

存档编号_____

华北水利水电大学

North China University of Water Resources and Electric Power

毕 业 设 计

题目 基于 PHP 的毕业论文选题系统

学 院 信息工程

专 业 网络工程

姓 名 王文帅

学 号 201316602

指导教师 郑作勇

完成时间 2016.05

教务处制

独立完成与诚信声明

本人郑重声明：所提交的毕业设计（论文）是本人在指导教师的指导下，独立工作所取得的成果并撰写完成的，郑重确认没有剽窃、抄袭等违反学术道德、学术规范的侵权行为。文中除已经标注引用的内容外，不包含其他人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确的说明并表示了谢意。本人完全意识到本声明的法律后果由本人承担。

毕业设计（论文）作者签名：

指导导师签名：

签字日期：

签字日期：

毕业设计（论文）版权使用授权书

本人完全了解华北水利水电大学有关保管、使用毕业设计（论文）的规定。特授权华北水利水电大学可以将毕业设计（论文）的全部或部分内容公开和编入有关数据库提供检索，并采用影印、缩印或扫描等复制手段复制、保存、汇编以供查阅和借阅。同意学校向国家有关部门或机构送交毕业设计（论文）原件或复印件和电子文档（涉密的成果在解密后应遵守此规定）。

毕业设计（论文）作者签名：

导师签名：

签字日期：

签字日期：

目 录

摘 要.....	I
Abstract.....	II
第 1 章 绪论.....	1
1.1 研究背景.....	1
1.2 目的与意义.....	1
1.3 国内外选课系统现状.....	2
1.3.1 国外高校选课系统的研究现状	2
1.4 系统主要内容及功能.....	3
1.5 论文组织结构.....	4
第 2 章 系统实现的相关技术.....	5
2.1 .PHP 相关技术.....	5
2.1.1 PHP 语言的开发特征介绍.....	5
2.1.2 MVC 设计模式.....	5
2.1.3 ThinkPhp 框架简单介绍.....	6
2.2 Mysql 数据库编程.....	6
2.2.1 Mysql 数据库的常用字段介绍.....	7
2.2.2 Mysql 数据库的基本操作.....	7
2.2.3 Mysql 数据库多表查询操作.....	8
2.3.1 PHP 开发工具 Sublime.....	8
2.3.2 Mysql 数据库管理工具-Navicat for MySQL.....	9
2.3.3 系统的运行环境 phpstudy2016.....	9
2.3.4 远程服务器上传工具 flashFXP.....	10
第 3 章 系统需求分析与数据库设计.....	11
3.1 系统需求分析.....	11
3.1.1 系统模块的划分.....	12
3.1.2 系统功能详情.....	13
3.2 数据库的设计.....	13

3.2.1 系统的数据需求及数据关系.....	14
3.2.2 设计数据表结构.....	15
第 4 章 系统总体设计及前后台应用的实现.....	19
4.1 系统的总体设计.....	19
4.2 系统前台应用的实现.....	20
4.2.1 游客身份模块的实现.....	20
4.2.2 学生身份模块的实现.....	22
4.2.3 教师身份模块的实现.....	26
4.3 系统后台应用的实现.....	29
第 5 章 系统的调试与测试.....	30
5.1 系统的调试意义及技巧.....	30
5.2 系统的测试.....	31
5.3 系统功能预览.....	31
5.3.1 游客模块的功能展示.....	31
5.3.2 学生模块的功能展示.....	32
5.3.3 教师模块的功能展示.....	35
5.3.4 后台管理员模块的功能展示.....	37
第 6 章 软件维护.....	38
第 7 章 总结.....	39
参考文献.....	40
致 谢.....	41
附 录.....	42
附录 I.....	42
外文原文.....	42
中文译文.....	54
附录 II 毕业设计任务书.....	63
附录 III 开题报告.....	66

基于 php 的毕业论文选题系统

摘 要

每一年的毕业生论文进行论文选题的时候都会感觉很麻烦，主要原因有以下几部分：首先大多数的毕业班同学不在学校里面，这样和老师的交流上是很少的，其次就是大部分的指导教师在进行毕设指导的时候还有其他的班级的课程，没有充足的时间对学生在选课的过程进行全程指导。这样造成的后果是每一年的毕业论文选题总是一遍又一遍，一茬又一茬，大大浪费了学校资源并且延长了选题的时间，效率低是一方面，这样混乱选课很容易出现某两个同学所选的课题是一样的。所以，本次围绕这样的校园问题进行开发，其主旨在于提高选题的效率以及选题的准确度，保证每一位同学都可以进行顺利的选上自己所希望的课题，并且实现教师与学生可以双向选择，双方及时反馈，从而缩短毕业论文选题的周期，让更多的时间利用在毕业论文后期的设计制作中。

此次开发的论文选题系统能够方便学生在选题的时候及时进行课题查看以及在线选题，并且能够实时跟踪课题的动态信息，对与教师而言同样是一种便利，教师只需要将自己的题目上传发布上去，并且能够查看到本课程当前的选课情况，同时还可以对同学们的选课申请予以同意或者拒绝，这样能够有效得让自己的课程题目被充分的选择，这样能够从本质上提高选课的效率和资源的利用率。本系统主要使用 PHP 语言进行开发，其开发速度快节省成本的优势使他作为此次开发的语言，数据库的选择上选择了 MySQL 数据库，同样是一款免费开源的产品。设计模式采用了现在普遍流行的 MVC 设计模式，这样容易进行后期的维护以及整理，并且使用的 PHP 国内框架 Thinkphp 为项目 MVC 模式进行整体的搭建，远程服务器选取阿里云服务器，使用 Centos 操作系统作为服务器的运行系统，并且集成安装 PHP 的运行环境。以上是基础工作内容完成之后根据毕设任务以及需求分析，进行系统的层次构建以及数据库中数据表的设计，并进行相关的代码实现操作。最后系统的整体测试以及调试，加入大量的模拟数据并最终上传至服务器，找出系统中存在的不足以及逻辑思路上的缺陷。

关键字：选课系统；数据库；服务器；设计模式；

Selected topic system of graduation thesis based on PHP

Abstract

When every year graduation thesis topic is a very complicated problem, the main reason has the following parts: first and most of the students in this school, and teacher exchange listed a few, followed by most of the teachers in the thesis when there are other classes of courses no, sufficient time for the students in the course of the whole process guidance. As a result each year of the graduation thesis is always again and again, the crop of another crop, changed and changed, a huge waste of resources and time delay selection of students, low efficiency is not to say that it is easy to cause confusion of two students course topics, which is very awkward. The. So, this time around campus such problems in development, the main purpose is to improve the efficiency and accuracy of the elective course, ensure that every student can graduate tutor to the two-way choice of subject curriculum, let the students have their own initiative and the right, shorten the period of graduation thesis, let more time by graduation the late completion of the.

This thesis selection system is developed to facilitate students in elective courses in time to check and choose online, and can real-time dynamic information tracking course, but also as a convenience for teachers, teachers only need to own the title upload up, and to check the current course of elective courses, at the same time the students of the course to apply to be agreed or refused, so it can effectively make their own curriculum topics are full of. In this way, the efficiency of selecting courses and the utilization of resources can be improved in essence. This system mainly uses the PHP language development, the development speed of cost saving advantage as the development of language data on the choice of the use of the MySQL database, is also free and open source products, this system can lay a solid database in a long run. Design patterns using MVC design patterns popular, so easy to maintenance and consolidation, and use the Thinkphp PHP framework for domestic MVC project through the whole building, the remote server selects Ali cloud server, using the CentOS operating system as the operating system of the server, and the running environment of PHP integrated installation. The above is the basic work, after the completion of the content, according to the completion of tasks and needs analysis, the hierarchical construction of the system, as well as functions and data tables in the design of MSYQL, and related practical operations, realize the

function. Finally, the overall system testing and debugging, adding analog data and upload servers, to identify deficiencies in the system, as well as logical thinking defects.

Keywords: course selection system; database; server; design pattern;

第 1 章 绪论

1.1 研究背景

目前几乎所有的高等院校都有一个选修课的选题系统，但是大部分的高校却没有一套完整的毕业论文选题系统，原因有两个方面：首先选修课系统使用人数比较多，使用的频率比较高，选题简单，并且模式固定，而对于毕业论文的选题系统来说，每一个学年只会使用一次，并且选题的随机性太大，高校不会愿意出资金开发一套这样的论文的选题系统。所以就目前大多数的院校选题来看，很多都是人工收集整理数据，使用纸质操作统计进行选课的，很少采用互联网的方式进行论文选题的，这样带来的后果看到的不是科技在进步，而是重新退步到传统的手动操作了，造成了资源的大量浪费，人力和物力的极度损耗，最后所得到的结果却很普通，完全是事倍功半。针对该类情况开发一套关于毕业生进行选课的系统实在很有必要，节约当下的各种资源，保证数据的真实可靠性以及信息的及时反馈性。考虑到容错性以及扩展性，实现资源信息的共享，将学生、教师、课程等进行分析设计，同步工作和提高效率。

1.2 目的与意义

计算机的使用本身在于方便人类的生活，程序的的意义同样也是源于大众而服务于大众的，使用该选题系统的最大的意义让计算机之外的其他资源能节省下来，增加计算机的利用率，真正的实现数据的共享化，快速化以及可靠化，加快各种任务完成的节奏。本系统旨在解决选题难，等题难，修改难等论文选课问题，方便快捷的选课信息，准确可靠的统计结果以及及时准确的选课反馈，从毕业生的各个层面上来讲此系统对于毕业生都是一种便利。对于教师来说无疑也是优势，节省教师的私有时间，可以抽出更多的时间进行论文指导，而不是停留在选课上面，最后对于整个高校来说无疑是解决了统计这一部分的难题，抛弃了传统的纸质化操作，保证信息透明，公布及时。

最后要说明的是在 21 世纪的大数据时代，数据对于我们太重要了，大数据不是需要动手去整理的，数据是需要计算机通过系统进行不断的整理加工，汇总的。我们使用计算机的初衷在于解决生活中的实际问题，此系统就是因为要解决论文选题中的问题才应运而生的，哪里有需要哪里就会有计算机以及源代码的身影。

1.3 国内外选课系统现状

网站的发展从最初的静态 html 页面展示到现在的数据库搭伴形成动态交互系统，功能上从单一的页面展示到后来的数据的处理，以及到现在的大数据整理和应用。发展过程中出现了各种各样的开发语言以及开发技术，不过我们从 PHP 的发展史来看，PHP 和 Mysql 数据库的完美结合在于更快更便捷的进行动态交互网站的开发，国内外已经有很多院系采用 PHP 进行校园选课系统的开发和定制，愈来愈多的高校认识到了选课系统对于学校的重大意义。

1.3.1 国外高校选课系统的研究现状

首先需要阐明的是，在我们的国家的高等教育中，在很长时间是没有选修这个概念的，我们接受的教育基本上都是教师们一年年的授课经验和教育局针对社会发展而制定的，通常情况下是不允许学生自由发挥的。但是在国外就不太一样，比如加拿大和美国的大学，他们的大学一选课为主，主课为辅，选修的课程占据他们课程的很大一个比重，每一个学期都会给学生一个选择课程的机会，而不是像我们这样把课程安排的满满的，所以，可以及看到国外的大学很早就有这一个选课的需求，一旦存在这种选课需求，势必会带来一股选课系统的开发热潮。

综上所述，对于国外的大学而言他们的选修课是特别的多同样也是特别重要的，这样的话每年会有很多次进行选课的机会，那么如果没有一套可靠的选课系统进行数据整理和维护，这样一个选课的过程该如何实现呢？这样对来看这个选修课进行选课是不是在中学时期就已经成了每一所学校需要解决的问题呢？所以国外的选课系统发展阶段是很长的，到目前为止已经拥有了一套完备的选课系统。国外高校的选课发展来看的话，每一所高校拥有自己的独立的程序设计师以及规模相当大的技术团队，况且还有政府对于整个教育产业的大量投入，他们的选课系统不断的进行完善与更新，涌现出了像哈佛和麻省理工曾经要价八万的选课系统。不要惊讶这个数字，因为买的技术，独一无二。

1.3.2 国内选课系统的研究现状

大致来看，目前高效的教学任务存在大量的必修课程，很少有选修课程，大学的几个学年中你也很少会有选修的课程存在的。由于使用的频率特别低，造成了目前的整个选课系统的相当不完善，一人开发，百校使用，选课系统没有独特的高校自己的元素和特色，定制开发性不够强，界面比较统一，系统更新率相当低。诸多不足在此也不一一

进行阐述了。相信随着我国教育的发展和体制的完善化，我国的高校可以分配更多的选修课程安排给学生，这样势必会给选课系统的发展带来契机，而不是现在的依赖性选课系统。希望在未来，有效的选课系统可以成为各大院校的必备管理工具用来进行数据的搜集和整理。

虽然国外的高校选课管理系统已经特别有规模且有特色，但毕竟我们的高校教学和国外的有着明显的不同，由于教学管理手段和方法的差异，我们还不能完全复制他们的选课系统，只能是在借鉴的基础上另外进行开发出适合我国选课教育的计算机系统。

1.4 系统主要内容及功能

本系统主要是进行毕业论文选题的统计和整理，方便学生在线进行选课以及实现教师在线进行发布课程的目的。保证后台管理员能够进行有效准确的统计和分析论文题目的选择结果，高效的完成毕业生的毕设选课工作。

本次是基于 php 的开发的一套毕业论文的选课系统，分为三个角色进行使用：

1) 毕业班学生进行注册登录，注册时候存在前端以及后端的数据验证操作，防止信息填写不准确，另外采用了第三方的登录操作无需注册即可实现登录操作并且常规登录也存在数据前端和后端的验证操作，个人中心会首先需要完成资料（基本信息，头像设置，密码重置等操作）补充并根据填写资料自动匹配所在班级的各种信息，倘若没有补充资料则不可进行额外操作，核心部分是毕业班学生可以在线直接选课，查看选课结果，撤销选课，选课限制，查看班级选课情况，并且查看其他班级及院系的选课安排及结果；

2) 毕业班教师进行注册登录，注册时候存在前端以及后端的数据验证操作，防止信息填写不准确，这里没有采用第三方的登录操作并且常规登录存在数据前端和后端的验证操作，个人中心同样需要首先完成资料补充并根据填写资料自动匹配所在班级的各种信息（基本信息，头像设置，密码重置等操作），倘若没有完善资料则不可进行额外操作，核心部分是指导教师可以在线发布选课，查看选课进度，结束选课，选课结果查看，通知消息，拒绝选课，接受选课，重置选课以及查看毕业班级选课情况；

3) 后台管理员没有进行角色管理，后期可以陆续进行补充，超级管理员的角色主要是进行添加院系和专业的操作，这样教师和学生可以在进行基本资料完善的时候获取院系和专业信息，从而进行这两个的选择，同样管理员具备查看所有教师的课程情况

以及所有学生的选课情况的权限，但是不可以随意修改选课的正选结果，方便统计与管理学生的选课情况。

4) 技术难点：毕业设计的实践过程中，在角色管理以及登陆权限管理上可能会是一个比较大的难题，因为系统存在三种不同的角色员会对应不同的逻辑处理，这个可能比较复杂一点，另外的又存在游客和非游客的权限，这样在控制器需要把控一下不同的身份会有不同的权限。

1.5 论文组织结构

本篇论文的结构和安排如下：

第 1 章：介绍本论文的设计背景以及设计的必要性，设计的目的和任务，理清思路；

第 2 章：介绍系统实现本系统所采用的相关核心技术以及相关的开发工具介绍；

第 3 章：介绍本系统的需求分析以及数据库的设计，对该系统有一个大致的脉络把控，将系统的需求分析和代码实现做一个综合的考虑，全方位的评估系统的难易度数以及实现的可能性；

第 4 章：介绍系统总体设计架构，前台教师和学生角色的具体实现以及后台超级管理员角的具体实现；

第 5 章：系统的测试和调试工作，对系统进行大量模拟数据的添加，分析系统的测试结果寻找程序中的 BUG，对于程序中的 BUG 进行查找并修复，并展示系统的测试结果；

第 6 章：对软件的后期维护性和可扩展做一个整理，完成开发的技术文档；

第 7 章：总结经验反思操作过程中的不足之处，未来的发展提出深度的展望。

第 2 章 系统实现的相关技术

由于本系统主要是采用 PHP 作为开发的主要编程语言，所以这里我们重点介绍一下 PHP 相关的技术要领，数据库方面使用了 Mysql 作为系统的首选数据库，所以数据库内容侧重点是 Mysql 的数据表设计以及 Mysql 相关的 SQL 语句的使用。编辑器上采用轻量级 Sublime IDE 编辑器并进行简单的指令操作介绍，服务器上的对阿里云服务器做一个简单的展示，同时最后介绍采用的 MVC 设计模式以及本次开发中所用的典型 MVC 模式的 Thinkphp 框架的核心部分。至于前端的 JS 以及 Bootstrap 相关的内容会放在附录中进行详细的解释。这里只是抽取其中的核心要领进行简单的介绍。

