

**《数据库原理》课程设计报告**



**小型药品ERP系统的设计与实现**

**班级 ： 计科186**

**学号 ： 201811404628**

**组长： 苏科昱**

**莆田学院新工科产业学院**

**2021年1月**

**课程设计完成基本情况及评分**

1. 小组成员情况：

组长：苏科昱 学号：201811404628 电话：13645912977 班级：计科186

组员：秦泽超 学号：201811404626 电话：13645912977 班级：计科186

郑亦楠 学号：201811404647 电话：13645912977 班级：计科186

1. 任务完成情况：

|  |  |
| --- | --- |
| 苏科昱 | 系统设计、数据库设计、系统实现、系统测试 |
| 秦泽超 | 需求分析、系统测试 |
| 郑亦楠 | 需求分析、系统测试 |

1. 组长评分（10分制）：

|  |  |  |
| --- | --- | --- |
| 苏科昱 | 秦泽超 | 郑亦楠 |
| 9.0 | 8.0 | 8.0 |

1. 终评成绩（五级计分制）：

|  |  |
| --- | --- |
| 苏科昱 |  |
| 秦泽超 |  |
| 郑亦楠 |  |

**指导教师签名：**

目录

[第一章 绪论 3](#_Toc4995)

[1.1背景 3](#_Toc28586)

[1.2开发意义 4](#_Toc24496)

[1.3系统目标 4](#_Toc13151)

[1.4本文结构 4](#_Toc31105)

[第二章 系统开发技术 5](#_Toc7741)

[2.1开发工具 5](#_Toc23185)

[2.2后台技术 6](#_Toc13874)

[2.3前端技术 7](#_Toc231)

[2.4数据库技术 7](#_Toc6575)

[第三章 系统分析 7](#_Toc28565)

[3.1可行性分析 7](#_Toc11352)

[3.2需求分析 8](#_Toc7452)

[第四章 数据库设计 13](#_Toc28247)

[4.1 概念设计 13](#_Toc1543)

[4.2 逻辑设计 15](#_Toc31487)

[4.3 物理设计 21](#_Toc3432)

[第五章 系统设计 22](#_Toc20974)

[5.1系统模块设计 22](#_Toc25102)

[5.2系统后端设计 22](#_Toc26057)

[5.3系统前端设计 25](#_Toc5814)

[5.4业务设计 27](#_Toc1603)

[第六章 系统实现 29](#_Toc19594)

[6.1 系统数据交互关系 29](#_Toc10014)

[6.2 系统模块实现 30](#_Toc2713)

[6.2 药品进销存模块实现 46](#_Toc21553)

[第七章 系统测试 58](#_Toc29247)

[7.1测试简述 58](#_Toc25286)

[7.2测试方式 58](#_Toc16270)

[7.3测试用例 59](#_Toc13006)

[总结 64](#_Toc15925)

[参考文献 65](#_Toc9911)

[附录 65](#_Toc24193)

1. **绪论**

**1.1背景**

随着疫情在全球各地快速蔓延，在我国疫情被快速控制，避免了疫情在我国更进一步恶化，但是在疫情稳定之后，大部分医院种的药品已经消耗殆尽，需要进行一次全方位的补充，所以医院对医药产品的需求与日俱增。众所皆知，在数量众多的出入库情况下，使用传统的医院药品进销存管理模式依靠人力是无法满足需求的，而且传统的模式需要消耗大量的人力物力时间，会导致效率低下，且出错率高。旧的药品进销存管理模式已经无法满足我国医院日益扩大的药品需求了。

伴随着互联网+的时代快速兴起，越来越多的计算机技术飞快普及，几乎所有的医院已经无法忽视计算机这种辅助工具为自己带来的便利了。医院可以利用计算机来提升自己的工作效率，对仓库中的库存进行电子化管理，在一定程度上避免不必要的资源浪费，从而增加医院的效益。因此，尽快建立一份功能齐备的专门为医院设计的进销存管理系统，完善一个现代医院的信息化管理机制，是医院库存管理和维护工作更加规范，更加系统，更加有效率是十分必要的。

这次想要开发这个系统是因为在疫情期间看新闻，发现有大量的医疗资源囤积在仓库外，只能靠人力去进行登记整理，所以我想做一个基于医院的药品进销存系统。基于此，我针对药品进销存系统在系统设计还有开发等方面做了重点研究，通过网上查阅资料等方式，对开发系统的过程中可能会涉及的一系列软件及技术进行学习，对系统的数据库进行设计。这样做的主要目的是为了能够从技术手段的角度来阐述怎样解决我国现代的医院如何从原来传统的人工管理模式向现代化的信息化的管理模式的转变，除此之外还有医药的进销存系统在医院管理信息化中所起到的重要作用。

**1.2开发意义**

本系统研究设计的意义是为了使医院运作过程中的信息维护、采购、入库等操作的电子化，加快现代的医院从原来传统的人工管理模式向现代化的信息化的管理模式的转变。

医院药品进销存系统主要用途在于使用电子化的操作代替人工管理，为医院提供一个高效，便捷的管理平台。在这个互联网+的新时代，使用计算机对医院的各项信息进行管理，具有使用手工管理所无法比拟的优点。例如：可维护性高、查找方便、管理方便、可靠性及保密性好、寿命长、成本低等。这些优点能够极大的提高医院的运行效率，也是一所医院的走向科学化、正规化管理和世界接轨的重要前提条件。因此，开发这样一个小型药品进销存系统是很有必要的一件事情。

**1.3系统目标**

实现一个小型药品进销存管理系统，实现对医院运作过程中的信息维护、采购、入库等操作的电子化，可以节省大量人力物力时间，可以说对医院的运作带来无法估计的好处。

首先实现一个简单实用，操作友好且界面美观的药品进销管理系统是我们进行系统开发首要解决的任务，因为用户是通过界面与系统进行交互的。然后系统需要实现一些主要的功能，一是要实现对系统用户的权限管理，设置系统对不同类型的用户进行界面显示功能；二是要实现对科室、药品、供应商、生产厂商等信息的维护；三是要实现对药品的采购、入库等进销存操作。最后系统需要具备有关进销存的一些基本的操作，例如采购入库、审核单据、作废单据等。

**1.4本文结构**

本文一共包含七个章节，具体的各个章节安排如下：

第一章绪论：主要叙述了小型药品进销存系统的开发背景和意义和系统开发的目标，开发这个系统对自身或者对社会有什么意义，还有阐述系统大致需要达成什么效果有什么目标。

第二章系统开发技术：本章节首先是概述了开发系统时所需要用到的开发工具和辅助工具，接下来分别简单的介绍了开发系统所用到的后台技术和前端技术，再者就是介绍在系统开发中选用哪种数据库对数据进行存储。

第三章系统分析：本章节主要是对系统进行分析，主要从可行性和需求两方面进行分析。可行性分析从技术可行性、经济可行性、操作可行性三个方面进行综合分析，最终得出系统开发的可行性结论；需求分析从需求调研、系统功能需求和用例图这三方面进行分析，对用户的需求进行全面的分析，得出系统所需要具备的功能，性能要求等。

第四章数据库设计：本章节主要是对数据库进行设计，分别从概念设计、逻辑设计、物理设计三个方面入手，概念设计中使用ER图来直观的把数据库表的一些属性表示出来；逻辑分析则是对系统E-R图进行转化，并从E-R图的最终转化结果分析出系统的数据模型；物理设计主要作用有确认数据的存放位置、系统配置和设计数据存取路径等。

第五章系统设计：本章节主要分为系统模块设计、系统后端设计、系统前端设计、关键业务设计这四个模块。系统模块设计包含系统的功能模块图；系统后端设计说明了项目的结构、对项目结构的设计以及项目的主要包图类图；系统前端设计包括了前端的模板界面设计，设计出系统的界面原型；关键业务设计则是使用UML图对也业务进行解析。

第六章系统实现：本章节主要是对系统的实现进行一个完整的介绍。首先是系统数据交互关系，指出用户在使用界面时操作的是哪张表；然后系统模块及药品进销存模块实现，首先是对模块中每个子模块的描述，然后附上实现的界面，接下来详细描述子模块的实现过程并且附上实现这些功能的核心代码。

第七章系统测试：本章节主要说明了在开发系统的过程中，我们使用哪些方法对系统进行测试，给出测试用例并分析测试结构是否达到预期结果，如未达成提出改进的方法

总结：本章主要是对小型药品进销存系统的开发设计的整个过程做一个总结，说明从一开始开发到开发完成的经过的一个简单阐述。并且说明系统还存在的问题，及可能改进的方向。还有说明自己在这次课程设计中收获了什么。

1. **系统开发技术**

**2.1开发工具**

**2.1.1 IDEA**

IDEA全称IntelliJ IDEA，是Java编程语言开发的集成环境。目前IntelliJ在业界被公认为最好的Java开发工具，我编写Java项目主要也是使用IDEA，因为IDEA中有良好的智能代码助手、代码自动提示、重构、JUnit、CVS整合、代码分析，IDEA中还可以自动生成项目类图，大大的提高了开发效率。在本次系统开发中我们使用 IDEA 编写代码实现系统后台。

**2.1.2 Visual Studio Code**

Visual Studio Code是一款免费开源的现代化轻量级代码编辑器，它可以与Vue技术进行良好的整合，而且它还支持开发语言的语法高亮、智能代码补全、自定义热键、括号匹配、代码片段、代码对比Diff、GIT等特性，尤其是它支持大量的插件扩展，提高开发效率，非常适合这次的系统开发。在本次系统开发中我们使用 Visual Studio Code 编写系统页面。

**2.1.3 Edraw Max**

亿图图示是一个简单易用的图形表达工作台，既能实现流程图、架构图、工程图、思维导图等数百种专业领域图形图表的绘制，它的一大特色就是UML图的制作，使用亿图图示很方便的对UML图进行制作，同时亿图图示提供了一个白板进行头脑风暴和任意编排，轻松的实现数据和创意的可视化呈现。还有亿图图示支持Windows、Mac、Linux及网页版，可以一端创作，多端同步。在本次系统开发中我们使用 Edraw Max 对项目图形进行制作。

**2.1.4 Axure RP9**

Axure RP 9.0是一款界面清晰直观、功能丰富全面、操作便捷快速的快速原型设计工具。在本次系统开发中我们使用 Axure RP9 对页面原型进行制作，使用这款软件对页面原型进行制作有助于节省系统开发时间，同时Axure RP 9.0拥有全新的硬件加速渲染引擎，专为加速保存和加载而设计的文件结构，以及用于平滑缩放和更快编辑的简化画布。

**2.2后台技术**

|  |  |  |
| --- | --- | --- |
| **组件** | **作用** | **官网** |
| SpringBoot | 微服务框架 | https://spring.io/projects/spring-boot/ |
| Mybatis-Plus | ORM框架 | https://mp.baomidou.com/ |
| Mybatis-Generator | 代码生成插件 | https://zhile.io/2019/04/23/mybatis-code-helper-pro-crack.html |
| Shiro | 安全框架 | http://shiro.apache.org/ |
| Zookeeper | 分布式协调服务 | http://zookeeper.apache.org/ |
| Druid | 数据库连接池 | https://github.com/alibaba/druid |
| Swagger2 | 接口测试框架 | http://swagger.io/ |
| Dubbo-Admin | 服务治理项目 | http://dubbo.apache.org/zh-cn/blog/dubbo-admin.html |
| Dubbo | 远程调用框架 | http://dubbo.apache.org/zh-cn/ |
| Log4J | 日志组件 | http://logging.apache.org/log4j/1.2/ |
| Git | 版本管理 | https://git-scm.com/ |
| Maven | 项目构建管理 | http://maven.apache.org/ |
| Jackson | 数据序列化 | https://github.com/FasterXML/jackson |
| Lombok | 简化编码插件 | <https://projectlombok.org/> |

**2.3前端技术**

**2.3.1 Vue**

Vue是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue被设计为可以自底向上逐层应用。Vue的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue也完全能够为复杂的单页应用提供驱动。

**2.3.2 Element UI**

Element UI是一套采用Vue 2.0作为基础框架实现的组件库，一套为开发者、设计师和产品经理准备的基于Vue 2.0的组件库，提供了配套设计资源，帮助网站快速成型。

**2.3.3 Vue.js Ajax**

Vue. js2.0版本推荐使用axios来完成ajax请求。Axios是一个基于Promise的HTTP库，可以用在浏览器和node.js中。

**2.4数据库技术**

**2.4.1 MySQL**

MySQL是一种关系型数据库管理系统，是目前最流行的关系型数据库管理系统之一，由瑞典 MySQL AB公司开发，属于Oracle旗下产品。在WEB应用方面，MySQL是最好的RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件之一。我们通常都是使用Mysql作为数据库，因为MySQL是开源免费的，并且方便扩展。因此在本次系统开发中我们使用Mysql存放系统中的所有数据信息。

**2.4.2 Redis**

Redis是目前比较热门的NOSQL系统之一，它是一个开源的使用 c 语言编写的 key-value 存储系统（区别于 MySQL 的二维表格的形式存储）。和 Memcache 类似，但很大程度补偿了 Memcache 的不足。和 Memcache 一样，Redis 数据都是缓存在计算机内存中，不同的是，Memcache 只能将数据缓存到内存中，无法自动定期写入硬盘，这就表示，一断电或重启，内存清空，数据丢失。所以 Memcache 的应用场景适用于缓存无需持久化的数据。而Redis不同的是它会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，实现数据的持久化。在本次系统开发中我们使用Redis对系统中的特定数据进行缓存。

1. **系统分析**

**3.1可行性分析**

**3.1.1技术可行性**

本系统采用IDEA作为后端开发工具，该软件在业界被认为是最好的Java开发工具，前端开发使用Visual Studio Code，这款软件对我所用到的前端技术有较好的支持。其中：前端使用Vue技术进行编写，使用vue-element-admin作为前端框架来进行二次开发；后台采用SpringBoot框架来进行分布式开发；使用Shiro框架为系统提供安全性；采用Mysql作为数据库来完成对数据的存储，连接数据库的技术使用的是Mybatis框架及其增强工具Mybatis-Plus，并使用Druid作为数据库连接池；使用Redis对字典数据进行缓存，提高查询速度；前后端采用前后端分离的模式，通过Vue.js Ajax(axios)作为接口获取后端数据，数据采用JSON格式进行交互；使用Swagger框架配合Yapi进行接口测试。这些技术大部分都是目前市面上流行的技术，实用性高。项目主要功能所涉及的一些技术都能是能够实现的，因此，本系统在技术上是可行的。

**3.1.2经济可行性**

经济可行性研究是对组织的经济现状进行分析，对系统建设运行和维护费用进行估算，对系统建成后可能取得的社会和经济效益进行估计。本系统拥有强大的进销存数据功能，大大简化了员工的日常工作，提了工作效率，大大降低了传统人工管理过程中数据易错所带来的一系列的不良反应，提高了医院的运行效益。可以节省大量的人力物力，并且现在不会存在之前人工操作产生的人工误差的情。可以说是本系统大大提高了医院的经济效益。在本系统的开发过程中，唯一需要用经济开销的地方就是阿里云的云服务器，但是这里的开销不大，在可接受范围。因此，本系统在经济上也是可行的。

**3.1.3操作可行性**

本项目拥有友好的交互界面，所以每位医院员工都能够轻松学会如何操作本系统。操作人员只需要把按步骤把数据输入进去，数据都是自动录入并进行相应的计算，因此可以省去以前需要手动计算的部分，并且大大的提高了效率。再者以前都是使用手动计算，这种方式不仅容易出错，而且还费时费力；现在是互联网+的时代，大部分计算都是由程序自动计算并得出结果，这种做法不仅在减小了出错概率的同时还节省了人力物力、提高了工作效率。可以说对医院的运转节省了许多不必要的时间浪费。因此，本系统在操作上是可行的。

**3.1.4可行性结论**

我们从技术、经济、操作这三个方面对本系统的可行性进行了分析。通过上面三个方面的可行性分析，我们得出结论，本系统通过可行性分析。

**3.2需求分析**

需求分析阶段是系统开发的第一个阶段，也是系统开发周期中一个非常重要的阶段。该阶段主要是为调查和分析用户的业务活动和数据的使用情况，弄清所用数据的种类、范围、数量以及它们在业务活动中交流的情况，确定用户对数据库系统的使用要求和各种的约束条件等，形成用户需求规约。

**3.2.1需求调研**

本系统的业务涉及到医院的宣传部、计划部、采购部、仓储部、审核部门及管理部门等多个部门，并且每个部门都有其各自的需求特点。

根据从网上查阅资料等方式我们得悉，医院的管理部门需要对用户、科室、角色等信息进行管理，其中管理人员需要为用户分配角色，并为角色分配权限。宣传部门还需要有公告功能，能够发放公告或者通知。药品的进销存业务流程的开始是制定采购计划，此任务由计划部门负责完成，完成的采购计划交由审核部门进行审核。如果审核计划未通过，审核计划会被打回，由计划部门进行重新指定。如果采购计划审核通过，采购部门就可以根据计划进行采购，采购回来的各种药品会送到仓储部门进行入库，入库成功后会生成一份入库单，记录每批次入库的详情情况。

通过上述分析，我们得出本系统的用户大致分为这六大类：计划部门、采购部门、仓储部门、宣传部门、审核部门、管理部门。下面是对各类用户进行需求分析的结果：  
 计划部门：负责制定采购计划，包括对采购计划的增加以及修改。同时，这部分人员可以查看系统的仓储管理、药品信息管理等相应资料。

审核部门：负责对采购计划进行审核，拥有对采购计划予以通过或不通过的权力。同时可以查看采购计划的详细信息，也可以查看仓储管理等相应信息。

采购部门：负责对审核成功的采购计划进行采购。

仓储部门：负责对已经采购成功的采购计划进行入库操作。同时仓储部门可以查看仓库中的各种信息，可以知道每批次的入库情况。

宣传部门：负责信息的通知，发放的类型有公告和通知。

管理部门：可以操作系统中的所有功能，并且可以对系统各部门数据进行维护，这些数据主要包括用户数据、科室数据、角色数据、菜单数据、字典数据等。同时管理部门能对用户进行分配角色等。

**3.2.2功能需求**

系统的功能性分析可以反映一个系统能够完成的各种功能，他能清晰明确的把这个系统要完成的功能展示给后续的开发人员和使用者。本系统的具体功能如下：

1. **功能分析**

* 系统允许用户登录
* 系统允许用户进行相应的数据操作
* 系统允许管理部门人员创建、修改和删除用户
* 系统允许管理部门人员重置用户密码
* 系统允许管理部门人员为用户分配角色
* 系统允许管理部门人员科室信息
* 系统允许管理部门人员对角色数据进行管理
* 系统允许管理部门人员对角色进行授权
* 系统允许管理部门人员对字典数据进行管理
* 系统允许宣传部门人员对公告或者通知进行发放。
* 系统允许宣传部门人员对公告进行修改、删除
* 系统允许仓储部门人员对采购计划进行入库
* 系统允许仓储部门人员对仓库信息进行查看
* 系统允许审核部门人员对采购计划进行审核
* 系统允许计划部门人员对药品信息进行访问
* 系统允许计划部门人员制定采购计划
* 系统允许采购部门人员对生产厂商及供应商信息进行访问
* 系统会记录每位用户的登录信息
* 系统会记录每位用户的操作信息

