

摘 要

随着我国各高校的不断扩招，2015 年全国高校毕业生总人数达到 749 万，比 2014 年增加 22 万，再创历史新高。毕业生人数的逐年攀升，不仅给学生个人的就业带来了很大困难，也给各个院校在对毕业生就业方面的管理工作带来了不小的压力。对于仍然使用传统手工方式进行管理的沈阳理工大学信息科学与工程学院（以下简称我院）来说，无论是在数据收集还是在数据统计上都占用了教职大量的时间和精力。因此，打造数字化、网络化、智能化就业管理系统，建立学生、学校用人单位之间沟通的桥梁显得尤为重要。

本设计在对我院毕业生管理工作的业务流程进行调查和分析之后，以面向对象的设计思路为导线，后端采用当今较为流行的基于 JavaEE 平台的 SpringMVC 框架和 MySQL 数据库，前端采用基于 jQuery 的 EasyUI 框架设计和实现了针对我院实际需求的学生就业信息管理系统。该系统大大简化了就业管理工作，免除了此项工作中的数据汇总、数据统计等大部分的手工操作，并且具有易于操作和维护的特点，基本解决了我院在毕业生管理工作中遇到的各项问题。

关键词：毕业生；就业管理系统；JavaEE；SpringMVC

Abstract

With the increasing number of undergraduates in our country, the number of graduates will reach 7,490,000 in 2015, which is 220,000 higher than that of 2014 and will become a new record. The increase of the number of graduates has brought a lot of pressure to not only students, but also the management work of the process of individual students' job hunting. As the Information Science and Engineering College of Shenyang Ligong University still using traditional manual approach to manage this process, the collection and statistics of data occupied a lot of time of the staff in our university. So, it is very important to create a digital, network, intelligence employment management system and a bridge between students and employers.

In this design, after the investigation and analysis of business processes and the method of employment management of students, I successfully created a Student Employment Information Management System which based on object-oriented design idea, relatively popular JavaEE platform. The Back-End of this system use SpringMVC Framework and MySQL Database, and the Front-End of this system use jQuery Framework and EasyUI Framework. This system can save a lot of time in managing the process of the job hunting of students, and make it possible to get rid of traditional manual of summary and statistics of those data. Furthermore, this system has the feature in easy operation and maintenance. Finally, to some extent, it has solved all sorts of problem in managing the process of job hunting of students in our college.

Keywords: Graduates, Employment Management System, JavaEE, SpringMVC

目 录

1 绪论.....	1
1.1 研究背景和现状	1
1.2 研究目的和意义	1
1.3 研究方法和内容	2
1.3.1 研究方法	2
1.3.2 研究内容	2
2 系统关键技术.....	4
2.1 C/S 结构和 B/S 结构	4
2.2 JavaEE 技术	4
2.3 SpringMVC 框架	6
2.4 MySQL 数据库.....	8
2.5 jQuery 框架和 EasyUI 框架.....	8
2.5.1 jQuery 框架	8
2.5.2 EasyUI 框架	9
2.6 AJAX 技术.....	10
2.7 系统开发方式.....	10
2.7.1 瀑布模型	11
2.7.2 极限编程	12
2.7.3 敏捷开发	12
3 就业管理系统的需求分析.....	14
3.1 系统背景和总体目标.....	14
3.2 功能性需求	14
3.2.1 用例图	14
3.2.2 业务流程图	19
3.3 非功能性需求	20
3.4 系统可行性分析	21
3.4.1 技术可行性分析	21
3.4.2 社会可行性分析	21
3.4.3 经济可行性分析	22
4 就业管理系统概要设计.....	23

4.1 项目的搭建	23
4.1.1 版本控制	23
4.1.2 依赖管理	24
4.2 项目架构设计	25
4.3 系统模块设计	35
4.4 数据库设计	35
4.4.1 数据库模型设计	35
4.4.2 数据表设计	38
5 就业管理系统详细设计	41
5.1 界面设计	41
5.2 学生信息管理模块设计	44
5.2.1 前台部分	45
5.2.2 后台部分	51
5.3 字段管理模块设计	55
5.4 统计报表模块设计	56
5.5 毕业设计指导教师管理模块设计	58
5.6 系统设置模块设计	59
5.7 用户权限模块设计	59
5.8 异常处理机制	62
6 系统的测试与部署	64
6.1 系统的测试	64
6.1.1 测试的目的	64
6.1.2 测试的方法	64
6.1.3 该系统中采用的测试	65
6.2 系统的部署	69
结论	72
致谢	73
参考文献	74
附录 A 英文原文	75
附录 B 中文译文	83
附录 C 程序代码	91

1 绪论

1.1 研究背景和现状

我国目前的高校毕业生就业体制是学生和企业之间的双向选择，加上我国大中型企业也进入了转型期，中小民营企业如雨后春笋，机械化也逐渐代替了手工操作，人员的吸纳能力也随之逐渐下降，毕业生的就业形势日趋严峻。另一方面，近年来，我国的高等教育已获得巨大的进步，取得了令人瞩目的成绩，为社会主义现代化建设培养了大批的专业人才。随着党中央对高等教育越来越重视，大学逐渐扩招、师资力量大幅度增长，而高校在管理各方面的发展又没有与其同步，传统的就业管理方式，在就业形势竞争激烈的环境下，已经不能满足社会的需求。人工化管理的缺陷造成数据在采集、处理、存储、管理、检索和传输上无法跟上实际需求。

如何更好地为毕业生提供更加良好的管理，是越来越多的高校都希望解决的问题。我国高校就业管理目前还处于起步阶段，与发达国家相比，我们还有不小的距离，欧美等发达国家，高等教育发展的历史也有几百年了，加之计算机和网络技术方面一直处于领先的地位，使之在就业管理自动化方面有了很大的成就，具有建设时间长、技术领先、设备先进、系统成熟规范等优点，使得毕业生就业管理系统得到了充分的运用。

对比国内外高校的学生就业信息系统，在发展时间上存在着十几年的差距，虽然目前国内的就业管理系统也日益增多，但是功能仍然不够完善，不能适合所有的高校，各个高校的就业管理模式也较大差异，所以系统在学校和学校之间的适用性仍然较差。

1.2 研究目的和意义

本校的信息科学与工程学院专业繁多，就业流程复杂，数据量巨大，在管理和配合学校就业管理工作上要花费大量的人力、财力和时间，面对这一难题，根据近年来本院的就业管理需要，本文拟设计与实现一套就业管理系统来完成该学校的毕业生就业工作，采用现代计算机信息技术手段实现就业管理的科学化、网络化、高效率化，建设一个智能可行的学生就业管理系统。

对于学校毕业生的就业情况，历年来积累的数据对学校来说是一份巨大的财富，学校可以对这些数据进行分析利用，得出市场对人才有何需求、哪些方面的人才比较紧缺、学生的就业率如何，学生的就业满意度如何，哪些企业更能让学生满意等信息。这些信

息如果能够被完整无损地保存在就业管理系统当中的话,也就间接地帮助学校完善自我管理模式,无论是在教学方面,还是在学校长期发展角度上来看,都是很有益处的。因此,本课题的开发具有重要的意义。

1.3 研究方法和内容

1.3.1 研究方法

1、文献研究

- 1) 查阅大量期刊、文献和网络资料。
- 2) 参考已有就业管理系统的相关案例与实践。
- 3) 了解基于 Web 开发技术的现状,明确就业管理信息系统的设计开发所需要的关键技术、实现途径和功能模块。
- 4) 找出不足并力求避免或者改进。

2、需求调研

- 1) 和校领导以及从事学生就业信息管理的相关教职人员召开座谈会。
- 2) 掌握就业管理信息系统实施的具体软硬件环境及使用的真实情况。
- 3) 收集整理不同用户在使用过程中的反馈意见。

1.3.2 研究内容

本系统的目标是要建立一个基于 B/S 结构的,实现网上访问的学生就业信息管理系统。本论文的主要内容有以下几个方面:

- 1) 对系统开发过程中使用到的主要技术进行分析研究和选择。
- 2) 研究软件开发的流程,并按照流程进行实践。
- 3) 提出了开发本系统的意义和目标,根据软件工程开发过程,从多方面分析系统需求,确立系统实现功能,对系统进行总体设计。
- 4) 研究软件开发环境、测试环境、版本控制、项目管理、版本管理环境的搭建和使用。
- 5) 根据需求分析对系统的数据库进行逻辑设计和物理结构设计。
- 6) 对本系统的功能模块进行设计,确定了本系统涵盖的主要内容。
- 7) 利用所选技术进行实际的程序开发。

- 8) 对系统进行测试与分析，分析存在的问题及改进措施。
- 9) 对系统进行发布和部署。

2 系统关键技术

2.1 C/S 结构和 B/S 结构

随着计算机技术的高速发展，基于 Internet 技术的应用也随之兴起，其中尤为典型的就属 C/S 结构（Client/Server 的简称，客户机/服务器模式）了。该结构在上个世纪末和本世纪初得到了大量应用，但是，由于该结构的每台客户机都需要安装相应的客户端程序，分布功能弱且兼容性差，不能实现快速部署安装和配置，因此缺少通用性，具有较大的局限性，要求具有一定专业水准的技术人员去完成。近年来，随着网络技术不断发展，尤其是基于 Web 的信息发布和检索技术、Java 计算技术以及网络分布式对象技术的飞速发展，导致了很多应用系统的体系结构从 C/S 结构向更加灵活的多级分布结构演变，使得软件系统的网络体系结构跨入一个新阶段，即 B/S 体系结构（Browser/Server 的简称，浏览器/服务器模式）。基于 Web 的 B/S 方式其实也是一种客户机/服务器方式，只不过它的客户端是浏览器，客户无需只要有一个浏览器就无需再安装其他软件，易于升级与维护。

以目前的技术看，建立 B/S 结构的网络应用，相对易于把握、成本也是较低的。它是一次性到位的开发，能实现不同的人员，从不同的地点，以不同的接入方式访问和操作共同的数据库；它能有效地保护数据平台和管理访问权限，服务器数据库也很安全。特别是在 Java 这样的跨平台语言出现之后，B/S 架构管理软件更是方便、速度快、效果优。

2.2 JavaEE 技术

目前，Java 平台有三个版本，它们分别是适用于小型设备和智能卡的 JavaME（Java Platform Micro Edition，Java 微型版），适用于桌面系统的 JavaSE（Java Platform Micro Edition，Java 标准版），以及适用于企业级应用的 JavaEE（Java Platform Enterprise Edition，Java 企业版）。

JavaEE 是一种利用 Java 平台来简化企业级软件开发的集成化解决方案。其核心 JavaSE，它不仅巩固了标准版中的许多优点，例如“一次编写、处处运行”的特性，能够应用于网络应用中的保护安全的模型，和能够对数据库方便读写的 JDBC 技术，同时还提供了对 JavaBean、Servlet、JSP（Java Server Pages）以及 XML 技术的全面支持。其最终目的就是成为一个能够使企业开发者大幅缩短投放市场时间的体系结构。JavaEE

体系结构提供中间层集成框架用来满足无需太多费用而又需要高可用性、高可靠性以及可扩展性的应用的需求。通过提供统一的开发平台,JavaEE 降低了开发多层应用的费用和复杂性,同时提供对现有应用程序集成强有力支持,有良好的向导支持打包和部署应用,添加目录支持,增强了安全机制,提高了性能。

JavaEE 使用多层的分布式应用模型,应用逻辑按功能划分为组件,各个应用组件根据他们所在的层分布在不同的机器上。事实上,Sun 公司(目前被 Oracle 公司收购)设计 JavaEE 的初衷正是为了解决两层模式(C/S 结构)的弊端,在传统模式中,客户端担当了过多的角色而显得臃肿,在这种模式中,第一次部署的时候比较容易,但难于升级或改进,可伸展性也不理想,而且经常基于某种专有的协议,通常是某种数据库协议。它使得重用业务逻辑和界面逻辑非常困难。现在 JavaEE 的多层企业级应用模型将两层化模型中的不同层面切分成许多层。一个多层化应用能够为不同的每种服务提供一个独立的层,以下是 JavaEE 典型的四层结构(如图 2.1 所示):

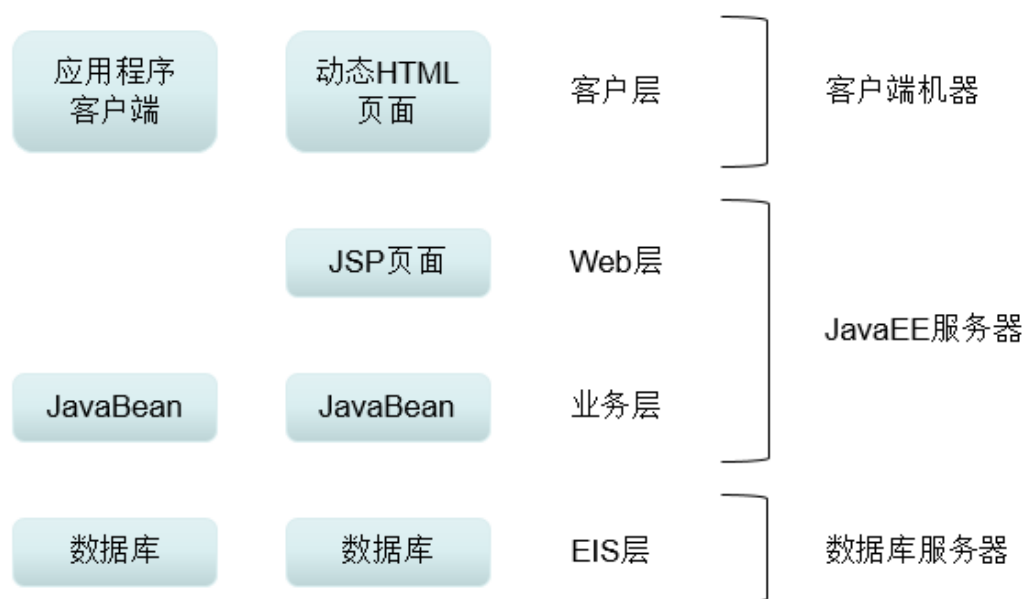


图 2.1 JavaEE 体系结构

- 1) 客户端组件: JavaEE 应用可以基于 Web,也可以是不依赖于 Web。
- 2) Web 层组件: 可以是 JSP 页面或 Servlet。Web 层处理用户输入,并把输入发送给业务层中的 JavaBean 来进行处理。
- 3) 业务层组件: 采用业务层逻辑来满足特殊领域的需要,由业务层中的 JavaBean

进行处理。

- 4) 企业信息系统层 (EIS): 包括企业基础建设系统, 例如 ERP 系统、大型机事务处理系统、数据库系统, 和其它的遗留信息系统。

2.3 SpringMVC 框架

SpringMVC 是非常优秀的 MVC 框架, 现在有越来越多优秀的团队开始选用 SpringMVC 框架, SpringMVC 结构简单, 但不失灵活, 性能也很优秀。它把控制器、模拟器对象、分派器以及处理对象的角色进行分离, 这种分离让他们更易于进行开发, 方便和 View 框架无缝集成, 运用 IoC 测试更加方便。框架是典型的 MVC 框架, 它是一个纯正的 Servlet 系统。

SpringMVC 包含以下特点:

- 1) 清晰的角色划分: Controller (控制器)、Validator (数据验证器)、Form Object (表单)、Model Object (模型)、Dispatcher Servlet (请求分发器)、Handler Mapping (请求处理映射器)、View Resolver (视图解析器) 等等。任何一个角色都由专门的对象来负责;
- 2) 方便的配置方式: 不仅可以在配置文件中配置, 也可以利用注解在代码间进行配置;
- 3) 非侵入的控制器: 你可以将任何基本的 Java 类作为控制器来使用;
- 4) 可定制的请求映射器和视图解析器: 从最简单的 URI 映射, 到复杂的策略, SpringMVC 都能够轻松支持, 与某些 MVC 框架相比, Spring 显得更加灵活;
- 5) 灵活的 Model 转换: 在 SpringWeb 框架中, 使用基于 Map 的名/值对来达到轻易地与各种视图技术的集成;
- 6) 可定制的应用程序本地化 (i18n) 和主题 (Theme): 可以在 JSP 中有选择地使用 Spring 标签库、JSTL 标签、或者 Velocity 等等, 而且不需要其他中间层。

SpringMVC 核心架构的具体流程步骤如下 (如图 2.2 所示):

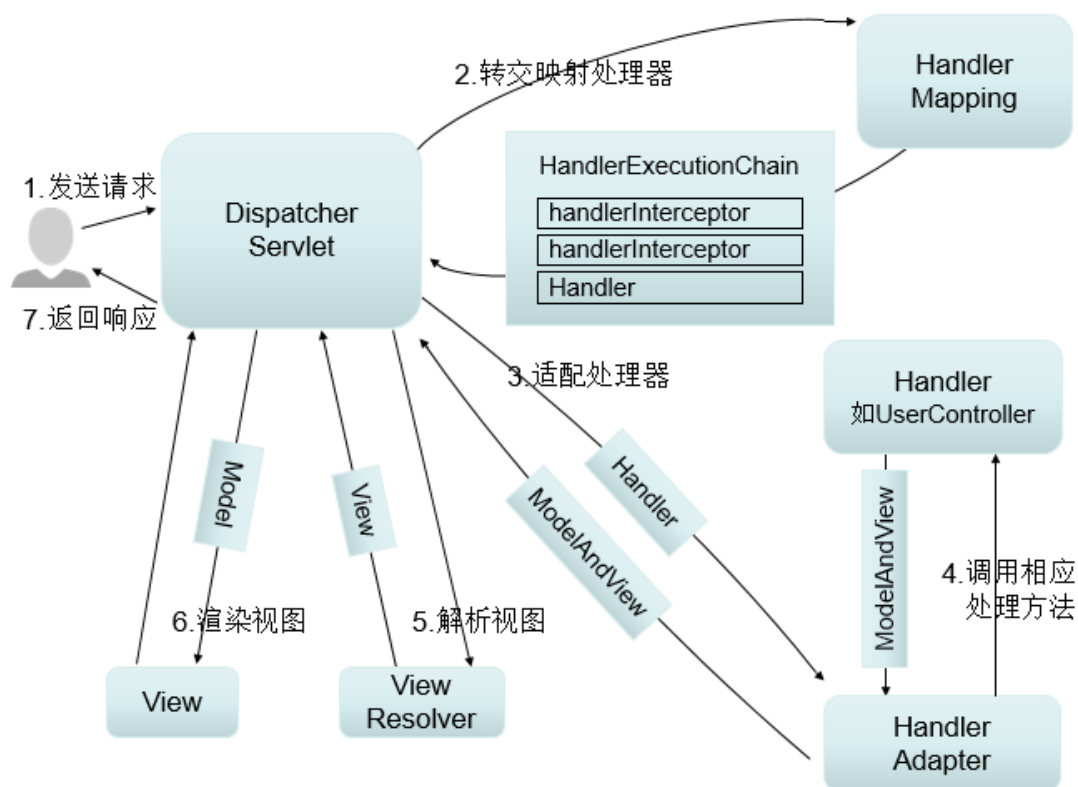


图 2.2 SpringMVC 运行机制

- 1) 首先用户发送请求到 **DispatcherServlet**: 前端控制器收到请求后自己不进行处理，而是委托给其他的解析器进行处理，作为统一访问点，进行全局的流程控制；
- 2) 从 **DispatcherServlet** 到 **HandlerMapping**: **HandlerMapping** 把请求包装为 **HandlerExecutionChain** 对象，包括一个 **Handler** 和很多个可以自由添加的 **HandlerInterceptor**；
- 3) 从 **DispatcherServlet** 到 **HandlerAdapter**: **HandlerAdapter** 把处理器包装为适配器，这里使用到了适配器的设计模式，很容易地提供了对于多类型处理器的支持；
- 4) 从 **HandlerAdapter** 从处理器功能处理方法: **HandlerAdapter** 根据适配结果调用真正处理器中的处理方法，并返回一个 **ModelAndView** 对象（包含 **Model** 模型数据和 **View** 视图名）；
- 5) 从 **ModelAndView** 到 **ViewResolver**: **ViewResolver** 把图名解析为具体的 **View**，通过这种策略模式，使得切换视图技术成为了可能；
- 6) 从 **View** 到页面的渲染，**View** 根据传进来的 **Model** 模型数据进行渲染；
- 7) 返回控制权给 **DispatcherServlet**: 由 **DispatcherServlet** 返回响应。

2.4 MySQL 数据库

MySQL 是一个多线程、多用户的 SQL 数据库服务器，即便它是开源免费的，也同样支持正规的 SQL 查询语言和各种各样的数据类型，能灵活方便地对数据进行增删改查，支持 SELECT、WHERE、JOIN、UNION 等语句的全部运算符和函数，并且可以在同一查询中混用来自不同数据库的表，从而使得查询变得方便快捷。核心程序采用多线程技术，可以灵活地为用户提供服务，它轻量级的线程又不过多的系统资源，能充分地利用 CPU 和内存，因为其拥有一个非常快速而且稳定的内存分配系统。它不仅支持 Linux，而且可以完美运行于 Windows 上，这就方便地实现了系统的移植。除此之外 MySQL 还具有一个非常安全的权限和口令系统。

虽然 MySQL 具备如上的特点，它仍然十分的稳定，仍然可以支持大型的数据库，存储上千万条数据也毫无压力。作为一个开源数据库，MySQL 可以针对不同的应用进行裁剪。

2.5 jQuery 框架和 EasyUI 框架

2.5.1 jQuery 框架

多年以来，JavaScript（简称 js）难学难用的缺点，一直困扰着开发者们。人们对 JavaScript 的不满日甚，一些有抱负的开发者为解决此问题开始编写 JavaScript 库，这些库又称为 JavaScript 框架。这些库致力于降低 JavaScript 编程的复杂程度，用一些易于使用的控制函数替掉那些困难吃力的日常任务，使已有开发者和新人都容易上手。由于用原生 JavaScript 进行 AJAX（Asynchronous JavaScript and XML）开发相当复杂，JavaScript 库在这个领域特别有用。

JavaScript 库用更简单的语法完成常见任务，这不但提高了老手的开发效率，也降低了新手的学习难度。在跨浏览器开发过程中，可方便地使用库内建的兼容所有浏览器的方法而不必手工编码进行浏览器兼容性检查，这极大地节省了编码时间，解决了跨浏览器开发过程中的棘手问题。

随着 Web 前端技术的发展，JavaScript 框架层出不穷，jQuery 就是其中之一。它是轻量级的 js 库，它支持 CSS3，对各种浏览器都能兼容。jQuery 使用户能更方便地处理 HTML（标准通用标记语言下的一个应用）、事件（Event）、实现动画效果，并且方便地为网站提供 AJAX 交互。jQuery 还有一个比较大的优势是，它的文档说明很全，而

且各种应用也说得很详细，同时还有许多成熟的插件可供选择。jQuery 能够使用户的 HTML 页面保持代码和 HTML 内容分离，也就是说，不用再在 HTML 里面插入一堆 js 来调用命令了，只需要定义 id 即可。它的核心理念是 Write less, do more（写得更少,做到更多）。jQuery 在 2006 年 1 月由美国人 John Resig 在纽约的 Barcamp 发布，吸引了来自世界各地的众多 JavaScript 高手加入，由 Dave Methvin 率领团队进行开发。如今，jQuery 已经成为最流行的 javascript 库，在世界前 10000 个访问最多的网站中，有超过 55% 在使用 jQuery。

jQuery 是免费、开源的，使用 MIT 许可协议。jQuery 的语法设计可以使开发者更加便捷，例如操作文档对象、选择 DOM 元素、制作动画效果、事件处理、使用 Ajax 以及其他功能。除此以外，jQuery 提供 API 让开发者编写插件。其模块化的使用方式使开发者可以很轻松的开发出功能强大的静态或动态网页。jQuery，顾名思义，也就是 JavaScript 和查询（Query），即是辅助 JavaScript 开发的库。

2.5.2 EasyUI 框架

jQuery EasyUI 是一组基于 jQuery 的 UI 插件集合体，而 EasyUI 的目标就是帮助 Web 开发者更轻松的打造出功能丰富并且美观的 UI 界面。开发者不需要编写复杂的 JavaScript，也不需要 CSS 样式有深入的了解，开发者需要了解的只有一些简单的 HTML 标签。EasyUI 为提供了大多数 UI 控件的使用，如：accordion, combobox, menu, dialog, tabs, validatebox, datagrid, window, tree 等等。同时页面支持各种主题（Theme）以满足使用者对于页面不同风格的喜好。

EasyUI 有以下特点：

- 1) 基于 jQuery 用户界面插件的集合
- 2) 为一些当前用于交互的 js 应用提供必要的功能
- 3) 支持两种渲染方式分别为 JavaScript 方式和 HTML 标记（Tag）方式
- 4) 支持 HTML5（通过 data-options 属性）
- 5) 开发产品时可节省时间和资源
- 6) 简单，但很强大
- 7) 支持扩展，可根据自己的需求扩展控件

