软件设计模式与体系结构课程设计报告

姓名：汤茂行

学号：161630321

一、课程设计应用概述

> **本系统已在阿里云服务器上部署，地址 https://tomax.xin/wa-poem，开放账号为15195963968 密码为123456**

1、系统描述

系统名称为"Wa poem -admin"，即"我爱写诗"后台管理系统，主要服务于微信小程序“我爱写诗”后台(暂未上线)，具有常规后台管理系统功能，主要包括小程序管理、管理员管理、用户管理、系统管理、数据管理等功能。

2、技术栈

系统采用 B/S 架构，主要使用了如下技术

类别	选择
前端语言	html+css+js+jsp
js框架	jQuery
UI框架	layui
后端语言	java
web框架	spring mvc
服务器容器	tomcat
服务器服务	阿里云
数据库	mysql
版本控制系统	git
集成开发环境	Intellij IDEA
文档描述语言	markdown

3、系统主要业务

- **管理员业务**

包括管理员登录、管理员新增、管理员查看、管理员删除、管理员编辑等

- **用户管理业务**

包括用户查看、用户编辑、用户锁定等

- **小程序数据管理业务**

包括话题管理、诗词管理、推荐管理等

- **后台系统常规业务**

包括日志记录、系统配置等

二、spring框架特点与设计模式分析

1、系统配置方式

本系统采用 `java config` 的形式进行配置，基本回避 `spring` 框架传统的基于 `xml` 配置的方式，更加灵活、便捷。基于 `java config` 的配置方式特点在于可以充分利用注解进行 `bean` 的定义、对象的注入，通过传参、实例化等多种形式实现 `bean` 的构造，而不用写大量冗余的 `xml` 的 `<bean>` 标签，同时直接编写 `java` 语言进行配置的形式可以充分发挥语言本身的优势，具有更大的发挥空间。

框架配置的作用：

- 通过配置，可以进行一定程度的解耦。
- 通过配置，框架可以获取主动执行的能力，而不总是被动执行编写好的代码。
- 通过配置，框架可以完成一些高频代码块的功能封装，实现复用。
- 通过配置，可以提高系统的可维护性，避免大量魔法值的存在。
- 通过配置，more things can be done

2、spring架构与核心概念理解

2.1 主要架构

开源、社区活跃、大量应用,使得spring的功能覆盖逐渐趋于全面，最新版中已有20多个模块，每个版本的发布都会带来许多优秀的特性。

基本可以归为6种模块分类：spring核心容器、Web编程、数据访问、AOP编程、测试业务、Instrumentation。

其核心模块主要包括：Spring Core、Spring Context、Spring Web、Spring Web MVC、Spring Dao、Spring ORM、Spring AOP。

在本系统中，主要应用了Core、Context、Web、Web MVC、Dao五个模块,而Spring的主要业务实现便是基于核心容器，即Core、Context等模块实现的，在这些模块的支持下，spring可以充分实现工厂模式管理，进行依赖注入、控制反转，为java bean的管理提供了一整套完善的流程。而其他模块则或多或少会对一些其他框架进行集成，使得轻量级的框架spring仍然可以很便捷地完成各类业务。

2.2 核心概念

spring的核心概念主要在于Spring比较早地引入了依赖注入与面向切切面编程的特性。

- **依赖注入**

传统java对象的构造方式为 `new` 一个类的实例，从而获得该类的一个对象，这种直接面向编码者的构造方式大量地存在程序的编码过程中，而依赖注入作为一种设计模式，则将对象的构造方式对编码者隐藏，只需要按照规范编写类的构造函数或是setter方法，该类就初步具备了被注入的基础。此时，只需在需要注入的类或接口上加上注入类的注解，就可以跳过对象的构造过程。

依赖注入实现了控制反转，利用注解，不再由编码者直接指定接口或是抽象类的实现类，而是由系统主动的判断具体使用哪种实现方式，一方面可以实现各个工作组件间的松耦合，更加灵活地组织功能组件，另一方面可以减轻编码的工作量。

依赖注入的背后是由spring提供的一套完整的工作流程，包括bean工厂的初始化、各个bean的创建，这个过程也是链式的，如同一个真正的工厂，部件一次装配进生产线中，最终产出业务。

- **面向切面编程**

在面向切面编程以前，实际的业务代码中往往充斥着大量与业务本身无关的代码，这些代码的功能通常是关注业务代码，而不同业务中的这些闲散代码具有大量的相似处，或者本身即属于一类代码，这样的代码很常见，包括日志、事务管理、安全控制等等。而面向切面编程则有效地解决了这种问题，通过将这些无关业务代码封装、再使用声明的方式将封装好的功能模块应用于需要影响的模块中去，大大提高这些代码的复用程度，另一方面也极大地减轻了维护该类代码地难度。

面向切面的实现应用了代理模式，通过代理，将事务代码环绕在业务代码的执行前、执行过程后，业务本身并不关心是否执行了该事务代码，但是实际上系统灵活地完成了事务代码地应用。

总结而言，AOP帮助将大量重复的业务无关代码分离，并且组件化。

2.3 spring框架的一些优点

可以灵活地集成不同的框架实现功能的扩展。

具有功能强大的容器，并提供应用多种上下文，可以赋予POJO强大的功能。

通过模板消除样板代码，如JdbcTemplate，避免了大量重复的JDBC基本代码、异常处理，并提供较好的事务管理。

还有很多优点 ...