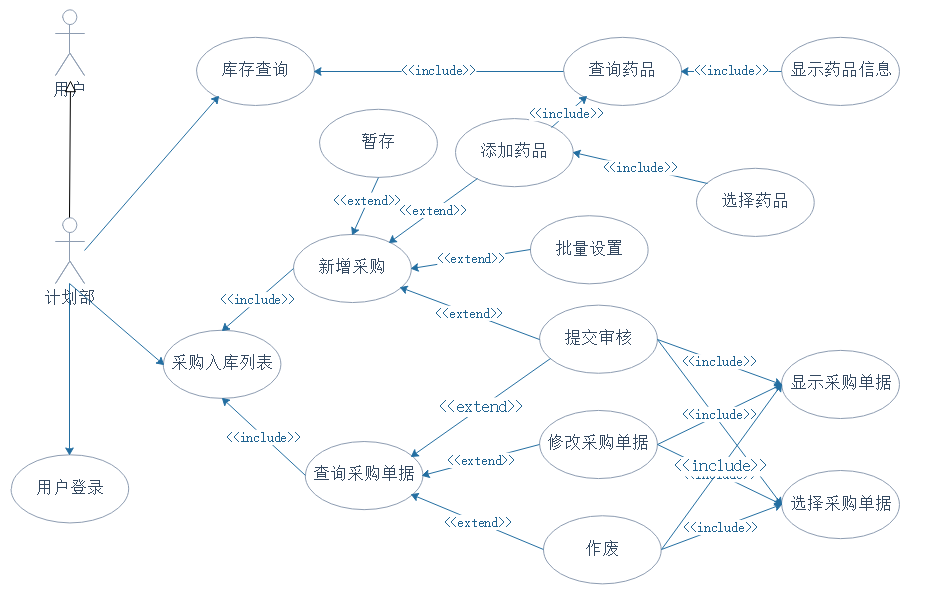
1. **非功能性分析**

* 网页的界面应美观
* 网页相应速度应该尽量快
* 用户填写的信息应该尽量详细且方便，采用选择、勾选和输入方式

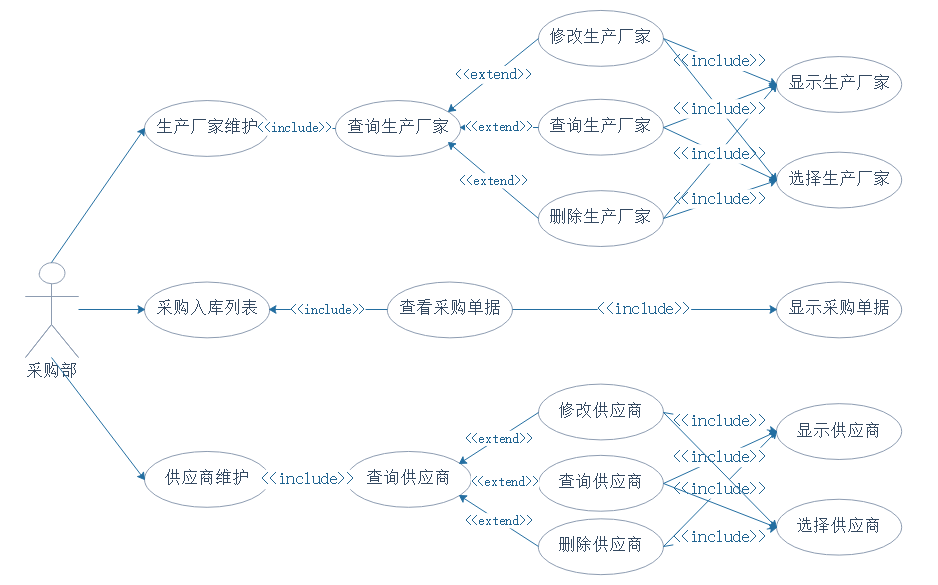
**3.2.3用例图**

本系统可以分为系统管理员和宣传部门等六种不同类型的用户角色。本系统根据部门划分成六个用例图。以下是通过部门类型进行划分的六个用例图：

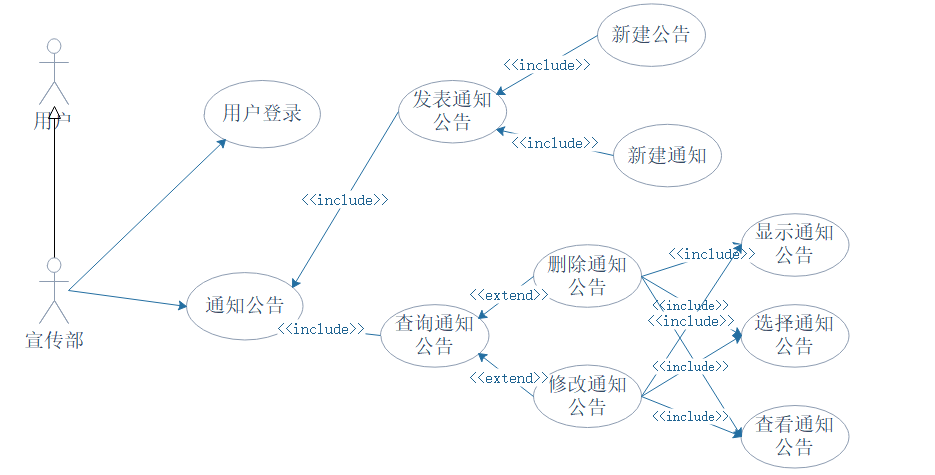
1. **计划部**



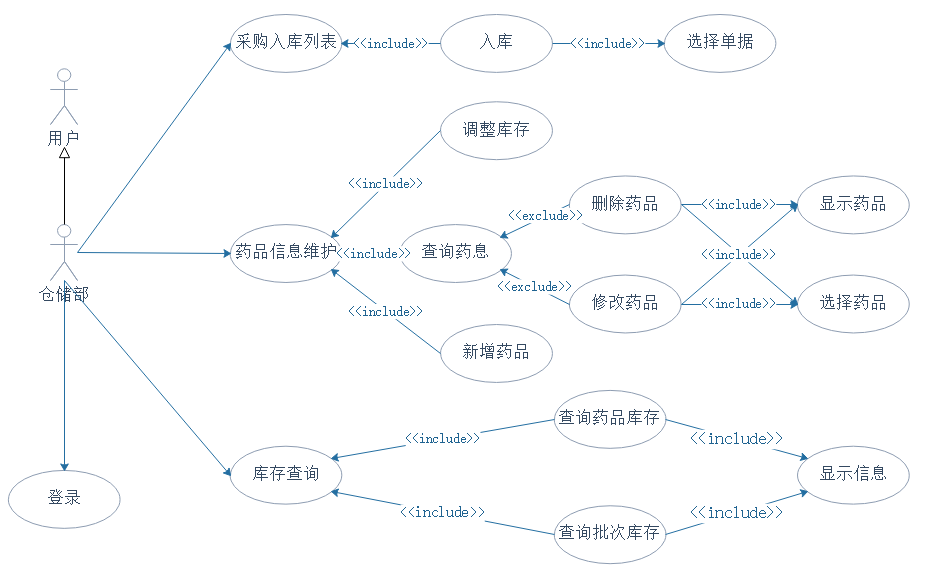
1. **采购部**



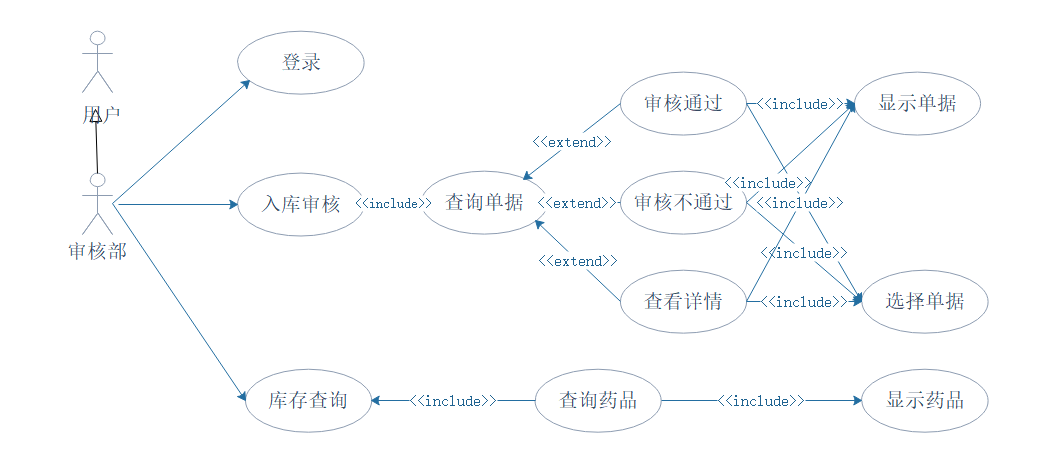
1. **宣传部**



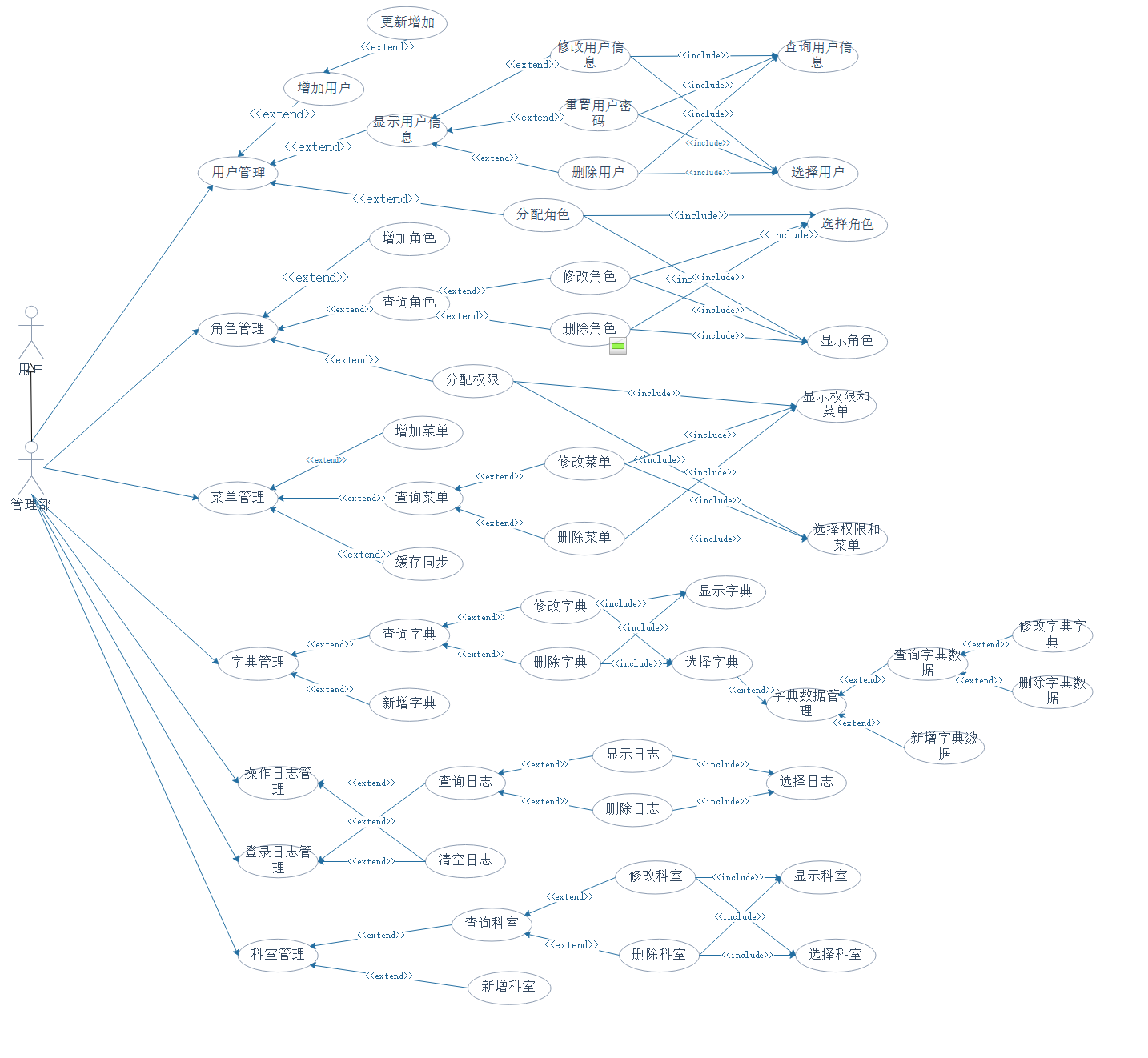
1. **仓储部**



1. **审核部**



1. **管理部**



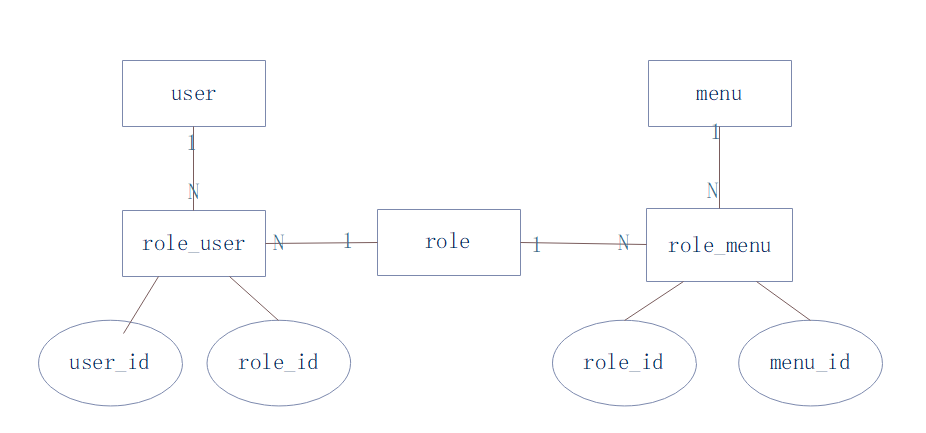
1. **数据库设计**

**4.1 概念设计**

概念设计阶段是将用户要求描述的现实世界，通过对其中住处的分类、聚集和概括，建立起抽象的概念数据模型。该概念模型常用的表达方式为实体-关系（E-R）模型。

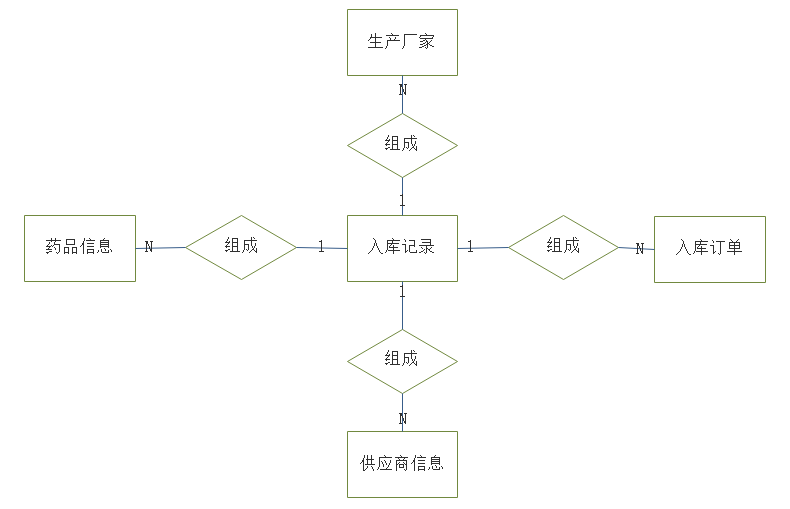
1. **用户、角色、菜单**

用户、角色、菜单三张表使用RBAC的设计模式进行设计的关系，其中用户表与角色表通过角色用户表进行关联，菜单表与角色表通过角色菜单进行关联，下图为五张表的ER设计图：



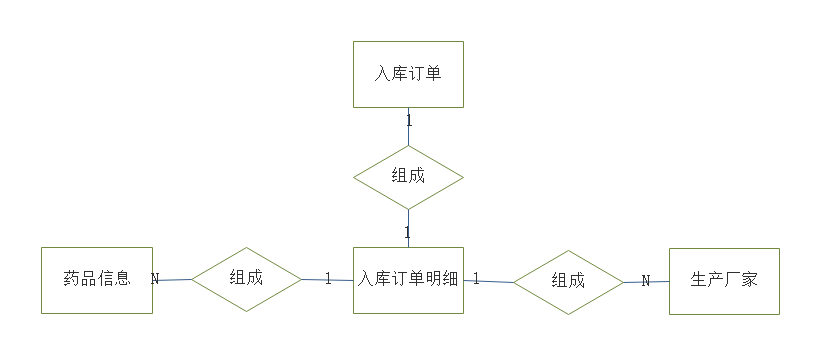
1. **入库记录**

入库记录中包含生产厂家、药品、供应商、入库订单的信息。



1. **入库订单明细**

一个入库订单明细由多种药品，多个生产厂家和一个入库订单组成。



1. **其余表的E-R图**

有操作日志、科室、字典数据、登录日志、通知公告等表。



**4.2 逻辑设计**

数据库逻辑设计的主要任务是将现实世界的概念数据模型设计成数据库的一种逻辑模式，即根据某种规则对系统E-R图进行转化，并从E-R图的最终转化结果分析出系统的数据模型。与此同时，可能还需为各种数据处理应用领域产生相应的逻辑子模式。这一步设计的结果就是所谓的“逻辑数据库”。由于本系统使用MySQL数据库，所以在这一步的目标是将系统E-R图转化为系统的关系模型。

根据上一阶段概念设计所得的E-R图，得出一个基本的小型药品资源管理系统数据库，本系统中大致包括19张表，分别存放相应功能的数据信息，其中有关身份授权的数据库使用了RBAC模式进行设计。

**4.2.1系统表RBAC**

1. **表名：sys\_dept：部门/科室表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | dept\_id | bigint(20) | NO | 是 | 部门科室 id |
| 2 | dept\_name | varchar(30) | YES | 否 | 部门名称 |
| 3 | reg\_number | int(11) | YES | 否 | 挂号编号 |
| 4 | dept\_number | varchar(50) | YES | 否 | 科室编号 |
| 5 | order\_num | int(4) | YES | 否 | 显示顺序 |
| 6 | dept\_leader | varchar(20) | YES | 否 | 负责人 |
| 7 | leader\_phone | varchar(11) | YES | 否 | 联系电话 |
| 8 | status | char(1) | YES | 否 | 部门状态（0正常 1停用） |
| 9 | del\_flag | char(1) | YES | 否 | 删除标志（0正常 1删除） |
| 10 | create\_time | datetime | YES | 否 | 创建时间 |
| 11 | update\_time | datetime | YES | 否 | 更新时间 |
| 12 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 13 | update\_by | varchar(64) | YES | 否 | 更新者 |

1. **表名：sys\_role：角色信息表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | role\_id | bigint(20) | NO | 是 | 角色 ID |
| 2 | role\_name | varchar(30) | NO | 否 | 角色名称 |
| 3 | role\_code | varchar(30) | YES | 否 | 角色权限编码 |
| 4 | role\_sort | int(4) | NO | 否 | 显示顺序 |
| 5 | remark | varchar(500) | YES | 否 | 备注 |
| 6 | status | char(1) | NO | 否 | 角色状态（0正常 1停用） |
| 7 | create\_time | datetime | YES | 否 | 创建时间 |
| 8 | update\_time | datetime | YES | 否 | 更新时间 |
| 9 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 10 | update\_by | varchar(64) | YES | 否 | 更新者 |
| 11 | del\_flag | char(1) | YES | 否 | 删除标志（0存在 2删除） |