2.6 AJAX 技术

AJAX 是 Web2.0 阶段系列技术和相关产品服务中非常重要的一种技术。其全称为异步 JavaScript 和 XML（即 Asynchronous JavaScript and XML）。AJAX 技术使浏览器可以为用户提供更为自然的浏览体验。在 AJAX 之前，Web 站点强制用户进入提交、等待、重新显示范例，用户的动作总是与服务器的“思考时间”同步。AJAX 提供与服务器异步通信的能力，从而使用户从请求、响应的循环中解脱出来。借助于 AJAX，可以在用户单击按钮时，使用 JavaScript 和 DHTML 立即更新 UI，并向服务器发出异步请求，以执行更新或查询数据库。当请求返回时，就可以使用 JavaScript 和 CSS 来相应地更新 UI，而不是刷新整个页面。最重要的是，用户甚至不知道浏览器正在与服务器通信：Web 站点看起来是即时响应的。

AJAX 并非一种新的技术，而是几种原有技术的结合体。它由下列技术组合而成：

- 1) 使用 CSS 和 XHTML 来表示
- 2) 使用 DOM 模型来交互和动态显示
- 3) 使用 XMLHttpRequest 来和服务器进行异步通信
- 4) 使用 JavaScript 来绑定和调用

2.7 系统开发方式

软件工程模型在软件开发过程中起着至关重要的作用，采用不同模型，在开发过程中遇到的问题也就不同，对软件质量也有着直接的影响。

软件开发是一种对人类智慧的管理，对人大脑思维的“工厂化”管理。人是有感情的、有情绪的、变化的、相对独立的工作单元，这与冰冷的机器是不可比的，所以在中国的历史上，管理人是最难的工作；“学而优则仕”的观点就是让最聪明的人应该选出来做官，做官就是管理人的。软件开发不仅是代码编程，而是人员的有效组织，如何既发挥人的主观能动性，避免情绪变化对工作的影响，又可以让大家有效的交流，让多个大脑的思路统一，快速完成目标呢？多年来软件企业的管理者一直在不断地探索。

另外有一个问题一直是软件开发管理人员的心病：软件是工具，开发的是客户业务的应用，但客户不了解软件，开发者不了解业务，如何有效沟通是软件质量的重大障碍。把开发者变成客户业务的专家是个没有办法的办法，让软件企业付出的代价也是昂贵的。

瀑布模型、极限编程、敏捷开发是有代表性的开发模式，在对开发者、客户、最终的产品关注上的变化，体现了软件开发管理者在管理模式上的变化。

2.7.1 瀑布模型

瀑布模型（Waterfall Model）是 Royce 在 1970 年提出的，他把大型软件开发分为：分析与编程，象工厂流水线一样把软件开发过程分成各种工序，并且每个工序可以根据软件产品的规模、参与人员的多少进一步细分成更细的工序。该模型非常符合软件工程的分层设计思路，所以成为软件开发企业使用最多的开发模型。瀑布模型过程图如图 2.3 所示。

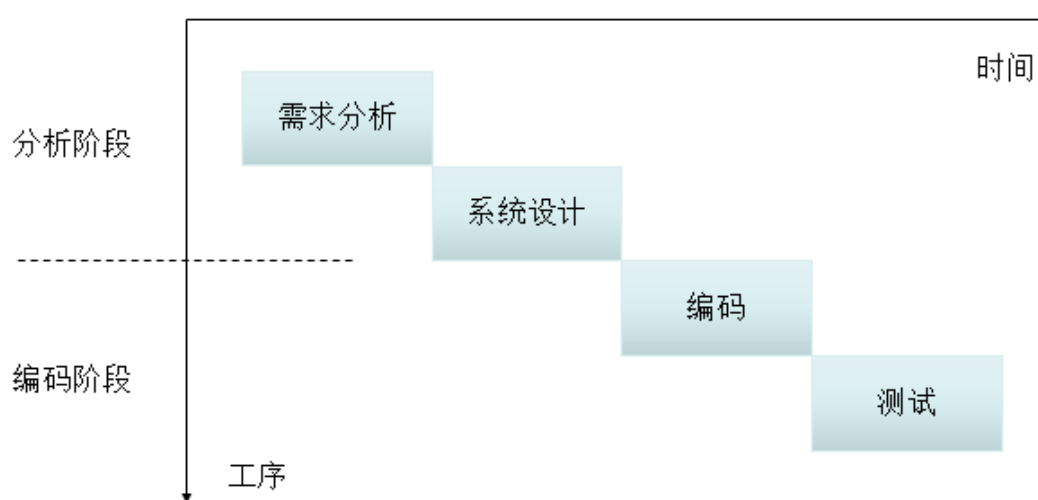


图 2.3 瀑布模型过程图

瀑布模型的特点：

- 1) 强调文档，前一个阶段的输出就是下一个阶段的输入，文档是个阶段衔接的唯一信息。所以很多开发人员好象是在开发文档，而不是开发软件，因为要到开发的后期，才可以看到软件的“模样”。
- 2) 没有迭代与反馈。瀑布模型对反馈没有涉及，所以对变化的客户需求非常不容易适应，瀑布就意味着没有回头路。
- 3) 管理人员喜欢瀑布模型的原因是把文档理解为开发的速度，可以方便地界定不同阶段的里程碑。

2.7.2 极限编程

极限编程诞生于一种加强开发者与用户的沟通需求，让客户全面参与软件的开发设计，保证变化的需求及时得到修正。要客户能方便地与开发人员沟通，一定要用客户理解的语言，先测试再编码就是先给客户软件的外部轮廓，客户使用的功能展现，让客户感觉到未来软件的样子，先测试再编码与瀑布模型显然是背道而驰的。同时，极限编程注重用户反馈与让客户加入开发是一致的，让客户参与就是随时反馈软件是否符合客户的要求。有了反馈，开发子过程变短，迭代也就很自然出现了，快速迭代，小版本发布都让开发过程变成更多的自反馈过程，有些象更加细化的快速模型法。当然极限编程还加入了很多激励开发人员的“措施”，如结队编程、40 小时工作等。

极限编程把软件开发过程（如图 2.4 所示）重新定义为聆听、测试、编码、设计的迭代循环过程，确立了测试->编码->重构(设计)的软件开发管理思路。

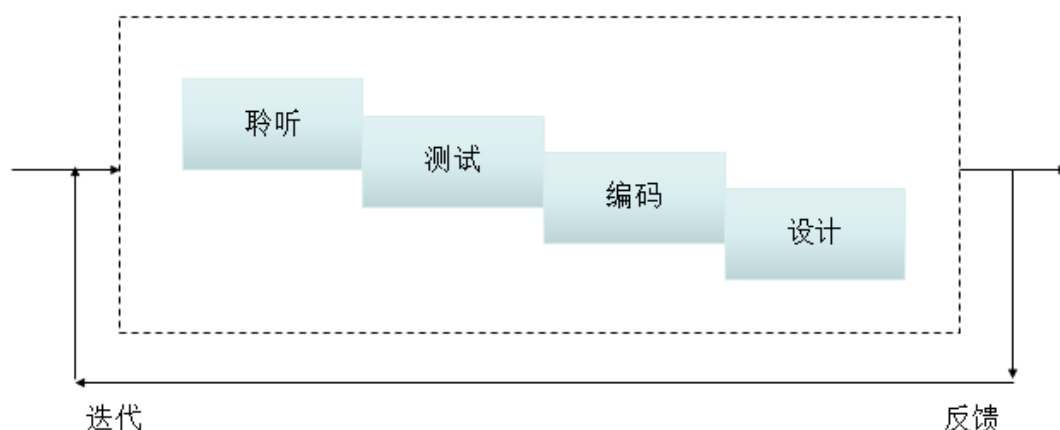


图 2.4 极限编程过程图

2.7.3 敏捷开发

敏捷开发以用户的需求进化为核心，采用迭代、循序渐进的方法进行软件开发。在敏捷开发中，软件项目在构建初期被切分成多个子项目，各个子项目的成果都经过测试，具备可视、可集成和可运行使用的特征。换言之，就是把一个大项目分为多个相互联系，但也可独立运行的小项目，并分别完成，在此过程中软件一直处于可使用状态。

敏捷就是“快”，快才可以适应目前社会的快节奏；要快就要发挥个人的个性思维多一些，个性思维的增多，虽然通过结队编程、代码共有、团队替补等方式减少个人对

软件的影响力，但也会造成软件开发继承性的下降，因此敏捷开发是一个新的思路，但不是软件开发的终极选择。

对于长时间、人数众多的大型软件应用的开发，文档的管理与衔接作用还是不可替代的。如何把敏捷的开发思路与传统的“流水线工厂式”管理有机地结合，是软件开发组织者面临的新课题。

3 就业管理系统的需求分析

3.1 系统背景和总体目标

沈阳理工大学信息科学与工程学院原先采用的就业管理方式为手工管理，首先由各班班长在 Excel 中统计本班学生个人信息并交给本班辅导员，辅导员对所管辖范围内的各班学生信息进行汇总，再进行打印信息确认。班长定期统计本班的就业情况上报辅导员，辅导员在 Excel 中更新学生的就业状态。学院每月需要将所有辅导员的数据进行汇总，并统计就业情况产生报表。每年对于学生的就业管理工作占用了工作人员大量的时间，并且经常出错。

就业管理系统的主要目的就是学生的就业管理达到信息化、系统化、规范化和管理的自动化，以便能够更好地实现学生就业信息的管理、就业协议信息的管理、系统操作人员对系统的管理等，有利于工作人员能够及时地掌握学生的就业情况和用人单位的招聘信息。结合我国现在毕业生就业管理现状，根据我院毕业生就业管理的特点，设计实现了一套适合我校毕业生就业的管理系统。

3.2 功能性需求

学生就业信息管理系统包括以下子系统：学生自然信息管理子系统、学生就业意向管理子系统、学生就业情况管理子系统、字段管理子系统、就业率报表子系统、毕业设计指导教师管理子系统和系统设置子系统。

3.2.1 用例图

1、学生自然信息管理

学生的自然情况要求能够通过现存的 Excel 进行批量导入，且不仅辅导员可以编辑自己的信息，学生也应该能够编辑自己的个人信息，以提高管理效率。学生自然信息管理子系统的用例图如图 3.1 所示：

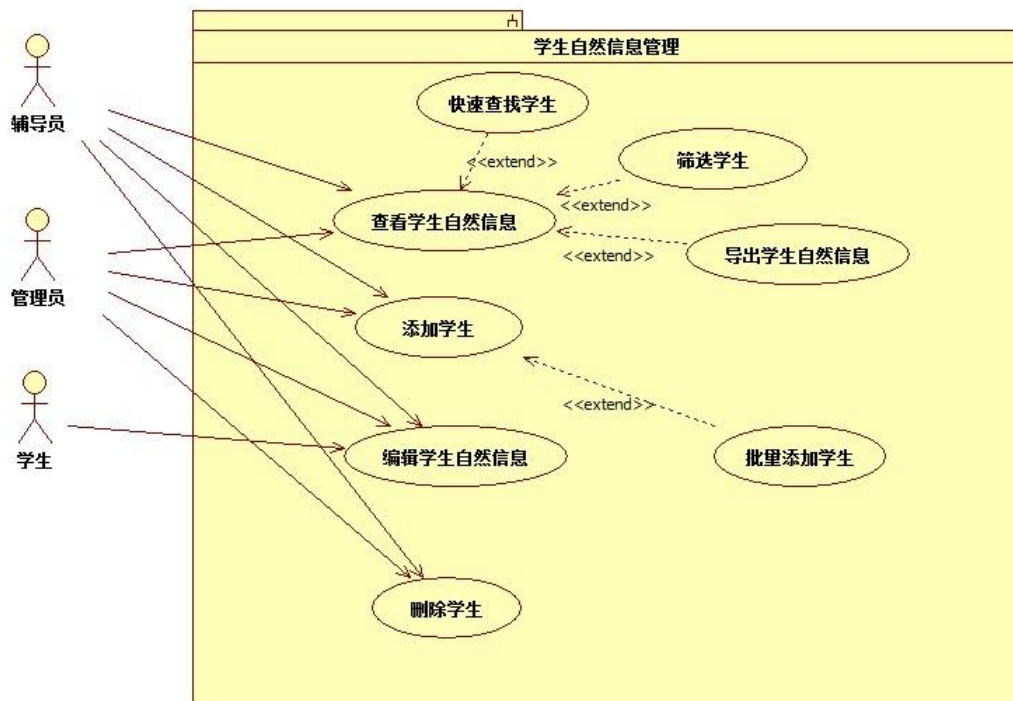


图 3.1 学生自然信息管理用例图

2、学生就业意向管理

学生可以通过该模块填写自己的就业意向，以便企业进行查询。学生就业意向管理子系统的用例图如图 3.2 所示：

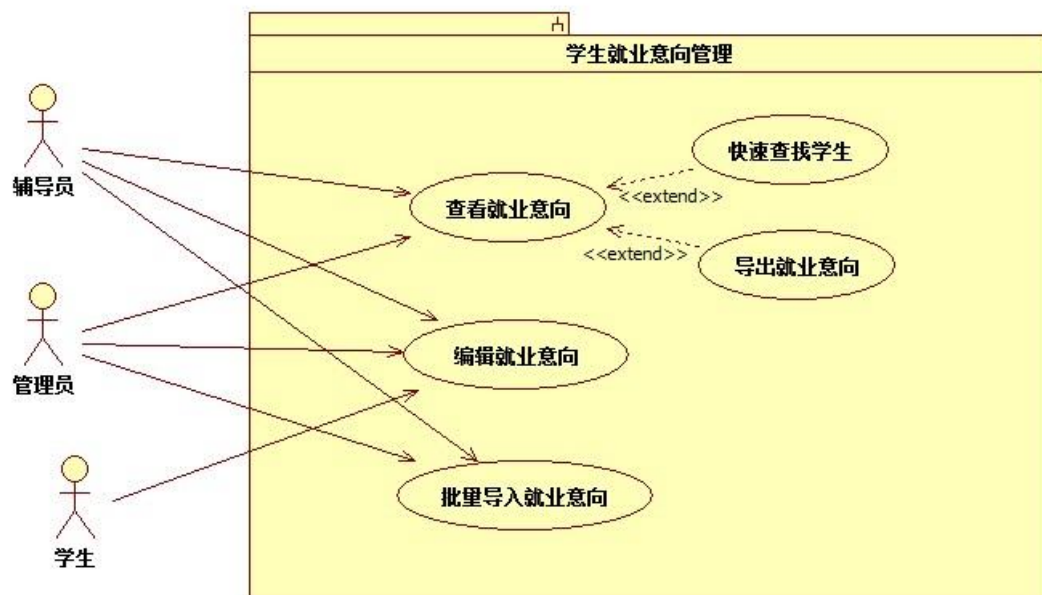


图 3.2 学生就业意向管理用例图

3、学生就业情况管理

学生就业情况需要学生和辅导员共同维护，学生就业情况管理子系统的用例图如图 3.3 所示：

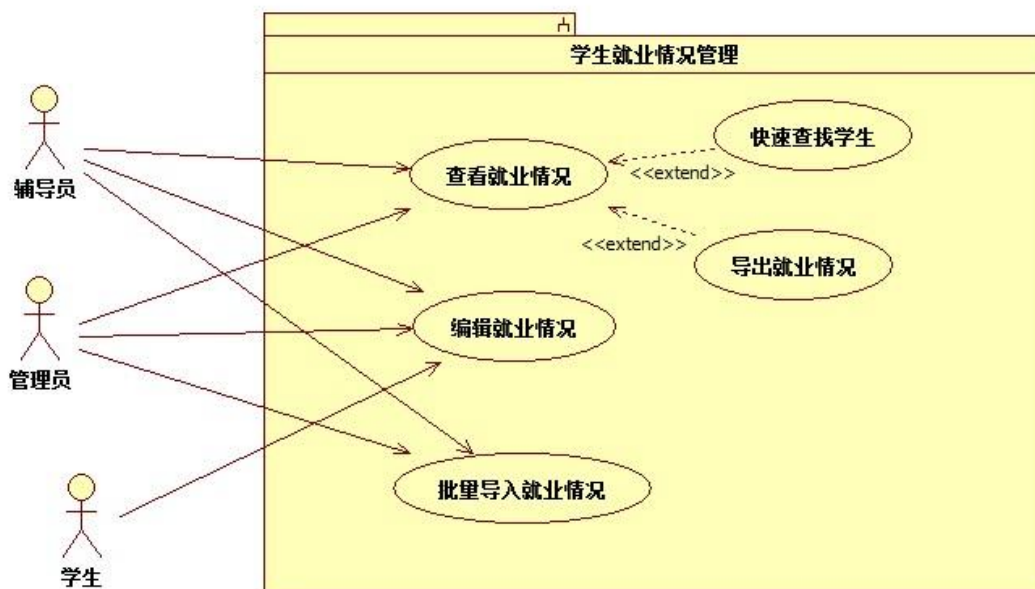


图 3.3 学生就业情况管理用例图

4、就业率报表

系统需要能自动统计就业率，且就业率应该可以从专业、班级、辅导员、毕业设计指导教师等多个维度进行统计。就业率报表子系统的用例图如图 3.4 所示：

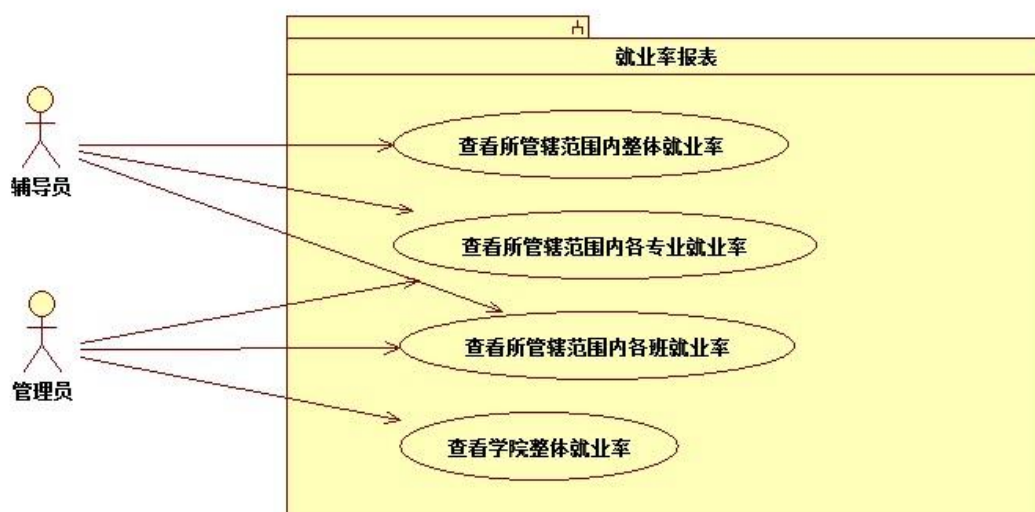


图 3.4 就业率报表用例图

5、字段管理

由于系统今后有可能要添加或者删除某些字段，为了便于系统的维护，要求系统具有字段管理的功能。字段管理子系统中由包含三个子系统：学生自然情况字段管理子系统、学生就业意向字段管理子系统和学生就业情况管理子系统。该系统的用例图如图 3.5 所示：

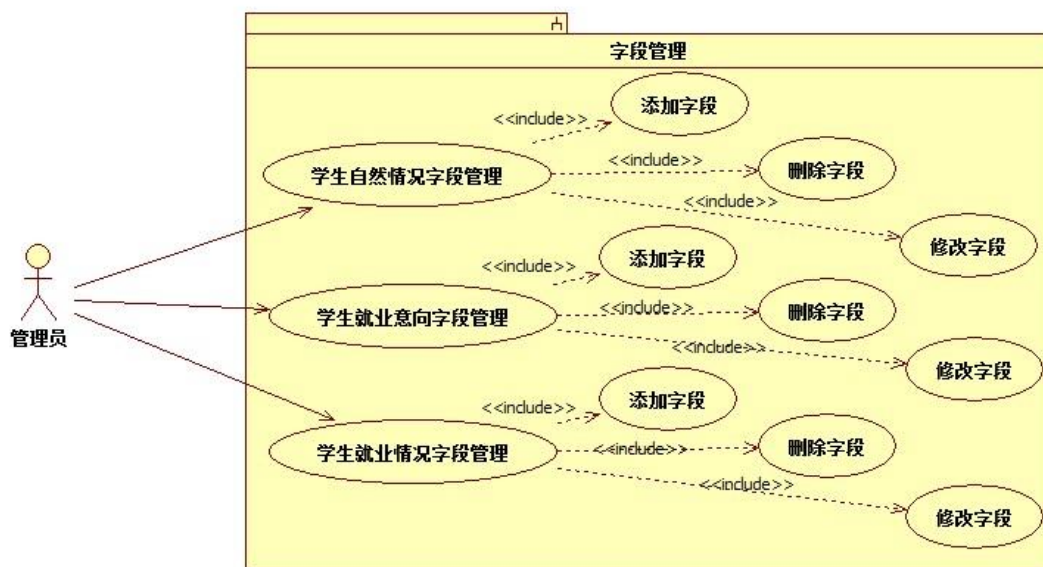


图 3.5 字段管理用例图

6、毕业设计指导教师管理

由于就业率需要从毕业设计指导教师和教研室的角度进行计算，所以需要对毕业设计指导教师和教研室进行管理。毕业设计指导教师管理子系统的用例图如图 3.6 所示：

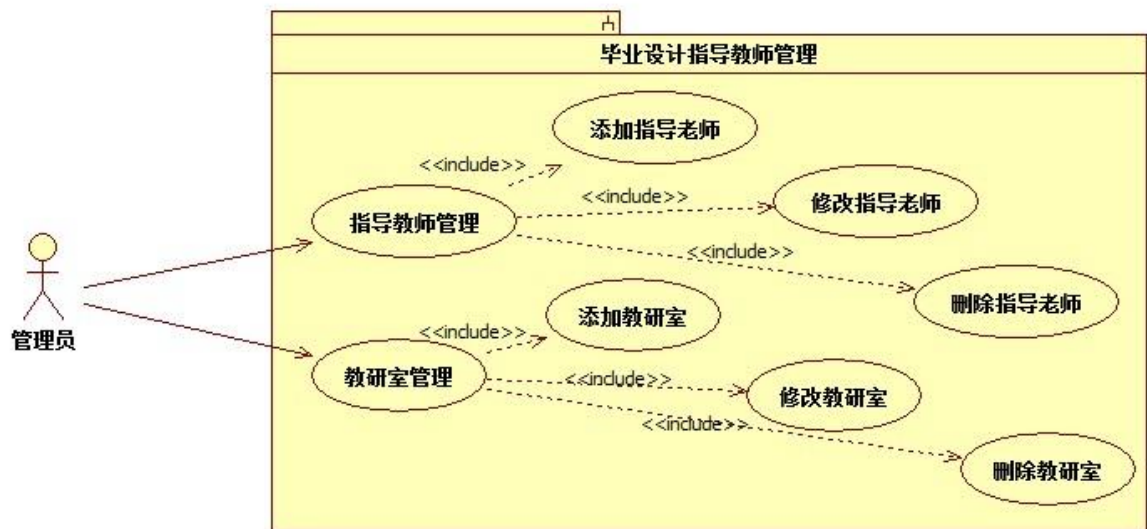
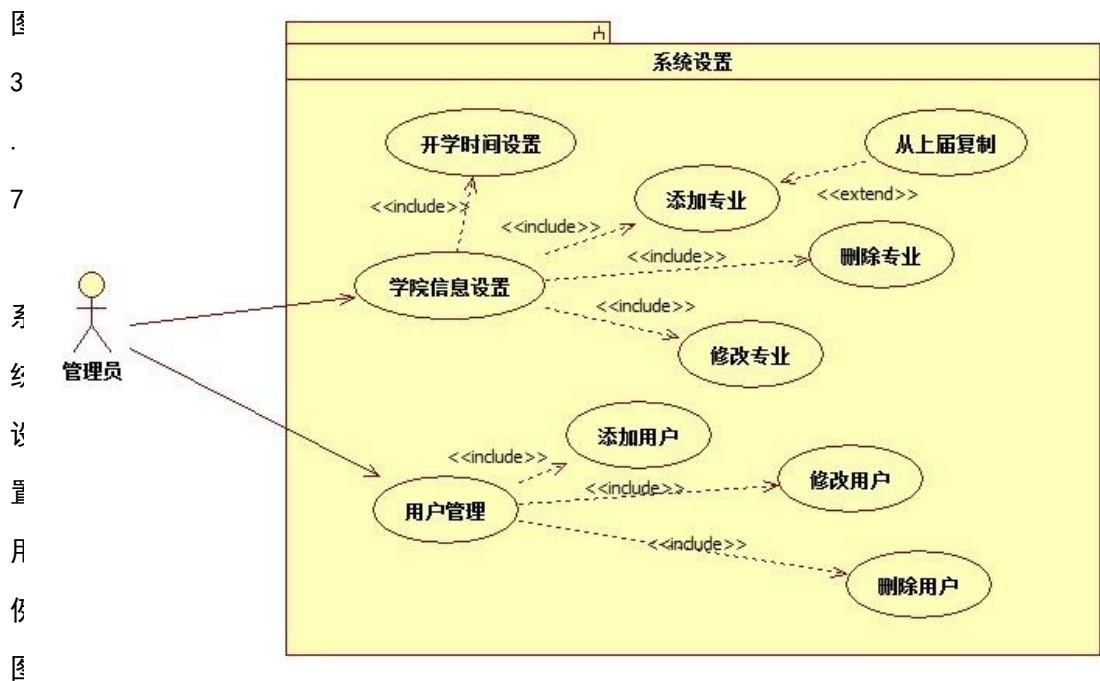