2.1 . PHP 相关技术

2.1.1 PHP 语言的开发特征介绍

(1) 作为一种服务器语言，PHP 全称是超文本预处理器，是一门除了 .Net、Jsp 等等之外的一种服务器语言，在 Web 开发领域中运用的特别多；

(2) PHP 的运行环境移植性特别强，能够在各大操作系统中运行起来，对环境的要求不高，不想 JAVA 那样还需要操作系统设置运行环境，相对 PHP 来说，JAVA 的那一套环境太过于臃肿，PHP 的解析器仅仅三十多兆，环境部署特别简单，自身的扩展比较完善，这也注定了 PHP 能够占据 JAVA 的 web 市场；

(3) PHP 可以和多种数据库相联系，支持的数据库种类特别多，不管是关系型数据库还是非关系型数据库都有良好的支持，目前典型的的就是 PHP 和 MySQL 的结合在网站领域中占的比重比较大；

(4) PHP 属于开源免费的软件，存在大量的开放源代码，并且它的语法特别接近 C 语言的编程；开发速度比较快，大大的缩短了 Web 开发的开发周期。PHP5 中又出现了面向对象编程，顺利的将 PHP 推向了 Web 开发的王者荣耀。

2.1.2 MVC 设计模式

设计模式的概念，设计模式 (Design pattern) 简单来说就是之前的程序员进行不断的项目积累外加长期的代码规范，摸索出来的一适合程序开发的固有思路，是一种程

序开发的典型范畴。使用好的设计模式进行开发项目，可以是项目开发的速度更快，开发出来的程序更容易让知晓此模式的其他程序员尽快看懂所写的程序。

MVC 设计模式是目前比较流行的一种 Web 开发的设计模式，基本原理就是将程序处理代码和页面显示代码进行分离，让程序编程区域化的操作，这种模式给程序的修改和扩展都带了便利。这里大致介绍一下其中的三个部分：

(1) Model（模型）：处理应用程序数据逻辑的部分。其功能主要是通常模型对象负责在数据库中存取数据。是 web 开发中的核心部分，所有的业务逻辑必须由他完成；

(2) View（视图）：应用程序中处理数据显示的部分。其功能主要是通常是用来显示数据和页面的，通常是由纯粹的 html 和 js 代码组成的；

(3) Controller（控制器）：应用程序中处理用户交互的部分。其功能控制器负责从视图获取所传送的数据并且获取从模型中返回的数据并发送给视图，是一个核心的交互部分。

2.1.3 ThinkPhp 框架简单介绍

ThinkPHP 是国内技术团队开发的一款适合国人进行 Web 开发的一套框架产品，采用当前的 MVC 设计模式以及 PHP5 新增的面向对象而研发的轻型 PHP 开发框架，框架提供了一些的基本数据库操作模型，保证了整个程序不存在 Model 的情况下也能进行各种数据库的操作，列出该框架文件作为开发时候基本的目录结构如下所示：

```
└--index.php    主 入口文件
└--README.md    README 文件
└--App          应用目录
└--Public       资源文件目录
└--ThinkPHP     框架目录
```

2.2 Mysql 数据库编程

数据库是用来存放系统开发中所使用的数据，保证数据能够有效长期的保存在里面，可以随时随地地共享数据并可以进行查询修改删除增添登操作的一种数据管理系统，可以理解作为一种高级数组和具备识别能力的文件管理系统。

2.2.1 Mysql 数据库的常用字段介绍

a.数值型的

int float tinyint

int 的长度（无符号的时候是 $2^{32}-1$ unsigned）如果存储的数据是大于 $2^{32}-1$ 的数值的情况下，应该用字符串进行存储数据，字符串进行存储数据时候不会丢失,int 的时候不需要进行长度设置，只需要进行 1 和 0 进行标记，标记进行有符号和无符号的。

b.字符串类型

varchar(40) 括号中的是字符串的长度

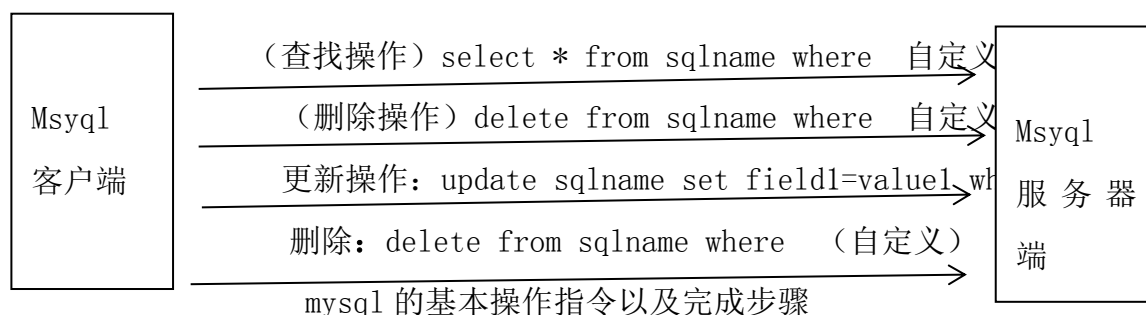
char(255) 长度小于 255

Text() 用于存储长度特别长的字符串

char 和 varchar 的区别 char 是固定长度 varchar 是可变的长度空间

2.2.2 Mysql 数据库的基本操作

mysql 作为众多数据库中的一个之所以选取它的重要原因是它作为一种免费产品面向个人用户，这无疑也是它作为数据库市场上最大赢家的重要原因，数据库分为客户端和服务端两层，我们的网站就是充当客户端的角色，进而进行数据的增删改查，但是数据是在服务器端进行保存的，客户端向服务器发出命令，服务器端进行一系列的数据库操作，是典型 C/S 类型的管理软件。虽然 Mysql 作为一款免费数据库被广泛的使用，即使平时我们开发的项目数据过百万的时候，它也可以完成，我们看一下 mysql 的基本用法，如图



2.2.3 Mysql 数据库多表查询操作

核心要点：属性要跟着主题走，不要把属性拴在一个表上；用户名可以拆拆看，但是不能分的太多了，尽量在一个主体上进行拆的。一个主体可有多个属性，不同的主体之间会有一定的关联联系；主体与主体之间的关系一个就差不多了。如下查询：

```
select a.name,b.username from a,b where a.class_id=b.id;
```

常规的设计规律：

|---有关系和没有关系的数据表；

|---有关系的是进行合并还是不合并；

|----原则是：一个表是一个主体，凡是一对一的关系就可以进行合并。

一对多：两个主体之间又包含的关系；

例如：一个会员有很多的评论，一个班级有很多学生，在比较大的范围中增加一个；

一对一：两个主体之间相互=对应的关系，两个一一相互对应，一个会员有一个积分。

2.3 系统开发中所用的工具及环境

2.3.1 PHP 开发工具 Sublime

Sublime 是一套超轻量级的程序开发的 IDE 开发工具，软件仅仅几十兆，运行速度快，插件库比较强大，和其他同一级别的 IDE 相比速度快和性能高绝对没有可比性，这就好比拿着 WPS 和 Microsoft word 相比较一样，结合了众多优势，考虑到该编辑器具备代码提示，错误报警，运行速度快，git 插件，sftp 远程同步插件，使用该编辑器编写核心的 PHP 代码实在没有问题，如下图所示为软件截图。

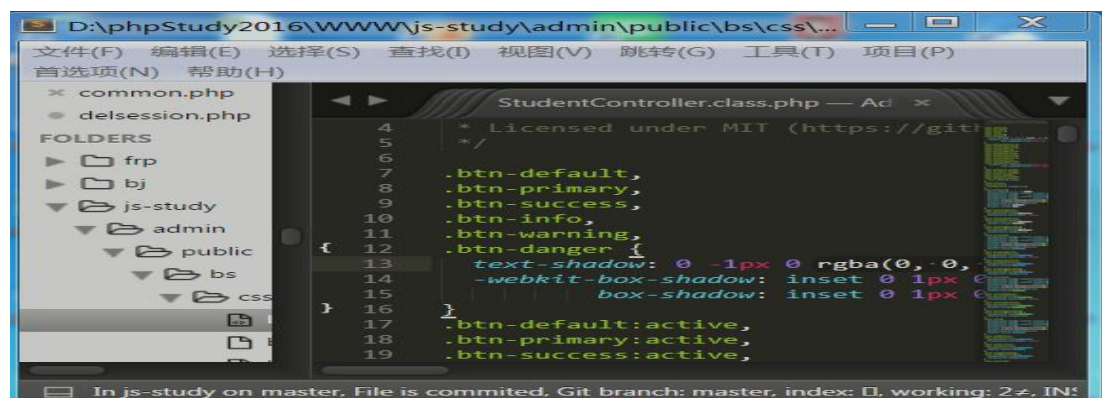


图 2.3.1 为 Sublime 软件图

2.3.2 Mysql 数据库管理工具-Navicat for MySQL

可视化操作绝对是我们选择该软件的基本原因，毕竟命令行窗口操作数据的时候是很难把控住的，另外该数据库管理软件的最大优势在于可以在软件中建立数据模型，并且通过软件进行数据模型转换，将数据模型生成我们所需要的数据表，可以清晰的看见表与表之间的关系图，不用查看字段信息就可以把控整个数据库的关系是如何的。软件如下图所示：

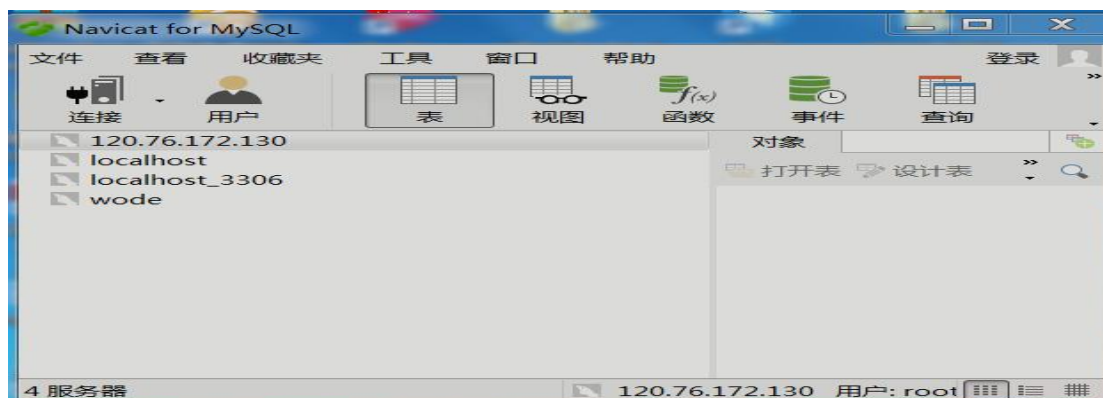


图 2.3.2 Navicat 软件图

2.3.3 系统的运行环境 phpstudy2016

前面已经谈到 PHP 的运行环境很简单，没有想 JAVA 那样臃肿，开发 PHP 项目，可以单独搭建运行环境，但是需要 Apache 和 PHP 配置相当熟悉才可以，不过 PHP 对于开发者来说有一系列的集成开发环境比如 Xamp、Wamp 等等，其中最好的搭配环境是 LAMP 和 LNMP，但是由于我们是在 Windows 平台下做开发，所以这种环境是满足不了的，其中 PHPstudy2016 提供了一个拥有多种搭配环境的集成包，可以随时切换，可以满足对同一个网站使用的不同服务器环境的测试。如下图：



图 2.3.3 phpstudy2016 软件图

2.3.4 远程服务器上传工具 flashFXP

当整个项目完成之后，需要将本地的项目上传到服务器，就是利用这个软件进行本地--->远程的上传工作，支持在线修改，将程序上传至远程的 Linux 环境中进行运行，查看其中和 Windows 平台中的差异性。

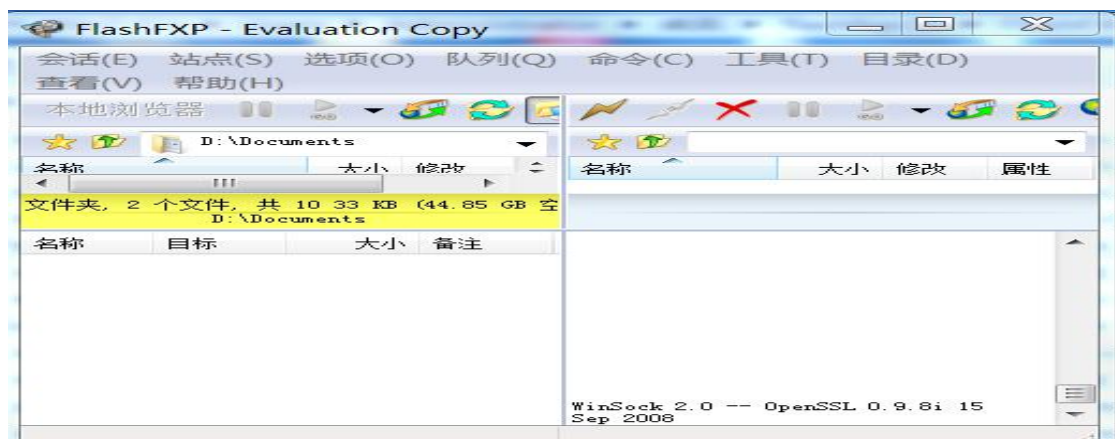


图 2.3.4 FlashFxp 软件图

第3章 系统需求分析与数据库设计

在系统进行开发的时候，先对需求分析做一个全面的把握，这样在后期的代码实现中就具备重点和非重点，不至于在写程序中不断的变换思路，这样只能是事倍功半，最好的是要进行需求分析，设计思路，其实代码的实现是一个很纯粹的工作，只要懂点编程语法的人你给他一个任务他肯定能给你以代码的形式表现出来。但是倘若让他把这个设计思路弄出来，那实在是太难了，这就好比编程架构师和普通程序员的区别，程序的代码操作往往是最容易实现的，关键是前期的这一个设计思路，所以必须将系统需求弄清楚，充分了解此次系统要是实现什么功能，要如何实现，要如何显示呢？这样一系列的问题就是对需求的一种自我解剖，等将需求弄彻底后在写程序就如探囊取物了，绝对是事半功倍的效果。说到设计思路那就不得不说一下数据库的设计了，其实需求分析的结果是将实际生活转化为数据库模型，最后再将数据库模型转化数据表，数据表就可以通过程序来进行控制，这样就可以达到编程实现系统功能的目的了。所以本章旨在挖掘需求，设计关系型数据库，在这一阶段，数据流图以及实体关系图的创建是必不可少的。

