编　　号：**2015211841**

审定成绩：_____

重庆邮电大学
毕业设计（论文）

中文题目：	个人信息定制与共享平台的设计与实现
英文题目：	**Design and Implementation of Personal Information Customization and Sharing Platform**
学院名称：	计算机科学与技术学院
学生姓名：	杨思成
专　　业：	计算机科学与技术
班　　级：	**04011507**
学　　号：	**2015211841**
指导教师：	夏英　　教授
答 辩 组： 负 责 人：	王进　　教授

二 O 一九 年 六 月

重庆邮电大学教务处制

计算机科学与技术学院本科毕业设计(论文)诚信承诺书

本人郑重承诺：

我向学院呈交的论文《个人信息定制与共享平台的设计与实现》，是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明并致谢。本人完全意识到本声明的法律结果由本人承担。

年级

专业

班级

承诺人签名

年　　月

学位论文版权使用授权书

本人完全了解重庆邮电大学有权保留、使用学位论文纸质版和电子版的规定，即学校有权向国家有关部门或机构送交论文，允许论文被查阅和借阅等。本人授权重庆邮电大学可以公布本学位论文的全部或部分内容，可编入有关数据库或信息系统进行检索、分析或评价，可以采用影印、缩印、扫描或拷贝等复制手段保存、汇编本学位论文。

（注：保密的学位论文在解密后适用本授权书。）

学生签名：　　指导老师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘要

随着互联网技术的不断发展，信息传输速度的不断增加，让各种信息的存储交互成为可能。对于我们个人而言，每个人也都相当于一个巨大的数据库，携带了大量的信息，伴随着数据量的增多，数据结构越来越复杂就会带来以下两个问题。第一个问题是我们如何管理自己所掌握的信息，第二个问题是我们如何使用所拥有的信息为自身创造价值。因此，探究如何利用现有的开发技术对于解决个人信息个性化带来的问题具有重要意义。

针对以上互联网用户所遇到的两个问题，本文设计并实现了一个基于 Java 的个人信息定制与共享平台。平台主要提供个人信息定制化管理、个人信息定制化分享、他人信息检索等功能。系统总体上基于三层 B/S 架构设计开发，通过 HTTP 网络协议实现网页、应用服务器、数据服务器之间的数据通信。本文选择 MySQL 数据库作为存储数据库、Java 开发语言为主要开发语言、AdminLTE2 作为前端开发模板。其中前端和后端都是严格遵循 MVC 三层架构的开发模式，数据库建库也是根据企业级开发要求进行搭建。

最后，本文对平台进行了功能测试，并且通过网页客户端和数据库中的内容来验证是否能够稳定准确的返回数据。测试结果显示，所有功能均正常，获取到的数据都是准确数据，并且能够一定程度上解决上述两个问题。

关键词：个人信息定制与共享，Java，MVC 三层架构，B/S 架构

Abstract

With the continuous development of Internet technology, the speed of information transmission has increased, making it possible to store and exchange various kinds of information. For us personally, each person is also equivalent to a huge database, carrying a large amount of information, accompanied by an increase in the amount of data, the data structure is more and more complex will bring the following two problems. The first question is how we manage the information we have. The second question is how we use the information we have to create value for ourselves. Therefore, exploring how to use existing development techniques is of great significance for solving the problems caused by the personalization of personal information.

In view of the two problems encountered by the above Internet users, this paper designs and implements a Java-based personal information customization and sharing platform. The platform mainly provides functions such as customized management of personal information, customized sharing of personal information, and information retrieval by others. The system is generally designed and developed based on the three-layer B/S architecture, and realizes data communication between the webpage, the application server, and the data server through the HTTP network protocol. This paper chooses MySQL database as the storage database, Java development language as the main development language, and AdminLTE2 as the front-end development template. The front-end and back-end are strictly follow the MVC three-tier architecture development model, and database database construction is also based on enterprise-level development requirements.

Finally, this article tests the platform and verifies whether it can return data stably and accurately through the content of the web client and database. The test results show that all functions are normal, the data obtained is accurate data, and the above two problems can be solved to some extent.

Keywords: Personal information customization and sharing, Java, MVC three -tierarchitecture, B/S architecture

目录

第 1 章 引言

科技的飞速发展使得信息的传输速度越来越快，与此同时，每个人拥有的信息越来越多，也越来越个性化、多元化。如何合理的管理我们的信息并且能够使其发挥价值就成为一个需要探讨的话题。接下来本文将针对这个话题进行探讨并且寻求合理的解决方案。

1.1 研究背景和意义

在当今的社会背景下，随着互联网的高速发展，网络传输速度越来越快，信息之间的交互速度也飞速增长。我们每个人都相当于一个自媒体，携带并且拥有大量的信息，那么应如何利用这些信息，在向外分享和向内摄取中均能实现方便获取且合理分享的目的呢？

用学校举例，实验室在招收学生或者学校在选择实验室老师时，就需要充分了解候选老师与候选学生各自的特点，从而确定他们是否匹配学校的项目[1]；在公司中，由于员工对自己不够明确的定位或在谋求新的岗位时，就会出现岗位与人才匹配的情况；以我们即将毕业的大学生为例，本系统也可以成为我们与用人单位之间沟通的桥梁[2]，能够让彼此有一个相互了解和互相选择的机会。

所以，综上我们可以看到，个人信息定制与共享平台不只是单一的模式、单一的受众、单一的群体，而是可以模式转换、用户转换，从而依托在更多的平台上，它使得信息的合理化关系和共享变为可能。

本文选择以教师为用户群体进行平台搭建[3]，模拟教师拥有哪些数据、教师如何管理自己的数据、教师如何定义自己的数据[4]，以及教师如何合理的管理和共享自己的数据，让自己的数据发挥最大价值的同时也能够获取自己想要的数据。

共赢包含着对利益的追求，但并不止于对利益的追求。个人信息定制与共享平台不只是一个学校学生间的共享平台；也不只是老师们互相交流信息的平台；更不只是公司内部互相合作、内部招聘的依托平台，它更是一种模式：一种对自己信息进行管理、对需要的信息进行检索的平台。开发这个平台的目的就是解决一切需要

人才资源和“岗位”资源的场所中所存在的资源匹配不合理的问题，合理的分配才能够达到更好的效果。

个人信息定制与共享平台可以更好的解决内外部信息的沟通；更好的协调人员和资源，从而将效率提升到一个更高的层次；它也能方便个人实现自己更大更合理的价值。

1.2 国内外研究现状

1.2.1 国外研究现状

科技的发展带来的信息管理问题是全球化的，为解决这一问题，国外也出现了很多优秀的社交平台，如：Facebook，Twitter，这两个平台是全球领先的社交平台，为其用户提供交流与部分信息管理的功能。这两个平台都偏向社交，类似于我国的QQ 与微信，它们都有一个共同点：所有信息只对自己的圈子开放，不能达到信息价值化的目的，因此无法被他人检索。而信息共享需要达到的信息价值化则需要向外界分享自己的信息。

相比而言，领英（LinkedIn）作为一个全球知名的职业社交平台，它在信息共享方面就做得很好。它能够代表绝大多数的招聘平台，允许用户自定义和发布信息，但是由于产品定位的原因，领英只适合作为招聘平台，而并不适合作为一种可以复刻的模式：既能将其应用于学校、公司内部、又能作为招聘平台等。国外还有比较知名的 Pinterest 和 Instagram 两款软件，但是管理的主体都是图片，虽然提供了个性化管理，但是都不能算作纯粹的个人信息定制化管理与分享平台。

社交媒体平台既可以促进知识共享，又可以方便个人数据管理[5]。但是越来越多的消费者被要求提供个人信息以换取个性化定制的服务，如果披露的个人信息（质量和完整性）与信息发挥的价值存在差异[6]，那么这将达不到共享的价值。

国外很早就意识到，随着时代科技的发展，个人信息管理将变成一个值得探讨的话题，所以从 20 世纪上半叶开始就进行了相关研究。

国外个人信息管理研究可分为三个阶段[7]。

第一阶段开始于 20 世纪上半叶，美国科学家 Vannevar Bush 于 1945 年在《诚如我思》（As we may think）一文中首次提出个人信息管理的概念和 MEMEX 系统

的雏形[7]。此系统针对科学家，提供个人书籍，档案或者信函等信息的管理功能。但是由于科技不够发达的原因，使得当时个人信息的个性化和多元化远不及现在，并且当时对于个人信息的管理仅限单一的信息，如：微文档系统、文件管理系统。这一阶段可视为国外的个人信息管理研究的启蒙阶段。

第二阶段。20 世纪 80 年代末，微型计算机技术快速发展。Mark Lansdale 于 1988 年在《PIM 心理》一文中首次提出“personal information management”（PIM）一词，他认为 PIM 即个人分类和检索信息的方法和程序[7]。随后个人信息管理系统进入兴起阶段，C.Moon 提出用个人信息管理系统来管理个人信息的获取、存储和检索。

第三阶段。随着个人信息关系的研究受到越来越多的关注，2005 年首届个人信息管理会议在美国举办[7]，直到今天，该会议已经成功举办了若干届。随着信息时代的到来，每个人都会遇到个人信息关系的问题，个人信息管理进入普遍应用阶段。

如今，国外的个人信息管理研究范围不断扩大，研究对象从研究科研人员逐渐转为普通个体。个人信息类型、媒介以及载体研究范围不断扩大、细化，其中又以个人电子邮件，网页，任务或工作流等研究居多。

1.2.2 国内研究现状

国内的个人信息管理系统也趋近于复杂化，个人化[8]。相应的也出现了针对各种个人信息的管理系统，比如人才管理系统，学生管理系统，职工管理系统等。都是将个人主要信息存储下来方便管理和检索。为了切合本文，下面将使用分享和管理及岗位与人才匹配度来讲解国内个人信息管理研究现状。

在当今的社会背景下，人才和资源趋于多元化，所以人才与“岗位”之间能否合理的匹配，达到人尽其责的效果，就会变得尤其重要了。

而且这种矛盾可以出现在各种地方，用学校举例，学校的实验室在招收学生或者学校在选择实验室的老师选择合适的合作伙伴的时候，就需要充分的了解各个候选老师与候选学生的优缺点是否匹配自己的项目；或者是公司里，可能有些员工刚开始对于自己的定位不是很明确，或者在一个领域工作了足够久的时间想要换个岗位继续发光发热等各种理由，会出现岗位与人才匹配的情况的时候。或者是拿

到了大项目需要找实力强劲的队友合作的时候；再比如以我们即将毕业的大学生为例，也可以是我们与用人单位之间沟通桥梁，能够让我们有一个相互了解和互相选择的机会。

综上所述，我们都能够提炼出来一个共同点，那就是如何合理的匹配人才资源与“岗位”资源。这里的岗位不仅限于工作岗位，也可能是老师之间的合作，或者是学生加入实验室。为了解决这种复杂的资源分配问题，有必要开发一个平台。对于每一位普通用户都开放两个功能，一个是个人信息的管理与维护。另一个是他人信息的检索，这样就给了每一个人选择与被选择的权利。在这个平台上大家完全可以将自己的被动需求与主动需求发挥出来，以达到人才资源与“岗位”资源的合理分配。

在已有的系统中，最相近的就是各个招聘网站的招聘系统了，它提供了自我信息的建立修改与编辑。一方面它提供了海量的人才信息，另一方面对应的还有海量的“岗位”信息。用人单位可以在上面选择自己需要的人才，而人才同样可以在上面寻找自己心仪的岗位。缺点是过于冗杂毕竟注册是免费的，所以难免会出现广告，买排名的情况出现，使其情况出现失实的可能性。而且模式比较单一，它属于将这种模式具象化的一种体现。

个人信息定制与共享平台虽然最后会依托于学校的教师个人信息管理与分享展现。但是更重要的是提出这种模式。就像前面举例的矛盾，平台可以出现在任何有人才资源和“岗位”资源有分歧的地方[8]。优点是功能清晰，系统更纯粹化和轻量化，界面友好。可以依托在现有工作平台上，成为一个分支。

1.3 主要内容和工作安排

本论文共分为 6 章，主要章节内容概括如下：

第 1 章为引言，引入课题的研究背景及意义，阐述个人信息管理平台国内外的研究现状，对此次研究的主要内容和工作安排做了简要概述。

第 2 章主要介绍了本系统开发用到的工作和一些技术，以及其中遵循的开发原理和模式。

第 3 章主要阐述了系统的需求分析以及可行性分析。

第 4 章主要介绍了系统的设计过程，系统的整体设计以及个别重要用例设计。

第 5 章则详细的描述系统中主要用例的代码实现。

第 6 章对此次工作做总结，发现其中的不足和有待改进的地方，并合理的展望未来。

第 2 章 相关技术与理论基础

2.1 开发工具介绍

2.1.1 IntelliJ IDEA 编译器

IDEA 是 java 编程语言开发的集成环境[9]。应该也是目前世界上功能最完备最智能的 java 编译软件之一，值得一提的是它最受开发者喜爱的原因是它所提供的代码提示功能、重构提示功能等、代码分析功能。另外它也增加了对 J2EE 支持、各类版本工具的支持(gitHub、svn 等)、CVS 整合等。为了最大限度的提高开发人员工程的开发速度，可以说 IntelliJ IDEA 的每个方面都是专门设计的，尤其是其强大的静态代码分析能力和符合人体工程学编码设计使开发不仅具有高效性，而且还给能开发者带来舒适的编码体验。

在 IntelliJ IDEA 为开发者的代码生成索引之后 [9]，它就能够通过在每个上下文中提供相关修改意见提供高效的解决方案：高效而智能的完成修改。所以 IDEA 是一个非常可靠的动态代码分析和重构的工具。

2.1.2 Tomcat 服务器

Tomcat 是 Apache 软件基金会（Apache Software Foundation）的 Jakarta 项目中的一个核心项目[10]，由 Apache、Sun 和其他一些公司及个人共同开发而成。由于 Tomcat 性能稳定、技术先进，而且开源，因而深受 Java 开发者的喜爱并得到了部分软件开发商的认可，成为目前比较流行的 Web 应用服务器[10]。

Tomcat 服务器不仅免费并且还是开源项目，在中小型系统和并发访问量不是很大的场合下被普遍使用，是开发和调试 JSP 程序的首选[10]。可以简单的理解为，只要在一台 PC 机上配置好 Apache 服务器，就可以利用它来响应 HTML 页面发出的访问请求。

当配置正确时，Apache 为 HTML 页面服务[10]，而实际上 Tomcat 则运行了 JSP 页面和 Servlet。除此之外 Tomcat 还具备处理 HTML 页面的能力。不仅如此，它还能像一个容器一样装载 servlet 和 Jsp。

