

**信息工程学院 14 届本科毕业设计**

**设计题目 基于J2EE的Web系统的设计与实现**

**姓 名**： 蒋磊

**学 号**： 2014110514

**专 业**： 软件工程

**班 级**： 14软件工程1班

**指 导 教 师**： 张锏

2018年 5 月

# 摘要

云收藏系统是一个可以在线随时随地的简单收藏的一个网站。可以在这个网站上分类整理收藏的网站或者文章，也可以查看别人都收藏了些什么。可以从浏览器收藏夹导入到我们的网站，也可以从我们的网站导出去做备份。根据共同的收藏可以找出我们相互兴趣的人。

本文以云收藏系统为基础，并结合实际需求，详细的叙述了云收藏系统的设计，开发的全过程。本系统的的主要功能包括收藏网站，评论，点赞，发私信等功能。在开发过程中主要采用了MVC模式的分层思想。

本系统基于B/S架构，采用java语言，以MySql为数据库，并使用Tomcat服务器。结合最新的功能强大的Springboot框架，使用spring Data Jpa作持久层操作，从而使得整个系统更加的稳定，快速与高效。而使用springboot就使开发过程更加的快捷与轻便，避免了使用spring的大量的配置文件的编写。前端方面主要采用Html,css,jQury。

关键词：云收藏， B/S， springboot

# abstract

The Cloud collection system is a simple collection of web sites that can be online anytime, anywhere. You can sort out your favorite websites or articles on this site, or you can see what other people have collected. You can import from your browser favorites to our website, or you can export from our website to do backup. We can find people who are interested in each other according to a common collection. Based on the cloud collection system and combined with the actual needs, this paper describes the design and development of the cloud collection system in detail.

This system's main function includes collects the website, the comment, the point praise, sends the DMS and so on function. In the process of development, we mainly adopt the layered idea of MVC pattern.

This system is based on B/S architecture, adopts Java language, takes MySQL as database and uses Tomcat server. Combined with the latest powerful springboot framework, the spring Data JPA is used as a persistence layer to make the entire system more stable, fast and efficient. Using Springboot makes the development process faster and lighter, avoiding the large number of configuration files that use spring. The front end mainly uses Html,css,jqury.

**Key words**: Cloud Collection, B/S, springboot.

# 前言

J2EE (Java 2Platform, Enterprise Edition )是使用 Java 技术开发企业级应用的一种事实上的工业 标准。 目前, Java 平台有三个版本:适用于小型设备和智能卡的 J2ME(Java2Platform , Micro Edition) 、 适用于桌面系统的 J2SE (Java 2Platform, Standard Edition ) 和适用于企业级应用的 J2EE 。 Sun 推出 J2EE 的目的是为了克服传统 C/S(Client/Server)模式的弊病,迎合 B/S(Browser/Server)架构的潮流,为应 用 Java 技术开发服务器端应用提供一个平台独立的、可移植的、多用户的、安全的和基于标准的企 业级平台,从而简化企业应用的开发、管理和部署。

现在的市场越来越重视 Web 服务在互联网中的作用, 新一代的浏览器是以基于服务的模式 为主要特征。在这种新的模式中,Web 服务迅速取代了单一的、宿于肥客户机的,并要花费很大精力 维护和管理的软件应用。 J2EE 正是降低了开发这种中间层服务的成本和复杂程度,因而使得服务可 以被快速地展开,具有高可用性、安全性、可依赖性以及可扩展性。 J2EE 平台提供了一个多层结构 的分布式的应用程序模型,该模型具有重用组件的能力、基于扩展标记语言 (XML)的数据交换、统一 的安全模式和灵活的事务控制。 不仅可以比以前更快地发表对市场的新的解决方案, 而且独立于平台 的基于组件的 J2EE 解决方案不再受任何提供商的产品和应用程序编程界面 (APIs)的限制。

随着互联网技术的高速发展和日益的普及, 互联网用户迅速增加,越来越多的人在网上查找资料，浏览信息。 使得用户对互联网的寄托发生了翻天覆地的变化, 用户不甘于只单单的使用浏览器自带的收藏功能, 更希望能够有好的收藏工具来维护自己所收藏关注的一些资料与网页。本设计就是想解决这一问题。采用基于 J2EE 架构设计的基于 Web 云收藏系统将可以最大限度保留现有 IT 资源,实现高效开发;支持异构环境,且系统模块 具有伸缩性和稳定的可用性、扩展性。

# 第1章 相关技术与配置

## 1.1 IDEA

IDEA 全称IntelliJ IDEA，是[java语言](https://baike.baidu.com/item/java%E8%AF%AD%E8%A8%80)开发的集成环境，IntelliJ在业界被公认为最好的java开发工具之一，尤其在智能代码助手、代码自动提示、重构、[J2EE](https://baike.baidu.com/item/J2EE)支持、[Ant](https://baike.baidu.com/item/Ant/16001870)、[JUnit](https://baike.baidu.com/item/JUnit)、[CVS](https://baike.baidu.com/item/CVS)整合、代码审查、 创新的[GUI设计](https://baike.baidu.com/item/GUI%E8%AE%BE%E8%AE%A1)等方面的功能可以说是超常的。IDEA是JetBrains公司的产品，这家公司总部位于[捷克共和国](https://baike.baidu.com/item/%E6%8D%B7%E5%85%8B%E5%85%B1%E5%92%8C%E5%9B%BD)的首都[布拉格](https://baike.baidu.com/item/%E5%B8%83%E6%8B%89%E6%A0%BC/632)，开发人员以严谨著称的东欧[程序员](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E5%91%98/62748)为主。

Idea 在2015年官方主页这样介绍自己 ： Excel at enterprise, mobile and web development with Java, Scala and Groovy, with all the latest modern technologies and frameworks available out of the box. 简明翻译：IntelliJ IDEA 主要用于支持 Java、Scala、Groovy 等语言的开发工具，同时具备支持目前主流的技术和框架，擅长于企业应用、移动应用和 Web 应用的开发。IntelliJ IDEA 对自己的定义是很清晰的，对于新人来讲可能还不太理解，可能还会有误会，认为它博而不精，但是对于老用户来讲应该是非常认可上面这句话的。

如果用一句话来形容 IntelliJ IDEA，我会说：**IntelliJ IDEA 是目前所有 IDE 中最具备沉浸式的 IDE，没有之一**。

## 1.2 Navicat

Navicat 是一套快速、可靠并价格相宜的数据库管理工具，专为简化数据库的管理及降低系统管理成本而设。它的设计符合[数据库管理员](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E5%91%98)、开发人员及中小企业的需要。Navicat 是以直觉化的[图形用户界面](https://baike.baidu.com/item/%E5%9B%BE%E5%BD%A2%E7%94%A8%E6%88%B7%E7%95%8C%E9%9D%A2)而建的，让你可以以安全并且简单的方式创建、组织、访问并共用信息。Navicat提供多达 7 种语言供客户选择，被公认为全球最受欢迎的数据库前端用户界面工具。

