

本科生毕业论文

基于MVC的仓储管理系统的

设计与实现

|  |  |
| --- | --- |
| 姓　　名 | 张广斐 |
| 学　　号 | 20134070224 |
| 专　　业 | 物联网 |
| 指导教师 | 梁雪 讲师 |

2017年5月15日

摘 要

无论是临街商铺，还是连锁购物超市，或是综合性购物中心，在它们的日常运营当中必定存在着货物的进、销、存过程，那么必然会伴随着资金链的流转过程。通过对郑州几家大型综合性购物中心和一些规模较大的连锁超市的调研，发现这些大中型类的企业希望仓储管理软件不但能够实时跟踪各种产品的进、销、存情况，并且能实现解决仓库管理部门与采购、销售、财务等有业务往来的部门之间的协同办公问题。同时最好具有对使用此系统的大中型企业的之间建立借贷关系的平台。

这个借贷平台的用户都是大中型企业老板。很多大型超市和综合性购物中心往往会销售很多高价值的商品，比如翡翠、和田玉等，因为任何商品的价格都会出现周期性地涨落，所以经常会出现高价值商品因价格变动造成商品滞销积压情况的发生，随之带来的就是资金链可能会发生短缺的问题。从调研结果的实际情况出发，为了解决大中型企业因仓库中产品种类繁多、数量庞大、交易频繁而引发的管理难题，提升仓库管理的效率从而提高企业的效益，不但帮助部分因高价值商品积压等原因造成暂时资金周转困难的企业渡过难关，还能为资金富集的企业提供安全可靠的投资平台，设计了仓储管理系统及配套的借贷平台系统。

论文从对现实的实际需求分析，概念设计，逻辑设计，到系统设计，再到系统实现等多个方面进行详细论述，并且对于在系统开发过程中遇到的瓶颈问题是如何解决的也进行了解释说明。

**关键词：**仓储管理；集群；MVC；负载均衡；数据库防篡改；主从同步；读写分离

**Abstract**

Whether the street shops, or chain supermarket, or comprehensive shopping center in their day-to-day operations, there must be a goods import and sales process, it must be accompanied by the circulation of capital chain. Through the investigation on the larger Zhengzhou several large comprehensive shopping center and the size of some of the supermarket chain, found that these large and medium-sized enterprises hope class warehouse management software can not only real-time tracking of various products, sales and inventory, and can improve warehouse management and procurement, sales and finance a collaborative office between business department. At the same time, it is best to have a platform to establish the relationship between the large and medium sized enterprises.

Users of this lending platform are large and medium-sized business owners. Many large supermarkets and comprehensive shopping centers tend to sell a number of high value goods, such as jade, Hetian jade, because the price of any commodity will appear periodic fluctuation, so often there will be a high value of goods due to price changes caused by the backlog of unmarketable goods, the attendant is the chain of funds may is there a shortage problem. According to the actual situation of the investigation, in order to solve the management problems of large and medium-sized enterprises in the warehouse for a wide range of products, the large number of transactions frequently caused, improve the efficiency of warehouse management so as to improve the efficiency of enterprises, not only help create temporary cash flow difficulties due to the high value of enterprise through the backlog of goods, but also provide safe and reliable investment platform for the design of the enterprise capital accumulation, lending platform warehouse management system and supporting system.

This paper analysis, concept design, logic design from the actual needs of the reality, to many aspects of system design, and system realization in detail, and the bottleneck in the process of system development is how to solve the explained.

**Key Words:** Warehouse management, colony,MVC, load balancing, Database tamper proofing, Master-slave synchronization, Read-write separation

**目　录**

[1 引 言 1](#_Toc13209)

[1.1 系统开发的背景 1](#_Toc6311)

[1.2 系统开发的意义 1](#_Toc13307)

[1.3 仓库管理的现状 2](#_Toc1970)

[1.4 本文的主要工作 2](#_Toc19434)

[1.4.1 研究内容 2](#_Toc28637)

[1.4.2 研究方法 3](#_Toc26145)

[1.4.3 本文结构 3](#_Toc306)

[2 相关技术及工具介绍 4](#_Toc2516)

[2.1 面向对象的Java语言 4](#_Toc10269)

[2.2 开发工具Intellij IDEA 4](#_Toc5028)

[2.3 开发模型 6](#_Toc28950)

[2.4 设计模式 7](#_Toc24350)

[3 系统需求分析 9](#_Toc3747)

[3.1 仓库管理的现状 9](#_Toc11561)

[3.2 系统可行性分析 9](#_Toc15493)

[3.3 系统开发任务 9](#_Toc14110)

[3.4 功能分析 10](#_Toc20731)

[4 系统总体设计 13](#_Toc7001)

[4.1 系统结构设计 13](#_Toc5516)

[4.2 总体功能设计 13](#_Toc30590)

[4.2.1 总功能模块 13](#_Toc2743)

[4.2.2 权限管理 14](#_Toc26063)

[4.2.3 采购订单管理 15](#_Toc5744)

[4.2.4 系统菜单管理 15](#_Toc32320)

[4.2.5 报表管理 16](#_Toc23395)

[4.2.6 满标二审 16](#_Toc16692)

[4.2.7 用户还款 16](#_Toc10051)

[4.2.8 系统催款 16](#_Toc10051)

[4.3 数据库设计 16](#_Toc16640)

[4.3.1 概念结构设计 16](#_Toc6331)

[4.3.2 逻辑结构设计 18](#_Toc18600)

[4.4 类的设计 19](#_Toc25014)

[4.4.1 权限管理 20](#_Toc29049)

[4.4.2 系统菜单管理 21](#_Toc28955)

[4.4.3 满标二审 21](#_Toc26212)

[4.4.4 用户信息查看与修改 21](#_Toc26414)

[4.5 接口设计 22](#_Toc19275)

[5 系统具体实现 24](#_Toc4662)

[5.1 登录 24](#_Toc12190)

[5.2 采购入库管理 25](#_Toc23834)

[5.3 报表管理 28](#_Toc28412)

[5.4 绑定邮箱 29](#_Toc10251)

[5.5 服务器集群及负载均衡 30](#_Toc14253)

[5.6 数据库主从同步及读写分离 31](#_Toc899)

[5.7 系统优化 31](#_Toc899)

[6 系统测试与维护 33](#_Toc28220)

[6.1 系统测试 33](#_Toc18453)

[6.1.1 测试计划 33](#_Toc7790)

[6.1.2 执行测试 33](#_Toc6628)

[6.2 系统维护 34](#_Toc28013)

[6.2.1 影响系统维护的因素 34](#_Toc7362)

[6.2.2 提高系统可维护性 35](#_Toc24038)

[7 总结与展望 36](#_Toc1597)

[参考文献 37](#_Toc32354)

[致 谢 38](#_Toc25895)

1 引 言

改革开发以来，我国经济飞速发展，人们的物质文化需求日益增高，随之应运而生的综合性购物中心、大中型购物连锁超市如雨后春笋般在祖国大地上遍地开花落户，而多年以来一直困扰企业快速发展、影响企业效率关键因素之一就是仓库管理。因为靠人工去管理规模巨大的仓库既费时费力且效率低，并~~且~~不能做到实时、无误，往往会造成决策层不能很好地获取仓库各类商品的进、销、存情况，影响决策层及时准确地决策，从而影响企业的效率，进而阻塞的企业的快速发展。甚至有些企业的决策层因为不能及时获取仓库的库存情况，对某些商品的销售情况产生误判，造成某些高价值的商品脱销、滞销等严重问题，直接影响到企业的利润和声誉。更有些企业因为未能及时获取仓库库存的实时数据，在某些商品已经脱销无库存的情况下与客户签订销售合同，造成不能按时给客户发货导致客户亏损引发客户把企业告上法庭的结局，致使企业不但在经济上承受了巨大受损而且声誉也受到很大影响。目前由于人们消费会受很多外界的因素影响，尤其是现在的大学生、青年朋友，会受所追电视剧、电影的影响在短期内频繁更换潮流目标，从而也会造成早“跟风”的企业受益颇斐，使企业资金大量富集；而有些晚“跟风”的企业则会产品滞销，使企业资金周转困难。针对企业遇到的这些实际困难，设计研发了这套以仓储管理为核心辅助以资金周转的系统。

## 1.1 系统开发的背景

随着脱贫致富攻坚战的打响，国民生活水平日益提高，老百姓对购买商品的种类、品牌等都有了更高的诉求。随之各种大中型连锁超市也从省会城市蔓延到了市级城市和县级城市，甚至连农村都出现了大型规模的购物超市。在现实中，大中型超市脱销、滞销情况严重，高效管理商品种类繁多、数量巨大的仓库难题重重，尤其是综合性购物中心和大型超市尤为明显。无疑实现高效仓储管理成了大中型企业老板们追求的目标之一，而因高价值商品滞销引起的资金周转困难又成了压在老板身上的另一座大山。令人匪夷所思的是，现实中又有另外一些老板苦于资金富集却无有效的安全投资渠道。

计算机在实际生产生活中充当着重要的角色，利用计算机软件实现高效的仓储管理无疑是解决仓储管理难题的最佳选择。设计并实现一款能够实现对企业多部门数据及时汇总，对仓库中所有商品的进、销、存情况实时跟踪，并能生成实时报表的系统是非常必要的。给所有使用同一套仓储管理平台的企业提供一个安全的投资、融资平台，在他们需要扩展业务、增添连锁店、扩大经营范围、或因高价值商品滞销引起资金短期周转困难时可以通过本平台进行融资，资金富集的企业可以通过本平台进行投资获利，实现资金的高效配置。因为本套仓储管理系统是针对大中型企业的综合性仓库设计的一套高效管理系统，所以使用本套仓储管理系统的企业规模都比较大，偿还能力不是问题，其他企业老板投资风险相对要低很多。

仓储管理系统是借助计算机平台，把企业的财务、采购、销售等多个部门实现软连同，降低各个部门之间沟通的时间成本，简化了各个部门之间协同办公的流程，实现了多部门间数据同步的及时性，提高各个部门之间协同办公的效率。同时也提供了实现资金高效配置的功能。既可以帮助有投资意向的老板解决投资无门的困境，又能帮助一些因资金周转陷入困境的企业解决棘手的资金问题。这款软件完全符合现代化综合性购物中心和购物连锁超市对仓储管理及对资金周转、高效配置的需要。

## 1.2 系统开发的意义

仓储管理在某种意义上在企业管理中占据着重要的地位，如采购入库与销售出库之间，批发与零售之间，多地区多仓库之间的调度等等。有效的仓库管理可以帮助企业加快商品流动的速度，降低成本，保证销售链的顺利进行，实现对资源的有效控制和整合。

资金链的稳定健壮流转，资金高效配置又是企业实现盈利最大化必不可少的基础条件。在调查中就发现有不少大中型连锁超市就曾经出现过高价值商品滞销引起的资金周转困难，进而影响整个销售链的效益。也有不少个体经营的规模较大的超市因为经营得当，决策英明使自己的可支配资金非常充沛，即使在满足自己扩展经营规模的前提下，账户上依然有不少待支配资金，然而这些钱存银行利息太低，去进行又怕风险太大，害怕投资不当使自己多年的心血功亏一篑。

这款软件的使用是基于pc端的，操作规范简单易懂。使用这款软件对综合性仓库进行管理，能够实现对所有商品采购、销售、库存数据的实时查询，数据清新、准确、及时，还能根据需要生成线形图、柱形图、饼状图等等类型的报表，降低决策层的决策难度。仓库清点以及与财务人员的对账工作量大大减少，降低了多部门之间协同工作的门槛，提高了企业的整体运作效率。在本系统中加入了入库审核、出库审核等功能，大大降低了审核流程的复杂度还极大地提高了审核的效率，并且因为用于保存审核数据的数据库表是做了数据防篡改功能的，因此每单数据的审核都能准确找到审核人，能够达到在无形中提高审核人员的工作态度。使用本套仓储管理系统的企业还可以借助配套提供的的借贷模块来高效配置自己的资金，既可以利用富集的资金投资高效盈利，又可以解决资金周转困难的燃眉之急。借贷模块和仓库管理系统模块是分离的，当企业购买了仓库管理系统后，仓库管理模块是部署在企业的服务器上的。而资金高效配置模块是由产品研发中心来管理的，仅仅给使用该仓库管理系统的企业提供访问地址。这样做的目的一是为提高子系统的安全性，二是只有真正的大中型企业才能使用配套的子系统，可以提高投资安全性，降低投资的风险。因为使用本套系统的企业都是大中型的连锁购物超市，甚至是综合性购物中心，他们可能是要扩大经营规模需要筹款、或者是因为高价值商品因为价格周期性变动积压滞销导致资金链周转出现了暂时性困难需要筹款改善资金链（比如黄金、翡翠、和田玉等等高价值商品因为周期性的价格变动导致商品积压滞销，这些商品的又不易进行打折促销，所以会造成短期的资金周转紧张），他们是有实力还款的，可以让资金充足的老板选择进行投资，极大降低了投资风险。

## 1.3 仓库管理的现状

通过实际考察发现，在企业各部门协同工作的过程中，各个环节对需求量的预测都会存在一定的偏差。随着企业规模的不断扩大，各部门之间需要协同工作量的增加，由于没有一套完善的管理系统，各部门之间在协同工作的时候数据往往是滞后的，偏差也会逐渐增加，导致仓库各类商品的库存量高于或低于实际的需求量。大量的供应偏差，会增加企业的保管成本或造成严重脱销，给企业造成严重损失。因此，在提高服务水平、控制库存、缩短进货销售之间的时间、降低仓库运营成本的多重压力下，高效无误的仓库管理显得越来越重要。从调查中可以知道，综合性购物中心以及大中型连锁购物超市的发展计划中，都将降低仓库的运营成本，提高仓库管理的实时性、准确性做为重要一项。

2015年在十二届全国人大三次会议上李克强总理也提出互联网+的概念，将互联网融入到各行各业成为一个新的市场。截至目前，计算机在各行各业都大显身手，无论是新兴制造业，还是传统行业，都加上了计算机网络的翅膀。很多软件公司都开始为各行各业量身打造合适的软件系统，仓库管理系统就是其中的重要一项。但很多公司在设计的时候过于强调“量身打造”了，使仓库管理系统仅仅就只服务于仓库管理部门，没有考虑把仓库管理部门与其他有业务相关的部门通过软件实现协同办公，比如可以与采购部门、审核部门、销售部门、财务部门对接，降低仓库管理的审核流程，方便与财务对账汇总等等。资金链的顺利流转也是保障实现高效管理仓库系统中非常重要的一环。因为仓库商品的滞销和脱销都直接影响到企业的效益，所以必须在因商品滞销等原因引发资金链流转困难时，及时得到解决。在商品即将脱销时，库存量达到设定的阈值的时候，及时提醒采购部门的相关负责人，以免因商品脱销造成盈利损失。基于这些实际问题出发，设计的这套软件即加强了仓库管理部门与其他有关部门的协同工作，简化了各个部门之间系统办公的流程，提高了多部门协同工作的效率，又实时汇总各个部门的数据，提供给决策层及时准确的仓库数据变化。还提供了资金流转平台供企业使用，给企业的资金链的正常流动提供了保障，从而提高了企业的运营效率。

## 1.4 本文的主要工作

通过对高效仓储管理系统的需求分析，设计并实现代化高效智能的仓储管理系统。

### 1.4.1 研究内容

（1）现代化的仓储管理应该包括仓库日常作业管理，资金链流转管理两大模块。所以此次研究的内容是设计一款方便企业对自己的仓库进行实时管理、搞活企业资金链的软件系统。尤其对于综合性购物中心、大中型企业超市、连锁店商户而言使用本套软件系统对其仓库进行管理不但方便高效，而且可以解决资金链流动问题，从而实现企业的更高收益。现在很多企业都处于转型期，有很多企业想扩展业务扩大运营范围却苦于资金不足，有些已具规模的企业老板想投资却担心风险太大，如果企业资金富集且短期内不会使用，也可以借助该软件系统提供的配套借贷平台进行投资获益，实现资金高效的配置。如果企业因为高价值商品因周期性价格变动引起的商品滞销积压等原因造成资金周转暂时出现困难的时候也可以借助本借贷平台得到有效解决。

（2）本软件系统主要分为两个主模块：第一个主模块就是对仓库所有商品的进、销、存情况进行实时管理、查询、汇总并根据需求生成各种报表，简化仓库管理部门与企业的其他相关部门协同办公的流程，加强企业的仓储、采购、销售、财务等部门之间实时沟通、数据实时同步的仓储管理模块；第二个主模块主要是给账户资金充裕的企业提供投资平台从而也是帮助因决策失误而造成某种或某些商品积压引起的资金周转困难或想扩展企业现有规模需要资金的企业实现融资的借贷模块。

（3）软件系统的仓储管理模块主要实现了用户管理、角色管理、菜单分配管理（以及在这三个功能的基础上实现的非常细粒度的权限管理）、采购订单管理、入库单管理、出库单管理、部门管理、客户管理、报表管理等等功能。软件系统的借贷模块主要实现了用户风控资料上传、风控人员审核、用户发起借款、发标前审核、满标一审、满标二审、用户提现、用户还款、登录日志查看、借款明细列表、数据库防篡改、敏感字段加密、利用Nginx实现服务器集群的负载均衡等功能。

仓库管理模块的重难点是细粒度的权限拦截，因为采用了非常细粒度的权限划分，每个登陆的用户只能够在页面上看到自己拥有的权限所对应的功能菜单，并进行相关操作，其余不在其权限范围内的功能菜单是看不见的，这样就有效地避免了越权操作和恶意操作，保证了系统的安全行和稳定性；软件系统的核心是可以实时地查询每种商品在各个仓库的采购、销售和库存情况，以及每个仓库中每种商品的进、销和存情况。为了方便商户的使用，软件系统还使用了图形报表的形式汇总库存、销售、订货报表信息，以便商户及时作出进货、退货、打折优惠等决策，并且新生成的入库单和出库单系统都会标注为未审核状态，只有经过审核后的入库单，其入库单明细中的商品才会被录入到现有的库存信息，否则现有的库存信息不会被更新；同理只有经过审核后的出库单，软件系统才会根据出库单的明细中的商品信息更新现有的库存信息，否则现有的库存信息也不会被更新，这样就能够确保入库和出库管理的安全性。

借贷模块的重难点是系统服务器集群的负载均衡，以及数据库服务器的读写分离和主从同步，还有就是数据库防篡改和高并发处理。因为在投标的时候可能会出现抢标的情况，为了避免在高并发投标的时候出现数据覆盖丢失的问题，采用乐观锁的方式来防范。

### 1.4.2 研究方法

设计这款软件的最终目的是简化仓库管理部门与企业其他各有关部门协同办公流程，实时地对综合性仓库的日常作业进行高效管理，提高企业运营效率。所以软件系统也必须操作简单易用，因此采取面向对象的开发语言和开发方法，遵照所见即所得的用户体验原则来设计和开发这款软件系统。

