

</servlet>

<servlet-mapping>

<servlet-name>SpringMVC</servlet-name>

<url-pattern>\*.do</url-pattern>

<!—定义匹配所有.do请求-->

</servlet-mapping>

</web-app>

## 3.3 VO类Lunboimage.java和Catalog.Java

两个简单对象类，一个用来存放轮播图信息，一个用来存放目录信息

public class Lunboimage {

private int imageid;

private String path;

private String href;

//构造方法、getter,setter，toString方法,暂不列出

}

public class Catalog {

private int firstCatalogId;

private int secondCatalogId;

private String catalogValue;

private String href;

//构造方法、getter,setter，toString方法, 暂不列出

}

## 3.4 DAO接口类ICatalogDAO.java和ILunboimageDAO.java

Dao接口类，写出要实现的各种方法和其参数返回值，例如crud，此处只需要查询即可。

public interface ICatalogDAO {

//查询全部目录

public ArrayList<Catalog> queryCatalog();

}

public interface ILunboimageDAO {

//查询轮播图

public ArrayList<Lunboimage> queryimages();

}

## 3.5 接口映射文件Lunboimage.xml和Catalog.xml

Dao接口的实际实现文件

<?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="dao.ILunboimageDAO">

<!—实现queryimages接口，从数据库中查询图片-->

<select id="queryimages" resultType="vo.Lunboimage">

select \* from t\_images

</select>

</mapper>

<?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="dao.ICatalogDAO">

<!—实现queryCatalog接口，从数据库中查询目录信息-->

<select id="queryCatalog" resultType="vo.Catalog">

select \* from manu

</select>

</mapper>

## 3.6 服务接口类ICatlogService.java和ILunboimageService.java

服务的接口类，定义了需要服务需要实现的方法。

public interface ICatalogService {

public List<Catalog> queryCatalog();

}

public interface ILunboimageService {

public List<Lunboimage> queryimages();

}

## 3.7 服务接口实现类 CatlogServiceImpl.java和LunboimageServiceImpl.java

服务的实现类，实现自己的业务逻辑。主要是返回dao的执行结果。

@Service

public class CatalogServiceimpl implements ICatalogService{

@Autowired

private ICatalogDAO CatalogDAO;

@Override

//简单的服务方法实现，直接返回dao的查询结果

public ArrayList<Catalog> queryCatalog() {

return CatalogDAO.queryCatalog();

}

}

@Service

public class LunboimageServiceimpl implements ILunboimageService {

@Autowired

private ILunboimageDAO Lunboimage;

@Override

//简单的服务方法实现，直接返回dao的查询结果

public List<Lunboimage> queryimages() {

// TODO Auto-generated method stub

return Lunboimage.queryimages();

}

}

## 3.8 控制器类CatlogController.java，LunboimageController.java

控制器类，每一个方法对应一个请求，简单的返回数据库的查询结果。

@Controller

public class CatalogController {

@Autowired

private ICatalogService catalogService;

@RequestMapping(value = "/querymanu.do", method = RequestMethod.POST)

@ResponseBody

//查询目录列表，存到map中，经过框架处理返回给前端

public Map<String, Object> querymanu() {

List<Catalog> list =new ArrayList<Catalog>();

Map<String, Object> map = new HashMap<String, Object>();

list=catalogService.queryCatalog();

map.put("rows", list);

return map;

}

}

@Controller

public class LunboimageController {

@Autowired

private ILunboimageService LunboimageService;

@RequestMapping(value = "/queryimages.do", method = RequestMethod.POST)

@ResponseBody

//查询图片情况列表，存到map中，经过框架处理返回给前端

public Map<String, Object> queryimages() {

List<Lunboimage> list =new ArrayList<Lunboimage>();

Map<String, Object> map = new HashMap<String, Object>();

list=LunboimageService.queryimages();

map.put("rows", list);

return map;

}

}

# 4 系统测试

## 4.1 目录获取模块

如图所示，从数据获取目录列表和列表超链接正常



## 4.2 轮播图获取模块

如图所示，从数据获取轮播图和轮播图超链接正常



# 5 系统总结

我这次选择的一个项目比较的简单，运用ssm框架之后呢也是非常迅速的完成了任务，在其中，我设计的vo对象类一开始是没有包括哪些导航栏已经轮播图的图片以及文字链接的，后来我发现其他的轮播图点击后都会跳转到其他页面，导航栏也是如此，超链接地址也有可能会变化，所以我在这两个vo对象中加入了href超链接字段，能够在获取信息的通知附上其超链接。让后就是一些js的问题，js的语法还是不太熟悉，一直在变敲代码边学习，也是套用了之前的一些代码解决了这次的问题。在bootstrap的下拉框中，需要点击才能显示下拉框，经过查询资料后，使用了css的方法进行该井，将其改进为悬停时候显示下拉框，更贴合我们平时的操作习惯。

系统存在的不足在于只是一个简单的主页，没有实现其他的功能，其次，我写的javascript代码也有点啰嗦，不够精简。

在经过一个学期的web后端开发之后，我从一个小白，也变成了一个能够使用框架进行复杂开发的人。从一开学时候对各种软件环境都难以适应，没有学习java，在经过两三个星期的自学和老师的教学下，也竟然完成了难度很高的第一次项目，虽然其中的实现还是不够完美，用了很多比较迟钝的方法，比如把所有的用户信息从数据库查询到前服务器中保存，再在服务器中遍历用户信息。这样做无疑是不可取的，在看了老师的代码后，我也明白了应该直接从数据库查询。本门课程之后，我把前端学习的不太好的部分全部复习了一遍，而且第一次将我以前学过的数据库进行实际的运用，最后是学习的java的大部分语法操作和了解了整个web开发的过程。

对于课程的建议是:很好，做项目才能学到更多的东西。

以后的发展方向还没有仔细的想过，虽然这个学期学的最好的课程就是web开发，但是我觉得我对偏硬件点的更加感兴趣。

这个主页我是仿造老师的主页完成的，完成度非常高，基本的要求也全部完成了，使用ssm框架简单的完成了开发，自评成绩85分吧。