它可以用来对本机或远程的 MySQL、SQL Server、SQLite、Oracle 及 PostgreSQL 数据库进行管理及开发。

Navicat的功能足以符合专业开发人员的所有需求，而且对[数据库服务器](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E6%9C%8D%E5%8A%A1%E5%99%A8)的新手来说又相当容易学习。有了极完备的图形用户界面 (GUI)，Navicat 让你可以以安全且简单的方法创建、组织、访问和共享信息。

Navicat适用于三种平台 - Microsoft Windows、Mac OS X 及[Linux](https://baike.baidu.com/item/Linux)。它可以让用户连接到任何本机或远程服务器、提供一些实用的数据库工具如数据模型、数据传输、[数据同步](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%90%8C%E6%AD%A5)、结构同步、导入、导出、备份、还原、报表创建工具及计划以协助管理数据。

## 1.3 SSH框架

### 1.3.1 springboot

Spring Boot 是由 Pivotal 团队提供的全新框架，其设计目的是用来简化新 Spring 应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。通过这种方式，Spring Boot 致力于在蓬勃发展的快速应用开发领域成为领导者。

特点：

1. 创建独立的Spring应用程序

2. 嵌入的Tomcat，无需部署WAR文件

3. 简化Maven配置

4. 自动配置Spring

5. 提供生产就绪型功能，如指标，健康检查和外部配置

6. 绝对没有代码生成和对XML没有要求配置[1]

### 1.3.2 springMVC

MVC是一个设计模式，它强制性的使应用程序的输入、处理和输出分开。使用MVC应用程序被分成三个核心部件：模型(Model)、视图(View)、控制器(Controller)。它们各自处理自己的任务。

优点如下：

1. 低耦合性 视图层和业务层分离：这样就允许更改视图层代码而不用重新编译模型和控制器代码，同样，一个应用的业务流程或者业务规则的改变只需要改动MVC的模型层即可。因为模型与控制器和视图相分离，所以很容易改变应用程序的数据层和业务规则。

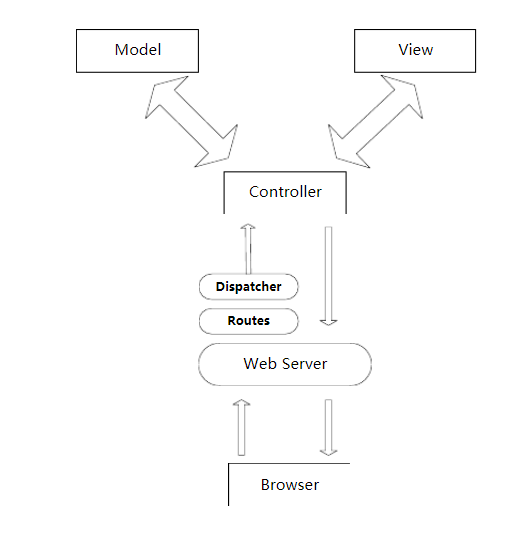
2. 高重用性和可适用性 随着技术的不断进步，现在需要用越来越多的方式来访问应用程序。MVC模式允许你使用各种不同样式的视图来访问同一个服务器端的代码。它包括任何WEB（HTTP）浏览器或者无线浏览器（wap），比如，用户可以通过电脑也可通过手机来订购某样产品，虽然订购的方式不一样，但处理订购产品的方式是一样的。由于模型返回的数据没有进行格式化，所以同样的构件能被不同的界面使用。例如，很多数据可能用HTML来表示，但是也有可能用WAP来表示，而这些表示所需要的仅令是改变视图层的实现方式，而控制层和模型层无需做任何改变。

3.较低的生命周期 MVC使降低开发和维护用户接口的技术含量成为可能。

4.快速的部署 使用MVC模式使开发时间得到相当大的缩减，它使程序员集中精力于业务逻辑，界面程序员（HTML开发人员）集中精力于表现形式上。

5.可维护性 分熟视图层和业务逻辑层也使得WEB应用更易于维护和修改。

6.有利于软件工程化管理 由于不同的层各司其职，每一层不同的应用具有某些相同的特征，有利于通过工程化、工具化管理程序代码



MVC模型图

### 1.3.3 Spring-Data-Jpa(Hibernate)

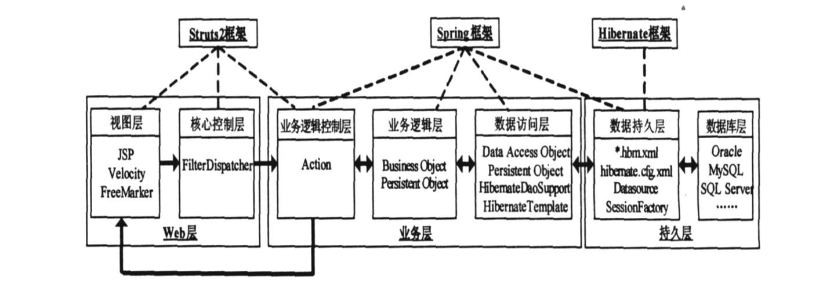
Hibernate是一个开放源代码的对象关系映射框架，它对JDBC进行了非常轻量级的对象封装，使得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。 Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用，最具革命意义的是，Hibernate可以在应用EJB的J2EE架构中取代CMP，完成数据持久化的重任。Hibernate的核心接口一共有5个，分别为:Session、SessionFactory、Transaction、Query和Configuration[5]。这5个 核心接口在任何开发中都会用到。通过这些接口，不仅可以对持久化对象进行存取，还能够进行事务控制

JPA诞生的缘由是为了整合第三方ORM框架，建立一种标准的方式，百度百科说是JDK为了实现ORM的天下归一，目前也是在按照这个方向发展，但是还没能完全实现。在ORM框架中，Hibernate是一支很大的部队，使用很广泛，也很方便，能力也很强，同时Hibernate也是和JPA整合的比较良好，我们可以认为JPA是标准，事实上也是，JPA几乎都是接口，实现都是Hibernate在做，宏观上面看，在JPA的统一之下Hibernate很良好的运行。

