



**本科毕业设计（论文）**

**基于JAVA的Web购物网站**

**学 院 计算机学院**

**专 业 软件卓越**

**年级班别 2014级（1）班**

**学 号 3114006535**

**学生姓名 王荣俊**

**指导老师 滕少华**

**2018年 5 月**

**基于JAVA的Web购物网站**

**王荣俊**

**计算机学院**

**摘 要**

随着社会的进步，人民生活水平的提高，电子商务的业务也迅速增长。该项目旨在模拟电子商城项目，建立一个通过软件和信息技术来达到在线购物的平台，提高卖家的售卖力，提升买家购物的便捷性，提高卖家产品的竞争力。

该项目的主要角色有两个，分为买家和卖家。对于买家来说，功能模块有注册，登录，修改个人信息，浏览商品，添加商品到购物车，管理收货地址，下单，修改订单状态，评价订单。对于卖家来说，功能模块有注册，登录，修改个人信息，管理商店，管理商品，修改订单状态，查看评论，统计商品销售情况。该项目程序后端的编写语言是使用java语言，数据库使用MySql数据库，前端使用jsp + vue + jquery + bootstrap，系统的架构是javaEE架构，使用Spring + SpringMVC + Hibernate框架。从而使项目具有良好的可伸缩性、灵活性和易维护性。

本文首先介绍了项目的背景和意义，概述项目内容，然后根据市场观察，提出项目的可行性分析和需求分析，在此基础上，进行系统的总体设计，数据库设计和系统结构设计。然后根据设计的内容一步一步实现各个功能。最后对系统进行总结和展望，并感谢所有在此过程中提供帮助的老师和同学。

**关键词：**Web购物网站，电子商城，电商，java语言，SSH框架

# 绪论

## 项目背景及意义

随着中国互联网和电子商务的飞速发展，社会的进步和消费的增长，各行各业蒸蒸日上，利益矛盾和竞争日益激烈，电商也不例外。除了业务上的竞争，对于顾客的竞争也不能落后，因此如何进一步提高服务质量，提升顾客的满意程度，改善运营管理水平和办事效率是一个企业保持竞争力的关键因素。在顾客就是上帝的服务行业(包括电商)，在时间就是金钱的今天，很多顾客对于在购物过程中将大量的毫无意义的时间耗费在货比三家地赶路，商品信息价格不透明，排队等待付款很不满意，因此，提供一个良好的交易环境已经为竞争的重要手段。

低效的货比三家，商品信息的不透明排队会大大降低买到优惠，合适商品的比率。人们为了买到比较优惠的商品，需要到多家店进行一一比较，既考验记忆力，又把大量的时间精力浪费在寻找店铺的过程中。此外，商品信息的不透明，没有统一的商品属性说明，也使顾客很难买到合适自己的商品。

虽然现在部分商店提供了送货上门的服务，能够电话预约，希望能够方便顾客购买商品，刺激消费，但是这并没有完全发挥它的最大作用。主要是因为存在一部分客户对其质量心存疑虑，怕送上门的与想象中不一致，维权又麻烦。

该项目旨在创建一个网上商城，建立一个通过软件和信息技术来达到公平，规范交易的电商平台，增加交易的透明度，交易时遵守既定流程，使交易更公平，更规范，提升顾客的购买满意度。

## 项目概述

该项目是Web购物网站，是一个买家购物，卖家管理商品的一个平台。

其主要功能是：卖家注册并登录，完善个人信息后，就可以注册多个商店，每个商品都有店铺名，描述信息，介绍图片，以及其下的商品。商店注册成功后就可以发布商品到这个商店中，并且定好商品的标题，类型，价格，描述，属性等基本信息。此时买家就可以浏览卖家发布的商品，查看商品的详细信息，包括商品评价，销量等。不过买家要添加商品到购物车或者对商品下单，就需要先注册登录并且完善个人信息后才能执行。买家下单后可以查看相应的订单信息，了解到订单的状态。

一旦买家下单，该商品所属的卖家就会收到通知，通知处理订单。等卖家发出商品后，需要修改订单的状态，告诉买家已经发货了。等到买家收货，就需要在系统上确认收货，本次交易就算是完成了。之后，买家可以对这一次购买进行评价，可以根据实际情况给好评，中评或者差评。之后其他买家就可以在商品详情页看到该评论。

当然，假如交易有问题，比如买家下单后由于某些原因不想买了，又或者货物运送过程中出现问题，又或者买家收到货后发现质量问题或者其他状况，买家可以申请退款。

# 系统分析

## 2.1可行性分析

### 2.1.1市场可行性

该系统是Web购物网站，这种网站虽然在市面上早就已经出现，但是仍有广大的实体店是没有进驻购物网站。在如此众多数量的实体店中，几乎每一个实体店都会有发展更多销售渠道的需求，所以该系统是非常有市场的，市场可执行性很高。

### 2.1.2技术可行性

程序后端的编写语言是使用 java 语言，数据库使用 MySql 数据库，前端使用 jsp + vue + jquery + bootstrap，系统的架构是 javaEE 架构，使用 Spring + SpringMVC + Hibernate 框架。从而使项目具有良好的可伸缩性、灵活性和易维护性。

Java是一种可以撰写[跨平台](http://baike.baidu.com/view/469855.htm" \t "_blank)应用程序的[面向对象](http://baike.baidu.com/view/125370.htm" \t "_blank)的程序设计语言。[Java](http://baike.baidu.com/subview/29/12654100.htm" \t "_blank) 技术具有卓越的通用性、高效性、平台移植性和安全性，广泛应用于[PC](http://baike.baidu.com/subview/13621/4882082.htm" \t "_blank)、[数据中心](http://baike.baidu.com/view/754520.htm" \t "_blank)、游戏控制台、科学[超级计算机](http://baike.baidu.com/view/296324.htm" \t "_blank)、移动电话和互联网，同时拥有全球最大的开发者专业社群。Java是功能完善的通用程序设计语言，可以用来开发可靠的、要求严格的应用程序。

Spring + SpringMVC + Hibernate 框架系统从职责上分为四层：[表示层](http://baike.baidu.com/view/239615.htm" \t "_blank)、控制层、[业务逻辑层](http://baike.baidu.com/view/1030527.htm" \t "_blank)、[数据持久层](http://baike.baidu.com/view/1862970.htm" \t "_blank)，以帮助开发人员在短期内搭建结构清晰、可复用性好、[维护](http://baike.baidu.com/view/1123731.htm" \t "_blank)方便的[Web应用程序](http://baike.baidu.com/view/138452.htm" \t "_blank)。其中使用 SpringMVC 作为系统的整体基础架构，负责MVC的分离，利用Hibernate框架对[持久层](http://baike.baidu.com/view/198047.htm" \t "_blank)提供支持，Spring做管理，管理 SpringMVC 和 Hibernate。采用 Spring + SpringMVC + Hibernate 框架，不仅实现了视图、控制器与模型的彻底分离，而且还实现了[业务逻辑层](http://baike.baidu.com/view/1030527.htm" \t "_blank)与持久层的分离。这样无论前端如何变化，模型层只需很少的改动，并且数据库的变化也不会对前端有所影响，大大提高了系统的可复用性。而且由于不同层之间[耦合度](http://baike.baidu.com/view/1599212.htm" \t "_blank)小，有利于团队成员并行工作，大大提高了开发[效率](http://baike.baidu.com/view/47610.htm" \t "_blank)。