1. **表名：sys\_menu：菜单权限表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | menu\_id | bigint(20) | NO | 是 | 菜单 ID |
| 2 | parent\_id | bigint(20) | YES | 否 | 父菜单 ID |
| 3 | parent\_ids | varchar(255) | YES | 否 | 父节点 ID 集合 |
| 4 | menu\_name | varchar(50) | NO | 否 | 菜单名称 |
| 5 | menu\_type | char(1) | YES | 否 | 菜单类型 |
| 6 | percode | varchar(100) | YES | 否 | 权限标识 |
| 7 | path | varchar(200) | YES | 否 | 路由地址 |
| 8 | remark | varchar(500) | YES | 否 | 备注 |
| 9 | status | char(1) | NO | 否 | 菜单状态（0正常 1停用） |
| 10 | create\_time | datetime | YES | 否 | 创建时间 |
| 11 | update\_time | datetime | YES | 否 | 更新时间 |
| 12 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 13 | update\_by | varchar(64) | YES | 否 | 更新者 |

1. **表名：sys\_user：用户信息表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | user\_id | bigint(20) | NO | 是 | 用户 ID |
| 2 | dept\_id | bigint(20) | YES | 否 | 部门 ID |
| 3 | user\_name | varchar(30) | NO | 否 | 用户账号 |
| 4 | user\_type | char(1) | YES | 否 | 用户类型 |
| 5 | sex | char(1) | YES | 否 | 用户性别 |
| 6 | age | int(11) | YES | 否 | 年龄 |
| 7 | picture | varchar(255) | YES | 否 | 头像 |
| 8 | background | char(255) | YES | 否 | 学 历 |
| 9 | phone | varchar(11) | NO | 否 | 电话 |
| 10 | email | varchar(50) | YES | 否 | 用户邮箱 |
| 11 | strong | varchar(255) | YES | 否 | 擅长 |
| 12 | honor | varchar(255) | YES | 否 | 荣誉 |
| 13 | introduction | text | YES | 否 | 简介 |
| 14 | user\_rank | char(2) | YES | 否 | 医生级别 |
| 15 | password | varchar(100) | YES | 否 | 密码 |
| 16 | last\_login\_time | datetime | YES | 否 | 最后一次登录时间 |
| 17 | last\_login\_ip | varchar(50) | YES | 否 | 最后登陆 IP |
| 18 | status | char(1) | YES | 否 | 帐号状态 |
| 19 | create\_time | datetime | YES | 否 | 创建时间 |
| 20 | update\_time | datetime | YES | 否 | 更新时间 |
| 21 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 22 | update\_by | varchar(64) | YES | 否 | 更新者 |
| 23 | salt | varchar(35) | YES | 否 | 盐 |
| 24 | del\_flag | char(1) | YES | 否 | 删除标志 |
| 25 | scheduling\_flag | char(1) | YES | 否 | 是否需要参与排班 |

1. **表名：sys\_role\_menu：菜单和角色关联表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | role\_id | bigint(20) | NO | 是 | 角色 ID |
| 2 | menu\_id | bigint(20) | NO | 是 | 菜单 ID |

1. **表名：sys\_role\_user：用户和角色关联表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | user\_id | bigint(20) | NO | 是 | 用户 ID |
| 2 | role\_id | bigint(20) | NO | 是 | 角色 ID |

**4.2.2 ERP业务表**

1. **表名：stock\_inventory\_log：入库记录表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | inventory\_log\_id | varchar(25) | NO | 是 | 入库 ID |
| 2 | purchase\_id | varchar(25) | YES | 否 | 采购单据 ID |
| 3 | medicines\_id | varchar(20) | YES | 否 | 药品 ID |
| 4 | inventory\_log\_num | int(10) | YES | 否 | 入库数量 |
| 5 | trade\_price | decimal(10,2) | YES | 否 | 批发价 |
| 6 | prescription\_price | decimal(10,2) | YES | 否 | 处方价 |
| 7 | trade\_total\_amount | decimal(10,2) | YES | 否 | 批发额 |
| 8 | prescription\_total\_amount | decimal(10,2) | YES | 否 | 处方额 |
| 9 | batch\_number | varchar(20) | YES | 否 | 药品生产批次号 |
| 10 | medicines\_name | varchar(120) | YES | 否 | 药品名称 |
| 11 | medicines\_type | char(1) | YES | 否 | 药品分类 |
| 12 | prescription\_type | char(1) | YES | 否 | 处方类型 |
| 13 | producter\_id | varchar(50) | YES | 否 | 生产厂家 ID |
| 14 | conversion | int(10) | YES | 否 | 换算量 |
| 15 | unit | varchar(50) | YES | 否 | 单位（g/条） |
| 16 | provider\_id | varchar(20) | YES | 否 | 供应商 ID |
| 17 | create\_time | datetime | YES | 否 | 创建时间 |
| 18 | create\_by | varchar(64) | YES | 否 | 创建者 |

1. **表名：stock\_medicines：药品信息表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | medicines\_id | bigint(20) | NO | 是 | 药品 ID |
| 2 | medicines\_number | varchar(60) | YES | 否 | 药品编号 |
| 3 | medicines\_name | varchar(120) | YES | 否 | 药品名称 |
| 4 | medicines\_type | char(1) | YES | 否 | 药品分类 |
| 5 | prescription\_type | char(1) | YES | 否 | 处方类型 |
| 6 | prescription\_price | decimal(10,2) | YES | 否 | 处方价格 |
| 7 | unit | varchar(50) | YES | 否 | 单位（g/条） |
| 8 | conversion | int(10) | YES | 否 | 换算量 |
| 9 | keywords | varchar(20) | YES | 否 | 关键字 |
| 10 | producter\_id | varchar(50) | YES | 否 | 生产厂家 ID |
| 11 | status | char(1) | YES | 否 | 药品状态 |
| 12 | medicines\_stock\_num | decimal(10,0) | YES | 否 | 库存量 |
| 13 | medicines\_stock\_danger\_num | decimal(10,0) | YES | 否 | 预警值 |
| 14 | create\_time | datetime | YES | 否 | 创建时间 |
| 15 | update\_time | datetime | YES | 否 | 更新时间 |
| 16 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 17 | update\_by | varchar(64) | YES | 否 | 更新者 |
| 18 | del\_flag | char(1) | YES | 否 | 删除状态 |

1. **表名：stock\_producter：生产厂家表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | producter\_id | bigint(20) | NO | 是 | 厂家 ID |
| 2 | producter\_name | varchar(80) | YES | 否 | 厂家名称 |
| 3 | producter\_code | varchar(80) | YES | 否 | 厂家简码 搜索用 |
| 4 | producter\_address | varchar(80) | YES | 否 | 厂家地址 |
| 5 | producter\_tel | varchar(80) | YES | 否 | 厂家电话 |
| 6 | producter\_person | varchar(80) | YES | 否 | 联系人 |
| 7 | keywords | varchar(100) | YES | 否 | 关键字 |
| 8 | status | char(1) | YES | 否 | 状态标志 |
| 9 | create\_time | datetime | YES | 否 | 创建时间 |
| 10 | update\_time | datetime | YES | 否 | 更新时间 |
| 11 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 12 | update\_by | varchar(64) | YES | 否 | 更新者 |

1. **表名：stock\_provider：供应商信息表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | provider\_id | bigint(20) | NO | 是 | 供应商 id |
| 2 | provider\_name | varchar(30) | YES | 否 | 供应商名称 |
| 3 | contact\_name | varchar(50) | YES | 否 | 联系人名称 |
| 4 | contact\_mobile | varchar(20) | YES | 否 | 联系人手机 |
| 5 | contact\_tel | varchar(11) | YES | 否 | 联系人电话 |
| 6 | bank\_account | varchar(30) | YES | 否 | 银行账号 |
| 7 | provider\_address | varchar(100) | YES | 否 | 供应商地址 |
| 8 | status | char(1) | YES | 否 | 状态 |
| 9 | del\_flag | char(1) | YES | 否 | 删除标志 |
| 10 | create\_time | datetime | YES | 否 | 创建时间 |
| 11 | update\_time | datetime | YES | 否 | 更新时间 |
| 12 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 13 | update\_by | varchar(64) | YES | 否 | 更新者 |

1. **表名：stock\_purchase：入库订单表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | purchase\_id | varchar(25) | NO | 是 | 制单号 ID 全局ID雪花算法 |
| 2 | provider\_id | varchar(20) | YES | 否 | 供应商 ID |
| 3 | purchase\_trade\_total\_amount | decimal(10,2) | YES | 否 | 采购批发总额 |
| 4 | status | char(1) | YES | 否 | 单据状态 |
| 5 | apply\_user\_id | bigint(20) | YES | 否 | 申请人 ID |
| 6 | apply\_user\_name | varchar(20) | YES | 否 | 申请人名称 |
| 7 | storage\_opt\_user | varchar(20) | YES | 否 | 入库操作人 |
| 8 | storage\_opt\_time | datetime | YES | 否 | 入库操作时间 |
| 9 | create\_time | datetime | YES | 否 | 创建时间 |
| 10 | update\_time | datetime | YES | 否 | 更新时间 |
| 11 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 12 | update\_by | varchar(64) | YES | 否 | 更新者 |
| 13 | examine | varchar(255) | YES | 否 | 审核信息 |

1. **表名：stock\_purchase\_item：入库订单明细表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | item\_id | varchar(25) | NO | 是 | 详情 ID |
| 2 | purchase\_id | varchar(25) | YES | 否 | 采购单据 ID |
| 3 | medicines\_id | varchar(20) | YES | 否 | 药品 ID |
| 4 | purchase\_number | int(10) | YES | 否 | 采购数量 |
| 5 | trade\_price | decimal(10,2) | YES | 否 | 批发价 |
| 6 | trade\_total\_amount | decimal(10,2) | YES | 否 | 批发额 |
| 7 | batch\_number | varchar(20) | YES | 否 | 药品生产批次号 |
| 8 | remark | varchar(30) | YES | 否 | 备注 |
| 9 | medicines\_name | varchar(120) | YES | 否 | 药品名称 |
| 10 | medicines\_type | char(1) | YES | 否 | 药品分类 |
| 11 | prescription\_type | char(1) | YES | 否 | 处方类型 |
| 12 | producter\_id | varchar(50) | YES | 否 | 生产厂家 ID |
| 13 | conversion | int(10) | YES | 否 | 换算量 |
| 14 | unit | varchar(50) | YES | 否 | 单位（g/条） |
| 15 | keywords | varchar(20) | YES | 否 | 关键字 |

**4.2.3系统其它配置表**

1. **表名：sys\_dict\_data：字典数据表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | dict\_code | bigint(20) | NO | 是 | 字典编码 |
| 2 | dict\_sort | int(4) | YES | 否 | 字典排序 |
| 3 | dict\_label | varchar(100) | YES | 否 | 字典标签 |
| 4 | dict\_value | varchar(100) | YES | 否 | 字典键值 |
| 5 | dict\_type | varchar(100) | YES | 否 | 字典类型 |
| 6 | status | char(1) | YES | 否 | 状态 |
| 7 | remark | varchar(500) | YES | 否 | 备注 |
| 8 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 9 | create\_time | datetime | YES | 否 | 创建时间 |
| 10 | update\_by | varchar(64) | YES | 否 | 更新者 |
| 11 | update\_time | datetime | YES | 否 | 更新时间 |

1. **表名：sys\_dict\_type：字典类型表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | dict\_id | bigint(20) | NO | 是 | 字典主键 |
| 2 | dict\_name | varchar(100) | YES | 否 | 字典名称 |
| 3 | dict\_type | varchar(100) | YES | 否 | 字典类型 |
| 4 | status | char(1) | YES | 否 | 状态 |
| 5 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 6 | create\_time | datetime | YES | 否 | 创建时间 |
| 7 | update\_by | varchar(64) | YES | 否 | 更新者 |
| 8 | update\_time | datetime | YES | 否 | 更新时间 |
| 9 | remark | varchar(500) | YES | 否 | 备注 |

1. **表名：sys\_notice：通知公告表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | notice\_id | int(4) | NO | 是 | 公告 ID |
| 2 | notice\_title | varchar(50) | NO | 否 | 公告标题 |
| 3 | notice\_type | char(1) | NO | 否 | 公告类型 |
| 4 | notice\_content | varchar(2000) | YES | 否 | 公告内容 |
| 5 | status | char(1) | YES | 否 | 公告状态 |
| 6 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 7 | create\_time | datetime | YES | 否 | 创建时间 |
| 8 | update\_by | varchar(64) | YES | 否 | 更新者 |
| 9 | update\_time | datetime | YES | 否 | 更新时间 |
| 10 | remark | varchar(255) | YES | 否 | 备注 |

1. **表名：sys\_login\_info：系统访问记录表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | info\_id | bigint(20) | NO | 是 | 访问 ID |
| 2 | user\_name | varchar(50) | YES | 否 | 用户名称 |
| 3 | login\_account | varchar(50) | YES | 否 | 登陆账号 |
| 4 | ip\_addr | varchar(50) | YES | 否 | 登录IP 地址 |
| 5 | login\_location | varchar(255) | YES | 否 | 登录地点 |
| 6 | browser | varchar(50) | YES | 否 | 浏览器类型 |
| 7 | os | varchar(50) | YES | 否 | 操作系统 |
| 8 | login\_status | char(1) | YES | 否 | 登录状态 |
| 9 | login\_type | char(1) | YES | 否 | 登陆类型 |
| 10 | msg | varchar(255) | YES | 否 | 提示消息 |
| 11 | login\_time | datetime | YES | 否 | 访问时间 |

1. **表名：sys\_oper\_log：操作日志记录表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | oper\_id | bigint(20) | NO | 是 | 日志主键 |
| 2 | title | varchar(50) | YES | 否 | 模块标题 |
| 3 | business\_type | char(1) | YES | 否 | 业务类型 |
| 4 | method | varchar(100) | YES | 否 | 方法名称 |
| 5 | request\_method | varchar(10) | YES | 否 | 请求方式 |
| 6 | operator\_type | int(1) | YES | 否 | 操作类别 |
| 7 | oper\_name | varchar(50) | YES | 否 | 操作人员 |
| 8 | oper\_url | varchar(255) | YES | 否 | 请求 URL |
| 9 | oper\_ip | varchar(50) | YES | 否 | 主机地址 |
| 10 | oper\_location | varchar(255) | YES | 否 | 操作地点 |
| 11 | oper\_param | varchar(2000) | YES | 否 | 请求参数 |
| 12 | json\_result | varchar(2000) | YES | 否 | 返回参数 |
| 13 | status | char(1) | YES | 否 | 操作状态 |
| 14 | error\_msg | varchar(2000) | YES | 否 | 错误消息 |
| 15 | oper\_time | datetime | YES | 否 | 操作时间 |

1. **表名：sys\_check\_item：检查费用表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | check\_item\_id | bigint(20) | NO | 是 | 项目费用 ID |
| 2 | check\_item\_name | varchar(255) | NO | 否 | 项目名称 |
| 3 | keywords | varchar(10) | YES | 否 | 关键字【查询用】 |
| 4 | unit\_price | decimal(10,2) | NO | 否 | 项目单价 |
| 5 | cost | decimal(10,2) | NO | 否 | 项目成本 |
| 6 | unit | varchar(50) | YES | 否 | 单位 |
| 7 | type\_id | char(1) | YES | 否 | 项目类别 |
| 8 | status | char(1) | YES | 否 | 状态 |
| 9 | create\_time | datetime | YES | 否 | 创建时间 |
| 10 | update\_time | datetime | YES | 否 | 更新时间 |
| 11 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 12 | update\_by | varchar(64) | YES | 否 | 更新者 |

1. **表名：sys\_registered\_item：挂号项目表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 行号 | 字段 | 字段类型 | 可否为空 | 是否主键 | 描述 |
| 1 | reg\_item\_id | bigint(20) | NO | 是 | 挂号项 ID |
| 2 | reg\_item\_name | varchar(50) | YES | 否 | 挂号项目名称 |
| 3 | reg\_item\_fee | decimal(10,2) | YES | 否 | 金额 |
| 4 | create\_time | datetime | YES | 否 | 创建时间 |
| 5 | update\_time | datetime | YES | 否 | 更新时间 |
| 6 | create\_by | varchar(64) | YES | 否 | 创建者 |
| 7 | update\_by | varchar(64) | YES | 否 | 更新者 |
| 8 | status | char(1) | YES | 否 | 状态 |
| 9 | del\_flag | char(1) | YES | 否 | 删除标志 |

**4.3 物理设计**

物理设计阶段为根据特定数据库管理系统所提供的多种存储结构和存储方法等依赖于具体计算机结构的各项物理设计措施，对具体的应用任务选定最合适的物理存储结构、存取方法和存取路径等。这一步设计的结构就是“物理数据库”。

物理设计阶段一般都由数据库管理系统来完成，本系统选取Navicat Premium作为数据库后台实现平台，为此，设计完成的逻辑结构对应的物理存储结构、存取方式和存取路径均为数据库管理系统实现。

1. **系统设计**

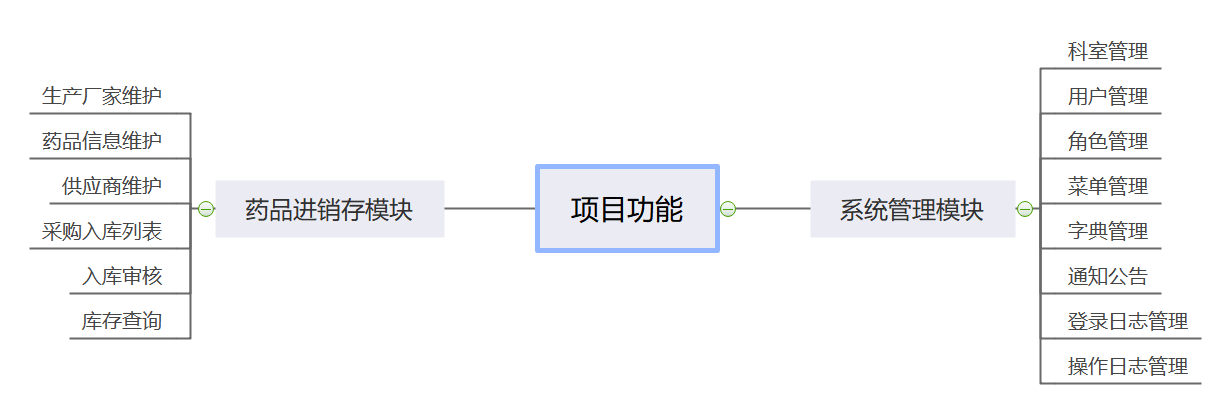
**5.1系统模块设计**

通过系统分析，本系统采用前后端分离的方式进行开发，前后端采用接口调用的方式实现分离。同时经过分析我们把系统分成两个模块：系统管理模块和药品进销存模块。

在系统管理模块中包含八个有关系统管理的子模块。科室管理用于对医院的科室部门信息进行管理；其中用户管理、角色管理、菜单管理是系统管理模块中最重要的三个子模块，这三个模块分别对用户、角色、菜单信息进行管理，这三个子模块的数据库使用RBAC的设计思维，主要通过对角色进行菜单授权从而完成对用户进行权限控制，是本次系统开发中的一个重点也是难点；通知公告模块用于对医院信息进行通知；登录日志管理和操作日志管理模块分别记录用户登录和用户操作系统的详情信息；字典管理模块包含对字典类型和字典数据的管理。