3、MVC及其他基本设计模式在spring中的体现

3.1 MVC

- **基本理解**

Model，模型层，在本系统中体现为dao层、service层以及基于POJO的实体类entity包，主要负责系统中的业务逻辑、数据包装和持久化，是系统功能的基础与支撑。

View，视图层，本系统主要由html+css+js+jsp完成视图层的渲染，通过异步请求向后台请求数据,并在控制器的响应下取回数据渲染界面，更加高效，并有助于前后端分离，回避了spring的ViewResolver只是渲染出jsp界面的限制，利用HttpMessageConverter实现用跨语言的json数据完成前后端的数据交互。

Controller，控制器层，本系统将控制器放入controller包中，通过@Controller进行注解，使用@RequestMapping注解将Controller中的方法接口暴露给客户端，实现View与Model间交互的桥梁，View通过异步请求访问Controller，Controller将请求分发到具体的业务层进行处理，包装好数据模型后再经过Controller返回到视图层进行渲染。

- **特点**

一个模型可以为多个视图提供服务，减少重复的工作并且使得业务更加容易维护。

单一视图中可以加载不同模型，模型数据的处理与视图显示没有直接关系，使得模型可以为各种视图渲染技术服务，而不需要专门针对某一种视图编程，更加面向服务。

通过控制器层进行系统的解耦，增强面向服务的能力。同时利用控制器层可以方便地实现一些事务控制功能的切入。

- **综述**

> 以加载管理员列表为例

第一步，进入管理员列表页，发送请求管理员列表的请求，携带page，limit作为分页参数。

第二步，spring框架解析请求，通过请求路径找到相应的controller以及其中响应该请求的方法，这时，框架根据该方法的参数从请求中拿到请求参数，控制传参并且调用该方法，在响应请求的controller的控制下，请求管理员列表数据交到了IAdministratorService进行处理。

第三步，在业务层中，通过调用AdministratorDao中的查询方法，使用分页参数并通过sql语句进行相应的调用。

第四步，在dao层中，利用封装了JDBCtemplate的helper简化JDBC操作，实现对Mysql本系统数据库的管理员表的访问，并通过查询语句获取数据，在相关库的帮助下，表中数据被映射到该表对应的管理员实体类上去，从而获取一个管理员对象列表，最终返回给业务层。

第五步，由业务层将管理员列表进一步包装返回给控制器层。

第六步，由控制器层通过response响应结果返回到请求该数据的管理员界面。

第七步，管理员界面发送出获取管理员列表的异步请求在响应成功后进行回调，加载出管理员列表并显示到页面中。

3.2 基本设计模式

- 工厂类模式

spring中的BeanFactory及通过工厂完成对象的创建，在web容器启动的过程中，对配置的包路径进行扫描，获取注解为Component的或是由继承自Component注解的类，通过工厂按照依赖一次创建这些类的实例。

- 单例与原型模式

由spring工厂创建的对象默认是单例，使用@Autowired或是@Resource等注解将这些bean实例注入到相应的成员中去，可以不用直接new对象，在某些业务的需求下，有时也会需要通过@Scope注解将对象的创建为原型，通过Object类的clone方法获取新的对象，使得对象具备与单例不一样的生命周期。

- 代理模式

面向切面功能的最关键实现就是通过动态代理，使用相应的注解来增强被代理的业务的功能。那么在调用者调用目标对象时，就可以由代理类完成切面的功能。

- 模板方法模式

spring中模板方法最典型的便是JDBCtemplate，将编写JDBC产生的大量样板代码对编码者隐藏，简化编程过程同时又增强处理能力。

- 观察者模式

mvc中实现多个视图可以使用一个模型进行渲染便是应用观察者模式的基础，基于观察者模式，模型数据的改变时，加载该模型数据的视图都会随之调整，更多的应用是listener、回调函数。