图 3.6 毕业设计指导教师管理用例图

7、系统设置

系统设置子系统中又包含：学院信息设置子系统和用户设置子系统。其用例图如图 3.7 所示：



3.2.2 业务流程图

1、学生自然信息录入业务流程

该流程涉及到管理员、辅导员、班长、学生等多个角色，业务流程较为复杂，该业务的流程图如图 3.8 所示：

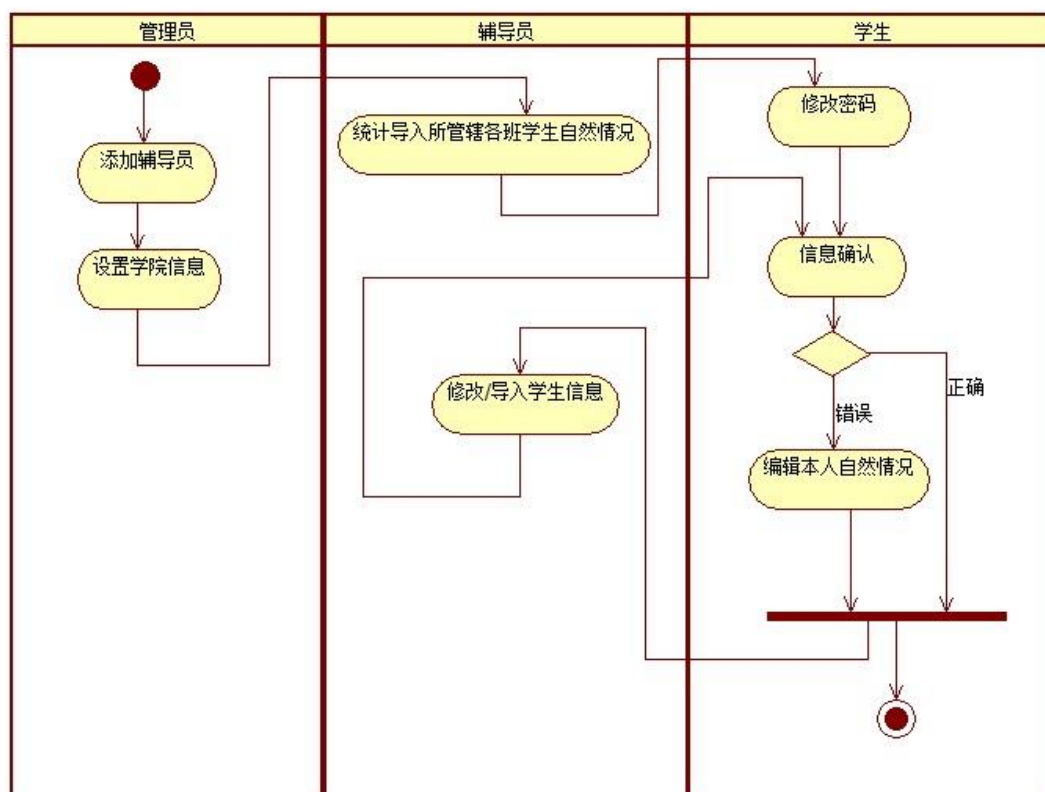


图 3.8 学生自然信息录入业务流程图

2、学生就业情况录入流程

在学生填写自己的就业情况之后，需要辅导员进行确认，其业务流程图如图 3.9 所示：

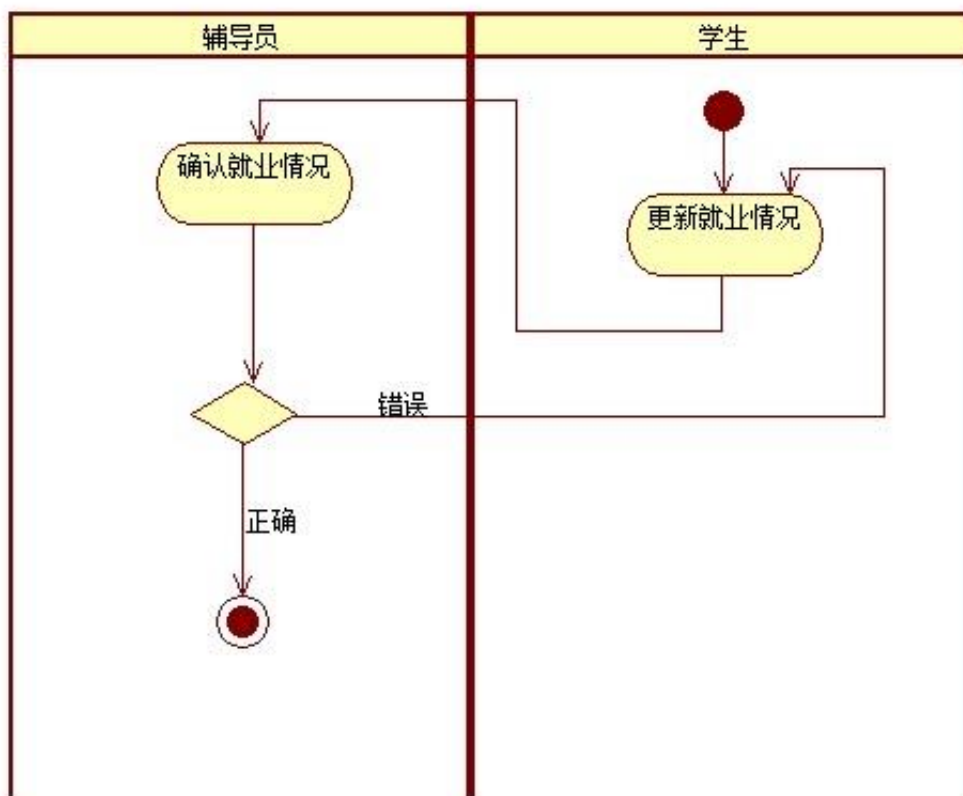


图 3.9 就业情况录入业务流程图

3.3 非功能性需求

1、运行环境

由于开发需求的不确定性，暂无法确定系统运行于办公室局域网内还是部署在校园网服务器上，考虑到这种情况，本系统需要能够部署在 Windows XP/7/8 各版本和 Linux 各发行版上。

2、用户界面

由于系统用户较广，涉及到学院领导、辅导员和学生，不能保证所有用户具有很强的计算机操作能力，且开发成本有限，所以要求系统界面基于 Web 设计，简洁大方、操作简单、易学。系统应尽量减少工作人员的数据录入量，录入数据尽量通过设计下来列表框来选择录入，这样避免了录入数据的错误，同时也避免了录入数据的异常现象的发生。除此之外还应该具有数据导入功能。

3、性能

由于运行环境的不确定，要求系统尽量具有体积小、运行快的特点。

4、安全

由于系统涉及院系机密和学生个人信息等，一方面要保证数据的不外泄，具有严格的权限机制，另一方面又要在异常情况下保证数据的丢失减少到最低限度，具有数据的备份和恢复功能。

3.4 系统可行性分析

3.4.1 技术可行性分析

基于系统分析的结果是技术可行性的支撑，并利用现有开发技术分析是否能够实现待开发的系统。本系统运用 Web 开发技术，通过 JDBC 数据源实现与 MySQL 数据库的交互操作，本系统采用 B/S 结构模式，把整个系统分为独立三层结构，界面展示层主要接收用户的前台服务请求，并且把得到的响应结果返回给前台展示，为客户端提供应用程序的访问。应用逻辑层主要负责对数据访问层的操作，也就是说把一些数据访问层的操作进行封装。数据访问层主要通过数据源与数据库进行交互。由于 Web 技术的系统平台，操作简单，安全可靠，维护方便，并且具有开发工具 Eclipse 开源。根据高校现有开发能力和开发人员的技术水平，完全能够达到系统技术要求，完成系统的开发工作，因此本系统在技术上是可行的。

3.4.2 社会可行性分析

随着计算机技术的飞速发展，计算机软件已覆盖了家庭、企业、学校应用的各个方面，大部分企业和学校都有适合自己特征的专用的办公软件。对于学校就业市场来说，就业管理系统软件不仅能很大的节约人力成本、提高工作效率，并且对学校的管理会变得透明化，学校决策人员可以随时了解学生就业的情况，并且根据就业率来制定相应的招生方案和学生质量培养策略。就业管理系统的易操作性能极大的方便就业指导教师和管理人员的操作，并且会很好的让用人单位了解学校有意向的毕业生信息，也极大的方便了毕业生掌握招聘单位的第一手招聘岗位资料，从而为毕业生成功的走向工作岗位赢得了时间，也把潜在的企业招聘信息变为实际招聘工作人员，从广义上来看，不仅能够更快、更好的完成繁重的就业工作，也极大的提高了学校的就业工作特色和办学知名度。

3.4.3 经济可行性分析

对于就业管理工作来说，经过多年的工作经验相关积累，聚集了大量的企业人脉，也有往届的毕业生提供招聘职务，为学校提供了大量的就业工作岗位。但由于就业市场不断的变化，近年来，毕业生生源也逐年增加，2014 年全国毕业生总量达到 727 万创下全国历史以来之最，给就业工作带来了巨大的压力，因此学校就业也开始思考未来的就业渠道。在经济方面，各个高校都有自己的局域网，实现开发就业管理系统后布署到本校的内部网络容易实现对接。本系统将采用 B/S 结构模式，整个系统会分为独立三层结构，对于软件和硬件的要求不是很高，与此相应的只需要有一定容量的 Web 服务器，MySQL 数据库以及个人电脑即可。由于 Web 化开发技术周期较短，相对来说能够节约大量的人力和物力，这不但能够保证系统的经济性，而且还能够保证技术的可行性。由于就业管理系统开发成本相对较低，不仅保证了该产品在学校就业市场的竞争优势，并且系统的易操作性更有助于该产品在学校市场得到推广。

4 就业管理系统概要设计

4.1 项目的搭建

4.1.1 版本控制

本项目使用 Subversion（简称 SVN）作为版本控制工具。软件配置管理中重要的一环就是版本控制，通过版本控制对软件项目中的文档、源码等资源的多重版本实施系统的管理，全面记载系统开发的历史过程包括为什么修改，谁作了修改，修改了什么，完整明确地记载开发过程中的历史变更。如果是采用人工的办法不仅费时费力，还极易出错，因此引入一些自动化工具是十分有裨益的，SVN 正是这样的一种自动化的版本控制工具。

使用树莓派和 SCM-Manager 可以方便地搭建一台版本控制服务器，首先从 Apache 官网下载 Tomcat 7.0, 然后下载 SCM-Manager 的 war 包将其放置在 Tomcat 中的 webapps 目录下即可。安装完成后建立用于本项目的版本控制仓库，配置如图 4.1 所示：

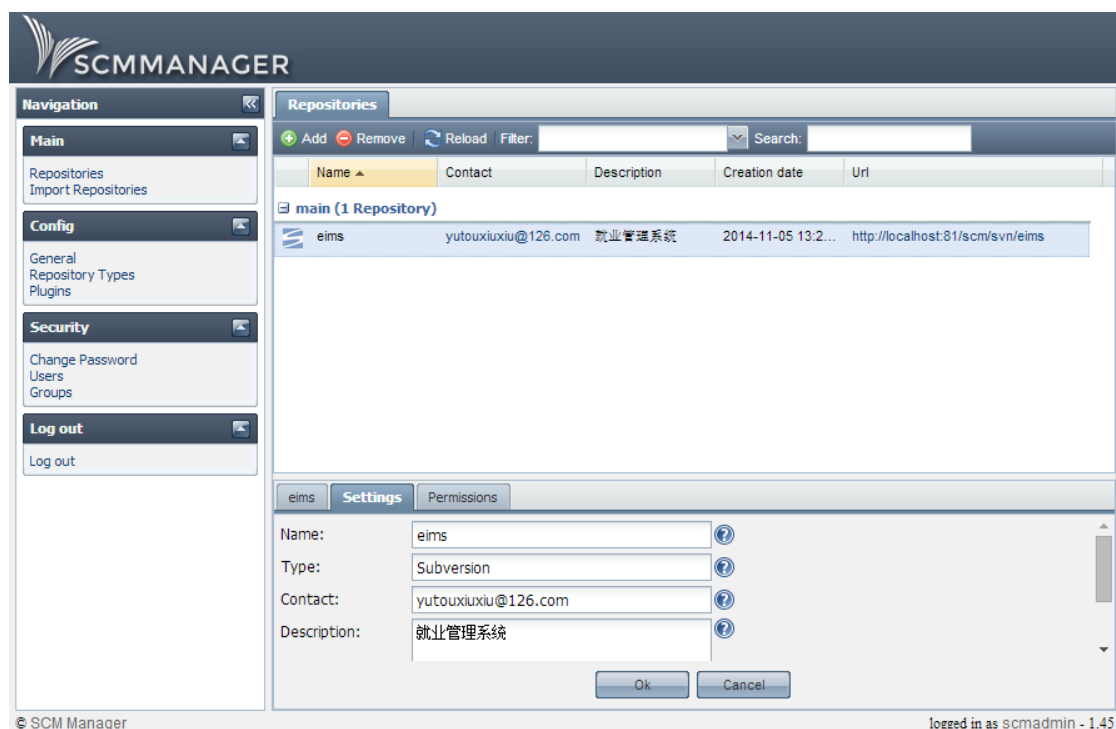


图 4.1 使用 SCM-Manager 建立 SVN 版本控制库

4.1.2 依赖管理

本项目使用 Maven 作为项目管理工具，Maven 作为 Apache 组织中的一个颇为成功的开源项目，主要服务于基于 Java 平台的项目构建、依赖管理和项目信息管理。无论是小型的开源类库项目，还是大型的企业级应用；无论是传统的瀑布式开发，还是流行的敏捷模式，Maven 都能大显身手。

Maven 使用 pom 文件进行依赖的配置。在 pom 文件的头部定义团体名称（groupId）为 com.yutouxixiu、项目名称（artifactId）为 eims、项目打包类型（packaging）为 war（以便于部署于 Tomcat）和版本号（version）为 0.0.1-SNAPSHOT。

```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.yutouxixiu</groupId>
  <artifactId>eims</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
```

接下来 dependencies 标签下声明对于 Spring 框架、JSTL 标签库、MySQL 数据库驱动、Selenium 自动化测试工具（在软件测试部分详细介绍）、POI 工具（在数据导入导出部分详细介绍）、Log4j 日志系统（在异常处理部分详细介绍）等项目的依赖关系。

```
<dependencies>
  <!-- spring -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
  </dependency>
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>${spring.version}</version>
</dependency>
.....
```

最后声明项目构建所需的 Tomcat 调试插件、JavaDOC 文档生成插件、maven-release 自动化发布插件等。

```
<build>
  <finalName>${project.artifactId}</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <source>${java.version}</source>
        <target>${java.version}</target>
        <encoding>UTF-8</encoding>
      </configuration>
    </plugin>
    .....
  </plugins>
</build>
```

4.2 项目架构设计

1、项目目录结构

项目整体架构于 SpringMVC 之上，这款 MVC（Model-View-Controller）框架将项目有机地分为三层，对软件进行了解耦，提高了程序的可读性，同时利于项目的分工。

在本项目中 com.yutouxiuxiu.eims.bean 和 com.yutouxiuxiu.eims.service 两个包下的

JavaBean 和 Service 类属于模型（Model）层，用于处理应用程序数据逻辑的部分，负责在数据库中的存取工作；webapp 目录下放置的是 JSP、Javascript、CSS 和图片文件，属于展示层（View），负责页面的显示，是用户操作软件的窗口；com.yutouxiuxiu.eims.controller 包下的 Controller 类属于控制（Controller）层，用于从视图读取数据，控制用户输入，并向模型发送数据。由于本项目是基于 Maven 构建的，所以源码文件需要放置在 src/main/java 目录中，项目目录结构如下图所示：

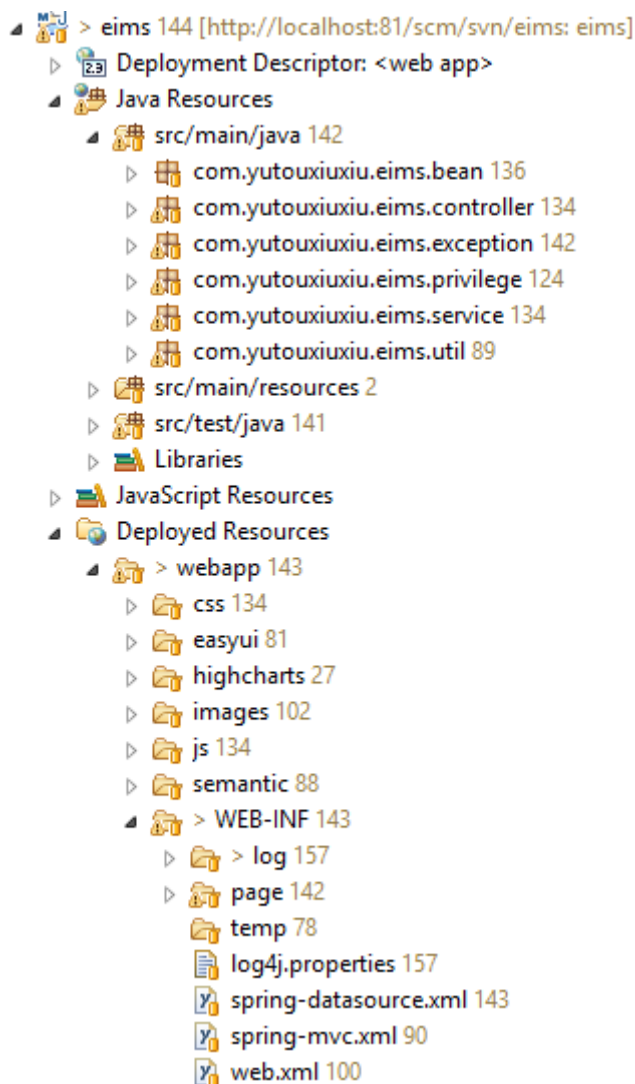


图 4.2 项目结构图

webapp 目录下的 css 目录用来存放样式表文件；easyui 目录是 EasyUI 框架代码；highcharts 目录下是 Highcharts 源码，Highcharts 是一个用纯 JavaScript 编写的一个图表库，能够很简单便捷的在 Web 网站或是 Web 应用程序添加有交互性的图表，HighCharts

支持的图表类型有曲线图、区域图、柱状图、饼状图、散状点图和综合图表等；images 目录中存放的是项目所需的图片；js 目录用来存放项目中使用的 JavaScript 源码；semantic 中为 Semantic UI 框架源码，本项目的登录页面和学生登录后的管理页面采用 Semantic UI 设计；WEB-INF 目录下是项目的 jsp 文件、日志文件、临时目录和项目配置文件。

2、web.xml 配置文件

首先是 web.xml 文件，配置 Log4j 配置文件的目录和 Web 的根目录，以及它们各自的 listener。

```
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
    <!-- log4j -->
    <context-param>
        <param-name>log4jConfigLocation</param-name>
        <param-value>/WEB-INF/log4j.properties</param-value>
    </context-param>

    <!-- web root -->
    <context-param>
        <param-name>webAppRootKey</param-name>
        <param-value>web.root</param-value>
    </context-param>

    <listener>
        <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>
    </listener>

    <listener>
```

```
<listener-class>org.springframework.web.util.WebAppRootListener</listener-cl
ass>
</listener>
```

再配置使得 SpringMVC 接管所有的 Web 请求，并配置 SpringMVC 配置文件的路径为 WEB-INF/spring-mvc.xml。

```
<!-- spring mvc -->
<servlet>
    <servlet-name>springMVC</servlet-name>

    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet
-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>springMVC</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

配置 CharacterEncodingFilter 来解决全站的 POST 中文乱码问题，本项目中数据库，程序和 Jsp 页面全部使用 UTF-8 编码。

```
<!-- encoding -->
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-cla
ss>
```



```
<init-param>
    <param-name>encoding</param-name>
    <param-value>utf-8</param-value>
</init-param>
</filter>
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

配置 URI “/” 映射到 “/index”

```
<!-- index -->
<servlet-mapping>
    <servlet-name>springMVC</servlet-name>
    <url-pattern>/index</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>index</welcome-file>
</welcome-file-list>
```

将 404 错误在自定义的 404 页面中显示，隐藏 Jsp 原本的错误页面。

```
<!-- error -->
<error-page>
    <error-code>404</error-code>
    <location>/WEB-INF/page/error/404.jsp</location>
</error-page>
</web-app>
```

3、SpringMVC 配置文件

接下来需要配置 SpringMVC 的配置文件，先引入数据源的配置文件

spring-datasource.xml，该文件在本节后面部分将详细介绍。

```
<import resource="spring-datasource.xml" />
```

配置 Controller 类和 Service 类所在的包，以便 SpringMVC 将需要的 Service 类注入到 Controller 类中。

```
<context:component-scan base-package="com.yutouxixiu.eims" />
```

配置异常处理机制，本项目中使用 AOP 环绕切面进行异常的统一处理，在异常处理的章节将专门介绍。

```
<!-- privilege -->
<bean id="privilegeAspect"
class="com.yutouxixiu.eims.privilege.PrivilegeAspect"></bean>
<aop:config>
    <aop:aspect ref="privilegeAspect">
        <aop:around method="doAround" pointcut="execution(*
com.yutouxixiu.eims.controller.*(..))" />
    </aop:aspect>
</aop:config>

<!-- exception -->
<bean id="exceptionResolver"
class="com.yutouxixiu.eims.exception.ExceptionResolver">
    <property name="exceptionMappings">
        <props>
            <prop key="java.lang.RuntimeException">error/500</prop>
            <prop
key="com.yutouxixiu.eims.exception.PrivilegeException">error/privilege</prop>
        </props>
    </property>
</bean>
```

配置注解驱动,使 Controller 中 Action 方法于 URI 的映射可以使用注解在方法上方配置。

```
<mvc:annotation-driven />
```

配置静态文件路径和 URI 的映射关系,在 SpringMVC 对 Web 请求的处理中排除静态文件。

```
<!-- static resources -->
```

```
<mvc:resources mapping="/css/**" location="/css/" />
```

```
<mvc:resources mapping="/js/**" location="/js/" />
```

```
<mvc:resources mapping="/images/**" location="/images/" />
```

```
<mvc:default-servlet-handler />
```

配置展示层的目录“/WEB-INF/page/”和展示层文件的后缀“.jsp”,这样当 Controller 中的 Action 方法返回字符串时, SpringMVC 拼接字符串与前后缀,找到对应的展示层 jsp 文件。

```
<!-- view -->
```

```
<bean
```

```
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
```

```
<property name="viewClass"
```

```
value="org.springframework.web.servlet.view.JstlView" />
```

```
<property name="prefix" value="/WEB-INF/page/" />
```

```
<property name="suffix" value=".jsp" />
```

```
</bean>
```

配置文件上传处理方式和上传文件允许的最大尺寸。

```
<bean id="multipartResolver"
```

```
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
```

```
<property name="maxUploadSize">
```

```
<value>10485760</value>
```

```
</property>
```

```
</bean>
```

4、数据源配置文件

首先配置数据源和 MySQL 数据库的连接。

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url"
value="jdbc:mysql://127.0.0.1:3306/eims?useUnicode=true&characterEncoding=utf
-8" />
    <property name="username" value="root" />
    <property name="password" value="root" />
</bean>
```

接下来配置事务管理器，事务管理是指作为单个逻辑工作单元执行的一系列操作，要么完全地执行，要么完全不执行。事务处理可以确保除非事务性单元内的所有操作都成功完成，否则不会永久更新面向数据的资源。通过将一组相关操作组合为一个要么全部成功要么全部失败的单元，可以简化错误恢复并使应用程序更加可靠。使用切入点表达式 “ execution(* com.yutouxixiu.eims.service.*(..)) ” 配置 com.yutouxixiu.eims.service 包中所有类中的所有以如下字符串开头的方法都采用事务管理。

```
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="save*" propagation="REQUIRED"/>
        <tx:method name="del*" propagation="REQUIRED"/>
        <tx:method name="update*" propagation="REQUIRED"/>
        <tx:method name="add*" propagation="REQUIRED"/>
        <tx:method name="find*" propagation="REQUIRED"/>
        <tx:method name="list*" propagation="REQUIRED"/>
        <tx:method name="get*" propagation="REQUIRED"/>
    </tx:attributes>
</tx:advice>
```