在药品进销存模块包含六个相关的子模块，该模块的难点在于采购入库列表和入库审核者两个子模块，采购入库列表包含新增单据、查询单据、作废、提交审核、入库等操作，入库审核包括对单据进行审核有审核通过和审核不通过两个功能，这两个子模块完成对采购单据进行处理，其中涉及了大量的数据、状态的改动，逻辑性较高，是系统开发的难点也是重点。生产厂家维护、药品信息维护和供应商维护这三个子模块主要是对生产厂家、药品信息和供应商的数据进行维护，例如简单的增加、修改、删除、查找操作。库存查询包含药品的库存信息和批次库存的详细情况。

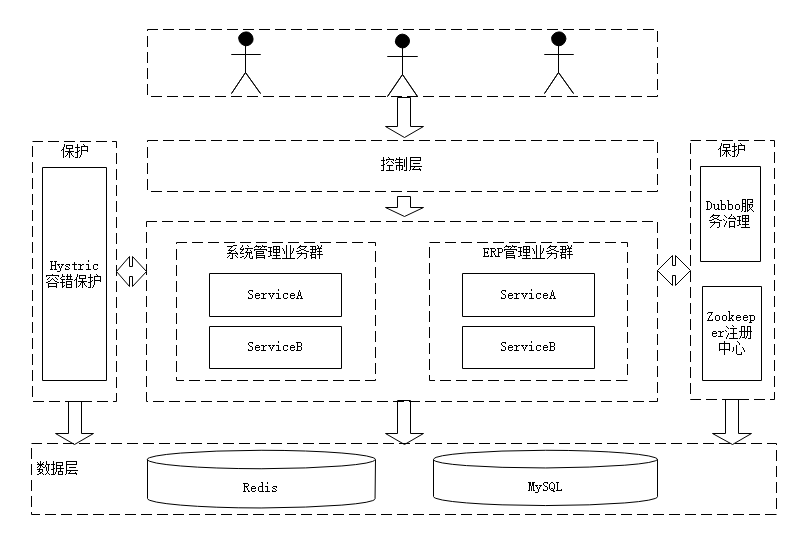
同时系统中的每个子模块具有相应的数据操作功能，例如增加、删除、修改、查找等操作。系统的模块功能如下图所示：



**5.2系统后端设计**

**5.2.1系统后端项目结构**

本系统为分布式系统，涉及到服务远程调用技术。我们把系统分为两个模块系统管理模块和药品进销存模块。用户通过控制层对模块进行操作，每个模块又包含各自的服务，其中每个模块通过dubbo进行远程服务治理，使用zookeeper对服务进行协调。通过启用Hystric对系统服务进行容错保护，解决服务提供者挂掉或者异常导致数据层出错的问题。最后数据层采用Redis和Mysql作为数据库，存储系统的数据信息。项目的结构图如下图所示：



**5.2.4项目结构设计**

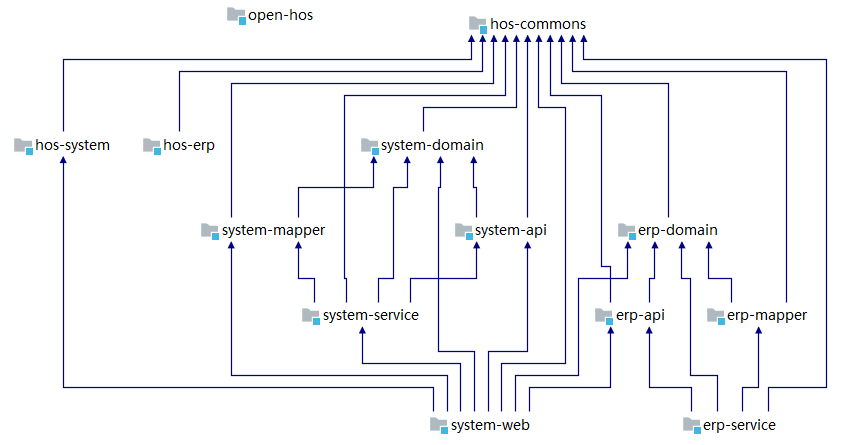
本系统使用Java语言进行开发，使用IDEA作为开发工具，项目名为：open-hos。open-hos作为本系统的父项目包含hos-commons、hos-system、hos-erp三个子项目。

hos-commons项目用于存放与系统相关的工具类，如Md5加密工具类、响应状态码常量、获取IP方法工具类等，并且hos-commons存放系统所需要的Maven依赖；hos-system和hos-erp项目都依赖于hos-commons项目。

hos-system项目作为系统管理模块，包含system-api、system-domin、system-mapper、system-service、system-web五个子项目。hos-erp项目作为药品进销存模块，包含erp-api、erp-domin、erp-mapper、erp-service这四个子项目。

项目中system-domain用于存放数据实体和数据传输对象，同时作为基础项目被其他项目所依赖；system-api主要用于编写与系统管理模块相关的业务逻辑接口；system-mapper用于对数据库进行持久化操作；system-service则是对system-api中的接口进行实现；药品进销存模块中的四个子项目与系统管理模块功能类似。

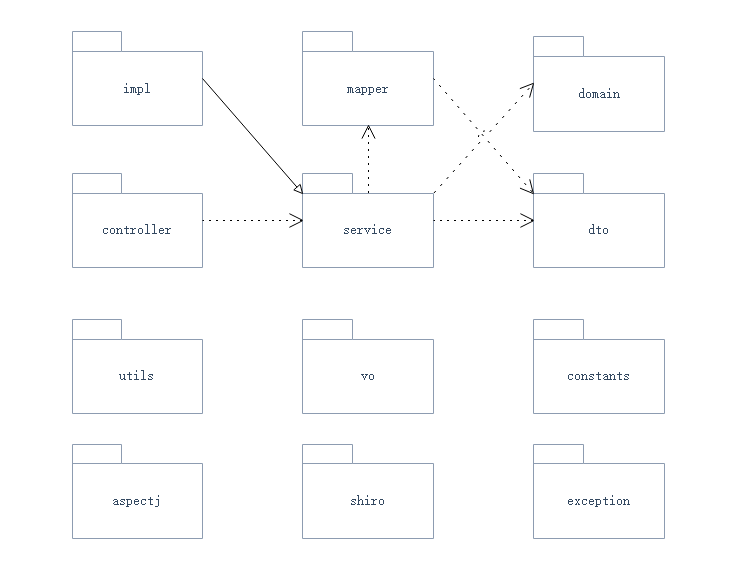
system-web存放有关系统操作的配置类，同时作为项目的controller层负责具体模块的业务流程控制，通过调用system-api和erp-api的逻辑设计接口来控制业务流程，通过接收前端H5或者App传过来的参数进行业务操作，再将处理结果返回到前端。项目结构设计图如下图所示：



**5.2.2系统包图类图**

包图说明：

* domain和dto包用于存放系统中的实体类和数据传输对象；
* mapper包作为数据持久层（dao层），作用为访问数据库，向数据库发送sql语句；
* service包作为业务逻辑层，作用为完成功能设计，调用dao层接口，接收dao层返回的数据；
* impl包对业务逻辑层进行实现；
* controller包作为系统的控制层负责前后端交互，接受前端请求，调用service层；
* utils包存放系统开发中需要用到的工具类；
* vo包存放表格数据传输对象；
* constants包用来存放系统开发中使用的常量；
* aspectj包用来存放与切面有关的类；
* shiro包用来存放与shiro框架配置有关的类；
* exception包用来存放处理系统异常的类。



**5.3系统前端设计**

本系统为药品进销存管理系统，会涉及到大量的数据显示给用户看，所以用什么方式去显示数据成为了我们首先需要考虑的问题。

通过需求分析我们知道用户需要一个美观、清晰、操作便捷的界面，使用户使用起来能得心应手，该界面操作应该简单便捷。因此我们选择使用vue-element-admin作为前端框架进行二次开发，使用Axure RP 9.0作为前端页面原型开发工具。因为vue-element-admin网站框架的界面美观且清晰，非常适合用于开发进销存系统。

解决完系统前端框架后，用什么方式呈现数据，成了我们需要考虑的一道难题。最后在经过分析之后，我们选择用表格的方式去显示系统中的数据，因为表格能够清晰的显示出数据库表中的数据，而且便于操作。下面是系统的主要模板界面，主要是查询、删除、修改和增加界面。

**5.3.1查询页面设计**

本系统每个模块的首界面都是查询界面，在查询界面中已经显示出了该模块的相关数据，如药品信息模块会显示所有的药品信息。

在查询界面上方有多个属性可供输入，输入完后点击搜索按钮，会通过你输入的信息对数据进行查询，如果没有相关信息则返回无，想要回到首页面可以点击重置按钮，其中数据的分页是使用Mybatis-Plus的分页插件实现的。

在查询页面中有新增、修改、删除这些按钮；未选择数据时修改和删除按钮是处于未激活状态，当选择了一条数据后，修改和删除按钮激活可以使用；当选择大于或等于两条数据时修改按钮处于未激活状态，而删除按钮依旧处于激活状态。



**5.3.2修改和增加页面设计**

修改和增加的界面是一致的。唯一的区别就是，增加数据的界面中表格是空的没有数据，而修改界面中，表格呈现用户选中修改数据的那一项数据。修改和增加界面的显示都是通过查询界面中的修改和增加按钮来跳转的。



**5.3.3删除页面设计**

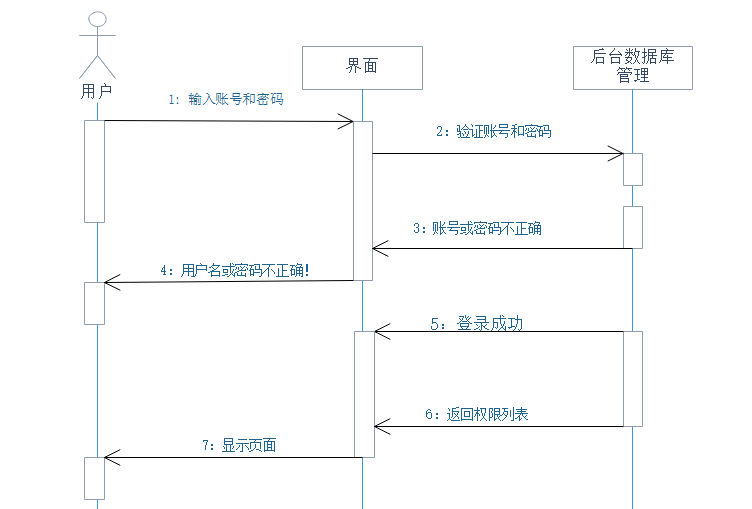
在本系统中涉及到删除数据的操作，都会显示这条提示。提示用户是否确认删除选中的数据。只有点击确认后才会删除数据。删除的提示界面也是本系统中所有需要显示提示操作的模板。



**5.4业务设计**

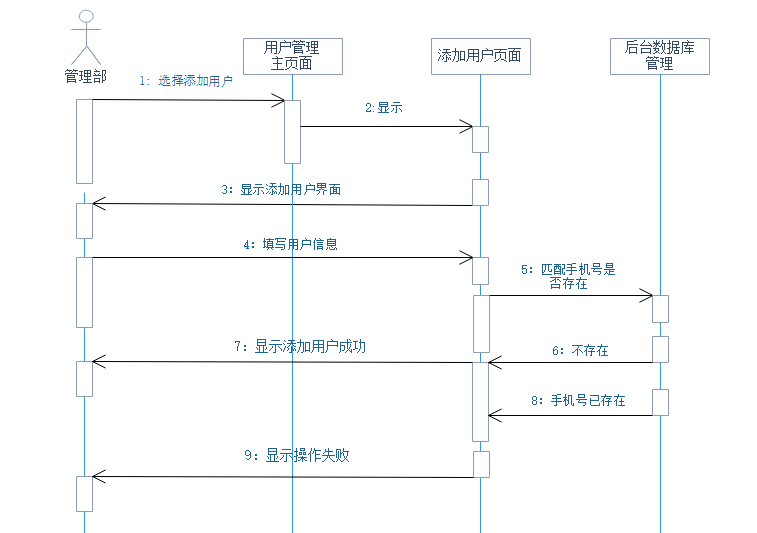
**5.4.1用户登录**

用户在登录界面输入账号和密码，前端会把输入的信息封装成一个数据结构，通过调用后端的接口传给后端，后端拿到数据后进行验证，如果验证不通过则返回用户名或密码不正确的信息给前端，前端界面接收从后端传来的信息并显示给用户看；如果验证成功，后端通过用户的角色进行查询权限列表，最后返回可显示的权限列表，前端界面通过权限列表显示用户可查看的界面。