Bootstrap 是一个快速开发前端页面的框架，而 Bootstrap 的目标就是帮助web开发者更轻松的打造出功能丰富并且美观的UI界面，是一个非常成熟的框架。

MySQL是目前最流行的[关系型数据库管理系统](http://baike.baidu.com/view/1450387.htm" \t "_blank)，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。MySQL是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，许多中小型网站为了降低网站总体拥有成本而选择了MySQL作为网站数据库。

JSP是一种跨平台的动态网页技术,在静态页面中嵌入Java代码片段,再由Web服务器中的JSP引擎来进行编译并执行嵌入的Java代码片段,生成的页面信息返回给客户端。可以实现与用户的交互,利于扩展,基于B/S技术的动态网页。

在B/S的系统中，用户可以通过浏览器向分布在网络上的许多服务器发出请求。B/S结构极大的简化了客户机的工作，客户机上只需安装、配置少量的客户端软件即可，服务器将担负更多的工作，对数据库的访问和应用程序的执行将在服务器上完成。

Hibernate是一个将对象映射到数据库中的轻量且快速的ORM解决方案。它减轻开发人员处理低级别的数据库需求，同时节省开发时间。写SQL和解析查询结果是相当繁琐和耗时的任务。hibernate使开发人员远离这些java对象映射到数据库表（称为ORM，对象/关系映射）的繁琐操作，做到使用一个简单的面向对象的API对java对象进行存储、删除、查询、更新。为使hibernate更好地应用到实际项目情景中，对其进行以下拓展封装和优化：增加指定字段默认值属性、数据库升级封装优化、注解属性优化。

### 2.1.3操作可行性

本系统会使用相对人性化的界面设计，尽量使使用者舒服的操作简单清晰的界面，尽量使用鼠标对界面进行操作。

## 2.2需求分析

### 2.2.1 功能需求概述（买家）

（1）注册：用户在登录页面输入手机号，密码，性别，昵称，姓名，用户身份（买家还是卖家）等个人信息进行注册。

（2）登录：用户输入手机号，密码，验证码进行登录。

（3）修改个人信息：登陆后可以对个人信息如昵称，姓名，密码，头像进行修改。

（4）首页展示商品：用户在首页可以看到商品类型列表以及商品列表。可以选择某个商品类型查看该类型的商品，也可以在搜索栏输入关键字查找感兴趣的商品。

（5）搜索商品：可以根据关键字，商品类型等查询条件查询出想要的商品。

（6）显示商品详情：用户点击进入某个商品的详情页，可以看到商品的详细信息，包括商品名称，描述，图片，价格，所属的商店，用户评价。

（7）添加商品到购物车：用户在商品详情页输入购买数量，点击添加购物车，就可以把商品放入到购物车中。

（8）购物车管理：打开购物车页面，可以看到存放在购物车中的商品列表以及对应的购买数量。可以修改购买数量，也可以把商品从购物车中移除。此外，还可以把购物车中选中的商品结账。

（9）下单：在商品详情页选好颜色和数量后，可以结账进行下单，也可以在购物车页面选中需要购买的商品结账下单。

（10）管理收货地址：在下单页面需要填写收货地址。收货地址包括联系人姓名，联系人电话，地址这三个信息，买家可以添加收货地址，选择已有的收货地址。在我的收货地址页面中，可以添加收货地址，修改收货地址，设置默认收货地址。

（11）查看订单：查看订单信息，包括订单对应的商品，购买数量，收货地址，订单状态。

（12）确认收货：收到商品后，需要到对应的订单确认收货。

（13）评价订单：确认收货后，可以对本次购物进行评价。评价分为评价等级和评价内容。评价等级分为好评，中评，差评。评价内容是买家对本次购物感受以及商品情况的一段描述文字。

（14）申请退款：买家可以因为收不到货，不想买，买错了，商品不符合预期而申请退款。

### 2.2.2 功能基本流程

（1）客户：

当客户来到营业厅取得票之后，在休息区等待柜员叫号，被叫到号的客户依次前往窗口办理业务

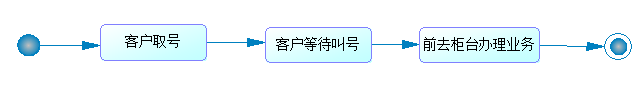


图2.1 客户取票流程图

（2）柜员：

柜员首先要登陆叫号系统，选择要办理的业务和窗口，然后开始叫号，如果有客户前来窗口办理，就为该客户办理业务，如没有客户前来办理，则叫下一个号。

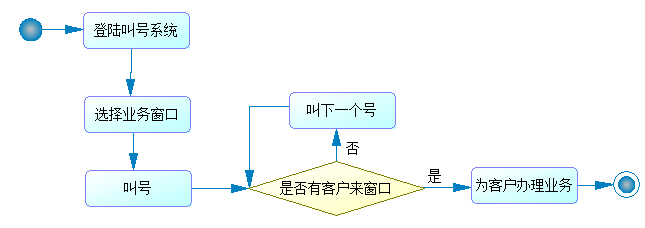


图2.2 柜员叫号流程图

### 2.2.3 功能用例描述

（1）取号叫号总体用例图：

本系统的主要参与者(Actor)有客户和柜员，主要的用例(Use Case)有取号，叫号，办理业务，选择窗口，登录系统。客户在大厅进行取号，然后等待排队，柜员需要登录叫号系统，根据业务类型选择窗口，然后进行叫号。被叫号的客户前去窗口办理业务，柜员则为客户办理业务。



图2.3 取号叫号总体用例图

（2）管理系统总体用例图：

系统管理员拥有管理业务，管理窗口，管理柜员信息，查看统计信息，查看历史记录的权限，管理着整个系统的后台数据。



图2.4管理系统总体用例图

### 2.2.4 用例详细文档

（1）取号用例

客户来到营业厅取号的时候可以选择会员类型和业务类型。等待被叫号。



图2.5取号用例图

**表2.1 取号用例说明**

|  |
| --- |
| 用例名称：取号 |
| 参与者：客户 |
| 简要说明：客户在大厅根据自己的会员类型和需要办理的业务进行取号 |
| 前置条件：前往营业厅 |
| 事件流：(1)基本流：   1. 客户前往营业厅办理业务 2. 选择业务类型和会员类型 3. 取号。 |
| 异常事件流： |
| 后置条件：取得票号，等待被叫号 |
| 注释：非vip选择普通会员，vip选择vip会员 |