开发语言使用的是面向对象的Java，它的跨平台特性一直以来被开发人员称赞。鉴于Java语言单继承特性、多实现的特点，在Service层使用面向接口的编程思想达到大大降低耦合度的效果。并使用Spring开源框架集中管理类对象及其之间的依赖关系，并将事务也交给Spring统一管理。

为了提高项目开发的效率，增强后期的可扩展性和可维护性，本套系统使用较为成熟的MVC架构模式，将控制层、业务逻辑层、数据持久层分离开来，这样控制层主要负责根据客户的具体请求调用具体相应的业务层，业务层负责具体的业务逻辑操作，数据持久层负责与数据库进行交互。业务层采用面向接口的编程方式降低代码模块之间的耦合度，为后期的可扩展性打下基础，如果日后现有的业务功能需求有变更，只需要更新业务实现层即可。并且在控制层根据业务功能需要依赖注入相应的业务层引用，在业务层中根据数据持久化需要依赖注入相应的数据持久层引用，这样业务层就把控制层和数据持久层联系起来。MVC模式在面向对象系统开发中使用非常广泛，因为这种模式不仅逻辑清晰、开发效率高，而且对后期的系统维护和功能扩展都极有帮助。

为了保证代码质量、降低代码的重复率，提高代码的重用性、提高编写效率，采用简单工厂模式，将多个类的公共部分抽取分离出来，成为一个基类，因为整个系统是有多个子系统模块组成的，借贷平台模块可能会遇到高并发访问的问题，为了解决高并发问题，需要对借贷平台做服务器集群，考虑到负载均衡等问题，又把借贷子系统模块又分为前台模块和后台模块，所以为了进一步提高代码的重用性，使用Maven管理项目，把前后台模块的公共部分抽取出来，放到一个parent项目中。如果某些子系统需要，而另外一些子系统不需要的jar包则放入到<dependencyManagement> </dependencyManagement>中，这样需要使用这些jar包的子系统只需要在其pom.xml文件中引入即可，且不需要再指定jar包的版本号。这样以来，如果多个子项目中都使用的jar包需要更新，只需要在parent里边更改pom.xml文件即可，不用再一个一个去修改子系统的pom.xml文件了。

### 1.4.3 本文结构

本文从以下部分进行设计实现：

第一部分，引言

第二部分，相关技术及工具介绍

第三部分，系统需求分析

第四部分，系统总体设计

第五部分，系统具体实现

第六部分，系统测试与维护

第七部分，总结与展望

2 相关技术及工具介绍

## 2.1 面向对象的Java语言

Java是一门面向对象的编程开发语言，它不仅吸收了C++编程语言的各种优点，还摒弃了C++编程语言里难以理解的多继承、指针等概念，因此Java开发语言具有功能强大和简单易用这两个特性。Java编程语言是静态面向对象编程语言的杰出代表，极妙地实现了面向对象编程理论，它允许程序员以优雅的思维方式进行复杂的编程操作[1]  。随着IT产业的蓬勃发展，许多企业使用的计算机应用程序也在随之转型，企业所使用的计算机模式逐渐向客户端较小、服务器随之增大（甚至得集群部署才能满足访问需求）、数据库容量也在相应增加（甚至数据库服务器也要集群部署才能满足高并发的访问量）的方向转变。此外，加上Java编程语言的独有特点，及其在各项服务器端应用程序的开发优势，企业版的J2EE为企业环境中的计算机模式的应用提供了良好的平台。

## 2.2 开发工具Intellij IDEA

IntelliJ IDEA是JetBrains公司的产品，它是Java语言开发的集成环境，IntelliJ在业界被公认为是最好的Java开发工具之一，尤其在代码自动提示、智能代码助手、J2EE支持、重构、各类版本工具(git、svn、github等)、代码分析、JUnit、CVS整合、创新的GUI设计等方面的功能可以说都是超常的。它的旗舰版本还支持HTML，CSS，PHP，MySQL，Python等。最突出的功能自然是调试（Debug），可以对Java代码、JavaScript、JQuery、Ajax等技术进行调试。其他编辑功能抛开不看，这一点就远胜Eclipse。首先查看Map类型的对象，如果实现类采用的是哈希映射，则会自动过滤空的Entry实例。不像Eclipse，只能在默认的toString()方法中寻找你所要的key。其次，动态Evaluate一个表达式的值，比如得到了一个类的实例，但是并不知晓它的API，可以通过Code Completion点出它所支持的方法，这点Eclipse无法比拟。最后，在多线程调试的情况下，Log on console的功能可以帮你检查多线程执行的情况。

## 2.3 开发模型

MVC三层架构模式是最为常见的开发模式，它能够恰当地处理好数据持久层、视图层和控制器层之间的通讯。Mode--业务层，其主要责任是进行相关的业务处理，然后返回处理后的数据。View--视图层，在本系统中使用了两种数据展示技术，一个是jsp技术另外一个是frameMark技术，其任务是把返回的数据与模板进行动态整合，然后回显到界面上供用户查看。ViewController--控制器层，它是业务层和视图层的中间桥梁，它负责收用户通过页面传递的数据，然后根据用户的具体请求的功能调用相应的业务层，并把数据传递到对应的业务层，等到业务层处理完相应的业务之后，控制器层再把业务层返回的经过处理的数据分发给视图层[9]。MVC三层架构的整体模式示意图如图2-1所示。



**图2-1 MVC模式示意图**

## 2.4 设计模式

所谓的设计模式其实就是一套能够被反复使用的、为众多开发人员知晓的代码设计经验的总结。使用设计模式除了可以提高代码编写的效率、提高代码的可读性、保证代码的可靠行以外，还便于后期技术人员对代码进行维护和修改[10]。设计模式细分的话共23种，通常把设计模式可分为三大类型：

创建型的模式：工厂模式、抽象工厂模式、单例模式、建造者模式、原型模式。

结构型的模式：代理模式、适配器模式、装饰模式、桥接模式、组合模式、外观模式、享元模式[11]。

行为型的模式：迭代器模式、观察者模式、职责连模式、模板方法模式、命令模式、中介者模式、备忘录模式、解释器模式、状态模式、策略模式、访问者模式[12]。

上述这23种设计模式各有千秋，都有各自不同的实际用途和优缺点。除了上述这23 种在技术界家喻户晓的设计模式之外还有一种简单工厂模式，美其名曰：静态方法模式，它其实应当归属创建型模式之列的，但是并没有把它归属于23种GOF设计模式之中。结合本系统特点，因为用到了SpringMVC、Spring两大框架，而这个框架在设计上大量使用了简单工厂设计模式、代理设计模式、单例设计模式，所以在本套系统中会大量使用这几种设计模式。

（1）简单工厂模式

简单工厂模式是工厂模式种较为常用的一种模式，这种设计模式的原理是，设计一个父类，实现公共的功能方法和公共的属性，而子类特有的属性和方法由子类自己去负责定义和实现。这种设计模式在本系统中典型的应用就是在DAO层，把增删改查和高级查询的业务功能放到父接类中，所有需要增删改查功能的类只需要继承这个父类即可。

（2）代理模式

代理模式又分为动态代理设计模式和静态代理设计模式，本套系统中采用的是动态代理设计模式。动态代理设计模式为其他对象提供一种代理以控制对这个对象的访问。代理模式在Java语言中非常常见，JDK提供了一套动态设计模式，CGLIB也是一套著名的动态设计模式，Spring对着两种设计模式都提供了很好的支持。之所以采用动态代理设计模式，是由于某些对象自身存在一些局限性，无法提供某些功能，就可以使用动态设计模式给这些对象增加一些他原本没有的功能，然后由生成的代理对象来代替这个对象。在本套系统中，动态代理的典型应用是在对数据库进行增删改操作之前通过动态代理的方式开启事务，在这些操作结束的时候关闭事务。

（3）单例模式

单例设计模式有两种最常见的设计方式，一种被称为饿汉式，另一种被称为懒汉式。单例模式的显著特点是在系统中一个类只会有一个实例对象，这个类的构造方法会被私有化，即类的构造器被private修饰符修饰，这样就不能通过new的方式创建其实例对象。单例类会提供一个被static修饰符修饰的方法获来取其实例对象，但这个方法返回的永远是同一个实例对象。例如，把用户的用户名和密码设计成一个用户登录类，当用户登录后实例化一个该类对象，从用户登录系统到退出系统这整个过程中凡是需要使用用户登录类对象的地方，使用的都是同一个用户登录类对象。

本套系统来采用这三种设计模式再配合Spring、SpringMVC等框架，在极大地降低了代码的重复率、提高了代码可读性的同时，还为后期的功能维护和业务扩展打下了良好的基石。

3 系统需求分析

## 3.1 仓库管理的现状

伴随着企业生产、销售规模的不断扩大，日常仓库管理中的统计数据不准确、数据更新滞后、仓库管理部门与财务和销售等部门的协同工作的流程复杂等等弊端日益彰显，而且成为影响企业效益的一大障碍。而且现在购买的仓库管理系统存在很多不尽人意的地方，比如权限问题，系统没有很好地划分权限，所有登录系统的用户可以查看系统的所有数据，甚至能够操作整个系统的所有功能，导致最后问题出现后不能准确定位是哪个环节出了错误。在调研过的所有超市和购物中心，他们的负责人都表示愿意出高价买一套稳定、安全、高效、权限功能完善的仓库管理系统。

结合现在超市、购物中心这些仓库货中商品种类繁多、数量巨大，进货、销售极其频繁的特征，从用户提出的具体需求和对现有仓库管理系统不满意之处出发，设计了这套功能齐全、权限更加细粒度划分、提供投资金周转功能的系统。

## 3.2 系统可行性分析

可行性分析就是要在系统实际开发之前，通过从不同的角度研究分析该系统是否具备开发的价值，避免开发一些没有实际使用价值或是不被人们认可的软件出来，浪费时间、精力、人力、物力。还要通过可行性分析尽可能低降低开发成本。通过以下三个角度对本套软件系统进行了可行性分析：

从需求角度来分析，这套仓储管理系统对系统的操作权限进行非常细粒度的划分，用户登录系统后只能看到其权限范围以内的操作菜单，只能访问其权限范围内的操作。而且系统中还与财务、采购和销售部门进行了功能对接，极大简化了仓储部门和财务部门、采购部门、销售部门协同办公的流程，从而提高了整个企业的工作效率。而且系统还提供了完善的在线审核流程，无论是入库单和出库单都能通过在线审核完成，而且还能及时地查询入库单和出库单的审核记录。这样一来不但提供了实时的数据汇总和查询功能，还解决了多部门协同办公过程中的各种问题。更重要的是，使用本套系统的用户有权利在产品提供的配套借贷平台上进行投资和融资，为企业解决了资金高效配置的问题。又因为本套系统是为综合性购物中心和大中型连锁购物超市设计的，所经营规模都是非常大的，这些用户之间融资相对容易，且用户的偿还能力非常强，投资风险相对非常低。所以开发套软件系统，从用户需求的来角度分析是非常可行的。

从开发角度来分析， 仓促管理系统分为两大模块：仓库管理模块和借贷模块。仓库管理模块使用Spring、Strust2、hibernate这三大框架作为基础主干框架，结合jsp技术、easyUI框架进行开发。借贷模块采用Spring、SpringMVC、myBatis这三大框架作为基础主干框架，结合frameMark技术、Bootstrap框架进行开发。所选用的都是世界上比较流行的框架，而且都是开源的技术。这些框架的维护人员众多，供能齐全，性能稳定。为了方便开发、测试、上线部署，使用了Docker技术为系统提供稳定的开发和部署环境。所以，从开发角度分析是完全可行的。

从开发成本角度来分析，本套系统虽然功能众多，但所使用的开发软件、框架都是开源的，项目部署后用户通过电脑的浏览器即可访问使用，不需要额外增加任何付费软件。而且由于使用了较为成熟的流行框架，开发周期必定不长。所以从开发成本的角度上进行分析是可行的。

## 3.3 系统开发任务

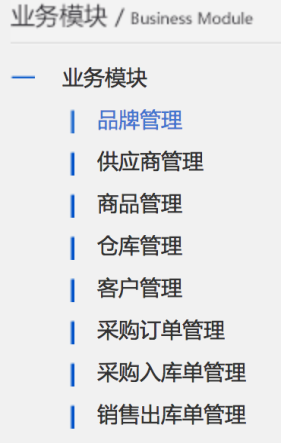
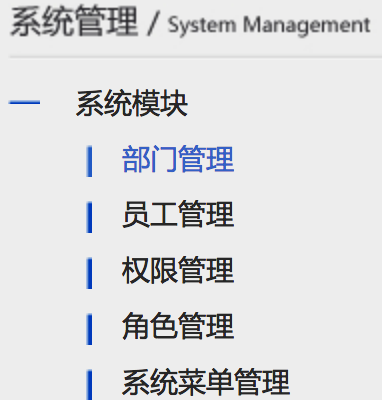
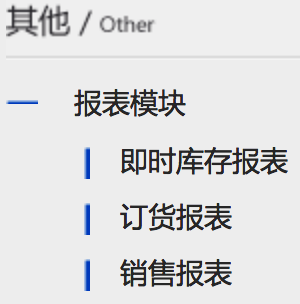
仓储管理系统有两大功能模块，分别是仓库管理模块和借贷平台模块，下面分别介绍这两大功能模块的主要功能。

仓库管理模块的主要功能有：仓库管理模块主要分为业务模块、系统模块、报表模块这三大功能模块，每个功能模块的具体功能见下表4-1。

表 4-1 **仓库管理模块功能表**

|  |  |
| --- | --- |
| 业务模块功能 | 品牌管理、供应商管理、商品管理、仓库管理、客户管理、采购订单管理、采购入库管理、销售出库管理等功能 |
| 系统模块功能 | 部门管理、员工管理、权限管理、角色管理、系统菜单管理等功能 |
| 报表模块功能 | 及时库存报表、订货报表、销售报表 |

仓库管理模块的主要功能模块如下：

**图3-1 仓库管理功能设计图**

借贷平台模块的主要功能有：借贷平台分为客户端和后台服务端，它们的主要功能见下表4-2。

表 4-2 **借贷模块功能表**

|  |  |
| --- | --- |
| 客户端主功能 | 用户实名认证、风控材料上传、用户借款申请、用户投资、用户提现申请、用户还款、账户流水查看、账户登录信息查看、绑定银行卡、绑定手机号、绑定邮箱、查看和修改个人资料等功能。 |
| 后台服务端主要功能 | 对用户实名认证进行审核、对用户上传的风控材料进行审核、发表前审核、放款一审、放款二审、提现审核、催款等功能。 |

仓库管理模块没有设置注册功能，而是在系统启动的时候，系统会去用户数据库~~表~~中检查有没有用户名为admin的用户，如果没有系统会创建一个并保存到用户数据库表中，这个名为admin的用户就是系统的超级管理员。系统中的其他管理员都由超级管理员进行创建并为其分配相应角色的。但借贷平台提供了注册功能，因为即使购买了仓储管理系统的企业可能不想使用提供的配套借贷平台进行资金高效配置，所以当企业购买使用本套系统的时候，需要将借贷平台的客户端链接地址告知企业即可，当企业想要使用的时候直接注册即可，通过审核后就可以在借贷平台上进行融资申请。

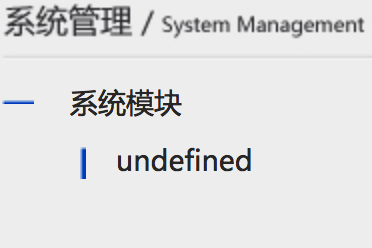
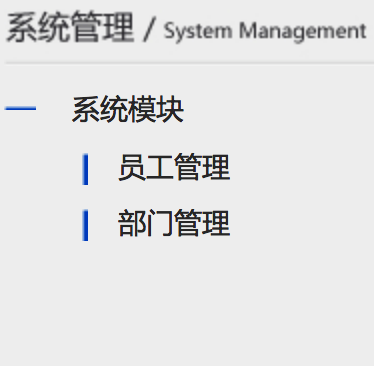
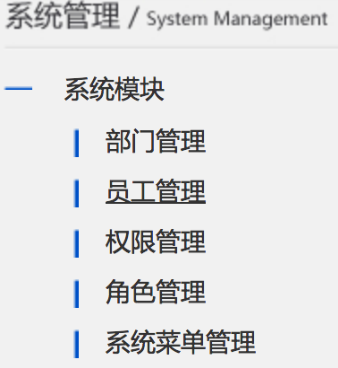
仓库管理模块和借贷平台模块的后端都加入了细粒度的权限拦截功能，在实现权限拦截功能的时候，采用的是给用户分配角色，给角色分配权限的方式。超级管理员拥有所有操作的权限，其中就有员工管理、角色管理和权限管理。超级管理员可以添加员工，并给员工分配或修改角色，给相应的角色分配或修改操作权限。

仓库管理模块和借贷平台的后端都是必须要客户登录后才能进行接下来的访问操作，所以自定义了一个拦截器用于检测用户是否登录，如果没有登录直接跳转到登录页面。用户登录成功后，根据根据用所拥有的权限返回允许他权限范围内可以访问的功能菜单。超级管理员拥有创建用户、给用户分配角色的权限，同时也拥有删除用户以及对用户已有角色修改的权限，超级管理员对系统中已有用户进行管理的设计图如下：



**图3-2 用户权限查看设计图**

如果是用户名为123的用户登录，则什么功能按钮都看不到，无法进行任何操作。如果是用户名为111的用户登录则只能看到部门管理、角色管理等功能按钮。如果是用户名222用户登录则可以看到认识管理、采购管理、仓库管理等功能按钮。如下图依次是用户名为123、222和admin（超级管理员）的用户登录时系统模块的功能按钮设计图如下：