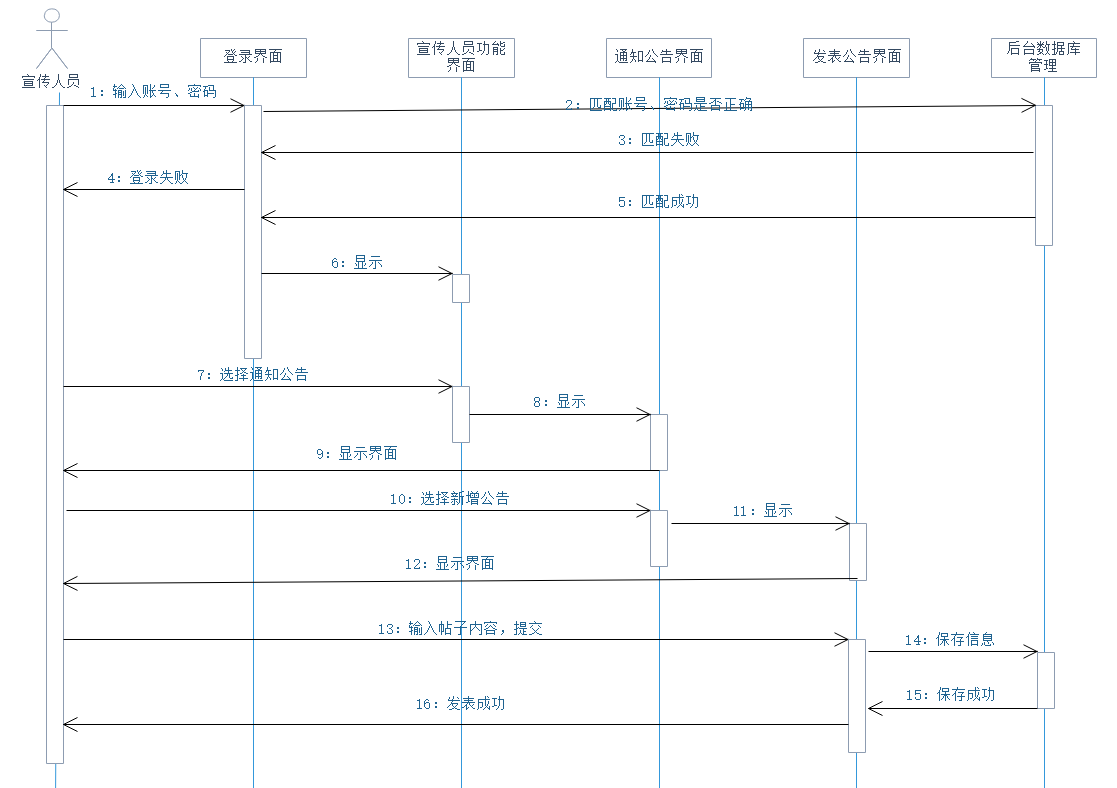


**5.4.2添加用户**

添加用户不可以添加手机号相同的用户。如果手机号在数据库中存在会提示操作失败，只有在手机号不存在情况下才能进行用户的添加。

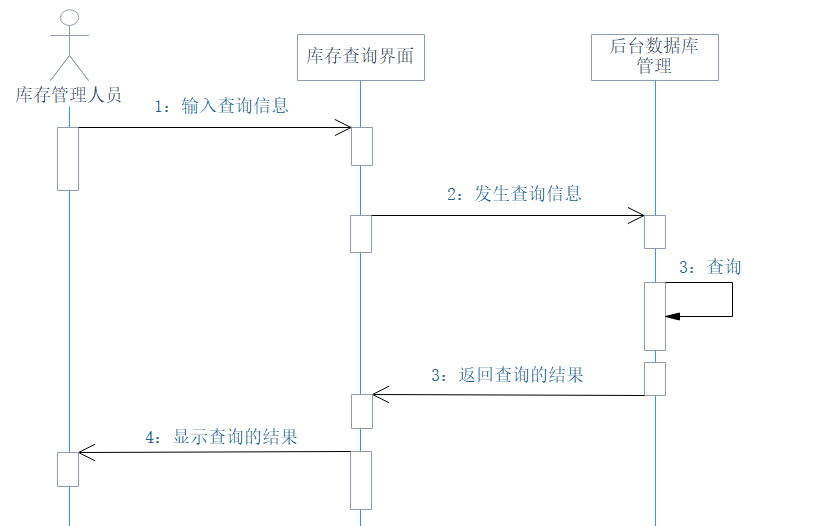


**5.4.3通知公告**



**5.4.1查询库存**

前端界面把用户输入的查询信息传送到后端，后端对数据库进行查询，最后返回查询结果给前端，前端拿到查询的结果显示给用户。

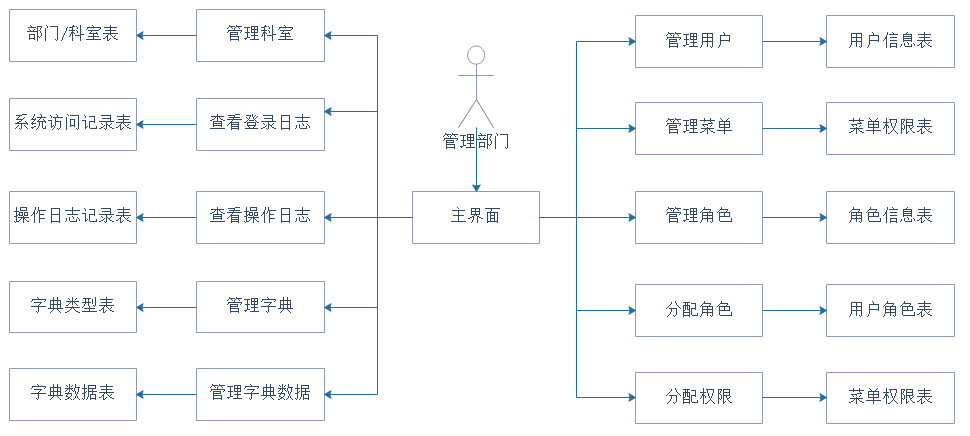


1. **系统实现**

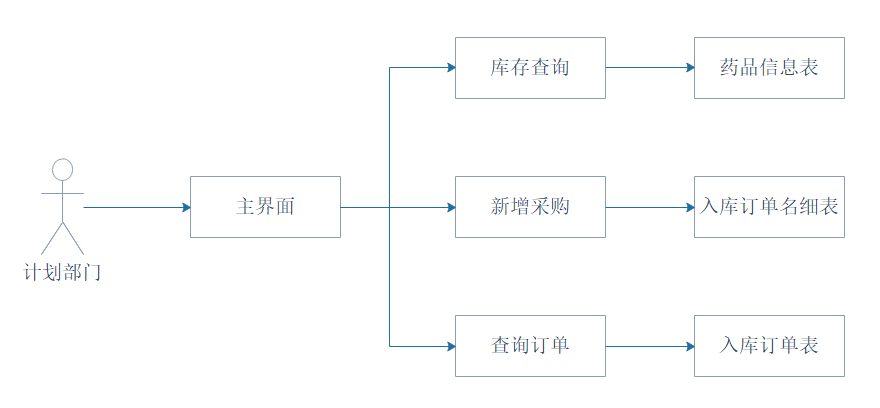
**6.1 系统数据交互关系**

根据前期需求分析的结果，用户登录后可以按照登录用户类型的不同转向不同的执行流程，系统用户界面与数据库之间的数据交互关系如下图所示。

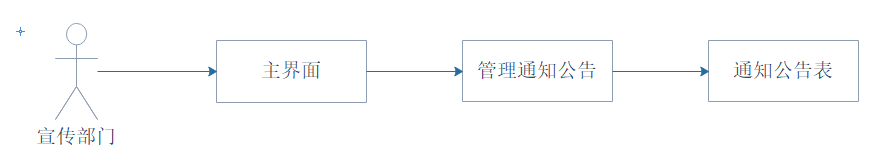
* + 1. **管理部门**



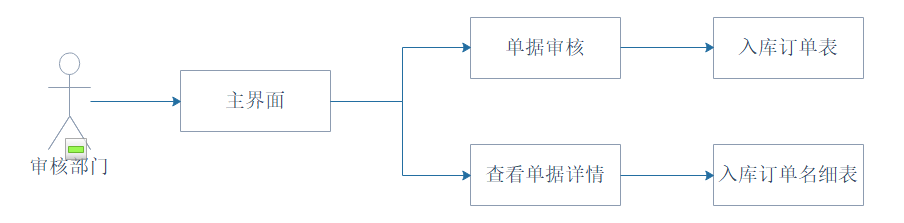
* + 1. **计划部门**



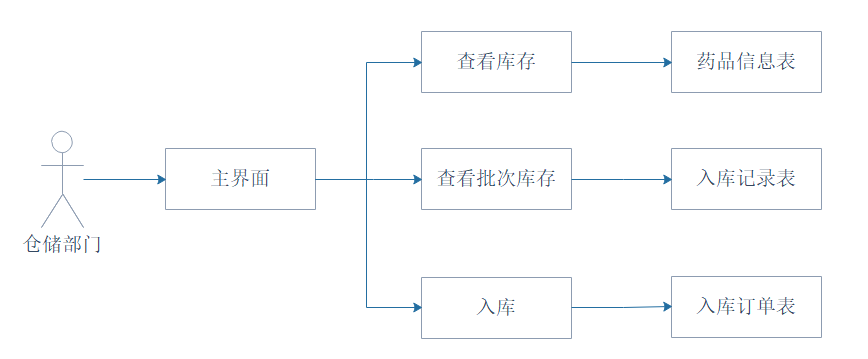
* + 1. **宣传部门**



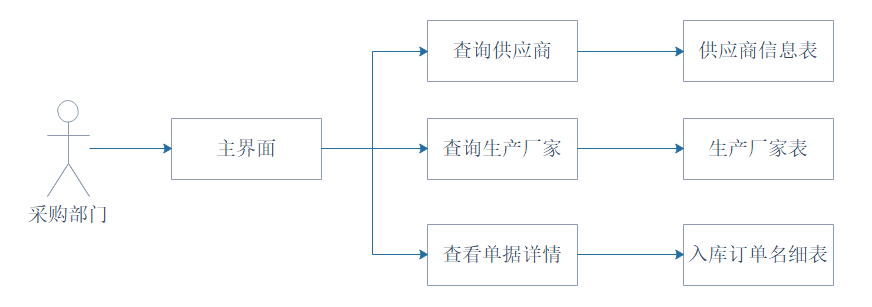
* + 1. **审核部门**



* + 1. **仓储部门**



* + 1. **采购部门**



**6.2 系统模块实现**

**6.2.1用户登录**

1. 模块描述

登录功能是对用户进入系统进行的一次身份验证的过程，同时登录功能也是可以解决用户权限问题的重要关键。登录界面要求简明易懂，每一个用户都有自己的账号和密码，通过输入账号和密码通过验证便可进入系统。因为本系统是进销存管理系统，对数据的管控比较严格，因此我们对用户登录进行了日志处理，用户每次登录都会被记录到登录日志中，这样做可以有效保证数据可追溯性，做到责任到人。用户登录模块界面如下图所示：



1. 实现流程

在登陆按钮添加单击事件，前端页面通过用户输入的账号密码构造用户登录信息，通过调用后端的接口实现登录功能。后端首先使用IDEA连接数据库创建用户的实体类，然后实现与用户有关的一些业务功能，进而使用shiro框架对用户进行认证，因为账号默认为用户的手机号，因此我们主要实现通过手机号查询用户的功能，

然后在controller层编写控制器代码，最后添加接口数据校验，在用户输入账号密码点击登陆后，先判断是否满足用户属性的验证方法，此方法是防止当账号密码为空时登录的提示，如果为空跳出一行提示词来提示用户账号和密码需要填。如果不为空，后台会把接受的用户名密码与数据库中用户信息比较核对，如果是一致则顺利登录到用户权限所对应的界面中，否则出现账号密码错误的提示，登录失败。

1. 主要代码

@Autowired

private MenuService menuService;

@Autowired

private LoginInfoService loginInfoService;

@PostMapping("login/doLogin")

public AjaxResult login(@RequestBody @Validated LoginBodyDto loginBodyDto, HttpServletRequest request) {

Logger logger = LoggerFactory.getLogger(this.getClass());

AjaxResult ajax = AjaxResult.success();

String username = loginBodyDto.getUsername();

String password = loginBodyDto.getPassword();

UsernamePasswordToken token = new UsernamePasswordToken(username, password);

Subject subject = SecurityUtils.getSubject();

LoginInfo loginInfo=createLoginInfo(request);

loginInfo.setLoginAccount(loginBodyDto.getUsername());

try {

subject.login(token);

Serializable webToken = request.getSession().getId();

ajax.put(Constants.TOKEN, webToken);

loginInfo.setMsg("登陆成功");

loginInfo.setLoginStatus(Constants.LOGIN\_SUCCESS);

loginInfo.setUserName(ShiroSecurityUtils.getCurrentUser().getUserName());

}catch (Exception e) {

logger.error("用户名或密码不正确！", e);

ajax = AjaxResult.error(HttpStatus.ERROR, "用户名或密码不正确！");

loginInfo.setMsg("用户名或密码不正确");

loginInfo.setLoginStatus(Constants.LOGIN\_ERROR);

}

loginInfoService.insertLoginInfo(loginInfo);

return ajax;

}

private LoginInfo createLoginInfo(HttpServletRequest request) {

LoginInfo loginInfo = new LoginInfo();

UserAgent userAgent = UserAgent.parseUserAgentString(request.getHeader("User-Agent"));

String ipAddr= IpUtils.getIpAddr(request);

String os=userAgent.getOperatingSystem().getName();

String browser=userAgent.getBrowser().getName();

String location= AddressUtils.getRealAddressByIP(ipAddr);

loginInfo.setIpAddr(ipAddr);

loginInfo.setLoginLocation(location);

loginInfo.setOs(os);

loginInfo.setBrowser(browser);

loginInfo.setLoginTime(DateUtil.date());

loginInfo.setLoginType(Constants.LOGIN\_TYPE\_SYSTEM);

return loginInfo;

}

@GetMapping("login/getInfo")

public AjaxResult getInfo() {

Subject subject = SecurityUtils.getSubject();

ActiveUser activeUser = (ActiveUser) subject.getPrincipal();

AjaxResult ajax = AjaxResult.success();

ajax.put("username", activeUser.getUser().getUserName());

ajax.put("password", activeUser.getUser().getPassword());

ajax.put("roles", activeUser.getRoles());

ajax.put("permission", activeUser.getPermissions());

return ajax;

}

@PostMapping("login/logout")

@Log(title = "用户退出", businessType = BusinessType.OTHER)

public AjaxResult logout() {

Subject subject = SecurityUtils.getSubject();

subject.logout();

return AjaxResult.success("用户退出成功！");

}

@GetMapping("login/getMenus")

public AjaxResult getMeuns() {

Subject subject = SecurityUtils.getSubject();

ActiveUser activeUser = (ActiveUser) subject.getPrincipal();

boolean isAdmin = activeUser.getUser().getUserType().equals(Constants.USER\_ADMIN);

SimpleUser simpleUser = null;

if(!isAdmin) {

simpleUser = new SimpleUser(activeUser.getUser().getUserId(), activeUser.getUser().getUser Name());

}

List<Menu> menus = menuService.selectMenuTree(isAdmin, simpleUser);

List<MenuTreeVo> menuTreeVos = new ArrayList<>();

for(Menu menu : menus) {

menuTreeVos.add(new MenuTreeVo(menu.getMenuId().toString(), menu.getPath()));

}

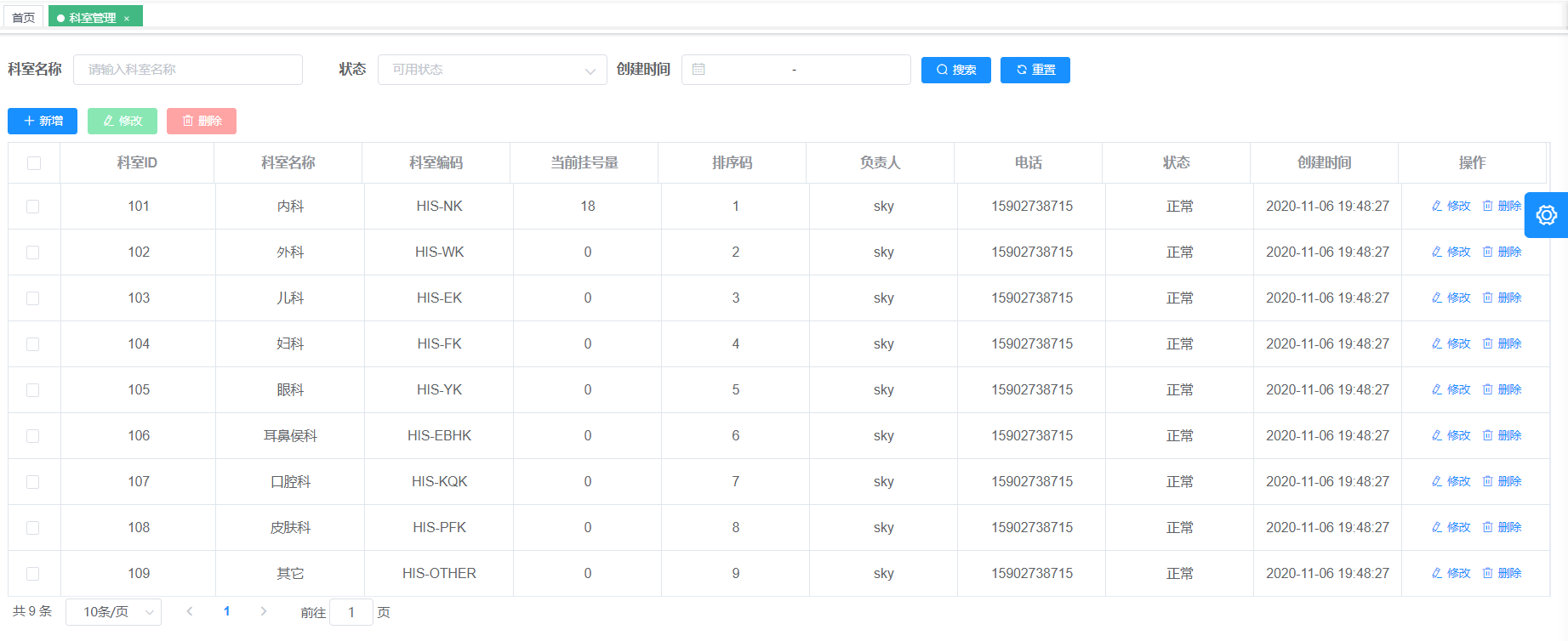
return AjaxResult.success(menuTreeVos);

}

**6.2.2科室管理**

1. 模块描述

科室管理是对医院的科室进行管理，系统初始设置科室管理只有管理员可以访问。通过科室管理，管理员可以清除的知道医院中存在多少科室。在科室管理模块中，管理员可以对科室进行增加、删除、修改科室信息、查找科室。管理员通过当前挂号量等属性，可以清楚了解科室目前的紧急情况，是否需要增添人手。科室管理模块界面如下图所示：

****

1. 实现流程

首先使用IDEA工具连接数据库生成科室的实体类和mapper层接口，创建数据传输对象，用来把数据传输到前端，创建service业务逻辑层的接口，进而创建业务逻辑实现类。最后创建科室模块的controller控制层，通过前端传来的参数请求，调用对应的业务逻辑接口，返回数据给前端，以此来实现前后端的交互。

1. 主要代码

@Autowired

private DeptService deptService;

@GetMapping("listDeptForPage")

public AjaxResult listDeptForPage(DeptDto deptDto){

DataGridView gridView = this.deptService.listPage(deptDto);

return AjaxResult.success("查询成功",gridView.getData(),gridView.getTotal());

}

@PostMapping("addDept")

@Log(title = "添加科室",businessType = BusinessType.INSERT)

public AjaxResult addDept(@Validated DeptDto deptDto) {

deptDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.deptService.addDept(deptDto));

}

@PutMapping("updateDept")

@Log(title = "修改科室",businessType = BusinessType.UPDATE)

public AjaxResult updateDept(@Validated DeptDto deptDto) {

deptDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.deptService.updateDept(deptDto));

}

@GetMapping("getDeptById/{deptId}")

public AjaxResult getDeptById(@PathVariable @Validated @NotNull(message = "科室ID不能为空") Long deptId) {

return AjaxResult.success(this.deptService.getOne(deptId));

}

@DeleteMapping("deleteDeptByIds/{deptIds}")

@Log(title = "删除科室",businessType = BusinessType.DELETE)

public AjaxResult deleteDeptByIds(@PathVariable @Validated @NotEmpty(message = "要删除的ID不能为空") Long[] deptIds) {

return AjaxResult.toAjax(this.deptService.deleteDeptByIds(deptIds));

}

@GetMapping("selectAllDept")

public AjaxResult selectAllDept(){

return AjaxResult.success(this.deptService.list());

}

**6.2.3用户管理**

1. 模块描述

用户管理模块是对医院的医护人员进行管理。通过该模块可以清晰的了解每个医护人员的信息，获取他们所在部门、手机号、学历等信息。该模块系统初始设置只有管理员可以访问。在用户管理模块中，管理员可以对用户进行查找、修改、增加、删除，管理员还可以重置用户的密码，重置的密码为手机号的后六位，在数据库存放的是密码通过MD5两次加盐之后的数据，这样做能提高了数据的安全性，并有效的防止了非法的操作对数据进行读取。同时管理员可以分配角色给用户，相应的角色具有相应的操作功能，用户获得某种角色的身份就可以进行该角色的相应操作。用户管理模块界面如下图所示：



1. 实现流程

通过IDEA创建用户的实体类和数据层接口，编写业务逻辑接口及其实现类，实现对用户增删改查操作，其中要删除角色时需要先删除用户角色表及角色菜单表中的相关数据，因为这三张表是相互连接的，需要修改mapper层的代码，最后创建controller层。其中新增用户在密码方面做了加密操作，使用shiro框；首先生成盐和加密字符串，盐使用UUID的格式，加密字符串通过盐和原本的密码使用MD5的加密方式加密两次得到，并保存在数据库中。重置密码默认重置为手机号码的后六位。同样删除用户需要实现根据用户IDS删除sys\_role\_user里面的数据的方法。分配角色功能需要修改角色模块的代码添加根据用户ID查询用户拥有的角色ID和保存角色和用户的关系的方法。

1. 主要代码

@Autowired

private UserService userService;

@GetMapping("listUserForPage")

public AjaxResult listUserForPage(UserDto userDto){

DataGridView gridView = this.userService.listUserForPage(userDto);

return AjaxResult.success("查询成功",gridView.getData(),gridView.getTotal());

}

@PostMapping("addUser")

@Log(title = "添加用户",businessType = BusinessType.INSERT)

public AjaxResult addUser(@Validated UserDto userDto) {

userDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.userService.addUser(userDto));

}

@PutMapping("updateUser")

@Log(title = "修改用户",businessType = BusinessType.UPDATE)

public AjaxResult updateUser(@Validated UserDto userDto) {

userDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.userService.updateUser(userDto));

}

@GetMapping("getUserById/{userId}")

public AjaxResult getUserById(@PathVariable @Validated @NotNull(message = "用户ID不能为空") Long userId) {

return AjaxResult.success(this.userService.getOne(userId));

}

@DeleteMapping("deleteUserByIds/{userIds}")

@Log(title = "删除用户",businessType = BusinessType.DELETE)

public AjaxResult deleteUserByIds(@PathVariable @Validated @NotEmpty(message = "要删除的ID不能为空") Long[] userIds) {

return AjaxResult.toAjax(this.userService.deleteUserByIds(userIds));

}

@GetMapping("selectAllUser")

public AjaxResult selectAllUser(){

return AjaxResult.success(this.userService.getAllUsers());

}

@PostMapping("resetPwd/{userIds}")

public AjaxResult resetPwd(@PathVariable Long[] userIds){

if(userIds.length > 0){

this.userService.resetPassWord(userIds);

return AjaxResult.success("重置成功");

}

return AjaxResult.fail("重置失败,没有选择用户");

}

**6.2.4角色管理**

1. 模块描述

角色管理模块系统初始只有管理员可以访问该模块。在角色管理模块，管理员可以对角色进行查找、增加、修改、删除这几个基本操作。其中角色管理模块最重要的功能就是分配权限，管理员可以通过这个功能对角色进行授权，被授权的角色会得到相应的功能，比如可以为库存管理人员分配入库审查、库存查询等功能。角色管理模块界面如下：



1. 实现流程

通过IDEA创建角色的实体类和数据层接口，编写业务逻辑接口及其实现类，实现对角色增删改查操作，其中要删除角色时需要先删除用户角色表及角色菜单表中的相关数据，因为这三张表是相互连接的，需要修改mapper层的代码，最后创建controller层。其中实现分配权限的功能，需要实现根据角色ID查询菜单权限ID数据以及保存角色和菜单之关的关系的方法。

1. 主要代码

@Autowired

private RoleService roleService;

@GetMapping("listRoleForPage")

public AjaxResult listRoleForPage(RoleDto roleDto) {

DataGridView gridView = this.roleService.listRolePage(roleDto);

return AjaxResult.success("查询成功", gridView.getData(), gridView.getTotal());