（2）办理业务用例

客户前去窗口办理业务，柜员为客户办理业务，办理结果有成功和失败两种，办理失败需要柜员填上办理失败原因。



图2.6办理业务用例图

**表2.2 办理业务用例说明**

|  |
| --- |
| 用例名称：办理业务 |
| 参与者：客户，柜员 |
| 简要说明：客户去对应的窗口办理业务，柜员为客户办理业务 |
| 前置条件：客户被叫到号，柜员登录到叫号系统 |
| 事件流：(1)基本流：   1. 客户被叫到号，前去对应窗口办理业务 2. 柜员工具客户的需求为客户办理业务 3. 客户业务办理完成   (2)备选流：如果办理失败，客户可以重新取号，柜员叫下一个号 |
| 异常事件流： |
| 后置条件：柜员叫下一个号 |
| 注释： |

（3）叫号用例

柜员登录系统后，可以叫号，如果临时有事离开，可以选择暂停叫号，如果叫号后客户没有立即前来办理，可以选择重复叫号，办理成功或没人前来办理，都可以选择叫下一个号



图2.7叫号用例图

**表2.3 叫号用例说明**

|  |
| --- |
| 用例名称：叫号 |
| 参与者：柜员 |
| 简要说明：柜员根据取号顺序为排队客户办理业务 |
| 前置条件：客户取到号，柜员登录到叫号系统 |
| 事件流：(1)基本流：   1. 柜员根据票号顺序叫号 2. 为所叫号的客户服务，办理其所需业务 3. 为客户业务办理完成   (2)备选流：  1.如果叫号失败，柜员叫下一个号  2.如果叫号一次没人到窗口，重复叫号一次到两次  3.如果办理失败，柜员叫下一个号 |
| 异常事件流： |
| 后置条件：柜员叫下一个号 |
| 注释： |

（4）登录与选择窗口用例

柜员在登陆的时候要选择自己所在的窗口号，然后输入信息登录到叫号系统。



图2.8登录用例图

**表2.4 登录用例说明**

|  |
| --- |
| 用例名称：登录、选择窗口 |
| 参与者：柜员 |
| 简要说明：柜员根据自己所在窗口好选择窗口进行登陆 |
| 前置条件：柜员有对应的账户密码 |
| 事件流：(1)基本流：   1. 柜员根据自己所在窗口好选择窗口进行登陆 2. 为所叫号的客户服务，办理其所需业务 3. 为客户业务办理完成   (2)备选流：如果登陆失败，请联系管理员 |
| 异常事件流：登录密码不准确或者账号不存在 |
| 后置条件：柜员登录到叫号系统 |
| 注释： |

（5）管理业务用例

后台系统管理员可以登录系统，对业务进行管理，包括增删改查，以及禁用业务和开启业务。



图2.9管理业务用例图

**表2.5 管理业务用例说明**

|  |
| --- |
| 用例名称：管理业务 |
| 参与者：管理员 |
| 简要说明：管理员对业务进行管理 |
| 前置条件：管理员登录了管理系统 |
| 事件流：(1)基本流：   1. 管理员根据需求对于业务进行增删改查和禁用。   (2)备选流：如果操作失败，请查看数据库和限制条件 |
| 异常事件流：a. 修改不了，提示修改异常  b. 增加不了业务，提示添加异常  c. 删除不了，提示删除异常  d. 禁用不了，提示禁用异常 |
| 后置条件： |
| 注释： |

（6）管理窗口用例

后台系统管理员可以登录系统，对窗口进行管理，包括增删改查，以及禁用窗口和开启窗口。



图2.10管理窗口用例图

**表2.6 管理窗口用例说明**

|  |
| --- |
| 用例名称：管理窗口 |
| 参与者：管理员 |
| 简要说明：管理员对窗口进行管理 |
| 前置条件：管理员登录了管理系统 |
| 事件流：(1)基本流：   1. 管理员根据需求按照营业厅的窗口数增加窗口书数 2. 管理员根据需求对窗口增删改查和禁用   (2)备选流：如果操作失败，请查看数据库和限制条件 |
| 异常事件流：a. 修改不了，提示修改异常  b. 增加不了业务，提示添加异常  c. 删除不了，提示删除异常  d. 禁用不了，提示禁用异常 |
| 后置条件： |
| 注释： |

(7) 管理柜员信息用例

后台系统管理员可以登录系统，对柜员信息进行管理，包括增删改查。



图2.11管理柜员用例图

**表2.7 管理柜员信息用例说明**

|  |
| --- |
| 用例名称：管理柜员信息 |
| 参与者：管理员 |
| 简要说明：管理员对柜员进行管理 |
| 前置条件：管理员登录了管理系统 |
| 事件流：(1)基本流：   1. 管理员根据需求按照增加厅的柜员数 2. 管理员根据需求对柜员信息进行增删改查   (2)备选流：如果操作失败，请查看数据库和限制条件 |
| 异常事件流：a. 修改不了，提示修改异常  b. 增加不了业务，提示添加异常  c. 删除不了，提示删除异常  d. 禁用不了，提示禁用异常 |
| 后置条件： |
| 注释： |

（7）查看统计信息用例

后台系统管理员可以登录系统，可以查看统计信息，包括流量统计以及成功率和失败率的统计



图2.12查看统计信息用例图

**表2.8 查看统计信息用例说明**

|  |
| --- |
| 用例名称：查看统计信息 |
| 参与者：管理员 |
| 简要说明：管理员对各个时段不同业务的统计信息进行查看 |
| 前置条件：管理员登录了管理系统 |
| 事件流：(1)基本流：   1. 管理员选择查看流量统计或者成功与失败率。 2. 管理员选择某个时段某个业务类型进行查看 3. 系统生成形象的统计图形 |
| 异常事件流： |
| 后置条件： |

（8）查看办理情况用例

后台系统管理员可以登录系统，可以查看办理情况，包括当前办理情况和历史记录



图2.12查看办理情况用例图

**表2.8 查看办理情况用例说明**

|  |
| --- |
| 用例名称：查看办理情况 |
| 参与者：管理员 |
| 简要说明：管理员对当前和历史办理情况的查看 |
| 前置条件：管理员登录了管理系统 |
| 事件流：(1)基本流：   1. 管理员选择查当前办理情况或者历史记录   (2)备选流：1. 当管理员选择查看当前办理情况时，当前办理和已办理以及未办理人数。  2. 当管理员选择查看历史记录时，可以查看业务办理的历史记录 |
| 异常事件流： |
| 后置条件： |
| 注释： |