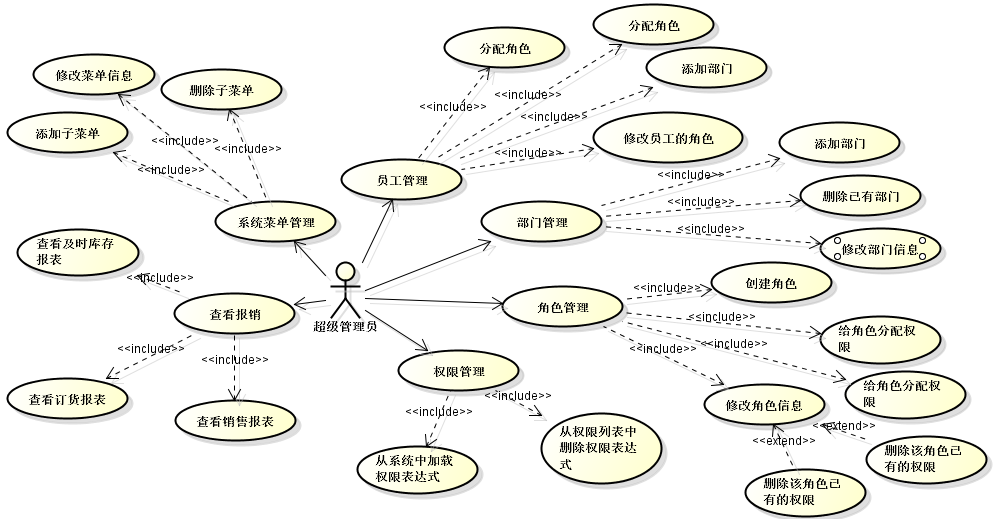
  

**图3-1 123用户登录图 图3-2 222用户登录图 图3-3 admin用户登录图**

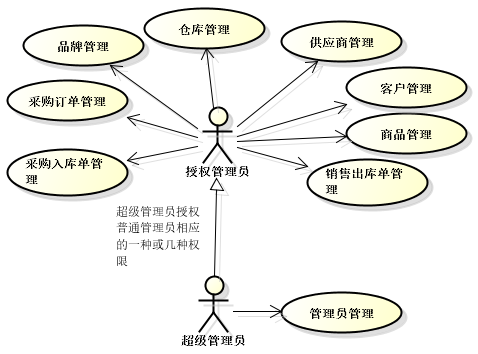
这样用户登录后就只能访问其权限范围内的操作，不在其权限范围内的操作他是无法知道得知的。通过这样细粒度的权限划分，能够保证各个部门在系统办公时责任明确，用户责任具体，避免了有意无意的越权操作引起各种无法解决的问题。这样也就保证了问题出处的可追查性，准确定位责任人。

## 3.4 功能分析

因为采用了细粒度的权限划分，所以本系统中设计的用例有两个参与者，一个是超级管理员，一个是普通管理员。超级管理员拥有所有全部的操作访问权限，而普通管理员是由超级管理创建并分配角色的，普通管理员只拥有被分配角色的相应操作访问权限。所以通过分析得出本系统的用力图如下：



**图3-4 超级管理员用例图**



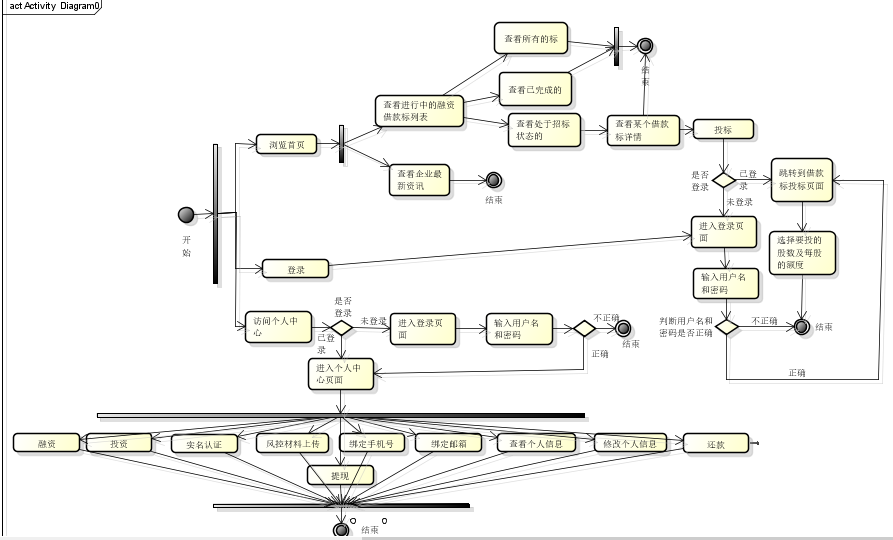
**图3-5 超级管理员和普通管理员之间的关系图**

由于系统是为大中型超市或综合性购物中心设计的仓库管理系统，考虑这些企业库中中的产品种类繁多、数量巨大等需求，因此专门设计可将高级查询结果分页显示的功能，查询功能包含了模糊查询，只要输入关键字即可查询出包含关键字的所有种类商品，不必输入完整的商品名称就能查询的到。同时提供了时间段查询，可以查询某个时间段内某种商品的进、销、存情况。

借贷模块的后端和仓库管理模块都设计了登录拦截功能，即只有当员工登录之后才能进行下一步的操作，可有效避免有人恶意通过地址来输入请求路径并带上参数对数据库中数据数据进行更改。但借贷模块的前端虽然也有登录拦截功能，但与仓库管理模块的登录拦截功能不完全一样，因为借贷模块的前端有些内容即使用户没有登录也是能够看到的，类似于淘宝和京东的首页，即使用户没有登录也能查看，比如在本系统中未登录的用户也可查看当前有哪些融资招标信息、平台首页统计信息等等。但如果用户想要发布融资申请或是想要发起投资等要求，需要用户登录后才能进行的操作。

虽然借贷平台仅仅对购买本套产品的企业开放，但如果用户想要通过借贷平台进行融资，也必须要进行分控材料上传，后台会对客户上传的风控材料进行审核，甚至是牌风控人员到客户那里进行实地考察。出于对用户的投资安全的考虑，即使申请融资招标的人员的风控材料审核通过，用户发标后也不能直接就把这个融资借款标发布到平台上，而是要进行发表前审核，由风控人员再次确认无误之后才能把这个标发布到平台上。等到融资借款标招标满额的时候，发标人也不能直接把钱提现，此时系统会进行满标一审和满标二审。只有满标二审通过后，借款人才能发起提现申请。通过以上安全流程的层层把关，极大地提高了用户投资的安全性。

通过活动图，可以了解系统所具有的功能以及系统对用户使用功能的条件[14]，对于仓库管理模块和借贷模块的后端，用户必须登陆才能进行后续的操作。而借贷模块的客户端，用户在未登录状态下可以浏览借贷平台的首页、查看目前平台发布招标信息。用户登录之后能够访问自己的个人中心、进行实名认证、上传风控材料、绑定手机号、绑定邮箱、绑定银行卡、查看和修改个人信息、查看个人账户流水信息、查看自己的登录信息、进行融资申请、提现申请等等操作。具体的系统活动图如图3-2 所示。



**图3-2 系统活动图**

4 系统总体设计

## 4.1 系统结构设计

本套系统的两大模块都是采用典型的MVC三层框架进行设计的，在持久化层选择了MySQL关系型数据库存储数据，并选用了NoSQL非关系型数据库中著名的Redis作为内存数据库。使用Maven作为项目管理工具。从安全、功能的扩展、项目维护的角度出发，把本套系统的借贷模块是分为客户端和后台服务端的，部署的时候也是客户端和后台服务端分别部署到不同的服务器上，并对客户端建立服务器集群，通过Nginx实现负载均衡，采用Spring+Redis+Session完成session同步的功能；对数据库服务器也进行了集群部署，把数据库集群设计成了主从同步、读写分离模式。在系统中采用了akka解决并发问题、采用kafka解决了消息队列问题、采用storm建立了拓扑结构实现了对业务功能更具体的抽取达到对系统进行了功能拆分的目的。

仓储管理系统是借助计算机对大中型企业的仓库作业进行高效管理，实现仓库管理同其他多个部门完美协同办公，提高企业效率的软件系统。并辅助以提供的借贷平台，给使用本套仓储管理系统的企业提供了安全可靠的融资、投资渠道。根据具体功能需求，在本章节中详细讨论本套系统中主要功能模块的设计思路。从系统层面的功能模块划分，本套系统分为仓库管理模块、借贷平台客户端模块、借贷平台后端模块。其中最主要的功能有权限管理、采购订单管理、采购入库单管理、销售出库单管理、满标二审、用户还款、催款等。在本章节中就针对这七个功能模块以及他们的数据表分别进行设计，通过本章节的设计，就能够看到本套系统的整体轮廓和基本功能。

借款平台因为是客户端和后台服务端分为两个项目进行开发的，所以他们之间必然会存在很多功能相同的地方，为了提高代码的复用性，便于后期的维护和功能扩展，采用面向对象三大特性之一的继承思想，把两个这两个项目中共同的地方抽取出来作为父项目存在，这样这子两个项目都继承自父项目，把子项目的sevice、dao、domain都抽取到父项目中，子项目中仅仅保留各自特有的controller、jsp或frameMark即可。

## 4.2 总体功能设计

### 4.2.1 总功能模块

根据以上对仓储管理系统的需求分析，一个仓储管理系统中最基本也是最重要的功能是权限管理、采购订单管理、系统菜单管理、报表管理、满标二审、用户还款、催款等功能。所以系统总体功能结构图如图4-1所示。



**图 4-1 总体功能结构图**

### 4.2.2 权限管理

权限管理是一套系统安全可靠的基本保证。在实现该功能的时候，因为考虑系统的使用者会有很多，因此需要对用户进行分类。在本系统中将所有的用户分为以下几大类：超级管理员，他拥有操作所有功能的权限；人事管理员，负责员工角色的分配；采购管理员，专项负责商品的采购；仓储管理员，专项负责商品入库和出库管理；行政管理员，专项负责公司的行政事宜。最初的设计构想是：因为不管是什么管理员，他们都是员工，所以给员工类设置一个属性，用以表明该员工在公司的职能。从而在用户进行访问时根据用户的这个属性判断他有没有这个权限。但在实现中发现，因为一个用户可能会同时拥有多个角色，而一个角色又可能同时拥有多个权限，比如一个用户他即是人事管理员，同时又是采购人员，这样一来单在员工类中添加一个属性是无法标示出他到底被分配了多个角色。况且一个用户到底会被分配多少个角色，每个角色又拥有多少种权限都是未知。所以通过在员工类中添加属性来标示用户在公司的职能，以实现当其进行操作时判断他是否拥有权限的功能，是不明智的。经过反复琢磨，又添加了两个类，一个是角色类（Role），一个是权限类（Permission）。并设计了如下关系：一个用户可以拥有多个角色，所以在用户类中添加一个List集合存放分配给该用户的角色，而一个角色有课以分配给多个用户，所以他们在数据库表中的关系是多对多的关系；一个角色可以拥有多个权限，所以在角色类中添加一个List集合用来存放分派给该角色的权限，而一个全下又可以同时分派给多个角色，所以他们在数据库中的关系也是多对多的关系。在后续的开发中又发现，用户访问操作时随机的，登录的用户可能会任意进行访问操作，要想达到理想的权限拦截，就必须给所有的可访问资源加以允许访问的判断，这无疑会出现非常庞大的冗余代码而且效率也极为底下。进过反复尝试，最终通过在模拟Struts2中拦截器的功能，自定义了一个权限拦截器，把权限拦截的代码写到这个拦截器中，并把这个拦截器整合到Struts2的拦截器栈中。每当用户进行访问操作的时候，都必须经过Struts2的拦截器栈的过滤，这样一来就意味着用户的每次访问都必将经过自定义的拦截器的过滤。这样就实现了权限拦截代码一次写，到处用的效果。不但大大减少了冗余的代码，也提高了效率，更有利于程序日后的维护和扩展。权限拦截流程图设计如图4-2-1所示。



**图 4-2-1 权限拦截流程图**

从权限拦截流程图中可以看出，即使有用户知道某个功能的访问地址以、业务功能对应的业务方法名、业务方法需要的参数类型和名称，直接在地址栏中输入相应的地址、业务方法名、相应数据类型的参数进行操作时，如果用户没有相应的权限是根本无法完成访问操作的。比如有人通过抓包等方式获取了超级管理员分配权限的业务访问地址和业务方法名，他直接在地址栏中输入地址、业务方法名、参数(比如http://localhost:8080/employee\_saveOrUpdate.action? employee.id=52&employee.name=fieran&employee.email=1111%40qq.com&employee.age=21&employee.dept.id=6&employee.admin=true&\_\_checkbox\_employee.admin=true&employee.roles.id=5&employee.roles.id=4&employee.roles.id=1&employee.roles.id=2&employee.roles.id=3&employee.roles.id=4&employee.roles.id=5&employee.roles.id=6&\_\_multiselect\_employee.roles.id=)，他也无法进行想要的非法操作，因为要想完成目标业务功能首先是登录检查接着是权限检查，这其中的任何一个不能满足都无法完成操作，系统的权限拦截会将其操作拦截下来。为了使权限系统更加稳定地工作，提供可靠安全的权限拦截，对系统的数据库对敏感字段进行了多次加盐MD5加密，即使拿到加密后的密文，通过撞库获取到的第一次解密敏感字段值也不是真正的原始敏感字段值，因为是多次加盐加密。

### 4.2.3 品牌管理

综合性仓库中产品的品牌类型繁多，所以要想实现高效的仓库管理就必须能够精确定位每一种商品所属的品牌类型。商品所属品牌划归明确就能够方便日后对入库单和出库单的审核。该模块的主要功能包括新增品牌的录入、对系统中已录入商品品牌数据行进删改查操作。

### 4.2.4 供应商管理

大型企业的业务范围广泛，采购人员也会非常的多，不同采购人员对采购同一种商品时选择的供货商也会不同。不同地区仓库中同一类或同一种商品的供货商也会不同。所以采购入库的清单中必须指明本次采购的商品是由哪家供应商供货的。在订货报表查询中可查询出不同供应商提供的产品类型、数量及总金额，该模块的主要功能包括对新供应商信息的录入、对系统中已经录入过的供应商信息进行删改查操作。

### 4.2.5 商品管理

仓库管理系统的功能归根到底就是对仓库中存放的商品进行管理，无论是入库操作还是出库操作或者是对仓库中现有商品统计查询，都是要明确到具体每一种商品的。无论采购入库或销售出库的单笔交易量有多大，都必须明确每一种商品的具体数量。商品管理主要功能包括对录入新的商品信息，对系统中已有的商品信息进行单个删、修改、高级查询以及批量删除操作。

### 4.2.6 仓库管理

很多大型企业会有多个仓库，而且这些仓库可能会分布在不同的地区，在进行即时库存报表查询的时候，是需要明确给出查询条件，是对哪个仓库进行查询的。并且每一笔入库单都要知道这批商品是存放到了哪个仓库中。设计的功能包括对新建设仓库的信息录入、对系统中已有的仓库信息进行删除修改查询操作。

### 4.2.7 客户管理

在销售出库的时候必须要知道本次交易的这批商品是买给谁了，在销售报表中所有的客户可以作为数据字典，用于精确查询出指定客户购买的商品信息。实现的功能包括对客户信息的采集录入，对老客户信息进行查询、修改、删除操作。

### 4.2.8 采购订单管理

采购订单管理页面包括高级查询加分页功能、新增功能和批量删除功能。每一笔订单必须包含业务发生的时间、订单的供应商、录入人、审核人、审核状态等信息。在订单被审核之前录入人是可以对订单内容进行修改的，一旦订单审核通过，则不能再对订单内容进行修改。高级查询包括按照时间段查询、按照供应商查询、按照审核状态查询，并且后面两种查询都是使用的模糊查询，这样用户就可以根据输入的关键字匹配想要查询的订单。

### 4.2.9 采购入库单管理

采购入库单的设计功能同采购订单相似，每一笔入库单必须包含入库单号、业务时间、商品存入的仓库、采购商品的总数量、采购金额、录入人、审核人、审核状态。

### 4.2.10 系统菜单管理

超级管理员拥有对系统菜单维护的权限，系统有三个根菜单分别是系统模块、业务模块、报表模块。其余的系统功能菜单都是这三个根菜单中某一个的子类。菜单是分级管理的，根菜单是第一级菜单，它的parent\_id是null，所以他的上一级菜单为空，URL也为空。其余所有的子菜单的parent\_id是其父菜单的id。当用户登录之后，请求访问某一个父菜单下边的子菜单时，系统就会去数据库中对employee、role、permission、systemmenu这几张表进行多张表联合查询，查询出该登录用户权限范围内可以访问这个父菜单下边的哪些子菜单。

### 4.2.11 报表管理

报表管理是一个企业进行决策的关键，及时准确的报表汇总数据能够帮助决策层作出最佳决策，提高企业的整体效益。为了提高报表的可读性，帮助决策层更好把握决策方向，使用了HighCahrts插件，它可以根据需求生成饼状图、柱状图、折线图等等，并且为了更加细化报表查询功能，根据高级查询的不同条件把报表查询分为及时库存报表、订货报表和销售报表这三大类，这样就能把具备相同高级查询条件的功能分类管理，避免了所有高级查询在一个页面里，不但不易区分还可能会把某一商品没有的属相作为高级查询条件，导致查询失败。而且每一类报表都可以根据页面选择的具体图形针对性以相应的图表回显数据。HighCahrts插件的一个功能就是图形分区明显，鼠标放到任何一个分区上，该分区的详细数据都会回显。

### 4.2.12 满标二审

满标二审是所有审核流程中最复杂的，需要实现的业务逻辑有，当满标二审通过时：修改借款表的状态,设置其处于"还款中"的状态、修改属于该借款标的所有投标对象的状态、去掉借款人的借款状态码、借款人的可用信用额度减少、借款人的待还借款总额增加、支付借款手续费,生成一条支付手续费流水对象、平台收取手续费;平台的账户余额增加,生成一条平台账户流水对象、遍历该借款标的投标对象、减少账户冻结金额,增加一条投标成功的流水、生成针对这个借款标的还款信息、增加待收利息和待收本金；当满标二审被拒绝时：修改标的状态,设置状态为满标审核被拒绝、修改所有投了次标的投标对象的状态、遍历每一个投了次标的投标对象,然后找到投标对象所属的用户,减少该用户的账户上的冻结金额,增加其可用金额。

### 4.2.13 用户还款

用户还款是和满标二审一样发杂的业务操作。具体业务流程为：得到这一期的还款对象、获取借款人(即借款人)在本平台的账户、借款人账户可用余额减少,生成还款流水对象、借款人账户待还金额减少，剩余信用额度增加、得到投资人账户,增加投资人账户的可用余额,生成一条成功收款的流水、减少该投资人的待收本金和待收利息、支付利息管理费,生成支付利息管理费的流水、系统账户收取利息管理费、对于借款标对象而言,如果当前还的是最后一期,则借款标对象变成"完成状态",修改该借款标的所有投标对象的状态。

### 4.2.14 系统催款

系统催款其实就是当借款用户的还款计划已经到期但却未还，系统发一条短信提醒用户。系统要想提醒用户，必须要先知道哪些还款计划即将到期，然后找到这些还款计划所属的借款用户，对这些用户进行短信提现。所以这个过程就必须要进行频繁地查询数据库的操作。为了避免系统频繁访问数据库造成的压力，使用双重定时器的方式解决这个问题。第一个定时器每次查询一个小时内即将到期的用户，把这些用户保存到Redis内存数据库中，然后第二个定时器每秒钟从与Redis交互一次，查询出其中还有已经到期的用户，并发短信提醒用户，他有还款计划已经到期。这样使用两个定时器就有效避免了频繁直接从数据库读取数据而给数据库服务器造成的压力，而Redis是属于内存数据库，读取速度很快，能接受每秒百万次的并发访问，所以把。