三、系统功能展示

> 系统主要功能的展示

1、系统功能

1.1 个人信息管理

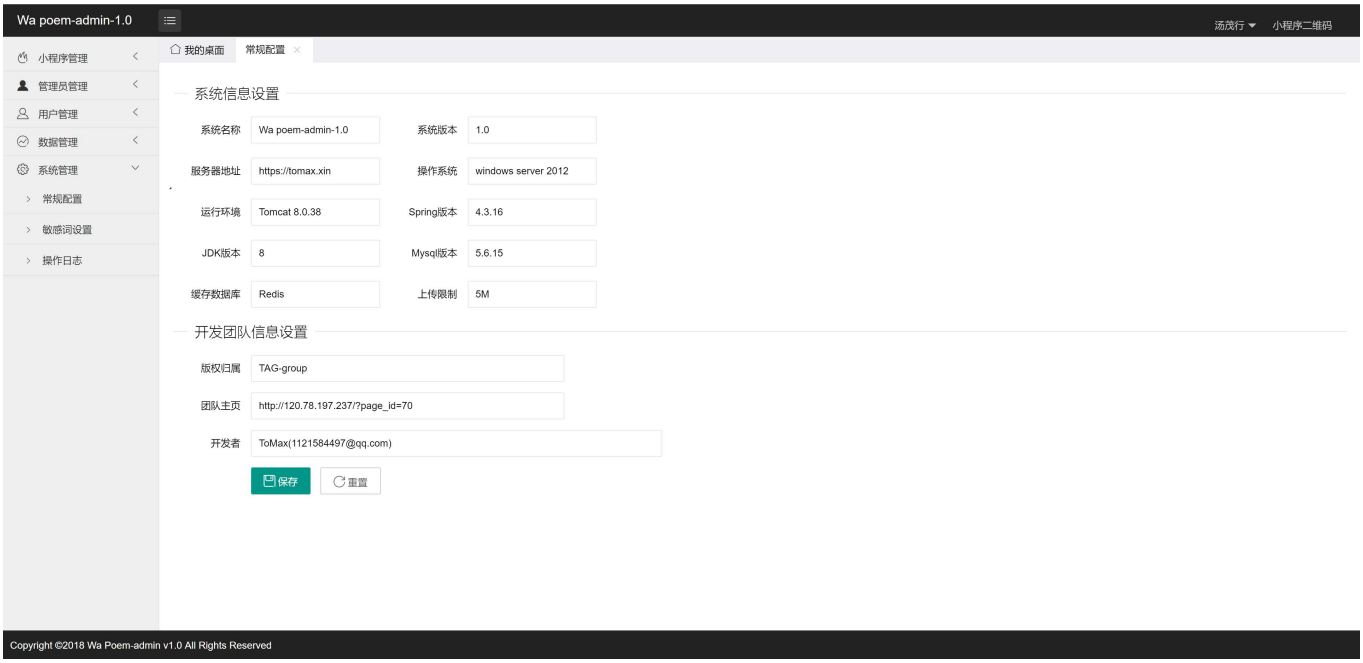
管理员个人信息管理


```
//主要业务代码
@Override
public ResultCause updateAdministrator(AdministratorEntity admin) {
    String sql = "update administrator set "+
            "name = ?, "+
            "id_number = ?, "+
            "password = ?, "+
            "phone = ?, "+
            "email = ?, "+
            "sex = ? "+
            "where id = ?";
```

```
    return administratorDao.updateAdministrator(sql, new Object[]{
                admin.getName(), admin.getIdNumber(), admin.getPassword(),
                admin.getPhone(), admin.getEmail(), admin.getSex(),
                admin.getId()
            }) ?
            new ResultCause(ConstResult.RESULT_SUCCESS_CODE, "更新成功") :
            new ResultCause(ConstResult.RESULT_FAIL_CODE, "更新失败");
}
```