### 2.2.5 非功能性需求

非功能需求是指依一些条件判断系统运作情形或其特性，而不是针对系统特定行为的需求，我们的软件产品为了满足用户业务的需求就必须具有除功能需求以外的特性，它一般和系统的状态有关而与系统需要提供的功能无关，是功能需求的有力补充。一般软件产品的非功能需求包括系统的可用性、性能、可靠性、可支持性、其他等。

1、可用性

可用性是一个非常宽泛的概念，它泛指那些能让用户顺利使用系统的指标，包括易用性（易操作、易理解）、安全性（权限体系、访问限制）、兼容性（服务器、客户端的兼容度），等等

易用性：本系统叫号与取号的窗口均为全中文的windows窗口菜单形式，后台管理系统是以一个简单易用的网页展示，基于web浏览器访问，非常容易操作。

安全性：

兼容性：服务器使用tomcat，客户端使用浏览器，即B/S架构，兼容性方面没有什么问题。

2、性能需求。

系统的数据量不会太大，在数据库方面会使用缓存和索引等，性能是可以得到保障的

3、可靠性

系统可以可靠运行。数据要按时进行备份

4、可支持性

由于使用Java语言来编写此系统，对于跨平台是毫无问题的；但对于不同的浏览器的兼容问题是需要考虑的一个问题。

# 系统设计

## 3.1总体设计

### 3.1.1系统总体结构图

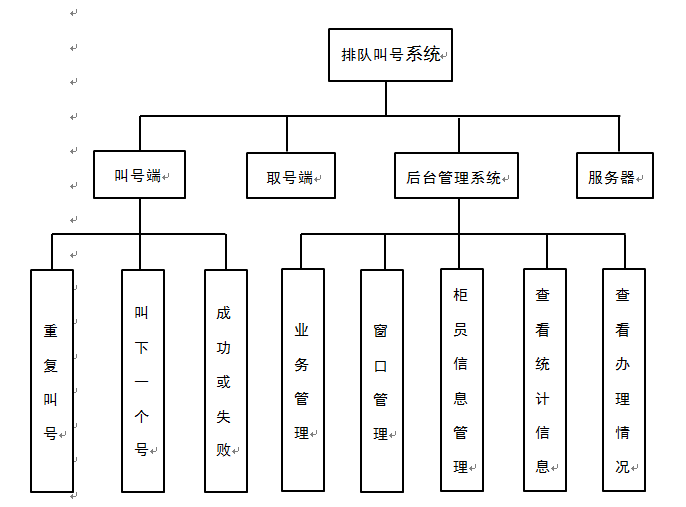


图3.1系统总体结构图

排队叫号系统包括叫号端，取号端，服务器以及后台管理系统四个大的模块，只有当服务器开启式，才可以打开叫号端和取号端，服务器与叫号端还有取号端之间使用socket进行连接，连接协议是tcp/ip协议，遵守三次握手的原则。取号端可以根据客户自己的需要选择相应的业务，叫号端可以对号码进行重复叫号并且统计办理失败或者成功。后台管理系统主要有五个模块，对业务进行管理，对窗口进行管理（配对哪个窗口办理什么业务），对柜员的信息进行管理，查看统计信息以及查看办理情况。

### 3.1.2开发运行环境

系统开发环境：windows

运行平台：windows

数据库：mysql 5.1.73

Web服务器：tomcat

系统开发工具: Eclipse

系统开发技术: Struts + Spring + Hibernate + Maven + jQuery EasyUI;

## 3.2数据库设计

### 3.2.1数据库的需求分析

依据项目的需求以及业务逻辑，可以设计如下数据表及其功能：

业务表：保存业务信息，主要的属性有业务代号，业务名称，业务的状态（是否被禁用），以及业务的的限制量。

管理员信息表：保存管理员的信息。其主要属性有员工号，员工姓名，管理系统登录账号以及密码。

柜员信息表：保存柜员信息，其主要属性有员工号，员工姓名，叫号系统登录账号以及密码。

取的票的表：保存票的信息，其主要属性有票号，票的状态（是否被叫过），票的取票时间，办理情况以及等待人数。

窗口表：保存窗口信息，主要的属性包括窗口的窗口号，窗口的描述，以及窗口的状态（是否被禁用）。

### 3.2.2数据库概念结构设计

#### 3.2.2.1实体与属性

1、业务实体，属性包括业务代号，名称，状态，限制量

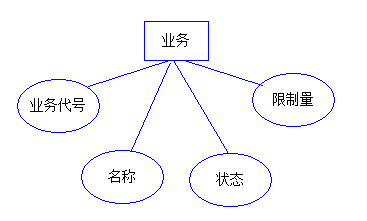


图3.2业务实体E-R图

2、窗口实体，属性包括窗口号，状态，办理业务代号，使用者

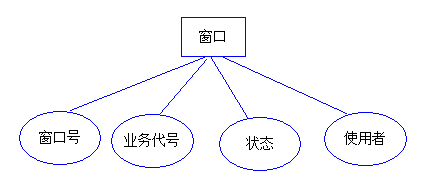


图3.3窗口实体E-R图

3、柜员实体，属性包括用户名，姓名，密码

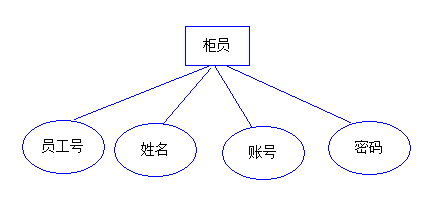


图3.4柜员实体E-R图

4、管理员，属性包括

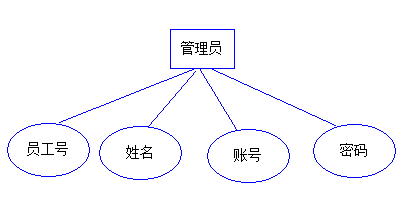


图3.5管理员实体E-R图

5、票实体，属性包括

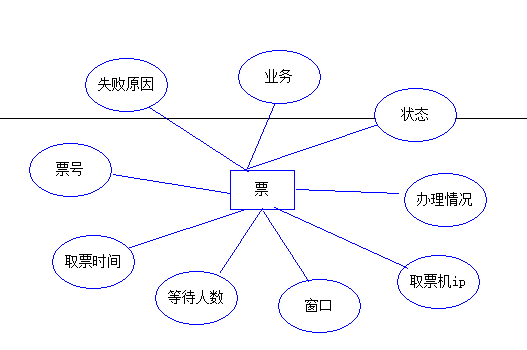


图3.5票实体E-R图

注：状态指的是是否被叫到，办理情况是指是否成功办理

#### 3.2.2.2实体与实体之间的关系

由于各个实体的属性在上面已经列出，在这里只列出实体和实体与实体之间的关系。在系统中，管理员管理柜员，窗口以及业务，可以查看票的相关信息，柜员办理业务，和业务之间是一对多的关系，业务和窗口对应，一个窗口只能办理一种业务，一种业务可以有多个窗口办理，在客户拿的票当中，存在业务名，和业务是多对一的关系，柜员需要处理票，与票是一对多的关系。如图3.6所示。