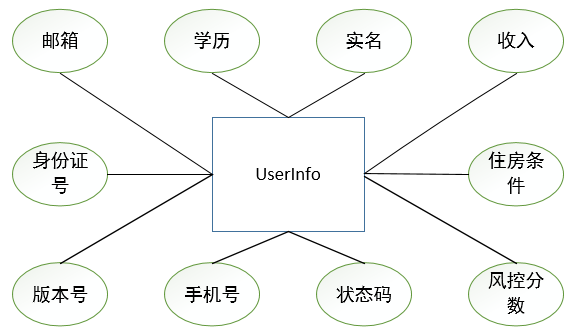
## 4.3 数据库设计

### 4.3.1 概念结构设计

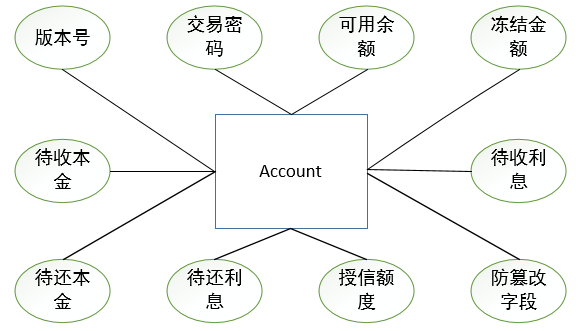
（1）用户信息

系统中字段最多的是借贷平台的用户信息表，为了优化数据库，从用户信息字段的功能用途、是否经常被修改、是否适合放入Redis缓存等角度考虑把借贷平台的用户信息拆分成了用户登录信息表，用户账户信息表，用户基本信息表。用户基本信息表（UserInfo表）存放的是用户实名认证信息、户籍所在地、绑定的邮箱、上传的风控材料等一些不变或不会经常变更的信息，所以用户今本信息表中的这些信息可以放入到Redis缓存中定期同步到数据库中即可，当需要使用的时候直接从Redis中获取，减少对数据库访问，从而降低服务器数据库的查询压力。用户所对应的账户表中的信息可能是随时会变化的，比如用户的可用余额、冻结金额、待收本金、待收利息、待还本息等信息要求必须实时更新，且对脏读零容忍，所以用户账户表中的这些信息不能放在Redis中，必须实时更新到数据库中持久化。用户登陆信息表中（LoginInfo表）中存放的是用户的账号和密码，相对于用户基本信息表和用户账户表而言，用户登录信息表中的数据的特点是用户每次登陆都必须要从数据库中查询出来跟用户输入的账号和密码进行匹配，而如果用户不查看个人信息或不修改个人信息的时候是没有必要把用户的基本信息从数据库中查询出来的，同理如果用户不查看个人账户信息的时候也没必要把用户的账户信息从数据库中查询出来，而且还要把用户登陆对象放入session中，作为检验用户是否登陆使用，所以就把用户的账户和密码单独抽取出来，又因为用户密码的变更情况根据用户个人的习惯而定，不好界定是否是经常变更的字段，所以不能放入到Redis中，只能实时更新到数据库中持久化保存。

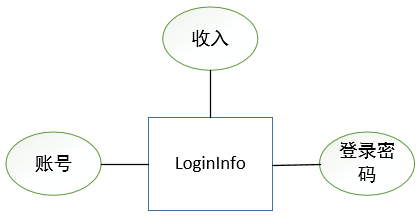
用户信息实体及其属性图如图4-3所示。



**图4-3-1 用户基本信息**



**图4-3-2 用户账户信息**

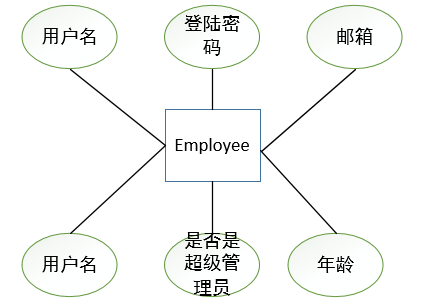


**图4-3-3 用户登陆信息**

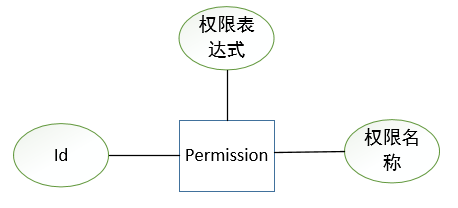
**图4-3 用户信息实体及其属性图**

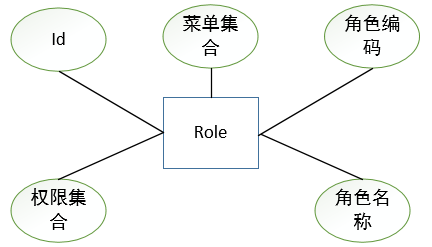
（2） 权限管理信息

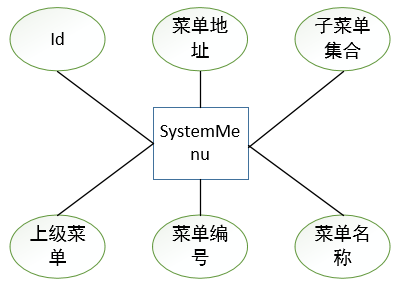
权限管理信息涉及到Employee、Permission、Role、SystemMen这四个实体类。在用户登录成功后，当其要访问某个根菜单或父菜单下的子菜单的时候，首先要获取当前用户所拥有的角色，然后拿到这些角色对应的权限信息，根据这些权限信息判断应该把用户当前要访问的根菜单或父菜单下边的哪些子菜单返回显示到页面上，供用户操作使用。权限管理信息实体及其属性图如图4-4所示。



**图 4-4-1 用户信息实体及其属性图**



**图 4-4-2 权限信息实体及其属性图**

**图 4-4-3 角色信息实体及其属性图**

**图 4-4-4 系统菜单信息实体及其属性图**

**图 4-4 权限管理信息实体及其属性图**

### 4.3.2 逻辑结构设计

（1） 由以上用户信息实体及其属性图可以得出UserInfo信息表如表4-1所示。

表 4-1 **UserInfo信息表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | 说明 |
| Id | bigint(20) | NO | PK | (NULL) | 用户信息ID |
| Version | Int(11) | NO |  | (NULL) | 乐观锁版本号 |
| bitState | varchar(50) | NO |  | (NULL) | 状态码 |
| reanName | varchar(50) | NO |  | (NULL) | 真是姓名 |
| idNumber | varchar(50) | NO |  | (NULL) | 身份证号 |
| phoneNumber | varchar(50) | NO |  | (NULL) | 绑定的电话号码 |
| Email | varchar(50) | NO |  | (NULL) | 绑定的邮箱 |
| authSource | varchar(50) | NO |  | (NULL) | 风控分数 |
| incomeGrade\_id | bigint(30) | NO |  | (NULL) | 收入对象id |
| realAuthid | bigint(30) | NO |  | (NULL) | 实名认证id |
| educationBackgroud\_id | bigint(30) | NO |  | (NULL) | 关联学历对象的id |
| houseCondition\_id | bigint(30) | NO |  | (NULL) | 关联住房条件对象的id |

（2）由以用户信息实体及其属性图可以得到Account信息表如表4-2所示。

表 4-2  **Account信息表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | 说明 |
| Id | bigint(20) | NO | PK | (NULL) | 账户ID |
| tradePassword | varchar(50) | NO |  | (NULL) | 交易密码 |
| usableAmount | decimal | NO |  | 0.0 | 可用余额 |
| freezedAmount | decimal | NO |  | 0.0 | 冻结金额 |
| borrowLimit | decimal | NO |  | 5000.0 | 授信额度 |
| Version | int(11) | NO |  | 0 | 乐观锁版本号 |
| unReceiveInterest | decimal | NO |  | (NULL) | 待收利息 |
| unReceivePrincipal | decimal | NO |  | (NULL) | 待收本金 |
| unReturnAmount | decimal | NO |  | (NULL) | 待还本息 |
| remainBorrowLimit | decimal | NO |  | (NULL) | 剩余授信额度 |

（3） 由以上用户信息实体及其属性图可以得出LoninInfo信息表如表4-3所示。

表 4-3  **LoginInfo信息表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | 说明 |
| Id | bigint(20) | NO | PK | (NULL) | LoinInfo的ID |
| userName | varchar(50) | NO |  | (NULL) | 账号 |
| Password | varchar(50) | NO |  | (NULL) | 登陆密码 |
| State | tinyint | NO |  | 正常状态 | 用户状态 |

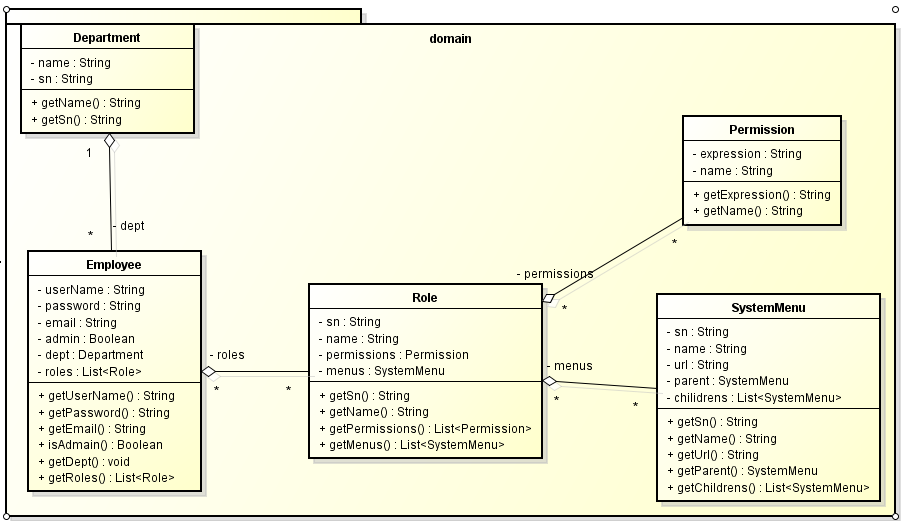
（4）由以上权限管理信息实体及其属性图可以得到SystemMenu用户信息表如表4-4所示。

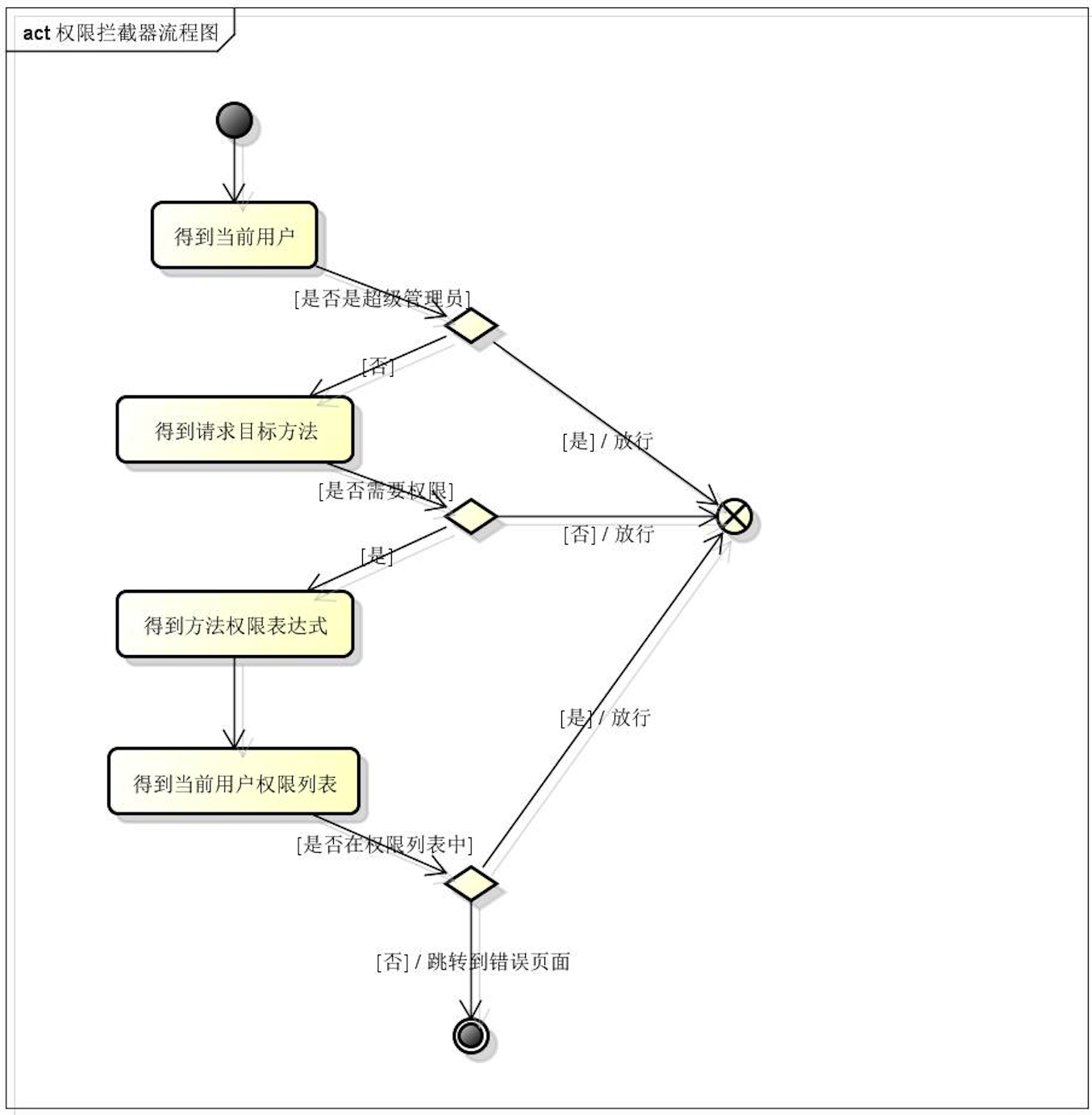
表 4-4 **SystemMenu息表**

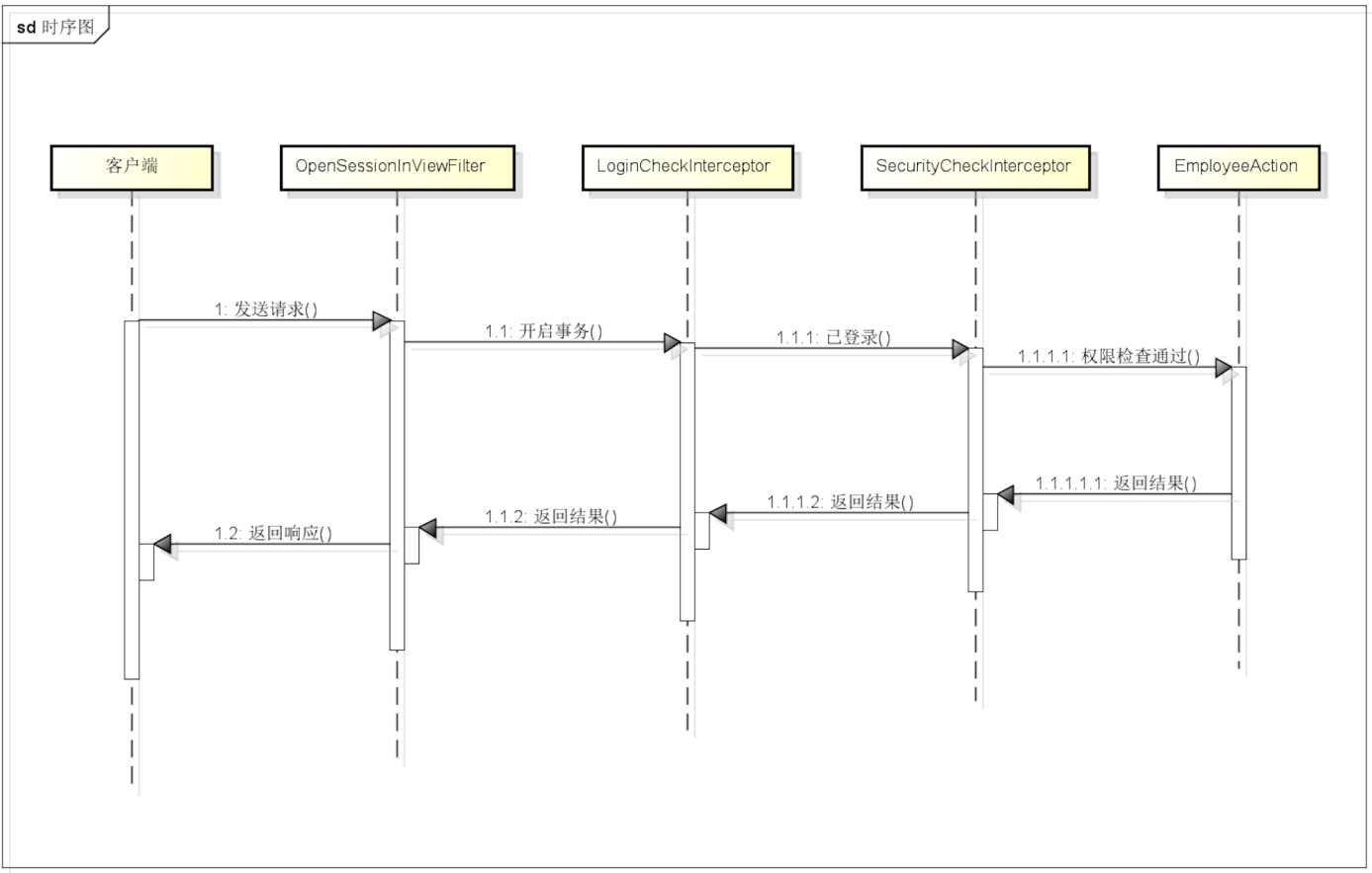
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | 说明 |
| Id | bigint(20) | NO | PK | (NULL) | 菜单ID |
| Name | varchar(50) | NO |  | (NULL) | 菜单名称 |
| url | varchar(50) | NO |  | (NULL) | 菜单的URL |
| Sn | varchar(50) | YES |  | (NULL) | 菜单编码 |
| parent\_id | bigint(20) | YES |  | (NULL) | 父菜单 |

## 4.4 类的设计

系统采用的是MVC三层模式进行设计开发的，结合对系统的功能需求分析，该系统必须满足细粒度的权限管理控制，经过反复尝试最终采用的方式是：给用户分配角色，给角色分配权限的方式实现。具体就是：给系统中每一个菜单下的每一个功能方法的请求路径设定一个权限表达式，权限表达式由功能方法所在类的全限定名加上方法名组成。并设计了一个角色类，角色类里边包含了不同角色对应的权限表达式和这些表达式对应的菜单对象，然后给不同的管理员用户分配不同的角色，这样管理员登录之后在页面上就只能看到自己权限范围内的菜单和这些菜单下的功能按钮。权限拦截系统类图如图4-5所示。根据数据库设计的分析可知，每一个用户的信息分别存放在LoginInfo、UserInfo、Account这三张表中，这三张表之间的关系是UserInfo作为主表，其余两个作为从表，并且采用LoginInfo、Account两表的id既是本表的主键同时又作为本表的外键关联到UserInfo表的主键上的方式来维护三张表之间的关系。用户信息类图如图4-8所示。