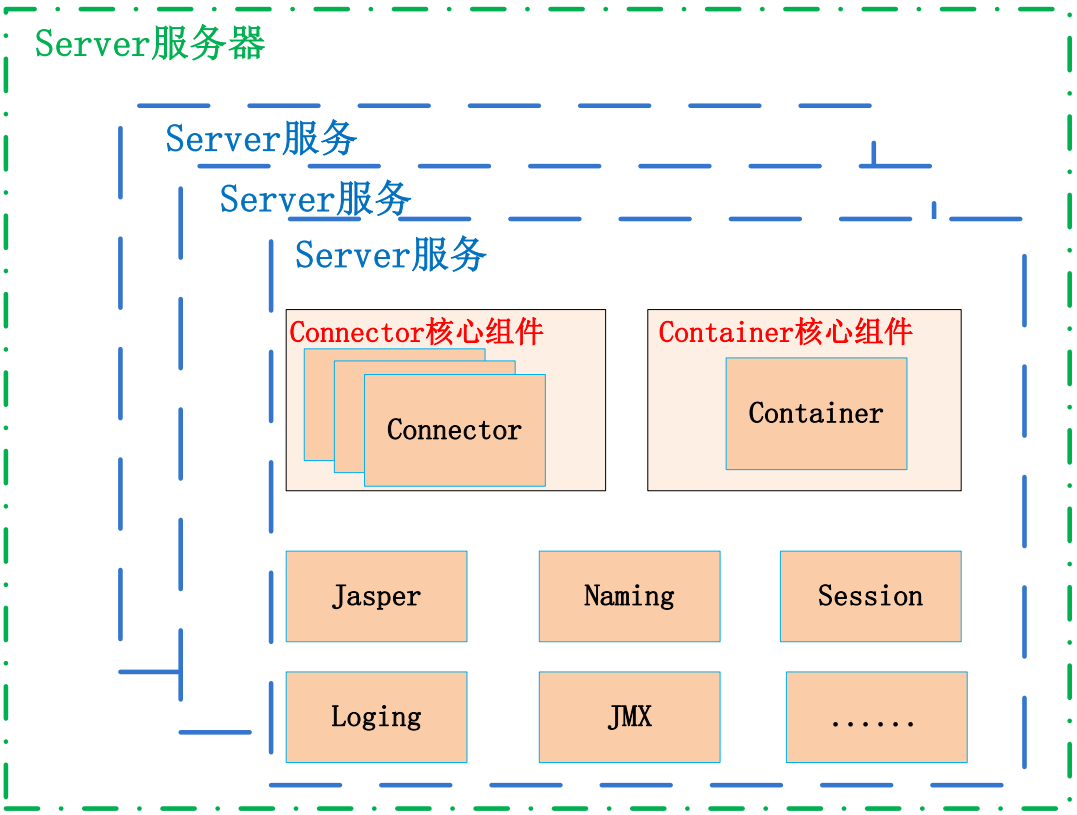


图 2.1 Tomcat 服务器核心结构图[10]

如图 2.1 我们可以看到 Tomcat 服务器核心结构图。

Tomcat 主要组件：服务器 Server，服务 Service，连接器 Connector、容器 Container。连接器 Connector 和容器 Container 是 Tomcat 的核心[10]。

一个 Container 容器和一个或多个 Connector 组合在一起[10]，加上其他一些支持的组件共同组成一个 Service 服务。当有了 Service 服务器的时候便可以向外界提供能力了。不过并没有这么简单，还需要一个适当的环境。这个环境便是 Server，Server 便可以为 Service 服务的正常使用提供生存环境，更值得一提的是，一个 Server 组件可以同时管理一个或多个 Service 服务。

2.1.3 Wampserver64 集成环境包

WampServer 是一款建立在 PC 端的整合软件包 [11]、包括 PHP 解析器、Apache 服务器以及 MySQL 数据库 [11]。其特点就是集成，只需要启动一个软件便可以创建三种开发环境。

特点：

1.增加了对中文的支持，方便国内开发者；

2.集成度高，并且支持 PHP 扩展、Apache 的 mod_rewrit；

3.操作方式是一键式操作，方便快捷；

2.1.4 Navicat for MySQL 数据可视化工具

Navicat for MySQL 是业界基本公认的的管理 MySQL 和开发 MySQL 数据库最优秀的工具[12]，它不仅支持支持单一程序，更可以同时连接到 MySQL 和 MariaDB 两个数据库。值得一的是只要 MySQL 版本大于 3.21 均可以使用 Navicat 作为可视化工具，另外 Navicat 还支持大部分 MySQL 最新版本的功能，包括权限管理、函数、触发器等等。

由此可见，Navicat for MySQL 作为 MySQL 的开发工具[10] ，为开发者提供了非常舒适的开发体验。Navicat for MySQL 基于 Windows 平台，可以说是为 MySQL 量身订作的可视化界面管理工具[10]。使数据库开发者可以用一种更为容易并且更安全高效的方式快速地创建、存取、组织和共享信息。

2.2 MVC 设计模式

MVC 设计模式理解，图 2.2 为 MVC 设计模式运行流程图。

MVC 是其中包含的组件的首字母的缩写连拼，三个组件分别是模型(model)－视图(view)－控制器(controller)[13]。这种设计模式的意义就是将数据，界面，业务逻辑层进行分析。这种编码方式通常方便后期管理与扩展，无论是单独修改模型、更改前端页面样式还是修改后台代码逻辑，均不需要重新从头到尾修改业务逻辑。

Model（模型）：提供要展示的数据，是与数据库进行交互的部分，并且在交互完成之后把数据返还给控制器，所以既包含业务也包含数据。主要使用的技术：数据模型：实体类（JavaBean）[13]。

View（视图）：是逻辑业务比较少的部分，主要功能是将 Model 进行修饰和包装之后在前端向用户展示[13]。

Controller（控制器）：主要职责为接收用户请求，并把具体操作转入具体的 Model 层进行执行，并且处理完毕的模型返回给视图层，由视图负责展示。主要使用的技术：SpringMVC 中的 Controller 类，servlet 等[13]。

严格的来说，MVC 是一个框架模式，在它的规则下，开发者必须将输入，处理，和输出分开。三个由于逻辑被区分开来的模块，理论上互不干涉又互相依赖。最典型的 MVC 结构就是 JSP + servlet + javabean 的模式[13]。

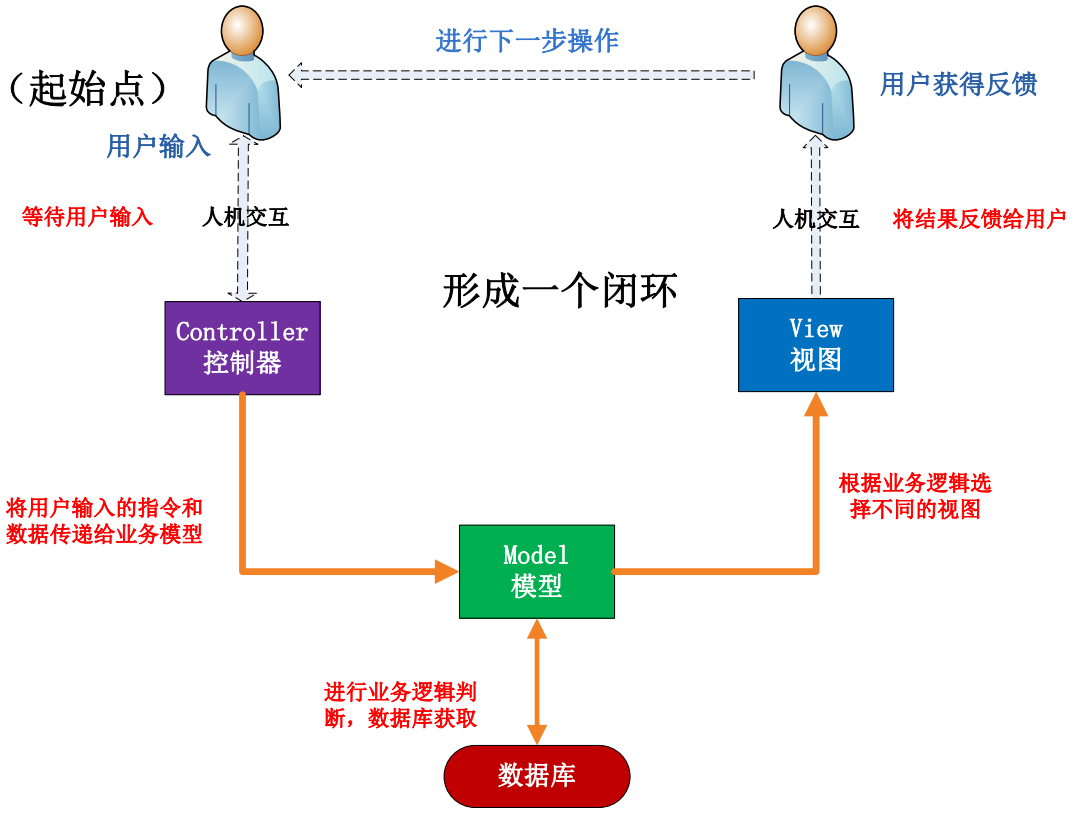


图 2.2 MVC 框架模式运行流程图[13]

2.3 B/S 开发模式及选模原因

B/S 开发模式就是 Browser/Server 模式（浏览器/服务器）[14]。是以网络浏览器为契机，当网络浏览器成为人们日常的上网方式时，这种开发模式也应运而生。

这种模式在一定意义上选定了客户端，简化了系统的开发、维护和使用[14]。另外，这种开发模式对于用户终端要求比较低，客户机上只要安装一个浏览器就可以对服务器发起请求。

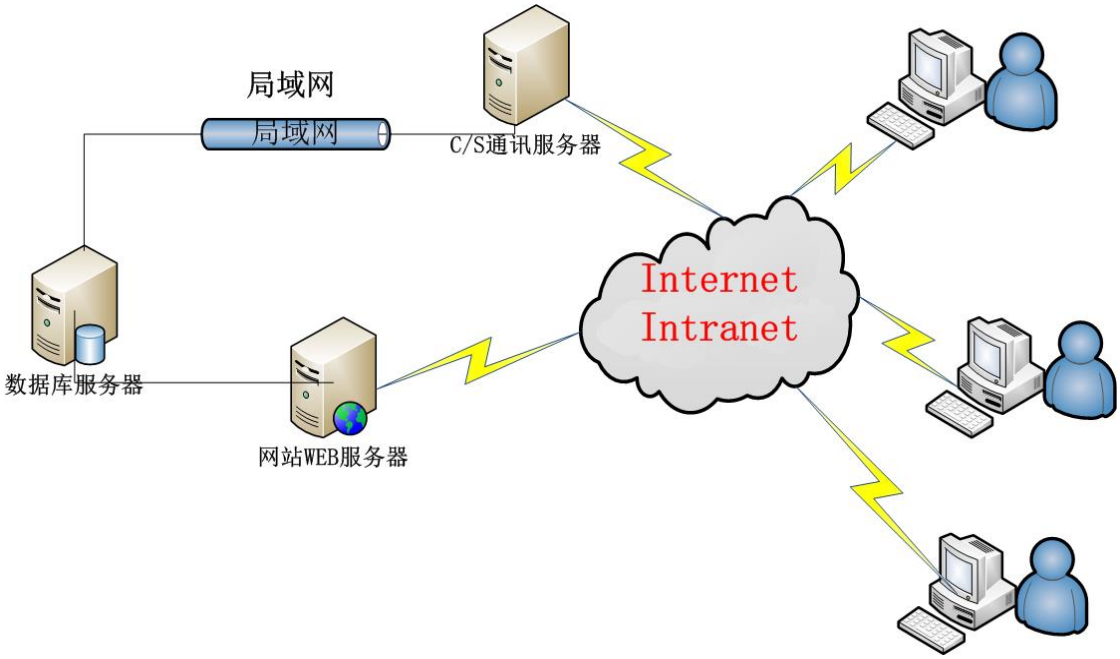


图 2.3 C/S 模式模拟运作图[14]

选模原因：

零安装：拥有一个浏览器即可访问。因此，它所面向的范围更为的广阔。

扩展性：比较常见的扩展方式就是增加页面即可。

零维护：通过增加页面即可增加功能。

综合本文的系统，再加上对于系统的理解，由于要展示的是一位教师拥有的信息，可能量级比较大，所以我们也选择了较为友好的 PC 端进行展示，这样容易显示更多数据。

2.4 SSM 框架开发技术

2.4.1 SSM 框架子框架

SSM（Spring+SpringMVC+MyBatis）框架集由 Spring、MyBatis 两个开源框架整合而成（SpringMVC 是 Spring 中的部分内容）[15]。常作为数据源较简单的 web 项目的框架。

2.4.2 SSM 框架机制

SSM 框架是 springMVC，spring 和 mybatis 框架的整合，可以说就是 MVC 模式的实现，将整个系统划分为四层：View 层，Controller 层，Service 层，Dao 层[16]。

Spring MVC 主要负责请求管理和视图管理

Spring 运用 IOC 和 AOP 思想实现业务对象管理

Mybatis 作为数据持久层框架主要进行数据对象管理 [16]

2.5 系统所用依赖包

commons-lang3 通用函数包：commons-lang3 版本号 3.5

jstl 标签库包：jstl 版本号 1.2

Log4J 日志包：log4J 版本号 1.2.17

Lervlet 包：servlet 版本号 3.1.0

Slf4j 日志包：slf4j 版本号 1.7.25

Spring 框架包：spring 版本号 4.3.17.RELEASE

阿里巴巴第三方数据库链接包：alibaba-druid 版本号 1.1.6

MySQL 框架包：MySQL 版本号 5.1.46

Mybatis 框架包：mybatis 版本号 3.2.8

Mybatis 整合 Spring 包：mybatis-spring 版本号 1.3.1

测试包：org.springframeword 版本号 4.0.0

测试包：junit 版本号 4.12

Excel 文件解析包：org.apache.poi 版本号 3.6

Excel 文件解析包：commons-fileupload 版本号 1.2.1

构造函数生成包：Lombok 版本 1.18.6

2.6 本章小结

本章节对系统开发工具进行了比较详细的解释，并且对于主要遵循的 MVC 视图层结构也进行了解释，并浅谈了 B/S 开发模式等。

第 3 章 系统需求分析

通常在开发一个程序或系统之前，需要做需求分析，应首先了解系统的详细设计、详细需求，其中包括：产品功能需求、性能需求、客户需求等。在了解了相关需求，有了明确的设计思路和设计构想之后，才能设计出满足需求的产品，从而提高开发完整度和客户体验满意度。

3.1 系统功能性需求

为了解决个人信息管理和共享的问题，本系统应该具有个人信息增加、删除、修改功能，通过这些功能用户可以方便的管理个人信息。

本系统是以教师作为用户，结合系统特色，系统应该具有科研信息管理、教研信息管理、模板管理、个人信息管理这四个大板块。

相应的，在科研信息管理、教研信息管理、个人信息管理和模板管理大功能下，本文对于教研信息管理和科研信息管理这两个模块下的具体内容提供新增、编辑、删除等基础功能，并且提供单条论文置顶功能、模糊信息查询功能和导入导出 Excel 功能；对于个人信息管理模块，提供修改密码和修改个人信息的功能；在模板管理功能中，为用户提供模板进行选择，并且在对未来的展望中，我希望本系统能做到定制化信息模块和定制化模板样式。

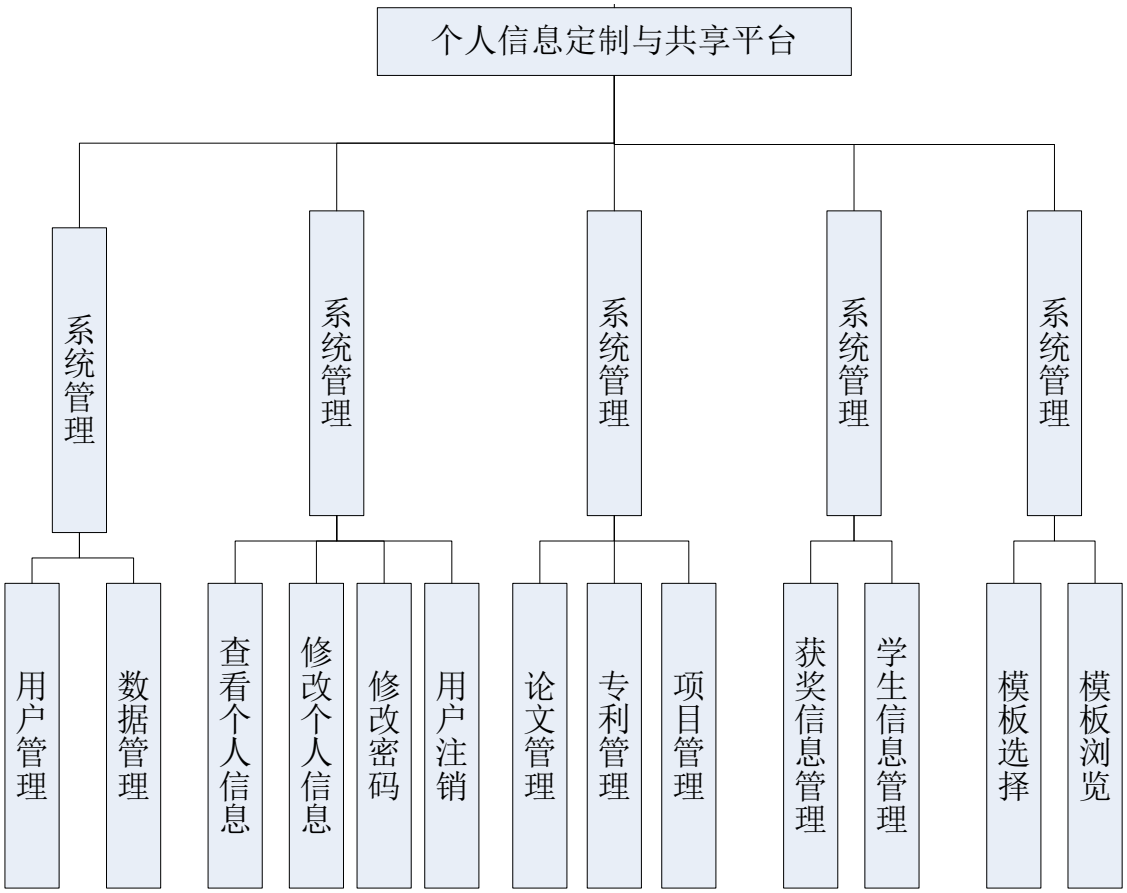


图 3.1 个人信息定制与共享平台功能模块设计

3.2 可行性分析

一个项目或者产品是否能够获得成功，除了是否满足客户对于项目或者产品的特殊功能要求之外，还应该检测项目是否具有理论可行性。下面我们将从经济可行性和技术可行性分别进行探讨。