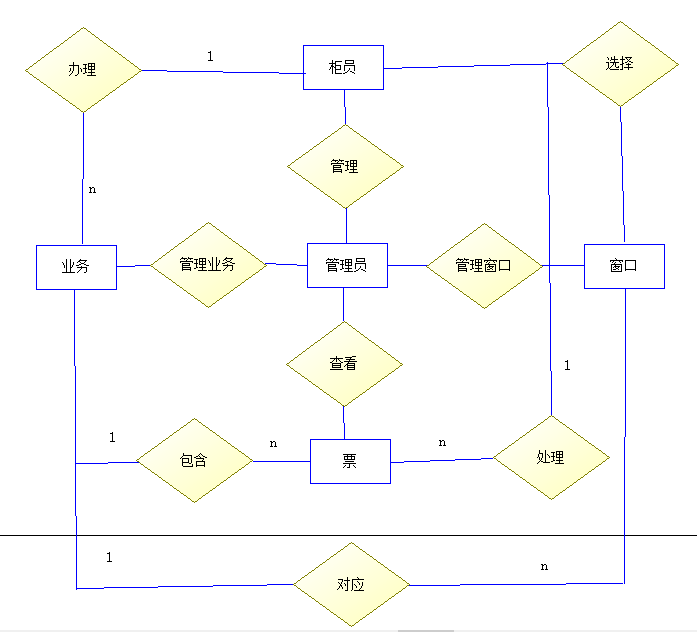


图3.6实体之间关系图

### 3.2.3数据库逻辑结构设计

根据以上实体属性以及实体与实体之间的关系，我们可以设计各个表的字段值，以及各个表之间的关联，表的逻辑结构。从而建立一个数据库的逻辑结构模型，如图3.7所示。

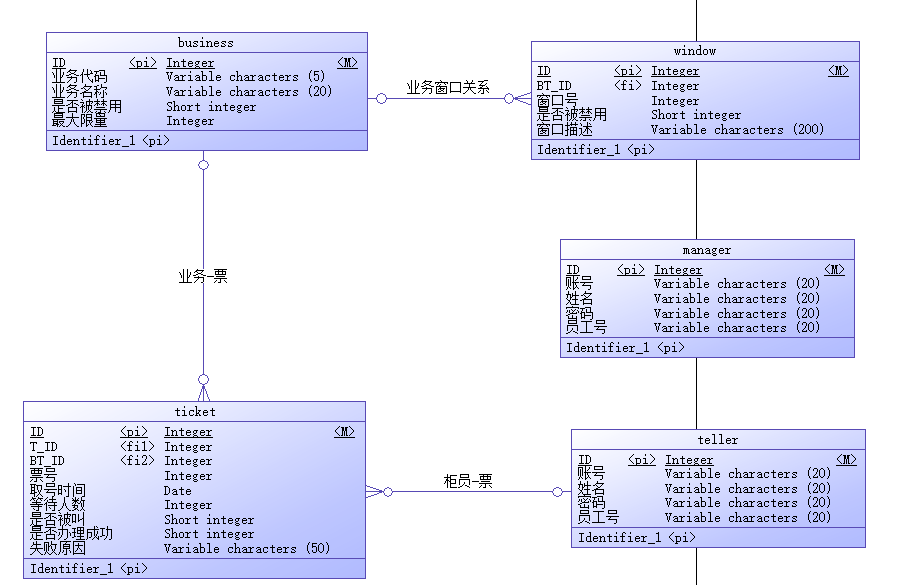


图3.7数据库逻辑结构图

### 3.2.4数据库物理结构设计

1、业务表

**表3.1 业务表字段说明**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 说明 | 描述 |
| BT\_ID | int | 11 | 主键，自动增长 | 自增id |
| BT\_CODE | varchar | 5 | Unique，不可重复 | 业务代号 |
| BT\_NAME | varchar | 16 |  | 业务名称 |
| BT\_DISABLED | tinyint | 1 |  | 是否被禁用 |
| BT\_LIMIT\_COUNT | int | 11 |  | 最大限量 |

2、管理员表

**表3.2 管理员表字段说明**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 说明 | 描述 |
| M\_ID | int | 11 | 主键，自动增长 | 自增id |
| M\_ACCOUNT | varchar | 20 | Unique，不可重复 | 账号 |
| M\_NAME | varchar | 20 |  | 姓名 |
| M\_PASSWORD | varchar | 20 |  | 密码 |
| M\_NUMBER | varchar | 20 |  | 员工号 |

3、柜员表

**表3.3 柜员表字段说明**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 说明 | 描述 |
| T\_ID | int | 11 | 主键，自动增长 | 自增id |
| T\_ACCOUNT | varchar | 20 | Unique，不可重复 | 账号 |
| T\_NAME | varchar | 20 |  | 姓名 |
| T\_PASSWORD | varchar | 20 |  | 密码 |
| T\_NUMBER | varchar | 20 |  | 员工号 |

4、窗口表

**表3.4 窗口表字段说明**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 说明 | 描述 |
| W\_ID | int | 11 | 主键，自动增长 | 自增id |
| W\_NO | int | 11 | Unique，不可重复 | 窗口号 |
| W\_BT\_ID | int | 11 | 外键，关联业务表 | 业务id |
| W\_DISABLED | tinyint | 1 |  | 是否被禁用 |
| W\_DESC | varchar | 200 |  | 窗口描述 |
| W\_USER | varchar | 20 |  | 使用者 |

5、票表

**表3.5 票表字段说明**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 说明 | 描述 |
| TICKET\_ID | int | 11 | 主键，自动增长 | 自增id |
| TICKET\_NO | int | 11 | Unique，不可重复 | 票号 |
| TICKET\_BT\_ID | int | 11 | 外键，关联业务表 | 业务id |
| TICKET\_WINDOW\_NO | Int | 11 |  | 窗口号 |
| TICKET\_TAKE\_DATE | datetime |  |  | 取号时间 |
| TICKET\_WAIT\_COUNT | int | 11 |  | 等待人数 |
| TICKET\_IS\_CALLED | tinyint | 1 |  | 是否被叫 |
| TICKET\_TELLER\_ID | int | 11 | 外键，关联柜员表 | 柜员id |
| TICKET\_IS\_SUCCESS | tinyint | 1 |  | 是否办理成功 |
| TICKET\_FAIL\_REASON | varchar | 20 | 可以为空 | 失败原因 |