```
</tx:attributes>
</tx:advice>

<aop:config>
    <aop:pointcut          id="txServicePointcut"          expression="execution(*
com.yutouxixiu.eims.service.*(..))"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="txServicePointcut" />
</aop:config>

<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
```

配置 JdbcTemplate 使用的数据库是上面定义的数据源，JdbcTemplate 是对 JDBC 的封装，其中涵盖了很多简便操作数据库的方法。

```
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource" />
</bean>
```

5、Log4j 配置

为了记录系统中产生的错误，便于开发人员跟踪于维护，在该项目中使用了 Log4j 日志系统进行日志的记录与管理。Log4j 是 Apache 的一个开放源代码项目，通过使用 Log4j，我们可以控制日志信息输送的目的地是控制台、文件、GUI 组件，甚至是套接口服务器、NT 的事件记录器、UNIX Syslog 守护进程等；我们也可以控制每一条日志的输出格式；通过定义每一条日志信息的级别，我们能够更加细致地控制日志的生成过程，Log4j 建议使用的日志级别有 5 个，分别是：DEBUG（细粒度信息事件，对调试应用程序是非常有帮助的，就是输出调试信息）、INFO：表明消息在粗粒度级别上突出强调应用程序的运行过程，就是输出提示信息）、WARN（会出现潜在错误的情形，就是显示警告信息）、ERROR（虽然发生错误事件，但仍然不影响系统的继续运行，就是显

示错误信息)、FATAL (严重的错误事件将会导致应用程序的退出)。之前已经在 web.xml 中加载了该配置文件,下面对配置文件做以详细的说明。

当前项目在生产环境下使用的是名为 file 的日志记录器,记录 ERROR 及以上级别的日志。

```
# log4j.properties
log4j.rootLogger=ERROR,file
```

配置 file 日志记录器在文件中记录日志,日志文件的存放路径为“/WEB-INF/log/system.log”,当文件超过 1024KB 时将另外新建一个文件保存新的日志,并且最多保存 20 个文件,配置中还对日志的格式做了规定,先显示错误的级别,在显示错误的发生时间,最后显示错误信息。

```
#file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=${web.root}/WEB-INF/log/system.log
log4j.appender.file.Threshold=ERROR
log4j.appender.file.Append=true
log4j.appender.file.MaxFileSize=1024KB
log4j.appender.file.MaxBackupIndex=20
log4j.appender.file.Encoding=UTF-8
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=[%p]           %d{yyyy-MM-dd
HH:mm:ss,SSS} %m%n
```

在程序调试过程中在文件中查看错误信息使得开发过程变得麻烦,所以还配置了 console 日志记录器,它可以将日志输出到控制台,便于实时查看。

```
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.Threshold=ERROR
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=[%p] %d{HH:mm:ss,SSS} %m%
n
```

4.3 系统模块设计

在综合考虑了业务需求和技术实现方面的问题之后，由于某些子系统的实现方式相当类似，固在本文中系统分为如下几个模块进行叙述：用户权限模块、学生信息管理模块、字段管理模块、统计报表模块、毕业设计指导教师管理模块、系统设置模块等。

用户权限模块主要是对用户进行管理，以及处理用户的验证。包括对用户的增删改查、对用户权限的修改、处理用户的登录请求，对每个方法的执行进行权限检查、处理用户退出等业务。

学生信息管理模块是本系统的核心部分，对学生的自然信息、就业意向和就业情况进行管理。包括对学生的增删改查，对学生自然情况、就业意向和就业情况的筛选、修改，处理学生数据的导入和导出。该模块分为前台和后台两个部分，学生用学号登录之后可以进入系统的前台，对自己的个人信息、就业意向和就业情况进行信息确认和修改；系统的后台面向辅导员和管理员，对学生进行整体管理。

字段管理模块是对学生自然信息、就业意向和就业情况中的字段进行管理，完成字段的增删改查。该模块是该系统中的难点所在。

统计报表系统目前负责对学生就业率进行不同纬度上的统计，将来还有可能增加其他报表。

毕业设计指导教师管理模块负责对学院的教研室和毕业设计指导教师进行增删改查。

系统设置模块中包括对院系的设置，将来还有可能增加其他的设置项。

4.4 数据库设计

4.4.1 数据库模型设计

本项目使用 Power Designer 进行数据库的设计。Power Designer 是 Sybase 公司的 CASE 工具集，使用它可以方便地对管理信息系统进行分析设计，他几乎包括了数据库模型设计的全过程。利用 Power Designer 可以制作数据流程图、概念数据模型、物理数据模型，还可以为数据仓库制作结构模型，也能对团队设计模型进行控制。

首先设计学生信息管理模块的数据库模型，学生表（students）主要存放学生的登录密码和自然情况，其中包括学生的 id、学号（number）、密码（password）、信息是否被学生确认标记（confirmed）、姓名（name）、性别（gender）、身份证号（idCard）等学生

的必要信息，其他字段为后期在系统中动态添加的，这里不做介绍；就业意向表（intentions）存放关于学生就业意向的信息，表中的 student 作为外键和学生表的 id 进行关联，表中的其他字段为后期再系统中动态添加的；就业情况表（obtains）中存放关于学生就业情况的信息，其中的 student 作为外键和学生表中的 id 进行关联，表中还包含信息是否被学生确认标记（confirmed）、就业状态（state）、就业日期（signDate）等必要的字段，就业日期存在就表示已经就业，程序会根据就业日期字段来更新就业状态。学生表与就业意向表和就业情况表都是一对一的关系。该部分的数据库模型如图 4.3 所示：

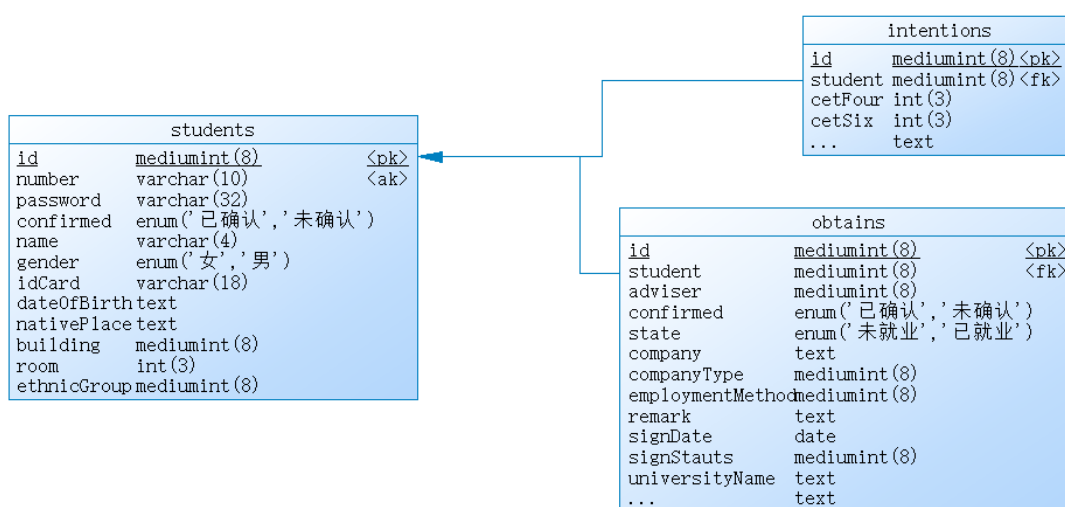


图 4.3 学生信息管理数据模型图

然后设计最为关键的字段管理模型，字段表（columns）负责存储系统中自定义的字段，包括学生自然信息中的字段、学生就业意向中的字段和学生就业情况中的字段，使用分类字段（category）进行区分。使用顺序字段（order）储存字段在数据表格中显示的顺序，数字越大显示越靠后。包括中文名称（name）和在代码中使用的标识名称（english），用于区分数据显示方式的字段类型（type），数据类型（dataType）字段储存的是存储该字段数据实际使用到的 MySQL 数据类型。其中还包括了字段是否为必填（required），字段是否不可编辑（noneditable），用于进行数据验证的正则表达式（validate），字段在数据表格中显示的宽度（width），用于在数据表中显示和编辑所用到的 formatter、editor 和 other 几个字段。

当字段类型（type）为单选（radio）时，需要另一个表（column_options）来存储可

供选择的选项，其中包括 id，作为外键和字段表关联的 column 字段，选项名称（name）和选项显示顺序（order）字段。字段选项表和字段表是一对多的关系，因为一个单选类型的字段可以对应多个选项。该部分的数据模型图如图 4.4 所示：

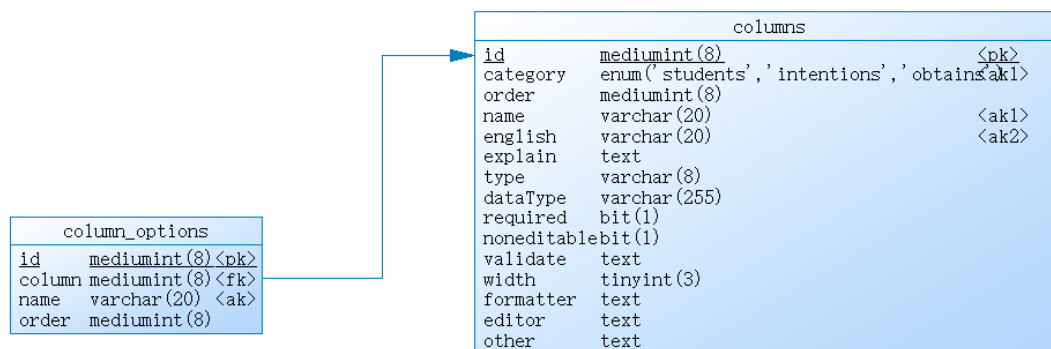


图 4.4 字段管理数据模型图

最后设计较为简单的一些表，毕业设计指导教师表（advisers）中包含毕业设计指导教师的基本信息，教研室表（sections）中包含教研室的基本信息，毕业设计指导教师表和教研室表是多对一的关系，用字段 section 作为外键进行关联。

用户表（users）保存了辅导员和管理员，用类型字段（type）进行区分。专业表（majors）储存学院各个年级的专业，包括入学年份字段（year），专业编号字段（number），专业名称字段（name），专业简称字段（abbreviation）和用于储存该专业开设了几个班级的 classes 字段，辅导员字段（instructor）作为外键和用户表进行关联。

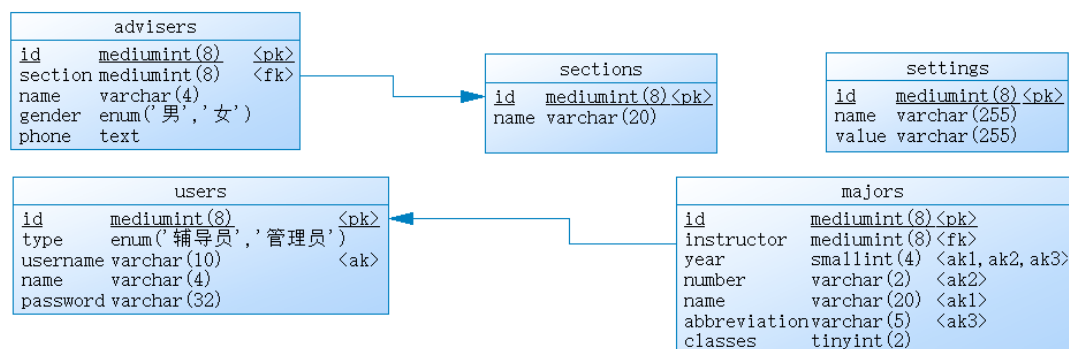


图 4.5 毕业设计指导教师、用户、系统设置数据模型图

4.4.2 数据表设计

有了数据模型，数据表可以使用工具直接生成，这里同样使用 Power Designer 工具进行。生成的数据表如下：


名	类型	长度	小数点	允许空值	
▶ id	mediumint	8	0	<input type="checkbox"/>	 1
name	varchar	4	0	<input type="checkbox"/>	
gender	enum	0	0	<input type="checkbox"/>	
section	mediumint	8	0	<input type="checkbox"/>	
phone	text	20	0	<input checked="" type="checkbox"/>	

图 4.6 毕业设计指导教师表 (adviser) 结构

名	类型	长度	小数点	允许空值	
▶ id	mediumint	8	0	<input type="checkbox"/>	 1
column	mediumint	8	0	<input type="checkbox"/>	
name	varchar	20	0	<input type="checkbox"/>	
order	mediumint	8	0	<input checked="" type="checkbox"/>	

图 4.7 字段选项表 (column_options) 结构

名	类型	长度	小数点	允许空值	
▶ id	mediumint	8	0	<input type="checkbox"/>	 1
category	enum	0	0	<input type="checkbox"/>	
order	mediumint	8	0	<input type="checkbox"/>	
name	varchar	20	0	<input type="checkbox"/>	
english	varchar	20	0	<input type="checkbox"/>	
explain	text	0	0	<input checked="" type="checkbox"/>	
type	varchar	8	0	<input type="checkbox"/>	
dataType	varchar	255	0	<input checked="" type="checkbox"/>	
required	bit	1	0	<input type="checkbox"/>	
noneditable	bit	1	0	<input type="checkbox"/>	
validate	text	0	0	<input checked="" type="checkbox"/>	
width	tinyint	3	0	<input checked="" type="checkbox"/>	
formatter	text	0	0	<input checked="" type="checkbox"/>	
editor	text	0	0	<input checked="" type="checkbox"/>	
other	text	0	0	<input checked="" type="checkbox"/>	

图 4.8 字段表 (column) 结构

名	类型	长度	小数点	允许空值	
▶ id	mediumint	8	0	<input type="checkbox"/>	 1
student	mediumint	8	0	<input type="checkbox"/>	
cetFour	int	3	0	<input checked="" type="checkbox"/>	
cetSix	int	3	0	<input checked="" type="checkbox"/>	

图 4.9 就业意向表 (intentions) 结构


名	类型	长度	小数点	允许空值	
▶ id	mediumint	8	0	<input type="checkbox"/>	 1
year	smallint	4	0	<input type="checkbox"/>	
number	varchar	2	0	<input type="checkbox"/>	
name	varchar	20	0	<input type="checkbox"/>	
abbreviation	varchar	5	0	<input type="checkbox"/>	
classes	tinyint	2	0	<input type="checkbox"/>	
instructor	mediumint	8	0	<input checked="" type="checkbox"/>	

图 4.10 专业表 (majors) 结构


名	类型	长度	小数点	允许空值	
▶ id	mediumint	8	0	<input type="checkbox"/>	 1
adviser	mediumint	8	0	<input checked="" type="checkbox"/>	
student	mediumint	8	0	<input type="checkbox"/>	
confirmed	enum	0	0	<input checked="" type="checkbox"/>	
state	enum	0	0	<input checked="" type="checkbox"/>	
company	text	0	0	<input checked="" type="checkbox"/>	
companyType	mediumint	8	0	<input checked="" type="checkbox"/>	
employmentMethod	mediumint	8	0	<input checked="" type="checkbox"/>	
remark	text	0	0	<input checked="" type="checkbox"/>	
signDate	date	0	0	<input checked="" type="checkbox"/>	
signStauts	mediumint	8	0	<input checked="" type="checkbox"/>	
universityName	text	0	0	<input checked="" type="checkbox"/>	

图 4.11 就业情况表 (obtains) 结构

名	类型	长度	小数点	允许空值	
▶ id	mediumint	8	0	<input type="checkbox"/>	 1
name	varchar	20	0	<input checked="" type="checkbox"/>	

图 4.12 教研室表 (sections) 结构


名	类型	长度	小数点	允许空值	
▶ id	mediumint	8	0	<input type="checkbox"/>	 1
name	varchar	255	0	<input type="checkbox"/>	
value	varchar	255	0	<input checked="" type="checkbox"/>	

图 4.13 设置表 (settings) 结构

名	类型	长度	小数点	允许空值	
▶ id	mediumint	8	0	<input type="checkbox"/>	 1
number	varchar	10	0	<input type="checkbox"/>	
password	varchar	32	0	<input checked="" type="checkbox"/>	
confirmed	enum	0	0	<input checked="" type="checkbox"/>	
name	varchar	4	0	<input type="checkbox"/>	
gender	enum	0	0	<input type="checkbox"/>	
idCard	varchar	18	0	<input checked="" type="checkbox"/>	
dateOfBirth	text	0	0	<input checked="" type="checkbox"/>	
nativePlace	text	0	0	<input checked="" type="checkbox"/>	
building	mediumint	8	0	<input checked="" type="checkbox"/>	
room	int	3	0	<input checked="" type="checkbox"/>	
ethnicGroup	mediumint	8	0	<input checked="" type="checkbox"/>	

图 4.14 学生表 (students) 结构

名	类型	长度	小数点	允许空值	
▶ id	mediumint	8	0	<input type="checkbox"/>	 1
type	enum	0	0	<input checked="" type="checkbox"/>	
username	varchar	10	0	<input type="checkbox"/>	
name	varchar	4	0	<input type="checkbox"/>	
password	varchar	32	0	<input type="checkbox"/>	

图 4.15 用户表 (users) 结构

5 就业管理系统详细设计

5.1 界面设计

登录页面的设计使用 Semantic UI 前端框架实现，该框架的网格系统（Grid System）给页面布局带来了很大的方便，登录采用 Ajax 技术，登录的错误信息抖动显示，便于用户察觉。

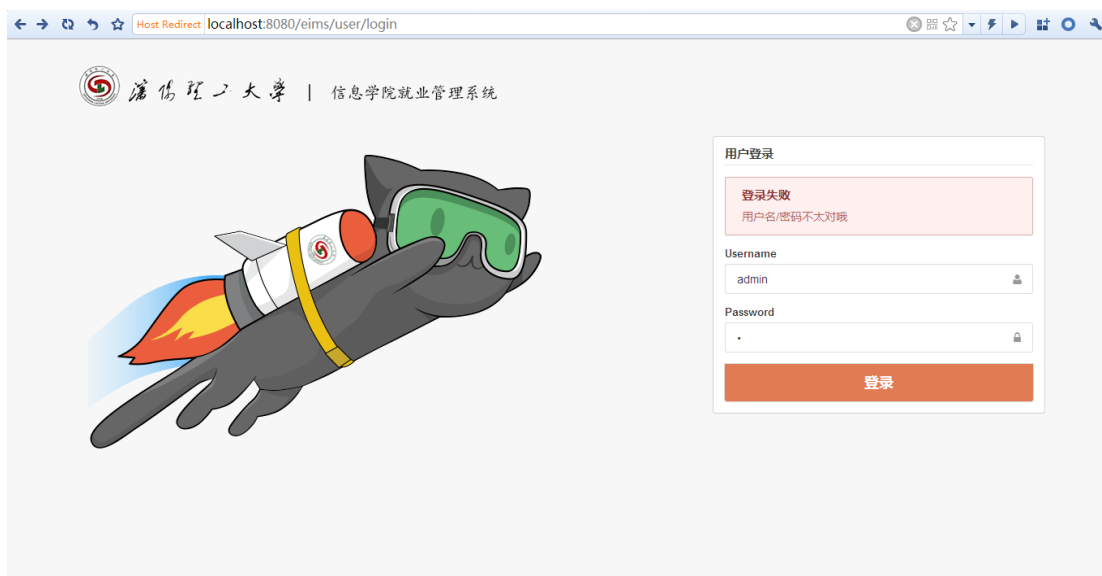


图 5.1 登录页面

辅导员和管理员登录之后进入管理后台，后台使用 EasyUI 设计，整个页面的 body 标签使用 EasyUI 的布局（Layout）功能，顶部、底部、左侧导航菜单和体部分分别设置 region 属性为北（north）、南（south）、西（west）、东（east）。左侧菜单引入了 left.jsp 文件，然后使用 EasyUI 的标签（Tabs）功能在页面的主体部分打开相应的页面。

```
<body class="easyui-layout">
    <div
                                data-options="region:'north',border:false"
style="height:60px;background:#f7f7f7;padding:10px;overflow:hidden">
        
        <div style="float:right">
```

```

        <a href="#" id="btn-change-password">修改密码</a> | <a href="#"
id="btn-logout">注销</a>
    </div>
</div>
<div id="left" data-options="region:'west',split:true,title:' 功 能 列 表 '"
style="width:220px;padding:10px">
    <jsp:include page="/left.jsp"></jsp:include>
</div>
<div data-options="region:'center'">
    <div id="main-tabs" class="easyui-tabs" data-options="fit:true,border:false">
        <div title="欢迎" style="padding:10px">
            <iframe id="main-iframe"
src="http://www.yutouxiuxiu.com/products/eims?pdctid=${productId}&id=${sessionScope.u
ser.id}&name=${sessionScope.user.name}&type=${sessionScope.user.type}"
frameborder="0" width="100%" height="100%"></iframe>
        </div>
    </div>
</div>
<div data-options="region:'south',border:false"
style="height:50px;background:#A9FACD;padding:10px;color:#888888">
    <p>沈阳理工大学信息科学与工程学院学生就业信息管理系统</p>
    <p>Powered by XiuXiu. &lt;yutouxiuxiu@126.com&gt; | v1.2.5</p>
</div>
</body>

```



图 5.2 管理后台首页

学生信息管理页面主要是使用了 EasyUI 的表单组件 (Form) 做信息筛选功能, 使用数据表格组件 (Datagrid) 和可编辑数据表格插件 (eDatagrid) 做数据的显示, 使得数据可以行内编辑, 无需打开新的页面, 方便快捷易于操作, 只需双击一条数据, 后者点击上方的编辑选中按钮即可对某一条数据进行编辑。各个模块的管理页面基本相似, 在这里就不一一赘述。

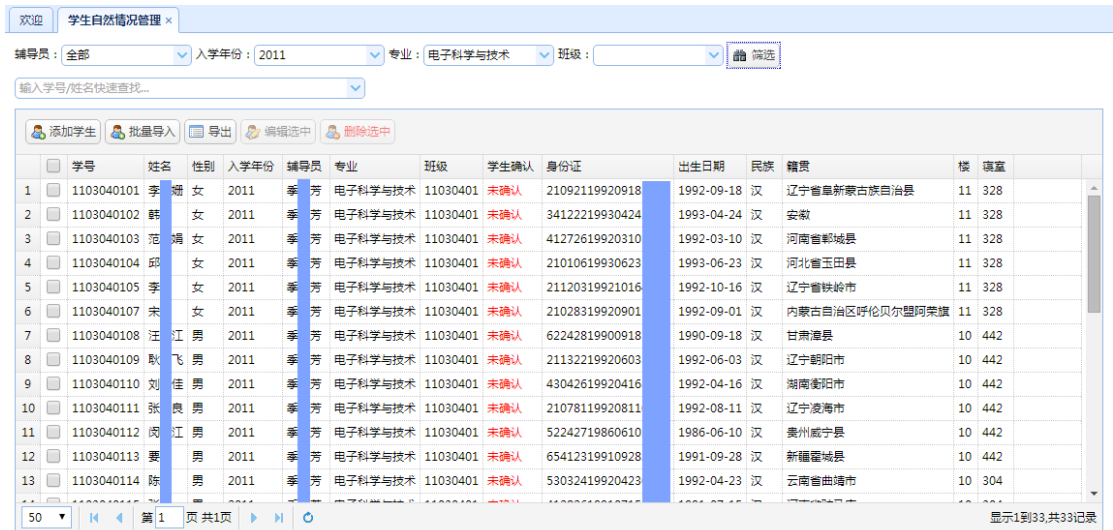


图 5.3 学生信息管理界面

	班级	学生确认	身份证	出生日期	民族	籍贯	楼	寝室	
技术	11030401	未确认	21092119920918	11	请以正确的格式输入	辽宁省阜新蒙古族自治县	11	328	保存 取消
技术	11030401	未确认	34122219930424	1993-04-24	汉	安徽	11	328	
技术	11030401	未确认	41272619920310	1992-03-10	汉	河南省郸城县	11	328	
技术	11030401	未确认	21010619930623	1993-06-23	汉	河北省玉田县	11	328	
技术	11030401	未确认	21120319921016	1992-10-16	汉	辽宁省铁岭市	11	328	

图 5.4 数据行内编辑

前台页面只用于学生编辑和确认数据，所以并不需要在一个屏幕内显示很多数据，使用 Semantic UI 进行设计更加美观。这里主要用到了 Semantic UI 中的菜单组件（Menu）、面板组件（Segment）和表单组件（Form），表单组件具有很好的表单验证功能，使开发变得更加便捷，只需在框架中注册验证规则即可。

图 5.5 前台界面

5.2 学生信息管理模块设计

学生信息管理模块中包含了学生自然信息管理、学生就业意向管理和学生就业情况管理，还分为前台和后台两个部分，其中包括了学生自然信息确认和学生就业情况确认等两个业务流程，本节会详细阐述该部分的设计思路。

本模块包含两个控制器类，它们分别是 StudentController 和 EmploymentController，StudentController 处理学生自然信息的增删改查、学生数据的导入和导出还有上述的两个业务流程的处理。

StudentController 继承了 BaseController，在 BaseController 中包括了将从数据库中查询到的数据转化成 EasyUI 中 Datagrid 组件能够识别的格式的方法、进行数据验证的方法，由于系统中的字段是可以动态修改的，所以还包括防止注入漏洞的字段过滤方法。

首先将 StudentController 映射到 URI “/student”，并通过注解@Autowired 自动注入依赖的 Service。