3.2.1 技术可行性

本系统的开发主要分为前端开发、后端开发、以及数据库开发。用到的技术框架，工具都是目前主流并且成熟的。前端使用到的技术主要是 Html、Css、Jquery[14] 等，后端主要使用 Java，数据库则使用 MySQL 进行建表。整体架构使用 SSM 框架[17]，开发过程都是成熟的企业级开发流程，并且网上有很多文献可供查找与学习，所以从技术层面讲也是可行的。

3.2.2 经济可行性

对于此系统而言，此系统采用 Java[18]语言作为开发语言，前端采用免费模板 AdminLte2[19]作为前端开发模板。除了编译器 IntelliJ IDEA 是试用版之外，其他的软件基本都为免费使用，或者都有一定的试用时间。并且在系统开发过程中用到的第三方依赖包，例如 Alibaba-druid，Spring 框架等都是开源项目，所以开发起来并不需要多大的成本。所以，从经济可行性方面来说，此项目是可行的。

3.3 系统非功能性需求

3.3.1 系统安全性需求

一个系统能否为用户提供可靠的服务，最主要的就是安全性。本文系统在前端页面层面密码框里采用的是“type=”password”” 的密码输入框。目的是让用户是以暗码的形式进行密码输入，而不是明码。这样可以保证输入安全性。并且用户的密码在后台的数据库中存储的时候，本文系统采用了 MD5 加密技术。使密码并非以铭文的形式存在，而是以加密过后的形式进行存放。比如“cqupt”在经过加密之后的存储序列应该是“3a6705cf3c970f4b4fc2a61f870af2d1”。这样保证了，即使黑客知晓数据库中密码，也无法正确进行密码匹配从而达到登陆效果的情况。

能否登陆系统进行数据的查阅以及修改等操作，取决于系统能否进行有效登陆。本文系统后端配置了 Interceptor 拦截器，凡是以“.login”（登录请求界面）结尾的请求都会被拦截，并且进行分析。只有与数据库匹配成功才会进行放行进入主界面，否则拦截器将请求打回登陆界面。从而无法登陆系统。

为了进一步保证数据库系统的安全性，系统在实体类 VO 的传输层面并没有一直使用原实体。而是只在 Dao（数据访问层）才使用原实体 VO。而在服务层 Service（服务层）和 Controller（请求控制层）使用 Dto 的形式进行数据传输，不将所有数据库信息进行暴露。

3.3.2 系统响应速度需求

系统能否为用户提供良好的使用体验与系统的响应速度息息相关。一般而言用户在页面加载时的忍耐时间有限，如果页面响应时间过长，则会严重影响用户体验，可能会使用户直接关闭系统。

在系统架构层面，本文采用的框架以及工具都是现在实行的开发工具，来确保程序的健壮性以及响应速度。在编码层面，本系统尽量的简略不需要的步骤，用最短的代码来表达程序的逻辑。

数据库建库引擎选择 Myisam 引擎，该引擎有一条特点是读取速度快占用资源相对少。也是对于响应速度的一个保证。

3.3.3 系统可扩展性需求

本系统的版本号是 1.0.0，这是第一版程序但并不是最后一版。如果以后能力允许，或者有需要进行再次开发的时候，程序要能够进行扩展维护。所以系统的可扩展性也是重中之重。

本文在建立系统架构的时候，采用的是项目集群的方式，用功能将系统结构进行了划分，不仅可以进行担任开发，更可以进行协同合作。只需将工程建立在 GitHub 即可进行协同开发。

并且在主项目中，采用三层架构的开发模式，并在视图层采用 MVC 三层架构[1]，无论是增加功能，还是增加界面，都不会影响其他功能以及界面。并且在前端页面中也将公共部分进行了抽取，只需要新界界面，并且导入所需头文件，即可进行开发。

3.4 本章小结

本章从系统层面出发，主要阐述了系统功能性需求和系统非功能性需求的思考。并且针对开发系统的现实性也进行了合理的分析。

第 4 章 系统设计

针对第三章提出的系统要求，需求分析等要求。本章将从系统的架构，界面设计，用例设计三个方面对系统进行设计。

4.1 系统架构设计

常见的个人管理系统也都大多以 B/S 体系结构进行开发[12]，本文也不例外，综合了需求分析之后，本文也是使用浏览器-服务器的形式进行开发的。

正如前面所介绍的，这种开发模式一定意义上选定了客户端。并且将系统的功能性实现都放在服务器上。并且对于用户来说也更加友好，用户只需要在个人 PC 机上安装一个浏览器即可使用本系统。

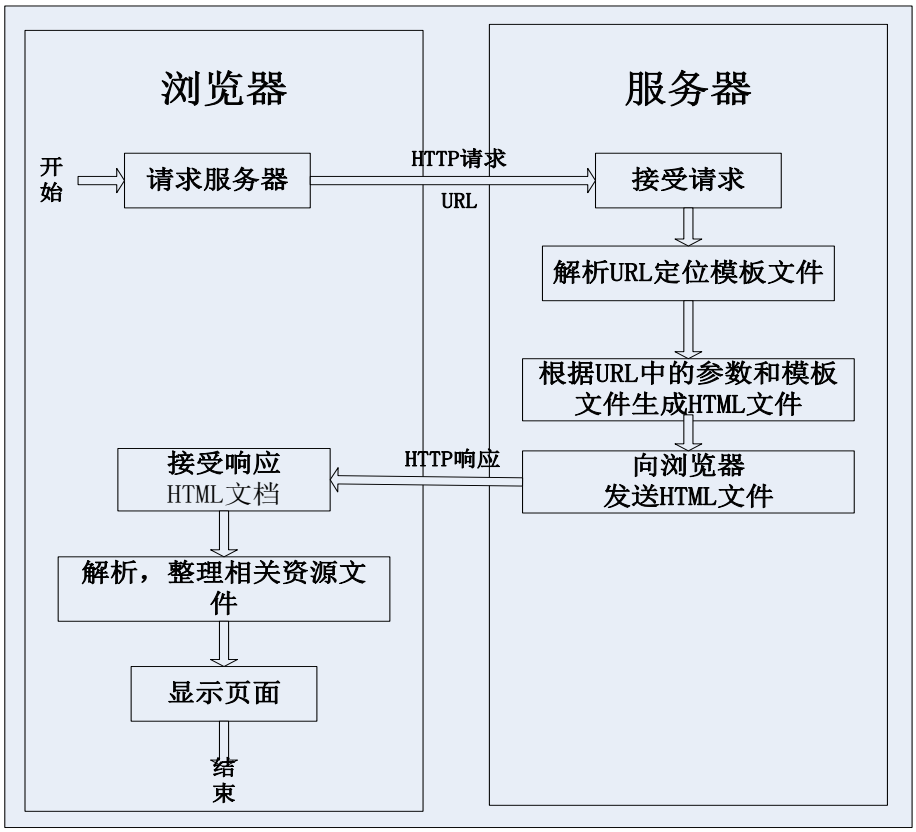


图 4.1 B/S 架构工作原理图

系统的编码架构上则采用三层架构。

三层架构顾名思义就是将系统的整个架构分为三层，分别为：数据访问层，业务逻辑层和界面层。区分层次的目的即为了“高内聚低耦合”的思想，每一层都各司其职，合理交互，这样无论是扩展还是维护都会将成本降到最低。

4.2 界面设计

用户操作界面一定要满足清晰，易理解的系统非功能需求。在进行用例设计之前，根据功能性需求分析的思路，AdminLTE2[17]模板对个人信息管理页进行绘制。

4.3 用例设计

4.3.1 系统登录功能用例设计

由于用户实际分为两种，一种是游客，一种是使用者（即教师）。所以如果当用户身份为教师并且需求对自己的信息进行管理的时候，我们需要进行登陆操作，在数据库进行比对且一致后，方可对个人信息进行修改等操作。

表 4.1 系统登录功能用例设计

用例编号	GP-00001
用例名称	系统登陆功能
用例描述	用户在不登录情况下是游客模式，但是如果用户需要使用本系统的个人模块就必须登陆，登陆成功之后才能对自己的数据进行管理
执行者	系统后台
前置条件	用户还没有登录
后置条件	用户身份校验成功
事件描述流	1.用户打开网页 2.用户点击“登录”（登陆界面跳转按钮） 3.用户输入正确的邮箱与密码 4.数据库匹配用户的账号与密码 5.登陆认证成功，进入个人信息管理界面。

表 4.1 系统登录功能用例设计(续)

异常流描述	用户账号密码匹配错误
业务规则	1.用户账号密码匹配错误，返回登陆界面。

4.3.2 共享信息检索用例设计

由于用户实际分为两种，一种是游客，一种是使用者（即教师）。所以如果当用户身份为教师并且需求对自己的

表 4.2 共享信息预览用例设计

用例编号	GP-00002
用例名称	系统登陆功能
用例描述	用户通过该用例搜索自己需要查看的教师信息。
执行者	系统后台
前置条件	用户输入模糊检索内容
后置条件	用户搜索信息成功
事件描述流	1. 用户打开系统主页 2. 用户在模糊搜索框键入需要查询的信息。 3. 点击搜索按钮 4. 系统搜索用户需要查询的个人信息 5. 系统向游客展示所查个人信息。 6. 共享个人信息查询完成
异常流描述	用户搜索的信息不存在
业务规则	1. 用户搜索的信息有效，即关键词有对应的教师。显示对应教师信息。

4.3.3 共享信息预览用例设计

前面已经提到了本系统需要解决的两个问题，另一个则是个人信息共享。本节就是针对个人信息预览进行的用例设计。

表 4.3 共享信息预览用例设计

用例编号	GP-00003
用例名称	共享信息预览
用例描述	每个用户都有可以选择自己的模板进行展示
执行者	系统后台
前置条件	已经选择好模板和要展示的内容
后置条件	进行模板展示
事件描述流	1. 用户点开我的模板然后选择模板 2. 用户点击我的模板
异常流描述	无
业务规则	1. 仅仅是预览

4.3.4 私有主页信息预览用例设计

个人信息功能作为核心功能之一，只能是用户在进行了有效登陆之后才能够进行显示。用户在进行了有效登陆之后，页面会转到“main.jsp”，即用户的主界面，在主界面里，用户可以自由的进行私人操作。

表 4.4 私有主页信息预览用例设计

用例编号	GP-00004
用例名称	私有主页信息预览
用例描述	等用户正确的输入了账号和密码之后，就可以来到私有信息预览界面了。
执行者	系统后台
前置条件	用户进行了有效登陆
后置条件	将页面跳转入用户主页
事件描述流	1. 用户进行有效登陆 2. 页面跳转入用户主页

表 4.4 私有主页信息预览用例设计(续)

异常流描述	无
业务规则	1. 用户成功会就会自动跳入这个界面

4.3.5 私有信息模糊查询用例设计

随着个人信息的增加，寻找某条记录将会变得越来越难，考虑到这里一点，本系统在每一种类型的信息管理中都提供模糊查询功能，这样用户就可以方便快捷的定位到目标记录。

表 4.5 私有信息模糊查询用例设计

用例编号	GP-00005
用例名称	私有信息模糊查询
用例描述	点击具体的私有信息，例如论文，在搜索框输入模糊查询的内容，点击搜索那按钮
执行者	系统后台
前置条件	正确键入私有信息
后置条件	显示经过检索的信息
事件描述流	1. 用户点击具体内容进入详情页 2. 用户在搜索框键入需要检索的内容 4. 点击搜索按钮 5. 系统后台进行检索，并把筛选过的信息显示在界面上。
异常流描述	无
业务规则	1. 如果用户进行检索的内容是存在的，就显示，否则显示空词条。

4.3.6 私有信息删除新增用例设计

用户对于私人信息的管理应该提供基础的功能，即对私人信息的增删改查功能。这样用户基本信息的量与质量才能得到保障。

表 4.6 私有删除用例设计

用例编号	GP-00006
用例名称	私有信息删除新增
用例描述	用户点击每条私有信息后面的删除和新增按钮，即可进行删除和新增。
执行者	系统后台
前置条件	如果是新增，则都可以点击，如果是删除，则需要信息词条正常加载的情况下才可以进行删除。
后置条件	无
事件描述流	1. 用户点击具体信息，比如论文。 2. 在具体信息加载完成后，就可以点击删除或者新增，进入具体页面，或者刷新界面。
异常流描述	无
业务规则	无

4.3.7 单条论文置顶用例设计

对于论文本文还提供了一种的独特的功能—论文置顶。由于考虑到以后信息的增多而导致用户不能每次都运用个人信息检索功能才能定位，所以本文增加了此功能，用户只需要一次检索，第二步点击置顶，就能够一劳永逸

表 4.7 单条论文置顶用例设计

用例编号	GP-00007
用例名称	单条论文置顶
用例描述	点击论文，在论文界面加载完成后，点击单条论文后面的制定按钮
执行者	系统后台
前置条件	用户有未删除的论文
后置条件	无

表 4.7 单条论文置顶用例设计(续)

事件描述流	1. 点击论文 2. 等待论文加载完毕，点击需要置顶论文的置顶按钮。
异常流描述	无
业务规则	1. 点至之后，系统后台自动进行置顶并刷新。

4.4 本章小结

本章从系统层面对系统进行了介绍。主要从系统架构设计与用例设计对系统进行了剖析。

第 5 章 系统实现

在明确了需求分析，并且做出了主要功能的用例设计之后，接下来就是进行系统的开发了，首先从搭建环境和服务器开始。

5.1 后台服务器的实现

1. 系统的主要框架搭建 SSM 框架

首先打开 IDEA，并且新建一个 Maven 项目。Maven 项目是以依赖的方式来管理项目的。我们只需要在 pom.xml 里面加入我们需要的依赖，Maven 便会自动帮我们进行依赖包的托管。

因为遵循“高内聚，低耦合”的设计理念。所以本系统选择了将依赖版本号抽取出来，进行统一的管理。如图 5.1

```
<!-- 统一的依赖管理 -->
<commons-lang3.version>3.5</commons-lang3.version>
<jstl.version>1.2</jstl.version>
<log4j.version>1.2.17</log4j.version>
<servlet-api.version>3.1.0</servlet-api.version>
<slf4j.version>1.7.25</slf4j.version>
<spring.version>4.3.17.RELEASE</spring.version>
<alibaba-druid.version>1.1.6</alibaba-druid.version>
<mysql.version>5.1.46</mysql.version>
<mybatis.version>3.2.8</mybatis.version>
<mybatis-spring.version>1.3.1</mybatis-spring.version>
<junit.version>4.12</junit.version>
```