### 3.2.5数据库物理结构实现

1、业务表

CREATE TABLE `t\_business\_type` (

`BT\_ID` int(11) NOT NULL AUTO\_INCREMENT,

`BT\_CODE` varchar(5) DEFAULT NULL,

`BT\_NAME` varchar(16) DEFAULT NULL,

`BT\_DISABLED` tinyint(1) DEFAULT '0',

`BT\_LIMIT\_COUNT` int(11) DEFAULT NULL,

PRIMARY KEY (`BT\_ID`),

UNIQUE KEY `BT\_CODE` (`BT\_CODE`)

) ENGINE=InnoDB AUTO\_INCREMENT=9 DEFAULT CHARSET=utf8;

2、管理员表

CREATE TABLE `t\_manage` (

`M\_ID` int(11) NOT NULL AUTO\_INCREMENT,

`ACCOUNT` varchar(20) NOT NULL,

`MNAME` varchar(20) DEFAULT NULL,

`PASSWORD` varchar(20) DEFAULT NULL,

PRIMARY KEY (`M\_ID`),

UNIQUE KEY `ACCOUNT` (`ACCOUNT`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8;

3、柜员表

CREATE TABLE `t\_teller` (

`ID` int(11) NOT NULL AUTO\_INCREMENT,

`TELLER\_NUM` varchar(20) DEFAULT NULL,

`TELLER\_ACCOUNT` varchar(20) DEFAULT NULL,

`TELLER\_NAME` varchar(20) DEFAULT NULL,

`TELLER\_PASSWORD` varchar(20) DEFAULT NULL,

PRIMARY KEY (`ID`),

UNIQUE KEY `TELLER\_ACCOUNT` (`TELLER\_ACCOUNT`),

UNIQUE KEY `TELLER\_NUM` (`TELLER\_NUM`)

) ENGINE=InnoDB AUTO\_INCREMENT=9 DEFAULT CHARSET=utf8;

4、票表

CREATE TABLE `t\_ticket` (

`TICKET\_ID` int(11) NOT NULL AUTO\_INCREMENT,

`TICKET\_NO` int(11) DEFAULT NULL,

`TICKET\_BUSINESS\_CODE` varchar(5) DEFAULT NULL,

`TICKET\_BUSINESS\_NAME` varchar(16) DEFAULT NULL,

`TICKET\_TAKE\_DATE` datetime DEFAULT NULL,

`TICKET\_WAIT\_COUNT` int(11) DEFAULT NULL,

`TICKET\_TAKE\_IP` varchar(20) DEFAULT NULL,

`TICKET\_WINDOW\_NO` int(11) DEFAULT NULL,

`TICKET\_IS\_CALLED` tinyint(1) DEFAULT NULL,

`TICKET\_IS\_SUCCESS` tinyint(1) DEFAULT '2',

`TICKET\_IS\_FAIL` varchar(20) DEFAULT NULL,

`TICKET\_TELLER\_ID` int(11) DEFAULT NULL,

PRIMARY KEY (`TICKET\_ID`),

KEY `FK7F60BD374CF03749` (`TICKET\_TELLER\_ID`),

CONSTRAINT `FK7F60BD374CF03749` FOREIGN KEY (`TICKET\_TELLER\_ID`) REFERENCES `t\_teller` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=87 DEFAULT CHARSET=utf8;

5、窗口表

CREATE TABLE `t\_window` (

`W\_ID` int(11) NOT NULL AUTO\_INCREMENT,

`W\_NO` int(11) DEFAULT NULL,

`W\_BT\_ID` int(11) DEFAULT NULL,

`W\_BUSINESS\_CODE` varchar(5) DEFAULT NULL,

`W\_BUSINESS\_TYPE\_NAME` varchar(20) DEFAULT NULL,

`W\_DISABLED` tinyint(1) DEFAULT '0',

`W\_USER` varchar(20) DEFAULT NULL,

`W\_DESC` varchar(200) DEFAULT NULL,

PRIMARY KEY (`W\_ID`),

UNIQUE KEY `W\_NO` (`W\_NO`),

KEY `W\_BT\_ID` (`W\_BT\_ID`),

CONSTRAINT `W\_BT\_ID` FOREIGN KEY (`W\_BT\_ID`) REFERENCES `t\_business\_type` (`BT\_ID`)

) ENGINE=InnoDB AUTO\_INCREMENT=10 DEFAULT CHARSET=utf8;

## 3.3系统结构设计

### 3.3.1系统体系结构设计

#### 3.3.1.1网络结构模式

1、该系统的的后台管理系统使用B/S结构，即浏览器/服务器模式。它是C/S架构的一种改进，可以说是属于三层C/S架构主要利用不断成熟的浏览器技术，用通用浏览器就实现啦原来需要复杂专用软件才能实现的强大技术，并节约开发成本。

**表3.6 网络结构**

|  |
| --- |
| 第一层：客户端  表现层，用于界面引导，接受用户输入，并向服务器发送请求 |
| 第二层：应用服务器  业务逻辑层：执行业务逻辑，向数据库发送请求 |
| 第三层：数据库服务器  数据存储层：执行数据逻辑，运行SQL或者存储过程 |

第一层是浏览器，即客户端，只有简单的输入和输出功能，几乎不处理事务逻辑。客户不需要安装客户端，在浏览器就能浏览，因此它适合大众化，界面比较简洁好用。

第二层是WEB服务器，主要作用是信息传送。当用户想要访问数据库时，就会首先向WEB服务器发送请求，WEB服务器统一请求后会向数据库服务器发送访问数据库的请求，数据库就会通过SQL语句得到用户想要的数据。

第三层是数据库服务器，它存放着大量的数据。当数据库服务器收到了WEB服务器的请求后，会对SQL语句进行处理，并将返回的结果发送给WEB服务器，接下来，WEB服务器将收到的数据结果转换为HTML文本形式发送给浏览器，也就是我们打开浏览器看到的界面。