1.2 系统配置

用于配置系统界面元素

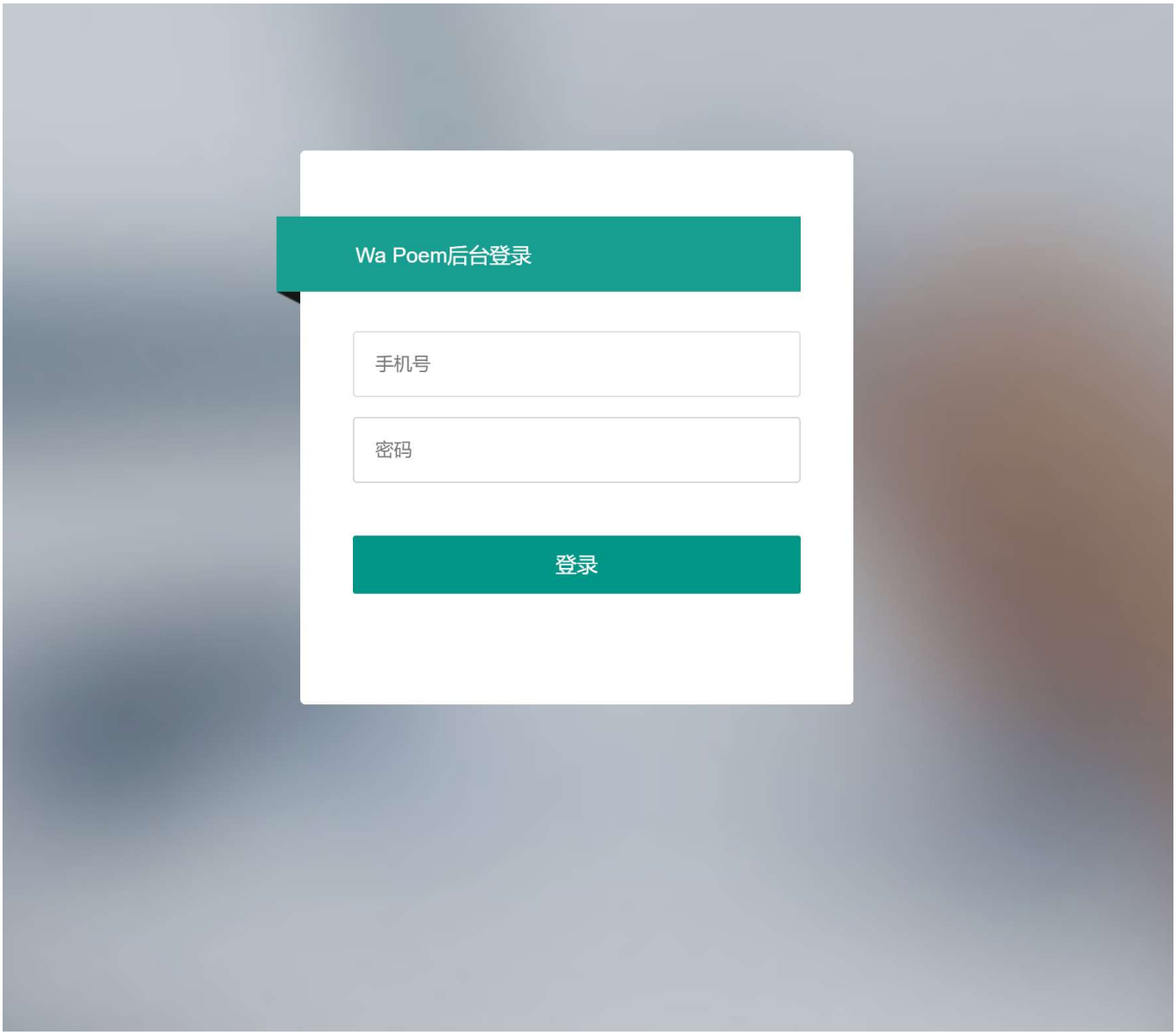



```
// 主要接口
@RequestMapping(value = "/setting/getElements", method = {RequestMethod.GET,RequestMethod.POST})
public @ResponseBody
AdminElementEntity getElements(){
    return adminSettingService.getAdminSystemElements();
}

@RequestMapping(value = "/updateAdministrator", method = {RequestMethod.GET, RequestMethod.POST})
public @ResponseBody
ResultCause updateAdministrator(AdministratorEntity admin){
    return administratorService.updateAdministrator(admin);
}
```