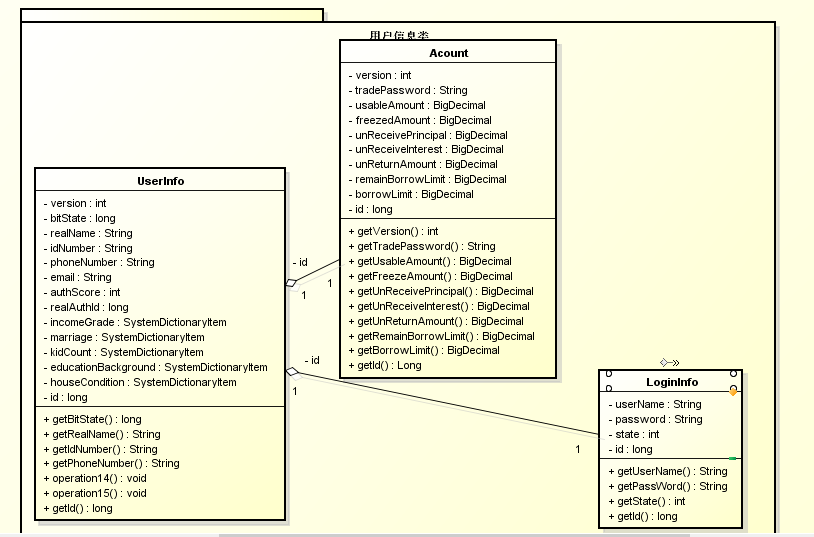


**图 4-5 权限拦截系统类图**

**图 4-6 权限拦截系统流程图**

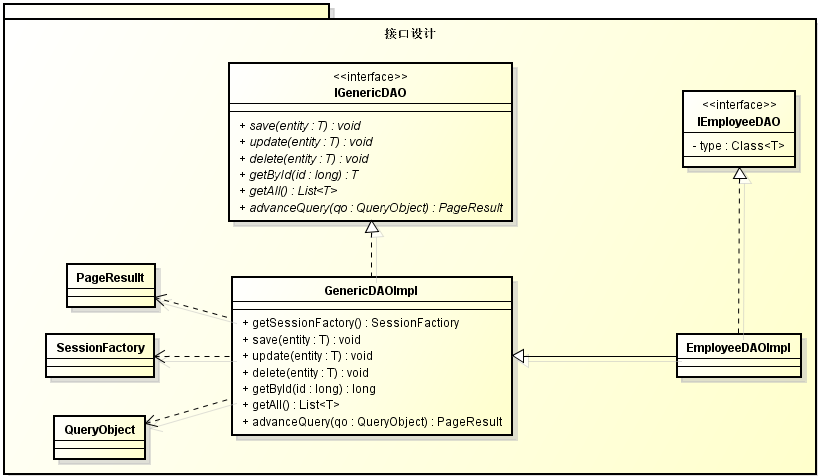
**图 4-7 权限拦截系统时序图**

|  |  |
| --- | --- |
| 权限系统类间关系图对照表 | |
| **Employee** | 员工类，指的是系统中管理员的概念 |
| **Department** | 部门类，指的是系统中部门的概念 |
| **Role** | 角色类，指的是系统中角色的概念，角色是一组权限和一组系统菜单的集合 |
| **SystemMenu** | 系统菜单类，指的是系统中菜单的概念，一个系统的菜单对应一个SystemMenu对象，可以用SystemMenu来控制用户登录系统之后菜单的可见性 |
| **Permission** | 系统权限类，指的是系统中操作权限的概念，一个系统权限对应一个系统操作，可以用Permission来控制用户在系统中的操作 |
| 说明：  1、一个员工属于某一个部门，一个部门可以拥有多个员工，所以员工和部门之间是多对一的关系。  2、一个员工可以被赋予多个角色，一个角色可以分配给多个员工，所以员工和角色之间是多对多的关系。  3、一个角色可以被赋予分配多个系统菜单和系统权限，一个系统菜单和一个系统全下可以分配给多个角色，所以角色和系统菜单、角色和系统权限之间是多对多的关系。  4、系统菜单用来限定某一个用户成功登录系统之后能够查看哪些菜单。  5、系统权限是来限定某一个用户成功登录系统之后可以进行哪些操作。 | |



**图 4-8 用户信息类图**

## 4.5 接口设计

为了提高系统后期的可维护性和易扩展性，本系统采用流行MySQL数据库，使用Java EE进行开发，采取SpringMVC+Spring+ MyBatis平台的B/S架构。数据库设计原则上符合第三范式，且规范、易于维护。程序需使用MVC模式，采用三层架构。一般情况下使用MVC三层架构都会在业务层与控制层之间通过接口化调用实现解耦的功能。在本系统中不但在控制层与业务层之间通过面向接口的编程方式降低了代码的耦合度，经过不断尝试也成功地在业务层与持久化层之间实现了接口化调用并且通过泛型接口抽取了一个通用的高级查询加分页的父接口（IGenericDAO<T>），并提供了该父接口的实现类IGenericDAOImpl<T>。把增、删、改、查以及高价查询加分页的功能都抽到了IGenericDAOImpl类中，这样以来让所有持久化层的接口的实现类都继承IGenericDAOImpl类，每个不同数据类型的持久化类只需要在继承IGenericDAOImpl的时候把相应的数据类型写入到泛型对应的位置传入就行了。通过这种实现方式极大地降低了这两层之间代码的重复率，提高了代码的复用性，不用在每一个持久化类中针对不同数据类型的增、删、改、查去写除了数据类型不同其他完全相同的代码了。持久化层与业务层之间的接口设计关系如图4-9所示。

**图 4-8 接口设计图**

IGenericDAOImpl类的关键代码如下：

|  |
| --- |
| public class IGenericDAOImpl<T> implements IGenericDAO<T> {   protected SessionFactory sessionFactory;   public void setSessionFactory(SessionFactory sessionFactory) {  this.sessionFactory = sessionFactory;  }   private Class<T> Ttype;    public IGenericDAOImpl(){  ParameterizedType parameType = (ParameterizedType)this.getClass().getGenericSuperclass();  Type[] types = parameType.getActualTypeArguments();  Ttype = (Class<T>)types[0];  }    public void save(T entity) {  Session session = sessionFactory.getCurrentSession();  session.save(entity);  }   public void update(T entity) {  Session session = sessionFactory.getCurrentSession();  session.update(entity);  }   public void delete(T entity) {  Session session = sessionFactory.getCurrentSession();  session.delete(entity);  }   public T getById(Long id) {  Session session = sessionFactory.getCurrentSession();  T t = (T) session.get(Ttype, id);  return t;  }   public List<T> getAll() {  Session session = sessionFactory.getCurrentSession();  String hql = "select e from "+Ttype.getSimpleName()+" e";  Query query = session.createQuery(hql);  List<T> list = query.list();  return list;  }    public PageResult advanceQuery(QueryObject qo) {  Session session = sessionFactory.getCurrentSession();  Integer currentPage = qo.getCurrentPage();  Integer pageSize = qo.getPageSize();  Integer totalCount;    //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*查询满足条件的数据的总条数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  String countHQL = "select count(obj) from "+Ttype.getSimpleName()+" obj";// 拼接查询语句  countHQL = countHQL + qo.getQuery();    Query query = session.createQuery(countHQL);  this.setParamsForCondition(query, qo.getParams());  Long count = (Long) query.uniqueResult();  totalCount = count.intValue();  // 如果总页数为零，下边的操作就无需进行了  if (totalCount==0) {  PageResult.*empty*(qo.getPageSize());  }  // 处理页面传递过来的当前页大于总页数的情况  Integer totalPage = totalCount % pageSize == 0 ? totalCount / pageSize : totalCount / pageSize + 1;  if (currentPage > totalPage){  currentPage = totalPage;  qo.setCurrentPage(currentPage);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*按条件分页查询\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  String hql = "select obj from "+Ttype.getSimpleName()+" obj" + qo.getQuery();  query = session.createQuery(hql);  this.setParamsForCondition(query, qo.getParams());    query = query.setFirstResult((currentPage - 1) \* pageSize).setMaxResults(pageSize);  List<?> listDate = query.list();    PageResult pageResult = new PageResult(listDate, totalCount, currentPage, pageSize);    return pageResult;  }   private void setParamsForCondition(Query query,List<?> params){  if (params.size()>0) {  for (int i = 0; i < params.size(); i++) {  query.setParameter(i, params.get(i));  }  }  } } |

## 5 系统具体实现

## 5.1 登录

因为仓储管理模块和借贷模块的后台管理是必须登录才能进行下一步操作的，并对权限拦截系统，在地址栏中数据地址链接到服务器之后会弹出一个登录页面。因为这两个模块都是在项目部署启动完毕的同时自动创建超级管理员，其他管理员由超级管理员创建并分配相应的权限，所以没有注册页面。而借贷模块的客户端是有用户注册页面的。用户在登录页面点击登录按钮后跳出微博授权页面，用户输入微博账号密码进行授权，等待授权成功后登录页面消失，用户继续使用此系统，当微博授权失败时授权页面不消失，用户验证微博账号密码后重新授权登录。登录模块实现效果如图5-1所示。

|  |  |
| --- | --- |
| **(a) 仓库模块登录页面**  **(c) 借贷平台模块注册页面** | **(b) 借贷平台模块登录页面** |

**图 5-1 登录和注册模块实现效果图**

## 5.2 商品管理

## 拥有商品管理权限的管理员可以把新增的商品信息录入到系统中，为后续的报表查询提供字典数据，并未商品管理提供商品所属品牌分类，方便实现仓库准确高效管理的的功能。 图 5-1 商品管理主页面实现效果图 图 5-2 修改商品信息页面实现效果图

## 5.2 供应商管理

## 不同批次或不同仓库中相同商品的供应商可能不会，同一个供应商也会提供多种类型的商品，所以完善的供应商信息对系统中订单管理等功能的实现是非常重要的。系统中录入供应商可以为后续的查看订单报表中提供数据字典。 图 5-3 供应商管理主页面实现效果图 图 5-4 修改供应商信息页面实现效果图

## 5.2 商品管理

## 仓储管理系统的最终目的还是实现对各种商品具体数据信息的实时跟踪，所以系统中要能查询到仓库中的每一种商品的信息，换句话说必须把采购的所有商品都录入到系统中。 图 5-5 供应商管理主页面实现效果图 图 5-6 修改商品信息页面实现效果图

## 5.2 仓库管理

## 企业可能会在不同地区建设很多仓库，这些仓库中的商品种类也各不相同。每一笔采购的商品都应该明确是存入了哪个仓库。在对商品进行统计的时候，也要能知道每个仓库中各种商品的进销存情况。 图 5-7 仓库管理主页面实现效果图 图 5-8 修改仓库信息页面实现效果图

## 5.2 客户管理

## 5.2 采购入库管理

采购入库管理包括采购订单管理和采购入库单管理这两部分功能。采购订单由采购部门的采购人员负责，采购订单的审核也由采购部门的人负责，每一笔采购订单在审核前是可以对其进行修改和删除的，一旦审核通过就只能够查看这笔订单的信息。入库单由仓库管理人员负责，当采购订单审批通过，财务拨款，供应商发货后，仓库管理员根据供应商送来的商品生成相应的入库单。入库单生成之后，由仓库管理员根据审核通过的订单进行核对，商品的名称、品牌、数量都无误之后，就审核通过该入库单，否则审核拒绝。

采购订单主要包含订单编号、业务时间、供应商、订单明细这些信息。订单明细包括商品名称、商品品牌、进货单价、进货数量、每笔订单明细的金额小计、备注这些信息。订单和订单明细是一对多的关系。采购订单功能实现页面如下**图 5-1**到**图 5-5**所示。

**图 5-1 订单管理主页面实现效果图图 5-2 添加订单实现效果图图 5-3 编辑订单实现效果图**

**图 5-4 高级查询实现效果图图 5-5 订单审核实现效果图**

采购入库单信息主要包括入库单编号、业务时间、商品存放的仓库、入库单明细这些信息，采购入库单和入库单明细是一对多的关系。采购入库单明细信息和采购订单明细信息一一对应，在此就不再赘述。页面风格也和采购入库单的页面类似，此处也不再截取功能实现图片。关键代码如下，业务流程都已写好注释：

添加采购订单关键代码：

|  |
| --- |
| public void save(OrderBill orderBill) {  if (orderBill.getItems().size() > 0 ) {// 只有当定单的明细不为空的时候,才允许执行添加操作   // 1、设置该订单的录入人,即当前登录的用户  orderBill.setInputUser(UserContext.*getUserContext*());  // 2、设置订单的录入时间  orderBill.setInputTime(new Date());  // 设置订单的详细信息  this.resolveItems(orderBill);   // 3、保存对象  orderBillDAO.save(orderBill);  }else {  throw new RuntimeException("明细不能为空");  } }  private void resolveItems(OrderBill orderBill){  // 1、新增加的订单,在保存之前,要设置其审核状态为未审核  orderBill.setStatus(OrderBill.*NORMAL*);   // 2、初始化该订单的总数量和总价格  orderBill.setTotalNumber(BigDecimal.*ZERO*);  orderBill.setTotalCostPrice(BigDecimal.*ZERO*);   List<OrderBillItem> items = orderBill.getItems();  for (OrderBillItem item : items) {  // 3、设置many方和one放的关系,因为在多对一的关系中,我们一般都是让many放维护关系的  item.setOrderBill(orderBill);  // 4、计算该订单中每一条明细的总价格  BigDecimal number = item.getNumber();// 计算当前明细中商品的个数  BigDecimal costPrice = item.getCostPrice();// 计算当前明细中商品的单价  BigDecimal totalPricet = number.multiply(costPrice);// 计算当前明细的总价  totalPricet.setScale(2, RoundingMode.*HALF\_UP*);  item.setAmount(totalPricet);  // 5、计算订单的总数量和总价格  BigDecimal currentTotalNumber = orderBill.getTotalNumber();  BigDecimal currentTotalCostPrice = orderBill.getTotalCostPrice();  orderBill.setTotalNumber(currentTotalNumber.add(item.getNumber()));  orderBill.setTotalCostPrice(currentTotalCostPrice.add(item.getAmount()));  } } |

采购订单审核代码：

|  |
| --- |
| */\*\*  \* 采购订单审核  \** ***@param*** *orderBill  \*/* public void audit(OrderBill orderBill) {  if (orderBill.getStatus() == OrderBill.*NORMAL*){// 只有定单没审核的时候才允许审核操作  orderBill.setStatus(OrderBill.*AUTDIT*);  orderBill.setAuditTime(new Date());  orderBill.setAuditor(UserContext.*getUserContext*());  orderBillDAO.update(orderBill);  }else {  throw new RuntimeException("该入库单已被审核过,请勿重复审核!!!");  } } |

添加入库单关键代码：

|  |
| --- |
| public void save(StockIncomeBill stockIncomeBill) {  if (stockIncomeBill.getItems().size() > 0 ) {// 只有当入库单的明细不为空的时候,才允许执行添加操作  // 设置录入人  stockIncomeBill.setInputUser(UserContext.*getUserContext*());  // 设置录入时间  stockIncomeBill.setInputTime(new Date());  // 设置入库单的详细信息  this.resolveItems(stockIncomeBill);   stockIncomeBillDAO.save(stockIncomeBill);  }else {  throw new RuntimeException("入库单明细不能为空!!!");  } }  private void resolveItems(StockIncomeBill stockIncomeBill) {  // 1、新增加的订单,在保存之前,要设置其审核状态为未审核  stockIncomeBill.setStatus(StockIncomeBill.*NORMAL*);  // 2、初始化该订单的总数量和总价格  stockIncomeBill.setTotalNumber(BigDecimal.*ZERO*);  stockIncomeBill.setTotalCostPrice(BigDecimal.*ZERO*);   List<StockIncomeBillItem> items = stockIncomeBill.getItems();  for (StockIncomeBillItem item : items) {  // 3、设置many方和one放的关系,因为在多对一的关系中,我们一般都是让many放维护关系的  item.setStockIncomeBill(stockIncomeBill);  // 4、计算该订单中每一条明细的总价格  BigDecimal number = item.getNumber();  BigDecimal costPrice = item.getCostPrice();  BigDecimal totalPrice = number.multiply(costPrice);  totalPrice.setScale(2, RoundingMode.*HALF\_UP*);  item.setAmount(totalPrice);  // 5、计算订单的总数量和总价格  BigDecimal currentTotalNumber = stockIncomeBill.getTotalNumber();  BigDecimal currentTotalCostPrice = stockIncomeBill.getTotalCostPrice();  stockIncomeBill.setTotalNumber(currentTotalNumber.add(item.getNumber()));  stockIncomeBill.setTotalCostPrice(currentTotalCostPrice.add(item.getAmount()));  } } |

入库单审核关键代码：

|  |
| --- |
| */\*\*  \* 入库单审核  \** ***@param*** *stockIncomeBill:需要被审核的入库单对象  \*/* public void audit(StockIncomeBill stockIncomeBill) {  // 通过页面仅仅对stockIncomeBill注入了Id值,stockIncomeBill的其余属性值都为null,所以如果在后续的操作需要使用到它的其他属性的时候,需要先根据id值将其从数据库中查询出来。  stockIncomeBill = stockIncomeBillDAO.getById(stockIncomeBill.getId());   if (stockIncomeBill.getStatus() == StockIncomeBill.*NORMAL*) {// 只有入库单没审核的时候才允许审核操作  // 设置审核人  stockIncomeBill.setAuditor(UserContext.*getUserContext*());  // 设置审核时间  stockIncomeBill.setAuditTime(new Date());  // 设置审核标志位已审核  stockIncomeBill.setStatus(StockIncomeBill.*AUTDIT*);   // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*库存操作\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 遍历入库单的明细 // ProductStock productStock = null;  List<StockIncomeBillItem> items = stockIncomeBill.getItems();  for (StockIncomeBillItem item : items) {  auditIncome(item.getProduct(),stockIncomeBill.getDepot(),item.getNumber(),item.getAmount(),item.getCostPrice());  }  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   stockIncomeBillDAO.update(stockIncomeBill);  }else {  throw new RuntimeException("该入库单已被审核过,请勿重复审核!!!");  }  }  */\*\*  \*  \** ***@param*** *product 入库单明细中的商品  \** ***@param*** *depot 入库单明细中商品所存放的仓库  \** ***@param*** *number 入库单明细中需要出库的商品的数目  \** ***@param*** *amount 入库单明细中需要出库商品的总金额  \*/* public void auditIncome(Product product, Depot depot, BigDecimal number, BigDecimal amount, BigDecimal costPrice) {  // 根据明细中商品的Id值和明细中仓库的Id,去库存中查找相应的库存情况,如果有找到就跟新数据  ProductStock productStock = productStockDAO.queryByPIdAndDepotId(product.getId(), depot.getId());  if (productStock != null) {  BigDecimal currentStockNumber = productStock.getStockNumber();// 获取当前已有库存的个数  BigDecimal currentStockTotalPrice = productStock.getStockTotalPrice();// 获取当前已有库存的总价  // 设置入库明细中商品的当前库存的个数,当前库存的个数 = 原来已有的库存量 + 当前明细中商品的个数  productStock.setStockNumber(currentStockNumber.add(number));  // 设置入库明细中商品的当前库存的总价,当前库存的总价 = 原来已有的库存总价 + 当前明细中商品的总价  productStock.setStockTotalPrice(currentStockTotalPrice.add(amount));  // 计算入库明细中商品的当前库存的售价,当前的库存售价 = 当前库存的总价 / 当前库存的个数  BigDecimal currentStockCostPrice = productStock.getStockTotalPrice().divide(productStock.getStockNumber(), 2, RoundingMode.*HALF\_UP*);  productStock.setStockCostPrice(currentStockCostPrice);// 设置入库明细中商品的当前库存的售价  productStockDAO.update(productStock);  } else {  // 没有找到,说明相应仓库中没有相应商品,就直接新增  productStock = new ProductStock();  productStock.setProduct(product);  productStock.setDepot(depot);  productStock.setStockCostPrice(costPrice);  productStock.setStockNumber(number);  productStock.setStockTotalPrice(amount);  productStockDAO.save(productStock);  } } |