图 5.1 主项目依赖版本管理图

2. 系统的结构分层

本系统是严格按照易扩展、可扩展、易管理的编程模式进行开发，所以在结构的划分上必须严格的按照功能进行划分。

本文的项目结构划分为六个子项目，目的是方便以后企业及开发，协同开发，协同工作等。

如图 5.2 就是本系统的结构划分。

接下来，我将对每个子项目进行介绍。

My-graduation-project-commons 项目公共工具项目，包括各种 Utils 工具包等

My-graduation-project-dependencies 项目父级依赖项目

My-graduation-project-domain 实体项目

My-graduation-project-web-admin 主体项目

My-graduation-project-web-ui ui 项目

My-graduation-project-api 对外暴露接口项目

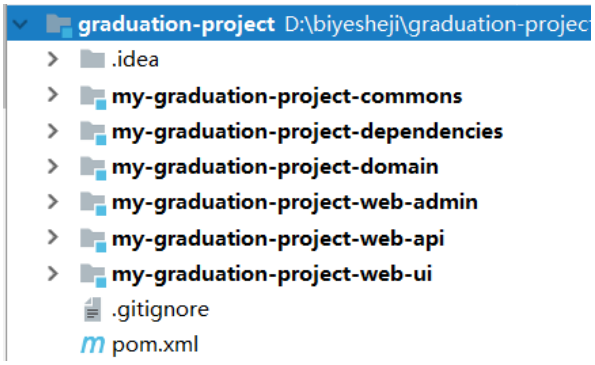


图 5.2 项目结构分层图

3. 总项目 pom 文件解析

总项目需要对对整个项目进行把控和管理，是整个项目的纽带所在，通过父子项目的配置，是整个项目有机的结合在一起。

并且在总项目中需要使用关键词抽取的方式对项目进行管理，在子项目中，只要制定了父项目。就可以使用父项目之中的参数。

本文将版本号进行了抽取，并且规定了整体项目的名称，如图 5.3(a)所示。版本号为“1.0.0-SNAPSHOT”。版本号的编写规范一般为：第一个数字代表大的变动，一般为项目结构的变化，例如原项目进行了重构等；第二个数字代表在大结构无变化的情况下，进行的修改，例如增加减功能等；第三个数字代表最小的变动，可能是原功能的逻辑变化，或者是修复了一个非常微小的 bug 等。

至此还不够，还需要在父项目中添加各子项的项目信息，使整个大的项目形成一个有序的整体。如图 5.3(b)所示就是在父项目中添加各子项目。

```
<groupId>com.cqupt.ysc</groupId>
<artifactId>graduation-project</artifactId>
<version>1.0.0-SNAPSHOT</version>
<packaging>pom</packaging>
```

(a) 主 pom 文件中项目名称及版本号配置图

```
<modules>
    <module>my-graduation-project-dependencies</module>
    <module>my-graduation-project-commons</module>
    <module>my-graduation-project-domain</module>
    <module>my-graduation-project-web-ui</module>
    <module>my-graduation-project-web-api</module>
    <module>my-graduation-project-web-admin</module>
</modules>
```

(b) 主 pom 文件中子项目依赖图

图 5.3 主 pom 文件解析图

4. 数据库的连接配置

一个项目的请求应该是前端->后端->数据库->后端->前端。在后端总体开发架构配置过后，就需要将后端和数据库进行连接。本文使用的是 MySQL 数据库,MySQL 的配置文件如图 5.4 所示。在 a 图中主要配置了数据库驱动信息、数据库连接地址、数据库账号和密码。在 b 图中则主要是数据库连接池的配置与数据库测试的配置。

首先要和数据库交互，就必须要有数据库驱动才能和数据库进行交互。在配置了数据库驱动之后就需要配置数据库地址，本文采用的是本地数据库，所以配置的地址就是相应的 localhost:你的数据库端口。后面的代码类似 GET 方式获取的请求，配置的是数据库编码等信息。再下来是数据库用户名的和密码的配置，因为要确保安全性所以需要有账号密码对数据进行保护。

接下来配置的是数据库线程池，合理的配置线程池可以将请求速度合理化最大化。本文配置的是只能注册一个线程池，最小连接一个，最大 Active 线程池 20 个。

```
# JDBC
# MySQL 8.x: com.mysql.cj.jdbc.Driver
jdbc.driverClass=com.mysql.jdbc.Driver
jdbc.connectionURL=jdbc:mysql://localhost:3306/myproject?\
  useUnicode=true&characterEncoding=utf-8&useSSL=false
jdbc.username=root
jdbc.password=
```

(a) 数据库配置文件图

```
# JDBC Pool
jdbc.pool.init=1
jdbc.pool.minIdle=3
jdbc.pool.maxActive=20

# JDBC Test
jdbc.testSql=SELECT 'x' FROM DUAL
```

(b) 数据库连接池参数图

图 5.4 数据库 propeties 文件图

5. 配置阿里巴巴开源连接池 DRUID 参数

上一步已经对数据库和连接池的配置进行了配置。下面就要在配置文件中将 propeties 文件进行加载。具体的配置方式见图 5.5(a)和 5.5(b)。配置中可见的“${jdbc.***}”就是因为前面我们已经配置过 propeties 文件，所以在这里我们只需要按照名字进行引用即可。

```
<!-- 数据源配置, 使用 Druid 数据库连接池 -->
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
      init-method="init" destroy-method="close">
    <!-- 数据源驱动类可不写, Druid默认会自动根据URL识别DriverClass -->
    <property name="driverClassName" value="${jdbc.driverClass}"/>

    <!-- 基本属性 url、user、password -->
    <property name="url" value="${jdbc.connectionURL}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>

    <!-- 配置初始化大小、最小、最大 -->
    <property name="initialSize" value="${jdbc.pool.init}"/>
    <property name="minIdle" value="${jdbc.pool.minIdle}"/>
    <property name="maxActive" value="${jdbc.pool.maxActive}"/>
```

(a) 阿里巴巴开源连接池连接配置图

```
<!-- 配置获取连接等待超时的时间 -->
<property name="maxWait" value="60000"/>

<!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
<property name="timeBetweenEvictionRunsMillis" value="60000"/>

<!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->
<property name="minEvictableIdleTimeMillis" value="300000"/>

<property name="validationQuery" value="${jdbc.testSql}"/>
<property name="testWhileIdle" value="true"/>
<property name="testOnBorrow" value="false"/>
<property name="testOnReturn" value="false"/>

<!-- 配置监控统计拦截的filters -->
<property name="filters" value="stat"/>
```

(b) 阿里巴巴连接池配置图

图 5.5 阿里巴巴开源连接池 DRUID 配置图

6. 配置 Mybaits 持久层框架

持久层框架是简化开发人员在 sql 语句的书写上花费的时间的框架。

为了和数据库进行交互我们首先是要配置的就是核心 sqlSession。后台与数据库的交流可以简单的认为 Mybatis 一系列的配置目的是生成类似 JDBC 生成的 Connection 对象的 SqlSession 对象，这样才能与数据库开启“沟通”，通过 SqlSession 可以实现增删改查，不过现在主流的方式是使用 Mapper 接口。

在 sqlSession 配置完毕后，我们还需要进行实体类映射的配置。由于前面已经提到过实现类项目存在的包（见图 5.2 所示）。这里我们只需要将路径映射过去即可，见图 5.6 的用于配置实体类所在的包。

上段提到过，要和数据库开启沟通，现在主流的方式是使用 Mapper 接口。这里我们使用“**mapper.xml”的形式进行 sql 语句的书写。所以需要配置 mapper 文件的映射（见图 5.5 中配置对象关系映射配置文件）。

而 mapper 文件适合 dao 层接口中方法一一对应的，这里我们还需要配置 dao 层的位置。这样在加载的时候才能生效（如图 5.6 扫描 Mapper）。

```
<!-- 配置 SqlSession -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <!-- 用于配置对应实体类所在的包，多个 package 之间可以用 ',' 号分割 -->
    <property name="typeAliasesPackage" value="com.cqupt.ysc.graduation.project.domain"/>
    <!-- 用于配置对象关系映射配置文件所在目录 -->
    <property name="mapperLocations" value="classpath:/mapper/**/*.xml"/>
    <property name="configLocation" value="classpath:/mybatis-config.xml"></property>
</bean>


<!-- 扫描 Mapper -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.cqupt.ysc.graduation.project.web.admin.dao" />
</bean>
```

图 5.6 持久层框架 Mybaits 配置图

7. SpringMVC 注解和视图文件解析配置

本文的开发是基于注解开发，让 SpringMVC 自己扫描被注解过的类，然后再将这些被注解过的类加载出来。所以就需要配置 Annotation 自动注册 Bean,并且需要 Annotation 注解映射的支持。

在方法映射时，本文希望直接通过返回 String 类型的字符串就可以定位到目的视图，所以还需要配置视图文件解析。值得一提的时返回的字符串相当于文件的名字，但是为了定位视图所在位置，还需要加上前置地址，即“prefix”和后置文件类型，即“suffix”。所有的配置见图 5.7

```
<!-- 加载配置属性文件 -->
<context:property-placeholder ignore-unresolvable="true"
                              location="classpath:myproject.properties"/>

<!-- 使用 Annotation 自动注册 Bean,只扫描 @Controller -->
<context:component-scan base-package="com.cqupt.ysc.graduation.project.web.admin"
                        use-default-filters="false">
    <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>
</context:component-scan>

<!-- 默认的注解映射的支持 -->
<mvc:annotation-driven />

<!-- 定义视图文件解析 -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="${web.view.prefix}"/>
    <property name="suffix" value="${web.view.suffix}"/>
</bean>
```

图 5.7 SpringMVC 注解和视图文件配置图

8. 全局拦截器配置

为了保证系统的安全性，所以本文配置了全局拦截器，对任何身份不明的请求进行拦截。如果没有按照要求进行登陆，而直接请求详情界面的，我们给予拦截并且将请求打回登陆界面。直到登录成功之后，将当前用户的信息存放在 session 之中，并且在拦截器进行拦截之后，在服务器端取到了有效登陆信息之后，才进行放行。所有的配置如图 5.8 所示。“mvc:exclude-mapping path”为拦截过滤路径，“mvc:mapping path”为拦截路径。

```
<mvc:interceptors>
    <mvc:interceptor>
        <!-- 拦截什么路径（所有路径） -->
        <mvc:mapping path="/**"/>
        <!-- 拦截过滤 -->
        <mvc:exclude-mapping path="/login"/>
        <mvc:exclude-mapping path="/static/**"/>
        <mvc:exclude-mapping path="/visitor/search"/>
        <bean class="com.cqupt.ysc.graduation.project.web.admin.web.interceptor.LoginInterceptor" />
    </mvc:interceptor>

    <mvc:interceptor>
        <!-- 拦截什么路径（所有路径） -->
        <mvc:mapping path="/**"/>
        <mvc:exclude-mapping path="/visitor/search"/>
        <bean class="com.cqupt.ysc.graduation.project.web.admin.web.interceptor.PermissionInterceptor"
    </mvc:interceptor>
</mvc:interceptors>
```

图 5.8 拦截器配置文件图

图 5.8 向我们展示了拦截器的配置，但是与前面的配置不同，拦截器是需要具体逻辑才能生效的工具。里面提到的 LoginInterceptor 登陆拦截器和 PermisionInterceptor 权限拦截器还需要我们手动去对象中实现我们的逻辑，具体实现如图 5.9 和图 5.10 所示。体现用户是否登陆的具体实现方法是，如果用于成功登录，系统就会将用户信息放入服务器的 Session 中，这样拦截器只要在拦截到请求后去 Session 中检查是否有用户信息即可。有则已经登录，没有则没有登陆成功，如果没有登录成功，就将页面返回登录页。

5.2 前端文件及结构剖析

用户界面点的美观程度将直接影响用户的体验。本文使用的的是轻量级框架 AdminLTE 2 作为模板，在其之上进行再开发。

```java
/**
 * 登录拦截器
 */
public class LoginInterceptor implements HandlerInterceptor {
    public boolean preHandle(HttpServletRequest httpServletRequest,
                             HttpServletResponse httpServletResponse,
                             Object o) throws Exception {
        TbUser user = (TbUser) httpServletRequest.getSession().getAttribute(ConstantUtils.SESION_USER);
        if (user == null){
            httpServletResponse.sendRedirect( s: "/login");
        }
        return true;
    }
```

图 5.9 登陆拦截器具体实现图

```java
/**
 * 权限拦截器
 */
public class PermissionInterceptor implements HandlerInterceptor {
    public boolean preHandle(HttpServletRequest httpServletRequest,
                             HttpServletResponse httpServletResponse, Object o) throws Exception {
        return true;
    }

    public void postHandle(HttpServletRequest httpServletRequest,
                           HttpServletResponse httpServletResponse,
                           Object o, ModelAndView modelAndView) throws Exception {
        //以login结尾的请求
        if (modelAndView.getViewName().endsWith("login")){
            TbUser user = (TbUser) httpServletRequest.getSession().getAttribute(ConstantUtils.SESION_USER);
            if (user != null){
                httpServletResponse.sendRedirect( s: "/main");
            }
        }
    }
```

图 5.10 权限拦截器具体实现图

5.2.1 前端界面文件的分层结构

前面提到过本文将严格遵循开发规则，所以前端界面的开发同样遵循“事不过三，三则重构原理”。将同样的模块进行了抽取，并在开发需要的时候进行注入。如图 5.11 所示为抽取出来的五个公共模块，分别是权限模块、两个依赖文件模块、菜单模块、导航模块。

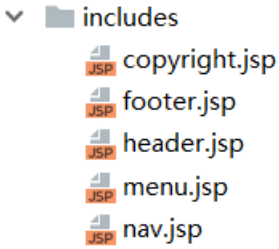


图 5.11 前端界面结构图

5.2.2 前端界面骨架结构解析

如图 5.11 的 includes 文件中包含的五个 jsp 文件组合即为前端界面骨架结构。

每个文件都各司其职，其中 header.jsp 和 footer.jsp 不是实际的样式界面，而是需要的样式依赖，每个前端页面都必须导入样式依赖，里面的组件才能够生效。而 menu.jsp，nav.jsp 和 copy.jsp 是具体的样式，如图 5.12 所示。

5.3 功能用例的实现

以下内容是对于上一章提出的用例设计进行的具体实现，包括具体配置，核心代码与前端页面展示。

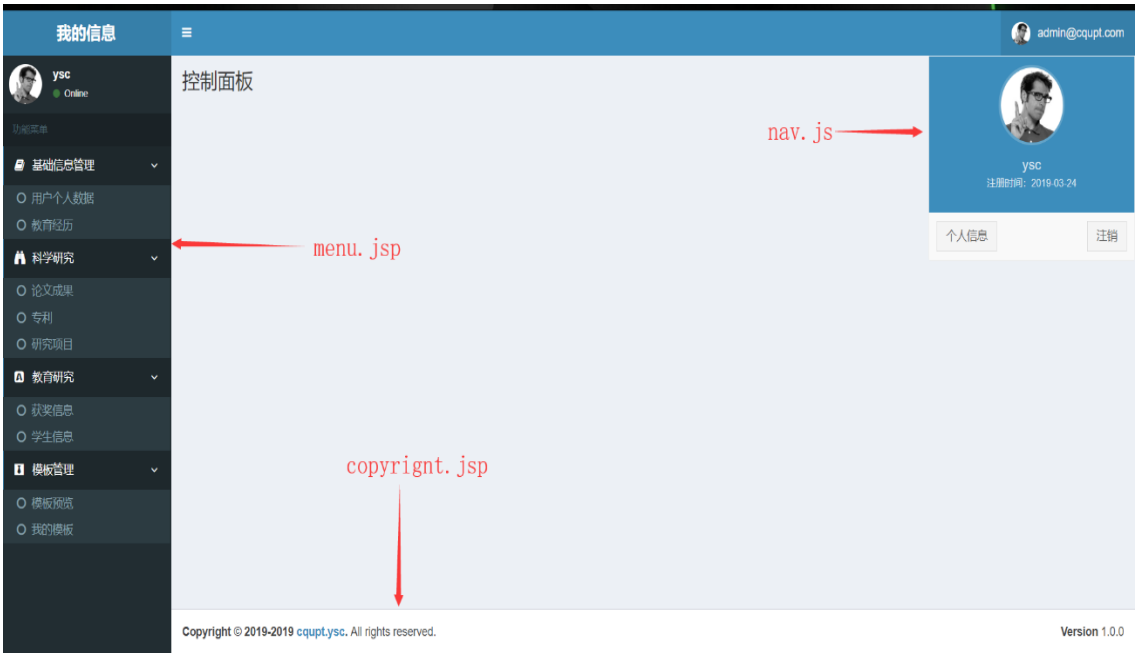


图 5.12 前端界面骨架结构解析图

5.3.1 系统登陆功能用例设计

系统登录功能是一个非常重要的功能，要同时保证有效和安全性。首先本系统的密码都是以 MD5 加密方式存储的。其次在系统后台我们也给登陆系统增加了两个拦截器（LoginInterceptor&PermissionIntercepor），保证无效登陆的状态时无法无法跳转个人主页，更无法请求数据库。