}

@GetMapping("selectAllRole")

public AjaxResult selectAllRole() {

List<Role> lists = this.roleService.listAllRoles();

return AjaxResult.success(lists);

}

@PostMapping("addRole")

@Log(title = "添加角色", businessType = BusinessType.INSERT)

public AjaxResult addRole(@Validated RoleDto roleDto) {

roleDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.roleService.addRole(roleDto));

}

@PutMapping("updateRole")

@Log(title = "修改角色", businessType = BusinessType.UPDATE)

public AjaxResult updateRole(@Validated RoleDto roleDto) {

roleDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.roleService.updateRole(roleDto));

}

@GetMapping("getRoleById/{roleId}")

public AjaxResult getRoleById(@PathVariable @Validated @NotNull(message = "角色ID不能为空") Long roleId) {

return AjaxResult.success(this.roleService.getOne(roleId));

}

@DeleteMapping("deleteRoleByIds/{roleIds}")

@Log(title = "删除角色", businessType = BusinessType.DELETE)

public AjaxResult deleteRoleByIds(@PathVariable @Validated @NotEmpty(message = "要删除的ID不能为空") Long[] roleIds) {

return AjaxResult.toAjax(this.roleService.deleteRoleByIds(roleIds));

}

@PostMapping("saveRoleMenu/{roleId}/{menuIds}")

public AjaxResult saveRoleMenu(@PathVariable Long roleId, @PathVariable Long[] menuIds) {

if (menuIds.length==1 && menuIds[0].equals(-1L)) {

menuIds = new Long[]{};

}

this.roleService.saveRoleMenu(roleId, menuIds);

return AjaxResult.success();

}

@GetMapping("getRoleIdsByUserId/{userId}")

public AjaxResult getRoleIdsByUserId(@PathVariable Long userId){

List<Long> roleIds=this.roleService.getRoleIdsByUserId(userId);

return AjaxResult.success(roleIds);

}

@PostMapping("saveRoleUser/{userId}/{roleIds}")

public AjaxResult saveRoleUser(@PathVariable Long userId,@PathVariable Long[] roleIds){

if(roleIds.length==1&&roleIds[0].equals(-1L)){

roleIds=new Long[]{};

}

this.roleService.saveRoleUser(userId, roleIds);

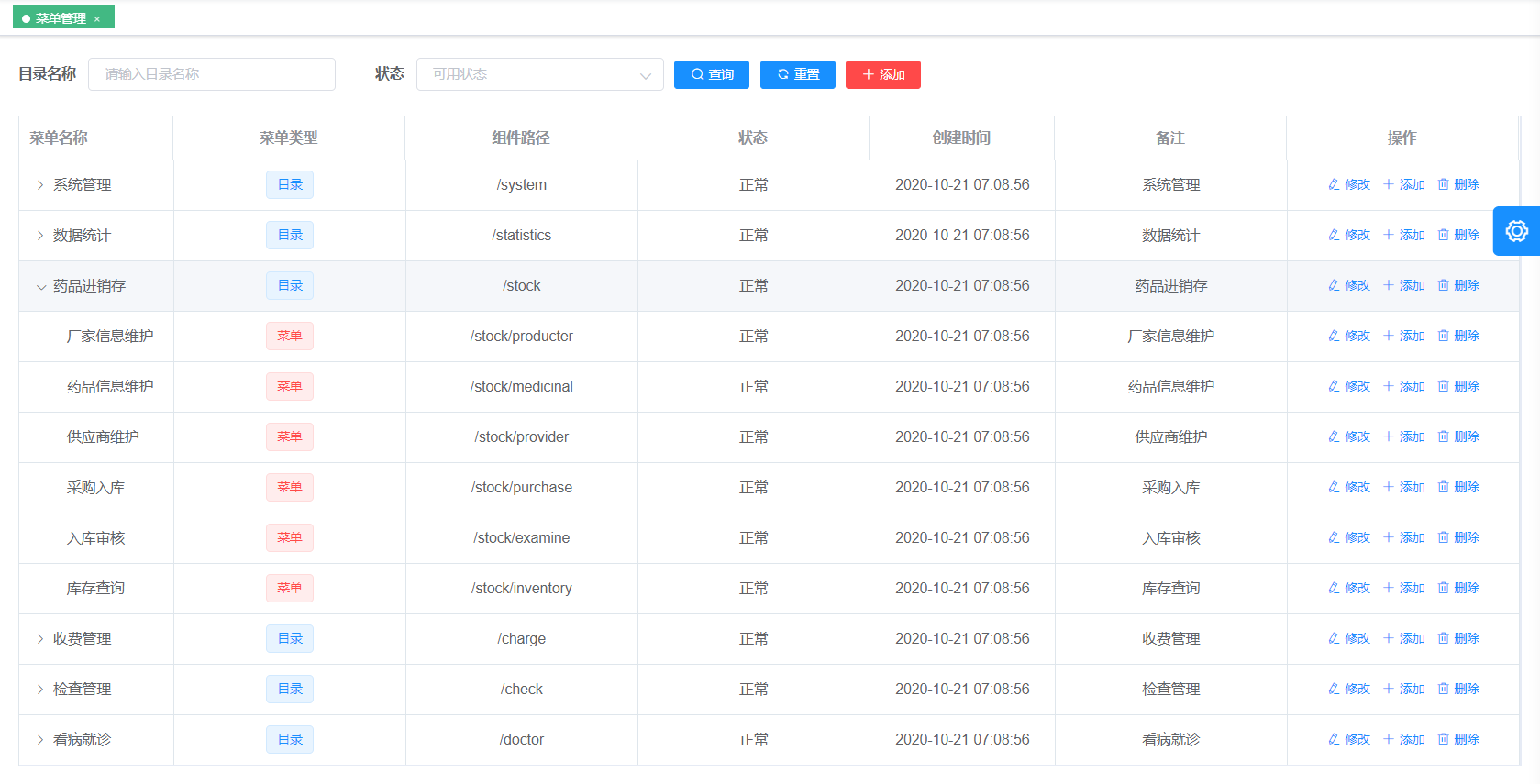
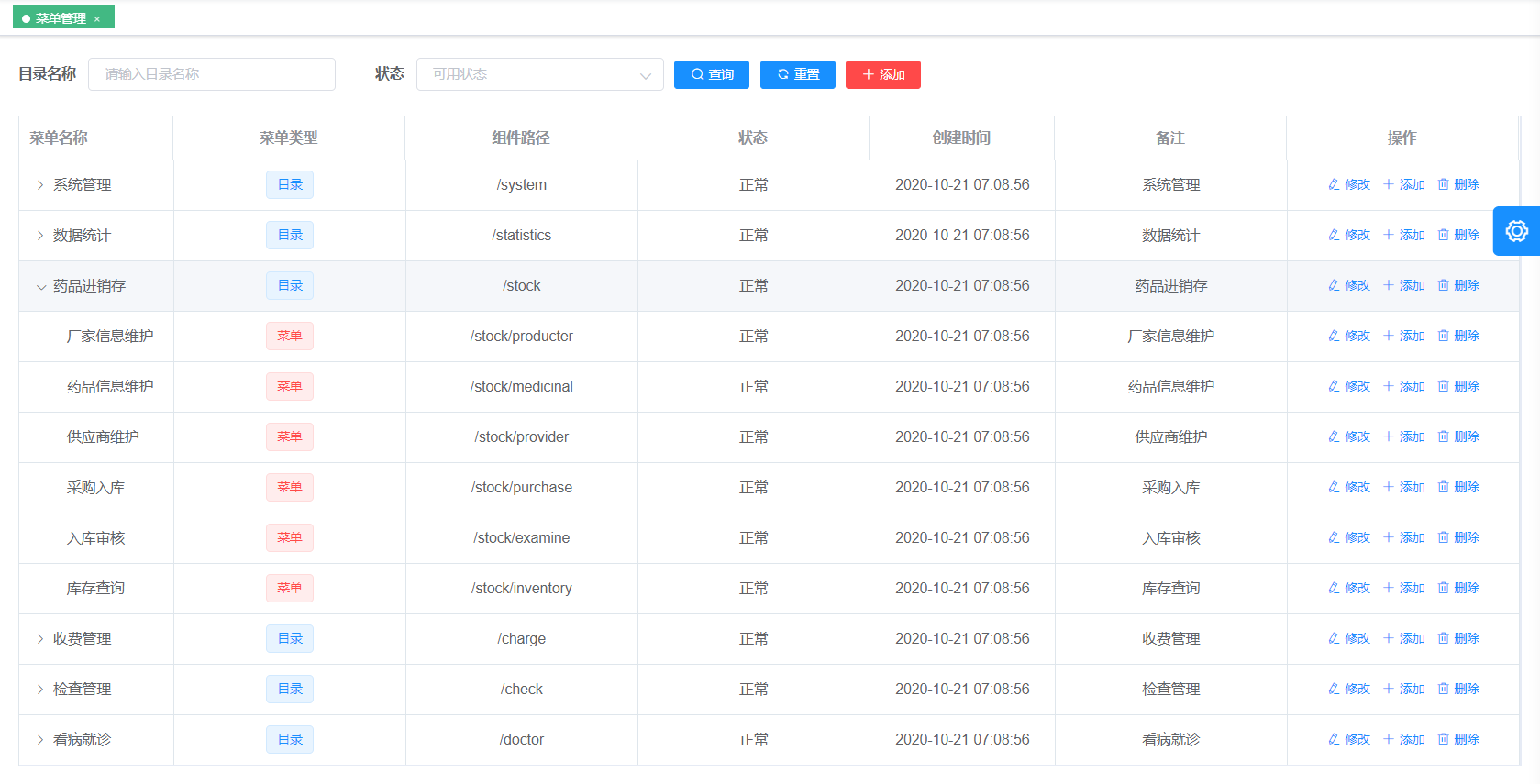
return AjaxResult.success();

}

**6.2.5菜单管理**

1. 模块描述

菜单管理对系统的菜单和目录进行管理，该模块系统初始只有管理员可以访问。在菜单管理模块，管理员可以添加三种不同类型的菜单，分别是菜单、目录、按钮，其中目录的级别最大，目录次之，按钮最小，因此不能在级别低的地方创建级别高的菜单，只能在级别高的菜单上创建同级或级别低的菜单；同样管理员可以对菜单进行查询、修改、删除这些基本操作。其中用户、角色、菜单这三个模块是相互联系的，通过授权角色拥有相关菜单的访问权限，同时角色所对应的用户也可以访问相应菜单，进而获得了某种功能的权力，因为它们的数据库使用RBAC的设计模式进行设计，在菜单和角色之间创建一个角色菜单表，在用户和角色之间创建用户角色表，因此通过角色用户获得相应的功能。菜单管理模块界面如下图所示：



1. 实现流程

首先使用Mybatis-Plus工具生成菜单的实体类和mapper（数据层），其中数据层中自己需要实现根据菜单 ID 查询子节点个数的方法。创建数据传输对象，因为系统的数据大多数以表格形式呈现所有我们创建一个公用的表格数据传输对象；然后创建service层编写字典类型相关的业务逻辑层接口代码，创建service实现类实现接口；最后创建controller层接收所有前台传过来的请求，再把它分配到需要调的service里，相当于前后台的入口。其中删除菜单需要实现根据菜单权限 ID 删除 sys\_role\_menu的方法。

1. 主要代码

@Autowired

private MenuService menuService;

@GetMapping("listAllMenus")

public AjaxResult listAllMenus(MenuDto menuDto){

List<Menu> list=this.menuService.listAllMenus(menuDto);

return AjaxResult.success(list);

}

@GetMapping("selectMenuTree")

public AjaxResult selectMenuTree(){

MenuDto menuDto=new MenuDto();

menuDto.setStatus(Constants.STATUS\_TRUE);

return AjaxResult.success(this.menuService.listAllMenus(menuDto));

}

@PostMapping("addMenu")

@Log(title = "添加菜单",businessType = BusinessType.INSERT)

public AjaxResult addMenu(@Validated MenuDto menuDto){

menuDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.menuService.addMenu(menuDto));

}

@PutMapping("updateMenu")

@Log(title = "修改菜单",businessType = BusinessType.UPDATE)

public AjaxResult updateMenu(@Validated MenuDto menuDto){

menuDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.menuService.updateMenu(menuDto));

}

@GetMapping("getMenuById/{menuId}")

public AjaxResult getMenuById(@PathVariable Long menuId){

Menu menu=this.menuService.getOne(menuId);

return AjaxResult.success(menu);

}

@DeleteMapping("deleteMenuById/{menuId}")

@Log(title = "删除菜单",businessType = BusinessType.DELETE)

public AjaxResult deleteMenuById(@PathVariable Long menuId){

if(this.menuService.hasChildByMenuId(menuId)){

return AjaxResult.fail("当前要删除的菜单有子节点，请先删除子节点");

}

return AjaxResult.toAjax(this.menuService.deleteMenuById(menuId));

}

@GetMapping("getMenuIdsByRoleId/{roleId}")

public AjaxResult getMenuIdsByRoleId(@PathVariable Long roleId){

List<Long> ids = this.menuService.getMenusIdsByRoleId(roleId);

return AjaxResult.success(ids);

}

**6.2.6字典管理**

1. 模块描述

字典管理模块对系统开发所使用的数据字典进行管理，同样本模块系统初始只有管理员可以使用。在字典管理模块中，管理员可以对字典进行crud操作；其中最有特色的功能是缓存同步功能，因为数据字典属于不经常被修改的数据，因此我们将数据字典存放到阿里云的Redis数据库中，因为Redis数据库采用键值对的方式存放数据，这样可以提升访问字典管理模块的速度，这样做也可以提高数据的安全性。通过点击字典类型可以跳转到字典数据的详情信息，可以清楚的知道该字典包含哪些数据。同样在字典数据模块中，管理员可以对字典的数据进行crud操作。字典模块及字典数据模块的界面如下图所示：



1. 实现流程

首先使用Mybatis-Plus工具生成字典类型的实体类和mapper（数据层）。创建数据传输对象，因为系统的数据大多数以表格形式呈现所有我们创建一个公用的表格数据传输对象；然后创建service层编写字典类型相关的业务逻辑层接口代码，创建service实现类实现接口；最后创建controller层接收所有前台传过来的请求，再把它分配到需要调的service里，相当于前后台的入口；其中缓存同步功能需要引入Redis的Maven依赖。字典数据模块的实现方式与字典类型的实现方式类似。

1. 主要代码

@Autowired

private NoticeService noticeService;

@GetMapping("listNoticeForPage")

public AjaxResult listNoticeForPage(NoticeDto noticeDto){

DataGridView gridView = this.noticeService.listNoticePage(noticeDto);

return AjaxResult.success("查询成功",gridView.getData(),gridView.getTotal());

}

@PostMapping("addNotice")

@Log(title = "添加通知公告",businessType = BusinessType.INSERT)

public AjaxResult addNotice(@Validated NoticeDto noticeDto) {

noticeDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.noticeService.addNotice(noticeDto));

}

@PutMapping("updateNotice")

@Log(title = "修改通知公告",businessType = BusinessType.UPDATE)

public AjaxResult updateNotice(@Validated NoticeDto noticeDto) {

noticeDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.noticeService.updateNotice(noticeDto));

}

@GetMapping("getNoticeById/{noticeId}")

public AjaxResult getNoticeById(@PathVariable @Validated @NotNull(message = "通知公告ID不能为空") Long noticeId) {

return AjaxResult.success(this.noticeService.getOne(noticeId));

}

@DeleteMapping("deleteNoticeByIds/{noticeIds}")

@Log(title = "删除通知公告",businessType = BusinessType.DELETE)

public AjaxResult deleteNoticeByIds(@PathVariable @Validated @NotEmpty(message = "要删除的ID不能为空") Long[] noticeIds) {

return AjaxResult.toAjax(this.noticeService.deleteNoticeByIds(noticeIds));

}

**6.2.7通知公告**

1. 模块描述

通知公告模块用于对系统的使用者通知信息，管理员可以通过通知公告模块，发布公共的通告给各个科室，每个系统用户通过通知公共模块都可以知道医院的重要信息。其中管理员可以发放通知和公共两种类型的信息。同时管理员可以对这些通知或公告进行crud的操作，管理员也可以对这些信息进行查看。通过通知公告模块，管理员可以快速传达重要信息，大大的提高了医院的运行效率。通知公告模块界面如下图所示：



1. 实现流程

通知公告模块的实现步骤与前面类似，从实体类开始，到mapper层，再到service层，最后实现controller层。实现完后端就可以是去实现前端代码。

1. 主要代码

@Autowired

private NoticeService noticeService;

@GetMapping("listNoticeForPage")

public AjaxResult listNoticeForPage(NoticeDto noticeDto){

DataGridView gridView = this.noticeService.listNoticePage(noticeDto);

return AjaxResult.success("查询成功",gridView.getData(),gridView.getTotal());

}

@PostMapping("addNotice")

@Log(title = "添加通知公告",businessType = BusinessType.INSERT)

public AjaxResult addNotice(@Validated NoticeDto noticeDto) {

noticeDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.noticeService.addNotice(noticeDto));

}

@PutMapping("updateNotice")

@Log(title = "修改通知公告",businessType = BusinessType.UPDATE)

public AjaxResult updateNotice(@Validated NoticeDto noticeDto) {

noticeDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.noticeService.updateNotice(noticeDto));

}

@GetMapping("getNoticeById/{noticeId}")

public AjaxResult getNoticeById(@PathVariable @Validated @NotNull(message = "通知公告ID不能为空") Long noticeId) {

return AjaxResult.success(this.noticeService.getOne(noticeId));

}

@DeleteMapping("deleteNoticeByIds/{noticeIds}")

@Log(title = "删除通知公告",businessType = BusinessType.DELETE)

public AjaxResult deleteNoticeByIds(@PathVariable @Validated @NotEmpty(message = "要删除的ID不能为空") Long[] noticeIds) {