　　上面阐述了JPA和Hibernate的关系，那么Spring-data-jpa又是个什么东西呢？这地方需要稍微解释一下，我们做Java开发的都知道Spring的强大，到目前为止，企业级应用Spring几乎是无所不能，无所不在，已经是事实上的标准了，企业级应用不使用Spring的几乎没有，这样说没错吧。而Spring整合第三方框架的能力又很强，他要做的不仅仅是个最早的IOC容器这么简单一回事，现在Spring涉及的方面太广，主要是体现在和第三方工具的整合上。而在与第三方整合这方面，Spring做了持久化这一块的工作，我个人的感觉是Spring希望把持久化这块内容也拿下。于是就有了Spring-data-\*\*这一系列包。包括，Spring-data-jpa,Spring-data-template,Spring-data-mongodb,Spring-data-redis，还有个民间产品，mybatis-spring，和前面类似，这是和mybatis整合的第三方包，这些都是干的持久化工具干的事儿。

我们都知道，在使用持久化工具的时候，一般都有一个对象来操作数据库，在原生的Hibernate中叫做Session，在JPA中叫做EntityManager，在MyBatis中叫做SqlSession，通过这个对象来操作数据库。我们一般按照三层结构来看的话，Service层做业务逻辑处理，Dao层和数据库打交道，在Dao中，就存在着上面的对象。那么ORM框架本身提供的功能有什么呢？答案是基本的CRUD，所有的基础CRUD框架都提供，我们使用起来感觉很方便，很给力，业务逻辑层面的处理ORM是没有提供的，如果使用原生的框架，业务逻辑代码我们一般会自定义，会自己去写SQL语句，然后执行。在这个时候，Spring-data-jpa的威力就体现出来了，ORM提供的能力他都提供，ORM框架没有提供的业务逻辑功能Spring-data-jpa也提供，全方位的解决用户的需求。使用Spring-data-jpa进行开发的过程中，常用的功能，我们几乎不需要写一条sql语句，至少在我看来，企业级应用基本上可以不用写任何一条sql，当然spring-data-jpa也提供自己写sql的方式，这个就看个人怎么选择

### 1.3.4 springboot springMVC hibernate 整合



使用springboot 大大的加快了框架的构建，因为springboot不需要写任何的配置文件就可以整合springmvc和hibernate。只需要在pom文件中加入相关依赖在编写相关数据库的配置（用户名，密码，数据库名称）即可。Springmvc全部使用注解来控制页面也数据的访问。

## 1.4 Mysql数据库

MySQL是一个[**关系型数据库管理系统**](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F)**，**由瑞典MySQL AB 公司开发，目前属于 [Oracle](https://baike.baidu.com/item/Oracle) 旗下产品。MySQL 是最流行的[关系型数据库管理系统](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F)之一，在 WEB 应用方面，MySQL是最好的 RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件。

MySQL是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

MySQL所使用的 SQL 语言是用于访问[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93)的最常用标准化语言。MySQL 软件采用了双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是[开放源码](https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E6%BA%90%E7%A0%81)这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

由于其社区版的性能卓越，搭配 [PHP](https://baike.baidu.com/item/PHP) 和 [Apache](https://baike.baidu.com/item/Apache) 可组成良好的开发环境。

# 第2章 系统需求分析

## 2.1 可行性研究

本设计从技术，经济和社会三个方面来论证本系统实施的可行性

### 2.1.1 技术可行性

本系统采用springboot 框架，结合springmvc和hibernate来完成对业务和数据的操作。Springboot是完全基于spring的，但又优化了spring开发中代码冗长和配置文件过多的问题。使用Mysql数据库

# 第3章 系统设计

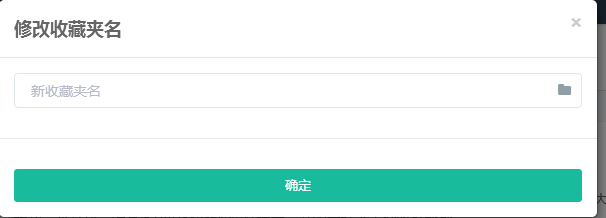
## 3.1 用户功能设计

### 3.1.1 收藏夹功能

（1）创建收藏夹：创建时输入收藏夹的名字，判断收藏夹是否已经存在，如果存在则提醒 “收藏夹名称已被创建”，修改名字后再数据库配置主键自增策略自动生成ID加入到数据库中。



（2）修改收藏夹：用户输入要修改的名字 修改收藏夹名称。



（3）删除收藏夹：根据Id删除收藏夹，并且删除收藏夹下收藏的所以文章或者网页链接。



### 3.1.2 收藏功能

（1）添加收藏：预览一个网页时点击收藏添加收藏，选择收藏在哪一个收藏夹。自动截取文章的标题作为收藏内容的标题和文章或者网页的简介作为收藏的简介。可以输入收藏备注以便下次再看，不输入也可。还可以设置收藏的类型PUBLIC和PRIVATE（私密收藏）控制别人是否可以看见。



（2）修改收藏：可以修改收藏的名称，类型，标题，简介，备注，属于的收藏夹。和添加的页面一样。



（3）删除收藏：选定要删除的收藏，点击确定即可删除。

（4）查询功能：收藏页面自动显示查询的收藏。



### 3.1.3 点赞评论功能

可以通过点击收藏或者评论来评论点赞自己或者他人的公开收藏。



## 3.2 登录注册模块设计

### 3.2.1 注册功能

点击注册即可注册用户，注册时输入邮箱，用户名和密码。会判断邮箱和用户名是否已经被注册如果已经被注册则提醒邮箱已存在，用户名已存在。密码通过正则表达式来规定只能是字母和数字的组合。点击创建账户则自动登录到首页。



### 3.2.2 登录功能

输入用户名或者邮箱和密码登录，如果用户名和密码错误提示“用户名或者密码错误”，正确输入后点击登录即可进入首页



### 3.2.3 忘记密码找回密码

如果忘记密码点击忘记密码即可通过输入邮箱发送邮件来找回密码。



### 3.2.4 登出功能

点击注销即可退出登录。

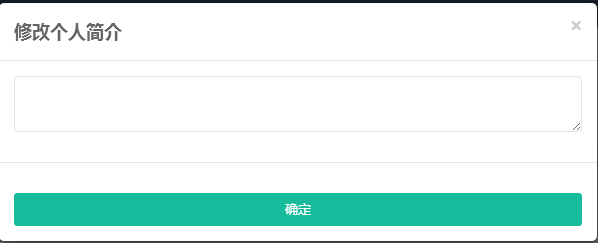


## 3.3 用户个人信息功能



### 3.3.1 修改个人简介

点击修改个人简介来修改个人简介



### 3.3.2 修改头像

点击修改头像来改变头像的显示，在文件夹中选择上传的图片，点击提交即可修改图像



### 3.3.3 修改昵称

点击修改昵称来修改用户的名称如果名称与原来一致则提示“新用户名与原用户名一致”。



### 3.3.4 修改密码

输入新旧密码来修改密码，如果旧密码输入错误提示“密码错误”。



## 3.4 数据库设计

### 3.4.1 数据库设计原则

为了建立冗余较小、结构合理的数据库，设计数据库时必须遵循一定的规则。在关系型数据库中这种规则就称为范式。范式是符合某一种设计要求的总结。要想设计一个结构合理的关系型数据库，必须满足一定的范式：