```
@Controller
@RequestMapping("/student")
public class StudentController extends BaseController {

    @Autowired
    private StudentService studentService;

    @Autowired
    private EmploymentService employmentService;

    @Autowired
    private ColumnService columnService;
```

5.2.1 前台部分

1、请求的处理

学生用学号和身份证号作为密码登录到前台之后，系统首先强制用户修改自己的密码，这样有助于系统的安全性。



菜单

- 个人信息
- 就业意向
- 就业情况
- 修改密码
- 退出

修改密码

有一些错误哦
请先重新设置您的密码！

原密码：

新密码：

确认密码：

修改

图 5.6 系统强制要求用户修改密码

登录之后，页面自动跳转到“student/home”，使用@RequestMapping 可以将 URI 映射到 Action 方法上，SpringMVC 框架会自动调用执行。value 表示映射的 URL，method 是映射的是请求的方法，这里为 GET 方式。Action 方法可以被框架自动传入 Model 和 HttpSession 对象，Model 对象中的数据可以被框架自动传入返回的 Jsp 页面中。在本方法中，先判断学生是否已经设置密码，如果没有设置，则强制转到密码修改页面并提示用户需要先设置密码。

```
@Privilege(User.USER_TYPE_STUDENT)
```

```
@RequestMapping(value = "/home", method = RequestMethod.GET)
```

```
public String studentBasicInfo(Model model, HttpSession session) {
```

```
    if (!checkIfSetPassword(session)) {  
        model.addAttribute("error", true);  
        model.addAttribute("message", "请先重新设置您的密码！");  
        model.addAttribute("page", "changepassword");  
        return "student/home";  
    }
```

```
    model.addAttribute("page", "home");
```

```

        model.addAttribute("columns",
columnService.getColumnList(ColumnService.CATEGORY_STUDENT));
        model.addAttribute("student", studentService.getStudent(((User)
session.getAttribute("user")).getUsername()));
        model.addAttribute("confirm", checkIfConfirmed(session));

        return "student/home";
    }

```

2、前端的显示

为了代码的重用，便于后期的重构，学生修改自然情况、就业意向、就业情况的 Jsp 页面都是一个文件，通过文件中的库标签来进行判断和显示。例如在前台页面左侧的导航菜单就是采用这种技术来实现的，判断 Model 对象中 page 变量的值，如果是响应的页面，就给 a 标签加上 active 样式，则该菜单项会显示被选中的效果：

```

<div class="ui vertical menu" style="width: 17rem">
    <div class="header item">
        菜单
    </div>
    <a href="<%=request.getContextPath()%>/student/home" class="<c:if test="{page
== 'home'}">active </c:if>teal item">
        个人信息
        <c:if test="{confirm.student == '未确认'}"><div class="ui label">待确认
    </div></c:if>
    </a>
    <a href="<%=request.getContextPath()%>/student/intention" class="<c:if
test="{page == 'intention'}">active </c:if>teal item">
        就业意向
    </a>
    .....

```

页面上的字段是根据 Model 传来的不同的类别（page 变量中提供）用 foreach 标签进行显示的，<c:forEach>标签遍历了 Model 类中的 columns 变量（StudentController 中的 studentBasicInfo 方法调用 Service 类中的方法从数据库中查询得到），在循环体内再使用<c:choose>标签配合<c:when>标签来区分不同类型字段的编辑方式，文本字段和文本域字段都是用 text 类型的 input 标签显示，单选使用的是 select 和 option 标签显示，对于日期类型的和不可编辑的字段也做了响应的处理：

```

<c:forEach items="${columns}" var="column" varStatus="status">
    <div class="<c:if test="${column.required}">required </c:if><c:if
test="${column.type == 'date'}">date </c:if>field">
        <label>${column.name}: </label>
        <div class="ui input">
            <c:choose>
                <c:when test="${column.type == 'text' || column.type == 'textarea'}">
                    <input type="text" name="${column.english}"
placeholder="${column.name}" value="${student[column.english]}">
                </c:when>

                <c:when test="${column.type == 'date'}">
                    <input type="text" name="${column.english}"
placeholder="yyyy-mm-dd" value="${student[column.english]}">
                </c:when>

                <c:when test="${column.type == 'radio'}">
                    <select name="${column.english}" class="ui dropdown">
                        <option value="">&nbsp;</option>
                        <c:forEach items="${column.options}" var="option">
                            <option <c:if test="${student[column.english]} ==
option.id}">selected="selected" </c:if>value="${option.id}">${option.name}</option>
                        </c:forEach>
                    </select>

```

```
        </c:when>
    </c:choose>
</div>
</div>
</c:forEach>
```

对于表单动态验证，则是使用到了 SemanticUI 中提供的 validation 组件，扩展 \$.fn.form.settings.rules 变量即可向框架中添加验证方法，这里遍历了所有字段，并将带有正则表达式的字段添加到验证规则当中。

```
$.fn.form.settings.rules = $.extend($.fn.form.settings.rules, {
    <c:forEach items="${columns}" var="column">
        <c:if test="${column.validate != null && fn.trim(column.validate) != ""}>
            rule${column.english}: function(value) {
                return /${column.validate}/i.test(value);
            },
        </c:if>
    </c:forEach>
});
```

之后在初始化表单的时候，给表单中的各个需要验证的字段加入验证规则，这样在点击提交按钮的时候就可以由 SemanticUI 完成自动验证了。

```
$('.ui.form').form({
    <c:forEach items="${columns}" var="column" varStatus="status">
        <c:if test="${column.validate != null && fn.trim(column.validate) != ""}>
            ${column.english}: {
                identifier : '${column.english}',
                rules: [
                    {
                        type : 'rule${column.english}',
                        prompt : '请用正确的格式输入'
                    }
                ]
            }
        </c:if>
    </c:forEach>
});
```

```

        }
    ]
    },
</c:if>
</c:forEach>
}, {
    inline: true,
    on: 'blur'
});

```

3、后端验证与储存

当然，为了安全起见，防止黑客绕过前端直接编造 POST 请求，除了前端验证这里还实现了后端验证。下面的方法接收了上面表单提交的数据，使用 `HttpServletRequest` 类可以接收表单中的数据，首先是输入的参数进行过滤，将不是系统中的字段以及不允许用户更改的保密字段过滤掉，然后再对字段使用字段表中保存的正则表达式进行验证，接着将信息确认状态（`confirmed`）变为“已确认”，完成了数据确认的业务流程，最后调用 `Service` 里的方法将数据保存到数据库中。

```

Map<String, String> parameterMap =
filterStudentColumnParameterMap(ColumnService.CATEGORY_STUDENT,
request.getParameterMap());

parameterMap.put("confirmed", "已确认");

try {
    validateColumn(ColumnService.CATEGORY_STUDENT, parameterMap);
    studentService.updateStudentInline(parameterMap);
} catch (StudentException e) {
    model.addAttribute("error", true);
    model.addAttribute("message", e.getMessage());
}

```

上文中体现的是学生自然信息的实现思路，学生就业意向、就业情况和密码修改与其相似，在这里就不在一一赘述。

5.2.2 后台部分

1、前端的显示

后台部分和前台部分类似，也是学生自然情况、就业意向和就业情况三个页面共用一个 Jsp 页面，然后通过 Model 类中的 category 变量进行区分。与前台部分的不同之处，就在于前台只显示登录学生的信息，后台则可以显示该辅导员所管辖的全部同学（管理员可以查询全学院的信息），而且后台中可以对學生进行筛选和查询，所以后台的页面是使用了 EasyUI 中的 Datagrid 实现的，Datagrid 向后端请求的数据格式如下：

```
{
    page: 当前页数,
    rows: 当前页数据条数,
    其他参数...
}
```

其接收的 Ajax 响应数据数据格式如下：

```
{
    rows: [
        {当前分页的第 1 条数据},
        {当前分页的第 2 条数据},
        .....
    ],
    total: 数据总条数
}
```

这样处理是为了数据表格的分页，total 中返回的数据表示数据的总条数，rows 中的数据为当前页显示的全部数据，Datagrid 组件会自动计算分页的情况并予以显示。该部分的前端代码设计思路与前台相似，只是使用的框架不同，所以在这里就不详细说明了。

2、数据筛选

这部分的难点在于数据的筛选，由于用户的筛选条件的数量不确定（例如，可以只筛选入学年份，也可以既筛选入学年份又筛选专业，甚至筛选班级），这里采用了动态

拼接 SQL 语句的方式实现。以查看学生自然情况列表为例,该函数(`buildStudentsListSql`)实现了 SQL 语句的拼接,在 `WHERE` 字句中,判断是否填写了某个筛选条件,如果没有填写则不会出现在 SQL 语句中。

```
private String buildStudentsListSql(Integer schoolYear, Integer instructor, Integer major,
Integer clazz, Integer id) {
    String sql = "SELECT s.*, m.year schoolyear, u.name instructor, m.name major,
SUBSTRING(s.number, 1, 8) class"
        + " FROM students s"
        + " LEFT JOIN majors m ON (m.number = SUBSTRING(s.number, 5, 2) AND
SUBSTRING(m.year, 3, 2) = SUBSTRING(s.number, 1, 2))"
        + " LEFT JOIN users u ON (u.id = m.instructor)";
    sql += " WHERE 1 = 1";
    if (schoolYear != null) sql += " AND SUBSTRING(s.number, 1, 2) = ?";
    if (instructor != null) sql += " AND u.id = ?";
    if (major != null) sql += " AND m.id = ?";
    if (clazz != null) sql += " AND SUBSTRING(s.number, 7, 2) = ?";
    if (id != null) sql += " AND s.id = ?";
    sql += " ORDER BY s.number ASC";

    return sql;
}
```

为了防止 SQL 注入漏洞,这里采用了 `PreparedStatement` 进行数据查询,它将 SQL 语句进行预先编译,然后再传入参数,这样有效地防止了黑客对系统的恶意攻击。向 `JdbcTemplate` 类中的 `queryForList` 传入 `Object` 的数组,就使用到了 `PreparedStatement`。

```
jdbcTemplate.queryForList(sql, buildParams(schoolYear, instructor, major, clazz, id));
```

3、数据批量导入

在本系统中,数据的批量导入使用到了 Apache 软件基金会的开放源码函式库 POI, POI 提供了 API 给 Java 程序对 Microsoft Office 格式档案读和写的功能。通过其中的 HSSF

（Horrible SpreadSheet Format）组件，可以用纯 Java 代码来读取、写入、修改 Excel 文件。

为了方式权限越界，导入之前需要进行学号检查，首先从数据库中查出该辅导员所管辖的全部专业，并生产一个存有该辅导员允许操作的学号前 8 位的数组，再对学号进行检查。数据导入的流程图如下所示：

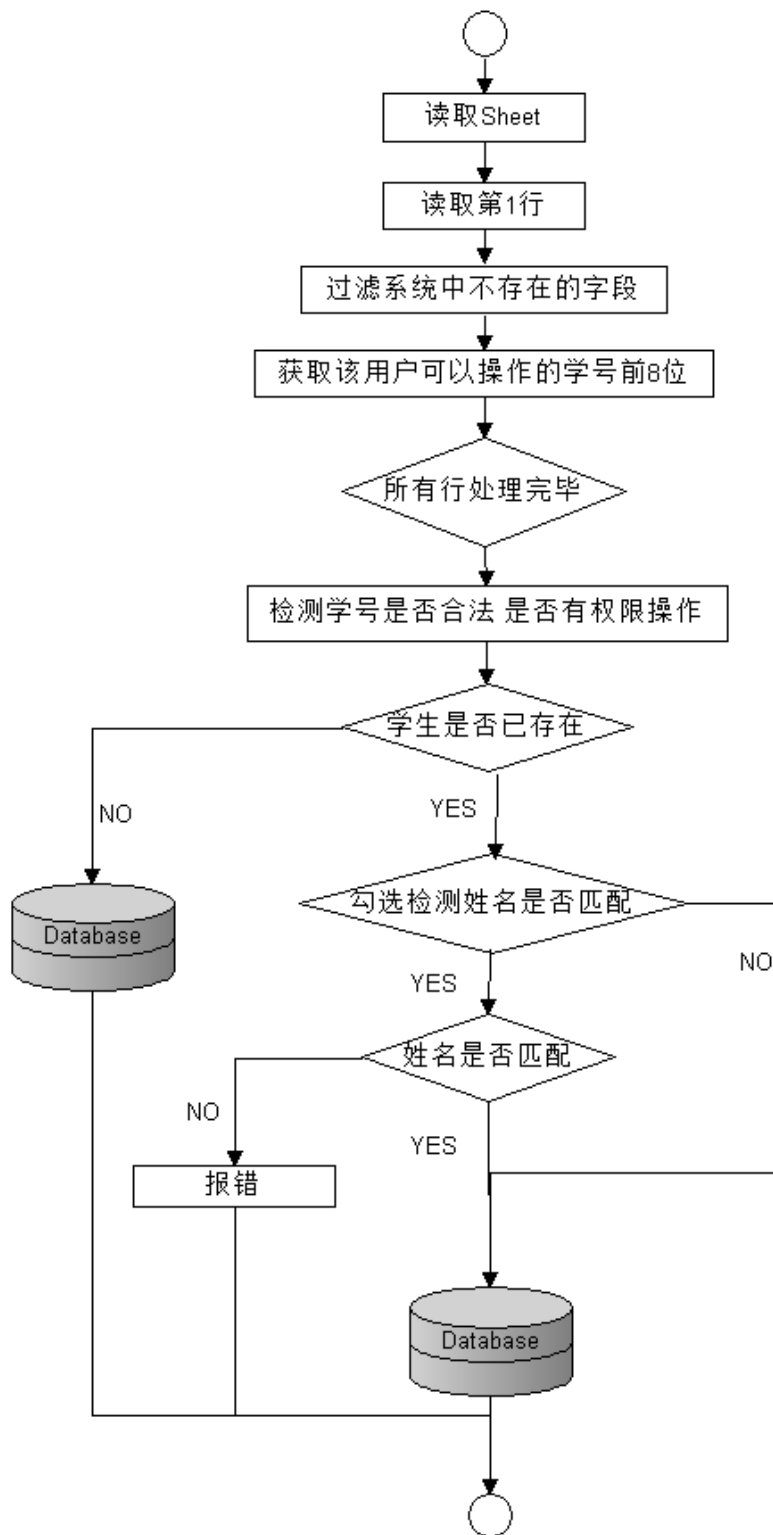


图 5.7 数据导入流程图

5.3 字段管理模块设计

为了系统具有良好的可扩展性，设计了动态管理字段的功能，管理员使用该功能可以在系统内添加想要的字段，包括学生自然信息字段、学生就业意向字段和学生就业情况字段。添加字段的页面如图 5.8 所示：

基本信息

字段名称:

变量名称:

说明文字:

字段内容

字段类别:

MySQL数据类型:

非空: ☐ 不可为空

可编辑: ☐ 不可编辑

验证规则:

图 5.8 添加字段界面

字段分为多种类型，文本框输入的字段可以设置它的验证规则，系统中有几个内置的规则，也可以由用户来自定义验证的正则表达式；单选字段需要添加选项，如图 5.9 所示：

字段内容

字段类别:

选项:

选项	排序编号
该输入项为必填项	1

非空: ☐ 不可为空

图 5.9 添加单选字段

系统处理添加字段时，先尝试在对应的表中按照指定的数据类型添加字段（添加自然信息字段，则在 `students` 表中添加字段；添加就业意向字段，则在 `intentions` 表中添加字段；添加就业情况字段，则在 `obtains` 表中添加字段），添加成功之后将该字段的信息保存在 `columns` 表中（如果是单选类型，则还要在 `column_options` 中插入记录）以便系统的显示。

5.4 统计报表模块设计

统计报表模块中主要使用了 Highcharts 框架来生成统计图标，这里使用了饼图和柱状图。

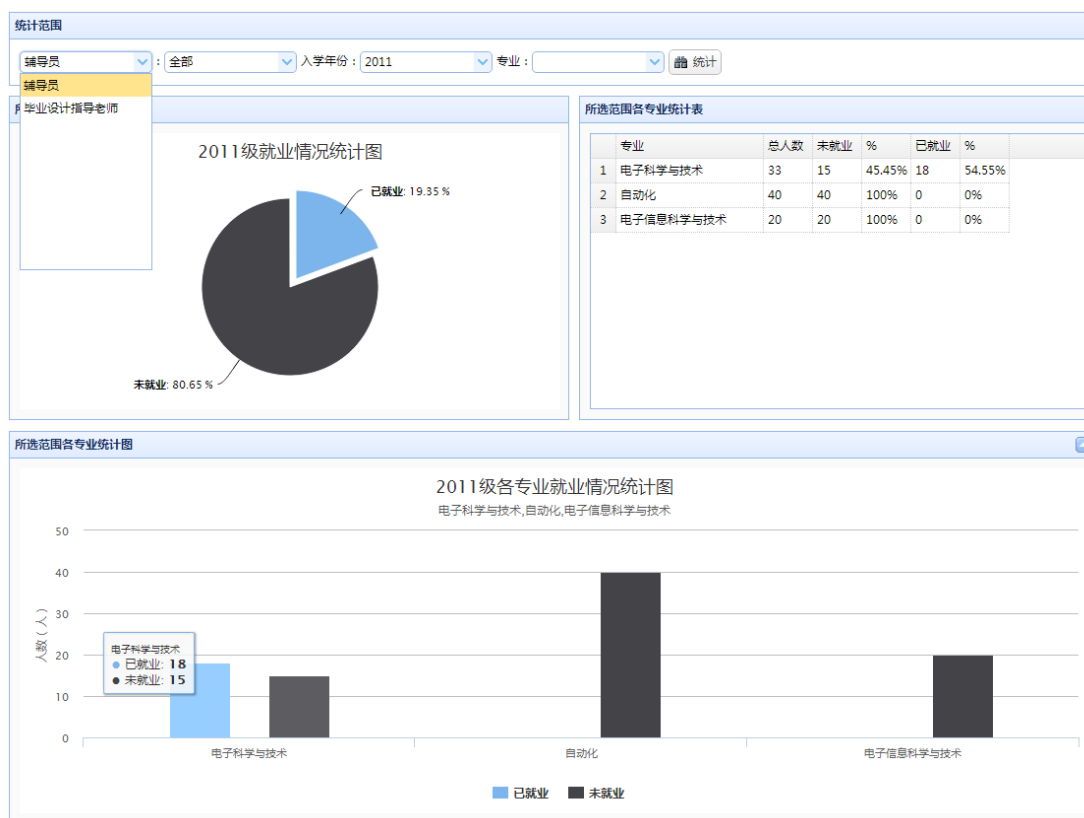


图 5.10 就业率报表页面

这部分的难点在于 SQL 查询语句，以所查范围内的整体就业率为例，在查询语句中，将已就业用 1 表示，未就业用 0 表示，然后通过 SUM 函数计算该列的值，即是就业人数，反之为未就业人数，用 COUNT 函数计算总的的数据条数，据此便可求得就业率和未就业率了。生成 SQL 语句的方法在 `StatementService` 中，如下所示：


```
private String buildOverallEmploymentRateSql(Integer schoolyear, String by, Integer
teacher, Integer major, String clazz, Boolean gender) {
    String sql = "SELECT SUM(CASE WHEN state = '已就业' THEN 1 ELSE 0
END) employed, "
        + "SUM(CASE WHEN state = '已就业' THEN 0 ELSE 1 END)
unemployed, "
        + "COUNT(s.id) `all`, (SUM(CASE WHEN state = '已就业' THEN 1 ELSE
0 END) / COUNT(s.id)) employed_rate, "
        + "(SUM(CASE WHEN state = '已就业' THEN 0 ELSE 1 END) /
COUNT(s.id)) unemployed_rate, "
        + "CONCAT('20', SUBSTRING(s.number, 1, 2)) year, "
        + "SUBSTRING(s.number, 5, 4) major_class, "
        + "SUBSTRING(s.number, 7, 2) clazz, "
        + "s.gender, m.name major "
        + "FROM students s "
        + "LEFT JOIN obtains o ON (o.student = s.id) "
        + "LEFT JOIN advisers a ON (a.id = o.adviser) "
        + "LEFT JOIN majors m ON (m.number = SUBSTRING(s.number, 5, 2) "
        + "AND SUBSTRING(m.year, 3, 2) = SUBSTRING(s.number, 1, 2)) "
        + "WHERE year = ?";
    if (major != null) sql += " AND m.id = ?";
    if (by.equals("adviser")) sql += " AND a.id = ?";
    return sql;
}
```

```
private String buildEmploymentRateSql(Integer schoolyear, String by, Integer
teacher, Integer major, String clazz, Boolean gender) {
    String sql = buildOverallEmploymentRateSql(schoolyear, by, teacher, major,
clazz, gender) +
        " GROUP BY";
```


```
if (by.equals("instructor")) {  
    if (major == null) {  
        sql += " m.id";  
    } else {  
        sql += " clazz";  
    }  
} else {  
    sql += " a.id";  
}  
if (gender != null && gender) {  
    sql += ", s.gender";  
}  
sql += " HAVING (s.gender != "")";  
  
return sql;  
}
```

5.5 毕业设计指导教师管理模块设计


该模块很简单，即是对毕业设计指导教师和教研室的增删改查，不存在复杂的 SQL 语句，也不存在复杂的业务逻辑，在此处只给出该模块的页面，如图 5.11 和 5.12 所示：



添加老师



编辑选中



删除选中

<input type="checkbox"/>	姓名	性别	教研室	电话	
1	<input type="checkbox"/> 和晓军	女	电子科学与技术		
2	<input type="checkbox"/> 崔旭晶	女	电子科学与技术		
3	<input type="checkbox"/> 张爽	女	电子科学与技术		
4	<input type="checkbox"/> 李汇明	女	电子科学与技术		
5	<input type="checkbox"/> 王玲	男	电子科学与技术		
6	<input type="checkbox"/> 鲁艳艳	女	电子科学与技术		
7	<input type="checkbox"/> 陈慕群	女	电子科学与技术		
8	<input type="checkbox"/> 马平全	男	电子科学与技术		

图 5.11 毕业设计指导教师管理页面

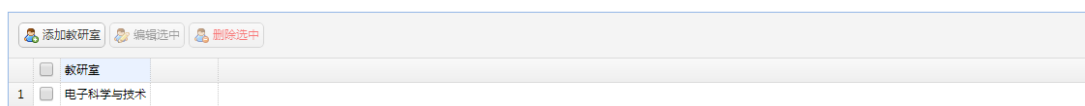


图 5.12 教研室管理页面

5.6 系统设置模块设计

系统设置模块包括了对学院编号的设置，用于核查学号是否合法，包含开学日期的设置，通过设置开学日期，系统可以自动生成新的学年的数据。还包括专业列表的设置，学生信息页面中专业、班级的显示都是从这里获得的。还包括了专业负责辅导员的设置，通过这个设置，系统可以划分每个辅导员可以管辖的范围，同样可以作为一个维度来统计就业率。这里也并不复杂，仅仅是一个表单，不过它提供了从上届复制的功能，解决了每年都需要维护专业列表的麻烦，点击一个按钮就可以将上一届的专业列表复制过来，如果有变化，只需进行简单的修改即可。



图 5.13 系统设置页面

5.7 用户权限模块设计

用户权限模块是该系统中尤为重要的部分，一旦有所疏忽就很有可能造成无法逆转的后果，该模块的设计使用到了 Spring 框架中的 AOP 技术（Aspect Oriented Programming），即面向切面编程。通过预编译方式和运行期动态代理实现程序功能的统

一维护的一种技术。AOP 是 OOP 的延续，是软件开发中的一个热点，也是 Spring 框架中的一个重要内容，是函数式编程的一种衍生范型。利用 AOP 可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。AOP 技术主要用于日志记录，性能统计，安全控制，事务处理，异常处理等，将它们的代码从业务逻辑代码中划分出来，通过对这些行为的分离，我们希望能将它们独立到非指导业务逻辑的方法中，进而改变这些行为的时候不影响业务逻辑的代码。

用户权限的控制使用了环绕切面（Around），该种切面可以在某方法执行之前执行，并且可以控制主体方法执行与否。在该系统中还定义了 Privilege 注解，用于在 Controller 类中声明各个 Action 方法所需要的用户权限，该注解的定义如下：

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@Inherited
public @interface Privilege {

    public String value() default User.USER_TYPE_INSTRUCTOR;
}
```

该注解是运行时的注解，表明在编译中不会将该注解去掉。该注解的目标是 METHOD，也就是用于在方法之上的。不使用注解表示任何人都可以访问，如果只使用注解，不给出方法所需的权限，默认为辅导员可以访问。