管理员访问后台管理系统的数据过程，如图3.8所示。

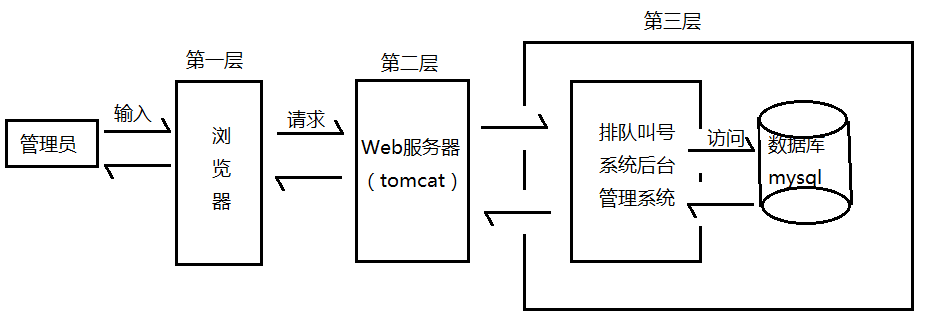


图3.8后台管理系统访问数据过程图

2、该系统的取号叫号系统是采用C/S架构，C/S架构是一种比较早的软件架构，主要应用于局域网内，一般的桌面应用采用的都是C/S架构。C/S架构中客户端和服务器直接相连，而服务器端有两种，一种是数据库服务器端，客户端通过数据库连接访问服务器端的数据；另一种是Socket服务器端，服务器端的程序通过Socket与客户端的程序通信，我们使用的就是Socket服务器端。C/S架构这种点对点的模式使得系统更安全，由于只有一层交互，因此响应速度较快。

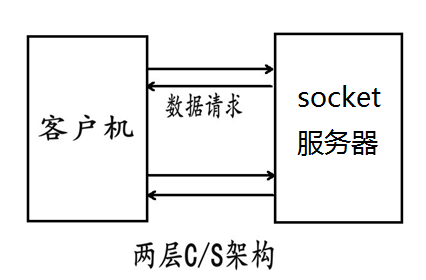


图3.9两层C/S架构图

#### 3.3.1.2网络协议

一般因特网可以分为五层，[应用层](http://baike.baidu.com/subview/239619/16373356.htm" \t "_blank)、[传输层](http://baike.baidu.com/view/239605.htm" \t "_blank)、[网络层](http://baike.baidu.com/subview/239600/16505450.htm" \t "_blank)、[链路层](http://baike.baidu.com/subview/4874044/4874488.htm" \t "_blank)和[物理层](http://baike.baidu.com/subview/239585/239585.htm" \t "_blank)。

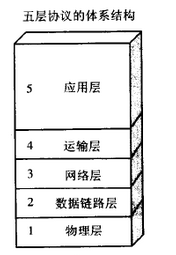


图3.9五层网络协议图

其中传输层负责为信源和信宿提供应用程序进程间的数据传输服务，这一层上主要定义了两个传输协议，传输控制协议即TCP和用户数据报协议UDP，而网络层负责将称为数据报([datagram](http://baike.baidu.com/subview/638347/638347.htm" \t "_blank))的网络层分组从一合主机移动到另一台主机

TCP（Transmission Control Protocol，传输控制协议）是面向连接的协议，也就是说，在收发数据前，必须和对方建立可靠的连接。一个TCP连接必须要经过三次“对话”才能建立起来。

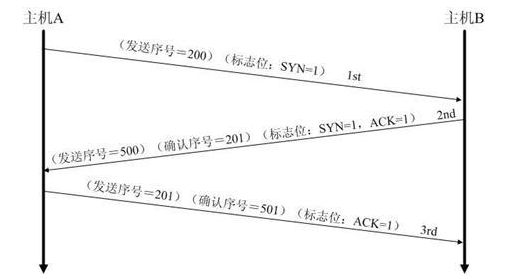


图3.10三次握手过程图

TCP三次握手过程：

1)、主机A通过向主机B 发送一个含有同步序列号的标志位的数据段给主机B ,向主机B 请求建立连接,通过这个数据段,  
主机A告诉主机B 两件事:我想要和你通信;你可以用哪个序列号作为起始数据段来回应我.  
 2)、主机B 收到主机A的请求后,用一个带有确认应答(ACK)和同步序列号(SYN)标志位的数据段响应主机A,也告诉主机A两件事:  
我已经收到你的请求了,你可以传输数据了;你要用哪佧序列号作为起始数据段来回应我  
 3)、主机A收到这个数据段后,再发送一个确认应答,确认已收到主机B 的数据段:"我已收到回复,我现在要开始传输实际数据了  
 这样3次握手就完成了,主机A和主机B 就可以传输数据了。

UDP是一个非连接的协议，传输的开始端和终端不建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。在发送端，UDP传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP把每个消息段放在队列中，应用程序每次从队列中读一个消息段。由于传输数据不建立连接，因此也就不需要维护连接状态，包括收发状态等，因此一台服务机可同时向多个客户机传输相同的消息。

在取号叫号中我们需要服务器的响应，需要可靠安全不丢失数据，所以我们将采用TCP协议。

我们的客户端和服务端是采用socket进行连接的，所以需要采用socket的TCP编程模式来创建客户端和服务端。

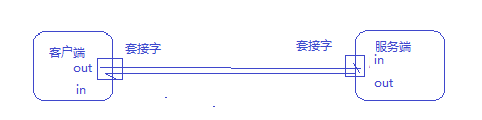


图3.11客户端与服务端Socket连接图

#### 3.3.1.3系统分层设计

系统可以分为四大部分，取号端，叫号端，服务端以及管理端。管理端使用javaWeb实现，以Struts + Spring + Hibernate + Maven + EasyUI 为基本框架，以MVC三层模型为基本模型，实现web页面对数据的访问。取号端和叫号端以及服务器端都是桌面程序，使用javaSwing绘制界面。取号端和叫号端使用socket与服务器进行连接通信，保证速度和安全性。取号端和叫号端只有表现层，业务层和网络层，其中网络层是用来和服务端进行通信的，服务端则比取号端和叫号端多一层持久层，持久层是用来访问数据库的。为了保证数据的安全，取号端和叫号端不允许非直接访问数据库，需要发送指令给服务器，服务器通过相关指令将相应的数据返回给叫号端和服务端