（1）第一范式(确保每列保持原子性)

第一范式是最基本的范式。如果数据库表中的所有字段值都是不可分解的原子值，就说明该数据库表满足了第一范式。

（2）第二范式(确保表中的每列都和主键相关)

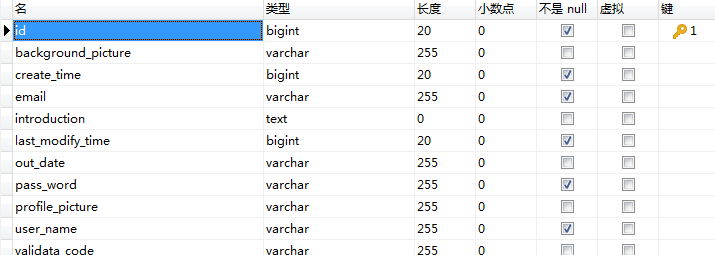
第二范式在第一范式的基础之上更进一层。第二范式需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。也就是说在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中。

（3）第三范式(确保每列都和主键列直接相关,而不是间接相关)

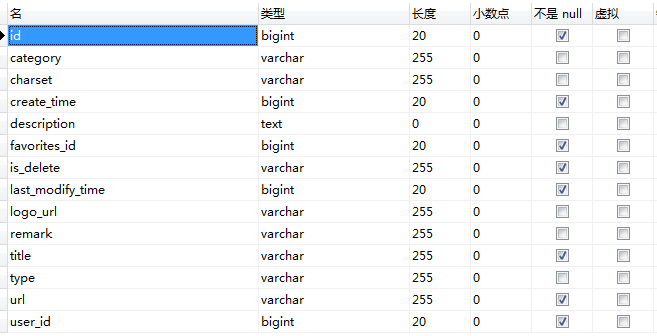
第三范式需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关。

### 3.4.2 数据表的设计

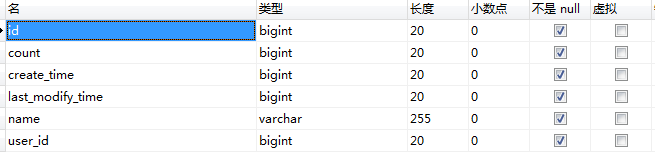
（1）用户（user）表



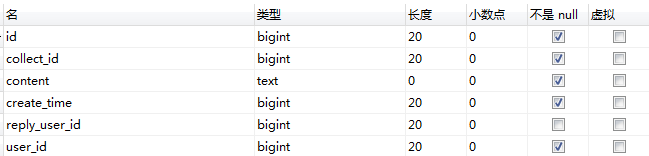
（2）收藏（collect）表



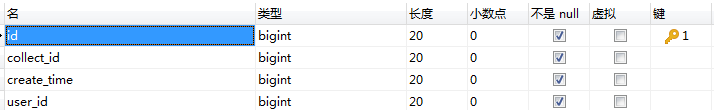
（3）收藏夹（favorites）表



（4）评论（comment）表



（5）点赞（praise）表



（6）私信（letter）表



（7）通知（notice）表



# 第4章 系统的详细设计与实现

系统采用三层架构的思想，分为数据层（Dao）,业务层（Service）,控制层（Controller）。数据层采用的是hibernate框架来进行对数据库的操作为业务层或者控制层提供数据服务。业务层主要是针对具体的问题的操作，也可以理解为对数据层的操作，对数据业务逻辑处理。如果说数据层是积木，那业务层就是对这些积木的搭建。而控制层主要是对用户的请求接受，以及数据的返回，为客户端提供应用程序的访问和交互式操作的页面。采用三层架构的思想可以降低层与层之间的依赖，这样开发人员可以只关注整个结构中的其中某一层。在有更好的方式时也更加易于用新的实现来替换原有层次的实现。主要就是体现了“[高内聚低耦合](https://baike.baidu.com/item/%E9%AB%98%E5%86%85%E8%81%9A%E4%BD%8E%E8%80%A6%E5%90%88)”的思想。

## 4.1 POJO对象的设计

根据数据库表进行对象的创建，利用Lombok使用注解的方式来创建Getter ,Setter方法和构造器来简化代码。其中具体注解的作用：

@Data 创建Setter和Getter方法；

@NoArgsConstructor 创建无参构造器；

@AllArgsConstructor 创建带所有参数的构造器；

系统使用hibernate来对数据库进行增删改查操作，所以要使POJO对象与数据库表关联起来。其中具体注解的作用：

@Entity 是关联数据库表和对象的；

@Id 和 @GeneratedValue 表示表中主键自增的Id；

@Colum 是对象的属性对应的数据库表中的的字段；

@Transient 表示数据库表中没有该字段。

部分代码如下：

（1）收藏对象

@Entity   
@Data

@NoArgsConstructor // 无参构造器  
**public class** Collect **implements** Serializable {  
 **private static final long *serialVersionUID*** = 1L;  
 @Id  
 @GeneratedValue  
 **private** Long **id**;  
 @Column(nullable = **false**)  
 **private** Long **userId**;  
 @Column(nullable = **false**)  
 **private** Long **favoritesId**;  
 @Column(nullable = **false**)  
 **private** String **url**;  
 @Column(nullable = **false**)  
 **private** String **title**;  
 @Column(length = 65535, columnDefinition = **"Text"**)  
 **private** String **description**;  
 @Column()  
 **private** String **logoUrl**;  
 @Column()  
 **private** String **charset**;  
 @Enumerated(EnumType.***STRING***)   
 @Column()  
 **private** CollectType **type**;  
 @Column()  
 **private** String **remark**;  
 @Column(nullable = **false**)  
 @Enumerated(EnumType.***STRING***)   
 **private** IsDelete **isDelete**;  
 @Column(nullable = **false**)  
 **private** Long **createTime**;  
 @Column(nullable = **false**)  
 **private** Long **lastModifyTime**;  
 @Column()  
 **private** String **category**;  
 @Transient  
 **private** String **collectTime**;  
 @Transient  
 **private** String **newFavorites**;  
 }