并且本文在前端也规范了邮箱和密码的输入（前端校验），邮箱格式必须有邮箱结尾，否则前端界面检测无法通过。密码则采用暗码（type = “password”）的方式书写，只能看到字符个数，无法看到具体字符。进一步保证了安全性。

并且长远考虑，本系统还预留了第三方登录接口（QQ 和微信）。以便本系统在以后的扩展和更新中可以方便的与第三方合作。

图 5.13 系统登陆界面图

登陆是一个复杂的请求，核心代码见图 5.14。用户在正确的键入账号密码的情况下，系统允许登录。并且将当前用户的信息存放在服务器的 session 中，在进行

请求的时候拦截器就会使用 getSession()去 session 里面查询是否有用户信息，如果有就放行，如果没有就转回登陆界面。

```
/**
 * 登录拦截器
 */
public class LoginInterceptor implements HandlerInterceptor {
    public boolean preHandle(HttpServletRequest httpServletRequest,
                             HttpServletResponse httpServletResponse,
                             Object o) throws Exception {
        TbUser user = (TbUser) httpServletRequest.getSession().getAttribute(ConstantUtils.SESION_USER);
        if (user == null){
            httpServletResponse.sendRedirect( s: "/login");
        }
        return true;
    }
```

图 5.14 系统登陆功能核心代码图

5.3.2 共享信息预览用例设计

游客界面是本文的个人信息管理与共享平台比较独特的功能，及就是分享功能，用户通过首页的一个 input 搜索框键入自己需要查询的关键字，然后前端页面通过 http 协议请求到系统后台。然后系统后台通过“CONCAT，LIKE，”%”” 等关键字进行 SQL 语句拼接，并通过 Mabatis 持久层框架进行查询。然后将各种结果封装好并放入 Model 返回给前端界面，前端界面通过 jstl 的 core 标签进行循环取值并显示。

功能实现的第一步是配置拦截器，由于此功能是在未登录状态下进行请求，所以首先得配置拦截器。让拦截器对“/visitor/search”这个请求放行。配置详情见图 5.15(b)。添加了此配置之后，拦截器便不再对此请求进行拦截，才能够跳转到共享的个人信息展示页面。

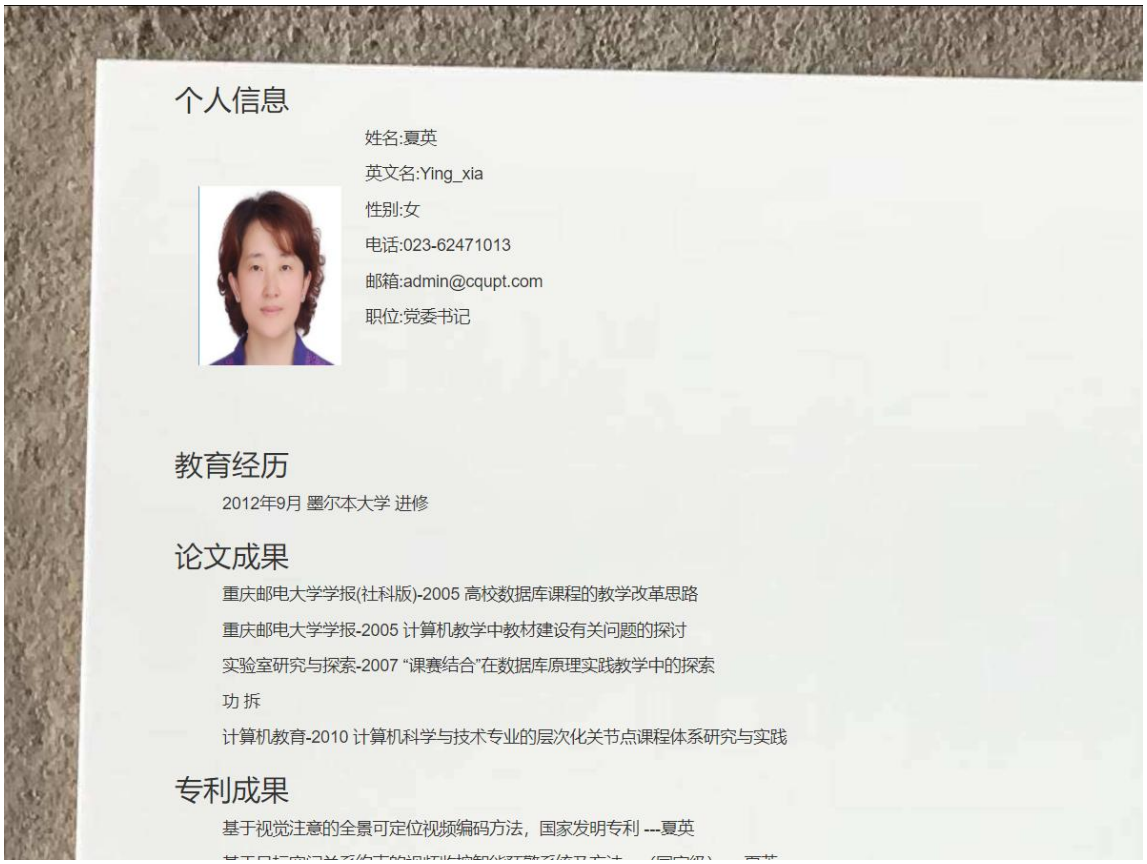
然后就是在模糊搜索框键入关键词，后台会保存为“Keyword”,游客页面搜索框见 5.16(b)。模糊搜索允许对五个字段进行搜索，分别是：名字的拼音，英文名，用户，职位与学历。

在获取到信息之后，交由前端页面展示，具体样式见图 5.15(b)所示。

```
<!-- 拦截器配置 -->
<mvc:interceptors>
    <mvc:interceptor>
        <!-- 拦截什么路径（所有路径） -->
        <mvc:mapping path="/**"/>
        <!-- 拦截过滤 -->
        <mvc:exclude-mapping path="/login"/>
        <mvc:exclude-mapping path="/static/**"/>
        <mvc:exclude-mapping path="/visitor/search"/>
        <bean class="com.cqupt.ysc.graduation.project.web.admin.web.interceptor.LoginInterceptor" />
    </mvc:interceptor>

    <mvc:interceptor>
        <!-- 拦截什么路径（所有路径） -->
        <mvc:mapping path="/**"/>
        <mvc:exclude-mapping path="/visitor/search"/>
        <bean class="com.cqupt.ysc.graduation.project.web.admin.web.interceptor.PermissionInterceptor"
    </mvc:interceptor>
</mvc:interceptors>
```

(a) 拦截器对共享信息界面放行配置图



(b) 共享信息界面预览图

图 5.15 共享信息预览用例主要代码及前端页面展示图

5.3.3 共享信息检索用例设计

游客界面是本文的个人信息管理与共享平台比较独特的功能，也是需要设计感的页面。本文选择了比较简洁的设计，整个界面上只有两个元素可供点击，一个 input 元素，一个 button 元素，一个是登录，一个是模糊搜索。本文并没有提供注册服务，因为本系统是依托于教师系统实现的，所以不开放社会人员注册。

主要代码实现见图 5.16(a)图所示，实现效果见 5.16(b)图所示。

```
<a target="view_window" style="..." href="/login" class="div1">
    <button button type="button" class="btn btn-primary">登录</button>
</a>

<c:if test="${message != null}">
    <div class="alert alert-danger alert-dismissible">
        <button type="button" class="close" data-dismiss="alert" aria-hidden="true">×</button>
            ${ message}
    </div>
</c:if>
<form action="/visitor/search" method="get">
    <div class="input-group input-group-sm" style="...">
        <input type="text" name="keyword" class="form-control pull-right" placeholder="搜索">

        <div class="input-group-btn">
            <button type="submit" class="btn btn-default"><i class="fa fa-search"></i></button>
        </div>
    </div>
</form>
```

(a) 共享信息前端主要代码图

(b) 共享信息检索功能前端实现图

图 5.16 共享信息功能主要代码及前端实现图

共享信息检索的主要流程为：

1. 用户输入要检索的信息作为 keyword

2. 后台获取到用户的输入继而去数据库进行模糊搜索。

2. 在获取到输入后向前端页面返回数据，并封装成前端界面向用户展示。

5.3.4 私有主页信息预览用例设计

私有页是在用户进行了正确登陆之后会跳转的页面，首先在登录页用户需要进行合法登陆，正确的键入账号密码。系统后台在获取到用户输入后，将用户数据与数据库中数据进行比对。正确无误后，将用户信息放在 Session 中后将页面请求转向 main.jsp 页面。此时权限拦截器 PermissionIntercepor 会对请求进行拦截，在拦截器内部查看 Session 中是否有合法用户信息，校验完成后对请求进行放行。

此页面总的来说实现难度较低，主要就是引用前端界面五个框架（主要利用 JSP 标签的 include 方法）。

主要代码见图 5.17 所示。具体实现参见图 5.18。

```
<html>
<head>
    <title>我的信息 | 控制面板</title>
    <jsp:include page="../includes/header.jsp"/>
</head>

<body class="hold-transition skin-blue sidebar-mini">
<div class="wrapper">
    <jsp:include page="../includes/nav.jsp"/>
    <jsp:include page="../includes/menu.jsp"/>
```

(a) 头文件、导航文件、菜单文件引用图

```
    <jsp:include page="../includes/copyright.jsp"/>
</div>

<jsp:include page="../includes/footer.jsp"/>
```

(b) 权限、底部依赖文件引用图

图 5.17 主页面文件引用图

图 5.18 私有主页信息预览图

5.3.5 私有信息模糊查询用例设计

可能由于时间的积累，个人数据就会变得越来越多，管理的难度也会相应的变大。尤其当我们需要检索某一条具体信息的数据的时候，所以在这种思考下。本系统在每种信息下都增加了模糊搜索，这里就用论文管理距离。

用户在搜索框键入自己想要进行搜索的内容，继而请求转入后台，带着关键词的请求到了后台之后。后台将关键词（keyword）封装在特殊的字段（需要被用来模糊查询的字段）中，并在数据库中用数据库语言“CONCAT，LIKE,”%”” 等关键字进行 SQL 语句拼接，并通过 Mabatis 持久层框架进行查询。然后将结果重新封装成一个 list。然后刷新当前界面，完成模糊搜索功能。模糊搜索框位置如图 5.19 所示，具体代码实现见图 5.20。

图 5.19 模糊搜索框展示图

```
/**
 * 模糊查询
 * @param keyword
 * @param model
 * @return
 */
@RequestMapping(value = "searchPaper",method = RequestMethod.POST)
public String searchPaper(String keyword, Model model){
    List<Paper> papers = researchService.searchPaper(keyword);
    model.addAttribute( s: "papers",papers);
    return "research_paper";
}
```

图 5.20 模糊搜索具体代码实现图

5.3.6 私有信息删除新增管理用例设计

本系统中固定有几种信息的管理，如论文，专利，研究项目等。相应的本系统也会提供单个对象的新增删除等功能，新增论文界面如图 5.21 所示，后台具体逻辑见图 5.22。

图 5.21 新增论文展界面展示图

```
/**
 * 新增论文
 */
@RequestMapping(value = "savePaper",method = RequestMethod.POST)
public String savePaper(HttpServletRequest httpServletRequest,PaperDto paperDto){
    if (paperDto == null){
        return "paper_form";
    }

    paperDto.setEmail(EmailUtils.getUserEmail(httpServletRequest));
    paperDto.setOrdernum(paperOrdNum+1);

    //封装完成，进行插入。
    researchService.savePaper(paperDto);

    return "redirect:/research/paper";
}
```

图 5.22 新增论文展界面展示图

删除论文功能前端界面展示见图 5.23，具体后台接口见图 5.24。

图 5.23 论文删除功能前端展示图

```
/**
 * 删除paper
 */
@RequestMapping(value = "deletePaper")
public String deletePaperById(Long id){
    researchService.deletePaperById(id);
    return "redirect:/research/paper";
}
```

图 5.24 论文删除功能后台接口展示图

5.3.7 单条论文置顶用例设计

越来越多的数据记录会增加我们的维护难度和管理复杂度，处于这种考虑。本文增加了论文置顶功能，用户可以随意的将访问的论文置顶。继而达到论文的最快速度定位，从而减少检索时间，增加用户体验。

具体的解决方案是：在数据库建表时对于论文表（Paper）增加 orderNum（论文排序号）字段，此字段用来给单条论文 Paper 增加唯一序列号。

论文置顶功能前端页面展示见图 5.25，接口见图 5.26

图 5.25 论文置顶功能前端页面展示图

```
/**
 * 论文置顶
 */
@RequestMapping(value = "paperToTop",method = RequestMethod.GET)
public String paperToTop(Long id){
    PaperDto paperDto = new PaperDto();
    paperDto.setId(id);
    paperDto.setOrdernum(paperOrdNum+1);
    System.out.println(paperDto);
    researchService.updatePaperById(paperDto);
    return "redirect:/research/paper";
}
```

图 5.26 论文置顶功能后台接口展示图

5.4 用例测试

本节测试了电脑浏览器客户端的基本操作功能。

5.4.1 测试环境

本文选择了“华硕 Zx50J”机型作为电脑的客户端的测试设备，具体配置见图 5.1 所示。后台服务器的配置在第五章第一节已经介绍过了，这里不再进行赘述。

表 5.1 个人电脑具体配置

设备型号	华硕 Zx50J
内存	8G
网络带宽	8M
操作系统	Windows10
测试位置	重庆邮电大学

5.4.2 功能测试

按照普通用户的操作流程和方式模拟用户操作，进行基本的功能测试。测试结果如表 5.2 所示。

表 5.2 功能测试结果

操作	预期结论	测试结果
系统登录	使用邮箱和密码进行登陆，登录成功才能转入个人主页	符合预期
共享信息预览	在主页键入需查教师信息，如果查到信息则进行展示	符合预期
共享信息检索	在主页输入需查教师信息，点击搜索，系统进行搜索	符合预期
私有主页信息预览	登陆成功后，系统进行页面跳转，转入对应个人信息页面	符合预期
私有信息模糊查询	在具体信息界面键入所需查找关键字，系统进行检索并展示	符合预期
私有信息删除新增	删除具体个人信息，如论文	符合预期
单条论文置顶	一键论文置顶	符合预期

本文在第五章的第三节的用例实现中，每个用例均有运行成功的系统展示图。由于在前面已经进行过具体功能的实例展示图，所以在这里就不重新展示图片。如果需要参考具体前端展示页面，请参考第五章的第三节。

5.5 本章小结

本章主要对系统的前端架构，后台系统架构两个方面对于系统进行了剖析。对于重要功能，均有核心代码对于功能进行描述。并且按照第四章的用例设计完成了系统的开发工作并进行了相关功能的测试。

第 6 章 总结与展望

6.1 总结

本文从人才和“岗位”匹配的角度出发，充分的考虑到个人信息管理的复杂度。我们每个人拥有的信息会越来越多，首先不方便管理，其次我们还需要这些信息能够为我们创造价值。为了解决这两个问题，我决定开发一款具有个人信息管理与共享的系统，虽然本文是依托于教师系统实现的。但是实际上我们可以将模式照搬入公司内部招聘，或者是人才管理系统等等，这里强调的永远是一种模式，而不是一成不变的教师系统。

本文以 Java 做为后台开发语言，Html、CSS、Jqery 作为前端开发语言设计并实现了一个“个人信息定制与共享平台”。本系统实现了游客状态下的他人信息检索功能、他人信息预览功能；登录状态下的个人信息的管理功能、论文管理、专利管理、项目管理、学生管理、模板管理、模板预览、获奖管理。从而满足用户的功能性需求，带来比较好的用户体验。除此之外，基于数据安全性的考虑，本系统在后台实现了 Interpetor 拦截器对于不明请求进行拦截，并且数据库中密码也经过 MD5 加密之后再进行存储，最大程度保证系统安全性。基于开发规范的考虑，本系统严格遵循开发规则，无论是大到项目的建立（父子依赖）还是小到实体中属性的命名（驼峰命名法）都严格遵循企业开发规范。这样做的目的是方便未来的更新与维护。

最后，本文对系统进行了功能测试，对系统的各个功能以及预期实现都进行了测试。测试结果显示，系统的各个功能都可以稳定运行。

6.2 后续研究工作展望

本次设计虽然基本完成了需求中涉及到的分析点，但是还有很多不足和需要修缮的地方。比如说，本系统并没有开设模块定制功能，让用户真正意义上的定制自己的管理系统，定制模块，定制信息等。并且本文一直强调的是一种管理模式，只是这种管理模式依托于教师系统及进行了实现。并没有做到范式开发，如果做到

了这样更可以体现“模式”概念。因此本系统还有很大的进步空间，在以后的学习生活中，随着个人专业技能的不断提升，希望可以完善更多功能，真正做到一次编写，到处使用。

参考文献

[1] 章仁棠. 基于 Web 三层架构的高校科研管理系统的设计与实现[J]. 通信信息，2019, (03), 182-183.