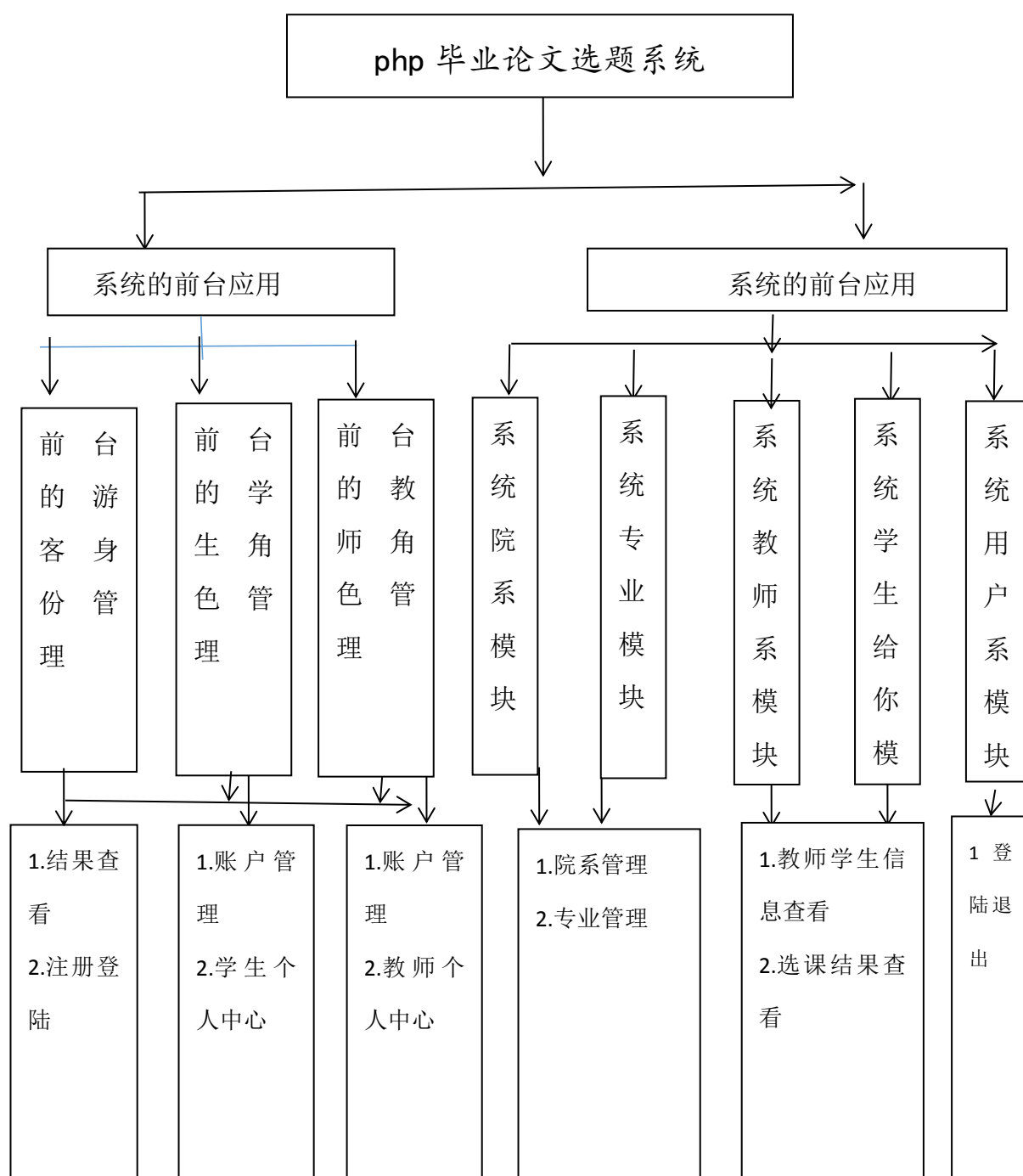
3.1 系统需求分析

需求分析往往是一个系统开发之前的重要前提，需要对实际的生活需求进行数据化和物理化，将复杂的生活场景简单化成容易懂的物理模型，然后将物理模型实现成程序的逻辑思路，需要不断的进行推敲，找出关系，挖出核心，探出何为重点，何为侧重点，将整个需求进行图表化，接下来我们就会用相关的思维导图来详细的展示该系统中的依赖关系。

需求分析是其实简单来说就是你在做数学题的时候读完题目进行一个简单的思考的过程，但是编程体系中这是一个很繁琐的过程，如果面向的是公司软件的开发那么这个流程会更佳的复杂，什么技术协议，网络安全，部门意见，还有调研结果，这一大堆的需求需要进行整合和归纳，最后按照编写规范完成需求分析书，期间会不断的进行修改，由此可见，需求分析是决定一套系统好坏，性能强弱的重要参考指标，一旦需求分析没有把控好，偏离了方向，做出来的产品绝对是畸形的，所以需求分析是这一节的重点和难点。

3.1.1 系统模块的划分

根据第一章已经列出来系统的主要功能。主要将系统在这里分为前台和后台两个不同应用，其中的前台又分为两个不同的角色（教师和学生），前台又分为登陆，改密，查看，选择等等不同的模块。后台也是由几个不同的模块组成的。



3.1.2 系统功能详情

1. 前台教师和学生以及游客模块：

- (1) 游客可以进行系统的正常登陆；并且查看基本的选课结果；
- (2) 教师或学生个人信息的第一次登录的强制添加以及后期的修改；
- (3) 教师或学生进行登录时候的密码的修改，其中使用第三方登录的无修改权限，修改密码系统会自动强制退出需要重新登录；
- (4) 教师可以进行自己相应课程的增加，删除和编辑操作，当学生进行选课时候教师会收到选课申请，可拒绝可接受，一旦没有一个人选中的课程教师可以重置再次让学生进行选择，教师是不可以进入到学生选课系统的中去，能进去选课的只有学生这唯一的角色。
- (5) 学生可以进行在线的选择题目，并且查看相应课程的选课进度同时可以进行撤销，选课一旦被老师查看，其反馈结果也会马上显示出来。如果没有一位老师接受申请了，可以进行第二次选课。不过必须重置选课。

2. 后台超级管理管理员模块：

主要实现以下几项功能：

- (1) 管理员的正常登陆，此时没有注册，密码强度比前台更高；
- (2) 管理员首先对选课院系及专业基本信息的增加；
- (3) 管理员可以对选课院系基本信息的编辑、增添、删除；
- (4) 管理员可以对选课院系所属专业基本信息的编辑、增添、删除；
- (5) 管理员可以查看院系以及专业所有正选结果；
- (5) 管理员对教师的信息浏览以及所发布课程的具体内容和选课状态的查看；
- (6) 管理员对学生的信息浏览以及所选课程的具体内容和选课状态的查看。

3.2 数据库的设计

在进行任何数据操作之前必须将系统所需要的数据库设计好，包括整个数据库是需要何种的数据存储类型，数据表的创建以及创建的表与表之间的数据关系，在一个系统中会很频繁的操作数据库，处理大量的数据，所以建立一个最优的数据库是提高系统的效率以及系统稳定性的关键点。因此，在正式书写代码之前，数据库的设计和建立是一项重要的内容。另外对于系统的多重性来讲，设计的数据库不是一天两天的时间，那

是一个系统开发的阶段，可能会反复的完善修改，达到一个最优的数据库才是系统应该正式应用的数据库设计。

3.2.1 系统的数据需求及数据关系

通过对功能的一一列举，所需的数据就显而易见，数据的需求是根据相应的具体功能进行一一添加的，需要对其中的数据的数据进行整理，找出哪些数据中的实体（比如学生，教师，课程），哪些是这些实体的属性，把实体和属性分开来，这样就是将复杂的现实关系转变为简单的描述性关系，最后将此种关系再抽象成一种数据化的模型就可得到所设计的数据库了。通过本系统的分析可以找出如下的几组关系：

（1）教师的角色作为一个实体，他的属性有姓名，所属专业，所属院系，头像，课程，学号，登录用户名，登录密码；

（2）学生的角色作为一个实体，他的属性同样也有有姓名，所属专业，所属院系，头像，选择的课程，学号，登录用户名，登录密码；

（3）管理员作为一个实体，他的属性有用户名和密码，其他的查看都是他产生的行为；

（4）将课程看成一个实体，它的属性有课程名字，课程介绍，课程所属教师，课程限制人数，课程的状态；

（5）将院系看做为一个实体，他的属性扩有院系名称，院系介绍，院系图标；

（6）将专业看做实体，他的属性包括专业名称，专业介绍，专业所属院系。

以的数据关系的，可以画出一下简单的关系如图 3.2.1。

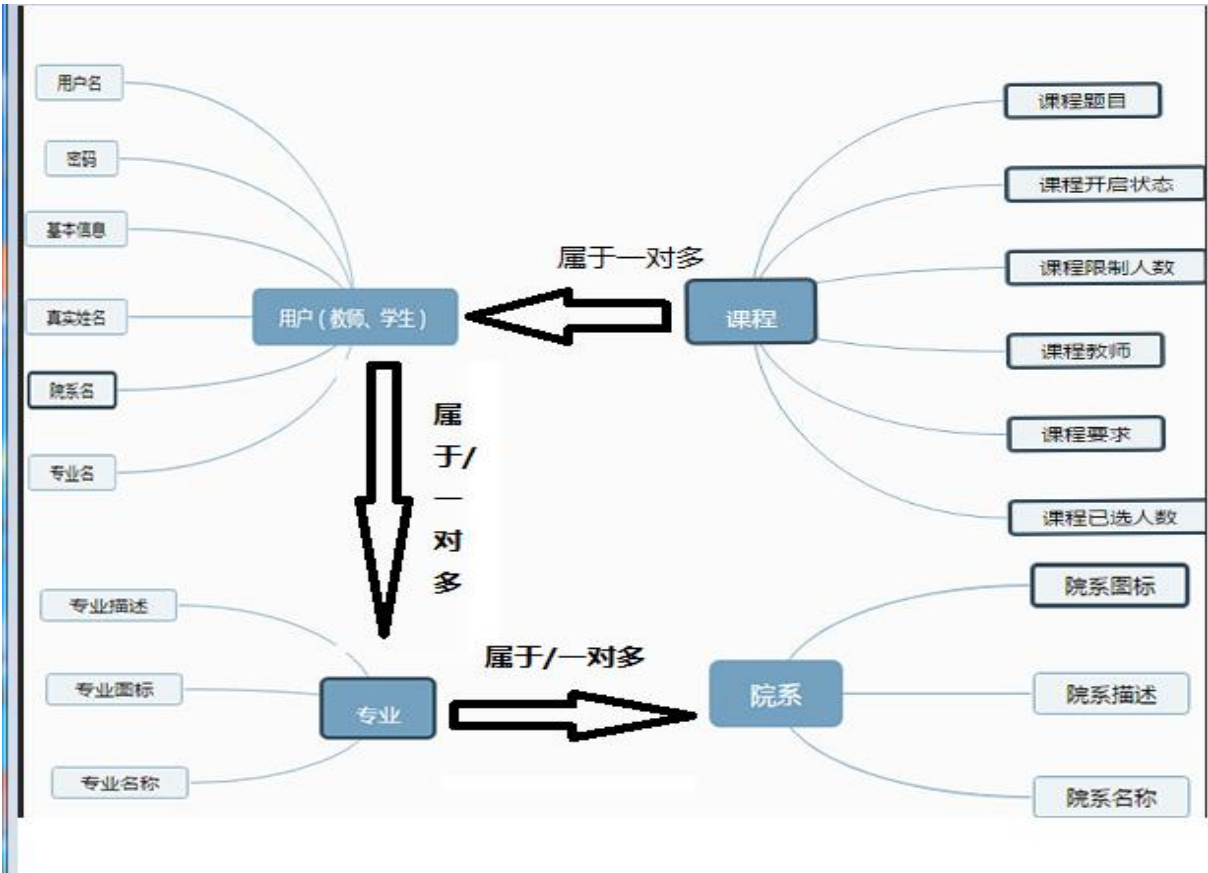


图 3.2.1 数据表简单关系图

3.2.2 设计数据表结构

以上的设计阶段只是简单的把其中的数据关系挖掘出来了，这里是数据库的设计的落地一步，将其中的关系逐一的进行整理并将其中的属性转换为表的字段，其中的实体转换为一个完整的数据表，数据关系通常用外键进行表示。由此可以轻松的得出来此次系统的数据表结构，详情如下所示。

表 3.2.2 学生 xk_student 表

tablename	xk_student(前缀 xk)_用于存储学生的基本信息,使用 MyISAM 类型		
列名	数据类型	属性/约束	说明
id	int(11)	主键自增非空无符	学生编号
username	VARCHAR(16)	非空	登录用户名
pwd	VARCHAR(32)	非空	学生登录密码
realname	VARCHAR(16)	非空	学生的真实姓名

class_id	int(11)	非空默认 0 外键索引	学生的专业 id
depart_id	int(11)	非空默认 0 外键索引	学生的院系 id
photo	char(80)	默认为空	学生的头像路径
createtime	timestamp	时间自动	学生注册的时间
studentid	char(12)	非空默认 0	学生的学号
iscomplete	Tinyint	非空默认 0	学生此时的状态

表 3.2.2 教师 xk_teacher 表

tablename	xk_teacher, (前缀 xk_) 用于存储教师的基本信息,使用 MyISAM 类		
列名	数据类型	属性/约束	说明
id	int(11)	主键自增非空无符	教师编号
username	VARCHAR(16)	非空	登录用户名
pwd	VARCHAR(32)	非空	登录密码
realname	VARCHAR(16)	非空	教师的真实姓名
class_id	int(11)	非空默认 0 外键索引	教师的专业 id
depart_id	int(11)	非空默认 0 外键索引	教师的院系 id
photo	char(80)	默认为空	教师的头像路径
createtime	timestamp	时间自动	教师注册的时间
studentid	char(12)	非空默认 0	教师的工号

表 3.2.2 课程 xk_course 表

tablename	xk_course(前缀 xk_),用于存储课题的基本信息,使用 MyISAM 类型		
列名	数据类型	属性/约束	说明
id	int(11)	主键自增非空无符	课题的编号
coursename	VARCHAR(16)	非空	课题的名称

desc	VARCHAR(200)	非空	课题的简介
limitnum	tinyint	非空	课题的限制人数
choosenum	tinyint	非空默认 0 外键索引	课题的已选人数
teacher_id	int(11)	非空默认 0 外键索引	教师的 id
status	tinyint	默认为空	课题的状态信息
createtime	timestamp	时间自动	课题创建的时间

表 3.2.2 选择课程 xk_order 表

tablename	xk_order(前缀 xk_),用于存储选择课程的基本信息,使用 MyISAM 类		
列名	数据类型	属性/约束	说明
id	int(11)	主键自增非空无符	选项课题的编号
student_id	int	非空默认 0 外键索引	课题的选择学生 id
calss_id	int	非空默认 0 外键索引	课题的专业 id
depart_id	int	非空默认 0 外键索引	课题的院系 id
isreceive	tinyint	非空默认 0	教师是否给学生回复
is_success	tinyint	非空默认 0	课题是否被选择成功
Course_id	tinyint	非空默认 0 外键索引	课题的 id
createtime	timestamp	时间自动	课题创建的时间

表 3.2.2 院系 xk_depart 表

tablename	xk_depart(前缀 xk_),用于存储院系的基本信息,使用 MyISAM 类型		
列名	数据类型	属性/约束	说明
id	int(11)	主键自增非空无符	院系的编号
name	VARCHAR(16)	非空	院系的名称
desc	VARCHAR(200)	非空	院系的简介

pic	cha(80)r	非空	院系的图标
-----	----------	----	-------

表 3.2.2 专业 xk_class 表

tablename	xk_class(前缀 xk)_,用于专业的基本信息,使用 MyISAM 类型		
列名	数据类型	属性/约束	说明
id	int(11)	主键自增非空无符	专业的编号
name	VARCHAR(16)	非空	专业的名称
pic	char(80)	非空	专业的图标
depart_id	int	非空外键	院系的 id

表 3.2.2 管理员 xk_user 表

tablename	xk_user(前缀 xk)_,用与后台管理员的基本信息,使用 MyISAM 类型		
列名	数据类型	属性/约束	说明
id	int(11)	主键自增非空无符	管理员的编号
username	VARCHAR(16)	非空	管理员的用户名
password	VARCHAR(20)	非空	管理员的密码
realname	char(5)	非空	管理员的名称
role_id	tinyint	非空外键	管理员的角色

第 4 章 系统总体设计及前后台应用的实现

前台应用的主要覆盖了有三大模块，即游客身份模块，教师身份模块，学生身份模块，在这三个模块下方存在有不同的操作实现，本系统使用 Thinkphp 开发对此进行三大模块的划分，首先了解一下利用 MVC 模式进行开发本套系统的总体设计。

4.1 系统的总体设计

采用 MVC 的设计模式，系统总体划分为三个典型的区域，视图显示区域，逻辑处理区域，交互控制区域，这三大部分相互分离同时又依赖着控制区域进行一个联系。根据之前对 Tp 的介绍这里列出来本系统的具体的设计结构，其中显示的多为目录具体文件部分不再进行一一展示，结构如下图 4.1 所示