2、管理员业务

2.1 登录

用于管理员登录

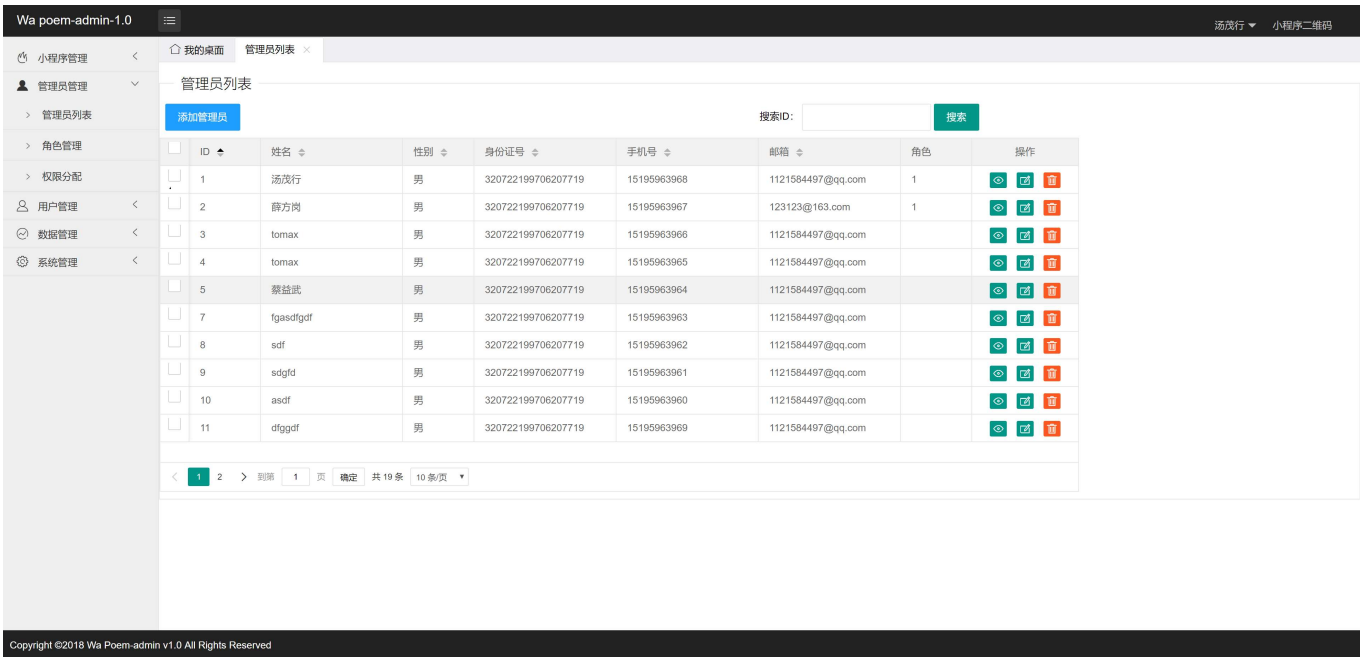



```
// 控制器类接口
@RequestMapping(value = "/login", method = {RequestMethod.POST, RequestMethod.GET})
public @ResponseBody
ResultCause login(String phone, String password){
    AdministratorEntity administratorEntity  =  administratorService.login(phone,password);
    HttpSession session = request.getSession();
    session.removeAttribute("user_info");
    if (administratorEntity != null){
        administratorEntity.setPassword("");
        session.setAttribute("user_info",administratorEntity);
        return new ResultCause(ConstResult.RESULT_SUCCESS_CODE,"登录成功");
    }
    return new ResultCause(ConstResult.RESULT_FAIL_CODE, "用户名或密码错误");
}

// 业务代码
@Override
public AdministratorEntity login(String phone, String password) {
    String sql = "select * from administrator where phone = ?";
    List<AdministratorEntity> administratorEntities = administratorDao.listAdministrators(sql,new
Object[]{phone});
    return administratorEntities != null && administratorEntities.size() > 0 ?
            administratorEntities.get(0) : null;
```

```
}
```

2.2 管理员管理

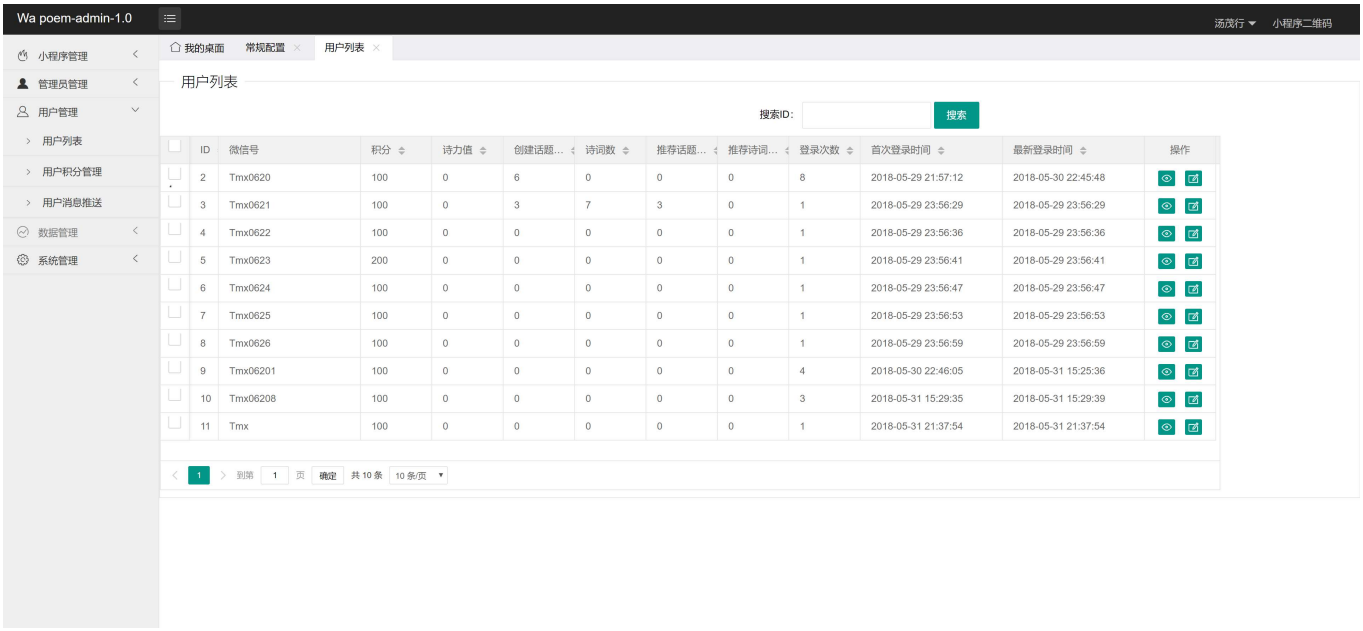
用于管理员添加、查看、编辑、删除




```
// 主要接口

@RequestMapping(value = "/administratorList", method = {RequestMethod.GET, RequestMethod.POST})
public @ResponseBody
ResponseEntity getAdministratorList(int page, int limit){
    return new ResponseEntity<AdministratorEntity>(
            ResponseEntity.GET_DATA_SUCCESS_CODE,
            "获取数据成功",
            administratorService.getAdministratorCount(),
            administratorService.listAdministrator(page, limit)
    );
}

@RequestMapping(value = "/administratorInfo", method = {RequestMethod.GET, RequestMethod.POST})
public @ResponseBody
AdministratorEntity getAdministratorInfo(int id){
    return administratorService.getAdministratorInfo(id);
}

@RequestMapping(value = "/addAdministrator", method = {RequestMethod.GET, RequestMethod.POST})
public @ResponseBody
ResultCause addNewAdministrator(AdministratorEntity admin){
    return administratorService.addNewAdministrator(admin);
}

@RequestMapping(value = "/updateAdministrator", method = {RequestMethod.GET, RequestMethod.POST})
public @ResponseBody
ResultCause updateAdministrator(AdministratorEntity admin){
    return administratorService.updateAdministrator(admin);
}
```