（2）用户对象

@Entity  
@Data

@NoArgsConstructor  
**public class** User **extends** Entitys **implements** Serializable {  
 **private static final long *serialVersionUID*** = 1L;  
 @Id  
 @GeneratedValue  
 **private** Long **id**;  
 @Column(nullable = **false**, unique = **true**)  
 **private** String **userName**;  
 @Column(nullable = **false**)  
 **private** String **passWord**;  
 @Column(nullable = **false**, unique = **true**)  
 **private** String **email**;   
 @Column(nullable = **true**)  
 **private** String **profilePicture**;  
 @Column(nullable = **true**,length = 65535,columnDefinition=**"Text"**)  
 **private** String **introduction**;  
 @Column(nullable = **false**)  
 **private** Long **createTime**;  
 @Column(nullable = **false**)  
 **private** Long **lastModifyTime**;  
 @Column(nullable = **true**)  
 **private** String **outDate**;  
 @Column(nullable = **true**)  
 **private** String **validataCode**;  
 @Column(nullable = **true**)  
 **private** String **backgroundPicture**;  
 **public** User(String email, String nickName, String passWord, String userName) {  
 **super**();  
 **this**.**email** = email;  
 **this**.**passWord** = passWord;  
 **this**.**userName** = userName;  
 }  
}

（3）收藏夹对象

@Entity  
@Data  
**public class** Favorites **extends** Entitys **implements** Serializable {  
 **private static final long *serialVersionUID*** = 1L;  
 @Id  
 @GeneratedValue  
 **private** Long **id**;  
 @Column(nullable = **false**)  
 **private** Long **userId**;  
 @Column(nullable = **false**)  
 **private** String **name**;  
 @Column(nullable = **false**)  
 **private** Long **count**;  
 @Column(nullable = **false**)  
 **private** Long **createTime**;  
 @Column(nullable = **false**)  
 **private** Long **lastModifyTime**;  
 @Transient  
 **private** Long **publicCount**;

}

（4）私信对象

@Entity  
@Data  
@NoArgsConstructor  
**public class** Letter **extends** Entitys **implements** Serializable {  
 **private static final long *serialVersionUID*** = 1L;  
 @Id  
 @GeneratedValue  
 **private** Long **id**;  
 @Column(nullable = **false**)  
 **private** Long **sendUserId**;  
 @Column(nullable = **false**, length = 65535, columnDefinition = **"Text"**)  
 **private** String **content**;  
 @Column(nullable = **false**)  
 **private** Long **receiveUserId**;  
 @Column(nullable = **true**)  
 **private** Long **pid**;  
 @Enumerated(EnumType.***STRING***)  
 @Column(nullable = **true**)  
 **private** LetterType **type**;  
 @Column(nullable = **false**)  
 **private** Long **createTime**;  
 @Transient  
 **private** String **sendType**; }

## 4.2 数据层（Dao）的设计

系统使用Hibernate来进行数据库的操作，所以在数据层只需要继承JpaRepository<T, ID extends java.io.Serializable>就可以实现大多数的增删改查功能和分页功能。

### 4.2.1 基本查询

只需要继承JpaRepository<T, ID extends java.io.Serializable>即可实现

**public interface** FeedbackRepository **extends** JpaRepository<Feedback, Long> {  
}

### 4.2.2 自定义简单查询

可以根据方法名来自定义的简单查询自动生成SQL，主要的语法是findXXBy,readAXXBy,queryXXBy,countXXBy, getXXBy后面跟属性名称，具体的关键字，使用方法和生产成SQL如下表所示：

| Keyword | Sample | JPQL snippet |
| --- | --- | --- |
| And | findByLastnameAndFirstname | … where x.lastname = ?1 and x.firstname = ?2 |
| Or | findByLastnameOrFirstname | … where x.lastname = ?1 or x.firstname = ?2 |
| Is,Equals | findByFirstnameIs,findByFirstnameEquals | … where x.firstname = ?1 |
| Between | findByStartDateBetween | … where x.startDate between ?1 and ?2 |
| LessThan | findByAgeLessThan | … where x.age < ?1 |
| LessThanEqual | findByAgeLessThanEqual | … where x.age ⇐ ?1 |
| GreaterThan | findByAgeGreaterThan | … where x.age > ?1 |
| GreaterThanEqual | findByAgeGreaterThanEqual | … where x.age >= ?1 |
| After | findByStartDateAfter | … where x.startDate > ?1 |
| Before | findByStartDateBefore | … where x.startDate < ?1 |
| IsNull | findByAgeIsNull | … where x.age is null |
| IsNotNull,NotNull | findByAge(Is)NotNull | … where x.age not null |
| Like | findByFirstnameLike | … where x.firstname like ?1 |
| NotLike | findByFirstnameNotLike | … where x.firstname not like ?1 |
| StartingWith | findByFirstnameStartingWith | … where x.firstname like ?1 (parameter bound with appended %) |
| EndingWith | findByFirstnameEndingWith | … where x.firstname like ?1 (parameter bound with prepended %) |
| Containing | findByFirstnameContaining | … where x.firstname like ?1 (parameter bound wrapped in %) |
| OrderBy | findByAgeOrderByLastnameDesc | … where x.age = ?1 order by x.lastname desc |
| Not | findByLastnameNot | … where x.lastname <> ?1 |
| In | findByAgeIn(Collection ages) | … where x.age in ?1 |
| NotIn | findByAgeNotIn(Collection age) | … where x.age not in ?1 |
| TRUE | findByActiveTrue() | … where x.active = true |
| FALSE | findByActiveFalse() | … where x.active = false |
| IgnoreCase | findByFirstnameIgnoreCase | … where UPPER(x.firstame) = UPPER(?1) |

### 4.2.3 复杂查询

复杂的操作比如连接查询可以使用@Query注解在里面写Sql语句即可实现对数据库的操作。部分代码展示：

**public interface** UserRepository **extends** JpaRepository<User, Long> {  
 User findByUserName(String userName);  
 User findByUserNameOrEmail(String username, String email);  
 User findByEmail(String email);  
  
 @Modifying(clearAutomatically=**true**)  
 @Transactional  
 @Query(**"update User set outDate=:outDate, validataCode=:validataCode where email=:email"**)  
 **int** setOutDateAndValidataCode(@Param(**"outDate"**) String outDate, @Param(**"validataCode"**) String validataCode, @Param(**"email"**) String email);  
  
 @Modifying(clearAutomatically=**true**)  
 @Transactional  
 @Query(**"update User set passWord=:passWord where email=:email"**)  
 **int** setNewPassword(@Param(**"passWord"**) String passWord, @Param(**"email"**) String email);  
  
 @Modifying(clearAutomatically=**true**)  
 @Transactional  
 @Query(**"update User set introduction=:introduction where email=:email"**)   
 **int** setIntroduction(@Param(**"introduction"**) String introduction, @Param(**"email"**) String email);  
   