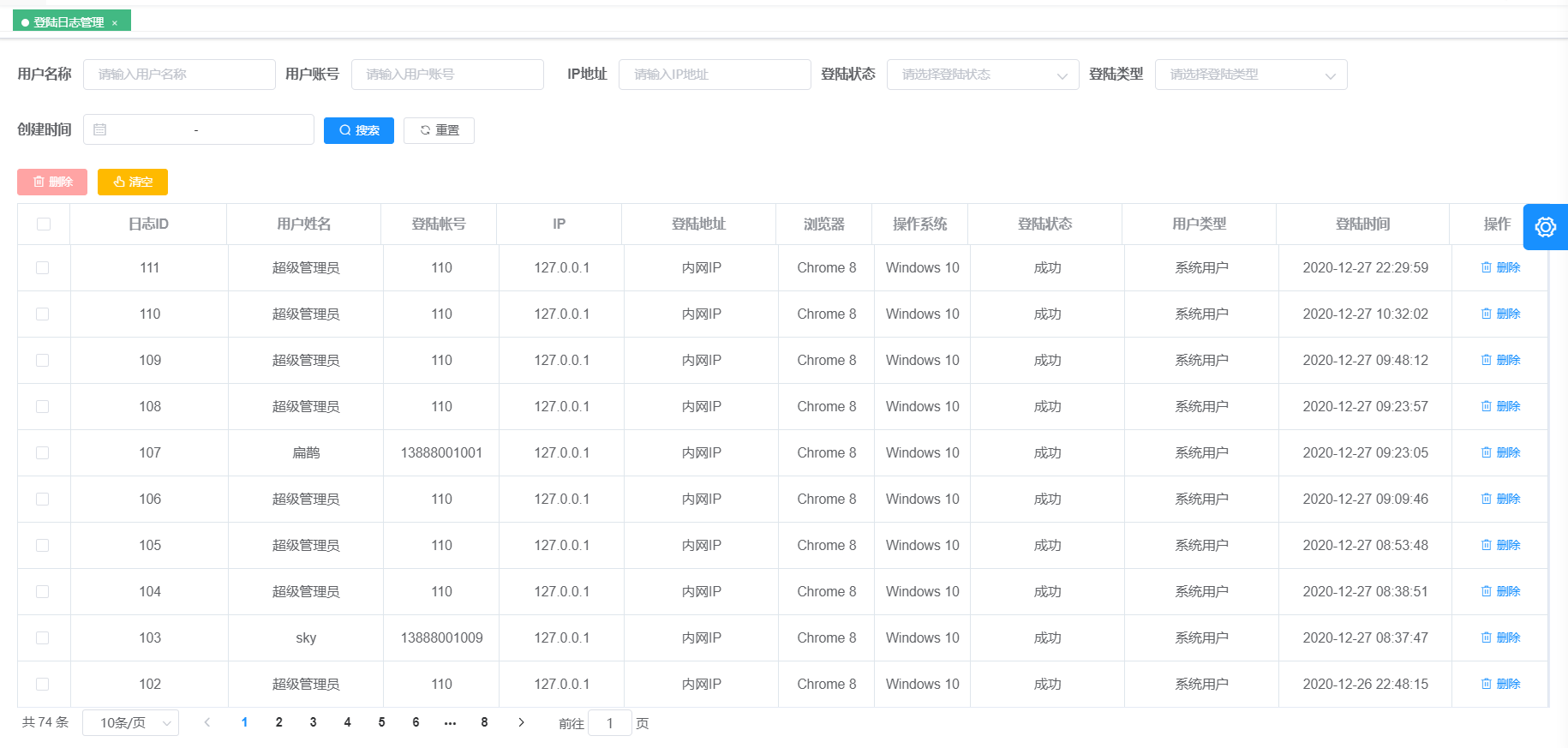
return AjaxResult.toAjax(this.noticeService.deleteNoticeByIds(noticeIds));

}

**6.2.8登录日志管理**

1. 模块描述

登录日志管理模块用于记录所有登录者的信息，通过登录日志管理模块管理员可以知道登录者的时间、IP地址、使用的浏览器等信息。登录日志管理模块与用户登录模块相关联，在用户登录时对其进行日志处理，加强了系统数据的安全性。在该模块，管理员可以进行查找和删除登录日志。登录日志管理模块的界面如下图所示：



1. 实现流程

使用IDEA生成登录日志的实体类、mapper层接口，创建数据传输对象。创建业务逻辑接口及其实现类。创建controller层调用service层的接口，最后修改用户登录的controller层，在用户登录的方法中记录用户的登录信息。

1. 主要代码

@Autowired

private LoginInfoService loginInfoService;

@GetMapping("listForPage")

public AjaxResult listForPage(LoginInfoDto loginInfoDto){

DataGridView dataGridView = loginInfoService.listForPage(loginInfoDto);

return AjaxResult.success("查询成功", dataGridView.getData(), dataGridView.getTotal());

}

@DeleteMapping("deleteLoginInfoByIds/{infoIds}")

public AjaxResult deleteLoginInfoByIds(@PathVariable Long[] infoIds){

return AjaxResult.toAjax(this.loginInfoService.deleteLoginInfoByIds(infoIds));

}

@DeleteMapping("clearLoginInfo")

public AjaxResult clearLoginInfo(){

return AjaxResult.toAjax(this.loginInfoService.clearLoginInfo());

}

**6.2.9操作日志管理**

1. 模块描述

操作日志管理模块与登录日志管理模块类似，操作日志管理模块比登录日志模式更加严谨，该模块对每个有关数据变动的功能进行监控，清晰的记录了用户的操作。该模块清晰的展现了用户在系统中进行的有关数据修改的操作，管理员通过该模块可以知道用户对数据进行操作的时间、操作的方法、操作类型、操作模块等，真正做到了保证数据的可追溯性，做到责任到人。操作日志管理模块界面如下图所示：



1. 实现流程

首先使用Mybatis-Plus工具生成操作日志的实体类和mapper（数据层），创建数据传输对象；然后创建service层编写操作日志相关的业务逻辑层接口代码，创建service实现类实现接口；然后编写与操作切面相关的方法，创建自定义操作日志记录Log注解，创建操作类型、业务操作类型和操作人类别工具类，创建客户端工具类获取servlet的request请求和response 回复，创建OperLogAspect切面类用来记录操作日志，然后修改其他模块的controller层在对有数据变动的操作上添加Log注解记录用户的操作信息，最后创建自身前后台的入口controller层接收所有前台传过来的请求。

1. 主要代码

@Autowired

private OperLogService operLogService;

@GetMapping("listForPage")

public AjaxResult listForPage(OperLogDto operLogDto) {

DataGridView dataGridView = this.operLogService.listForPage(operLogDto);

return AjaxResult.success("查询成功", dataGridView.getData(), dataGridView.getTotal());

}

@DeleteMapping("deleteOperLogByIds/{infoIds}")

public AjaxResult deleteOperLogByIds(@PathVariable Long[] infoIds) {

return AjaxResult.toAjax(this.operLogService.deleteOperLogByIds(infoIds));

}

@DeleteMapping("clearAllOperLog")

public AjaxResult clearAllOperLog() {

return AjaxResult.toAjax(this.operLogService.clearAllOperLog());

}

* 1. **药品进销存模块实现**

其中药品进销存模块的controller层都是在系统管理模块的controller包中实现的，所以系统管理模块的system-web项目需要关联药品进销存模块的erp-api项目，这样system-web才能调用进销存模块中的实体类。

我们使用dubbo对服务进行管理，由于网络延迟等原因有可能会造成访问超时导致非幂等操作的重试问题，我们需要修改业务逻辑实现类中的代码name = "addXXX",retries = 0，代表当前类里面的addXXX这个方法就去只调一次，不进行重试。同时我们还需要解决服务提供者挂掉或者异常的问题，我们修改消费者启动类启用hystrix 的断路保存对消费者进行保护，然后修改controller层的方法中添加@HystrixCommand注解实现熔断处理，创建BaseController实现远程服务不可用，或者出现异常，回调的方法，返回服务器内部异常，请联系管理员的信息。

**6.3.1生产厂家维护**

1. 模块描述

生产厂家模块记录了药品生厂商的信息，该模块主要与药品管理员、库存人员等工作人员有关。在生产厂家模块，工作人员可以获得厂家的名称、负责人、电话等信息，在药品紧缺时，可以通过生产厂家模块快速的联系厂家，及时补充药品。同时相关工作人员可以对生产厂家进行增删改查的操作，这些操作同样会被记录到操作日志中。生产厂家模块界面如下图所示：



1. 实现流程

首先实现生产厂家的实体类，生成mapper层的代码，实现业务逻辑层及其实现类，业务逻辑层主要通过调用mapper层的代码和配合Mybatis-Plus的工具类进行实现，最后实现controller，实现前后端的交互。

1. 主要代码

@Reference

private ProducerService producerService;

@GetMapping("listProducerForPage")

@HystrixCommand

public AjaxResult listProducerForPage(ProducerDto producerDto){

DataGridView gridView = this.producerService.listProducerPage(producerDto);

return AjaxResult.success("查询成功",gridView.getData(),gridView.getTotal());

}

@PostMapping("addProducer")

@HystrixCommand

@Log(title = "添加生产厂家",businessType = BusinessType.INSERT)

public AjaxResult addProducer(@Validated ProducerDto producerDto) {

producerDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.producerService.addProducer(producerDto));

}

@PutMapping("updateProducer")

@HystrixCommand

@Log(title = "修改生产厂家",businessType = BusinessType.UPDATE)

public AjaxResult updateProducer(@Validated ProducerDto producerDto) {

producerDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.producerService.updateProducer(producerDto));

}

@GetMapping("getProducerById/{producerId}")

@HystrixCommand

public AjaxResult getProducerById(@PathVariable @Validated @NotNull(message = "生产厂家ID不能为空") Long producerId) {

return AjaxResult.success(this.producerService.getOne(producerId));

}

@DeleteMapping("deleteProducerByIds/{producerIds}")

@HystrixCommand

@Log(title = "删除生产厂家",businessType = BusinessType.DELETE)

public AjaxResult deleteProducerByIds(@PathVariable @Validated @NotEmpty(message = "要删除的ID不能为空") Long[] producerIds) {

return AjaxResult.toAjax(this.producerService.deleteProducerByIds(producerIds));

}

@GetMapping("selectAllProducer")

@HystrixCommand

public AjaxResult selectAllProducer() {

return AjaxResult.success(this.producerService.selectAllProducer());

}

**6.3.2药品信息维护**

1. 模块描述

药品信息模块保存着医院具有的所有药品信息。通过该模块，医护人员可以对所需药品进行查找，快速了解药品信息。库存人员可以通过该模块，知道药品的库存含量，及时进货防止药品紧缺等问题出现，也可以通过该模块对药品的库存进行修改。同样该模块具备对药品的基本增删改查的操作。药品信息模块界面如下图所示：



1. 实现流程

药品信息维护的实现流程与生产厂家维护的实现流程类似。通过实现业务逻辑接口实现对数据的增删改操作。

1. 主要代码

@Reference

private MedicinesService medicinesService;

@GetMapping("listMedicinesForPage")

@HystrixCommand

public AjaxResult listMedicinesForPage(MedicinesDto medicinesDto){

DataGridView gridView = this.medicinesService.listMedicinesPage(medicinesDto);

return AjaxResult.success("查询成功",gridView.getData(),gridView.getTotal());

}

@PostMapping("addMedicines")

@HystrixCommand

@Log(title = "添加药品",businessType = BusinessType.INSERT)

public AjaxResult addMedicines(@Validated MedicinesDto medicinesDto) {

medicinesDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.medicinesService.addMedicines(medicinesDto));

}

@PutMapping("updateMedicines")

@HystrixCommand

@Log(title = "修改药品",businessType = BusinessType.UPDATE)

public AjaxResult updateMedicines(@Validated MedicinesDto medicinesDto) {

medicinesDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.medicinesService.updateMedicines(medicinesDto));

}

@GetMapping("getMedicinesById/{medicinesId}")

@HystrixCommand

public AjaxResult getMedicinesById(@PathVariable @Validated @NotNull(message = "药品ID不能为空") Long medicinesId) {

return AjaxResult.success(this.medicinesService.getOne(medicinesId));

}

@DeleteMapping("deleteMedicinesByIds/{medicinesIds}")

@HystrixCommand

@Log(title = "删除药品",businessType = BusinessType.DELETE)

public AjaxResult deleteMedicinesByIds(@PathVariable @Validated @NotEmpty(message = "要删除的ID不能为空") Long[] medicinesIds) {

return AjaxResult.toAjax(this.medicinesService.deleteMedicinesByIds(medicinesIds));

}

@GetMapping("selectAllMedicines")

@HystrixCommand

public AjaxResult selectAllMedicines() {

return AjaxResult.success(this.medicinesService.selectAllMedicines());

}

@PostMapping("updateMedicinesStorage/{medicinesId}/{medicinesStockNum}")

@HystrixCommand

@Log(title = "调整药品库存",businessType = BusinessType.UPDATE)

public AjaxResult updateMedicinesStorage(@PathVariable Long medicinesId, @PathVariable Long medicinesStockNum) {

int i = this.medicinesService.updateMedicinesStorage(medicinesId, medicinesStockNum);

return AjaxResult.toAjax(i);

}

**6.3.3供应商维护**

1. 模块描述

供应商模块与生产厂家模块类似，其中供应商模块记录的是医院主要的药品供应商的信息，该模块主要与药品管理员、库存人员等工作人员有关。在供应商模块，工作人员可以获得供应商家的名称、负责人、电话、银行卡号等信息。同时相关工作人员可以对供应商进行增删改查的操作，这些操作同样会被记录到操作日志中。供应商模块界面如下图所示：



1. 实现流程

供应商维护的实现流程与生产厂家维护的实现流程类似。通过实现业务逻辑接口实现对数据的增删改操作。

1. 主要代码

@Reference

private ProviderService providerService;

@GetMapping("listProviderForPage")

@HystrixCommand

public AjaxResult listProviderForPage(ProviderDto providerDto){

DataGridView gridView = this.providerService.listProviderPage(providerDto);

return AjaxResult.success("查询成功",gridView.getData(),gridView.getTotal());

}

@PostMapping("addProvider")

@HystrixCommand

@Log(title = "添加供应商",businessType = BusinessType.INSERT)

public AjaxResult addProvider(@Validated ProviderDto providerDto) {

providerDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.providerService.addProvider(providerDto));

}

@PutMapping("updateProvider")

@HystrixCommand

@Log(title = "修改供应商",businessType = BusinessType.UPDATE)

public AjaxResult updateProvider(@Validated ProviderDto providerDto) {

providerDto.setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.providerService.updateProvider(providerDto));

}

@GetMapping("getProviderById/{providerId}")

@HystrixCommand

public AjaxResult getProviderById(@PathVariable @Validated @NotNull(message = "供应商ID不能为空") Long providerId) {

return AjaxResult.success(this.providerService.getOne(providerId));

}

@DeleteMapping("deleteProviderByIds/{providerIds}")

@HystrixCommand

@Log(title = "删除供应商",businessType = BusinessType.DELETE)

public AjaxResult deleteProviderByIds(@PathVariable @Validated @NotEmpty(message = "要删除的ID不能为空") Long[] providerIds) {

return AjaxResult.toAjax(this.providerService.deleteProviderByIds(providerIds));

}

@GetMapping("selectAllProvider")

@HystrixCommand

public AjaxResult selectAllProvider() {

return AjaxResult.success(this.providerService.selectAllProvider());

}

**6.3.4采购入库列表**

1. 模块描述

采购入库模块是药品进销存模块中最重要的子模块。采购入库模块包含了查询、作废、提交入库、新增采购这四个功能，其中单据状态为审核失败或者未提交状态才能进行作废和提交审核操作，提交入库的使用前提是单据状态为审核通过。进货人员通过新增采购新建采购单据，待审核状态的单据会提交到入库审核模块，等待审核员的审核；若审核不通过，单据状态会变成审核失败并显示审核失败的信息，进货人员可通过审核失败信息对单据进行修改，同时进货人员可以通过点击的单据ID进入单据的详情页面，对单据的信息进行修改，并再次点击提交审核。这一模块需要多位工作人员协同合作。采购入库模块界面如下图所示：



1. 实现流程

通过IDEA生产单据、单据详情、入库详情这三个表的实体类和数据传输对象，生成各自的mapper层接口。通过实现业务逻辑接口，来实现对数据的操作。其中的数据状态转换如模块描述中所述。最后创建controller层进行前后端的交互。

1. 主要代码

@Reference

private PurchaseService purchaseService;

@GetMapping("listPurchaseForPage")

@HystrixCommand

public AjaxResult listPurchaseForPage(PurchaseDto purchaseDto){

DataGridView gridView = this.purchaseService.listPurchasePage(purchaseDto);

return AjaxResult.success("查询成功",gridView.getData(),gridView.getTotal());

}

@GetMapping("listPurchasePendingForPage")

@HystrixCommand

public AjaxResult listPurchasePendingForPage(PurchaseDto purchaseDto){

purchaseDto.setStatus(Constants.STOCK\_PURCHASE\_STATUS\_2);

DataGridView gridView = this.purchaseService.listPurchasePendingForPage(purchaseDto);

return AjaxResult.success("查询成功",gridView.getData(),gridView.getTotal());

}

@PostMapping("doAudit/{purchaseId}")

@HystrixCommand

@Log(title = "采购单管理--提交审核", businessType = BusinessType.UPDATE)

public AjaxResult doAudit(@PathVariable String purchaseId) {

Purchase purchase = this.purchaseService.getPurchaseById(purchaseId);

if(purchase.getStatus().equals(Constants.STOCK\_PURCHASE\_STATUS\_1)||purchase.getStatus().equals(Constants.STOCK\_PURCHASE\_STATUS\_4)) {

int i = this.purchaseService.doAudit(purchaseId, ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(i);

}else {

return AjaxResult.fail("当前单据不是【未提交】或【审核失败】状态，不能提交审核");

}

}

@PostMapping("doInvalid/{purchaseId}")

@HystrixCommand

@Log(title = "采购单管理--作废", businessType = BusinessType.UPDATE)

public AjaxResult doInvalid(@PathVariable String purchaseId) {

Purchase purchase = this.purchaseService.getPurchaseById(purchaseId);

if(purchase.getStatus().equals(Constants.STOCK\_PURCHASE\_STATUS\_1)||purchase.getStatus().equals(Constants.STOCK\_PURCHASE\_STATUS\_4)) {

int i = this.purchaseService.doInvalid(purchaseId);

return AjaxResult.toAjax(i);

}else {

return AjaxResult.fail("当前单据不是【未提交】或【审核失败】状态，不能作废");

}

}

@GetMapping("generatePurchaseId")

public AjaxResult generatePurchaseId() {

return AjaxResult.success(IdGeneratorSnowflake.generatorIdWithPrefix(Constants.ID\_PREFIX\_CG));

}

@PostMapping("addPurchase")

@Log(title = "采购单管理--暂存采购单位和详情数据", businessType = BusinessType.INSERT)

public AjaxResult addPurchase(@RequestBody PurchaseFormDto purchaseFormDto) {

if (checkPurchase(purchaseFormDto)) {

return AjaxResult.fail("当前单据状态不是【未提交】或【审核失败】状态，不能进行修改");

}

purchaseFormDto.getPurchaseDto().setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.purchaseService.addPurchaseAndItem(purchaseFormDto));

}

@PostMapping("addPurchaseToAudit")

@Log(title = "采购单管理--添加并提交审核采购单位和详情数据", businessType = BusinessType.INSERT)

public AjaxResult addPurchaseToAudit(@RequestBody PurchaseFormDto purchaseFormDto) {

if (checkPurchase(purchaseFormDto)) {

return AjaxResult.fail("当前单据状态不是【未提交】或【审核失败】状态，不能进行修改");

}

purchaseFormDto.getPurchaseDto().setSimpleUser(ShiroSecurityUtils.getCurrentSimpleUser());

return AjaxResult.toAjax(this.purchaseService.addPurchaseAndItemToAudit(purchaseFormDto));

}

private boolean checkPurchase(PurchaseFormDto purchaseFormDto) {

String purchaseId = purchaseFormDto.getPurchaseDto().getPurchaseId();

Purchase purchase = this.purchaseService.getPurchaseById(purchaseId);

if(null == purchase) { return false; }

return !purchase.getStatus().equals(Constants.STOCK\_PURCHASE\_STATUS\_1)

&& !purchase.getStatus().equals(Constants.STOCK\_PURCHASE\_STATUS\_4);

}

@GetMapping("queryPurchaseAndItemByPurchaseId/{purchaseId}")

public AjaxResult queryPurchaseAndItemByPurchaseId(@PathVariable String purchaseId) {