## 5.3 报表管理

报表管理分为查看即时库存报表、查看订货报表、查看销售报表。

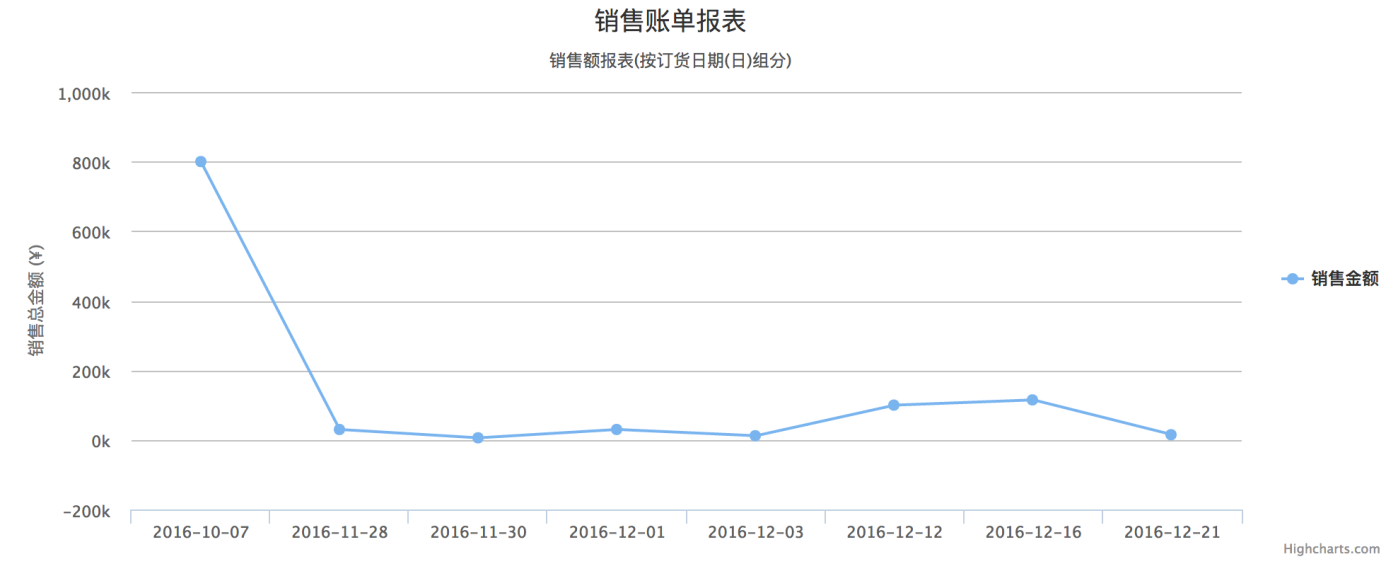
查询即时报表可以根据商品名称、仓库、品牌、闕值进行高级查询。订货报表可以根据业务时间范围、货品名称、供应商、品牌、订货日期（可以精确到月和日）、订货员进行高级查询。销售报表则可以根据销售人员、业务日期范围、客户、品牌进行高级查询。根据高级查询的结果，结合用户选择的报表样式生成相应的饼状图或线性图。



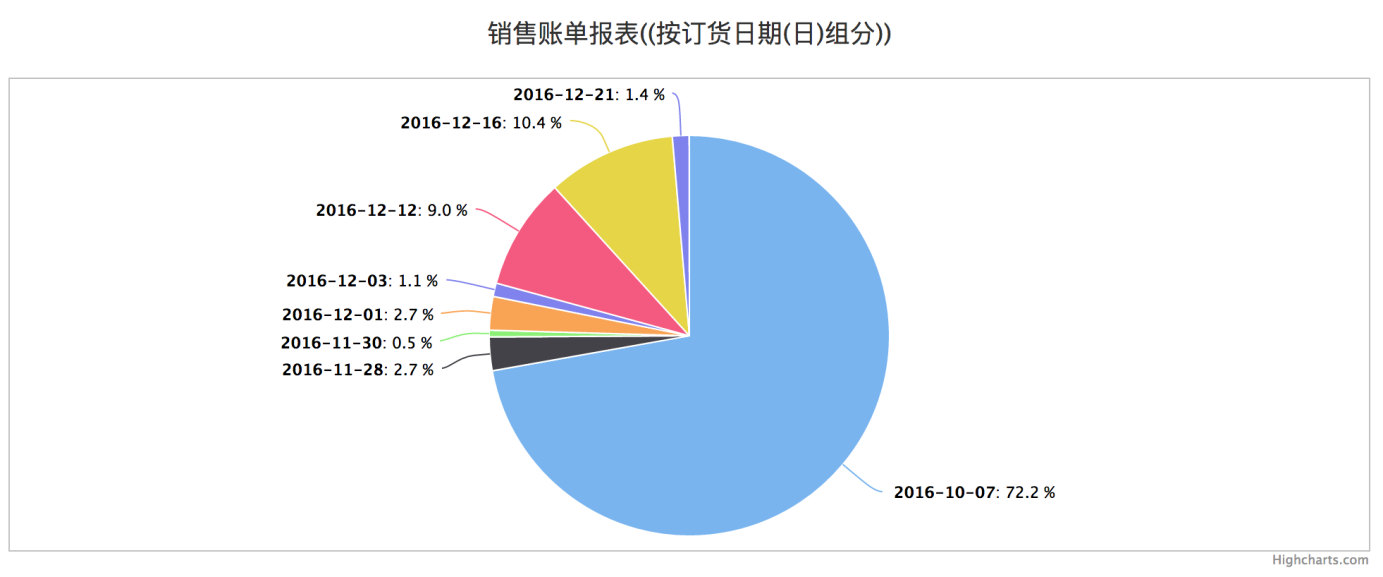
**图 5-6 即时库存报表实现效果图**

**图 5-7 订货报表实现效果图**

**图 5-8 销售报表实现效果图**

****

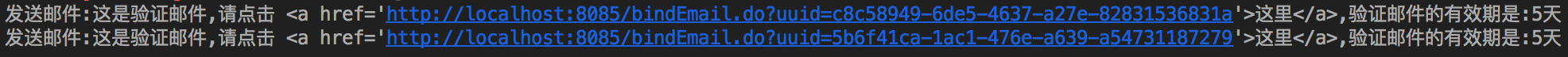
**图 5-9 根据**图5-8**销售报表生成的线程图实现效果图**

**图 5-10 根据**图5-8**销售报表生成的饼状图实现效果图**

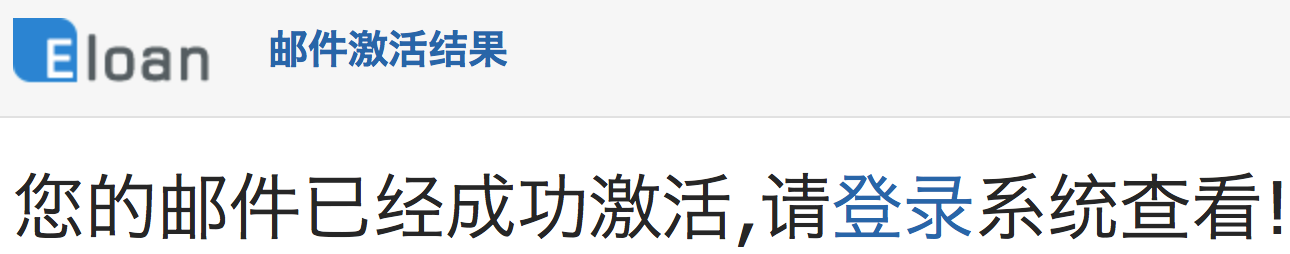
## 5.4 绑定邮箱

用户申请绑定邮箱时，页面上弹出输入邮箱的页面，用户输入邮箱地址后，系统会往用户指定的邮箱中发送一条验证邮件，之后用户登录自己的邮箱点击验证邮件链接即可激活邮箱，这是邮箱绑定成功。为了防止由于网络延迟或网络阻塞等原因造成的用户点击发送按钮之后页面没有立刻，而是依然停留在申请邮箱页绑定面，而此时用户在不知情的情况下多次点击发送按钮，此时系统会向用户指定的邮箱发送多个邮箱验证链接，此时用户不知道点击收到的那个邮件中的链接去验证绑定的邮箱的问题。设计了一个MailVerify类，该类中有uuid、loginInfoId、sendTime、email这四个字段，uuid是该类的唯一标示符，用于将来校验用户点击了验证链接后的响应，loginInfoId记录是哪个用户申请要绑定邮箱，sendTime记录邮件发送的时间，因为验证邮件的有效期是五天，email记录验证邮件发往了哪个邮箱，其中作用同uuid一样。每当用户点击了一次发送按钮之后，系统就会创建一个MailVerify的实例对象并存入到数据库中，无论用户用户点击收到的哪一个邮件中的验证链接，系统就会根据email和longInfoId取出往该用户指定的邮箱中发送有邮件时创建的MailVerify实例，循环遍历这些对象，只要有一个验证通过，就表明用户绑定邮箱验证成功。

 **(a) 申请绑定邮箱**



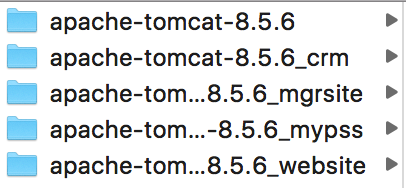
**(b) 收到的验证邮件**

** (c) 点击验证邮箱后的页面**

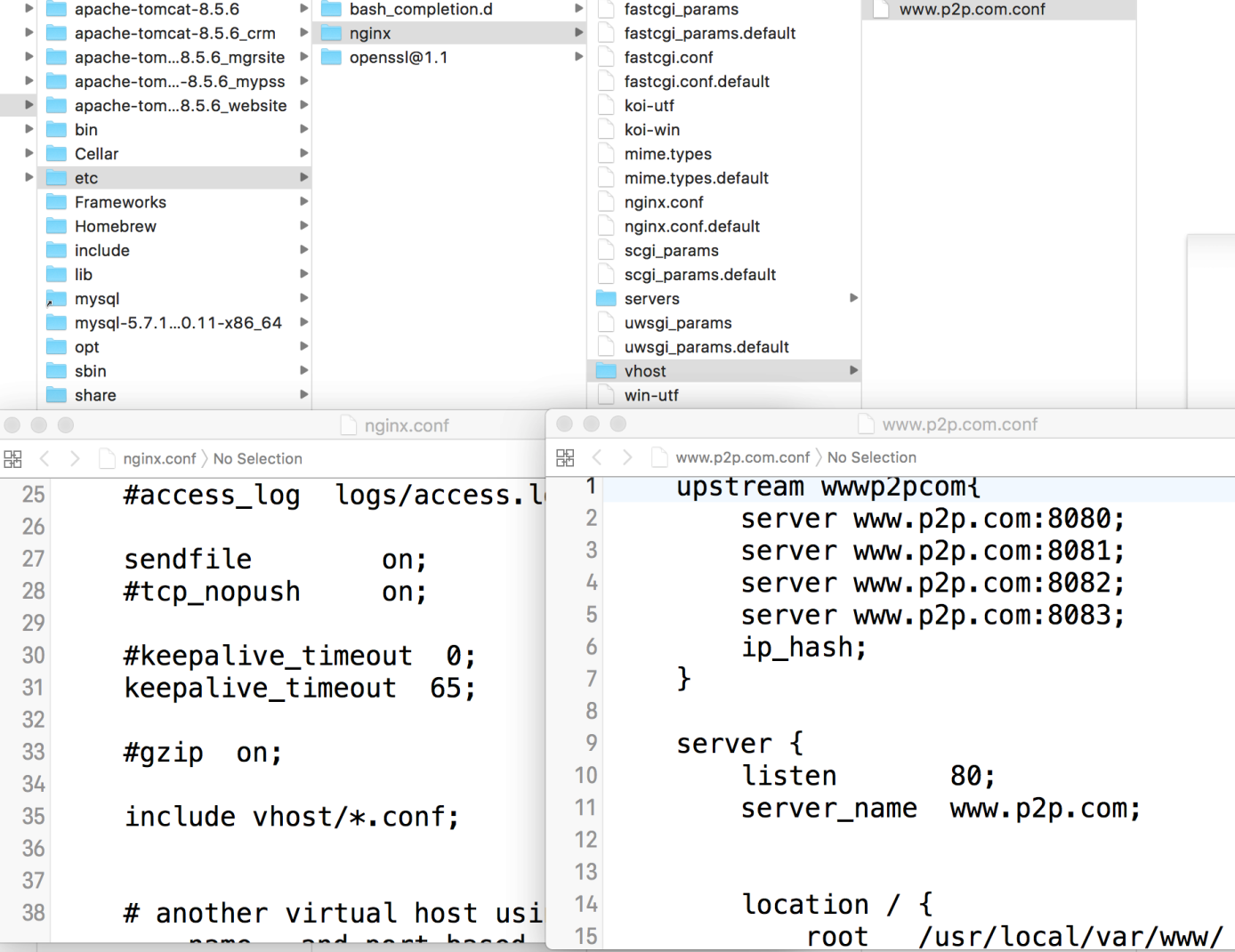
**图 5-11 用户绑定邮箱实现效果图**

## 5.5 服务器集群及负载均衡

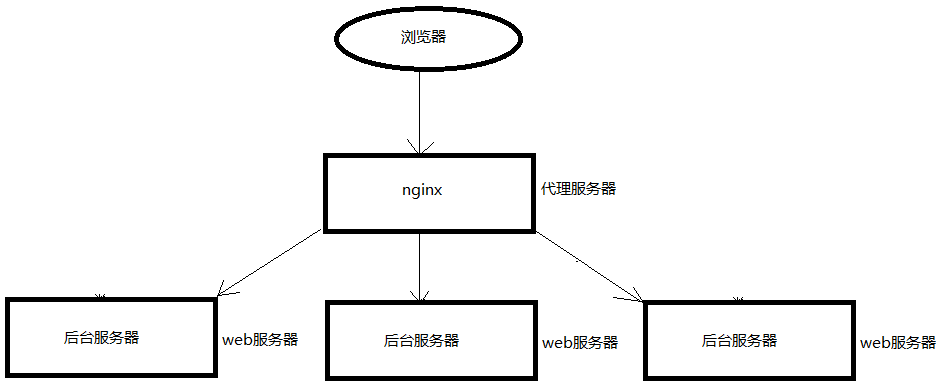
当系统注册使用的用户越来越多的时候，在同一时刻请求访问的并发量就会随之增大，尤其是在借款平台的前端服务器，随着系统的使用推广注册用户越来越多，当一个融资人的借款标发表前审核通过之后，瞬间会有多人进行抢标操作，这样就给服务器带来很大压力甚至会造成服务器宕机。为了有效避免这种情况的发生，除了对系统在业务层做了优化尽量避免对数据库的重复访问，对SQL查询语句做了优化，在进行联合查询的时候做到小表带动大表转动，对数据库表根据业务功能进行拆表之外，又对服务器进行了集群部署，并使用Ngnix对集群做负载均衡处理。负载均衡指的是后台服务器组成了一个服务器集群(多台服务器)，由中间服务器(nginx)接受到请求根据配置策略分发给不同的服务器后台，该nginx就是一个负载均衡服务器。负载均很的优点是分散后台服务器的压力、自动去掉挂掉的服务器、缓存后台服务器响应的内容。实现效果如图5-12所示。