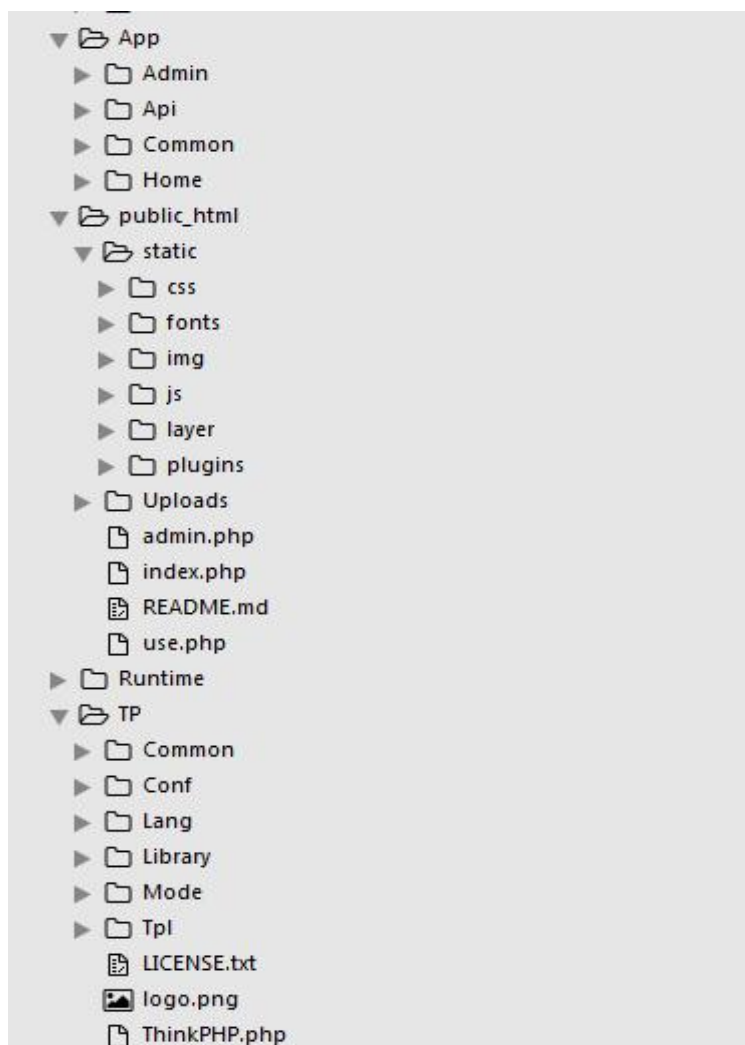


图 4.1 总体设计目录组织图

4.2 系统前台应用的实现

系统的前台应用有三大部分，其中的游客身份模块为公共模块，对于教师和学生完全开放的部分，简单来说就是用户登录前的操作，其次再分别把教师 and 学生的身份进行实现。

4.2.1 游客身份模块的实现

游客身份的模块主要可以进行一些简单的操作，只能是查看数据，不具备任何进行修改数据的权限，大致覆盖以下几个操作部分：首页查看，所有院系及结果查看，所有的教师及课程查看，所有的正选结果的查看，此外还有游客登录及注册的功能。针对以上的简单操作使用图标来进行详细的描述。

表 4-2-1-1 登录操作

操作名称	游客用户的登录操作
控制层文件名称	LoginController---dologin
相关的视图层模板	login.html, 包含前端的登录验证的 js 文件以及 ajax 提交文件
相关的数据层文件	暂不需要
用户访问输入	用户输入登录用户名和密码,并选取登录的角色
数据提交的方式	AJAX/POST
访问的返回输出值	数据验证通过跳转至所选角色的个人中心,验证失败则弹出错误信息。

表 4-2-1-2 注册操作

操作名称	游客用户的注册操作
控制层文件名称	IndexController-register
相关的视图层模板	register.html, 包含前端注册验证的 js 文件以及 ajax 提交文件
相关的数据层文件	StudentModel 以及 TeacherModel,包含后台数据验证代码
用户访问输入	用户注册时候用户名和密码以及角色, 验证码的输入
数据提交的方式	AJAX/POST

访问的返回输出值	数据验证通过则注册信息自动添加到数据库的学生表或者教师表中，弹出成功信息，验证失败则弹出错误信息。
----------	---

表 4-2-1-3 查看院系选课结果操作

操作名称	游客用户的院系结果查看操作
控制层文件名称	DepartController-detail
相关的视图层模板	detail.html
相关的数据层文件	暂不需要
用户访问输入	院系 id
数据提交的方式	GET
访问的返回输出值	选课结果表根据所传递的 id 进行查询，查询出表中所有 depart_id=id 的所有选课信息，并送至模板文件显示。

表 4-2-1-4 查看教师课题操作

操作名称	游客用户的教师课题发布查看操作
控制层文件名称	TeachersController--list
相关的视图层模板	list.html
相关的数据层文件	暂不需要
用户访问输入	教师 id
数据提交的方式	GET
访问的返回输出值	课程表根据所传递的 id 进行查询，查询出表中所有 course_id=id 的所有的课程信息，并送至模板文件显示。

4.2.2 学生身份模块的实现

学生是游客进行选择登录之后进入到的一个全新的模块，这里是本系统的学生部分的核心内容，尽管看上去页面的表现一样，但是其实已经完全由游客进入到了另外一个完全不同的模块区域。在本模块中主要进行的是退出操作，改密操作，基本资料填写操作，选课操作，查看课题操作，班级正选查看操作还有所有游客身份的操作（除了注册以及登录之外的）。

表 4-2-2-1 学生基本信息补充操作

操作名称	学生的基本信息的补充
控制层文件名称	StudentController--intro
相关的视图层模板	Intro.html, 包含前端用户输入信息的验证 js 以及 ajax 提交的 js
相关的数据层文件	StudentModel,后台验证用户的输入
用户访问输入	学生真实姓名, 学号, 所属院系, 所属专业的选择
数据提交的方式	POST/AJAX
访问的返回输出值	数据通过, 学生表根据当前登录用户的 id 在表中将所填写的数据进行更新, 数据失败则弹出错误信息。

表 4-2-2-2 学生头像设置的操作

操作名称	学生头像设置的操作
控制层文件名称	StudentController--photo
相关的视图层模板	photo.html, 包含前端用户输入信息的验证 js 以及 ajax 提交的 js
相关的数据层文件	StudentModel,后台验证用户的输入
用户访问输入	学生选择图片并上传
数据提交的方式	POST/AJAX
访问的返回输出值	数据通过, 学生表根据当前登录用户的 id 在表中将所上传的图片的路径信息更新到数据库, 数据失败则弹出错误信息。

表 4-2-2-3 学生修改密码的操作

操作名称	学生修改密码的操作
控制层文件名称	StudentController--safe

相关的视图层模板	safe.html, 包含前端用户输入信息的验证 js 以及 ajax 提交的 js
相关的数据层文件	StudentModel, 后台验证用户的输入信息
用户访问输入	新密码以及重复新密的输入
数据提交的方式	POST/AJAX
访问的返回输出值	数据通过, 学生表根据当前登录用户的 id 在表中新的密码更新值数据表中, 数据失败则弹出错误信息。

表 4-2-2-4 学生查看我的选课操作

操作名称	学生查看我的选课操作
控制层文件名称	StudentController--course
相关的视图层模板	Course.html
相关的数据层文件	OrderModel
用户访问输入	无参数, 后台代码自动查找
数据提交的方式	不提交
访问的返回输出值	控制层根据用户的 session 信息在选课表中查找出所有的 student_id=我的 id 的记录集, 并在前端动态判断状态信息。

表 4-2-2-5 学生班级正选操作

操作名称	学生班级正选操作
控制层文件名称	StudentController--classes
相关的视图层模板	class.html
相关的数据层文件	OrderModel
用户访问输入	无参数, 后台代码自动查找
数据提交的方式	不提交
访问的返回输出值	控制层根据用户的 session 信息在选课表中查找出所有的

	class_id=我的 class_id 的记录集，并在前端动态判断状态信息。
--	---

表 4-2-2-6 学生查看我的班级导师操作

操作名称	学生查看我的班级导师操作
控制层文件名称	StudentController--guide
相关的视图层模板	guide.html
相关的数据层文件	TeacherModel
用户访问输入	无参数，后台代码自动查找
数据提交的方式	不提交
访问的返回输出值	控制层根据用户的 session 信息在教师表中查找出所有的 id=我的 course_id 的记录集。

表 4-2-2-6 学生选择课题操作

操作名称	学生选择课题操作
控制层文件名称	Course--selectcourse
相关的视图层模板	index.html
相关的数据层文件	OrdeModel
用户访问输入	课题的 id
数据提交的方式	POST/AJAX
访问的返回输出值	后台所传递的课题 id,在选择课题表中根据用户登录的信息将数据添加到表中，返回成功提示，失败弹出错误信息。

表 4-2-2-6 学生撤销课题操作

操作名称	学生撤销课题操作
控制层文件名称	Course--delcourse
相关的视图层模板	index.html
相关的数据层文件	OrdeModel
用户访问输入	选择课题的 id
数据提交的方式	POST/AJAX
访问的返回输出值	后台所传递的课题 id,在选择课题表中根据传递的 id 删除该条记录，并弹回成功信息，失败弹出错误信息。

4.2.3 教师身份模块的实现

教师是游客进行选择登录之后进入到的另外一个全新的模块，这里是本系统的教师部分的核心内容，尽管看上去和学生身份模块页面的表现一样，但是其实已经完全由游客进入到了了一个完全不同的模块区域。在本模块中主要进行的是退出操作，改密操作，基本资料填写操作，这些操作和学生模块中的几乎一样不在做解释了，核心是来讲发布课题的操作，编辑课题的操作，修改课题的操作，查看课题选课情况，拒绝学生申请，接受学生申请，修改课程状态以及班级正选查看操作还有所有游客身份的操作（除了注册以及登录之外的）。

表 4-2-3-1 教师的课程增加与编辑操作

操作名称	教师的课程增加和编辑操作
控制层文件名称	TeacherController--save
相关的视图层模板	addclass.html
相关的数据层文件	CourseModel
用户访问输入	用户输入的名称，描述，限制人数，状态，
数据提交的方式	POST/AJAX
访问的返回输出值	控制层中将用户输入的课程信息并根据登录用户信息在课题表中增加一条该教师相关的记录集，失败则弹出错误信息。

表 4-2-3-2 教师的课程删除操作

操作名称	教师的课程删除操作
控制层文件名称	TeacherController--delete
相关的视图层模板	index.html
相关的数据层文件	CourseModel
用户访问输入	课程的 id
数据提交的方式	POST/AJAX
访问的返回输出值	控制层获取 ajax 传递的 id 值，并根据此 id 将数据表中的相关的记录集进行删除返回正确信息，失败返回错误信息。

表 4-2-3-3 教师课题查看操作

操作名称	教师的课程删除操作
控制层文件名称	TeacherController--course
相关的视图层模板	course.html
相关的数据层文件	CourseModel
用户访问输入	无需输入
数据提交的方式	无提交
访问的返回输出值	控制层中根据当前用户登录的信息，在课程表中将所有和当前用户相关课程信息查找出来，信息为空则显示提示信息。

表 4-2-3-4 教师的通知消息接受选课申请操作

操作名称	教师的通知消息接受选课申请操作
控制层文件名称	TeacherController--accept

相关的视图层模板	info.html
相关的数据层文件	OrderModel
用户访问输入	选择课程的 id
数据提交的方式	POST/AJAX
访问的返回输出值	控制层获取 ajax 传递的 id 值，并根据此 id 将选择课程表中的 is_success 字段进行更新操作为 1，失败返回错误信息。

表 4-2-3-5 教师的通知消息拒绝选课申请操作

操作名称	教师的通知消息拒绝选课申请操作
控制层文件名称	TeacherController--refuse
相关的视图层模板	info.html
相关的数据层文件	OrderModel
用户访问输入	选择课程的 id
数据提交的方式	POST/AJAX
访问的返回输出值	控制层获取 ajax 传递的 id 值，并根据此 id 将选择课程表中的 is_success 字段进行更新操作为 0，失败返回错误信息。

表 4-2-3-5 教师的正选结果查看操作

操作名称	教师的正选结果查看操作
控制层文件名称	TeacherController--classes
相关的视图层模板	info.html
相关的数据层文件	OrderModel
用户访问输入	无需输入
数据提交的方式	无需输入
访问的返回输出值	控制层根据用户的 session 信息在选课表中查找出所有的

	class_id=我的 class_id 的记录集,并在前端动态判断状态信息。
--	---

表 4-2-3-5 教师毕业班学生查看操作

操作名称	教师的正选结果查看操作
控制层文件名称	TeacherController--classes
相关的视图层模板	guide.html
相关的数据层文件	StudentModel
用户访问输入	无需输入
数据提交的方式	无需输入
访问的返回输出值	控制层根据用户的 session 信息在学生表表中查找出所有的 class_id=我的 class_id 的记录集。

4.3 系统后台应用的实现

系统的后台主要就一个管理员身份的角色,所以设计起来相对前天会比较简单,设计思路比较固定,实现的难度系数比较低,在管理员的实现中主要讨论管理员的登录以及登陆后的一系列操作,操作部分主要是对选课院系的添加增删改查,对专业的增删改查询,并且还有对专业和院系选课结果的查看,此外还有权限对所有的教师的课题进行查看以及课题目前的选课名单以及选课结果的查看,对于学生的具体名单学号以及学生所选的课程以及选课结果的反馈状态的查看。在管理员中没有设置注册的功能,考虑随意注册会对系统不安全,这里关闭了注册操作。

实际生活中,管理员通常是有不同的角色的,比如有普通管理员,超级管理员,学生管理员还有教师管理员,不同的管理员拥有不同的操作权限,鉴于该系统不需要那么多管理员身份,这里就不再进行管理员的权限划分问题,统一由超级管理员进行管理,不过在数据库部分已经对管理员分配了角色字段,方便后期进行管理员扩展。这里系统的设计不再一一展示了。

第 5 章 系统的调试与测试

系统在开大阶段的时候，注重是功能的实现，但是功能真的会一次性全部实现吗？答案肯定是否定，毕竟一套系统从开始开发到最终正式上线这是一个过程，没有哪一款软件能够一次性做到丝毫没有漏洞，这样的软件在自然界是不存在的，所以不可异想天开，没有经过测试和调试的软件对后期用户的数据安全性，还有软件的性能方面肯定会带来特别大的影响，不能等到系统出现大漏洞时候才进行各种修复排查。所以，软件开发完成后要在相当长的一个时间段内进行软件的调试与测试工作。

5.1 系统的调试意义及技巧

软件的调试工作，首先应该检查代码的完整性和代码的语法的准确性，因为现在大多数的 IDE 编辑器都拥有错误提示的机制，调试代码的时候如果发现有的地方出现了错的报警提示，首先应该从语法入手，检查代码中是否存在语法错误以及不规范的现象，这样可以在保证代码正确的前提下在进行代码的逻辑错误查找和测试这样有主有次的逐步调试有助于提高调试的效率。因为处于调试阶段 会出现反复的程序变动，所以这个时候建议使用 git 版本管理工具，这样一旦出现有些地方修改着修改着到最后觉得还不如不修改的时候，可以通过版本进行历史版本回退。

软件的调试的时候有几个典型的技巧，而不是凭空的进行调试，方法到位的话其实很容易就可以进行调试工作的。

首先，要清楚网站已经被分为 MVC 三个不同的区域，每一不同的区域都有不同的调试方法，对于视图显示页面的调试，单单从页面上是不出来任何东西的，错误就是错误的，正确的就是正确的，具体准确和错误原因如果不是作为开发者的是看不到，所以这里可以利用浏览器的开放性进行代码查看，将浏览器的控制台功能和代码查看功能全部打开，这样可以明显看到整个的网络情况，还有 js 的情况，以及从服务器传递过来的数据的具体情况。针对控制层，这里过多的调试工作是进行数据的输出显示，而不让他直接显示在视图模板中，首先打印数据先观察数据的正确性，一旦发现数据已经出现了错误那么及时传递到视图中去，依旧是错误的数据，这样会给调试工作带来更大的麻烦，所以控制层的调试必须要避开视图层来进行，这样将数据和视图进行分离进行一一调试，对将调试的区域大大缩小，控制范围减小对于查错是很有必要的。对于逻辑层，

这里基本上就是和控制层相互联系的，逻辑层不能单独工作，必须要在控制层中进行实例化的操作才会产生意义，所以这点必须依赖控制层进行，可以先设置一个简单的控制层，在此控制层中进行对逻辑层的实例化会更加方便的进行逻辑层的调试工作，这样不会对现有的控制层产生影响。

5.2 系统的测试

软件的测试必须要在开发的时候就开始不断地进行，这样可以及时的发现系统漏洞，当然系统测试也不是没有条例和章程的，软件的测试原则大致可以分为以下几部分内容：首先数据必须按照本系统的数据库设计来进行添加，不可随意的将数据塞入，保证数据的可用性，其次是对测试的结果全面的进行分析，将各种测试情景都进行模拟一下，尽量发现不同用户在使用系统中可能出现的问题，最后程序员应该避免测试自己的程序；这样有利于保证用户的随机性，更能对系统的稳定性进行一个有力的评估，把握以上几部分要点，对软件进行一个全方位测试。

5.3 系统功能预览

本次系统在不断的测试下，基本上已经可以正式上线，从前台到后台都做了一个全面的评估，暂时没有发现其他漏洞，这里进行一个本系统的功能一个初步预览，还是将系统分为前台和后台来进行划分的。