[2] 余久久, 杨丽萍. 基于 Android 平台的轻量级招聘系统的设计[J]. 西昌学院学报(自然科学版), 2019, 33(01), 84-87+108.

[3] 段嘉奇, 柴玉梅. 基于 B/S 架构的高校教务管理系统设计与实现[J]. 科技传媒, 2019, 11(07), 113-114+137.

[4] 管清望. 校园信息定制系统的应用研究[D]. 吉林大学, 2017.

[5] Yuan Sun, Xinjie Zhou, Anand Jeyaraj, Rong-An et al. The impact of enterprise social media platforms on knowledge sharing[J]. Journal of Enterprise Information Management, 2019, 32(2).

[6] Yuan Sun, Xinjie Zhou,Anand Jeyaraj, Rong-An Shang,Feng Hu. The impact of enterprise social media platforms on knowledge sharing[J]. Journal of Enterprise Information Management, 2019, 32(2).

[7] 王文韬, 谢阳群, 童咏昕等. 国外个人信息管理现状及发展动态述评[J]. 图书馆论坛, 2015, 35(12), 124-133.

[8] 崔斌, 高军, 童咏昕等. 新型数据管理系统研究进展与趋势[J]. 软件学报, 2019, 30(1): 164-193.

[9] IntelliJ IDEA. IntelliJ IDEA[OL]. https://baike.baidu.com/item/IntelliJ%20IDEA, 2019. 1. 2.

[10] Apache Tomcat. Apache Tomcat[OL]. https://baike.baidu.com/item/tomcat/, 2019.

[11] Wampserver64. Wampserver64[OL]. https://baike.baidu.com/item/WampServer/8205773?fr=aladdin, 2019-02-21.

[12] Navicat for MySQL. Navicat for MySQL[OL]. https://baike.baidu.com/item/Navicat%20for%20MySQL/3815972?fr=aladdin/, 2018-08-04.

[13] MVC. MVC[OL]. https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6?fromtitle=mvc&fromid=85990/, 2017-04-23.

[14] B/S. B/S[OL]. https://baike.baidu.com/item/B%2FS%E7%BB%93%E6%9E%84?fromtitle=b%2Fs&fromid=219020/, 2019-04-13.

[15] SSM. SSM[OL]. https://baike.baidu.com/item/SSM/18801167, 2018-11-27.

[16] 黑马程序员. 响应式 Web 开发项目教程 (HTML5+CSS3+Bootstrap)[M]. 人民邮电出版社, 2017.

[17] 李洋. SSM 框架在 Web 应用开发中的设计与实现[J], 计算机技术与发展, 2016, 26(12), 190-194.

[18] AdminLTE2. AdminLTE2[OL]. http://adminlte.la998.com/documentation/, 2018-7-12.

[19] Oracle. Oracle[OL]. https://docs.oracle.com/en/java, 2018.

致谢

此次研究工作得以顺利进行并成功完成，首先要对我的指导老师致以真诚的敬意。感谢老师在此次工作中给予我们的耐心的讲解和疑难解答，尤其是在论文写作，开题报告写作等方面，感谢老师的悉心指导和实时的工作监督，使得我得以顺利完成此次的研究工作。

其次，我要感谢的是我们的学校。我们学校一直以来都帮我们买下了知网，维普等大型学术网址的下载权，对我们的资料查询提供了巨大的帮助。

最后要谢谢我的学校和学校老师们给予的栽培和悉心指导。在学习学习和生活的四年里，不管是在学习方面还是生活方面，自己都收获颇丰。我的大学生活因此而充实和美好。

另外毕业设计作为我在学校的最后一个也是最大的任务，而且我也将作为一名即将踏入社会的开发工程师。我想尽我所能将这个系统做到最好，因此我决定用大项目的结构，企业开发的规则约束自己。但是这个过程并不容易。面对这些挑战，通过一次次的在专业论坛寻找知识，查找资料，一次次的与同学相互沟通，最终所有的问题都能够迎刃而解。在这个过程中，不论是自己查阅资料的能力，阅读理解的能力，还是与人交流的能力，都有所提升。

感谢老师，感谢同学们，也感谢给我提供这个平台的我的大学。

附录

一、英文原文：

Designing an MVC Model for Rapid Web Application Development

Dragos-Paul Pop*, Adam Altar

1. Introduction

Web application development has come a long way since the beginning of the World Wide Web. A myriad of technologies and programming languages are now used to build web applications, but because of the way the World Wide Web evolved and because of the speed at which it did so these technologies didn’t really have time to evolve and cope with the pace. Many players tried to come up with different and exotic technologies to mainly improve the user experience and help developers build faster and more powerful web applications. Some of these technologies have played an important role in web development but have seen an important drop in usage over the last years, like Java Applets and Microsoft Silverlight. On the other hand many technologies have evolved from simple toys to powerful and important parts of todays’ web ecosystem, like JavaScript, Flash and XML.

The web environment today uses HTML and CSS to present data to users and interaction is accomplished via JavaScript. These technologies are called “front-end” or “client-side” technologies. On the other hand, “back-end” or “server-side” technologies refer to data storage and processing technologies.

2. Problem formulation

Front end and back end technologies come together to build web applications, but because the World Wide Web has evolved at such a fast pace and because developers need to use a rather large number of technologies to build just a single web application, the result of their work is oftentimes difficult to maintain and fix. Developers combine HTML code with server side programming languages to create

dynamic web pages and applications and this leads to highly entangled and unmaintainable code.

Another problem has grown out of the fact that web technologies are increasingly used to build all sorts of complex applications. Microsoft embraces web technologies to encourage developers to build applications for its' latest operating system, Windows 8. Also, a lot of frameworks exist that help web developers write applications for mobile devices, like PhoneGap and Appcelerator Titanium. Even more, a mobile operating system is in the works and devices are expect later this year that uses web technologies entirely for the developer API (Mozilla Firefox OS). For these reasons a web application is generally created by a whole team of specialized developers, each working with their favorite technology, like HTML and CSS for the presentation layer, JavaScript for client-side interaction, PHP (or ASP, Java, Python, Pearl, Ruby, etc.) for server-side logic and MySQL (or Oracle Database, Microsoft SQL Server, etc.) for data storage and management.

Each of these specialist needs to work with their colleagues in such a way that their code pieces fit inside the overall design of the application. For example, the client-side (data presentation) developer needs to alter the HTML and CSS code in such a way that he doesn't break the server-side developers' code that resides in the same file. Also, when a database developer alters the schema for an application the server-side developer may need to change a lot of code to make the application work.

The important thing to note here is that there is an acute need to separate presentation from logic and data storage in an application.

There are some application design paradigms and patterns that offer solutions to this problem, but the focus nowadays is on the MVC pattern.

In conducting the research we have used our experience is building web applications with various systems and frameworks and we have tried to identify both strengths and weaknesses of these systems while providing our own view on how these practices could be improved. Studied frameworks and systems include: Symfony, CakePHP, CodeIgniter Zend Framework, Laravel, Fuel PHP, Ruby on Rails and ASP.NET MVC.

3. The MVC pattern and literature overview

In this section we will review the current status of the research in this field and take a look at the literature behind the MVC pattern, describing the main functional parts of the pattern.

The MVC design pattern was first envisioned by Trygve Reenskaug in the 1970s at the Xerox Parc. According to him, “the essential purpose of MVC is to bridge the gap between the human user's mental model and the digital model that exists in the computer”.

Later on, in 1988, the MVC paradigm was described in detail by Krasner and Pope in their article “A cookbook for using the model-view controller user interface paradigm in Smalltalk-80”, published in the Journal of ObjectOriented Programming.

They stress out that there are enormous benefits to be had if one builds applications with modularity in mind. “Isolating functional units from each other as much as possible makes it easier for the application designer to understand and modify each particular unit without having to know everything about the other units.”

An application is divided into three main categories: the model of the main application domain, the presentation of data in that model and user interaction.The MVC pattern splits responsibilities into three main roles thus allowing for more efficient collaboration.These main roles are development, design and integration.

The development role is taken on by experienced programmers that are responsible for the logic of the application. They take care of data querying, validation, processing and more. The design role is for the developers that are responsible for the application look and feel. They display data that is fed from the developers working on the first role.

The integration role gathers developers with the responsibility to glue the work of the previous two roles.

The MVC design pattern is such a good fit for web application development because they combine several technologies usually split into a set of layers. Also, MVC specific behavior could be to send specific views to different types of user-agents.

“User interaction with an MVC application follows a natural cycle: the user takes an action, and in response the application changes its data model and delivers

an updated view to the user. And then the cycle repeats. This is a very convenient fit for web applications delivered as a series of HTTP requests and responses."

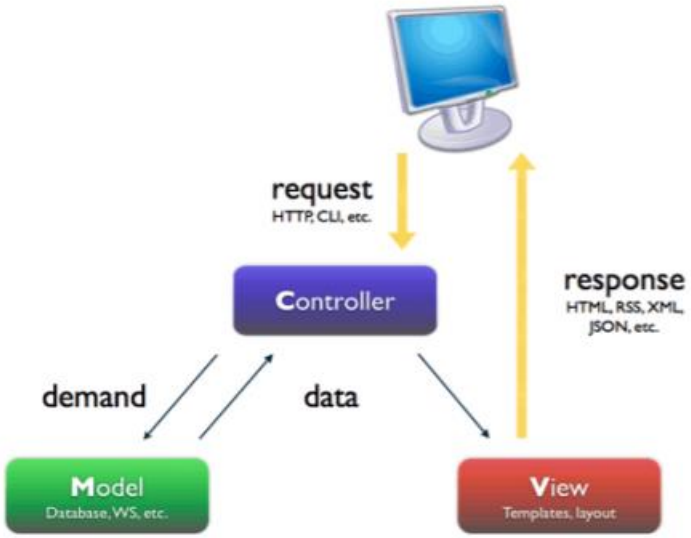


Figure 1 The MVC pattern

Source: http://stackoverflow.com/questions/5966905/which-mvc-diagram-is-correct-web-app

3.1. The Model

The Model is the part of the system that manages all tasks related to data: validation, session state and control, data source structure (database). The Model greatly reduces the complexity of the code the developer needs to write.

The Model layer is responsible with the business logic of an application. It will encapsulate methods to access data (databases, files, etc.) and will make a reusable class library available. Usually a Model is built with data abstraction in mind, validation and authentication.

Moreover, the Model is made up of classes that define the domain of interest. These objects that belong to the domain often times encapsulate data that is stored in databases, but also include code that is used to manipulate this data and enforce business rules.

As a conclusion, the Model mainly handles data access abstraction and validation. The Model holds methods for interaction with different data sources.

We believe in the Thin Model approach, which says that a model should be kept as simple as possible, only encompassing data processing that is strictly tied to the real life object that is modeled. The Thin Model is coupled with a Fat Controller,

which houses most of the data processing required by the application. This way, the models become highly reusable between applications and most of a developer's work is kept inside controllers.

The novelty our Model system brings to the MVC world is a system for versioning control, based on the idea of migrations, that doesn't only keep track of data structure but of data itself. The system uses xml files to store data between migrations, making the versioning process for databases an easily accomplishable thing.

3.2. The View

The View is responsible with graphical user interface management. This means all forms, buttons, graphic elements and all other HTML elements that are inside the application. Views can also be used to generate RSS content for aggregators or Flash presentations. By separating the design of the application from the logic of the application we greatly reduce the risk of errors showing up when the designer decides to alter the interface of that application by changing a logo or a table. In the same time, the developers' job is greatly reduced because he no longer needs to see HTML code elements, design elements and graphical elements.

The View layer is what can normally be called web design or templates. It controls the way data is displayed and how the user interacts with it. It also provides ways for data gathering from the users. The technologies that are mainly used in views are HTML, CSS and JavaScript.

As a general rule, a view should never contain elements that belong to application logic, in order to make it easier for the designer to work with it. This means logical blocks should be kept at a minimum.

Most web application frameworks today use some sort of template engine that takes advantage of generator elements in order to keep HTML code at a minimum and reduce the risk of typing mistakes. These generators are usually used to make complex web partials, like forms, tables, lists, menus and so on. The problem we identified in this case is that all implementations of this idea use behind the scenes or opaque generation of partials. This way, a front-end developer can only see the resulting code after it's been generated, having little to no way of modifying the template. Our system uses special HTML comments to insert and generate partials based on templates that are also HTML-only files. The pre-processing system uses

the comments to interpret special commands in order to insert data into the templates. This makes the whole process very transparent for the front-end developer, who can see the whole markup before rendering the view.

3.3. The Controller

The Controller is responsible for event handling. These events can be triggered by either a user interacting with the application or by a system process.

A controller accepts requests and prepares the data for a response. It is also responsible with establishing the format of that response. The Controller interacts with the Model in order to retrieve the needed data and generates the View. This process is also known as an action or a verb.When a request arrives at the server, the MVC framework dispatches it to a method in a controller based on the URL.

The Controller binds all application logic and combines the display in the View with the functionality in the Model. It is responsible with data retrieval from the View and with establishing the execution path for the application. The Controller will access the Model functionality and it will interpret the data received so that it can be displayed by the View. It is also responsible with error handling.

A Controller manages the relationship between a View and a Model. It responds to user requests, interacts with the Model and decides which View should be generated and displayed.As mentioned above, we embrace the Fat Controller approach, believing that all application-specific data processing should be handled at the controller level.

Our controller system is RESTful and supports JSON(P), text and XML data formats for requests and responses.

4. Database abstraction

In object oriented programming applications are built using objects. These objects reflect real life objects and they contain both data and behavior. The relational model that is widely used for data storage in applications uses tables to store data and data manipulation languages to interact with that data. Some database management systems have object oriented features, but they are not fully compatible. It is very clear that these two architectures are widely used and will be for a long time from now. Moreover, the relational model and object oriented programming are used together most of the time to build applications of every scale. But the way the two technologies combine is far from perfect.