**(a) Tomcat服务器集群**

****

**(b) Ngnix做负载均衡配置**



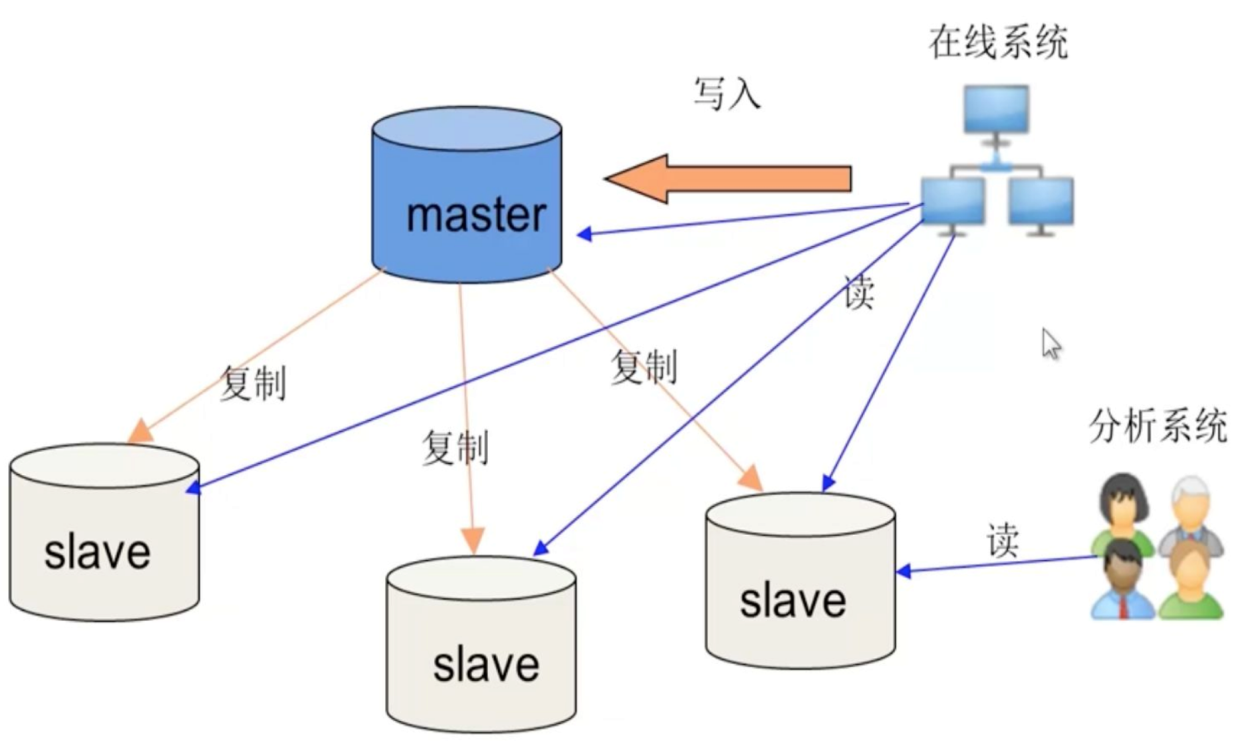
**(b) Ngnix做负载均衡示意图**

**图 5-12 服务器集群及负载均衡实现效果图**

使用Ngnix对服务器集群进行负载虽然可以解决单一服务器承受的并发访问压力过大时的宕机问题，但又有一个新的问题出现了，那就是session同步的问题。虽然使用hashIp的策略可以解决这一个问题，但ip\_hash策略是同一个ip的每一次访问都是访问同一台服务器，这样一来，当很多人的ip都映射到某一台服务器上的时候，依然会造成该服务器承受的并发访问压力过大相应缓慢甚至是宕机。因为采用的是ip轮训策略，所以即使是同一个用户连续两次操作也不会请求到同一台服务器上。这时就必须解决刚刚提到的session同步的问题了。进过摸索，解决方式是使用“过滤器+Redis内容数据库”，把用户的id作为key，用户当前会话的session作为value存入到Redis。然后自定义一个类MyHttpServletRequestWrapper继承自HttpServletRequestWrapper类，并重写getSession()方法，让getSession()方法的返回值是从Redis中根据当前用户的id值去取对应的session。接着自定义一个过滤器，在放行的时候使用自定义的MyHttpServletRequestWrapper类实例对象替换原来的HttpServletRequest对象，这样就能在服务器集群的情况下实现session同步。

## 5.6 数据库主从同步及读写分离

随着用户注册量的增加，数据库服务器也面临着和项目部署服务器同样的并发访问压力，也会因并发访问量过大而相应缓慢甚至是宕机。所以除了对数据库的表结构、SQL语句优化、使用Redis、Mamcached等缓存服务器以外，还需要给数据库服务器做集群。数据库集群面临的问题就是数据同步，因为如果数据库集群中的每天数据库服务器存放的数据不同步，当其中一台因不可抗拒的因素发生故障的时候，其他服务器是无法代替其工作的，所以必须对数据库服务器集群做同步。对数据库集群做数据同步必须要避免双向同步的问题，因为如果多台数据库服务器之间同时进行双向同步操作可能会造成数据覆盖的问题，也就是说在数据库服务器集群中数据同步必须是单向的，因此必须从集群中选出一个服务器作为主服务器（master），其余的作为从服务器。而且有所的从服务器（slave）中保存的数据都是从主服务器中同步过来的，因此系统与数据库集群的写操作都由主服务器承担，读操作有主从服务器共同承担，这就要求必须实现数据库服务器集群的读写分离。数据库服务器集群主从同步示意图如图5-13所示**。**



。 **图5-13 数据服务器集群主从同步示意图**

此时还要一个要解决的问题就是数据同步的时候是主动同步还是被动同步。主动同步是主服务器主动向从服务器发出同步请求，并把同步的数据发送给从服务器，但这样必须在配置文件中配置清楚集群中有几个从服务器，每个从服务器的地址。这种方式的弊端是如果集群中新增加了一个服务器或取出一个服务器或服务器地址改变的时候就必须重新写配置文件，并重启中服务器，这样灵活性就大大降低了。所以采用被动同步的方式，所谓的被动同步就是从服务器定时向主服务器发出同步请求，主服务器把需要同步的数据发送给从服务器。这样的好处是主服务器不需要知道有多少从服务器，新增加或减去一台从服务器或改变从服务器的地址都对主服务器没有响应。

因为本套系统使用的是MySQL数据库，所以就以MySQL集群来阐述数据库集群主从同步的步骤。要想实现数据库集群的主从同步，就必须让在Master上执行的所有的DML和DDL能够正确的在Salve上再执行一遍，注意MySQL选择使用文件来记录SQL语句。主从同步的步骤下表如下：

|  |  |
| --- | --- |
| 第一步 | 把在主服务器上的bin-log(二进制文件)打开，bin-log文件就可以记录在主MySQL上执行的所有的DML、DDL、TCL操作。 |
| 第二步 | 从MySQL后台一个线程发送一个请求到主服务器请求更新数据，最重要的请求参数是：本次请求，请求的是主MySQL的bin-log文件中哪一行数据之后的数据。 |
| 第三步 | 主MySQL后台一个线程接收到从MySQL发送的请求，然后读取其bin-log文件中指定的内容，并放在从MySQL的请求响应中。 |
| 第四步 | “从MySQL”的请求响应带回从主MYSQL中同步来的数据，然后写在从MYSQL中的relay-log日志文件(重做日志)中。relay-log日志文件中记录的数据就是从主MySQL中请求回来的那些需要同步的SQL数据。 |
| 第五步 | 从MySQL后台一个线程专门用于从relay-log中读取同步回来的SQL数据，并写入到从MySQL中，完成同步。 |

MySQL的主从同步是经过高度优化的，性能非常高。原因是首先这些同步过来的SQL都是写、修改的语句，没有查询语句，而且这些SQL都是主MySQL执行过的语句，所以都是没问题的sql语句，所以从MySQL在执行的时候就可以省去很多检查直接执行即可。其次MySQL中有专门的线程去执行这些操作，这些线程都是专门做过优化的。

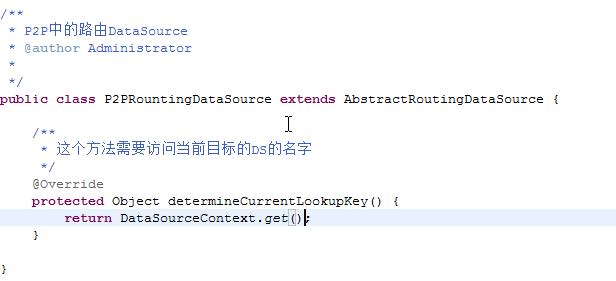
要想实现数据库服务器集群的读写分离功能，必须让系统的每一个业务方法都定位你到唯一一个数据库服务器上，也就是说必须在系统的业务功能对数据库进行操作时就已经明确本次要访问的是哪一台服务器，注意因为写操作必须访问主服务器。此时就需要配置一个充当路由器功能的数据源，经过查找资料发现在Spring中提供了一个叫做AbstractRountingDataSource的类，他就能提供所需要的路由器数据源的功能。该类有一下两个重要属性和一个重要方法：

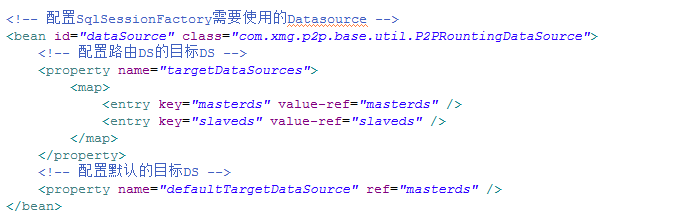
|  |  |
| --- | --- |
| targetDataSource | 用于配置真实的DataSource，这个属性是一个Map对象，Map的key就是DataSource的名字，Map的value是真实的DataSource对象。 |
| defaultTargetDataSource | 如果路由DS没有找到你当前请求的DS，直接使用默认的DS。 |
| abstract Object determineCurrentLookupKey() | 这个方法是需要自己实现的，这个方法有返回一个值，这个值就是DataSource的名字。其作用就是让应用提供给Spring怎么找DataSource名字的逻辑。 |

数据库集群实现读写分离的步骤如下：

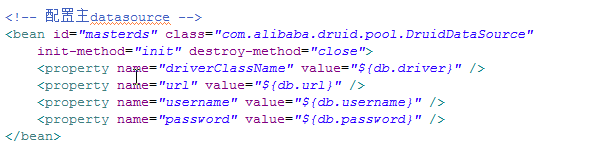
第一步、创建一个继承AbstractRountingDataSouddrce的对象，并在Spring中配置。

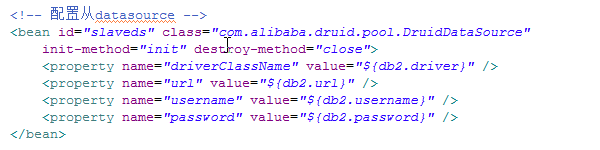






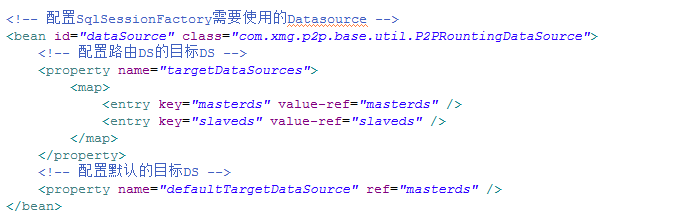
第二步、修改原来的DataSource的配置。



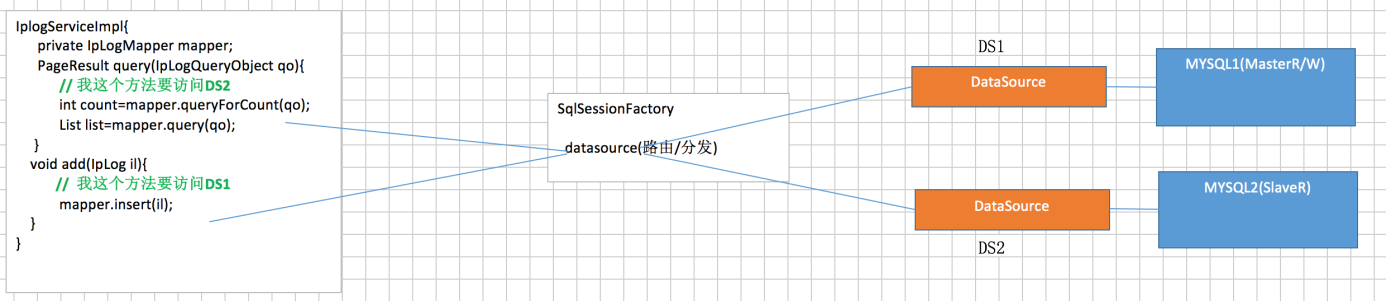
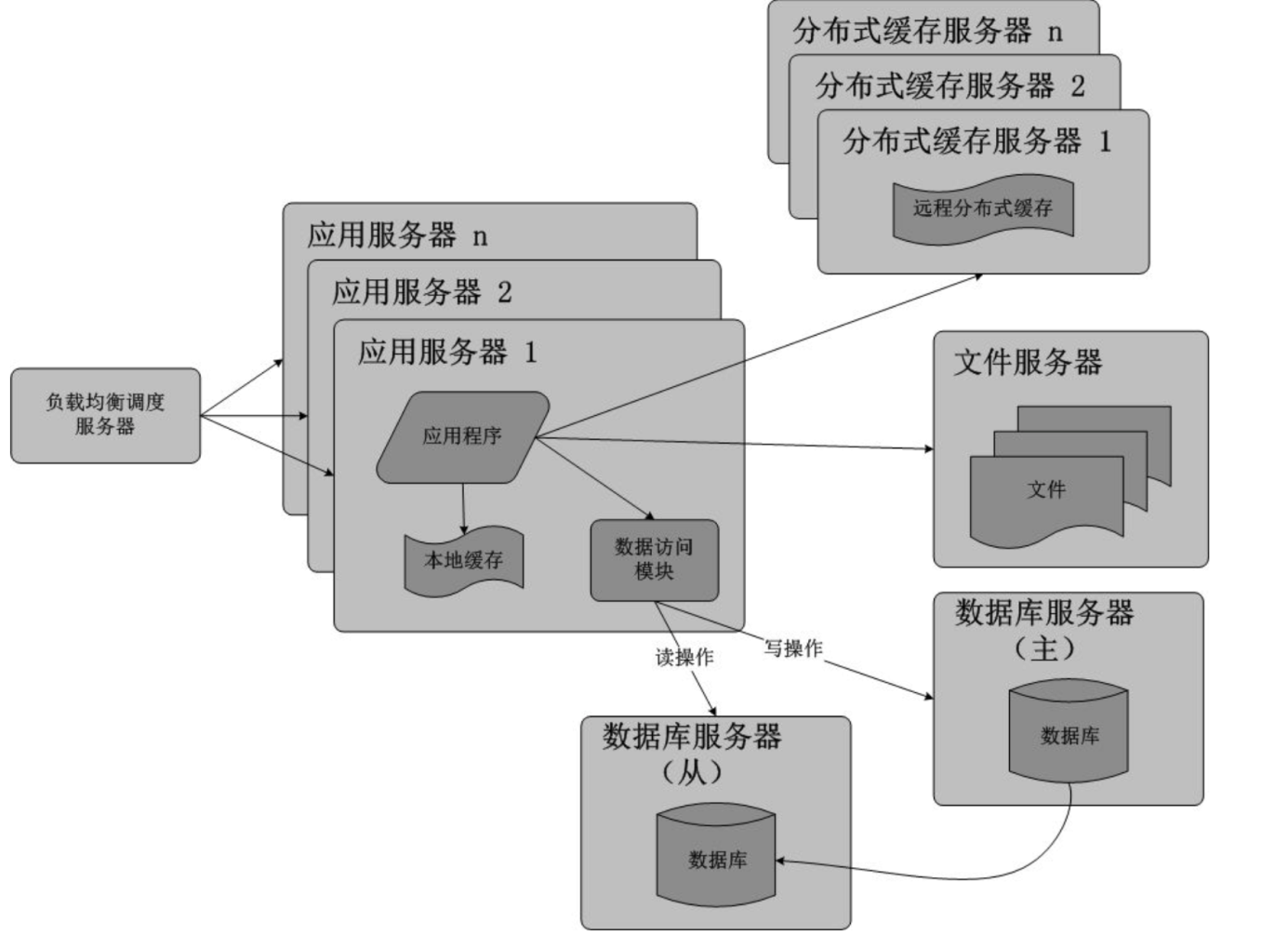


第三步、配置路由DataSource。

注意：SqlSessionFactoryz中使用的Datasource必须是路由Datasource。所以配置路由DataSource其实就是配置SqlSessionFactoryz中使用的Datasource。



数据库服务器集群主读写分离示意图如图5-14所示**。**



**图5-14 数据服务器集群读写分离示意图**

6 系统测试与维护

软件测试，指的是一套用来鉴定所开发软件的完整性、健壮性、安全性的手段。对所开发的软件系统进行测试是保证系统质量和稳定性的可靠防线。对于服务器开发而言，测试包括单元测试、有效性测试（黑盒测试）、压力测试。单元测试又称模块测试，是针对软件设计的最小单位 ─ 程序模块，进行正确性检验的测试工作。其目的在于发现各模块内部可能存在的各种差错。有效性测试是指在模拟的环境 (可能就是开发的环境) 下，运用黑盒测试的方法，验证被测软件是否满足需求规格说明书列出的需求。对系统不断施加压力的测试，是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。例如测试一个 Web 站点在大量的负荷下，何时系统的响应会退化或失败。我们要明确一点软件测试的目的并不是要给软件找问题，而是为了更好地保证软件的鲁棒性和使用过程中稳定的稳定性。针对服务器端软件系统的软件测试，既要保证正常操作情况下系统能够稳定运行，还必须要考虑到他人恶意操作系统时，系统要有足够的安全防范能力以保证程序的健壮性，比如数据库敏感字段加密、数据库防篡改等等[15]。

系统维护跟系统测试一样，是软件系统开发完毕之后必不可少部分，任何开发人员都不能保证系统部署上线之后能够长期运行不出现任何问题。系统后期的维护主要包括对系统使用过程当中出现的bug进行修复、有新业务需求时对现有的业务功能进行扩展、当对业务功能扩展时对数据库表的更新维护、页面样式及风格的更新、对数据库中的重要数据记性定期备份等内容。

## 6.1 系统测试

### 6.1.1 测试目的

在软件开发过程及上线部署之前制定一个完善的、细致的测试计划，不但能够追踪到测试的各个环节，还能够准确地对系统的各个功能是否达标都测试到位，保证系统各个功能的稳定性和健壮性。鉴于本系统所有功能比较明确，所以对本系统的测试计划是按照从角色分配、权限分配、管理员创建并分配角色到权限拦截到入库操作到出库操作到报表查询和从发表前审核到抢标再到满标二审结束最后到还款的顺序逐个功能点进行测试。

### 6.1.2 测试内容

在软件开发过程及上线部署之前制定一个完善的、细致的测试计划，不但能够追踪到测试的各个环节，还能够准确地对系统的各个功能是否达标都测试到位，保证系统各个功能的稳定性和健壮性。鉴于本系统所有功能比较明确，所以对本系统的测试计划是按照从角色分配、权限分配、管理员创建并分配角色到权限拦截到入库操作到出库操作到报表查询和从发表前审核到抢标再到满标二审结束最后到还款的顺序逐个功能点进行测试。首先要保证当系统部署上线成功的时候，系统会初始化完毕ApplicationContext的同时会去检查数据库的employee表中有没有admin超级管理员用户，没有就会创建一个admin用户并将其设置为超级管理员，因为ApplicationContext在整个应用中是单例的，所以能够保证系统只会自动去自动检测一次admin是否存在。这是以后能够登录系统，并能够创建其他管理员的关键。因为对数据库做了防篡改功能，如果系统启动的时候没有创建admin这个超级管理员的话，我们是无法通过手动修改数据添加的。使用admin用户登录上系统之后接着测试创建管理员并分配其一定的权限，并用创建的这个管理员去登录系统，看看他能不能看到超出他权限范围内的功能按钮以及分配给他的权限他是否都能正常访问。测试没问题之后再测试直接在地址栏输入删除管理的指令看看权限拦截能不能将其拦截下来。测试通过后在多创建几个用户并都赋予入库操作和出库操作权限，然后把这几个用户都登录上去，同时操作入库和出库操作以此来模拟高并发读写操作的情况，看看数据是否发生异常，是否有数据丢失、覆盖等情况。最后测试报表查询，还是使用多个用户同时查询，看看是否出现延迟现象。接着测试用户申请的融资招标通过发标前审核之后是否处于招标状态，而且是自己不能对自己发的标进行投资。没问题后就要高并发测试客户抢标，满标后满标额度是否刚好等于招标额度。测试通过后，然后测试满标二审之后，就要根据投资人及其投的股数创建明确的还款计划和每个还款计划的还款明细。最后测试用户还款时针对每个投资及其所有股数进行还款，所有还款计划执行完毕之后，应还利息和实际所还利息是否相等。