5.3.1 游客模块的功能展示

(1) 游客的全面展示，可以看在游客页面含有本系统的首页全部内容，游客可以记性相关的登录操作和院系以及教师的查看操作



图 5.3.1.1 游客首页展示

(2) 游客的登录及注册操作, 游客在这里可以注册以及登录的操作, 其中的登录操作并没有页面跳转而使用了 bootstrap 的模态框的操作, 这样可以简化登录的流程, 登录以及注册的功能全部使用 ajax 进行提交数据, 更好的用户体验, 防止页面的不断进行刷新。

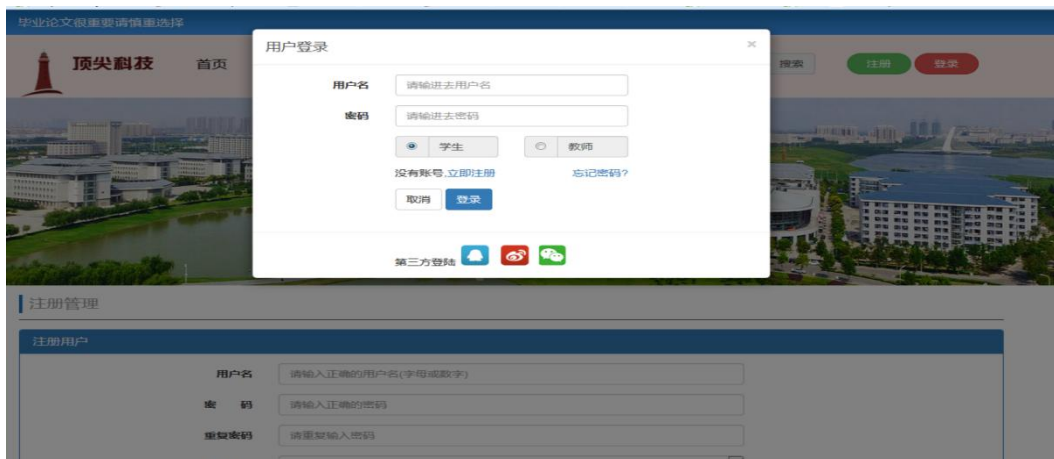


图 5.3.3.1 游客登录页面

(3) 游客具备院系和教师的查看权限, 将所有的教师和院系进行查看, 这里没有任何的信息无法进行筛选查询, 所以将全部的信息进行展示。

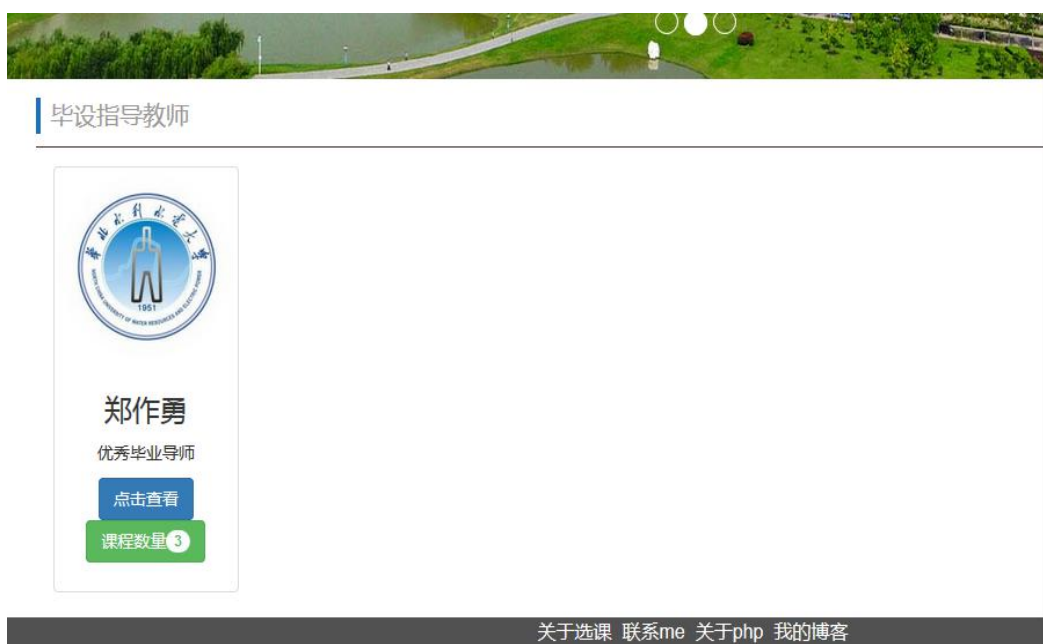


图 5.3.1.3 游客教师查看展示

5.3.2 学生模块的功能展示

(1) 学生首页直接跳转到我的选课的界面, 左侧是可选择的应用菜单, 右侧是菜单展示的具体功能和操作。



图 5.3.2.1 学生首页展示

(2) 首页的账户设置，账户设置中其中包含有基本信息的增加，修改，还有头像信息的增加和修改，密码的修改，这里全部使用 ajax 进行数据的提交，保证页面不进行提交刷新，提升用户的体验。账户菜单下的所有操作进行展示。



图 5.3.2.2 学生基本信息补充



图 5.3.2.3 学生头像设置



图 5.3.2.4 学生安全设置

(3)个人中心的功能展示，个人中心中可以查看所选的课题以及课题的反馈状态，同时在正选公示中查看所在班级的正选情况，以及所在班级的教师情况及教师的选课进度，其中还有核心的学生选课的部分。这里的学生选课仅仅学生有权限进入其他的均无进入权限。



图 5.3.2.5 学生选课查看

毕设选课详情

班级选题

选课结果

课程题目	指导教师	已选人数	课程要求	选课/
基于ph的订餐系统的	郑作勇		查看详情	在线选课

设计

共查到 1 条 共有1页

图 5.3.2.6 学生在线选课

个人中心

我的选课

正选公示

指导教师

账户设置

正选公示

学生学号	学生姓名	选课题目	指导老师	选课结果
201316602	王文帅	基于java的电子商城	郑作勇	正选
201316601	王帅	基于php的毕业论文	郑作勇	正选

选课系统注意

共查到 2 条 共有1页

图 5.3.2.7 学生班级正选公示

我的选课

正选公示

指导教师

账户设置

基本信息

郑作勇

图 5.3.2.8 学生指导教师查看

5.3.3 教师模块的功能展示

(1) 教师登录的首页和学生的首页展示几乎一样，其中的账户设置中的信息和学生的代码完全一样，只是其中的查找数据库选择的不用，主要阐述教师的个人中心，教师的在中心进行查看自己所发布的当前的课程的选择情况，并将反馈结果进行筛选显示，我的消息中查看学生的选课申请以及已经回复的学生申请情况。在班级公示中查看所在班级的学生名单并且查看该生的选课情况，在正选公示查看所在班级的正选情况。



图 5.3.3.1 教师首页



图 5.3.3.2 教师通知查看



图 5.3.3.3 教师班级公示查看



图 5.3.3.4 教师毕业班学生查看

5.3.4 后台管理员模块的功能展示

后台管理的模块含有的操作大部分都是增删改查的基本操作，其中含有多表查询，ajax 删除和修改等重技术手段，这里不在将功能一一展示，仅仅展示后台首页，具体的功能在附录中进行一一讲解。



图 5.3.4.1 后台管理员首页

第 6 章 软件维护

系统在开发的过程中是很重要的，后期的维护同样是一个关键的过程，每一个软件开发出来都不会说没有一丝丝的漏洞，所以后期的软件维护工作至关重要要，不可小视，通过对软件的测试时候发现不少的系统 BUG 要进行及时的修复，后期的系统维护工作必须要做精做细保证系统能够稳定的运行而不再出现其他 bug。要知道软件维护并非一朝一夕的，在软件维护的过程中，首先建立一个专业进行后期维护工作的机构，再之确立维护的过程和内容，对维护的处理步骤进行安排，建立起良好的维护流程。

第 7 章 总结

本次毕业设计开发的是一套关于学生进行论文课题选择的系统，通过对不同的资料进行查阅和以往的实际开发经验顺利的完成了本套系统的开发，期间出现这样或那样的棘手问题，但是总该会被解决掉，只要带着一颗不断求知解惑的心不管出现什么问题对于程序员来说都可以迎刃而解，完成此次系统的开发有三点最深的感触，这里进行分析一下。

首先是兴趣，兴趣是驱动力，无论做 JAVA 开发还是 C#开发的，必须将所用的领域做精做专，混乱的语言开发对于经验不足的程序员只会让程序很繁琐，选择的开发语言不一定是擅长但必须是感兴趣的，语言之间的差距微乎其微，时间的积累以及项目的实践必将会在该语言领域有一个很大的提高，所以，不求最擅长只要兴趣浓。

其次最大的感触是原来在必修课上讲解的许多东西有助于在实际开发中深入理解某一门语言，比如数据结构，内存存储，栈与堆，这些原来在课堂上其实都讲过，但是当你用面向对象来写程序的时候因为由于对这个栈或者堆没有深刻理解，所以有些属性和方法不能进行一个很好的把控。如果时间宽裕之余，应该将原来所学过的东西重新拾取这样在未来的开发中不管是旧知识还是新知识都会有一个更加深刻的见解，能够达到一个事半功倍的效果。

最后对本系统做一个自我评价，系统中的大部分功能已经实现，具体还存在哪些漏洞还需要进行大量的数据监测，否则小范围数据是看不出来有何等问题，需要经受得住大数据考验的系统才是好系统，所以还要后期尽心不断的性能测试以及稳定性测试，多目前的电脑端系统不是特别多，大部分存在的形式是以移动和手机应用，所以后期的大量工作是将系统的移动应用进行完善开发，保证拥有移动端和 pc 的双重系统。满足不同人群的使用需求，保证系统的使用率。

参考文献

- [1] 高洛峰. 细说 PHP 第二版[D]. LAMP 兄弟连 2011
 - [2] 潘凯华, 邹天思. PHP 开发实战宝典[M]. 北京: 清华大学出版社, 2010
 - [3] 陈湘扬, 陈国益. PHP5+MySQL 网页系统开发设计[M]. 北京: 电子工业出版社, 2007
 - [4] 邹天思, 孙鹏. PHP 从入门到精通[M]. 北京: 清华大学出版社, 2008
 - [5] 宋尚平 李兴保. PHP 模板引擎 Smarty 的安装配置及应用实现[M]. 2007
 - [6] 李峰, 晁阳. JavaScript 开发技术详解[M]. 北京: 清华大学出版社, 2009
 - [7] W. Jason Gilmore. PHP 与 MySQL5 程序设计[M]. 人民邮电出版社, 2007
 - [8] 徐鹏. 基于 Web 数据库的教务管理系统的设计与实现[J]. 价值工程. 2011 (03)
 - [9] 崔洋. MySQL 数据库应用从入门到精通[J]. 中国铁道出版社. 2013
 - [10] 王雨竹. MySQL 入门经典. 机械工业出版社[J]. 2013
 - [11] 王志刚. MySQL 高效编程. 人民邮电出版社[J]. 2012
 - [12] 钱雪忠. MySQL 数据库技术与实验指导[M]. 清华大学出版社. 2012
 - [13] 蔡旻. 基于 MVC 设计模式的协同设计系统的研究与实现[J] 西南交通大学, 20050601
 - [14] 林常须. 多客户端 MVC 设计模式的研究与应用. [M]. 兰州, 兰州理工大学, 200505
 - [15] (美) Alan Shallowly & James R. Trott. 设计模式精解. [M]. 北京, 清华大学出版社, 2004
-

致 谢

本次毕设的整个过程中，收获的不仅仅是完备的系统，更是毕业生在一块进行合作开发以及讨论所带来的喜悦，这份喜悦将是大学最难忘的，最值得纪念的，所以论文的成果不管如何，最重要的是这个制作的过程。同样还要感谢我的指导教师，从课程题目的选取到论文的开题的报告以及任务书，最后程序的制作过程中都有老师的陪伴，抽取大量的时间给我们进行毕业指导，送完毕业的最后旅程，感受到的是他兢兢业业的工作态，和这样的老师结伴而行，对于程序员来说真可谓莫大的幸福。编程的启蒙从指导老师这里开始的，当时连 C 语言都不懂得我听他讲的 JAVA 课程津津有味，庆幸的是现在已经把 C 语言完全想清楚了，但是依旧遗憾的是数据结构和计算机原理始终没有弄明白，也许是没有重新拾起，但是未来如果想要向程序开发的更高方面发展的话，这些理论是必修的。

最后，同样感谢所有曾经或是现在教导我的老师，你们孜孜不倦的教学情景似乎就停留在昨天，一届又一届，一年又一年，送走了一批批的毕业生，各位老师，你们辛苦了！

附 录

附录 I

外文原文

The World According to LINQ

Programmers building web- and cloud-based applications wire together data from many different sources such as sensors, social networks, user interfaces, spreadsheets, and stock tickers. Most of this data does not fit in the closed and clean world of traditional relational databases. It is too big, unstructured, denormalized, and streaming in real time. Presenting a unified programming model across all these disparate data models and query languages seems impossible at first. By focusing on the commonalities instead of the differences, however, most data sources will accept some form of computation to filter and transform collections of data.

Mathematicians long ago observed similarities between seemingly different mathematical structures and formalized this insight via category theory, specifically the notion of monads as a generalization of collections. Languages such as Haskell, Scala, Python, and even future versions of JavaScript have incorporated list and monad comprehensions to deal with side effects and computations over collections. The .NET languages of Visual Basic and C# adopted monads in the form of LINQ (Language-integrated Query) as a way to bridge the gap between the worlds of objects and data. This article describes monads and LINQ as a generalization of the relational algebra and SQL used with arbitrary collections of arbitrary types, and explains why this makes LINQ a compelling basis for big data.

LINQ was introduced in C# 3.0 and Visual Basic 9 as a set of APIs and accompanying language extensions that bridge the gap between the world of programming languages and the world of databases. Despite the continuing excitement about LINQ in the external developer community, the full potential of the technology has not yet been reached. Thanks to the foundational nature of LINQ, there is still enormous potential for its mapping scenarios outside object-relational (O/R), especially in the area of big data.

The advent of big data makes it more important than ever for programmers to have a single abstraction that allows them to process, transform, compose, query, analyze, and compute across at least three different dimensions: volume, big or small, ranging from billions of items to a handful of results; variety in models, structured or unstructured, flat or nested; and velocity, streaming or persisted, push or pull. As a result, we see a mind-blowing number of new data models, query languages, and execution fabrics. LINQ can virtualize all these aspects behind a single abstraction.

Take, for example, Apache's Hadoop ecosystem. It comes with at least eight external DSLs (domain-specific languages) or APIs: a set of low-level Java interfaces for MapReduce computations; Cascading, a "data-processing definition language, implemented as a simple Java API;" Flume, a "simple and flexible architecture based on streaming data flows;" Pig a "high-level language for expressing data analysis programs;" HiveQL, an "SQL-like language for easy data summarization, ad hoc queries, and the analysis of large data sets;" CQL, a "proposed language for data management in Cassandra;" Oozie, an XML-based "coordinator engine specialized in running workflows based on time and data triggers;" and Avro, a schema language for data serialization.

To create an end-to-end application, programmers need to use several of these external DSLs in addition to a general-purpose programming language such as Java to glue everything together. If data comes from an external RDBMS (relational database management system) or push-based source, then even more DSLs such as SQL or StreamBase are required. Using LINQ and C# or Visual Basic on the other hand, programmers can use internal DSLs to program against any shape or form of data inside a general-purpose OO (object-oriented) language that comes with tooling (Visual Studio or cross-platform solutions from Xamarin such as MonoDevelop, Mono Touch for iPhone, or Mono for Android) and an extensive collection of standard libraries (.NET Framework).

Standard Query Operators and LINQ

Assume that given a file of text—say, words.txt—you need to count the number of distinct words in that file, find the five most common ones, and visualize the result in a pie chart. If you think about this for a minute, it becomes clear that this is really an exercise in

transforming collections. This is exactly the kind of task for which LINQ was designed. To keep things simple, we have implemented this example using LINQ to Objects to process the data in memory; however, with minimal modification the same code runs on LINQ to HPC (high-performance computing) over terabytes of data stored in commodity clusters.

The standard `File.ReadAllText` method provides the content of the file as a single giant string. You first need to chop up this string into individual words by breaking it at delimiter characters such as space, comma, period, etc. Once you have a list of words, you need to clean it up, removing all empty words. Finally, normalize all words to lowercase.