3、用户业务

主要包括用户信息的展示、编辑以及用户锁定

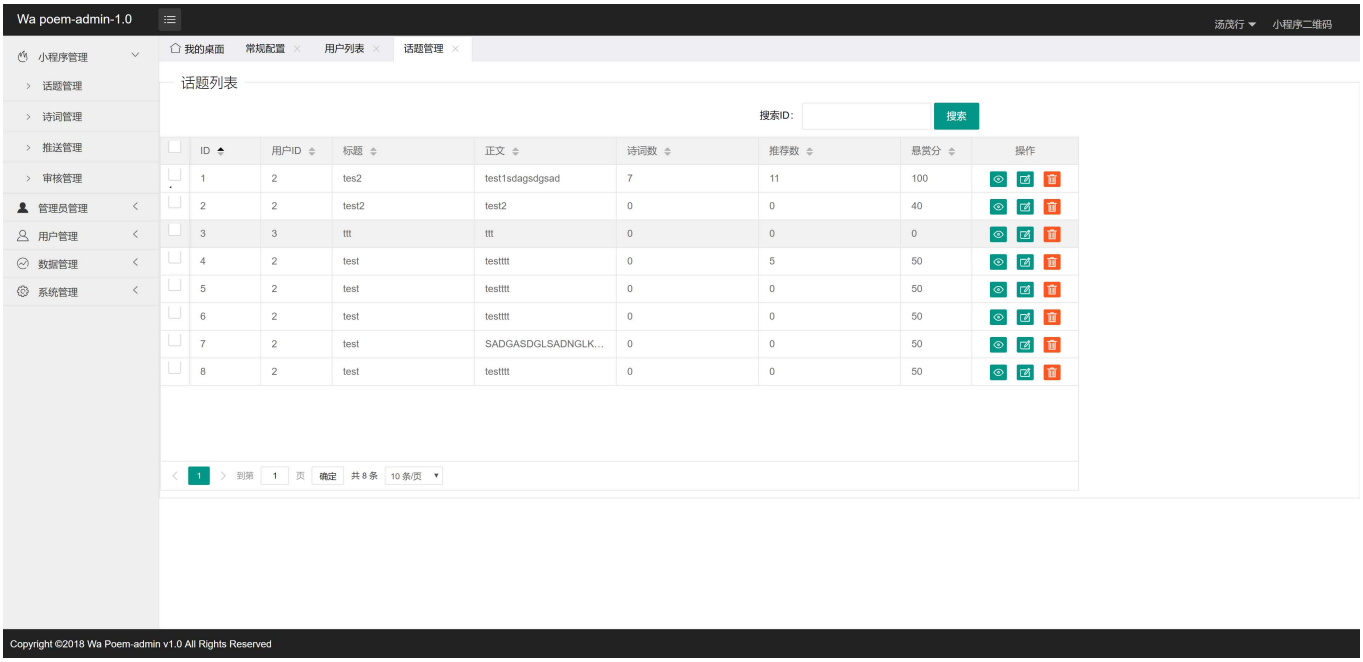



```
//主要接口

@RequestMapping(value = "/poemList", method = {RequestMethod.POST, RequestMethod.GET})
public @ResponseBody
ResponseEntity getPoemList(int page, int limit){
    return new ResponseEntity<PoemEntity>(
            ResponseEntity.GET_DATA_SUCCESS_CODE,
            "获取用户数据成功",
            poemService.getPoemNum(),
            poemService.getPoemList(page, limit)
    );
}
@RequestMapping(value = "/user/update", method = {RequestMethod.POST, RequestMethod.GET})
public @ResponseBody
ResultCause updateUser(UserEntity user){
    return userService.updateUserInfo(user);
}
```