 @Modifying(clearAutomatically=**true**)  
 @Transactional  
 @Query(**"update User set userName=:userName where email=:email"**)   
 **int** setUserName(@Param(**"userName"**) String userName, @Param(**"email"**) String email);  
   
 @Modifying(clearAutomatically=**true**)  
 @Transactional  
 @Query(**"update User set profilePicture=:profilePicture where id=:id"**)   
 **int** setProfilePicture(@Param(**"profilePicture"**) String profilePicture, @Param(**"id"**) Long id);  
  
 @Modifying(clearAutomatically=**true**)  
 @Transactional  
 @Query(**"update User set backgroundPicture=:backgroundPicture where id=:id"**)  
 **int** setBackgroundPicture(@Param(**"backgroundPicture"**) String backgroundPicture, @Param(**"id"**) Long id);  
 User findById(Long id);

}

## 4.3 业务层（Service）的设计

使用spring的依赖注入（IOC）来创建对象 避免大量的new对象而不能及时清理导致系统垃圾过多运行缓慢或者瘫痪。

@Autowired 是用在JavaBean中的注解，通过byType形式，用来给指定的字段或方法注入所需的外部资源;

@Service 用来注册Bean。

部分代码展示：

@Service  
**public class** LetterService{  
 @Autowired  
 **private** LetterRepository **letterRepository**;  
 @Resource  
 **private** NoticeService **noticeService**;  
 @Autowired  
 **private** UserRepository **userRepository**;  
  
 */\*\*  
 \* 发送私信  
 \** ***@param letter*** *\*/* **public void** sendLetter(Letter letter){  
 **if**(**"original"**.equals(letter.getSendType())){  
 letter.setType(LetterType.***ORIGINAL***);  
 }**else**{  
 letter.setType(LetterType.***REPLY***);  
 List<String> userNameList = StringUtil.*getAtUser*(letter.getContent());  
 **if**(**null** != userNameList && userNameList.size() > 0){  
 User receiveUser = **userRepository**.findByUserName(userNameList.get(0));  
 **if**(**null** != receiveUser){  
 letter.setReceiveUserId(receiveUser.getId());  
 }  
 String content = letter.getContent().substring(0,letter.getContent().indexOf(**"@"**));  
 **if**(StringUtils.*isBlank*(content)){  
 content = letter.getContent().substring(letter.getContent().indexOf(**"@"**)+receiveUser.getUserName().length()+1,letter.getContent().length());  
 letter.setContent(content);  
 }  
 }  
 }  
 letter.setCreateTime(DateUtils.*getCurrentTime*());  
 **letterRepository**.save(letter);  
 **if**(**null** == letter.getPid()){  
 letter.setPid(letter.getId());  
 **letterRepository**.updatePidById(letter.getId(),letter.getId());  
 }  
 *// 添加消息通知* **noticeService**.saveNotice(**null**,**"letter"**,letter.getReceiveUserId(),String.*valueOf*(letter.getId()));  
 }  
 */\*\*  
 \* 私信信息查询  
 \** ***@param userId*** *\** ***@param pageable*** *\** ***@return*** *\*/* **public** List<LetterSummary> findLetter(Long userId, Pageable pageable){  
 List<LetterView> viewList = **letterRepository**.findLetterByReceiveUserId(userId,pageable);  
 List<LetterSummary> summaryList = **new** ArrayList<LetterSummary>();  
 **for**(LetterView view : viewList){  
 LetterSummary summary = **new** LetterSummary(view);  
 summaryList.add(summary);  
 }  
 **return** summaryList;  
 }  
}

## 4.4 控制层（Controller）设计

使用注解的方式实现数据到前台页面的展示。系统采用统一Json格式的数据传输，把返回的数据统一封装方便前台页面的调用。

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
**public class** Response {  
 */\*\* 返回信息码\*/* **private** String **rspCode**=**"000000"**;  
 */\*\* 返回信息内容\*/* **private** String **rspMsg**=**"操作成功"**;  
   
 **public** Response(ExceptionMsg msg){  
 **this**.**rspCode**=msg.getCode();  
 **this**.**rspMsg**=msg.getMsg();  
 }  
   
 **public** Response(String rspCode) {  
 **this**.**rspCode** = rspCode;  
 **this**.**rspMsg** = **""**;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Response{"** +  
 **"rspCode='"** + **rspCode** + **'\''** +  
 **", rspMsg='"** + **rspMsg** + **'\''** +  
 **'}'**;  
 }  
}

@RestController注解等效于@Controller+@ResponseBody的组合，@ResponseBody注解是返回Json格式数据需要。

@RequestMapping注解来控制进入哪一个前台页面或者前台调用哪个方法的返回数据。其中使用@PostMapping表示用的是Post请求一般添加或者修改操作使用，而@GetMapping表示Get请求一般查询或者删除操作使用。

使用@PathVariable和@RequestParam来实现参数的绑定

@PathVariable：是获取请求里面携带的参数； @RequestParam：是获取请求路径里的变量参数。

部分代码展示：

@Controller  
@RequestMapping(**"/"**)  
**public class** IndexController **extends** BaseController{  
 @Autowired  
 **private** FavoritesRepository **favoritesRepository**;  
 @Autowired  
 **private** ConfigRepository **configRepository**;  
 @Autowired  
 **private** FollowRepository **followRepository**;  
 @Autowired  
 **private** CollectRepository **collectRepository**;  
 @Autowired  
 **private** NoticeRepository **noticeRepository**;  
 @Autowired  
 **private** CollectorService **collectorService**;  
 @Autowired  
 **private** CollectService **collectService**;  
 @Autowired  
 **private** UserRepository **userRepository**;  
  
 */\*\*  
 \* 随便看看 added by JiangL  
 \*/* @Autowired  
 **private** LookAroundService **lookAroundService**;  
  
 @GetMapping(value=**"/index"**)  
 @LoggerManage(description=**"首页"**)  
 **public** String index(Model model){  
 IndexCollectorView indexCollectorView = **collectorService**.getCollectors();  
 model.addAttribute(**"collector"**,indexCollectorView);  
 User user = **super**.getUser();  
 **if**(**null** != user){  
 model.addAttribute(**"user"**,user);  
 }  
 **return "index"**;  
 }  
  