Using the LINQ sequence operators, you can transliterate the description from the previous paragraph directly into code:

```
var file = System.IO.File.  
ReadAllText("words.txt");  
var words = file.Split(delimiters)  
                .Where(w => !w.IsNullOrEmpty()  
                .Where(w => !w.IsNullOrWhiteSpace())  
                .Select(w => w.ToLower());
```

Instead of using the sequence operators directly, LINQ also provides a more "declarative" query comprehension syntax. Using comprehensions, you can rewrite the code as follows:

```
var words =  
    from w in file.Split(delimiters)  
    where !w.IsNullOrEmpty()  
    select w.ToLower();
```

Once you have converted the file into a sequence of individual words, you can find the number of occurrences of each word by first grouping the collection by each word and then counting the number of elements in each group (which contains all occurrences of that word):

```
var wordcount =  
    from w in words  
    group w into group  
    select new { Word = group.Key,  
                Count = group.Count() };
```

Without using query comprehension syntax, the code would look like this:

```
var wordcount = words.GroupBy(w => w).  
                    Select(group =>  
                        new { Word =  
                            group.Key, Count =  
                            group.Count() });
```

To find the top five most frequent words, you can order each record by `Count` and take the first five elements:

```
var top5 = wc.OrderByDescending(p =>
    p.Count).Take(5);
```

Now that you have a collection of the top five words in the file, you can visualize them in a pie chart, as in Figure 1. A pie chart is really nothing more than a collection of slices, where each slice consists of a number that represents the proportion of the total pie and a legend that describes what the slice represents. This means that by defining the charting API to be LINQ friendly, you can create charts by writing a query over the Google image charts API:

```
var chart = new Pie(from w in top5
    select new Slice(w.Count){ Legend = r.Word })
{Title = "Top 5 words" };
var image = await chart;
```

The `await` keyword is used in an unorthodox way to make the expensive coercion from `Google.Linq.Charts.Pie` into an image that requires a network round-trip explicit.

This example just scratches the surface of LINQ. It provides a library of sequence operators such as `Select`, `Where`, `GroupBy`,... to transform collections, and it provides syntactic sugar in the form of query comprehensions that allows programmers to write transformations over collections at a higher level of abstraction.

To truly understand the power of LINQ, let's take a step back and investigate its origins and mathematical foundations. Don't worry, you need knowledge of only high school-level mathematics.

To truly understand the power of LINQ, let's take a step back and investigate its origins and mathematical foundations. Don't worry, you need knowledge of only high school-level mathematics.

Datacentric Interpretation

Relational algebra, which forms the formal basis for SQL, defines a number of constants and constructors for sets of values $\{\Sigma\}$, such as the empty set $\emptyset \in \{\Sigma\}$; injection of a value into a singleton collection $\{_ \} \in \Sigma \rightarrow \{\Sigma\}$; and the union of two sets into a new combined set $\cup \in \{\Sigma\} \times \{\Sigma\} \rightarrow \{\Sigma\}$; There are also a number of relational operators such as projection, which applies a transformation to each element in a set $\pi \in (\Sigma \rightarrow \Lambda) \times \{\Sigma\} \rightarrow \{\Lambda\}$; selection,

which selects only those elements in a set that satisfy a given property $\sigma \in (\Sigma \rightarrow \mathbb{B}) \times \{\Sigma\} \rightarrow \{\Sigma\}$; Cartesian product, which pairs up all the elements of a pair of sets $X \in \{\Sigma\} \times \{\Lambda\} \rightarrow \{\Sigma\Lambda\}$; and cross-apply, which generates a secondary set of values for each element in a first set $@ \in (\Sigma \rightarrow \{\Lambda\}) \times \{\Sigma\} \rightarrow \{\Lambda\}$.

Figure 2 depicts the relational algebra operators using clouds to denote sets of values. An SQL compiler translates queries expressed in the familiar SELECT-FROM-WHERE syntax into relational algebra expressions; to optimize the query it applies algebraic laws such as distribution of selection: $\sigma(p, \sigma(q, xs)) = \sigma(x \Rightarrow p(x) \wedge q(x), xs)$ and then translates these logical expressions into a physical query plan that is executed by the RDBMS.

For example, the SQL query `SELECT Name FROM Friend WHERE Likes(Friend, Sushi)` is translated into the relational algebra expression $\pi(f \Rightarrow f.Name, (\sigma(f \Rightarrow \text{Likes}(f, \text{Sushi}), \text{Friend})))$. To speed up the execution of the query, the RDBMS may use an index to quickly look up friends who like Sushi instead of doing a linear scan over the whole collection.

The cross-apply operator $@$ is particularly powerful since it allows for correlated subqueries where you generate a second collection for each value from a first collection and flatten the results into a single collection $@(f, \{a, \dots, z\}) = f(a) \cup \dots \cup f(z)$. All other relational operators can be defined in terms of the cross-apply operator:

$$\begin{aligned} xs \times ys &= @(x \Rightarrow \pi(y \Rightarrow (x, y), ys), xs) \\ \pi(f, xs) &= @(x \Rightarrow \{f(x)\}, xs) \\ \sigma(p, xs) &= @(x \Rightarrow p(x) ? \{x\} : \emptyset, xs) 1 \end{aligned}$$

As a programmer you can easily imagine writing up a simple implementation of cross-apply: you would just iterate over the items in the input set, apply the given function, and accumulate the results into a result set. Such an implementation, however, wouldn't need its argument to be as set $\{\Sigma\}$; anything that we can iterate over such as a list, array, or hash table would suffice. Similarly, there is no reason at all that relational algebra operations should be restricted to sets of values $\{\Sigma\}$. They can be implemented based on other types of collections as well.

Perhaps surprisingly, there is also no reason that the operations passed into π , σ , and $@$ should be restricted to concrete functions $\Sigma \rightarrow \Lambda$. In fact, you can use any representation of a

function from which to determine which computation to perform. For example, in a language such as JavaScript you could simply pass a string and then use `eval` to turn it into executable code.

What you are searching for is the underlying interface that relational algebra implements. As long as there is a type constructor for collections $M<\Sigma>$ that provides the operations that satisfy similar set-like algebraic properties as $\{\Sigma\}$, and a type constructor for computations $\Sigma \rightarrow \Lambda$ that satisfies similar function-like properties as $\Sigma \rightarrow \Lambda$, you can generalize relational algebra to the following set of operators and still be able to write SQL queries over these collections by desugaring query syntax:

```

 $\emptyset \in M<\Sigma>$ 
 $\{ \_ \} \in \Sigma \rightarrow M<\Sigma>$ 
 $\cup \in M<\Sigma> \times M<\Sigma> \rightarrow M<\Sigma>$ 
 $\otimes \in (\Sigma \rightarrow M<\Lambda>) \times M<\Sigma> \rightarrow M<\Lambda>$ 

```

For programmers this is just separating interface from implementation; mathematicians call the resulting structure monads, and instead of queries they speak of comprehensions.

An OO language such as C# uses the canonical interface for collections `IEnumerable<T>` as a specific instance of the abstract collection type $M<T>$ and uses delegates `Func< Σ , Λ >` to represent computations $\Sigma \rightarrow \Lambda$. By doing this, you recognize the operators from relational algebra as the LINQ standard query operators as defined in the `Linq.Enumerable` class, as shown in Figure 3.

Alternatively, you can use the `IQueryable<T>` interface to represent collections $M<T>$ and expression `treeExpression<Func< Σ , Λ >>` to represent computations $\Sigma \rightarrow \Lambda$. In that case you recognize the relational algebra operators as the LINQ standard query operators as defined in the `Linq.Queryable` class. The ability to treat code as data using morphisms—or in the C# case using the `Expression` type and lambda expressions for code literals—is a fundamental capability that allows the program itself to manipulate, optimize, and translate queries at runtime.

Instead of SQL syntax, the C# language defines XQuery-like comprehensions of the form from-where-select. The previous SQL query example looks like this:

```

from friend in friends where friend.
Likes(Sushi) select friend.Name

```

Just as in SQL, comprehensions are translated by the compiler into the underlying LINQ query algebra:

```
friends.Where(friend=>friend.  
Likes(Sushi)).Select(friend=>friend.Name)
```

Depending on the overloads of Where and Select, the lambda expressions will be interpreted as code or data. A simplified implementation of IQueryable is discussed later.

As already shown, monads and their incarnation in practical programming languages such as LINQ are simply a generalization of relational algebra by imagining the interface that relational algebra implements. The concepts and ideas behind LINQ should therefore be deeply familiar to both database people and programmers.

Theory into Practice

Unlike Haskell, which has incorporated monads and monad comprehensions in a principled way, the C# type system is not expressive enough for the mathematical signatures of the monad operators. Instead, the translation of query comprehensions is defined in a purely pattern-based way. In a first pass, the compiler blindly desugars comprehensions, using a set of fixed rules, into regular C# method calls and then relies on standard type-based overload resolution to bind query operators to their actual implementations.

For example, the method `Foo Select(Bar source, Func<Baz, Qux> selector)`, which does not involve any collection types, will be bound as the result of translating the comprehension

```
var foo = from baz in bar select qux
```

into the desugared expression

```
var foo = bar.Select(baz=>qux)
```

This technique is used extensively in the example presented next.

Another difference between LINQ and its monadic basis is a much larger class of query operators including grouping and aggregation, which is more SQL-like. Interestingly, the inclusion of comprehensions in C#, which was inspired by monad and list comprehensions in Haskell, has recursively inspired Haskell to add support for grouping and aggregation to its comprehensions.

Custom Query Providers

The Yahoo weather service (<http://developer.yahoo.com/weather/>) allows weather

forecast queries for a given location, using either metric or imperial units for the temperature. This simple service is a good way to illustrate a non-standard implementation of the LINQ query operators that is completely specialized for this particular target and that will allow only strongly typed queries of the form

```
var request = Yahoo.WeatherService().
    Where(forecast=>forecast.City == city).
    Where(forecast=>forecast.
        Temperature.In.units);
var response = await request;
```

or equivalently using query comprehensions

```
var request =
    from forecast in Yahoo.WeatherService()
    where forecast.City == city
    where forecast.Temperature.In.units
    select forecast;
var response = await request;
```

The implementation of the operators extracts the city and temperature unit from the query and uses them to create a REST call (<http://weather.yahooapis.com/forecastrss?w=woeid&u=unit>) to the Yahoo service as a result of using the `await` keyword to explicitly coerce the request into a response.

The technical trick in this style of custom LINQ provider is to project the capabilities of the target query language—in this case the Yahoo weather service that requires (a) a city and (b) a unit—into a type-level state machine that guides users in "fluent" style (and supported by IntelliSense) through the possible choices they can make (Figure 4).

At each transition in the state machine we collect the various parts of interest of the query—in this case, the particular city and the temperature unit. In principle, the city doesn't really need to come first, but it might be more natural for the graph to allow either type of where clause to be specified first, but with the restriction that both where clauses are required. I leave the lifting of this restriction in the state machine as an exercise for the reader.

Note that none of the types `Weather`, `WeatherInCity`, or `WeatherIn-CityInUnits` implements any of the standard collection interfaces. Instead they represent the stages in the computation of a request that will be submitted to the Yahoo Web service, for which you do not need to define an explicit container type. What also surprises many people is that neither of the two `Where` methods actually computes a Boolean predicate.

Even stranger is that each of the three occurrences of the range variable `forecast` in the query has a different type.

The `Weather` class defines a single method that picks the city specified in the query and passes it on to `WeatherInCity`, which is the next state in the type-based state machine:

```
WeatherInCity Where(Weather source,
Func<CityPicker,string> city)
{
    return new WeatherInCity{ City =
        city(new CityPicker()) };
}
```

The "predicate" in the `Where` method is a function that takes a value of type `CityPicker`, which has a single property that returns the phantom class `City` that exists only to facilitate `IntelliSense` and whose equality check immediately returns the string passed to the equality operator:

```
class CityPicker { City city; }
class City
{
    static string operator == (City c,
        string s) { return s; }
}
```

As a result of this, calling `Yahoo.Weather().Where(forecast=> forecast=="Seattle")` really is just a convoluted way of creating a new `WeatherInCity{City = "Seattle"}` instance using a `Where` method that does not take a Boolean predicate and an equality operator that returns a string.

You can use the same trickery in `WeatherInCityInUnits Where(Func<UnitPicker,Unit> predicate)`, so that calling `Where(forecast=> forecast.Temperature.In.Celsius)` on the result of the previous filter creates an instance of new `WeatherInCityInUnits{City = "Seattle", Unit = Unit. Celsius}`. The techniques used here are not only useful for defining custom implementations of the LINQ operators, but also can be leveraged for building fluent interfaces in general.

Since the Yahoo service requires the city as a WOEID (where on earth ID), we need to make two service calls under the hood in order to retrieve the weather forecast. The first service call retrieves the WOEID of a requested city via `http://where.yahooapis.com/v1/places.q(city)?appid=XXXX`. If that successfully returns,

then a second call is made to retrieve the weather forecast for that location. The calls to the Web server are performed asynchronously and both return a `Task<T>` (in Java you would use `java.util.concurrent.Future<T>` to represent the result of an asynchronous operation). Since we can consider a `Task<T>` as a kind of collection that contains (at most) one element, it also supports the LINQ query operators, and we have turtles all the way down; the LINQ implementation for `Weather` is defined using the LINQ implementation of `Task<T>` (see Figure 5).

Though this is an extremely small and limited example, it clearly illustrates many of the techniques used to create real-world LINQ providers such as LINQ to Objects, LINQ to SharePoint, LINQ to Active Directory, LINQ to Twitter, LINQ to Netflix, and many more.


Generic Query Providers

The weather service query provider example is structured as an internal DSL. While this provides a great user experience with maximum static typing, it allows little room for reusing the actual implementation of the provider. It is custom built for the particular target top-to-bottom. At the other end of the spectrum we can create a completely generic query provider that records a complete query "as is," using a little bit of meta-programming magic.

In C# a lambda expression such as `x => x > 4711` can be converted into either a delegate—say, of type `Func<int,int>`—or into an expression tree of type `Expression<Func<int,int>>`, which treats the code of a lambda expression as data. In Lisp or Scheme one would use syntactic quoting to treat code as data. In C# lambda expressions in combination with the type expected by the context provide a type-based quoting mechanism.

The class `Queryable` implements LINQ standard query operators that take expression trees as arguments and return an `Expression` representation of their selves, very much like a macro recorder as shown in Figure 6.

For example, given a value `xs` of type `Queryable<int>`, the call `xs.Select(x => x > 4711)` causes the lambda expression to be converted into an expression tree (shown in bold), and then returns an expression tree that represents the call

itself `xs.Select(x  x>4711)`. Now it is up to the specific query provider (such as LINQ to SQL, Entity Framework, LINQ to HPC) to translate the resulting expression tree and compile it into the target query language.

The IQueryable-based implementation that ships with the .NET Framework uses the same scheme as the simplified example code just shown, except that it is interface based, and it therefore relies on a second interface `IQueryProvider` to supply a factory for creating instances of `IQueryable`.

The advantage of a generic query provider is that you can offer general services such as query optimization, which implement rewrite rules such as `xs.Union(ys).Where(p) = xs.Where(p). Union(ys.Where(p))` that can be reused across many LINQ providers.

LINQ-Friendly APIs

All examples so far have dealt with implementing particular LINQ providers. An orthogonal aspect of LINQ is APIs that leverage particular LINQ implementations, often LINQ to Objects. For example, LINQ to XML is an API for manipulating XML documents that has been designed specifically with LINQ in mind, which eliminates the need for a DSL such as XQuery or XPath to query and transform XML.

The Google Chart API is a Web service that lets you dynamically create attractive-looking charts, using a simple URI (Uniform Resource identifier) scheme. The URI syntax for Google charts, however, is not very sequence friendly. For example, the URI for the earlier sample pie chart looks like this:

```
http://chart.apis.google.com/chart
?cht=p3&chtt=Top+5+words&chs=
500x200
&chd=t:21,12,7,7,6
&chl=the|of|a|that|is
```

The problem is that the specification for the labels (`chl=the|of|a|that|is`) and the specification for the data set (`chd=t:21,12,7,7,6`) of the chart are given in two separate collections. On the other hand, to generate a pie chart using a query, you want a single collection of pairs that specify both the value and the label for each slice as in `from w in top5 select new Slice(w.Count){Legend = r.Word}`.