4. 小程序相关业务

4.1 话题业务

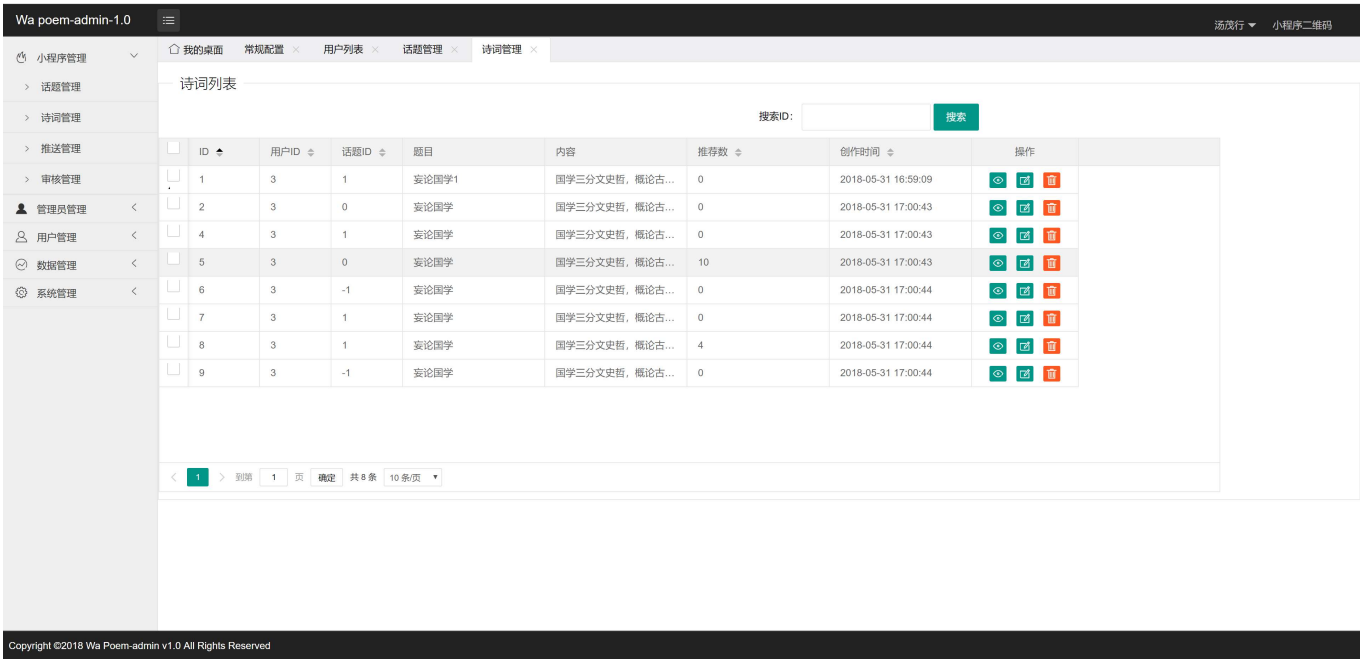
小程序中话题的查看、编辑、删除




```
// 主要接口
@RequestMapping(value = "/topicList", method = {RequestMethod.POST, RequestMethod.GET})
public @ResponseBody
ResponseEntity getTopicList(int page, int limit){
    return new ResponseEntity<TopicEntity>(
            ResponseEntity.GET_DATA_SUCCESS_CODE,
            "获取用户数据成功",
            topicService.getTopicNum(),
            topicService.getTopicList(page, limit)
    );
}
@RequestMapping(value = "/topic/update", method = {RequestMethod.POST, RequestMethod.GET})
public @ResponseBody
ResultCause updateTopic(TopicEntity topic){
    return topicService.updateTopic(topic);
}
```