该切面定义在 PrivilegeAspect 中，首先获得主方法，即在切面执行之后要执行的方法，然后获取主方法上的注解，取得该注解所需的权限，最后从 Session 中提取出用户的登录数据，将用户类型和所需权限进行比对，如果通过验证就执行主方法，否则就抛出权限不足异常。

```
String value = null;
Object result = null;

// get method
Method[] methods = proceedingJoinPoint.getTarget().getClass().getMethods();
```



```
Method method = null;
for (Method m : methods) {
    if (m.getName().equals(proceedingJoinPoint.getSignature().getName())) {
        method = m;
    }
}

// get privilege annotation
if (method.isAnnotationPresent(Privilege.class)) {
    Privilege privilege = method.getAnnotation(Privilege.class);
    value = privilege.value();
}

if (value == null) {
    return proceedingJoinPoint.proceed();
}

// get session
Object[] args = proceedingJoinPoint.getArgs();
HttpSession session = null;
for (Object arg : args) {
    if (arg instanceof HttpSession) {
        session = (HttpSession) arg;
    }
}

// check
if (session.getAttribute("user") != null) {
    User user = (User) session.getAttribute("user");
```

```

        if (value.equals(User.USER_TYPE_ADMIN) &&
user.getType().equals(User.USER_TYPE_ADMIN)) {
            // admin
            result = proceedingJoinPoint.proceed();
        } else if (value.equals(User.USER_TYPE_INSTRUCTOR) &&
            (user.getType().equals(User.USER_TYPE_INSTRUCTOR) ||
user.getType().equals(User.USER_TYPE_ADMIN))) {
            // instructor or admin
            result = proceedingJoinPoint.proceed();
        } else if (value.equals(User.USER_TYPE_STUDENT) &&
            user.getType().equals(User.USER_TYPE_STUDENT)) {
            // student
            result = proceedingJoinPoint.proceed();
        } else {
            throw new
PrivilegeException(PrivilegeException.PERMISSION_DENIED);
        }
    } else {
        throw new PrivilegeException(PrivilegeException.NEED_LOGIN);
    }

    return result;

```

5.8 异常处理机制

在该系统中，异常是统一处理的，无论是系统异常还是用户异常全部向上抛出，最后使用注册在 SpringMVC 框架中的 `ExceptionHandler` 进行处理，`ExceptionHandler` 继承了 Spring 框架提供的 `SimpleMappingExceptionHandler` 类，实现该类的 `doResolveException` 方法即可对异常进行处理。异常的处理流程如下图所示：

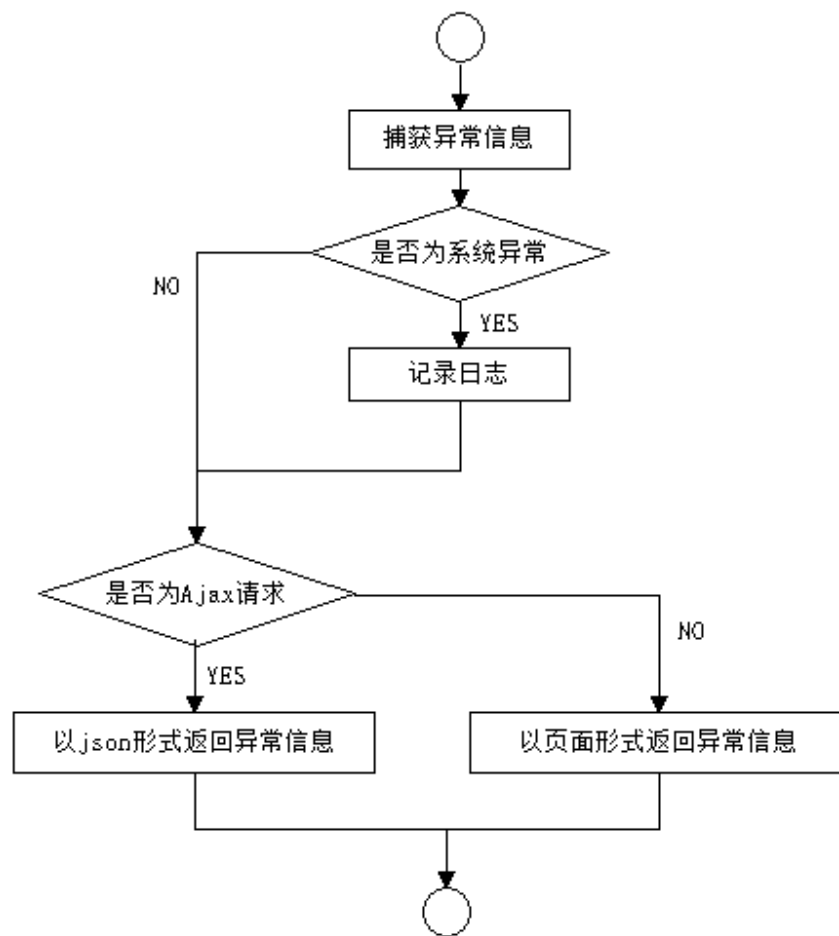


图 5.14 异常处理流程图

6 系统的测试与部署

6.1 系统的测试

系统测试是确保系统质量的关键部分，是对系统设计、编码的最终检查。系统测试就是根据实际需要编写或使用各种不同的测试工具，根据事先制定好的测试方案及测试用例，对软件系统的功能和性能进行检测。

6.1.1 测试的目的

测试的主要目的是验证软件需求和功能是否得到完整实现，其次是验证软件在正常和非正常情况下的功能和特性。测试不仅要验证软件在正常情况下的功能和特性是否可以使用和达到期望值，更多的是验证在非正常情况下功能和特性能否达到期望的要求。软件产品的质量信息也必须通过测试才能获取，没有经过测试的软件，软件质量的好坏是无从知道的，最多只能根据开发人员的水平进行推测。经过测试后，就可以得到开发各阶段发现的缺陷数，进而可以较为准确地推测出软件潜在的缺陷数。

6.1.2 测试的方法

测试的方法有很多，可以按照不同的维度进行划分。

1、根据项目流程划分

- 1) 单元测试：单元测试是对软件中的基本组成单位进行的测试。目的是检验软件基本组成单位的正确性。
- 2) 集成测试：集成测试是在软件系统集成过程中所进行的测试。目的是检查软件单位之间的接口是否正确。
- 3) 系统测试：系统测试是对已经集成好的软件系统进行彻底的测试，以验证软件系统的正确性和性能等是否满足其规约所指定的要求。
- 4) 验收测试：验收测试是部署软件之前的最后一个测试操作。验收测试的目的是确保软件准备就绪，向软件购买都展示该软件系统满足其用户的需求。

2、根据是否对代码可见

- 1) 黑盒测试：指的是把被测的软件看作是一个黑盒子，我们不去关心盒子里面的结构是什么样子的，只关心软件的输入数据和输出结果。它只检查程序功能是

否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息。黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试。

- 2) 白盒测试：也被叫做结构测试或者逻辑驱动测试，在系统检测时会对程序内部逻辑结构、逻辑路径进行测试。检测人员通过阅读程序代码或者通过使用开发工具中的单步调试来判断系统软件的质量。

3、根据是否使用自动化测试工具

- 1) 手工测试：手工测试就是由人去一个一个的去执行测试用例，通过键盘鼠标等输入一些参数，查看返回结果是否符合预期结果。
- 2) 自动化测试：自动化测试是把以人为驱动测试行为转化为机器执行的一种过程。通常，在设计了测试用例并通过评审之后，由测试人员根据测试用例中描述的规程一步步执行测试，得到实际结果与期望结果的比较。在此过程中，为了节省人力、时间或硬件资源，提高测试效率，便引入了自动化测试的概念。

4、其他测试

- 1) 回归测试：回归测试是指修改了旧代码后，重新进行测试以确认修改没有引入新的错误或导致其他代码产生错误。回归测试一般是在进行软件的第二轮测试开始的，验证第一轮中发现的问题是否得到修复。当然，回归也是一个循环的过程，如果回归的问题通不过，则需要开发人员修改后再次进行回归，直到通过为止。

6.1.3 该系统中采用的测试

在该系统的测试过程中，用到了多种类型的测试。

1、代码检查

在该系统中的白盒测试主要是通过开发人员在开发过程中的代码检查（Code Inspection）。代码最大的问题，不是一两个地方有技术缺陷，也不是业务逻辑错误，而是整个软件编写的不好。前两者都可以通过测试或使用来发现和更正，但后者就不同了。通过代码检查可以改正这种结构问题，是从编写可靠软件向编写精美软件迈进的重要方法。具体结构问题包括：重复拷贝代码（不封装函数，不用 Template/泛型），函数过长（超过一屏幕），错误封装（不恰当的 public/不用 Interface/不内聚/强耦合/在类中封装了无关方法等），内容错误（多个无关类置于一个文件/不恰当的命名等）等等。除此之外，

代码检查还可以修正业务逻辑问题，就是软件是否与需求的要求符合的问题。虽然在后期的测试过程中可以发现这种问题，可是测试一般发生在很久以后，有些逻辑测试还需要一定的触发条件，而且测试只会发现失效（**failure**，与预期不符）而不能发现缺陷（**defect**，具体哪里出了错），日积月累，就很难找到原因。

2、自动化回归测试

由于项目庞大，为了代码重用，抽取了很多公共的代码，但是在开发一个功能的时候，可能会发现公共代码的错误，此时更改底层代码可能导致之前可用的功能不可用，因此需要进行回归测试。由于回归测试会多次重复使用，这里用到了自动化测试工具 **Selenium**，**Selenium** 是一个用于 **Web** 应用程序测试的工具。**Selenium** 测试直接运行在浏览器中，就像真正的用户在操作一样。支持的浏览器包括 **IE**、**Mozilla Firefox**、**Chrome** 等。这个工具的主要功能包括：测试与浏览器的兼容性——测试你的应用程序看是否能够很好得工作在不同浏览器和操作系统之上。测试系统功能——创建回归测试检验软件功能和用户需求。支持自动录制动作和自动生成。

首先使用 **Firefox** 的 **Selenium** 插件录制测试使用的脚本，然后设置生成 **Java** 代码的模板，之后生成 **Java** 代码，这样就得到了一个测试用例（**Test Case**）。

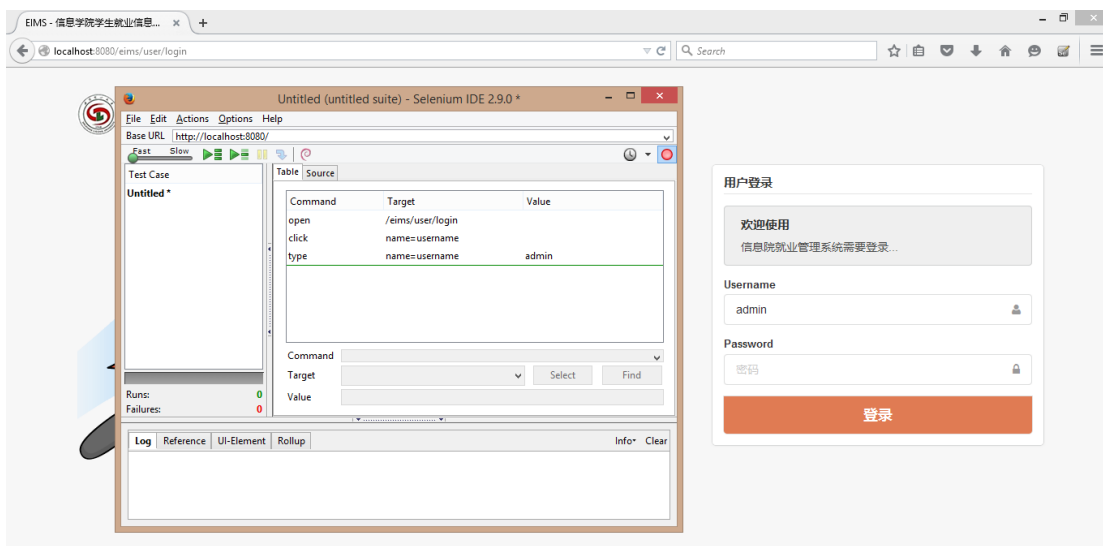


图 6.1 用 Selenium 录制测试脚本

以登录测试为例，下面是登录测试用例的代码，先打开/user/login，清除用户名表单项，输入 **admin**，清除密码表单项，输入 **admin**，点击登录按钮，等待新的页面打开之后使用断言验证是否有注销按钮。

```
public class LoginUseAdminAsPassword extends TestCase {

    private WebDriver driver = TestProject.driver;
    private String baseUrl = TestProject.baseUrl;
    private Logger logger = TestProject.logger;

    @Test
    public void testLoginUseAdminAsPassword() throws Exception {

        logger.info("Running testLoginUseAdminAsPassword");

        driver.get(TestProject.baseUrl + "/user/login");
        driver.findElement(By.name("username")).clear();
        driver.findElement(By.name("username")).sendKeys("admin");
        driver.findElement(By.name("password")).clear();
        driver.findElement(By.name("password")).sendKeys("admin");
        driver.findElement(By.id("btn-login")).click();
        assertEquals("注销", driver.findElement(By.id("btn-logout")).getText());
    }
}
```

在生产了多个测试用例之后，放在 src/test/java 中，建立一个 TestSuite，将它们有机地结合在一起，首先初始化 Chrome 浏览器的驱动，然后一一调用测试用例：

```
public class TestProject extends TestCase {

    public static WebDriver driver;
    public static String baseUrl;
    public static final Logger logger = Logger.getLogger(TestProject.class);

    static {
```

3、系统测试

6.2 系统的部署

在 JavaEE 项目开发中，通常部署过程包含以下步骤：

- 1) 检测是否有未提交到 SVN 的代码。
- 2) 检测是否有 SNAPSHOT 状态的依赖。
- 3) 将 POM 文件中的版本号提升。
- 4) 进行全部的代码测试。
- 5) 将 POM 文件提交到 SVN。
- 6) 将代码在 SVN 上创建 tag
- 7) 从 SVN 上检出所有的文件。
- 8) 构建项目，生成目标 war 包。
- 9) 发布项目到 Nexus 版本管理系统。
- 10) 将 war 包复制到 Tomcat 的 webapps 文件夹内。

在 Maven 中，可以使用 maven-release-plugin 插件来完成项目的自动化发布过程。先将代码检出（Checkout）到本地，然后在 trunk 目录中执行 mvn release:prepare 进行发布准备，发布准备过程中将进行代码测试等过程。等准备完成之后，继续执行 release:perform，代码将自动打包发送到 Nexus 服务器，打包的内容包括源文件、文档、war 包等等。

```
[INFO] [INFO] -----
[INFO] [INFO] Building eims 1.3.0
[INFO] [INFO] -----
[INFO] [INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ eims ---
[INFO] [INFO] Deleting D:\c\projects\edu.sylu\eims\eims\target
[INFO] [INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ eims ---
[INFO] [WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] [INFO] Copying 0 resource
[INFO] [INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ eims ---
[INFO] [INFO] Compiling 32 source files to D:\c\projects\edu.sylu\eims\eims\target\classes
[INFO] [INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ eims ---
[INFO] [WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] [INFO] Copying 1 resource
[INFO] [INFO] --- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) @ eims ---
[INFO] [INFO] Compiling 6 source files to D:\c\projects\edu.sylu\eims\eims\target\test-classes
[INFO] [INFO]
```

图 6.2 编译项目

```
[INFO] [INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ eims ---
[INFO] [INFO] Surefire report directory: D:\c\projects\edu.sylu\eims\eims\target\surefire-reports
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running com.yutouxiuxiu.eims.testsuite.TestProject
[INFO] Starting ChromeDriver (v2.9.248315) on port 40668
[INFO] 12:13:32,845 Running testLoginUseAdminAsPassword
[INFO] 12:14:21,033 Running testChangePasswordFromAdminTo123456
[INFO] 12:14:22,455 Running testLoginUse123456AsPassword
[INFO] 12:14:23,638 Running testChangePasswordFrom123456ToAdmin
[INFO] 12:14:24,824 Running testLoginUseAdminAsPassword
[INFO] 12:14:25,694 Running testLogout
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 57.555 sec
[INFO]
[INFO] Results :
[INFO]
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```

图 6.3 进行测试

```
[INFO] [INFO] --- maven-war-plugin:2.2:war (default-war) @ eims ---
[INFO] [INFO] Packaging webapp
[INFO] [INFO] Assembling webapp [eims] in [D:\c\projects\edu.sylu\eims\eims\target\eims]
[INFO] [INFO] Processing war project
[INFO] [INFO] Copying webapp resources [D:\c\projects\edu.sylu\eims\eims\src\main\webapp]
[INFO] [INFO] Webapp assembled in [11072 msecs]
[INFO] [INFO] Building war: D:\c\projects\edu.sylu\eims\eims\target\eims.war
[INFO] [INFO] WEB-INF\web.xml already added, skipping
```

图 6.4 生成 war 包

```
[INFO] 40714/40739 KB
[INFO] 40716/40739 KB
[INFO] 40718/40739 KB
[INFO] 40720/40739 KB
[INFO] 40722/40739 KB
[INFO] 40724/40739 KB
[INFO] 40726/40739 KB
[INFO] 40728/40739 KB
[INFO] 40730/40739 KB
[INFO] 40732/40739 KB
[INFO] 40734/40739 KB
[INFO] 40736/40739 KB
[INFO] 40738/40739 KB
[INFO] 40739/40739 KB
[INFO]
[INFO] Uploaded: http://localhost:8081/nexus/content/repositories/releases/com/yutouxiuxiu/eims/1.3.0/eims-1.3.0.war (40
739 KB at 1888.9 KB/sec)
[INFO] Uploading: http://localhost:8081/nexus/content/repositories/releases/com/yutouxiuxiu/eims/1.3.0/eims-1.3.0.pom
[INFO] 2/8 KB
[INFO] 4/8 KB
[INFO] 6/8 KB
[INFO] 8/8 KB
[INFO]
[INFO] Uploaded: http://localhost:8081/nexus/content/repositories/releases/com/yutouxiuxiu/eims/1.3.0/eims-1.3.0.pom (8
KB at 27.7 KB/sec)
[INFO] Downloading: http://localhost:8081/nexus/content/repositories/releases/com/yutouxiuxiu/eims/maven-metadata.xml
[INFO]
[INFO] Uploading: http://localhost:8081/nexus/content/repositories/releases/com/yutouxiuxiu/eims/maven-metadata.xml
[INFO] 299/299 B
[INFO]
[INFO] Uploaded: http://localhost:8081/nexus/content/repositories/releases/com/yutouxiuxiu/eims/maven-metadata.xml (299
B at 1.9 KB/sec)
[INFO] Uploading: http://localhost:8081/nexus/content/repositories/releases/com/yutouxiuxiu/eims/1.3.0/eims-1.3.0-source
s.jar
[INFO] 2/45 KB
[INFO] 4/45 KB
```

图 6.5 发布项目打包到 Nexus 版本管理

```

[INFO] [INFO] -----
[INFO] [INFO] BUILD SUCCESS
[INFO] [INFO] -----
[INFO] [INFO] Total time: 01:26 min
[INFO] [INFO] Finished at: 2015-06-18T11:48:43+08:00
[INFO] [INFO] Final Memory: 40M/319M
[INFO] [INFO] -----
[INFO] [INFO] Cleaning up after release...
[INFO] [INFO] -----
[INFO] [INFO] BUILD SUCCESS
[INFO] [INFO] -----
[INFO] [INFO] Total time: 02:02 min
[INFO] [INFO] Finished at: 2015-06-18T11:48:43+08:00
[INFO] [INFO] Final Memory: 10M/164M
[INFO] [INFO] -----

```

图 6.6 发布成功

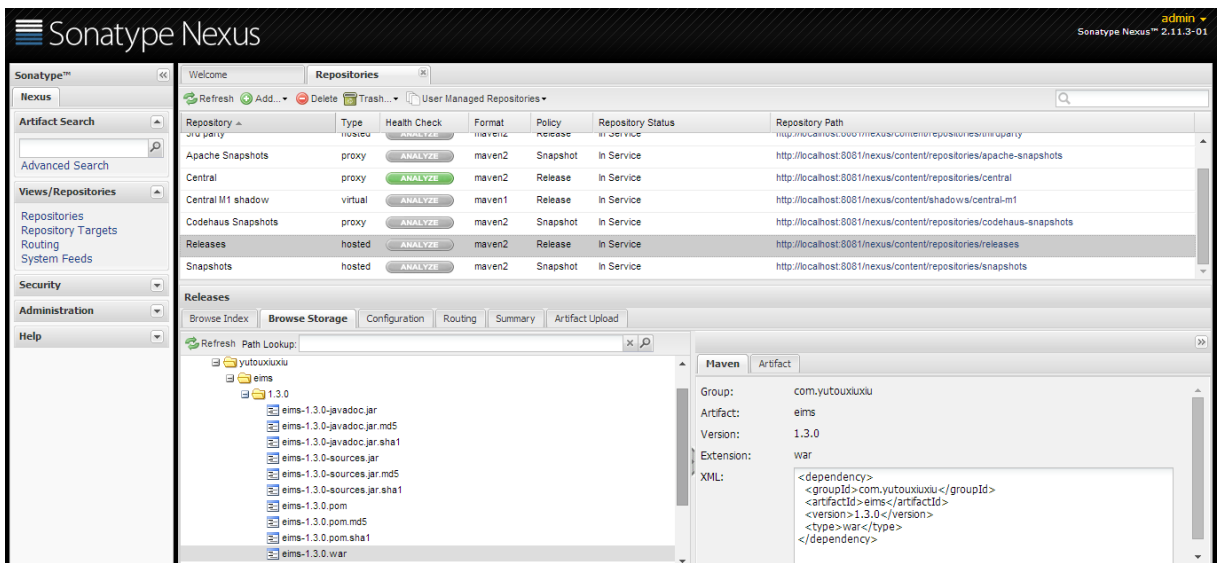


图 6.7 发布结果

结论

本设计主要实现了沈阳理工大学信息科学与工程学院的学生就业信息管理系统。本设计使用 JavaEE 平台编写，使用 Maven 进行项目管理，Subversion 作为版本控制，架构于 SpringMVC 之上，使用 jQuery、EasyUI、Semantic UI、Highcharts 等前端框架进行界面设计。实现了就业管理的自动化。

在要求设计者掌握 JavaEE 企业级软件开发的基础上，还要求对前端开发熟练掌握，尤其是 Jsp 的标签库和 JavaScript 代码。在此次设计的技术实现上还存在两个难点：一个是对于系统中字段的动态管理，涉及到了数据库的设计和前端页面的显示，数据表要求不仅能够满足需求，还应该能尽量减少数据冗余和空间占用，前端页面要求可以显示和供用户修改不同类型的字段的值；另一个难点在于，自定义字段的输入验证，包括前端验证和后端验证，前端验证不通过时，要求能清晰地显示给用户，后端验证要求能保证数据库的安全，其中涉及到了正则表达式，也是比较有难度的。

本次设计基本满足了业务需求，设计出了方便使用且安全的系统，不过还有一些需要改进的地方。比如今后应该给企业留有查询的接口，这样可以方便学生和企业之间的对接。还有，可以从优化就业管理本身的业务流程上下功夫，例如将一些工作交给各班班长完成，在系统中给予班长一定的管理权限，允许其管理自己班级的学生，这样可以大幅度地减轻辅导员的工作。

致谢

在这次设计中我需要感谢和晓军老师，因为当在设计中遇到我无法解决的问题时，和老师总能不厌其烦的为我讲解，鼓励我自己去解决问题，正是有了和老师的帮助，我才能自己解决一个又个问题。还需感谢北京传智播客教育科技有限公司，是他们引导我走进了 JavaEE 的大门，使我的软件开发水平更上一步。还要感谢那些帮助过我解决问题的同学，由于篇幅有限，就不一一列举，在此谨谢！

参考文献

- [1] 陆昆林. 基于 Web 的高校毕业生就业管理系统的开发与设计. 电子科技大学. 2012
- [2] 郭灵玲. 基于 B/S 结构的学校就业管理系统的设计. 华东师范大学. 2009
- [3] 郑黎. 基于 Web 的就业管理系统的开发. 西南交通大学. 2014
- [4] 李德培. 大学生就业管理系统设计与实现. 电子科技大学. 2012
- [5] 黄婷. 高校就业管理系统的设计与实现. 华南理工大学. 2013
- [6] 樊莉丽. 基于 AJAX 的九江学院就业管理系统的设计与实现. 电子科技大学. 2012
- [7] 游武鹏. 基于 J2EE 的高校就业管理系统研究. 复旦大学. 2011
- [8] 陈涛. 基于 Web 的就业管理信息系统的设计与实现. 山东大学. 2009
- [9] 尤嘉. 自动回归测试系统的设计与实现. 北京工业大学. 2008
- [10] 戴楠. 用 SVN 实现软件的版本控制. 电脑知识与技术. 2009
- [11] 杨杰荣. 软件测试过程的研究与应用. 西北工业大学. 2007
- [12] 周志雄. 基于 JQuery 框架技术开发的 WEB 应用. 科协论坛（下半月）. 2013
- [13] 邹存洁. 基于 MVC 模式的 Spring 框架的应用与研究. 大连海事大学. 2006
- [14] 刘行亮. 基于 J2EE 平台的 Spring 框架分析研究与应用. 武汉科技大学. 2006

附录 A 英文原文

HTML Parser

The job of the HTML parser is to parse the HTML markup into a parse tree.

The HTML grammar definition

The vocabulary and syntax of HTML are defined in specifications created by the W3C organization.

Not a context free grammar

As we have seen in the parsing introduction, grammar syntax can be defined formally using formats like BNF.

Unfortunately all the conventional parser topics do not apply to HTML (I didn't bring them up just for fun—they will be used in parsing CSS and JavaScript). HTML cannot easily be defined by a context free grammar that parsers need.

There is a formal format for defining HTML—DTD (Document Type Definition)—but it is not a context free grammar.

This appears strange at first sight; HTML is rather close to XML. There are lots of available XML parsers. There is an XML variation of HTML—XHTML—so what's the big difference?

The difference is that the HTML approach is more "forgiving": it lets you omit certain tags (which are then added implicitly), or sometimes omit start or end tags, and so on. On the whole it's a "soft" syntax, as opposed to XML's stiff and demanding syntax.

This seemingly small detail makes a world of a difference. On one hand this is the main reason why HTML is so popular: it forgives your mistakes and makes life easy for the web author. On the other hand, it makes it difficult to write a formal grammar. So to summarize, HTML cannot be parsed easily by conventional parsers, since its grammar is not context free. HTML cannot be parsed by XML parsers.

HTML DTD

HTML definition is in a DTD format. This format is used to define languages of the SGML family. The format contains definitions for all allowed elements, their attributes and hierarchy. As we saw earlier, the HTML DTD doesn't form a context free grammar.

There are a few variations of the DTD. The strict mode conforms solely to the specifications but other modes contain support for markup used by browsers in the past. The purpose is backwards compatibility with older content.

DOM

The output tree (the "parse tree") is a tree of DOM element and attribute nodes. DOM is short for Document Object Model. It is the object presentation of the HTML document and the interface of HTML elements to the outside world like JavaScript.

The root of the tree is the "Document" object.

The DOM has an almost one-to-one relation to the markup. For example:

```
<html>
  <body>
    <p>
      Hello World
    </p>
    <div> </div>
  </body>
</html>
```

This markup would be translated to the following DOM tree:

Like HTML, DOM is specified by the W3C organization. It is a generic specification for manipulating documents. A specific module describes HTML specific elements.

When I say the tree contains DOM nodes, I mean the tree is constructed of elements that implement one of the DOM interfaces. Browsers use concrete implementations that have other attributes used by the browser internally.

The parsing algorithm

As we saw in the previous sections, HTML cannot be parsed using the regular top down or bottom up parsers.

The reasons are:

The forgiving nature of the language.

The fact that browsers have traditional error tolerance to support well known cases of invalid HTML.

The parsing process is reentrant. For other languages, the source doesn't change during parsing, but in HTML, dynamic code (such as script elements containing `document.write()` calls) can add extra tokens, so the parsing process actually modifies the input.

Unable to use the regular parsing techniques, browsers create custom parsers for parsing HTML.

The parsing algorithm is described in detail by the HTML5 specification. The algorithm consists of two stages: tokenization and tree construction.

Tokenization is the lexical analysis, parsing the input into tokens. Among HTML tokens are start tags, end tags, attribute names and attribute values.

The tokenizer recognizes the token, gives it to the tree constructor, and consumes the next character for recognizing the next token, and so on until the end of the input.

The tokenization algorithm

The algorithm's output is an HTML token. The algorithm is expressed as a state machine. Each state consumes one or more characters of the input stream and updates the next state according to those characters. The decision is influenced by the current tokenization state and by the tree construction state. This means the same consumed character will yield different results for the correct next state, depending on the current state. The algorithm is too complex to describe fully, so let's see a simple example that will help us understand the principle.

Basic example—tokenizing the following HTML:

```
<html>
  <body>
    Hello world
  </body>
</html>
```

The initial state is the "Data state". When the `<` character is encountered, the state is changed to "Tag open state". Consuming an a-z character causes creation of a "Start tag token", the state is changed to "Tag name state". We stay in this state until the `>` character is consumed. Each character is appended to the new token name. In our case the created token is an html token.

When the > tag is reached, the current token is emitted and the state changes back to the "Data state". The <body> tag will be treated by the same steps. So far the html and body tags were emitted. We are now back at the "Data state". Consuming the H character of Hello world will cause creation and emitting of a character token, this goes on until the < of </body> is reached. We will emit a character token for each character of Hello world.

We are now back at the "Tag open state". Consuming the next input / will cause creation of an end tag token and a move to the "Tag name state". Again we stay in this state until we reach >. Then the new tag token will be emitted and we go back to the "Data state". The </html> input will be treated like the previous case.

Tree construction algorithm

When the parser is created the Document object is created. During the tree construction stage the DOM tree with the Document in its root will be modified and elements will be added to it. Each node emitted by the tokenizer will be processed by the tree constructor. For each token the specification defines which DOM element is relevant to it and will be created for this token. The element is added to the DOM tree, and also the stack of open elements. This stack is used to correct nesting mismatches and unclosed tags. The algorithm is also described as a state machine. The states are called "insertion modes".

Let's see the tree construction process for the example input:

```
<html>
  <body>
    Hello world
  </body>
</html>
```

The input to the tree construction stage is a sequence of tokens from the tokenization stage. The first mode is the "initial mode". Receiving the "html" token will cause a move to the "before html" mode and a reprocessing of the token in that mode. This will cause creation of the HTMLHtmlElement element, which will be appended to the root Document object.

The state will be changed to "before head". The "body" token is then received. An HTMLHeadElement will be created implicitly although we don't have a "head" token and it

will be added to the tree.

We now move to the "in head" mode and then to "after head". The body token is reprocessed, an HTMLBodyElement is created and inserted and the mode is transferred to "in body".

The character tokens of the "Hello world" string are now received. The first one will cause creation and insertion of a "Text" node and the other characters will be appended to that node.

The receiving of the body end token will cause a transfer to "after body" mode. We will now receive the html end tag which will move us to "after after body" mode. Receiving the end of file token will end the parsing.

Actions when the parsing is finished

At this stage the browser will mark the document as interactive and start parsing scripts that are in "deferred" mode: those that should be executed after the document is parsed. The document state will be then set to "complete" and a "load" event will be fired.

Browsers' error tolerance

You never get an "Invalid Syntax" error on an HTML page. Browsers fix any invalid content and go on.

Take this HTML for example:

```
<html>
  <mytag>
  </mytag>
  <div>
    <p>
      Really lousy HTML
    </p>
  </div>
</html>
```

I must have violated about a million rules ("mytag" is not a standard tag, wrong nesting of the "p" and "div" elements and more) but the browser still shows it correctly and doesn't complain. So a lot of the parser code is fixing the HTML author mistakes.

Error handling is quite consistent in browsers, but amazingly enough it hasn't been part of HTML specifications. Like bookmarking and back/forward buttons it's just something that developed in browsers over the years. There are known invalid HTML constructs repeated on many sites, and the browsers try to fix them in a way conformant with other browsers.

The HTML5 specification does define some of these requirements. (WebKit summarizes this nicely in the comment at the beginning of the HTML parser class.)

The parser parses tokenized input into the document, building up the document tree. If the document is well-formed, parsing it is straightforward.

Unfortunately, we have to handle many HTML documents that are not well-formed, so the parser has to be tolerant about errors.

We have to take care of at least the following error conditions:

The element being added is explicitly forbidden inside some outer tag. In this case we should close all tags up to the one which forbids the element, and add it afterwards.

We are not allowed to add the element directly. It could be that the person writing the document forgot some tag in between (or that the tag in between is optional). This could be the case with the following tags: HTML HEAD BODY TBODY TR TD LI (did I forget any?).

We want to add a block element inside an inline element. Close all inline elements up to the next higher block element.

If this doesn't help, close elements until we are allowed to add the element—or ignore the tag.

Let's see some WebKit error tolerance examples:

`</br>` instead of `
`

Some sites use `</br>` instead of `
`. In order to be compatible with IE and Firefox, WebKit treats this like `
`.

The code:

```
if (t->isCloseTag(brTag) && m_document->inCompatMode()) {  
    reportError(MalformedBRError);  
    t->beginTag = true;  
}
```

Note that the error handling is internal: it won't be presented to the user.

A stray table

A stray table is a table inside another table, but not inside a table cell.

For example:

```
<table>
  <table>
    <tr><td>inner table</td></tr>
  </table>
  <tr><td>outer table</td></tr>
</table>
```

WebKit will change the hierarchy to two sibling tables:

```
<table>
  <tr><td>outer table</td></tr>
</table>
<table>
  <tr><td>inner table</td></tr>
</table>
```

The code:

```
if (m_inStrayTableContent && localName == tableTag)
    popBlock(tableTag);
```

WebKit uses a stack for the current element contents: it will pop the inner table out of the outer table stack. The tables will now be siblings.

Nested form elements

In case the user puts a form inside another form, the second form is ignored.

The code:

```
if (!m_currentFormElement) {
    m_currentFormElement = new HTMLFormElement(formTag,
m_document);
}
```

A too deep tag hierarchy

The comment speaks for itself.

www.liceo.edu.mx is an example of a site that achieves a level of nesting of about 1500 tags, all from a bunch of s. We will only allow at most 20 nested tags of the same type before just ignoring them all together.

```
bool HTMLParser::allowNestedRedundantTag(const AtomicString& tagName)
{

    unsigned i = 0;
    for (HTMLStackElem* curr = m_blockStack;
         i < cMaxRedundantTagDepth && curr && curr->tagName == tagName;
         curr = curr->next, i++) { }
    return i != cMaxRedundantTagDepth;
}
```

Misplaced html or body end tags

Again—the comment speaks for itself.

Support for really broken HTML. We never close the body tag, since some stupid web pages close it before the actual end of the doc. Let's rely on the end() call to close things.

```
if (t->tagName == htmlTag || t->tagName == bodyTag )
    return;
```

So web authors beware—unless you want to appear as an example in a WebKit error tolerance code snippet—write well formed HTML.

附录 B 中文译文

HTML 解析器

HTML 解析器的任务是将 HTML 标记解析成解析树。

HTML 语法定义

HTML 的词汇和语法在 W3C 组织创建的规范中进行了定义。当前的版本是 HTML4，HTML5 正在处理过程中（译者注：在本文翻译时，HTML 已经是当前版本）。

非与上下文无关的语法

正如我们在解析过程的简介中已经了解到的，语法可以用 BNF 等格式进行正式定义。

很遗憾，所有的常规解析器都不适用于 HTML（我并不是开玩笑，它们可以用于解析 CSS 和 JavaScript）。HTML 并不能很容易地用解析器所需的与上下文无关的语法来定义。

有一种可以定义 HTML 的正规格式：DTD（Document Type Definition，文档类型定义），但它不是与上下文无关的语法。

这初看起来很奇怪：HTML 和 XML 非常相似。有很多 XML 解析器可以使用。在 HTML 中，存在一个 XML 变（XHTML），那么有什么大的区别呢？

区别在于 HTML 的处理更为“宽容”，它允许您省略某些隐式添加的标记，有时还能省略一些起始或者结束标记等等。和 XML 严格的语法不同，HTML 整体来看是一种“软性”的语法。

显然，这种看上去细微的差别实际上却带来了巨大的影响。一方面，这是 HTML 如此流行的原因：它能包容您的错误，简化网络开发。另一方面，这使得它很难编写正式的语法。概括地说，HTML 无法很容易地通过常规解析器解析（因为它的语法不是与上下文无关的语法），也无法通过 XML 解析器来解析。

HTML DTD

HTML 的定义采用了 DTD 格式。此格式可用于定义 SGML 族的语言。它包括所有允许使用的元素及其属性和层次结构的定义。如上文所述，HTML DTD 无法构成与上下文无关的语法。

DTD 存在一些变体。严格模式完全遵守 HTML 规范，而其他模式可支持以前的浏览器所使用的标记。这样做的目的是确保向下兼容一些早期版本的内容。

DOM

解析器的输出“解析树”是由 DOM 元素和属性节点构成的树结构。DOM 是文档对象模型（Document Object Model）的缩写。它是 HTML 文档的对象表示，同时也是外部内容（例如 JavaScript）与 HTML 元素之间的接口。

解析树的根节点是“Document”对象。

DOM 与标记之间几乎是一一对应的关系。比如下面这段标记：

```
<html>
  <body>
    <p>
      Hello World
    </p>
    <div> </div>
  </body>
</html>
```

和 HTML 一样，DOM 也是由 W3C 组织指定的。

我所说的树包含 DOM 节点，指的是树是由实现了某个 DOM 接口的元素构成的。浏览器在具体的实现中会有一些供内部使用的其他属性。

解析算法

我们在之前章节已经说过，HTML 无法用常规的自上而下或自下而上的解析器进行解析。

原因在于：

语言的宽容本质。

浏览器历来对一些常见的无效 HTML 用法采取包容态度。

解析过程需要不断地反复。源内容在解析过程中通常不会改变，但是在 HTML 中，脚本标记如果包含 `document.write`，就会添加额外的标记，这样解析过程实际上就更改了输入内容。

由于不能使用常规的解析技术，浏览器就创建了自定义的解析器来解析 HTML。

HTML5 规范详细地描述了解析算法。此算法由两个阶段组成：标记化和树构建。

标记化是词法分析过程，将输入内容解析成多个标记。HTML 标记包括起始标记、结束标记、属性名称和属性值。

标记生成器识别标记，传递给树构造器，然后接受下一个字符以识别下一个标记；如此反复直到输入的结束。

标记化算法

该算法的输出结果是 HTML 标记。该算法使用状态机来表示。每一个状态接收来自输入信息流的一个或多个字符，并根据这些字符更新下一个状态。当前的标记化状态和树结构状态会影响进入下一状态的决定。这意味着，即使接收的字符相同，对于下一个正确的状态也会产生不同的结果，具体取决于当前的状态。该算法相当复杂，无法在此详述，所以我们通过一个简单的示例来帮助大家理解其原理。

基本示例 - 将下面的 HTML 代码标记化：

```
<html>
  <body>
    Hello world
  </body>
</html>
```

初始状态是数据状态。遇到字符“<”时，状态更改为“标记打开状态”。接收一个 a-z 字符会创建“起始标记”，状态更改为“标记名称状态”。这个状态会一直保持到接收“>”字符。在此期间接收的每个字符都会附加到新的标记名称上。在本例中，我们创建的标记是 html 标记。

遇到 > 标记时，会发送当前的标记，状态改回“数据状态”。<body>标记也会进行同样的处理。目前 html 和 body 标记均已发出。现在我们回到“数据状态”。接收到 Hello world 中的 H 字符时，将创建并发送字符标记，直到接收</body>中的“<”。我们将为 Hello world 中的每个字符都发送一个字符标记。

现在我们回到“标记打开状态”。接收下一个输入字符“/”时，会创建 end tag token 并改为“标记名称状态”。我们会再次保持这个状态，直到接收“>”。然后将发送新的标记，并回到“数据状态”。</html>输入也会进行同样的处理。

树构建算法

在创建解析器的同时，也会创建 **Document** 对象。在树构建阶段，以 **Document** 为根节点的 **DOM** 树也会不断进行修改，向其中添加各种元素。标记生成器发送的每个节点都会由树构建器进行处理。规范中定义了每个标记所对应的 **DOM** 元素，这些元素会在接收到相应的标记时创建。这些元素不仅会添加到 **DOM** 树中，还会添加到开放元素的堆栈中。此堆栈用于纠正嵌套错误和处理未关闭的标记。其算法也可以用状态机来描述。这些状态称为“插入模式”。

让我们来看看示例输入的树构建过程：

```
<html>
  <body>
    Hello world
  </body>
</html>
```

树构建阶段的输入是一个来自标记化阶段的标记序列。第一个模式是“**initial mode**”。接收 **HTML** 标记后转为“**before html**”模式，并在这个模式下重新处理此标记。这样会创建一个 **HTMLHtmlElement** 元素，并将其附加到 **Document** 根对象上。

然后状态将改为“**before head**”。此时我们接收“**body**”标记。即使我们的示例中没有“**head**”标记，系统也会隐式创建一个 **HTMLHeadElement**，并将其添加到树中。

现在我们进入了“**in head**”模式，然后转入“**after head**”模式。系统对 **body** 标记进行重新处理，创建并插入 **HTMLBodyElement**，同时模式转变为“**in body**”。

现在，接收由“**Hello world**”字符串生成的一系列字符标记。接收第一个字符时会创建并插入“**Text**”节点，而其他字符也将附加到该节点。

接收 **body** 结束标记会触发“**after body**”模式。现在我们将接收 **HTML** 结束标记，然后进入“**after after body**”模式。接收到文件结束标记后，解析过程就此结束。

解析结束后的操作

在此阶段，浏览器会将文档标注为交互状态，并开始解析那些处于“**deferred**”模式的脚本，也就是那些应在文档解析完成后才执行的脚本。然后，文档状态将设置为“完成”，一个“加载”事件将随之触发。

您可以在 **HTML5** 规范中查看标记化和树构建的完整算法。

浏览器的容错机制

您在浏览 HTML 网页时从来不会看到“语法无效”的错误，这是因为浏览器会纠正任何无效内容，然后继续工作。

以下面的 HTML 代码为例：

```
<html>
  <mytag>
  </mytag>
  <div>
    <p>
      Really lousy HTML
    </p>
  </div>
</html>
```

在这里，我已经违反了很多语法规则（“mytag”不是标准的标记，“p”和“div”元素之间的嵌套有误等等），但是浏览器仍然会正确地显示这些内容，并且毫无怨言。因为有大量的解析器代码会纠正 HTML 网页作者的错误。

不同浏览器的错误处理机制相当一致，但令人称奇的是，这种机制并不是 HTML 当前规范的一部分。和书签管理以及前进/后退按钮一样，它也是浏览器在多年发展中的产物。很多网站都普遍存在着一些已知的无效 HTML 结构，每一种浏览器都会尝试通过和其他浏览器一样的方式来修复这些无效结构。

HTML5 规范定义了一部分这样的要求。WebKit 在 HTML 解析器类的开头注释中对此做了很好的概括。

解析器对标记化输入内容进行解析，以构建文档树。如果文档的格式正确，就直接进行解析。

遗憾的是，我们不得不处理很多格式错误的 HTML 文档，所以解析器必须具备一定的容错性。

我们至少要能够处理以下错误情况：

明显不能在某些外部标记中添加的元素。在此情况下，我们应该关闭所有标记，直到出现禁止添加的元素，然后再加入该元素。

我们不能直接添加的元素。这很可能是网页作者忘记添加了其中的一些标记（或者其中的标记是可选的）。这些标签可能包括：`html`，`head`，`body`，`tbody`，`tr`，`td`，`li`（还有遗漏的吗？）。

向 `inline` 元素内添加 `block` 元素。关闭所有 `inline` 元素，直到出现下一个较高级的 `block` 元素。

如果这样仍然无效，可关闭所有元素，直到可以添加元素为止，或者忽略该标记。

让我们看一些 WebKit 容错的示例：

使用了 “`</br>`” 而不是 “`
`”

有些网站使用了 “`</br>`” 而不是 “`
`”。为了与 IE 和 Firefox 兼容，WebKit 将其与 “`
`” 做同样的处理。

代码如下：

```
if (t->isCloseTag(brTag) && m_document->inCompatMode()) {  
    reportError(MalformedBRError);  
    t->beginTag = true;  
}
```

请注意，错误处理是在内部进行的，用户并不会看到这个过程。

离散表格

离散表格是指位于其他表格内容中，但又不在任何一个单元格内的表格。

比如以下的示例：

```
<table>  
    <table>  
        <tr><td>inner table</td></tr>  
    </table>  
    <tr><td>outer table</td></tr>  
</table>
```

WebKit 会将其层次结构更改为两个同级表格：

```
<table>  
    <tr><td>outer table</td></tr>
```

```
</table>
```

```
<table>
```

```
<tr><td>inner table</td></tr>
```

```
</table>
```

代码如下：

```
if (m_inStrayTableContent && localName == tableTag)
```

```
    popBlock(tableTag);
```

WebKit 使用一个堆栈来保存当前的元素内容，它会从外部表格的堆栈中弹出内部表格。现在，这两个表格就变成了同级关系。

嵌套的表单元素

如果用户在一个表单元素中又放入了另一个表单，那么第二个表单将被忽略。

代码如下：

```
if (!m_currentFormElement) {
```

```
    m_currentFormElement = new HTMLFormElement(formTag,
m_document);
```

```
}
```

过于复杂的标记层次结构

代码的注释已经说得很清楚了。

示例网站 www.liceo.edu.mx 嵌套了约 1500 个标记，全都来自一堆 “” 标记。我们只允许最多 20 层同类型标记的嵌套，如果再嵌套更多，就会全部忽略。

```
bool HTMLParser::allowNestedRedundantTag(const AtomicString& tagName)
```

```
{
```

```
    unsigned i = 0;
```

```
    for (HTMLStackElem* curr = m_blockStack;
```

```
        i < cMaxRedundantTagDepth && curr && curr->tagName == tagName;
```

```
        curr = curr->next, i++) { }
```

```
    return i != cMaxRedundantTagDepth;
```

```
}
```

放错位置的 `html` 或者 `body` 结束标记

同样，代码的注释已经说得很清楚了。

支持格式非常糟糕的 **HTML** 代码。我们从不关闭 `body` 标记，因为一些愚蠢的网页会在实际文档结束之前就关闭。我们通过调用 `end()` 来执行关闭操作。

```
if (t->tagName == htmlTag || t->tagName == bodyTag )  
    return;
```

所以网页作者需要注意，除非您想作为反面教材出现在 **WebKit** 容错代码段的示例中，否则还请编写格式正确的 **HTML** 代码。

附录 C 程序代码

1 学生自然信息控制器代码（部分）

```
/**
 * CONTROLLER - Student
 * @author yutouxiuxiu@126.com
 *
 */
@Controller
@RequestMapping("/student")
public class StudentController extends BaseController {

    @Autowired
    private StudentService studentService;

    @Autowired
    private EmploymentService employmentService;

    @Autowired
    private ColumnService columnService;

    @Autowired
    private InfoService infoService;

    /**
     * PAGE - student list
     * @param d the action to do, null to display, import or export
     * @param model model
     * @param session session
     * @return jsp: student/list.jsp
     */
}
```

```

    */

@RequestMapping(value = "/d/{d}", method = RequestMethod.GET)
@Privilege
public String studentsList(@PathVariable String d, Model model, HttpSession session) {
    model.addAttribute("category", "student");
    model.addAttribute("categoryUrl", "student");
    model.addAttribute("do", d);

    addColumnsInfoToModel(model);

    return "student/list";
}

/**
 * JS - Dynamic create list.js
 * @param d the action to do, null to display, import or export
 * @param model model
 * @param session session
 * @return jsp: student/list.js.jsp
 */
@RequestMapping("list.js/d/{d}")
@Privilege
public String studentsListJs(@PathVariable String d, Model model, HttpSession session) {
    List<Map<String, Object>> columns =
columnService.getColumnList(ColumnService.CATEGORY_STUDENT);
    model.addAttribute("category", "student");
    model.addAttribute("categoryUrl", "student");
    model.addAttribute("columns", columns);
    model.addAttribute("do", d);
    return "student/list.js";
}

```

```

    }

/**
 * JSON - get student's list for datagrid
 * @param schoolYear school year
 * @param instructor instructor's id
 * @param major major's id
 * @param clazz class's id
 * @param id student's id
 * @param pagination pagination object
 * @param session session
 * @return json: data of easyui datagrid pagination format
 */
@RequestMapping(method = RequestMethod.POST)
@Privilege
public @ResponseBody Map<String, Object> getStudentsList(Integer schoolYear, Integer
instructor, Integer major, Integer clazz, Integer id, Pagination pagination, HttpSession session)
{
    return buildDatagrid(studentService.getStudentsList(schoolYear, instructor, major, clazz,
id, pagination),
        studentService.getStudentListTotal(schoolYear, instructor, major, clazz, id));
}

// =====

/**
 * export student's
 * @param schoolYear school year
 * @param instructor instructor's id
 * @param major major's id

```

```
* @param clazz class's id
* @param id student's id
* @param request http request
* @param response http response
* @param session session
* @throws IOException io exception
*/

@RequestMapping(value = "/export")
@Privilege
public void exportStudentsList(
    @RequestParam(value = "schoolYear", required = false) Integer schoolYear,
    @RequestParam(value = "instructor", required = false) Integer instructor,
    @RequestParam(value = "major", required = false) Integer major,
    @RequestParam(value = "clazz", required = false) Integer clazz,
    @RequestParam(value = "id", required = false) Integer id,
    HttpServletRequest request, HttpServletResponse response, HttpSession session) throws
IOException {