 @GetMapping(value=**"/"**)  
 @LoggerManage(description=**"登陆后首页"**)  
 **public** String home(Model model) {  
 **long** size= **collectRepository**.countByUserIdAndIsDelete(getUserId(), IsDelete.***NO***);  
 Config config = **configRepository**.findByUserId(getUserId());  
 Favorites favorites = **favoritesRepository**.findOne(Long.*parseLong*(config.getDefaultFavorties()));  
 List<String> followList = **followRepository**.findByUserId(getUserId());  
 model.addAttribute(**"config"**,config);  
 model.addAttribute(**"favorites"**,favorites);  
 model.addAttribute(**"size"**,size);  
 model.addAttribute(**"followList"**,followList);  
 model.addAttribute(**"user"**,getUser());  
 model.addAttribute(**"newAtMeCount"**,**noticeRepository**.countByUserIdAndTypeAndReaded(getUserId(), **"at"**, **"unread"**));  
 model.addAttribute(**"newCommentMeCount"**,**noticeRepository**.countByUserIdAndTypeAndReaded(getUserId(), **"comment"**, **"unread"**));  
 model.addAttribute(**"newPraiseMeCount"**,**noticeRepository**.countPraiseByUserIdAndReaded(getUserId(), **"unread"**));  
 **logger**.info(**"collect size="**+size+**" userID="**+getUserId());  
 **return "home"**;  
 }  
  
 */\*\*  
 \* 随便看看 简单模式显示 added by JiangL  
 \** ***@return*** */lookAround/simple/ALL  
 \*/* @RequestMapping(value=**"/lookAround/simple/{category}"**)  
 @LoggerManage(description=**"随便看看页面"**)  
 **public** String lookAroundSimple(Model model,@RequestParam(value = **"page"**, defaultValue = **"0"**) Integer page,  
 @RequestParam(value = **"size"**, defaultValue = **"20"**) Integer size,  
 @PathVariable(**"category"**) String category) {  
 Sort sort = **new** Sort(Sort.Direction.***DESC***, **"id"**);  
 Pageable pageable = **new** PageRequest(page, size, sort);  
 model.addAttribute(**"category"**, category);  
 model.addAttribute(**"type"**, **"lookAround"**);  
 Favorites favorites = **new** Favorites();  
 List<CollectSummary> collects = **null**;  
 List<CollectSummary> fivecollects = **lookAroundService**.scrollFiveCollect();  
 List<UserIsFollow> fiveUsers = **lookAroundService**.queryFiveUser(**this**.getUserId());  
  
 collects =**lookAroundService**.queryCollectExplore(pageable,getUserId(),category);  
 User user = **super**.getUser();  
 **if**(**null** != user){  
 model.addAttribute(**"user"**,user);  
 }  
 model.addAttribute(**"fiveCollects"**, fivecollects);  
 model.addAttribute(**"fiveUsers"**, fiveUsers);  
 model.addAttribute(**"collects"**, collects);  
 model.addAttribute(**"favorites"**, favorites);  
 model.addAttribute(**"userId"**, getUserId());  
 model.addAttribute(**"size"**, collects.size());  
 **return "lookAround/simple"**;  
 }  
  
 @GetMapping(value=**"/login"**)  
 @LoggerManage(description=**"登陆页面"**)  
 **public** String login() {  
  
 **return "login"**;  
 }  
  
 @GetMapping(value=**"/register"**)  
 @LoggerManage(description=**"注册页面"**)  
 **public** String regist() {  
 **return "register"**;  
 }  
   
 @RequestMapping(value=**"/tool"**)  
 @LoggerManage(description=**"工具页面"**)  
 **public** String tool(Model model) {  
 String path=**"javascript:(function()%7Bvar%20description;var%20desString=%22%22;var%20metas=document.getElementsByTagName('meta');for(var%20x=0,y=metas.length;x%3Cy;x++)%7Bif(metas%5Bx%5D.name.toLowerCase()==%22description%22)%7Bdescription=metas%5Bx%5D;%7D%7Dif(description)%7BdesString=%22&amp;description=%22+encodeURIComponent(description.content);%7Dvar%20win=window.open(%22"** + Const.*BASE\_PATH* +**"collect?from=webtool&url=%22+encodeURIComponent(document.URL)+desString+%22&title=%22+encodeURIComponent(document.title)+%22&charset=%22+document.charset,'\_blank');win.focus();%7D)();"**;  
 model.addAttribute(**"path"**,path);  
 **return "tool"**;  
 }  
   
 @RequestMapping(value=**"/mobile"**)  
 @LoggerManage(description=**"移动客户端页面"**)  
 **public** String mobile() {  
 **return "mobile"**;  
 }  
   
 @RequestMapping(value=**"/import"**)  
 @LoggerManage(description=**"收藏夹导入页面"**)  
 **public** String importm() {  
 **return "favorites/import"**;  
 }  
   
 @RequestMapping(value=**"/newFavorites"**)  
 @LoggerManage(description=**"新建收藏夹页面"**)  
 **public** String newFavorites(){  
 **return "favorites/newfavorites"**;  
 }  
  
 @RequestMapping(value=**"/feedback"**)  
 @LoggerManage(description=**"意见反馈页面"**)  
 **public** String feedback(Model model){  
 User user = **null**;  
 user = **userRepository**.findOne(getUserId());  
 model.addAttribute(**"user"**, user);  
 **return "favorites/feedback"**;  
 }  
  
 @GetMapping(value=**"/collect"**)  
 @LoggerManage(description=**"收藏页面"**)  
 **public** String collect(Model model) {  
 List<Favorites> favoritesList = **favoritesRepository**.findByUserId(getUserId());  
 Config config = **configRepository**.findByUserId(getUserId());  
 List<String> followList = **followRepository**.findByUserId(getUserId());  
 **logger**.info(**"model："** + config.getDefaultModel());  
 model.addAttribute(**"favoritesList"**,favoritesList);  
 model.addAttribute(**"configObj"**, config);  
 model.addAttribute(**"followList"**,followList);  
 **return "collect"**;  
 }  
  
 @GetMapping(value=**"/logout"**)  
 @LoggerManage(description=**"登出"**)  
 **public** String logout(HttpServletResponse response,Model model) {  
 getSession().removeAttribute(Const.*LOGIN\_SESSION\_KEY*);  
 getSession().removeAttribute(Const.*LAST\_REFERER*);  
 Cookie cookie = **new** Cookie(Const.*LOGIN\_SESSION\_KEY*, **""**);  
 cookie.setMaxAge(0);  
 cookie.setPath(**"/"**);  
 response.addCookie(cookie);  
 IndexCollectorView indexCollectorView = **collectorService**.getCollectors();  
 model.addAttribute(**"collector"**,indexCollectorView);  
 **return "index"**;  
 }  
  
 @GetMapping(value=**"/forgotPassword"**)  
 @LoggerManage(description=**"忘记密码页面"**)  
 **public** String forgotPassword() {  
 **return "user/forgotpassword"**;  
 }  
  