Purchase purchase = this.purchaseService.getPurchaseById(purchaseId);

if(null == purchase) {

return AjaxResult.fail("单据号【" + purchaseId + "】不存在");

}

List<PurchaseItem> items = this.purchaseService.getPurchaseItemById(purchaseId);

Map<String, Object> res = new HashMap<>();

res.put("purchase", purchase);

res.put("items", items);

return AjaxResult.success(res);

}

@PostMapping("doInventory/{purchaseId}")

@Log(title = "采购单管理--入库", businessType = BusinessType.UPDATE)

public AjaxResult doInventory(@PathVariable String purchaseId) {

Purchase purchase = this.purchaseService.getPurchaseById(purchaseId);

if(purchase.getStatus().equals(Constants.STOCK\_PURCHASE\_STATUS\_3)) {

return AjaxResult.toAjax(this.purchaseService.doInventory(purchaseId,ShiroSecurityUtils.getCurrentSimpleUser()));

}else if (purchase.getStatus().equals(Constants.STOCK\_PURCHASE\_STATUS\_6)) {

return AjaxResult.fail("采购单【" + purchaseId + "】已入库，不能重复入库");

} else {

return AjaxResult.fail("采购单【" + purchaseId + "】没有审核通过，不能入库");

}

}

**6.3.5入库审核**

1. 模块描述

入库审核模块的主要使用者是与药品审核相关的工作人员，该模块是药品进销存模块中的一个重要的子模块。在这个模块，工作人员通过查看详情功能对采购单据详情进行查看，并对单据进行审核，如果认为单据符合则可以审核通过，如果认为单据存在不合规的地方，可以选择审核不通过然后打回重新修改。入库审核模块界面如下图所示：



1. 实现流程

入库审核主要实现审核通过与审核不通过，通过修改业务逻辑层的代码，实现单据状态的转换。页面中的其他功能主要调用其他模块的业务逻辑接口进行实现。

1. 主要代码

@PostMapping("auditPass/{purchaseId}")

@HystrixCommand

@Log(title = "采购单管理--审核通过", businessType = BusinessType.UPDATE)

public AjaxResult auditPass(@PathVariable String purchaseId) {

Purchase purchase = this.purchaseService.getPurchaseById(purchaseId);

if(purchase.getStatus().equals(Constants.STOCK\_PURCHASE\_STATUS\_2)) {

int i = this.purchaseService.auditPass(purchaseId);

return AjaxResult.toAjax(i);

}else {

return AjaxResult.fail("当前单据不是【待审核】状态，不能进行审核通过");

}

}

@PostMapping("auditNoPass/{purchaseId}/{auditMsg}")

@HystrixCommand

@Log(title = "采购单管理--审核不通过", businessType = BusinessType.UPDATE)

public AjaxResult auditNoPass(@PathVariable String purchaseId, @PathVariable String auditMsg) {

Purchase purchase = this.purchaseService.getPurchaseById(purchaseId);

if(purchase.getStatus().equals(Constants.STOCK\_PURCHASE\_STATUS\_2)) {

int i = this.purchaseService.auditNoPass(purchaseId, auditMsg);

return AjaxResult.toAjax(i);

}else {

return AjaxResult.fail("当前单据不是【待审核】状态，不能进行审核不通过");

}

}

@GetMapping("getPurchaseItemById/{purchaseId}")

@HystrixCommand

public AjaxResult getPurchaseItemById(@PathVariable String purchaseId){

List<PurchaseItem> list = this.purchaseService.getPurchaseItemById(purchaseId);

return AjaxResult.success(list);

}

**6.3.6库存查询**

1. 模块描述

库存查询模块主要的使用者是库存管理人员。在库存查询模块中，库存管理人员可以快速的知道医院中每种药品的总库存量，知道药品的生产厂家，通过这些信息库存管理人员可以快速做出判断是否需要补充药品。同时库存人员可以通过批次库存及价格，知道每次入库的详细信息，增加了信息的正确性，及数据的可追溯性。库存查询模块界面如下图所示：

****

1. 实现流程

通过创建controller层，调用采购入库模块和药品信息模块中相关的逻辑业务接口进行实现。

1. 主要代码

@Reference

private InventoryLogService inventoryLogService;

@GetMapping("listInventoryLogForPage")

@HystrixCommand

public AjaxResult listMedicinesForPage(InventoryLogDto inventoryLogDto){

DataGridView gridView = this.inventoryLogService.listInventoryLogPage(inventoryLogDto);

return AjaxResult.success("查询成功",gridView.getData(),gridView.getTotal());

}

1. **系统测试**

**7.1测试简述**

系统完成后，对系统的测试是非常重要的一环。首先在系统的每个模块完成后对每个模块做的单元测试，另外一个是在系统整体完成了后对整个系统进行的综合测试。

用户登录模块：当用户输入正确的用户名和密码时，能够正常登录并跳转到系统首页面；当用户输入错误或者不存在的用户名或者密码时，系统会提示用户名或者密码不正确；当用户没有输入用户名或者密码或者两者都没有输入的时候，系统会提示用户名、密码不能为空；当用户输入的密码小于4位时，系统会提示密码不能小于4位。

系统管理模块：包含科室、用户、角色、菜单、字典、登录日志、操作日志、通知公告这八个子模块，每一个子模块都需要对模块中包含的属性进行增删改查操作测试。其中用户、角色、菜单这三个子模块包含分配角色、分配权限的操作，通过分配权限对每个角色进行授权，通过分配角色对每个用户进行角色绑定，该模块主要测试通过分配角色、权限的方式是否实现指定用户显示指定的用户界面。对于字典管理模块，还需要测试缓存同步功能，使用缓存同步功能，字典类型及数据会保存在以及部署在阿里云的Redis数据库中，需要测试数据是否完整的保存在Redis数据库中。

药品进销存模块：包含生产厂家、药品信息、供应商、采购入库、入库审核、库存查询这六个子模块；其中生产厂家、药品信息和供应商模块主要测试对数据的增删改查这些基本操作，看是否符合预期结果；而采购入库、入库审核、库存查询涉及到单据状态的变化，提交审核功能只有在单据处于未提交或审核失败状态下才能提交审核；作废功能只有在单据处于未提交或审核失败状态下才能作废单据；审核通过功能只有在待审核状态下才能进行审核通过；审核不通过功能只有在待审核状态下才能进行审核不通过；入库功能只有在审核通过状态下，才能进行入库；库存查询，当输入药品或者批次库存的查询信息时，后台能根据查询信息查出该药品或批次库存的详情信息，并且能正确的返回到用户界面；在测试该模块是还需注意后台是否能够接受到药品信息，并且在提交单据的时候能够顺利把这些信息都持久化到数据库中去，并且在出错的时候能正确返回对应的错误码。

**7.2测试方式**

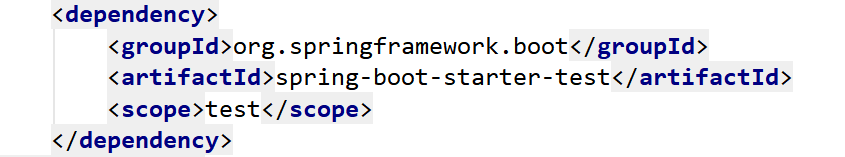
**7.2.1 黑盒测试**

黑盒测试，也称数据驱动测试。该测试方式的工作重点不是考虑程序的内部结构和特性。在测试时，测试者往往将被测试的程序看成一个黑盒，只是在程序接口处通过输入数据、及逆行操作、观察输出结果的方式进行测试，检查是否按照前期制定的需求说明书的规定正常运行，系统能否适当的接受输入数据而产生的输出信息，并且保持外部信息的完整性。本系统一共包含103个接口，采用Swagger + Yapi对系统中的所有接口进行测试，Yapi中的输出数据都是使用json格式进行显示。系统的Yapi接口信息如下图所示：



**7.2.2 单元测试**

单元测试指的是开发者所编写的一小段代码，这段代码用于检验被测代码的一个很小的、很明确的功能是否在结果和逻辑上正确。通常而言，一个单元测试是用于判断某个特定条件下某个特定函数的行为。单元测试是由开发者自己来完成，最终受益的也是程序员自己。可以这么说，程序员有责任编写功能代码，同时也就有责任为自己的代码编写单元测试。执行单元测试，就是为了证明这段代码的行为和我们期望的一致。本系统采用SpringBoot作为后台技术，所以我们采用SpringBoot Test作为测试技术，在开发过程中对系统进行单元测试。SpringBoot Test测试配置如下图所示：



**7.2.2 集成测试**

本次课程设计中在集成测试环境花费了大量时间，前面的单元测试保证了每个小模块功能能够正常的运行，但是到了集成测试，多个模块组合起来，各式各样的错误就暴露出来，不是这出问题就是那出问题，一开始逼得没法子，又重新把项目从头到尾敲了一遍，连续创建了几个一样的项目，去寻找里面出现的错误，到底是否是逻辑错误，终于通过一步步的测试修改，成功解决了问题。

**7.3测试用例**

**7.3.1用户登录模块**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 功能测试 | | | | | | |
| 概述 | | | | | | |
| 测试编号 | | | SYS001 | | | |
| 功能描述 | | | 用户登录 | | | |
| 功能URL | | | http://127.0.0.1/#/login?redirect=%2Fdashboard | | | |
| 用例目的 | | | 测试用户是否能登录成功 | | | |
| 前提条件 | | | 进入用户登录界面 | | | |
| 测试操作 | | | | | | |
| 编号 | 输入/动作 | 期望的输出响应 | | 实际情况 | 是否正确 | 错误编号 |
| 1 | 输入的密码小于4位数 | 系统提示密码不能少于4位 | | 系统提示密码不能少于4位 | 正确 |  |
| 2 | 输入正确的用户名，输入错误的密码 | 系统提示用户名或密码不正确 | | 系统提示用户名或密码不正确 | 正确 |  |
| 3 | 输入错误的用户名，输入正确的密码 | 系统提示用户名或密码不正确 | | 系统提示用户名或密码不正确 | 正确 |  |
| 4 | 不填写用户名及密码 | 系统提示用户名和密码不能为空 | | 系统提示用户名和密码不能为空 | 正确 |  |

**7.3.2角色权限模块**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 功能测试 | | | | | | |
| 概述 | | | | | | |
| 测试编号 | | | SYS002 | | | |
| 功能描述 | | | 分配权限 | | | |
| 功能URL | | | http://127.0.0.1/#/system/role | | | |
| 用例目的 | | | 测试角色是否获得对应权限 | | | |
| 前提条件 | | | 进入角色管理界面 | | | |
| 测试操作 | | | | | | |
| 编号 | 输入/动作 | 期望的输出响应 | | 实际情况 | 是否正确 | 错误编号 |
| 1 | 选择库存管理人员，点击分配权限按钮，分配库存查询权限 | 数据库role\_menu表中存放相应数据 | | 数据库role\_menu表中存放对应数据 | 正确 |  |
| 2 | 选择宣传人员，点击分配权限按钮，分配通知公告权限 | 数据库role\_menu表中存放相应数据 | | 数据库role\_menu表中存放对应数据 | 正确 |  |
| 3 | 选择计划人员，点击分配权限按钮，分配采购入库和库存查询权限 | 数据库role\_menu表中存放相应数据 | | 数据库role\_menu表中存放对应数据 | 正确 |  |
| 4 | 选择采购人员，点击分配权限按钮，分配供应商和生产厂家权限 | 数据库role\_menu表中存放相应数据 | | 数据库role\_menu表中存放对应数据 | 正确 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 功能测试 | | | | | | |
| 概述 | | | | | | |
| 测试编号 | | | SYS003 | | | |
| 功能描述 | | | 分配角色 | | | |
| 功能URL | | | http://127.0.0.1/#/system/user | | | |
| 用例目的 | | | 测试用户权限 | | | |
| 前提条件 | | | 进入用户管理界面 | | | |
| 测试操作 | | | | | | |
| 编号 | 输入/动作 | 期望的输出响应 | | 实际情况 | 是否正确 | 错误编号 |
| 1 | 选择扁鹊，分配采购人员的角色 | 系统只显示生产厂家和供应商界面 | | 系统只显示生产厂家和供应商界面 | 正确 |  |
| 2 | 选择李时珍，分配计划人员的角色 | 系统只显示采购入库和库存查询界面 | | 系统只显示采购入库和库存查询界面 | 正确 |  |
| 3 | 选择孙思邈，分配宣传人员的角色 | 系统只显示通知公告界面 | | 系统只显示通知公告界面 | 正确 |  |
| 4 | 选择华佗，分配库存管理人员的角色 | 系统只显示库存查询界面 | | 系统只显示库存查询界面 | 正确 |  |

**7.3.3药品进销存模块**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 功能测试 | | | | | | |
| 概述 | | | | | | |
| 测试编号 | | | ERP001 | | | |
| 功能描述 | | | 采购入库 | | | |
| 功能URL | | | http://127.0.0.1/#/erp/purchase/newPurchase | | | |
| 用例目的 | | | 测试能否实现采购入库功能 | | | |
| 前提条件 | | | 进入采购入库列表界面 | | | |
| 测试操作 | | | | | | |
| 编号 | 输入/动作 | 期望的输出响应 | | 实际情况 | 是否正确 | 错误编号 |
| 1 | 点击新增采购，添加药物，点击暂存 | 数据库记录数据，单据状态呈未提交 | | 数据库记录数据，单据状态呈未提交 | 正确 |  |
| 2 | 点击新增采购，添加药物，点击提交审核 | 数据库记录数据，单据状态呈未审核 | | 数据库记录数据，单据状态呈未审核 | 正确 |  |
| 3 | 选择未提交状态的单据，点击提交入库 | 系统提示没有审核通过，不能入库 | | 系统提示没有审核通过，不能入库 | 正确 |  |
| 4 | 选择待审核状态的单据，点击作废 | 系统提示当前单据不是未提交或审核失败状态，不能作废 | | 系统提示当前单据不是未提交或审核失败状态，不能作废 | 正确 |  |
| 5 | 选择审核通过状态的单据，点击入库 | 系统提示入库成功，单据状态为入库成功 | | 系统提示入库成功，单据状态为入库成功 | 正确 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 功能测试 | | | | | | |
| 概述 | | | | | | |
| 测试编号 | | | ERP001 | | | |
| 功能描述 | | | 入库审核 | | | |
| 功能URL | | | http://127.0.0.1/#/stock/examine | | | |
| 用例目的 | | | 测试能否实现入库审核功能 | | | |
| 前提条件 | | | 进入入库审核界面 | | | |
| 测试操作 | | | | | | |
| 编号 | 输入/动作 | 期望的输出响应 | | 实际情况 | 是否正确 | 错误编号 |
| 1 | 选择待审核状态的单据，点击审核通过 | 数据库记录数据，系统提示审核通过成功 | | 数据库记录数据，系统提示审核通过成功 | 正确 |  |
| 2 | 选择待审核状态的单据，点击审核不通过 | 数据库记录数据，系统提示审核不通过成功 | | 数据库记录数据，系统提示审核不通过成功 | 正确 |  |
| 3 | 选择未提交状态的单据，点击查看详情 | 系统显示单据的详情信息 | | 系统显示单据的详情信息 | 正确 |  |

**总结**

从开始开发系统到现在，已经过去一个月的时间，系统的开发工作大部分已经完成。本系统采用的是Maven来对项目进行构建管理，Dubbo进行远程服务调用，其他部分使用Spring boot、Mybatis-Plus、Vue等技术来实现的。对用户权限管理、药品采购、入库、库存管理等功能基本上都已经实现。

这次的课程设计运用了许多技术，涵盖了许多的知识点。在还没开始做这个项目前，我们只对这些技术有一个大致的了解，并不懂得这些技术聚合在一起，能做些什么。所以一开始选择开发一个分布式系统是一个大胆的尝试，对这些技术也不是很了解。但是在确定做分布式系统之后，我们首先对这些技术进行了一次全面的学习，通过去图书馆借一些关于分布式、Spring boot等技术相关的书籍，边开发边学习。在系统开发初期由于对技术的掌握程度不高，在整合各个技术的时候，出现了很多的错误，这些错误在一开始没有显示出来，直到后面这些问题才一一暴露出来，最终我们以大量的时间作为代价解决了这些问题，当然因为这些问题我们对技术的理解更进一层。在开发系统的过程中，我学到了很多，对这些技术的使用也更加熟练，也懂得了许多自己之前不会的，对我来说这次的课程设计是对我的一次很好的磨炼。从确定题目，到系统的设计，再到系统的实现，最后系统的完成，这一整个过程让我受益颇多。

完成整个系统也是对我们大三上这一学年所学的知识的一个综合运用，之前对于做一个网站一个系统并没有太多的概念，但是通过这次的课程设计，我们对如何开发一个网站一个系统有了更深的理解。通过本次的课程设计也让我们小组从中学到了很多。但是这个系统还有很多可以完善的地方，比如说可以添加一个小型的医患系统，还可以对系统中的内容进行完善，这些都是以后可以改进的地方。

**参考文献**

1. 王霓虹. 数据库系统设计与实践[M].哈尔滨：哈尔滨工业大学出版社，2012.
2. 杨弘平，吕海华，李波，史江萍. UML基础、建模与设计实战[M].北京：清华大学出版社，2012.

[3] 余金山. 实时UML与RationalRoseRealTime建模案例剖析[M].北京：电子工业出版社，2007.

[4] 申思维. Vue.js快速入门[M].北京：清华大学出版社，2019.

[5] 罗爱萍.仓库进销存管理系统的开发[D].电子科技大学，2006.

[6] 陶亮. 进销存管理系统的设计与实现[D].电子科技大学，2010.

[7] 小峰. 数据库系统导论[D].机械工业出版社，2000

[8] 施瓦茨，扎伊采夫，特卡琴科. 高性能Mysql[M].北京：电子工业出版社，2013.

[9] 范开勇,陈宇收.MySQL数据库性能优化研究[J].中国新通信,2019,21(01)：01-57.

**附录**

1. 系统运行环境：

系统运行是基于：Mysql5.7、SpringBoot2.2.6、Dubbo2.7.5、window10系统、云服务器ECS CentOS7.7

1. 管理员信息：

账号：110 密码：47a42ee6006fdf69e53de74cfe29072d（123456）

1. 用户信息：

账号：13888001001 密码：02f80d33e1e8aad07fa46280b36d0522（001001）

账号：13888001009 密码：83b3d1ecc9e5dfe8548c5480035f8c51（001009）

1. 系统端口：

|  |  |  |
| --- | --- | --- |
| 端口范围 | 授权对象 | 描述 |
| 目的: 3000/3000 | 源: 0.0.0.0/0 | Yapi的端口 |
| 目的: 7000/7010 | 源: 0.0.0.0/0 | Dubbo-Admin的端口 |
| 目的: 2181/2191 | 源: 0.0.0.0/0 | Zookeeper的端口 |
| 目的: 3306/3310 | 源: 0.0.0.0/0 | Mysql的端口 |
| 目的: 8080/8080 | 源: 0.0.0.0/0 | Tomcat默认端口 |
| 目的: 6385/6400 | 源: 0.0.0.0/0 | Redis可用端口 |
| 目的: 80/80 | 源: 0.0.0.0/0 | Nginx默认端口 |