The term used for this mismatch is “object-relational impedance mismatch”. The reason behind this mismatch is that the two technologies rely on different concepts. Object oriented programming relies on proved programming concepts, while the relational models follows mathematical principles.

Object-relational mapping systems were developed solely to address this issue.

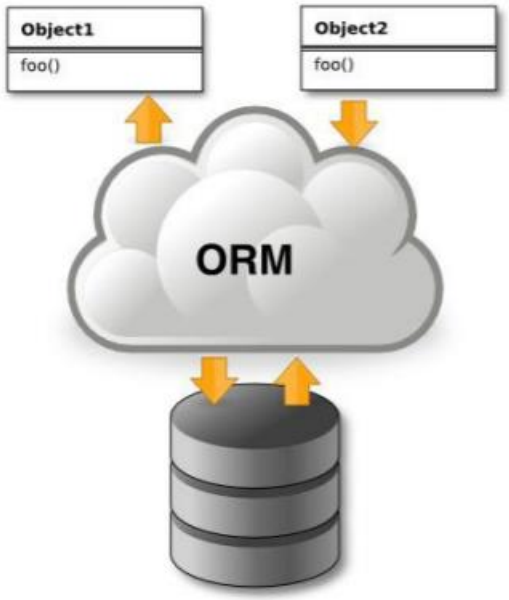


Figure 2 ORM System

Source:http://danhartshorn.com/2011/12/object-relational-mapping-wikipedia-the-free-encyclopedia

An object-relational mapping system (ORM) is defined as being a tool that provides a methodology and a mechanism for object oriented systems to store data in safe way and for a long period of time in a database, having transactional control over them, but being expressed, if needed, as objects inside the application.

An ORM system frees the developer from the worry of knowing the database structure or schema. Data access is done using a conceptual model that reflects the developers’ business objects.

In conclusion, we can say that an object-relational mapping system is a way to correlate data from a relational system with classes in a programming language, allowing the developer to work with well-known concepts from object oriented programming to manage data from that relational system.

An MVC platform must provide developers with a way to interact with a database management system to store and retrieve data. Most of the times, this is an integrated ORM system.

Because in the MVC architecture the Model is the layer that interacts with data, the ORM system is hidden behind it. Basically, each Model is connected to the ORM systems and it uses the system to interact with data. But developers may access the resources of the ORM system without the need to use a Model.

Our ORM system supports many-to-many relations between objects and has a setting for maximum depth level for object linkage.

Also, we provide an option to keep certain data sets in memory and skip database querying, reducing response times at a minimum.

As far as NoSQL data stores are concerned, we support an interface to access a number of such systems. This interface is part of the ORM system.

5. Security

Web application vulnerabilities are the main cause of attack on users from the business environment.

There are a lot of things developers need to be aware when building web applications.

Security risks can be divided into several categories:

•User input validation: buffer overflow, cross-site scripting, SQL injection, canonicalization;

•Authentication: network sniffing, brute-force attacks, dictionary attacks, cookie faking, identity theft;

•Authorization: access privilege violation, private data display, data altering, stalking attacks;

•Configuration management: unauthorized access to the management interface, unauthorized access to configuration zones, access to configuration data stored as text, lack of action logging;

•Sensible information: access to sensible data in the database, network sniffing, data altering;

•Session management: session theft, session altering, man in the middle attacks;

•Encryption: weak security keys, weak encryption;

•Parameter manipulation: query string, form field, cookie and HTTP header manipulation;

•Exception management: DOS attacks;

•Developers need to defend against all of these types of attacks and security needs to be handled at every layer in the MVC architecture.

•Our system proposal has built in CSRF attack prevention, a simple authentication schema, a complex role based security system and highly customizable access control lists for resources

6. Routing

The way the HTTP protocol handles URL writing is very similar to the way resource paths are written in the Unix environment.

Web servers bear this in mind and implicitly look for data in a hierarchical file system. This way of accessing resources is relatively easy and intuitive, but as web application complexity grows, the need for a better system arises. To solve this, we can configure the web server and the development framework in such a way that it will interpret results in a unique manner and access the resources.

These systems are called URL Mapping or URL Routing systems. The technique is also known as URL Beautifying and helps developers build ordered and nice URLs data both users and search engines can handle better.

Our system is built on the “controller/action/params” paradigm, but it also allows developers to configure their own static routes.

Access control is done at the routing level in order to reduce unneeded processing time.

7. Rapid prototyping

In the world of web application development a lot of application parts seem to be used multiple times. For example web forms are a tool that is used really often. So are tables and lists. But building such elements from scratch is rather difficult many times, because the involve writing a lot of complex markup.

Each system can benefit from a rapid prototyping module that not only generates these tools quickly, but also provides ways to validate data (for forms).

Such tools are really useful when combined with ORM systems to build CRUD systems from the ground up with very few lines of code.

As stated in the section on Views above, our system is based on a flexible and fully transparent template engine, that comes to benefit both back-end and front-end developers alike.

8. System proposal overview

Our platform proposal unifies all of the above aspects in one single package that is robust and easy to use and maintain. The structure above can be described as follows. The data source can be any type of database.

The ORM system is a set of classes built specifically for most well-known database management systems. One main class provides basic mapping, relations and management for a general type of table in the desired DBMS. Relations should be mapped as to allow fetching of related data in an object oriented way, without the need for complex queries written by the developer. Both eager and lazy loading of data should be considered.

Models are built as children of the main ORM class inheriting functionality from that main class. In addition, Models are enriched with unique capabilities that describe the real life object that should be modeled.

The Controller is a base class that handles Model querying, access security and route resolving. Controllers are built as children of the base Controller and inherit the functionality and are enriched with application logic corresponding with the desired actions.

The Rapid Prototyping system is as set of classes that allow developers to build complex HTML objects like forms and tables without the need to write HTML code. Forms should provide the possibility of data validation and tables that of sorting and nesting. The Rapid Prototyping system should provide DOM-like manipulation capabilities in order to access the data.

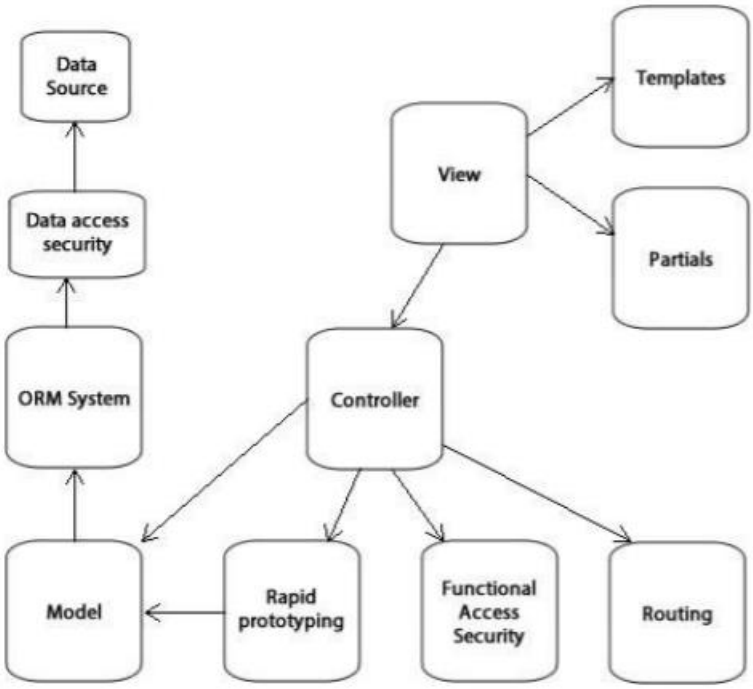


Figure 3 MVC structure proposal

The View is a base class containing all display logic. Views are built as children of this main class and should be tied to a specific action and controller. Views should be pre-rendered for performance reasons and should be capable of caching. The View system can depend of a Template system and a Partials system for code reuse.

9. Conclusion

Based on the proposal formulated above, we have built a solid framework that is capable of reducing web application development times drastically, allowing developers to focus on application specific tasks, rather than wasting time trying to implement well-known patterns and practices.

Web technologies have come a long way in little time and the market for web applications is ever growing. Our system helps developers improve their work speed and quality and provides a nice working framework and platform for both beginners and experienced users.

We have built a number of applications using components form the platform proposition and these applications showthat the system we are building is on the right track, while giving us valuable input on things that need to be improved.

10. Further research

The focus of our future research in this area is improving support for the various NoSQL systems that are available, because we think this area is of big future interest for the IT community. Also, we aim at improving the speed of the rendering engine by incorporating caching techniques for data and views.

Another important step is launching the project in the open source community in order to build more test case applications that generate valid results. These results will be used to further optimize the platform. Also, we aim to support packages built by the community in order to further extend the platform.

二、英文翻译：

面向 Web 应用快速开发的 MVC 模型设计

德拉戈什保罗·波普，亚当艾特

1、导言

自从万维网问世以来，Web 应用程序的开发已经取得了长足的进步。现在，很多技术和编程语言都被用来构建 Web 应用程序，但由于万维网的发展方式以及它的发展速度，这些技术并没有真正的时间来发展和应对这种速度。许多玩家视图想出不同的和奇异的技术来主要改善用户体验，并帮助开发人员更快更好地构建功能强大的 Web 应用程序。 其中一些技术在 Web 开发中起到了重要作用，但在过去几年中使用率大幅下降，比如 Java applet 和 Microsoft Silverlight。另一方面，许多技术已经从简单的玩具发展到今天 Web 生态系统中强大而重要的部分，比如 JavaScript、Flash 和 XML。

今天的 Web 环境使用 HTML 和 CSS 向用户呈现数据，交互是通过 JavaScript 实现的。这些技术被称为“前端”或“客户端”技术。 另一方面，“后端”或“服务器端”技术指的是数据存储和处理技术。

2、提出问题

前端和后端技术共同构建 Web 应用程序，但由于万维网的发展速度如此之快，而且开发人员只需使用相当多的技术来构建单个 Web 应用程序，

他们工作的结果往往很难维护和修复。开发人员将 HTML 代码与服务器端编程语言相结合来创建动态网页和应用程序，这导致代码高度纠缠和不可维护。

另一个问题是 Web 技术越来越多地用于构建各种复杂的应用程序。微软采用 Web 技术来鼓励开发人员为其最新的操作系统 Windows8 构建应用程序。此外，还有许多框架可以帮助 Web 开发人员编写移动设备应用程序，如 PhoneGap 和 AppCelerator Titanium。更重要的是，移动操作系统正在开发中，预计今年晚些时候将有使用 Web 的设备问世

完全针对开发人员 API（Mozilla Firefox OS）的技术。由于这些原因，Web 应用程序通常是由一组专门的开发人员创建的，每个开发人员都使用自己最喜爱

的技术，例如用于表示层的 HTML 和 CSS，用于客户端交互的 JavaScript，用于服务器端逻辑的 PHP（或 ASP、Java、Python、Pearl、Ruby 等），以及用于数据存储和管理的 MySQL（或 Oracle 数据库、Microsoft SQL Server 等）。

每个专家都需要以这样一种方式与他们的同事一起工作，即他们的代码片段适合于应用程序的总体设计。例如，客户端（数据表示）开发人员需要更改 HTML 和 CSS 代码，使其不会破坏驻留在同一文件中的服务器端开发人员的代码。另外，当数据库开发人员更改应用程序的架构时，服务器端开发人员可能需要更改大量代码才能使应用程序正常工作。

这里需要注意的重要一点是，在应用程序中迫切需要将表示与逻辑和数据存储分开。有一些应用程序设计范例和模式为这个问题提供了解决方案，但现在的焦点是 MVC 模式。

在进行这项研究时，我们使用了我们的经验，即使用各种系统和框架构建 Web 应用程序，我们视图找出这些系统的优点和缺点，同时对如何改进这些实践提供我们自己的看法。 研究的框架和系统包括:Symfony、CakePHP、CodeIgniter Zend Framework、Laravel、Fuel PHP、Ruby on Rails 和 ASP.NET MVC。

3、MVC 模式及文献综述

在这一节中，我们将回顾该领域的研究现状，并回顾 MVC 模式背后的文献，描述该模式的主要功能部分。

MVC 设计模式是 Trygve Reenskaug 在 20 世纪 70 年代在施乐 PARC 上首次提出的。他说:“MVC 的根本目的是在人类用户的心智模型和计算机中存在的数字模型之间架起一座桥

后来，在 1988 年，Krasner 和 Pope 在他们发表在面向对象编程杂志上的文章“在 Smalltalk-80 中使用模型-视图控制器用户界面范例的食谱”中详细描述了 MVC 范例。

他们强调，如果您在构建应用程序时考虑到模块化，那么将会有巨大的好处。 “尽可能多地将功能单元相互隔离，使应用程序设计人员更容易理解和修改每个特定单元，而不必了解其他单元的所有信息。

应用程序分为三大类:主应用程序域的模型、该模型中的数据表示和用户交互。MVC 模式将职责划分为三个主要角色，从而允许更有效的协作。这些主要作用是开发、设计和集成。

开发角色由有经验的程序员承担，程序员负责应用程序的逻辑。 他们负责数据查询、验证、处理等工作。 设计角色是为负责应用程序外观的开发人员设计的。 它们显示从处理第一个角色的开发人员馈送的数据。 集成角色将开发人员聚集在一起，负责粘合前两个角色的工作。

MVC 设计模式非常适合 Web 应用程序开发，因为它们结合了几种技术，通常分为一组层。另外，MVC 特定的行为可以是向不同类型的用户代理发送特定的视图。“用户与 MVC 应用程序的交互遵循一个自然循环:用户采取一个操作，作为响应，应用程序更改其数据模型并向用户传递一个更新的视图。然后循环重复。 这非常适合作为一系列 HTTP 请求和响应交付的 Web 应用程序。

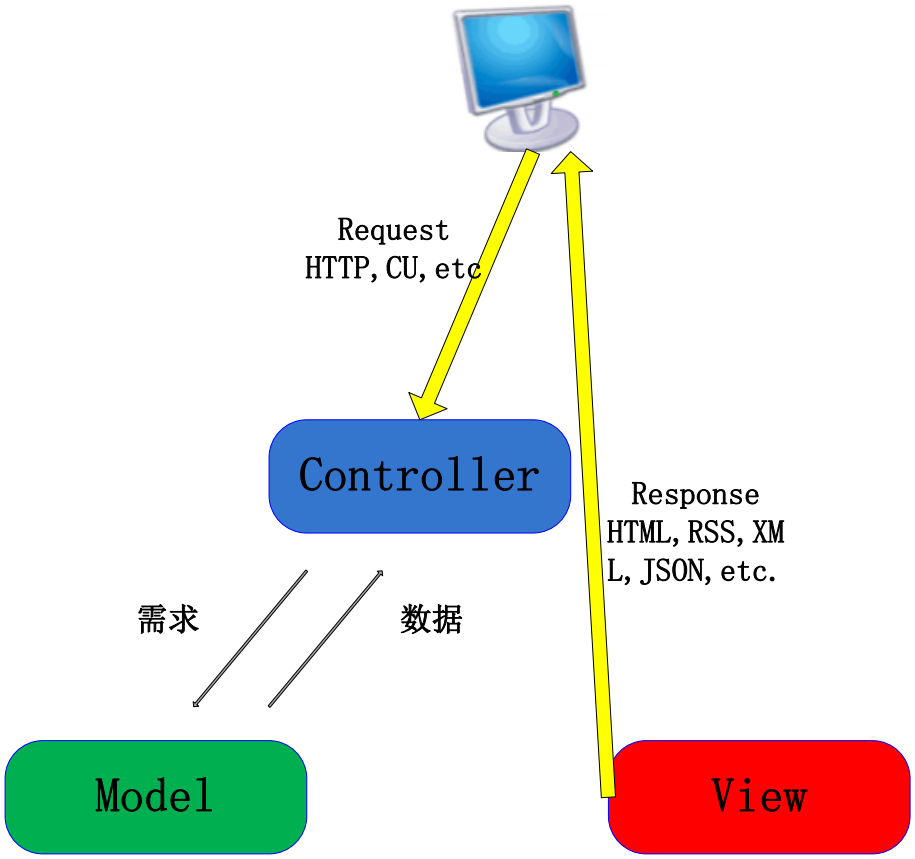


图 1 MVC 组成结构图

Source: http://stackoverflow.com/questions/5966905/which-mvc-diagram-is-correct-web-app