In other words, to make the Google Chart API sequence friendly, you must transpose a collection of pairs $M\langle S \times T \rangle$ into a pair of collections $M\langle S \rangle \times M\langle T \rangle$. Functional programmers immediately recognize this as an instance of the function $\text{Unzip} \in (R \rightarrow S \times R \rightarrow T \times M\langle R \rangle) \rightarrow M\langle S \rangle \times M\langle T \rangle$. Unzip can convert a chart that contains a sequence of slices into the URI format required by the Google Chart API by formatting the various collections using the separators prescribed by the chart service as shown in Figure 7.

Conclusion

Big data is not just about size. It is also about diversity of data, both in terms of data model (primary key/foreign key versus key/value), as well as consumption pattern (pull versus push), among many other dimensions. This article argues that LINQ is a promising basis for big data. LINQ is both a generalization of relational algebra and has deep roots in category theory—in particular, monads.

With LINQ, queries expressed in C#, Visual Basic, or JavaScript can be captured either as code or expression trees. Either representation can then be rewritten and optimized and subsequently compiled at runtime. We have also shown how to implement custom LINQ providers that can run in memory and over SQL and CoSQL databases, and we have presented LINQ-friendly APIs over Web services. It is also possible to expose streaming data so as to implement the LINQ standard query operators, resulting in a single abstraction that allows developers to query over all three dimensions of big data.

Acknowledgments

Many thanks to the Cloud Programmability team members Savas Parastatidis, Gert Drapers, Aaron Lahman, Bart de Smet, and Wes Dyer for all the hard work in building the infrastructure and prototypes for all flavors of LINQ and coSQL; to Rene Bouw, Brian Beckman, and Terry Coatta for helping to improve the readability of this article; and to Dave Campbell and Satya Nadella for providing the necessary push to actually write it.

中文译文

根据 LINQ 的世界

程序员从许多不同的来源，例如传感器，社交网络，用户界面，电子表格和股票行情构建 Web 和基于云的应用程序一起线数据。大多数这类数据不适合在传统的关系型数据库的封闭和清洁的世界。它太大了，非结构化，非规范化和流媒体的实时性。呈现在所有这些不同的数据模型和查询语言的统一编程模型起初似乎是不可能的。通过集中在共同点代替的差异，但是，大多数的数据源将接受某种形式的计算来过滤和转换数据的集合。

数学家看似不同的数学结构之间早就观察到的相似，正式通过范畴论，单子专门的概念作为收藏品的推广这种洞察力。语言如 Haskell 中，斯卡拉的 Python 和 JavaScript，甚至未来的版本中已纳入清单，单子解析来处理副作用和计算过的集合。Visual Basic 和 C# 的 .NET 语言通过了 LINQ（语言集成查询）的形式单子，以此来弥补对象和数据的世界之间的差距。本文介绍了单子和 LINQ 与任意类型的任意集合使用的关系代数和 SQL 的推广，并解释了为什么这使得 LINQ 大数据令人信服的依据。

LINQ 介绍在 C# 3.0 和 Visual Basic 9 为一组 API 和相应的语言扩展，弥补编程语言的世界和数据库世界之间的差距。尽管在外部开发者社区有关 LINQ 的持续刺激，该技术的全部潜力尚未达到。由于 LINQ 的基本性质，仍然有巨大的潜力，其映射情况之外的对象关系（O/R），尤其是在大数据领域。

大数据的出现，使得它比以往任何时候都程序员有一个抽象，让他们来处理，转换，作曲，查询，分析，并计算横跨至少有三个不同的维度更重要的是：音量，或大或小，从数十亿;项目成果屈指可数的各种模型中，结构化或非结构化，扁平或嵌套;和速度，流或持续，推或拉。其结果是，我们看到了新的数据模型，查询语言和执行面料令人兴奋的数字。LINQ 可以虚拟一个抽象背后所有这些方面。

举个例子来说，Apache 的 Hadoop 的生态系统。它配备了至少八个外部的 DSL（领域特定语言）或 API：一组低级别的 Java 接口为 MapReduce 的计算;层叠，一个“数据处理定义语言，作为一个简单的 Java API 实现，”，“基于流数据的简单和灵活的架构流程，”，承载大量数据的“高层次的语言表达数据分析程序;” HiveQL，一个“类似于 SQL 的语言，便于数据的汇总，即席查询和大型数据集分析;” CQL，一个“建议的语言对 Cassandra 的数据管理;” Oozie 的，基于 XML 的”专业从事基于时间和数据运

行的工作流协调引擎触发; “和 Avro 的, 对于数据串行化的模式语言。

要创建一个终端到终端应用, 程序员需要使用多个这些外部 DSL 的除了通用编程语言如 Java 胶水都在一起。如果数据来自外部 RDBMS (关系数据库管理系统), 或推基于源, 然后更 DSL 的诸如 SQL 或 StreamBase 是必需的。另一方面使用 LINQ 和 C# 或 Visual Basic, 程序员可以使用内部 DSL 的编程针对附带工具 (Visual Studio 或跨平台解决方案的任何形状或通用的面向对象的内部 (面向对象) 语言数据的形式从 Xamarin 如 MonoDevelop 的, 单触 iPhone 或单声道的 Android) 和标准库, 广泛收集 (.NET 框架)。

标准查询操作和 LINQ

假设给定的文本, 比如说一个文件, words.txt -你需要计算的不同词的数量在该文件中, 找到五个最常见的, 而在饼图中显现的结果。如果你觉得这个一分钟, 很明显, 这是真的转化集合练习。这正是那种任务为其 LINQ 的设计。为了简单起见, 我们已经实现了使用 LINQ to 对象来处理内存中的数据这个例子; 然而, 稍加修改同样的代码在 LINQ 对存储在商品集群 TB 级的数据运行到 HPC (高性能计算)。

标准 File.ReadAllText 方法提供的文件的内容作为单个巨型串。首先, 您需要通过分隔符如空格, 逗号, 句号等, 一旦你有一个单词列表打破它砍了这个字符串转换成个人的话, 你需要把它清理干净, 清除所有空话。最后, 规范所有单词小写。

用 LINQ 顺序运算符, 可以音译从前面的段落直接进入代码的描述:

```
var file = System.IO.File.  
ReadAllText("words.txt");  
var words = file.Split(delimiters)  
                .Where(w=>!w.IsNullOrEmptyOr-  
                WhiteSpace())  
                .Select(w=>w.ToLower());
```

而不是直接使用序列运营商, LINQ 还提供了更多的“声明”查询理解语法。使用解析, 你可以重写代码如下:

```
var words =  
    from w in file.Split(delimiters)  
    where !w.IsNullOrEmptyOrWhiteSpace()  
    select w.ToLower();
```

一旦你已将该文件转换成单个字的序列, 则可以通过首先由每个单词分组的集合, 然后计数各组中的元素的数量 (其包含该字的所有实例) 找到的每个单词的出现次数:

```
var wordcount =
    from w in words
    group by w into group
    select new{ Word = group.Key,
                Count = group.Count() };
```

如果不使用查询理解语法，代码是这样的：

```
var wordcount = words.GroupBy(w⇒w).
    Select(group⇒
        new{ Word =
            group.Key, Count =
            group.Count() });
```

要查找的五大最频繁的话，你可以命令每个记录计数，并采取先五个要素：

```
var top5 = wc.OrderByDescending(p⇒
    p.Count).Take(5);
```

现在，你有文件中的前五个词的集合，你可以想像他们在饼图中，如在图 1 中。饼图实在没有什么比片的集合，其中每个片由若干更多的代表总馅饼和介绍了片代表一个传奇的比例。这意味着，通过定义图表 API 是 LINQ 友好，你可以写在谷歌图片图表 API 查询创建图表：

```
var chart = new Pie(from w in top5
    select new Slice(w.Count){ Legend = r.Word })
{Title = "Top 5 words" };
var image = await chart;
```

该的 await 关键字以非正统的方式用于制作从昂贵的胁迫 Google.Linq.Charts。馅饼切成需要一个网络往返明确的图像。

这个例子仅仅触及 LINQ 的表面。它提供了顺序运算符，如库中选择，其中，GROUPBY，...改造收藏，并在查询解析的形式，允许程序员在更高的抽象水平写对集合转换提供了语法糖。

要真正了解 LINQ 的力量，让我们退后一步，并探讨其起源和数学基础。别担心，你只需要高中层次的数学知识。

要真正了解 LINQ 的力量，让我们退后一步，并探讨其起源和数学基础。别担心，你只需要高中层次的数学知识。

数据为中心的解读

关系代数，形成正式的基础 SQL，定义了数套值 $\{\Sigma\}$ ，常数和构造，如空集 $\emptyset \in \{\Sigma\}$ ；注入的值到一个单集合 $\{_ \} \in \Sigma \rightarrow \{\Sigma\}$ ；与工会的两套进入一个新的组合

集 $\{\Sigma \times X\} \rightarrow \{\Sigma\}$; 也有一些关系运算符如投影, 它适用于转型的每个元素在一组 π
 $(\Sigma \rightarrow \Lambda) \times \{\Sigma\} \rightarrow \{\Lambda\}$; 选择, 只选择在一组满足这些元素鉴于财产; 笛卡儿积, 这对
 一对的集合 X 中的所有元素 $\{\Sigma\} \times \{\Lambda\} \rightarrow \{\Sigma \wedge \Lambda\}$; 和跨应用, 其产生在第一组@每个元
 素的次级组值 $(\Sigma \rightarrow \{\Lambda\}) \times \{\Sigma\} \rightarrow \{\Lambda\}$ 。 $\bigcup \in \sigma \in (\Sigma \rightarrow B) \times \{\Sigma\} \rightarrow \{\Sigma\} \in \in$

图 2 描述了使用云来表示组值的关系代数运算符。一个 SQL 编译器将在表达的查询
 熟悉的 SELECT-FROM-WHERE 语法为关系代数表达式; 优化查询它适用代数的法律,
 如选择的分布: $\sigma(p, \sigma(q, xs)) = \sigma(x \mapsto p(x) \wedge q(x), xs)$ 然后将这些逻辑表达式成由
 RDBMS 执行的物理查询计划。

例如, SQL 查询 SELECT 名字从朋友 WHERE 喜欢(朋友, 寿司)被翻译成关系代
 数表达式 $\pi(F \Rightarrow f.Name, (\sigma(F \Rightarrow \text{喜欢}(F, \text{寿司}), \text{朋友})))$ 。为了加快执行查询时,
 可以 RDBMS 使用索引以便快速查找朋友谁喜欢寿司, 而不是在整个集合做线性扫描。

操作符@的申请十字架是特别强大, 因为它允许您在哪里产生从第一集合中的每个
 值的第二收集和扁平化的结果到一个单一的集合@相关子查询 $(F, \{A, \dots, Z\}) = F$
 (一) $\bigcup \dots \bigcup F(z)$ 的所有其它关系运算符可以在交叉应用算子来定义:

```
xs X ys = @ (x => pi (y => (x,y), ys), xs)
pi (f, xs) = @ (x => {f(x)}, xs)
sigma (p, xs) = @ (x => p(x) ? {x} : [], xs) 1
```

作为一个程序员, 你可以很容易地想象写了一个简单的实现跨应用: 你只是遍历输
 入集中的项目, 运用给定的函数, 结果积聚成一个结果集。这样的实施, 但是, 不会需
 要它的参数是作为集合 $\{\Sigma\}$; 任何我们可以遍历如列表, 数组或哈希表就足够了。同样,
 没有理由在所有的关系代数操作应限制值集 $\{\Sigma\}$ 的。它们可以基于其它类型的集合, 以
 及来实现。

令人惊讶的是, 还有没有理由的操作传递到 π , σ , 并@应仅限于具体的功能 $\Sigma \rightarrow$
 Λ 。事实上, 你可以使用一个函数, 从中确定要执行哪些计算任何声明。例如, 在语言
 如 JavaScript, 你可以简单地传递一个字符串, 然后使用 eval 把它变成可执行代码。

你所寻找的是底层接口的关系代数工具。只要存在用于集合类型构造中号 $\langle \Sigma \rangle$ 提供
 满足组类似状代数性质为 $\{\Sigma\}$ 的操作, 和用于计算一个类型构造 $\Sigma \rightarrow \Lambda$ 满足相似功能样特
 性如 $\Sigma \rightarrow \Lambda$, 则可以概括关系代数以下集合运算符, 并仍然可以通过脱糖查询语法来写
 了这些集合的 SQL 查询:

$$\begin{aligned} & \emptyset \in M<\Sigma> \\ & \{ _ \} \in \Sigma \rightarrow M<\Sigma> \\ & \cup \in M<\Sigma> \times M<\Sigma> \rightarrow M<\Sigma> \\ & \otimes \in (\Sigma \rightarrow M<\Lambda>) \times M<\Sigma> \rightarrow M<\Lambda> \end{aligned}$$

对于程序员来说这仅仅是接口和实现分离; 数学家调用生成的结构单子, 而不是查询他们讲的内涵。

一个面向对象的语言, 如 C# 使用规范的接口集合的 `IEnumerable<T>` 的抽象集合 M 型 $<T>$ 和使用委托 `Func` 键 $<\Sigma, \Lambda>$ 代表计算的特定实例 $\Sigma \rightarrow \Lambda$ 。通过这样做, 从关系代数为如在所限定的 LINQ 标准查询操作符识别运营 `Linq.Enumerable` 类, 如图图 3。

另外, 您也可以使用的 `IQueryable<T>` 接口表示集合中号 $<T>$ 和表达式树表达 $<Func$ 键 $<\Sigma, \Lambda>$ 代表计算 $\Sigma \rightarrow \Lambda$ 。在这种情况下, 你承认作为定义的关系代数运营商为 LINQ 标准查询运算符 `Linq.Queryable` 类。对待代码使用数据的能力态射, 或者使用 C# 的情况下表达代码类型和 `lambda` 表达式文字, 是一个基本的功能, 它允许程序本身来操作, 优化, 并在运行时转换查询。

相反, SQL 语法, C# 语言定义表单中的 XQuery 般的推导从-那里选。前面的 SQL 查询的例子如下:

```
from friend in friends where friend.
Likes(Sushi) select friend.Name
```

正如在 SQL 中, 解析由编译器进入下面 LINQ 查询代数翻译:

```
friends.Where(friend=>friend.
Likes(Sushi)).Select(friend=>friend.Name)
```

根据的重载凡和选择的 `lambda` 表达式将被解释为代码或数据。的简化实施的 `IQueryable` 将在后面讨论。

正如已经显示的, 单子和他们在实际编程语言如 LINQ 化身只是通过想象关系代数实现接口关系代数的推广。因此 LINQ 背后的理念和思路应该是深深熟悉的两个人数据库和程序员。

理论到实践

不像哈斯克尔, 这已纳入单子和单子内涵有原则的方式, C# 的类型系统是不是单子操作符的数学签名足够的表现力。相反, 查询内涵的翻译纯粹基于模式的方式进行定义。在第一遍时, 编译器一味 `desugars` 解析, 使用一组固定的规则, 进入正规 C# 的方法调用, 然后依赖于标准型为主的重载解析到查询操作结合自己的实际实现。

例如, 该方法美孚选择 (酒吧来源, `Func` 键 $<\text{巴兹}, \text{Qux}>$ 选择器), 这不涉及任何

集合类型，将被绑定为翻译理解的结果

```
var foo = from baz in bar select qux
```

进入脱糖表达

```
var foo = bar.Select(baz⇒qux)
```

这种技术在未来呈现的示例广泛使用。

LINQ 和其一元基础之间的另一个区别是一个更大的类查询操作符，包括分组和聚集，这是更类似 SQL 的。有趣的是，内涵在 C# 中的包容，这是在 Haskell 灵感单子和 list 解析，递归地激发了哈斯克尔增加对分组和聚合到它的内涵支持。

自定义查询供应商

雅虎气象服务 (<http://developer.yahoo.com/weather/>) 允许天气预报查询，对于一个给定的位置，使用公制或英制单位的温度。这个简单的服务是要说明一个非标准实施 LINQ 查询运算符是完全专门为这个特定目标，这将允许一个很好的方式只强类型形式的查询

```
var request = Yahoo.WeatherService().  
Where(forecast⇒forecast.City == city).  
Where(forecast⇒forecast.  
Temperature.In.units);  
var response = await request;
```

或等价使用查询解析

```
var request =  
from forecast in Yahoo.WeatherService()  
where forecast.City == city  
where forecastTemperature.In.units  
select forecast;  
var response = await request;
```

运营商的实施从查询中提取城市和温度单位，并用它们来创建一个 REST 调用 (<http://weather.yahooapis.com/forecastrss?w=woeid&u=unit>) 到雅虎服务作为使用的结果在的 await 关键字来明确强制要求到响应。