### 6.1.3 测试用例

对系统进行的功能测试结果如表6-1所示。

表6-1  **测试案例**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 用例编号 | Test Name（测试名称） | Step Name(步骤) | Description（实例描述） | Expected（预期结果） | Instance(实例结果) |
| 1 | 创建超级管理员 | 步骤1 | 在系统上线部署的时候，系统启动完毕的同时会自动穿件admin用户，其权限为超级管理员。 | 系统启动完毕的时候使用admin登录系统，能够正常登录。 | 通过，和预期一致 |
| 2 | 创建角色 | 步骤1 | 使用admin用户登录系统，然后点击系统模块 | 成功登录系统，系统回显系统模块子菜单功能正常 | 通过，和预期一致 |
| 步骤2 | 点击角色管理能够正常进入角色管理页面 | 跳转角色管理页面系 | 通过，和预期一致 |
| 步骤3 | 点击新增按钮正常跳转到新增页面，并且权限和菜单字典回显正确。 | 跳转到新增角色页面统创建角色、关联权限和菜单的功能正常 | 通过，和预期一致 |
| 步骤4 | 点击保存按钮，提交数据保存到数据库 | 回显“数据爆出成功” | 通过，和预期一致 |
| 3 | 采购订单管理 | 步骤1 | 点击业务模块 | 系统页面回显业务模块的所有子菜单 | 通过，和预期一致 |
| 步骤2 | 点击采购订单管理 | 跳转到订单管理页面 | 通过，和预期一致 |
| 步骤3 | 点击新增按钮 | 跳转到商品新增页面，页面中的商品选择功能正常 | 通过，和预期一致 |
| 步骤4 | 点击保存按钮，提交数据到数据库中 | 数据保存正常 | 通过，和预期一致 |
| 步骤5 | 回到采购订单管理页面，输入查询条件 | 模糊查询功能正常 | 通过，和预期一致 |
| 步骤6 | 点击查看按钮 | 跳转到商品信息页面，数据回显正常 | 通过，和预期一致 |
| 4 | 权限拦截 | 步骤1 | 普通管理员登录后，点击三个根菜单 | 只能在每个根菜单下看到其相应权限的子菜单，没有相应权限的子菜单看不到。 | 通过，和预期一致 |
| 步骤2 | 页面跳转到登录页面，数据库中数据没有变化 | 页面跳转到登录页面，数据库中数据没有变化。 | 通过，和预期一致 |
| 5 | 满标二审 | 步骤1 | 满标二审通过后查看借款标的详情 | 根据融资借款标的期数、投资人的人数和每个投资人所投的股数生成相应期数的还款计划，以及每一期还款计划的还款明细。 | 通过，和预期一致 |
| 6 | 用户还款 | 步骤1 | 用户点击“还款” | 跳转到还款页面 | 通过，和预期一致 |
| 步骤2 | 在还款页面用户可以选择即将到期的还款计划，点击该还款计划后边的“还款”按钮。 | 用户的可用余额减少，待还金额减少，待还利息减少，当前的还款计划变为已还状态 | 通过，和预期一致 |
| 7 | 系统催款 | 步骤1 | 更改系统时间为距离融资借款标还款的前一天 | 借款人绑定的手机号会受到系统发送的还款通知。 | 通过，和预期一致 |

系统的功能测试完全正常，没有出现任何问题。

使用LoadRunner测试工具对本系统进行负载压力测试，使用LoadRunner模拟多用户并发进行操作，测试结果见表6-2所示：

表6-2 **多用户并发操作时的测试结果（未使用Redis）**

|  |  |  |
| --- | --- | --- |
| 输入/动作 | 输出/响应时间（ms） | 能否正常进行 |
| 10个用户并发登录 | 10038 | 正常 |
| 50个用户并发登录 | 71659 | 正常 |
| 100个用户并发登录 | 2065365 | 正常 |
| 200个用户并发登录 | 12786435 | 故障 |
| 10个用户同时查询数据库 | 13567 | 正常 |
| 50个用户同时查询数据库 | 538956 | 正常 |
| 100个用户同时查询数据库 | 2012334 | 正常 |
| 200个用户同时查询数据库 | 13554898 | 故障 |

由测试的结果数据可知，当用户登录的并发操数小于等于10个的时候，系统平均响应的时间大约是一秒左右；但是当用户登录的并发操作数为200的时候，系统出现响应故障，所以本系统能够承受的用户并发登录的上限是200并发量。另外，系统对数据库进行读操作时，当并发量小于等于10的时候，平均耗时在2秒之内，这个时候系统性能的是比较好，而当并发量达到200的时候系统出现响应故障，所以本系统能够承受的用户并发读数据库操作的上限是200并发量。

根据上述编写的测试案例，覆盖了整套系统的全部功能，由此可以得出测试结果：本套系统的全部功能符合设计指标，在未达到并发量的瓶颈下能够运行正常，在上线部署到真实的生产环境时，可以通过加大服务器内存、使用Redis内存服务器、akka等技术手段解决并发量的瓶颈问题，综上测试结果分析可知，本套系统的测试结果达到了预期的开发需求。

## 6.2 系统后期的维护

系统后期的维护是软件系统生命周期当中耗时最长、延续时间最长的活动。系统后期的维护包括软件系统在使用过程当中bug的修复、功能的扩展、新业务需求的增加、数据库的优化、框架更新、软件系统结构的优化和前端界面内容及样式的美化等等。根据软件系统后期维护的不同性质，整天来说系统后期的维护可划分成一下四种不同类型：

|  |  |
| --- | --- |
| （1）纠错性维护 | 指的是软件系统在部署上线投入生产之后，对日常使用过程当中出现的功能性问题和漏洞进行调整和修复。 |
| （2）适应性维护 | 指的是为了使软件系统能够适应某种特定的生产环境变化而进行定制化修改。特别是互联网项目，新技术层出不穷，新框架日新月异，而且随着时间的推移用户量也会大量增加，这就需要做适应性维护，比如增加服务器做集群、对集群做负载均衡、。 |
| （3）完善性维护 | 指的是在软件系统使用的过程中，随着用户业务的扩展，用户可能会要求增加新的功能以满足需求，应用的户要求扩展新的功能而进行维护。 |
| （4）预防性维护 | 指的是对软件系统的维护不是等项目部署上线后，在使用过程中用户发现有不足之处提出要求后再进行修改，而是在设计的时候就主动进行预防性的维护，就是指一个新功能就完全可以在用户提出前就设计并推荐给用户。比如我们在设计数据库的时候就因该考虑到对数据库敏感字段进行加密、对数据库做防篡改等功能。 |

在系统维护中不仅仅要对应用程序的功能和性能进行维护，还要对数据库进行维护，比如定期备份数据库、对数据库服务器做灾备应对方案、保证数据库服务器集群读写分离但数据同步。只有保证数据库中的数据在不可抗拒性的灾难发生的时候，数据不丢失或者数据能及时恢复且不影响系统的使用才能够保证整个系统的健壮性。

### 6.2.1 影响系统维护的因素

在系统的后期维护中，往往会有一些不可抗拒的原因导致系统不能够继续进行维护或者维护成本大大超出重新开发一套系统的成本，而不得不放弃对该系统的维护，选择开发新的系统替代原有系统，影响系统维护的因素有以下几种：

|  |  |
| --- | --- |
| （1） 系统开发的时间和使用的年龄 | 如果是很早期的开发的项目，比如二三十年前开发的项目，已经投入生产将近三十年了，这些系统的维护价值微乎其微，而且维护难度非常大，可扩展性几乎为零。 |
| （2） 系统开发工具和开发语言 | 早期开发系统的工具可能早就停止了更行，当时使用的流行框架也停止了更行，对这样的系统进行更新，增加新的业务功能是非常困难的，而且新功能也不一定能够稳定使用。或者这些系统根本就无法满足新业务的需求，不存在维护的价值。 |
| （3）数据库和服务器相关技术 | 如果系统使用的早期的数据库技术，连分页查询功能都没有，也没有自带事务功能。而且当时配套使用的服务器根本无法应对20/s中以上的并发访问量，而新增加的业务功能是要有高级查询加分页，要抗住瞬间并发量100/s，这几乎是不可能的。 |
| （4）系统是否存在维护价值和维护难度系数 | 有些系统使用的开发语言早已停止更新，使用的相应技术手段早已被废弃，而且系统的当时整体设计就存在各种缺陷，仅仅是对某种业务很简单的使用目的而设计的。对它的维护已经没有任何价值，而且维护成本又极高，维护人员又如凤毛麟角。 |

### 6.2.2 提高系统的可维护性

系统后期的可维护性强是保证系统能够长期使用的基石，因此系统的后期可维护性必须被列入系统开发设计之列，在设计系统的时候就一定要考虑选用的设计方案对系统后期维护系难度数的影响，而不是在系统开发完毕之后再去考虑系统的维护工作。为了提高系统的后期可维护性，方便系统后期功能扩展、业务增加，可以通过使用以下几种方法来实现：

|  |  |
| --- | --- |
| （1）开发语言和开发工具的选择 | 选用开源的、参与维护的人数众多的、社区建设完善的、被广泛使用，尤其被众多知名企业使用的开发语言，比如Java等。选用功能完善、版本更新稳定、被广大开发人员喜爱的开发工具，比如IntellJ IDEA。 |
| （2）开发流程一定要规范 | 要有清晰、准确的注释，配备完善的开发文档。 |
| （3）在开发过程中做优化 | 开发过程中就要定期检查代码的逻辑，并及时做代码优化和重构，及时更新开发文档。 |
| （4）数据库要和代码同步优化 | 数据库开发人员要根据开发过程中业务逻辑的修正对数据库表的字段、外键关系以及多表之间的联合查询做优化，更要及时备份数据库中的数据以防万一。 |

7 总结与展望

经过查阅大量的资料和相应关键技术的研究学习，能够满足现代化大中型企业的仓储管理系统的设计与实现最终完成，预期的功能需求基本上都实现了。本系统是针对大中型企业的仓库管理而设计的，比如大型超市或综合性购物中心的仓库，这些企业的仓库往往存放的商品不但种类繁多、体量极大，而且交易频繁。仓库每天都有大量的商品入库、出库，所以这些的仓库管理部门必定要和采购部门、销售部门、财务部门、决策层及时有效地进行数据交流，以达到企业效益的最大化。而且能够实现资金的灵活配置也是企业决策层梦寐以求的功能。本套系统就以仓库管理的功能实现问核心，把系统的业务功能扩展到将仓库管理部门和企业的多个部门不但能够完美地协同办公，而且还大大简化了多部门间协同办公的流程。最重要的是数据报表能够在线生成，而且是以图表的形式展现给决策层，供其作为决策的依据和参考。另外为了解决一直以来最为困扰企业老板的资金链健壮性的问题也被纳入到本套系统的设计范围之内。因为仓库的日常作业必然是以资金流转作为支撑的，如果企业的资金链出现问题，仓库的正常作业就必然受到最直接的影响，比如某些商品滞销导致其他商品脱销了还没有资金及时采购入库，造成企业收益流失。所以本套系统中配备了借贷平台，而且是仅仅针对使用这套仓库管理系统的企业开放。而这套系统是针对大中型企业的仓库管理设计的，所以使用该仓储管理系统的企业规模都不小，都有非常强大的还款能力，他们融资最主要的原因就是为了扩展业务、扩建厂区或新建新的连锁店。而且也解决了降低资金充裕企业投资风险的目的。

本系统的设计和实现流程是：根据调查了解现在大中型企业对仓库管理系统的需求以及对现有仓库管理系统的不满意之处起草开发文档，接着根据起草的开发文档中的功能需求以及开发中应该注重考虑的功能细节来确定本系统的主要功能，考虑的系统后期的可维护性慎重考虑使用的开发语言和开发环境，然后确定系统的业务逻辑，最后进行系统的功能设计、数据库设计、系统前端页面设计。在对系统设计的过程中，从23中经典设计模式中选取了简单工厂模式、动态代理模式、单例模式，来达到尽可能地降低代码耦合度、提高代码质量的目的。考虑到用户的体验，根据“所见即所得”的原则，使界面美观大方、简约而不是优雅，使用户操作使用起来非常简单方便，降低拥有使用过程中的学习成本。

由于本人对相关技术掌握程度有限，系统在设计过程中还有一些瑕疵，比如在对服务器集群做负载均衡的时候使用的Ngnix框架，但对框架的整体配置细节了解的不是那么全面，没有完全实现页面的动静分离，仅仅做到了对集群中服务器的被动响应式负载均衡，虽然这个不足之处不影响本系统的使用，但为了使得系统更加健壮，会继续查找资料深入学习Ngnix的相关知识，彻底解决这个问题。

在对仓储管理系统的设计和开发过程中，遇到很多技术问题、难题和瓶颈，通过不断上网浏览各大知名博客网站、访问相关技术框架官网查看其开源的资料，在解决各种技术问题的同时，也提高自己技术水平，虽然过程非常艰辛，甚至有些技术问题让我一度怀疑人生，但是在自己的坚持下、不懈努力下攻克一个个技术难关，越过一个个技术障碍，最终将这款满足现代化大中型企业复杂管理需求的仓库管理系统成功设计出来，并且预期的功能需求最终基本上都实现了，这使我在技术方面又上了一个新的台阶，提升到了一个新的高度。在整个过程当中，不仅仅使我在技术水平上得以提高，最重要的是我掌握了对未接触过的技术的自学能力，以及解决开发过程中遇到的各种难题的能力。作为一名软件技术开发人员，虽然技术水平对我们来说非常重要，但对开发过程中遇到的各种问题的分析、解决能力也是至关重要的。我在设计和开发过程当中遇到的难题及解决方案如下：

参考文献

1. 李刚.疯狂Java讲义(第2版):电子工业出版社,2014.
2. 王向兵.JavaEE多层架构Struct2+Spring3+Hibernate3+Ajax的整合.大连海事，2009.
3. 林信良. Spring2.0技术手册[M]. 北京：电子工业出版社，2005.
4. 谢希仁.计算机网络[M].北京:电子工业出版社,2008.
5. 林寒超,张南平. Hibernate技术的研究[J]. 计算机技术与发展,2006, 16(11): 112-113,116.
6. 孙卫琴，李洪成. Tomcat 与 Java Web 开发技术详解[M]. 电子工业出版社，2003.6:1-205.
7. 萨师煊，王珊. 数据库系统概论[M]. 高等教育出版社，2002.2:3-460133.
8. 汤子瀛.计算机操作系统[M].西安:电子科技大学出版,2009.
9. 高昂,魏惠茹,李晓东,李博.MVC设计模式研究:电脑知识与技术[J],2016(1):16-17.
10. 郑阿奇.软件秘籍:设计模式那点事[M].北京:电子工业出版,2011.
11. Erich Gamma,Richard Helm,Ralph Johnson,John Vlissides 设计模式:可复用面向对象软件的基础[M].北京:机械工业出版社,2000.
12. 程杰.大话设计模式[M].北京:清华大学出版社,2007.
13. Ian Sommerville.软件工程[M]北京:机械工业出版社,2011.
14. 邵维忠,杨芙清.面向对象的分析与设计[M].电子工业出版社, 2010.
15. Java EE基于Hibernate的ORM框架http://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFD2010&filename=DZJS201002005&v=MTgzNTRIa1Y3L0lJVGZCZmJHNEg5SE1yWTlGWVlSOGVYMUx1eFlTN0RoMVQzcVRyV00xRnJDVVJMMmZZdVp1Rnk=
16. JavaEE核心模式研究http://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFD2014&filename=JSSG201401023&v=MzA2MDNZYWJHNEg5WE1ybzlIWjRSOGVYMUx1eFlTN0RoMVQzcVRyV00xRnJDVVJMMmZZdVp1RnlIa1ZMck9Mejc=
17. 张剑波,尚建嘎,李圣文.软件测试 原理方法与管理[M].北京:科学出版社,2015.

致 谢

时光荏苒四年的大学生活即将结束，大学生涯即将画上圆满的句号。回想其刚步入大学的时候院长和各位专业老师的谆谆教导，心中充满了依依不舍与期待，不舍的是一旦我们离开校园去天南海北闯荡，再想听听老师的不吝教诲是多么的困难，甚至是奢望，每当想起这些心中不满泛起酸楚。但同时也期待着把四年里从老师那里学来的专业知识结合实际需求灵活地运用到实际工作中，以此来向老师汇报自己用了大学四年时光铸造的成果。

在本次毕业论文设计与实现的过程中，我非常感谢梁雪老师的不吝指导，从论文题目选定、开题报告的撰写，到论文的构思与设计再到最终论文的定稿，梁雪老师都不厌其烦地给予我非常耐心、细心指导与鼓励，这才使我能够最终顺利完成的毕业论文。与此同时也要感谢院系的各位老师，感谢恩师们的辛勤无私奉献和孜孜不倦地教诲，在各位专业老师的传授和带领下，我不但学到了有用的专业技术，提高解决问题的能力，掌握了分析问题的方式，还养成了踏实严谨的学习态度。正是由于各位专业老师孜孜不倦的不吝教诲才有了我目前的优异成绩，才能达到顺利毕业的要求。我再次向各位兢兢业业，无私付出不吝付出的老师们致以我最由衷的感谢和最崇高的敬意！

学无止境，尤其是IT软件开发行业，技术更新迭代日新月异，如果没有踏实肯学的态度、没有良好的自学能力、没有恒心和毅力、没有扎实的理论功底是很难做到能够快速应用新技术，也就难以高效地融入软件开发者行列。在系统设计与实现的整个过程当中也暴露了自己在一些新技术掌握方面的不足、技术领域的接触面太狭窄、部门专业基础理论知识上还很匮乏，常常在查找资料解决开发中遇到的问题时会发现别的大牛给出的解决方中设计到的很多技术都是自己之前没有接触过的，在指导老师梁雪老师的帮助下才将问题最终顺利解决。这也是我惊醒大学生活的圆满结束并不意味着学习生涯就到此结束，而是在工作中需要学习更多的技术点来满足实际开发的功能需求，使我更加理解了老师们平时上课时总爱强调的学会分析问题比仅仅学会一个技术点要强的多，你们要学会如何学习而不是简单学会一个框架就行了。借助这次毕业论文的设计与实现使我弥补了很多之前没有接触过的技术方向，给我了一个查漏补缺的好机会，在今后的学习生涯和职业生涯中，我定会以老师们为楷模，时刻牢记各位老师的谆谆教导。

非常感谢在百忙之中抽时间对我的毕业论文进行审阅和评议的老师们，你们辛苦了！并向参加我毕业答辩的各位老师表示感谢！