 @GetMapping(value=**"/newPassword"**)  
 **public** String newPassword(String email) {  
 **return "user/newpassword"**;  
 }  
  
 @RequestMapping(value=**"/uploadHeadPortrait"**)  
 @LoggerManage(description=**"上传你头像页面"**)  
 **public** String uploadHeadPortrait(){  
 **return "user/uploadheadportrait"**;  
 }  
   
 @RequestMapping(value=**"/export"**)  
 @LoggerManage(description=**"收藏夹导出页面"**)  
 **public** String export(Model model){  
 List<Favorites> favoritesList = **favoritesRepository**.findByUserId(getUserId());  
 model.addAttribute(**"favoritesList"**,favoritesList);  
 **return "favorites/export"**;  
 }  
  
 @RequestMapping(value=**"/uploadBackground"**)  
 @LoggerManage(description=**"上传背景页面"**)  
 **public** String uploadBackground(){  
 **return "user/uploadbackground"**;  
 }

}

## 4.5 其他功能设计

### 4.5.1 发送邮箱功能

在我们忘记密码时需要使用JavaMailSender来发送邮件找回密码。在application.yml中配置发送者信息



使用JavaMailSender类发送短信如下：

@Resource  
**private** JavaMailSender **mailSender**; //依赖注入

**public** Response sendForgotPasswordEmail(String email) {  
 **try** {  
 User registUser = **userRepository**.findByEmail(email);  
 **if** (**null** == registUser) {  
 **return** result(ExceptionMsg.***EmailNotRegister***);  
 }   
 String secretKey = UUID.*randomUUID*().toString(); *// 密钥* Timestamp outDate = **new** Timestamp(System.*currentTimeMillis*() + 30 \* 60 \* 1000);*// 30分钟后过期* **long** date = outDate.getTime() / 1000 \* 1000;  
 **userRepository**.setOutDateAndValidataCode(outDate+**""**, secretKey, email);  
 String key =email + **"$"** + date + **"$"** + secretKey;  
 String digitalSignature = MD5Util.*encrypt*(key);*// 数字签名* String resetPassHref = **forgotpasswordUrl** + **"?sid="** + digitalSignature +**"&email="**+email;  
 String emailContent = MessageUtil.*getMessage*(**mailContent**, resetPassHref);  
 MimeMessage mimeMessage = **mailSender**.createMimeMessage();   
 MimeMessageHelper helper = **new** MimeMessageHelper(mimeMessage, **true**);  
 helper.setFrom(**mailFrom**);  
 helper.setTo(email);  
 helper.setSubject(**mailSubject**);  
 helper.setText(emailContent, **true**);  
 **mailSender**.send(mimeMessage);  
 } **catch** (Exception e) {  
 **logger**.error(**"sendForgotPasswordEmail failed, "**, e);  
 **return** result(ExceptionMsg.***FAILED***);  
 }  
 **return** result();  
}

mailFrom: 发送者的邮箱

email：接受者的邮箱

emailContent：发送内容

mailSubject：发送标题

### 4.5.2 日志功能

使用AOP面向切面编程的方式来实现日志功能。本系统是使用注解的方式实现AOP。

使用@Aspect来开启。

@Before 前置通知： 在方法执行之前先进行日志记录；

@AfterReturning 后置通知：在方法执行之后来执行日志记录；

@AfterThrowing 异常通知：在遇到异常情况下执行日志记录。

@Aspect  
@Service  
**public class** LoggerAdvice {  
   
 **private** Logger **logger** = Logger.*getLogger*(**this**.getClass());  
  
 @Before(**"within(com.graduation..\*) && @annotation(loggerManage)"**)  
 **public void** addBeforeLogger(JoinPoint joinPoint, LoggerManage loggerManage) {  
 **logger**.info(**"执行 "** + loggerManage.description() + **" 开始"**);  
 **logger**.info(joinPoint.getSignature().toString());  
 **logger**.info(parseParames(joinPoint.getArgs()));  
 }  
   
 @AfterReturning(**"within(com.graduation..\*) && @annotation(loggerManage)"**)  
 **public void** addAfterReturningLogger(JoinPoint joinPoint, LoggerManage loggerManage) {  
 **logger**.info(**"执行 "** + loggerManage.description() + **" 结束"**);  
 }  
   
 @AfterThrowing(pointcut = **"within(com.graduation..\*) && @annotation(loggerManage)"**, throwing = **"ex"**)  
 **public void** addAfterThrowingLogger(JoinPoint joinPoint, LoggerManage loggerManage, Exception ex) {  
 **logger**.error(**"执行 "** + loggerManage.description() + **" 异常"**, ex);  
 }  
  
 **private** String parseParames(Object[] parames) {  
 **if** (**null** == parames || parames.**length** <= 0) {  
 **return ""**;  
 }  
 StringBuffer param = **new** StringBuffer(**"传入参数[{}] "**);  
 **for** (Object obj : parames) {  
 param.append(ToStringBuilder.*reflectionToString*(obj)).append(**" "**);  
 }  
 **return** param.toString();  
 }  
}

其中 **"within(com.graduation..\*) && @annotation(loggerManage)"** [表示的是扫描com.graduation包下的所以带有@LoggeManage](mailto:表示的是扫描com.graduation包下的所以带有@LoggeManage)注解的方法。

### 4.5.3 文件上传功能

使用I/O流来进行文件的上传。在修改图片时需要用到图片的上传。因为图片不是字符所以需要用到字节输出流 FileOutputStram 来写入数据。具体方法如下：

**public static void** uploadFile(**byte**[] fileBytes, String filePath, String fileName) **throws** Exception {   
 File targetFile = **new** File(filePath);   
 **if**(!targetFile.exists()){   
 targetFile.mkdirs();   
 }   
 FileOutputStream out = **new** FileOutputStream(filePath+fileName);  
 out.write(fileBytes);  
 out.flush();  
 out.close();  
}

# 第5章 结论

# 致谢

本设计在张锏老师的悉心指导和严格要求下业已完成，从课题选取、方案论证到具体设计和调试，无不凝聚着张老师的心血和汗水，在四年的本科学习和生活期间，也始终感受着导师的精心指导和无私的关怀，我受益匪浅。在此向张锏老师表示深深的感谢和崇高的敬意。

　　不积跬步何以至千里，本设计能够顺利的完成，也归功于各位任课老师的认真负责，使我能够很好的掌握和运用专业知识，并在设计中得以体现。正是有了他们的悉心帮忙和支持，才使我的毕业论文工作顺利完成，在此向宿州学院，信息工程系的全体老师表示由衷的谢意。感谢他们四年来的辛勤栽培。