    // all columns
    String[] basicColumnNames = new String[]{"序号", "学号", "姓名", "性别", "入学年份", "辅导员", "专业", "班级"};

    String[] basicColumns = new String[]{"rowno", "number", "name", "gender", "schoolyear", "instructor", "major", "class"};

    List<Map<String, Object>> systemColumnList = new
    ArrayList<Map<String, Object>>();
    for (int i = 0; i < basicColumnNames.length; i++) {
        String name = basicColumnNames[i];
        String english = basicColumns[i];
        Map<String, Object> column = new HashMap<String, Object>();
```

```
        column.put("fullname", "column.students." + english);
        column.put("name", name);
        column.put("type", "text");
        column.put("english", english);
        systemColumnList.add(column);
    }
    systemColumnList.addAll(columnService.getAllColumnsConcatCatalogAndEnglish());

    // columns from checkbox
    List<String> inputColumnList = new ArrayList<String>();
    Map parameterMap = request.getParameterMap();
    for (Object key : parameterMap.keySet()) {
        if (key.toString().startsWith("column.") && ((Object[])
parameterMap.get(key))[0].toString().equals("true")) {
            inputColumnList.add(key.toString());
        }
    }

    if (inputColumnList.size() == 0) {
        throw new ExportException(ExportException.NO_COLUMN);
    }

    // columns you want to export in order
    List<Map<String, Object>> exportColumnList = new
ArrayList<Map<String, Object>>();
    for (Map<String, Object> systemColumn : systemColumnList) {
        for (String inputColumn : inputColumnList) {
            if (systemColumn.get("fullname").equals(inputColumn)) {
                exportColumnList.add(systemColumn);
                break;
            }
        }
    }
}
```

```
    }  
    }  
}  
  
// data  
List<Map<String, Object>> result = studentService.getStudentsAllInfoList(schoolYear,  
instructor, major, clazz, id);  
  
// add row number  
for (int i = 0; i < result.size(); i++) {  
    result.get(i).put("rowno", i + 1);  
}  
  
// create the xls file  
Workbook wb = new HSSFWorkbook();  
Sheet sheet = wb.createSheet("Sheet1");  
  
for (int i = 0; i < result.size() + 1; i++) {  
    Row row = sheet.createRow(i);  
    // first row  
    if (i == 0) {  
        for (int j = 0; j < exportColumnList.size(); j++) {  
            Cell cell = row.createCell(j);  
            cell.setCellType(Cell.CELL_TYPE_STRING);  
            cell.setCellValue(exportColumnList.get(j).get("name").toString());  
        }  
    } else {  
        // other lines  
        Map<String, Object> line = result.get(i - 1);  
        for (int j = 0; j < exportColumnList.size(); j++) {  
            Map<String, Object> column = exportColumnList.get(j);
```

```

        Object value = line.get(column.get("english"));

        // find the text of radio type
        if (column.get("type").equals("radio")) {
            List<Map<String, Object>> options = (List<Map<String, Object>>)
column.get("options");
            Boolean isGetOptionValue = false;
            for (Map<String, Object> option : options) {
                if (option.get("id").equals(value)) {
                    value = option.get("name");
                    isGetOptionValue = true;
                    break;
                }
            }
            if (value != null && !isGetOptionValue) {
                throw new RuntimeException("can not get the options of column " +
column.get("name"));
            }
        }

        Cell cell = row.createCell(j);
        cell.setCellValue(value == null ? "" : value.toString());
    }
}

// for (int i = 0; i < columns.size() + oriColumns.length; i++) {
//     sheet.autoSizeColumn(i);
// }

```

```

        // download
        response.reset();
        response.setHeader("Content-Disposition", "attachment; filename=" +
URLLEncoder.encode("导出", "UTF-8") + ".xls");
        response.setContentType("application/octet-stream; charset=utf-8");
        OutputStream os = response.getOutputStream();
        wb.write(os);
        os.flush();
        os.close();
    }

    /**
     * batch add student
     * @param request http request
     * @param session session
     * @return json
     * @throws IllegalStateException illegal state exception
     * @throws IOException io exception
     * @throws InvalidFormatException invalid format exception
     */
    @RequestMapping(value = "batchadd", method = RequestMethod.POST)
    @Privilege
    public @ResponseBody Map<String, Object> batchAddStudent(HttpServletRequest
request, HttpSession session) throws IllegalStateException, IOException,
InvalidFormatException {

        // get input
        if (!(request instanceof MultipartHttpServletRequest)) {
            throw new ImportException(ImportException.NO_FILE);
        }
    }

```

```

MultipartHttpServletRequest multipartRequest = (MultipartHttpServletRequest) request;
String table = multipartRequest.getParameter("table");
Boolean update = multipartRequest.getParameter("update") != null;
Boolean checkName = multipartRequest.getParameter("checkName") != null;
Boolean empty = multipartRequest.getParameter("empty") != null;

Map<String, Object> result = new HashMap<String, Object>();
result.put("error", false);

File file = null;
try {
    String path = uploadExcel(multipartRequest);
    file = new File(path);

    // open
    Sheet sheet = getSheet0(path);
    // get all columns
    List<String> columnList = getAllColumnsFormASheet(sheet);
    // get all data
    List<List<String>> dataList = getAllDataFromASheet(sheet, columnList);
    for (List<String> data : dataList) {
        for (String value : data) {
            value = (String) BasicUtil.prepareValue(value);
        }
    }

    List<Map<String, Object>> systemColumnInfoList =
columnService.getColumnList(table);

    // remove columns which do not in the system
    removeColumnsDoNoInSystem(columnList, systemColumnInfoList, dataList);

```

```

// get all possible number start
User user = (User) session.getAttribute("user");

List<Map<String, Object>> majorList =
infoService.getAllMajorList(user.getType().equals(User.USER_TYPE_ADMIN) ? 0 :
user.getId());

List<String> allPossibleNumberStart = new ArrayList<String>();
for (Map<String, Object> major : majorList) {
    for (int clazz = 1; clazz <= Integer.parseInt(major.get("classes").toString()); clazz++)
    {
        allPossibleNumberStart.add(major.get("year").toString().substring(2, 4) +
            infoService.getSetting("collegeNumber", String.class) +
            major.get("number").toString() +
            (String.valueOf(clazz).length() == 2 ? clazz : ("0" + clazz)));
    }
}

// add to database
for (int i = 0; i < dataList.size(); i++) {
    int row = i + 1;
    List<String> student = dataList.get(i);
    String number = student.get(columnList.indexOf("学号"));

    // validate number [VERY IMPORTANT]
    try {
        if (!BasicUtil.inArray(number.substring(0, 8), allPossibleNumberStart.toArray()))
        {
            throw new ImportException(" 您 不 具 有 管 理 该 学 号 ( " +
student.get(columnList.indexOf("学号")) + ") 学生的权限，在第" + row + "行");
        }
    }
}

```



```
    } catch (StringIndexOutOfBoundsException e) {
        throw new ImportException("学号 (" + student.get(columnList.indexOf("学号"))
+ ") 有误，在第" + row + "行");
    }

    Map<String, String> parameterMap = buildParameterMap(columnList, number,
student, systemColumnInfoList);

    Integer studentId = null;
    // if it is a new student
    if (table.equals("students")
&& !studentService.checkStudentIfExistByNumber(number)) {
        addStudentFromSheet(table, columnList, number, student, row);
    }
    if (!table.equals("students")) { // if it is a new record in other table
        try {
            studentId = studentService.getStudentIdByNumber(number);
        } catch (EmptyResultDataAccessException e) {
            throw new ImportException("该学生（学号：" +
student.get(columnList.indexOf("学号")) + ") 在系统中不存在，请先在自然情况管理中导
入学生自然信息，在第" + row + "行");
        }
        if (!studentService.checkIfExistByStudentId(table, studentId)) {
            parameterMap.put("student", studentId.toString());
            studentService.insertOneRecordIntoOtherTable(table, parameterMap);
        }
    }

    // if student's name in the sheet does not equal to it in the system
    if (checkName) {
```

```
        if (!columnList.contains("姓名")) {
            throw new ImportException("您勾选了检测姓名是否匹配的选项，但导入的
Excel 表格中不存在姓名字段");
        }
        try {
            Map<String, Object> studentInfo = studentService.getStudent(number);
            if (!student.get(columnList.indexOf("姓名")).equals(studentInfo.get("name")))
{
                throw new ImportException("检测到相同学号 (" + number + ") 的学生的
姓名和系统中的不匹配 (Excel 表格中为: " + student.get(columnList.indexOf("姓名")) + ",
系统中为:  " + studentInfo.get("name") + ", 导入失败，在第" + row + "行");
            }
        } catch (EmptyResultDataAccessException e) {
        }
    }

    // update employment status
    if (table.equals("obtains")) {
        if (parameterMap.get("signDate") != null
&& !BasicUtil.isBlank(parameterMap.get("signDate"))) {
            parameterMap.put("state", "已就业");
        } else {
            parameterMap.put("state", "未就业");
        }
    }

    parameterMap.put("confirmed", "未确认");

    // update student info
    if (table.equals("students")) {
```

```

        studentService.updateTableBySomething("students",    "number",    number,
parameterMap);
    } else {
        studentService.updateTableBySomething(table, "student",  studentId.toString(),
parameterMap);
    }
}

// delete
file.delete();
} finally {
    if (file != null) {
        file.delete();
    }
}

return result;
}

```

3 学生自然信息业务层代码（部分）

```

/**
 * SERVICE - student
 * @author yutouxiuxiu@126.com
 *
 */
@Service
public class StudentService extends BaseService {

    @Autowired
    private JdbcTemplate jdbcTemplate;

```

```

/**
 * get student's list with pagination
 * @param schoolYear school year
 * @param instructor instructor's id
 * @param major major's id
 * @param clazz class's id
 * @param id student's id
 * @param pagination pagination object
 * @return student list
 */

public List<Map<String, Object>> getStudentsList(Integer schoolYear, Integer instructor,
Integer major, Integer clazz, Integer id, Pagination pagination) {
    instructor = (instructor != null && instructor.equals(0)) ? null : instructor;

    String sql = buildStudentsListSql(schoolYear, instructor, major, clazz, id)
        + " LIMIT ?, ?";

    if (schoolYear != null) {
        schoolYear = Integer.parseInt(schoolYear.toString().substring(2, 4));
    }

    Object[] params = buildParams(schoolYear, instructor, major, clazz, id,
pagination.getStart(), pagination.getCount());

    List<Map<String, Object>> result = jdbcTemplate.queryForList(sql, params);

    logQuery("getStudentList", sql, params, result);

    return result;
}

```

```
}

/**
 * get students' list without pagination
 * @param schoolYear school year
 * @param instructor instructor's id
 * @param major major's id
 * @param clazz class's id
 * @param id student's id
 * @return student list
 */

public List<Map<String, Object>> getStudentsList(Integer schoolYear, Integer instructor,
Integer major, Integer clazz, Integer id) {
    instructor = (instructor != null && instructor.equals(0)) ? null : instructor;

    String sql = buildStudentsListSql(schoolYear, instructor, major, clazz, id);

    if (schoolYear != null) {
        schoolYear = Integer.parseInt(schoolYear.toString().substring(2, 4));
    }

    List<Map<String, Object>> result = jdbcTemplate.queryForList(sql,
buildParams(schoolYear, instructor, major, clazz, id));

    return result;
}

/**
 * build sql used to get students list
 * @param schoolYear school year
```

```

    * @param instructor instructor's id
    * @param major major's id
    * @param clazz class's id
    * @param id student's id
    * @return sql
    */

    private String buildStudentsListSql(Integer schoolYear, Integer instructor, Integer major,
Integer clazz, Integer id) {
        String sql = "SELECT s.*, m.year schoolyear, u.name instructor, m.name major,
SUBSTRING(s.number, 1, 8) class"
            + " FROM students s"
            + " LEFT JOIN majors m ON (m.number = SUBSTRING(s.number, 5, 2) AND
SUBSTRING(m.year, 3, 2) = SUBSTRING(s.number, 1, 2))"
            + " LEFT JOIN users u ON (u.id = m.instructor)";
        sql += " WHERE 1 = 1";
        if (schoolYear != null) sql += " AND SUBSTRING(s.number, 1, 2) = ?";
        if (instructor != null) sql += " AND u.id = ?";
        if (major != null) sql += " AND m.id = ?";
        if (clazz != null) sql += " AND SUBSTRING(s.number, 7, 2) = ?";
        if (id != null) sql += " AND s.id = ?";
        sql += " ORDER BY s.number ASC";

        return sql;
    }

    /**
    * build sql of getting total number of a student list
    * @param schoolYear school year
    * @param instructor instructor's id
    * @param major major's id

```

```

    * @param clazz class's id
    * @param id student's id
    * @return sql
    */

    private String buildStudentsListTotalSql(Integer schoolYear, Integer instructor, Integer
major, Integer clazz, Integer id) {
        String sql = "SELECT COUNT(*) FROM (" + buildStudentsListSql(schoolYear,
instructor, major, clazz, id) + ") a";

        return sql;
    }

```

3 学生自然信息后台页面前端 js 文件代码（部分）

```

$.fn.validatebox.defaults.rules = $.extend($.fn.validatebox.defaults.rules, {
<c:forEach items="${columns}" var="column">
    <c:if test="${column.validate != null && fn:trim(column.validate) != ''}">
        rule${column.english}: {
            validator: function(value, param) {
                return /${column.validate}/i.test(value);
            },
            message: '请以正确的格式输入'
        },
    </c:if>
</c:forEach>
});

/**
 * datagrid
 */
var is${category}${d}ListDatagridExport = false;

```

```

var ${category}${d}ExportParams = {};
${category}${d}ListDatagrid.edatagrid({
    url: contextPath + "/" + ${categoryUrl}",
    updateUrl: contextPath + "/" + ${categoryUrl}/updateinline",
    destroyUrl: contextPath + "/" + ${categoryUrl}/delete",
    infoBox: $("#${category}${d}-list-datagrid-toolbar").find(".datagrid-info"),
    pagination: true,
    pageSize: 50,
    pageList: [10, 20, 50, 100],
    rownumbers: true,
    toolbar: $("#${category}${d}-list-datagrid-toolbar"),
    onBeforeLoad: function(params) {
        // export
        if (is${category}${d}ListDatagridExport) {
            is${category}${d}ListDatagridExport = false;
            $.extend(params, ${category}${d}ExportParams);
            $.ajaxDownload(contextPath + "/student/export", "POST", params);

            return false;
        }
        // do not load at first time
        if (${category}${d}ListDatagridFirstLoadCount < 1) {
            return false;
        } else {
            return true;
        }
        ${category}${d}ListDatagridFirstLoadCount++;
    },
    columns: [[
        { field: "checkbox", checkbox: true, /*rowspan: 2*/},

```



```

/*{title: "基本信息", colspan: 12},
{title: "家庭信息", colspan: 2},
{field: "action", rowspan: 2, width: 70, formatter: function(value, row, index) {
    if (row.editing) {
        return '<a href="#" onclick="saveInlineEdit(\'#${category}${d}-list-datagrid\', ' +
index          +          ')">          保          存          </a>          <a          href="#"
onclick="cancelInlineEdit(\'#${category}${d}-list-datagrid\', ' + index + ') ">取消</a>;
    }
}},
], [*/
{field: "number", title: "学号"},
{field: "name", title: "姓名",
<c:if test="${category == 'student'}">
    editor: {
        type: "validatebox",
        options: {
            required: true
        }
    }
</c:if>
},
{field: "gender", title: "性别"},
<c:if test="${category == 'student'}">
    {field: "schoolyear", title: "入学年份", formatter: function(value, row, index) {
        return "20" + row["number"].substr(0, 2);
    }},
    {field: "instructor", title: "辅导员"},
    {field: "major", title: "专业"},
    {field: "class", title: "班级"},
</c:if>

```

```
<c:if test="\${category == 'obtain'}">
  {field: "adviser", title: "毕设指导", formatter: function(value, row, index) {
    if ("a" == "b") {}
    <c:forEach items="\${advisers}" var="adviser" varStatus="status">
      else if (value == "\${adviser.id}") {
        return "\${adviser.name}";
      }
    </c:forEach>
    else if (value != null && value != "") {
      return "(已删除)";
    }
  },
  editor: {
    type: "combobox",
    options: {
      valueField: "value",
      textField: "text",
      data: [
        <c:forEach items="\${advisers}" var="adviser" varStatus="status">
          {
            value: \${adviser.id},
            text: "\${adviser.name}\${adviser.section_name}",
          },
        </c:forEach>
      ],
    }
  },
  {field: "section", title: "教研室", formatter: function(value, row, index) {
    if (row.section != null) {
      return row.section.name;
    }
  }
}
```

```

    }
  }},
</c:if>
<c:if test="${category == 'obtain'}">
  {field: "state", title: "状态", formatter: function(value, row, index) {
    if (value == null || value == "") {
      return "未就业";
    } else {
      return value;
    }
  }},
</c:if>
<c:if test="${category == 'student' || category == 'obtain'}">
  {field: "confirmed", title: "学生确认", formatter: function(value, row, index) {
    if (value == "未确认") {
      return "<font style='color: red'>未确认</font>";
    } else {
      return "<font style='color: green'>已确认</font>";
    }
  }},
</c:if>
<c:forEach items="${columns}" var="column">
  {
    field: "${column.english}",
    title: "${column.name}",
    <c:choose>
      <c:when test="${column.formatter != null && fn:trim(column.formatter) != ''}">
        formatter: ${column.formatter},
      </c:when>
      <c:otherwise>

```

```
<c:choose>
  <c:when test="${column.type == 'radio'}">
    formatter: function(value, row, index) {
      if ("a" == "b") {}
      <c:forEach items="${column.options}" var="option" varStatus="status">
        else if (value == "${option.id}") {
          return "${option.name}";
        }
      </c:forEach>
    },
  </c:when>
</c:choose>
</c:otherwise>
</c:choose>
<c:if test="${!column.noneditable}">
  <c:choose>
    <c:when test="${column.editor != null && fn:trim(column.editor) != ''}">
      editor: ${column.editor},
    </c:when>
    <c:otherwise>
      <c:choose>
        <c:when test="${column.type == 'text' || column.type == 'textarea'}">
          editor: {
            type: "validatebox",
            options: {
              <c:if test="${column.required}">
                required: true,
              </c:if>
              <c:if test="${column.validate != null && fn:trim(column.validate) != ''}">
                validType: 'rule${column.english}',
```

```

        </c:if>
    }
},
    </c:when>
    <c:when test="${column.type == 'radio'}">
editor: {
    type: "combobox",
    options: {
        <c:if test="${column.required}">
        required: true,
        </c:if>
        editable: false,
        valueField: "value",
        textField: "text",
        data: [
            <c:forEach items="${column.options}" var="option" varStatus="status">
            {
                value: ${option.id},
                text: "${option.name}",
            },
            </c:forEach>
        ],
    }
}
    </c:when>
    <c:when test="${column.type == 'date'}">
editor: {
    type: "datebox",
}
    </c:when>

```

```

        </c:choose>
    </c:otherwise>
</c:choose>
</c:if>
<c:if test="\${column.other != null && fn:trim(column.other) != ""}">
    ${column.other},
</c:if>
    },
</c:forEach>
    {field: "action", /*rowspan: 2,* / width: 70, formatter: function(value, row, index) {
        if (row.editing) {
            return '<a href="#" onclick="saveInlineEdit(\'#\${category}\${d}-list-datagrid\', ' +
index          +          ')">    保        存        </a>        <a          href="#"
onclick="cancelInlineEdit(\'#\${category}\${d}-list-datagrid\', ' + index + ')">取消</a>;
        }
    }},
    ],
    destroyMsg:{
        norecord:{ // when no record is selected
            title: '提示',
            msg: '没有选择任何学生，无法删除'
        },
        confirm:{ // when select a row
            title: '警告',
            msg: '真的要删除选中的学生么？请仔细考虑，如果操作失误将造成无法逆转的
后果!!! '
        }
    },
    onDestroy: function() {
        changeBottonStatus(\${category}\${d}ListDatagrid);

```

```

    },
    onCancelEdit: function() {
        changeBottonStatus($${category} ${d}ListDatagrid);
    },
    onClickRow: function() {
        changeBottonStatus($${category} ${d}ListDatagrid);
    },
});

```

3 异常统一处理逻辑代码

```

public class ExceptionResolver extends SimpleMappingExceptionResolver {

    protected Logger logger = Logger.getRootLogger();

    @Override
    protected ModelAndView doResolveException(HttpServletRequest request,
        HttpServletResponse response, Object handler, Exception ex) {
        String viewName = determineViewName(ex, request);

        if (viewName != null) { // jsp
            if (((!(request.getHeader("accept").indexOf("application/json") > -1 ||
                (request.getHeader("X-Requested-With") != null &&
                request.getHeader("X-Requested-With").indexOf("XMLHttpRequest") >
-1))) ||
                (request.getParameter("page") != null &&
request.getParameter("page").equals("true")))) { // not ajax

                Integer statusCode = determineStatusCode(request, viewName);
                if (statusCode != null) {
                    applyStatusCodeIfPossible(request, response, statusCode);

```

```

    }

    // return message
    if (checkIfItIsASystemException(ex)) {
        request.setAttribute("message", "系统出现未知错误，请联系开发人
员");
    } else {
        request.setAttribute("message", ex.getMessage());
    }

    log(ex);

    return getModelAndView(viewName, ex, request);
} else { // json
    try {
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json");
        Map<String, Object> result = new HashMap<String, Object>();
        result.put("error", true);
        if (checkIfItIsASystemException(ex)) {
            result.put("message", "系统出现未知错误，请联系开发人员");
        } else {
            result.put("message", ex.getMessage());
        }

        log(ex);

        ObjectMapper mapper = new ObjectMapper();
        PrintWriter writer = response.getWriter();
        writer.write(mapper.writeValueAsString(result));
    }
}

```



```
        writer.flush();
    } catch (IOException e) {
        log(e);
    }
    return null;
}
} else {
    return null;
}
}

protected void log(Exception e) {
    if (checkIfItIsASystemException(e)) {
        StackTraceElement[] stackTraceElements = e.getStackTrace();
        String stackTrace = "";
        for (StackTraceElement stackTraceElement : stackTraceElements) {
            // only record stack trace element in eims package
            if (stackTraceElement.getClassName().startsWith("com.yutouxiuxiu.eims")
&& !stackTraceElement.getClassName().contains("$")) {
                stackTrace += "\n        at " + stackTraceElement.getClassName() + ":" +
stackTraceElement.getLineNumber();
            }
        }
        logger.error(e.getClass().getName() + ": " + e.getMessage() + stackTrace);
    }
}

protected boolean checkIfItIsASystemException(Exception e) {
    if (e instanceof AppUserException) {
        return false;
    }
}
```

```

    } else {
        return true;
    }
}
}

```

5 权限管理 AOP 环绕切面代码

```

public class PrivilegeAspect {

    public Object doAround(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
        String value = null;
        Object result = null;

        // get method
        Method[] methods = proceedingJoinPoint.getTarget().getClass().getMethods();
        Method method = null;
        for (Method m : methods) {
            if (m.getName().equals(proceedingJoinPoint.getSignature().getName())) {
                method = m;
            }
        }

        // get privilege annotation
        if (method.isAnnotationPresent(Privilege.class)) {
            Privilege privilege = method.getAnnotation(Privilege.class);
            value = privilege.value();
        }

        if (value == null) {
            return proceedingJoinPoint.proceed();
        }
    }
}

```

```

    }

    // get session
    Object[] args = proceedingJoinPoint.getArgs();
    HttpSession session = null;
    for (Object arg : args) {
        if (arg instanceof HttpSession) {
            session = (HttpSession) arg;
        }
    }

    // check
    if (session.getAttribute("user") != null) {
        User user = (User) session.getAttribute("user");

        if (value.equals(User.USER_TYPE_ADMIN) &&
        user.getType().equals(User.USER_TYPE_ADMIN)) {
            // admin
            result = proceedingJoinPoint.proceed();
        } else if (value.equals(User.USER_TYPE_INSTRUCTOR) &&
        (user.getType().equals(User.USER_TYPE_INSTRUCTOR) ||
        user.getType().equals(User.USER_TYPE_ADMIN))) {
            // instructor or admin
            result = proceedingJoinPoint.proceed();
        } else if (value.equals(User.USER_TYPE_STUDENT) &&
        user.getType().equals(User.USER_TYPE_STUDENT)) {
            // student
            result = proceedingJoinPoint.proceed();
        } else {
            throw new PrivilegeException(PrivilegeException.PERMISSION_DENIED);
        }
    }

```

```
    }  
    } else {  
        throw new PrivilegeException(PrivilegeException.NEED_LOGIN);  
    }  
  
    return result;  
    }  
}
```

6 权限管理注解代码

```
@Retention(RetentionPolicy.RUNTIME)  
@Target(ElementType.METHOD)  
@Inherited  
public @interface Privilege {  
  
    public String value() default User.USER_TYPE_INSTRUCTOR;  
}
```