3.1、模型

模型是系统的一部分，负责管理与数据相关的所有任务:验证、会话状态和控制、数据源结构（数据库）。该模型大大降低了开发人员需要编写的代码的复杂性。

模型层负责应用程序的业务逻辑。它将封装访问数据（数据库、文件等）的方法，并提供可重用的类库。通常，在构建模型时会考虑到数据抽象、验证和身份验证。

此外，模型由定义感兴趣域的类组成。这些属于域的对象经常封装存储在数据库中的数据，但也包括用于操作这些数据和强制执行业务规则的代码。

总之，该模型主要处理数据访问的抽象和验证。该模型包含用于与不同数据源交互的方法。

我们相信瘦模型方法，即模型应该保持尽可能简单，只包括严格绑定到被建模的真实对象的数据处理。瘦模型与 FAT 控制器耦合，FAT 控制器包含应用程序所需的大部分数据处理。这样，模型就变成了

在应用程序和开发人员的大部分工作之间具有高度的可重用性。我们的模型系统给 MVC 世界带来的新颖性是一个基于迁移思想的版本控制系统，它不仅跟踪数据结构，而且跟踪数据本身。系统使用 XML 文件在迁移之间存储数据，这使得数据库的版本控制过程很容易完成。

3.2、景色

视图负责图形用户界面管理。 这意味着应用程序中的所有表单、按钮、图形元素和所有其他 HTML 元素。视图还可用于生成聚合器或 Flash 演示文稿的 RSS 内容。通过将应用程序的设计与应用程序的逻辑分离，我们大大降低了当设计人员决定通过更改徽标或表来更改应用程序的接口时出现错误的风险。同样地

由于不再需要查看 HTML 代码元素、设计元素和图形元素，开发人员的工作时间大大减少。

视图层通常可以称为 Web 设计或模板。它控制数据的显示方式以及用户与数据的交互方式。它还提供了从用户收集数据的方法。视图中主要使用的技术是 HTML、CSS 和 JavaScript。

通常，视图绝不应包含属于应用程序逻辑的元素，以便让设计人员更容易使用它。这意味着逻辑块应该保持在最小值。

目前大多数 Web 应用程序框架使用某种模板引擎，该引擎利用生成器元素将 HTML 代码保持在最小值，并减少键入错误的风险。这些生成器通常用于生成复杂的 Web 局部，如窗体、表、列表、菜单等。我们在本例中发现的问题是，此思想的所有实现都在幕后使用或不透明地生成分部。这样，前端开发人员只能看到生成的 COD

在生成模板之后，几乎没有修改模板的方法。我们的系统使用特殊的 HTML 注释来插入和生成基于模板的部分，模板也是 HTML 专用文件。预处理系统使用注释来解释特殊命令，以便将数据插入模板中。这就是全部进程对于前端开发人员来说非常透明，他们可以在呈现视图之前看到整个标记。

3.3、控制器

控制器负责事件处理。这些事件可以由与应用程序交互的用户触发，也可以由系统进程触发。

控制器接受请求并为响应准备数据。 它还负责确定这一答复的格式。 控制器与模型交互以检索所需的数据并生成视图。 这个过程也被称为动作或动词。当请求到达服务器时，MVC 框架根据 URL 将其分派给控制器中的方法。

控制器绑定所有应用程序逻辑，并将视图中的显示与模型中的功能相结合。它负责从视图中检索数据并为应用程序建立执行路径。 控制器将访问模型功能，并解释接收到的数据，以便视图可以显示这些数据。 它还负责错误处理。

控制器管理视图和模型之间的关系。 它响应用户请求，与模型交互，并决定应该生成和显示哪个视图。如上所述，我们支持 FAT 控制器方法，认为所有特定于应用程序的数据处理都应在控制器级别处理。

我们的控制器系统是 RESTful 的，支持 JSON(P)，文本和 XML 数据格式的请求和响应。

4、数据库抽象

在面向对象编程中，应用程序是使用对象构建的。这些对象反映现实生活中的对象，它们包含数据和行为。在应用程序中广泛用于数据存储的关系模型使用表存储数据，使用数据操作语言与数据交互。 有些数据库管理系统具有面向对

象的特性，但它们并不完全兼容。很明显，这两种体系结构被广泛使用，并且将在很长一段时间内。

现在就开始。此外，关系模型和面向对象编程在大多数情况下一起使用来构建各种规模的应用程序。但这两种技术的结合方式还远远不够完善。

用于此失配的术语是“对象-关系阻抗失配”。造成这种不匹配的原因是，这两种技术依赖于不同的概念。面向对象的程序设计依赖于经过证明的编程概念，而关系模型则遵循数学原理。

对象-关系映射系统的开发完全是为了解决这个问题。

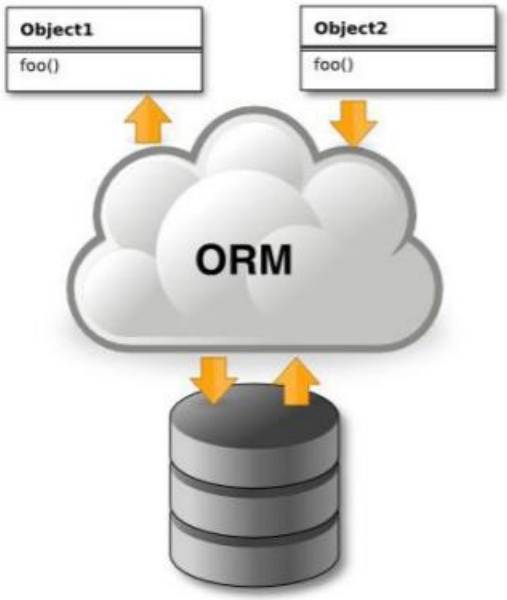


图 2 ORM 系统

Source:http://danhartshorn.com/2011/12/object-relational-mapping-wikipedia-the-free-encyclopedia

对象关系映射系统(ORM)被定义为一种工具，它为面向对象系统提供了一种方法和机制，以便在数据库中安全地和长时间地存储数据，对数据具有事务控制，但如果需要，可以表示为应用程序内部的对象。

ORM 系统将开发人员从了解数据库结构或模式的麻烦中解放出来。 数据访问是使用反映开发人员业务对象的概念模型完成的。

总之，我们可以说，对象关系映射系统是一种将关系系统中的数据与编程语言中的类相关联的方法，允许开发人员使用面向对象编程中的众所周知的概念来管理来自该关系系统的数据。

MVC 平台必须为开发人员提供一种与数据库管理系统交互以存储和检索数据的方法。 大多数情况下，这是一个集成的 ORM 系统。

因为在 MVC 体系结构中，模型是与数据交互的层，所以 ORM 系统隐藏在它的后面。基本上，每个模型都连接到 ORM 系统，并使用该系统与数据进行交互。 但是开发人员可以访问 ORM 系统的资源，而不需要使用模型。

我们的 ORM 系统支持对象之间的多对多关系，并为对象链接设置了最大深度级别。

此外，我们还提供了一个选项，用于将某些数据集保留在内存中并跳过数据库查询，从而最小化响应时间。

就 NoSQL 数据存储而言，我们支持一个接口来访问许多这样的系统。 此接口是 ORM 系统的一部分。

5、安全

Web 应用程序漏洞是从业务环境攻击用户的主要原因。在构建 Web 应用程序时，开发人员需要注意很多事情。

安全风险可分为几类:

•用户输入验证:缓冲区溢出、跨站点脚本、SQL 注入、规范化；

•身份验证:网络嗅探、暴力攻击、字典攻击、cookie 伪造、身份盗窃；

•授权:访问权限侵犯、私人数据显示、数据更改、跟踪攻击；

•配置管理:未经授权访问管理界面、未经授权访问配置区域、访问以文本形式存储的配置数据、缺少动作日志记录；

•敏感信息:访问数据库中的敏感数据、网络嗅探、数据更改；

•会话管理:会话盗窃、会话更改、中间人攻击；

•加密:弱安全密钥，弱加密；

•参数操作:查询字符串、表单字段、Cookie 和 HTTP 标头操作；

•异常管理:DoS 攻击；

•开发人员需要防范所有这些类型的攻击，并且需要在 MVC 体系结构的每一层处理安全性。

•我们的系统方案内置 CSRF 攻击预防功能，这是一种简单的身份验证 Schema，一个复杂的基于角色的安全系统和高度可定制的访问控制

6、路由

HTTP 协议处理 URL 写入的方式与 UNIX 环境中写入资源路径的方式非常相似。

Web 服务器记住了这一点，并隐式地在分层文件系统中查找数据。 这种访问资源的方式相对简单且直观，但是随着 Web 应用程序复杂性的增长，对更好系统的需求也随之产生。为了解决这个问题，我们可以配置 Web 服务器和开发框架，使其能够以独特的方式解释结果并访问资源。

这些系统称为 URL 映射或 URL 路由系统。 这项技术也被称为 URL 美化，帮助开发人员构建有序和良好的 URL 数据，用户和搜索引擎都能更好地处理这些数据。

我们的系统是基于“Controller/Action/Params”模式构建的，但它也允许开发人员配置自己的静态路由。

访问控制在路由层完成，以减少不必要的处理时间。

7、快速原型制作

在 Web 应用程序开发的世界中，许多应用程序部件似乎被多次使用。 例如，Web 窗体是一种非常常用的工具。 表格和列表也是。 但是从零开始构建这样的元素是相当困难的很多次，因为这涉及到编写很多复杂的标记。

每个系统都可以受益于快速原型模块，该模块不仅可以快速生成这些工具，而且还提供了验证数据（表单）的方法。

当这些工具与 ORM 系统结合在一起，以很少的代码行从头构建 CRUD 系统时，它们确实非常有用。

正如上面关于视图的部分所述，我们的系统基于一个灵活且完全透明的模板引擎，这对后端和前端开发人员都有好处。

8、系统提案概述

我们的平台提案将上述所有方面统一在一个健壮且易于使用和维护的包中。上述结构可以描述如下。数据源可以是任何类型的数据库。

ORM 系统是一组专门为大多数已知的数据库管理系统构建的类。一个主要类为所需 DBMS 中的一般类型的表提供基本的映射、关系和管理。关系应该映射为允许以面向对象的方式获取相关数据，而不需要开发人员编写复杂的查询。应考虑数据的急切加载和延迟加载。

模型是作为从主 ORM 类继承功能的主 ORM 类的子类构建的。此外，模型还具有描述应该建模的真实对象的独特功能。

控制器是一个基类，用于处理模型查询、访问安全性和路由解析。控制器被构建为基本控制器的子级，并继承了功能，并使用与所需操作相对应的应用程序逻辑进行了丰富。

快速原型系统是一组类，允许开发人员构建复杂的 HTML 对象，如表单和表格，而不需要编写 HTML 代码。 表单应提供数据验证的可能性，表单应提供排序和嵌套的可能性。快速原型系统应该提供类似 DOM 的操作能力，以便访问数据。

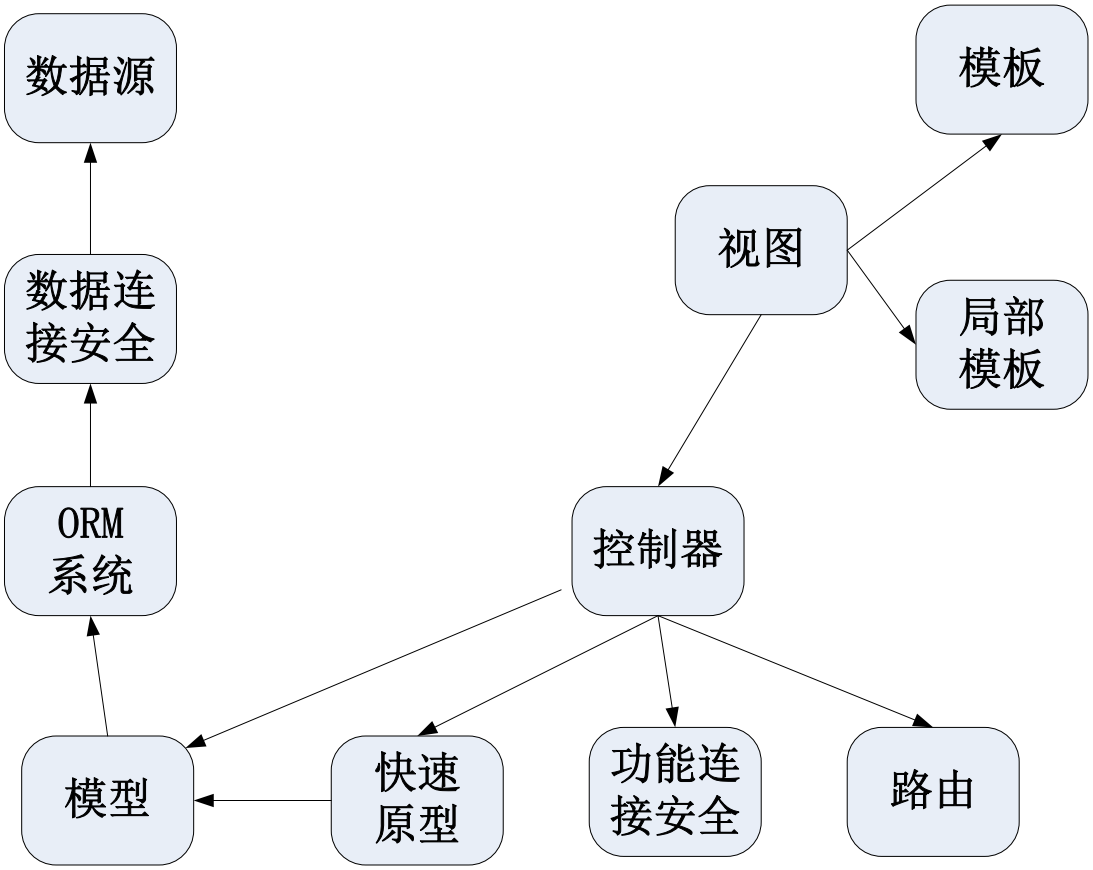


图 3 MVC 组织结构图

视图是包含所有显示逻辑的基类。视图是作为这个主类的子类构建的，应该绑定到一个特定的操作和控制器。视图应该出于性能原因进行预呈现，并且应该能够缓存。视图系统可以依赖于模板系统和用于代码重用的部分系统。

9、结论

基于上面提出的建议，我们构建了一个坚实的框架，它能够大幅减少 Web 应用程序的开发时间，允许开发人员专注于应用程序特定的任务，而不是浪费时间来实现众所周知的模式和实践。Web 技术在很短的时间内取得了长足的进步，Web 应用程序的市场也在不断增长。 我们的系统帮助开发人员提高了工作速度和质量，为初学者和有经验的用户提供了一个良好的工作框架和平台。

我们已经使用平台中的组件构建了许多应用程序，这些应用程序表明我们正在构建的系统是正确的，同时在需要改进的方面给我们提供了有价值的投入。

10、进一步研究

我们未来在这一领域的研究重点是改进对各种可用的 NoSQL 系统的支持，因为我们认为这一领域对 IT 社区具有很大的未来意义。此外，我们的目标是通过结合数据和视图的缓存技术来提高渲染引擎的速度。 另一个重要步骤是在开源社区中启动该项目，以便构建更多能够产生有效结果的测试用例应用程序。这些结果将用于进一步优化平台。此外，我们的目标是支持社区构建的软件包，以便进一步扩展平台。