4.2 诗词业务

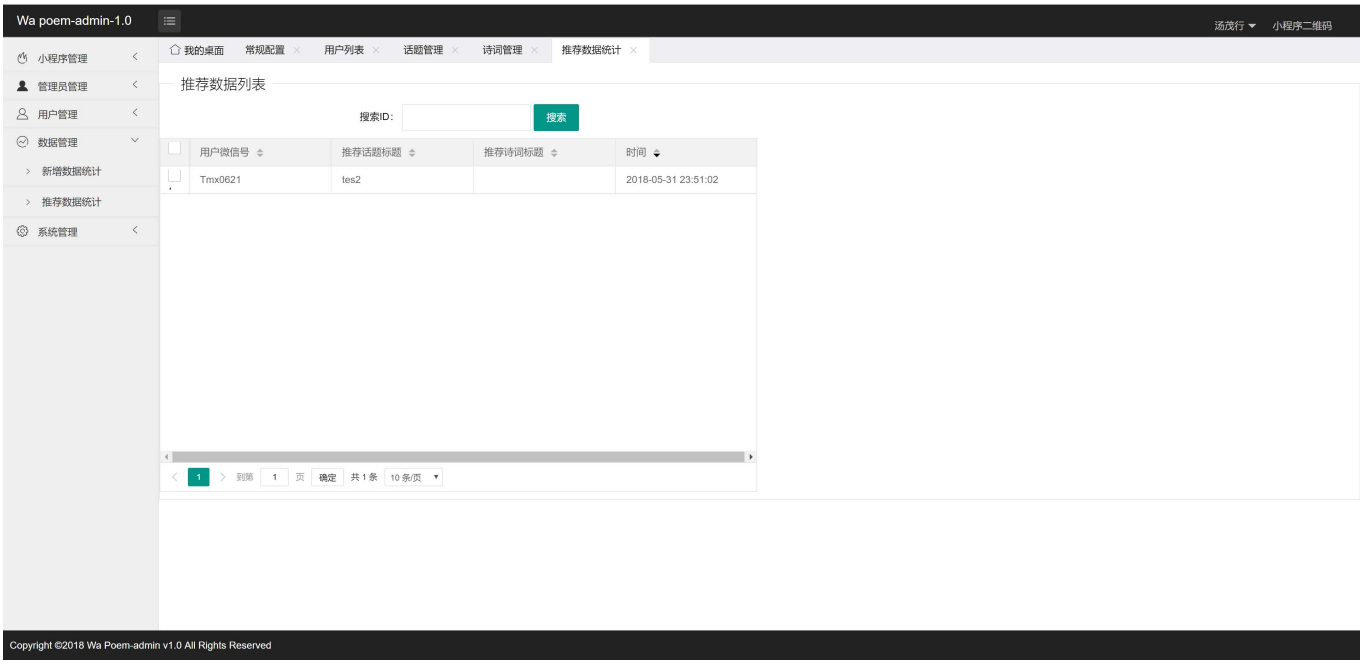
小程序中诗词的管理




```
//主要接口
@RequestMapping(value = "/poemList", method = {RequestMethod.POST, RequestMethod.GET})
public @ResponseBody
ResponseEntity getPoemList(int page, int limit){
    return new ResponseEntity<PoemEntity>(
            ResponseEntity.GET_DATA_SUCCESS_CODE,
            "获取用户数据成功",
            poemService.getPoemNum(),
            poemService.getPoemList(page, limit)
    );
}
@RequestMapping(value = "/poem/update", method = {RequestMethod.POST,RequestMethod.GET})
public @ResponseBody
ResultCause updatePoem(PoemEntity poem){
    return poemService.updatePoem(poem);
}
```

4.3 推荐业务

小程序中推荐信息的展示




```
//主要接口
@RequestMapping(value = "/recomList", method = {RequestMethod.POST, RequestMethod.GET})
public @ResponseBody
ResponseEntity getRecomList(int page, int limit){
    return new ResponseEntity<RecommendEntity>(
            ResponseEntity.GET_DATA_SUCCESS_CODE,
            "获取推荐数据成功",
            recommendService.getRecommendNum(),
            recommendService.getRecommendList(page, limit)
    );
}
```

四、总结

通过这次课程设计，更加深入的学习了web开发，并从设计模式的角度去理解框架本身的设计，而不是像之前只是单纯的会使用框架，只知其形，不明其意。虽然对于spring的原理及框架背后整个的运作原理还处于一个相对浅薄的认知层面，但是相信经过更加深入的了解，会逐步掌握。

框架的价值不仅在于它极大地简化了开发过程，使得很多复杂业务不必从底层开始做起，框架的设计理念、实现方式更加具备价值。下一步的计划是从阅读框架开始，从源码层面理解spring优秀特性的实现以及其本身的架构、对于各类设计模式的组织与实现，从而提升自己的能力。