在这种风格的自定义 LINQ 提供的技术诀窍是项目目标的查询语言，在这种情况下，雅虎的天气要求（一）城市和服务的能力（B）的单位，变成一种层次状态机引导中的“流畅”的风格用户（以及由智能感知支持）通过可能的选择，他们可以作出（图 4）。

在该状态机中的每个过渡我们收集的查询-在这种情况下，特别是城市和温度单位感兴趣的各个部分。原则上，这个城市并不真的需要是第一位的，但它可能是更自然的曲线图，让任一类型的，其中首先被指定的条款，但其限制是双方在这里的必备条款。我

离开这个限制在状态机的提升作为一个练习留给读者。

请注意，没有任何类型的天气，WeatherInCity 或 WeatherIn-CityInUnits 实现任何标准集合接口。相反，他们代表了将提交给雅虎的 Web 服务，为您不需要定义一个明确的容器类型要求的计算阶段。什么也很多人惊奇的是，无论是两凡方法实际计算布尔谓词。更奇怪的是，这三个事件的范围变量的预测在查询中有不同的类型。

在天气类定义挑选查询指定的城市，将其传递到一个方法 WeatherInCity，这是在基于类型的状态机的下一个状态：

```
WeatherInCity Where(Weather source,
Func<CityPicker,string> city)
{
    return new WeatherInCity{ City =
        city(new CityPicker()) };
}
```

在“上游”，在其中的方法是一个函数，类型的值 CityPicker，其中有一个返回幻象类的一个属性域只存在，以促进智能感知，其平等检查立即返回传递给相等操作字符串：

```
class CityPicker { City city; }
class City
{
    static string operator == (City c,
        string s) { return s; }
}
```

由于这个结果，主叫雅虎。天气（）。如果（预测⇒预报==“西雅图”），真的只是创建一个新的令人费解的方式 WeatherInCity {市=“西雅图”}例如使用其中的方法，这并不需要一个布尔谓词，并返回一个相等运算符字符串。

可以使用在相同的诡计 WeatherInCityInUnits 凡（Func 键<UnitPicker, 单元>谓词），以便在调用凡（预测⇒forecast.Temperature.In.Celsius）上先前的滤波器的结果创建的新的实例 WeatherInCityInUnits {市=“西雅图”，单位=单位。摄氏}。此处所使用的技术不仅用于限定 LINQ 运算符的自定义实现是有用的，但也可以利用在一般建筑物流利接口。

由于雅虎服务需要城市作为一个 WOEID（在地球上的 ID），我们需要做的引擎盖下两个服务电话以获取天气预报。第一个服务调用通过检索请求的城市的 WOEID [http://where.yahooapis.com/v1/places.q\(city\)?appid=XXXX](http://where.yahooapis.com/v1/places.q(city)?appid=XXXX)。如果成功返回，那么第二个调用来检索该位置的天气预报。对 Web 服务器的调用是异步执行，都返回一个任务<T>（在 Java 中你会使用的 java.util.concurrent。未来<T>表示异步操作的结果）。

因为我们可以考虑一个任务<T>作为一种集合，它包含（最多）一个元素，它支持 LINQ 查询运营商，我们有海龟一路下滑；为实现 LINQ 天气使用 LINQ 执行规定任务<T>（参见图 5）。

尽管这是一个非常小的和有限的例子，它清楚地说明了许多用于创建真实世界的 LINQ 提供程序，如 LINQ 到对象，LINQ 到 SharePoint，LINQ 到 Active Directory，LINQ 到 Twitter，LINQ 到 Netflix 和许多技巧更多。

通用查询供应商

天气服务查询提供的例子的结构为内部 DSL。虽然这提供了最大的静态类型的用户体验，它可以让小空间复用实际执行的供应商。它是定制的自顶至底的特定目标。在光谱的另一端，我们可以创建一个记录的完整查询完全通用的查询提供“按原样”使用的元编程魔术一点点。

在 C# 中的 lambda 表达式，如点 $\bar{x} \Rightarrow X > 4711$ 可以转换成任何一个代表说，类型的函数功能<INT, INT> -或成类型的表达式树表达<Func 键<INT, INT >>，其中对待 Lambda 表达式作为数据的代码。在 Lisp 或计划，应当使用语法引用对待代码的数据。在与由上下文期望的类型组合的 C# lambda 表达式提供一种基于类型的引用机制。

类可查询实现 LINQ 标准查询运算符采取表达式树作为参数，并返回一个表达如图他们自我的表现，非常像宏录制图 6。

例如，给定值 XS 型可查询<int>的，呼叫 xs.Select（点 $\bar{x} \Rightarrow X > 4711$ ）引起 lambda 表达式转换成表达式树（以粗体显示），然后返回表示表达式树调用本身 xs.Select（点 $\bar{x} \Rightarrow X > 4711$ ）。现在，它是由特定的查询提供程序（如 LINQ to SQL 中，实体框架，LINQ 到 HPC）翻译结果表达式树，并把它编译成目标查询语言。

该 IQueryable 的基础的附带的.NET 框架使用上面所示相同的方案简化示例代码，但它是基于接口的实现，因此它依赖于第二接口 IQueryProvider 用于创建实例提供一个工厂的 IQueryable。

一个通用的查询提供的好处是，你可以提供一般的服务，如查询优化，它们实现重写规则，如 xs.Union（YS）。凡（P）= xs.Where（P）。联盟（ys.Where（P）），可以在许多 LINQ 提供重复使用。

LINQ 友好的 API

所有的例子都涉及执行特定的 LINQ 提供程序。LINQ 的正交方面是利用特定的

LINQ 的实现，往往 LINQ 到对象的 API。例如，LINQ 到 XML 是用于处理已专门 LINQ 的想法，这样就无需对 DSL 等的 XQuery 或 XPath 查询和转换 XML 设计的 XML 文档的 API。

该谷歌图表 API 是一个 Web 服务，它可以让你动态地创建寻找有吸引力，图表，使用简单的 URI（统一资源标识符）方案。对于谷歌图表 URI 语法，但是，是不是非常友好的序列。例如，URI 的早期样品饼图看起来是这样的：

问题是，本说明书中为标签（叶绿素=在|的| A |即|是）和用于所述数据集的规范（CHD = T: 21,12,7,7,6）的图表中给出两个独立的集合。另一方面，以使用查询生成饼图，想要对的单个集合，同时指定的值，并为每个切片作为标签从 w 的 TOP5 选择新切片（瓦特数）{图例= R 。 字}。

换句话说，使谷歌图表 API 序列友好，你必须转 对集合中号 $\langle S \times T \text{ 对} \rangle$ 成一对藏品中号 $\langle S \rangle \times M \langle T \rangle$ 。功能的程序员立刻认识到这一点作为函数的一个实例解压 $\in (R \rightarrow S \text{ 个 } R \rightarrow T \text{ 的 } \times M \text{ 的 } \langle R \rangle) \rightarrow M \langle S \rangle \times M \langle T \rangle$ 。解压缩可以转换包含如图切片成通过格式化使用由图表服务规定的隔板的各种集合由谷歌图表 API 所需 URI 的格式的序列的图。

结论

大数据不只是大小。它也是对数据的多样性，无论是在数据模型而言（主键/外键与键/值），以及消费模式（拉与推），许多其它尺寸之间。本文认为，LINQ 是大数据有前途的基础。LINQ 既是关系代数的概括和范畴论，特别是单子有着很深的渊源。

随着 LINQ，查询在 C#，Visual Basic 中表示，或 JavaScript 可以被捕获或者作为代码或表达式树。然后或者表示可以改写和优化，并随后在运行时编译。我们还展示了如何实现可在存储器中，并且 SQL 和数据库 CoSQL 运行自定义 LINQ 提供程序，我们已经在 Web 服务提出了 LINQ 友好的 API。另外，也可以以露出流数据，以便实现 LINQ 标准查询操作符，产生一个单一的抽象，它允许开发人员查询过大数据的所有三个特点。

致 谢

非常感谢云可编程队员萨瓦斯 Parastatidis，格特·布商行，亚伦 Lahman，巴特迪斯和韦斯·戴尔在建设 LINQ 和 coSQL 的所有不同的基础设施和原型，所有的辛勤工作；勒内 BOUW，布赖恩·贝克曼和特里 Coatta 帮助改善这一文章的可读性；戴夫·坎贝尔和萨蒂亚·纳德拉提供必要的帮助写出来。

附录 II 毕业设计任务书

毕业设计任务书

基于 PHP 的毕业论文选题系统

一、毕业设计目的

毕业设计是教学过程的最后阶段采用的一种总结性的实践教学环节。通过毕业设计，学生可以综合应用所学的各种理论知识和技能，进行全面、系统、严格的技术及基本能力的练习。大学生参加毕业设计的目的，主要有两个方面；一是对学生的知识相能力进行一次全面的考核。二是对学生进行科学研究基本功的训练，培养学生综合运用所学知识独立地分析问题和解决问题的能力，为以后撰写专业学术论文打下良好的基础。

撰写毕业论文的过程，同时也是专业知识的学习过程，而且是更生动、更切实、更深入的专业知识的学习。首先，撰写论文是结合科研课题，把学过的专业知识运用于实际，在理论和实际结合过程中进一步消化、加深和巩固所学的专业知识，并把所学的专业知识转化为分析和解决问题的能力。其次，在搜集材料、调查研究、接触实际的过程中，既可以印证学过的书本知识，又可以学到许多课堂和书本里学不到的活生生的新知识。此外，学生在毕业论文写作过程中，对所学专业的某一侧面和专题作了较为深入的研究，会培养学习的志趣，这对于他们今后确定具体的专业方向，增强攀登某一领域科学高峰的信心大有裨益。

二、主要内容

本系统可以让毕业班学生进行注册登录，注册时候存在前端以及后端的数据验证操作，防止信息填写不准确，另外采用了第三方的登录操作无需注册即可实现登录操作并且常规登录也存在数据前端和后端的验证操作，个人中心会首先需要完成资料（基本信息，头像设置，密码重置等操作）补充并根据填写资料自动匹配所在班级的各种信息，倘若没有补充资料则不可进行额外操作，核心部分是毕业班学生可以在线直接选课，查看选课结果，撤销选课，选课限制，查看班级选课情况，并且查看其他班级及院系的选课安排及结果；毕业班教师进行注册登录，注册时候存在前端以及后端的数据验证操作，防止信息填写不准确，这里没有采用第三方的登录操作并且常规登录存在数据前端和后端的验证操作，个人中心同样需要首先完成资料补充并根据填写资料自动匹配所在班级的各种信息（基本信息，头像设置，密码重置等操作），倘若没有完善资料则不可进行

额外操作，核心部分是指导教师可以在线发布选课，查看选课进度，结束选课，选课结果查看，通知消息，拒绝选课，接受选课，重置选课以及查看毕业班级选课情况；

三、重点研究问题

如何通过网络访问选题系统？

使用何种方法获取教师系统的课题信息？

如何设计数据库结构？

怎样将课表界面设计的更漂亮

四、主要技术指标或主要参数

个人电脑操作系统：Windows 7

开发环境：phpstudy

数据库:MySQL

五、基本要求

服务器获得请求及响应：能够得到客户端上传的数据，能够对上传参数进行服务器本地的处理，得到数据后会返回给客户端。

信息本地存储及使用：根据下载的课表和课程信息，进行本地化的存储和使用，能够给用户一个比较直观和智能化的体验。

六、其它（包括选题来源）

选题来源：实践。

时间安排：

第3周：课题调研、查阅检索相关的文献以及资料，研究课题的可行性及项目具体实现

第4周：收集资料，完成毕业设计开题报告，制定较为健全的系统模块和功能，编制工作计划表。

第5~7周：准备清考暂停毕业设计进度。

第8~12周：正确的搭建框架，与服务器进行正常交互，并实现软件的各个功能。

第13周：调试程序，最后阶段的补充，对错误的更改。

第14周：提交成果，完成毕业论文的编写、修改，为答辩做好准备。

指导教师：

年 月 日

附录 III 开题报告

华北水利水电大学本科毕业生毕业设计（论文）开题报告

学生姓名	王文帅	学号	201316602	专业	网络工程
题目名称	基于 PHP 的毕业论文选题系统				
研究或设计 概述 (500 字左右)	<p>目前几乎所有的高等院校都有一个选修课的选题系统，但是大部分的高校却没有一套完整的毕业论文选题系统，原因有两个方面：首先选修课系统使用人数比较多，使用的频率比较高，选题简单，并且模式固定，而对于毕业论文的选题系统来说，每一个学年只会使用一次，并且选题的随机性太大，高校不会愿意出资金开发一套这样的论文的选题系统。所以就目前大多数的院校选题来看，很多都是人工收集整理数据，使用纸质操作统计进行选课的，很少采用互联网的方式进行论文选题的，这样带来的后果看到的不是科技在进步，而是重新退步到传统的手动操作了，造成了资源的大量浪费，人力和物力的极度损耗，最后所得到的结果却很普通，完全是事倍功半。针对该类情况开发一套关于毕业生进行选课的系统实在很有必要，节约当下的各种资源，保证数据的真实可靠性以及信息的及时反馈性。考虑到容错性以及扩展性，实现资源信息的共享，将学生、教师、课程等进行分析设计，同步工作和提高效率。</p>				
主要内容	<p>本系统主要结合现在大学生实际需求，为了更好的满足学生使用的需求以及更好的对毕业设计选题进行数字化管理，本系统的前台应用主要分为三个不同的角色，教师角色，学生角色和非游客角色，不同的角色具备不同的操作权限，教师角色的主要功能就是发布课程已经查看课程具体详情，对于学生角色最多的的选择课题并且查看课题状态，非游客的具体就是一些简单的查看操作；而对于后天管理员的权限基本都是一些查看操作，进行修改操作的不太多。</p>				

<p>主要参考文献（不少于10篇）</p>	<p>[1] 高洛峰. 细说 PHP 第二版[D]. LAMP 兄弟连 2011</p> <p>[2] 潘凯华, 邹天思. PHP 开发实战宝典[M]. 北京: 清华大学出版社, 2010</p> <p>[3] 陈湘扬, 陈国益. PHP5+MySQL 网页系统开发设计[M]. 北京: 电子工业出版社, 2007</p> <p>[4] 邹天思, 孙鹏. PHP 从入门到精通[M]. 北京: 清华大学出版社, 2008</p> <p>[5] 宋尚平 李兴保. PHP 模板引擎 Smarty 的安装配置及应用实现[M]. 2007</p> <p>[6] 李峰, 晁阳. JavaScript 开发技术详解[M]. 北京:清华大学出版社, 2009</p> <p>[7] W. Jason Gilmore. PHP 与 MySQL5 程序设计[M]. 人民邮电出版社, 2007</p> <p>[8] 徐鹏. 基于 Web 数据库的教务管理系统的设计与实现[J]. 价值工程. 2011 (03)</p> <p>[9] 崔洋. MySQL 数据库应用从入门到精通[J]. 中国铁道出版社. 2013</p> <p>[10] 王雨竹. MySQL 入门经典. 机械工业出版社[J]. 2013</p> <p>[11] 王志刚. MySQL 高效编程. 人民邮电出版社[J]. 2012</p> <p>[12] 钱雪忠. MySQL 数据库技术与实验指导[M]. 清华大学出版社. 2012</p> <p>[13] 蔡旸. 基于 MVC 设计模式的协同设计系统的研究与实现[J] 西南交通大学, 20050601</p> <p>-----</p>
-----------------------	---

采取的主要技术路线或方法	<p>1.后台使用 php 作为主要的开发语言，使用了 mvc 设计模式，采用了国内目前比较流行的 Thinkphp 开源框架，服务器的选择上使用了阿里云的服务器以及 centos 后台操作系统。</p> <p>2.前端的页面的设计上使用了 bootstrap 框架作为基本的前台开发，包含一系列的 ajax 提交和 js 功能函数，后台的页面上使用了基于 bootstrap 的一款后台设计框架进行开发。</p>
时间安排	<p>第 3 周：课题调研、查阅检索相关的文献以及资料，研究课题的可行性及项目具体实现；</p> <p>第 4 周：收集资料，完成毕业设计开题报告，制定较为健全的系统模块和功能；</p> <p>第 5-9 周：设计系统需要的数据表进行数据库的建立；</p> <p>第 10 周：为项目搭建总体的框架；</p> <p>第 11~12 周：所有代码的实现；</p> <p>第 13 周：调试程序，最后阶段的补充，对错误的更改。</p> <p>第 14 周：答辩、整理文档资料，根据答辩情况修改设计和设计说明书，装订提交归档；</p>
指导教师意见	<div style="text-align: right;"> 签 名： 年 月 日 </div>
备注	