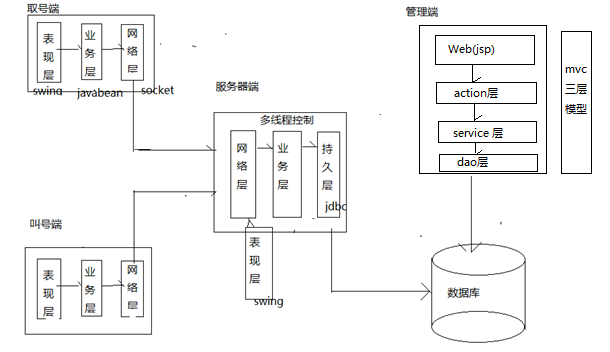


图3.11系统分层结构图

系统的四大部分都有着必然的联系，就如上描述，我们可以得到如图3.12所示的包图。

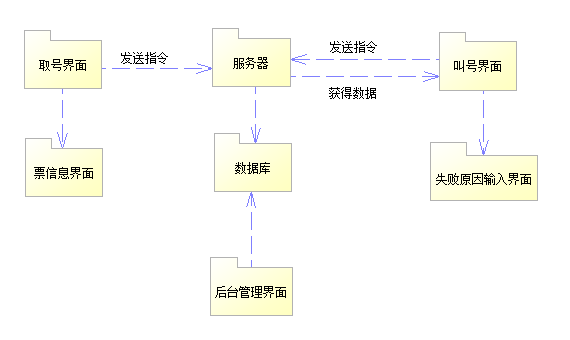


图3.11各个部分依赖关系包图

### 3.3.2系统功能模块设计

通过需求分析可以了解到，银行排队叫号系统业务办理的基本功能包括取票功能，叫号功能，后台管理与统计功能，其划分功能的功能模块结构如表3.7所示。

**表3.7 系统功能模块表**

|  |  |
| --- | --- |
| 功能类别 | 子功能 |
| 取票功能 | 选择业务 |
| 选择会员类型 |
| 叫号功能 | 叫号 |
| 重复叫号 |
| 叫下一个号 |
| 办理成功 |
| 办理失败 |
| 后台管理 | 管理员登录 |
| 柜员信息管理 |
| 业务管理 |
| 窗口管理 |
| 历史记录查询 |
| 图形统计 |

#### 3.3.2.1取票功能

取票功能包含两个子功能，选择业务和选择会员类型

1、选择业务。真实的排队叫号系统是客户在营业厅的取票机上通过触摸屏幕选择所要办理的业务，而我模拟这一情况，在桌面系统上通过鼠标点击按钮来选择所要办理的业务。一般银行的业务类型可以划分为存取款、汇款转账、缴费、理财业务、挂失换卡、外币、开户销户、对公现金等多种业务，我们可以结合自己的需求来选择业务类型。页面上的业务列表是通过数据库中的业务表的数据查找出来的，然后显示在页面上。每一个业务选项有绑定一个事件，当你选择业务点击去确定时，就会向服务端发送指令和数据，然后获得服务端返回的数据。

2、选择会员类型。一般银行会给vip用户留有专用窗口或者专用通道，如果你是银行的vip会员，你可以选择vip会员通道，会有专业人士为你服务，从而省去了大量的排队时间。

当客户在取票机上选择业务取票时，取票界面将会把取票的指令和信息发给服务器，服务器解析取票端发来的指令，然后对数据库进行操作，并返回票信息给取号界面，打印票信息给客户

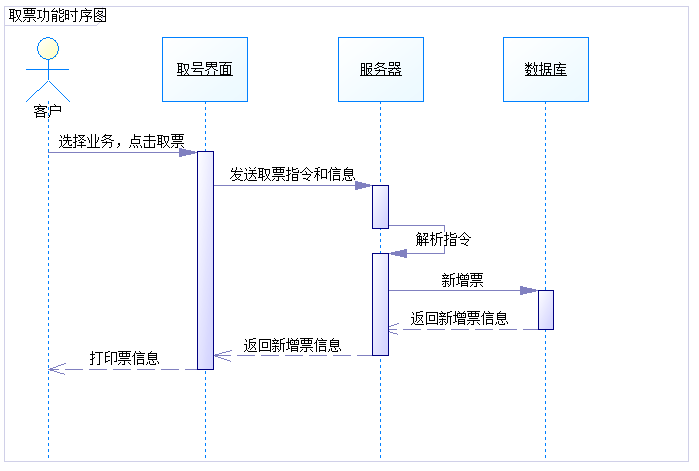


图3.12取号功能时序图

#### 3.3.2.2叫号功能

银行的柜员可以通过登录叫号系统，来确认办理完成一笔业务，并叫下一个业务号码。叫号功能分为叫号，重复叫号，叫下一个号，办理成功，办理失败等功能，当柜员叫号时，没人前来窗口办理业务，可选择重复叫号，如果还是没人前来，可以选择叫下一个号，如果业务业务办理失败，需要填写失败原因，如果成功，则点击办理成功，系统会自动叫下一个号。

在后台，数据需要通过叫号系统，服务器和数据的相互连接才能传递，当柜员登陆系统时，登陆界面会向服务器发送数据和查询指令，然后服务器区数据库查找，并返回查询结果，判断柜员是否登陆成功。同样，当柜员登录到叫号系统叫号时，数据和指令也会被传到服务器，通过服务器来与数据库来进行交流。具体过程如3.13所示。

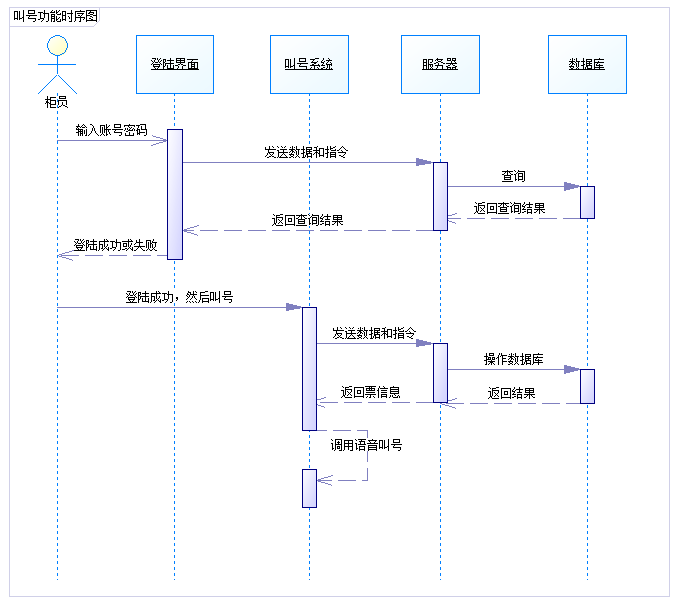


图3.13叫号功能时序图

#### 3.3.2.3后台管理

1、管理员登录

管理员可以通过有管理权限的账号登陆后台管理系统，然后对业务、窗口、柜员、历史记录等进行管理，并且查看统计图形。

2、柜员信息管理

登录后台管理系统的管理员可以对柜员的账号信息进行修改、删除和查询。可以方便管理柜员的账号，方便柜员的人事替换。当新来一个柜员或者离开一个柜员时，管理员可以方便的增加和删除一个柜员登录叫号系统的账号。

3、业务管理

管理人员可以通过后台管理进行业务类型的管理。可以添加、禁用、修改、删除一种业务类型，为某种业务类型可以设置每天最大办理量。一般银行都会有固定的几个业务，当需要增加新的业务或者废弃某一个旧的业务时，可以通过业务管理来方便的操作。

4、窗口管理

窗口管理适合业务管理息息相关的，当添加一个新的窗口时，要绑定一个该窗口所要处理的业务，当该业务被删除时，该窗口所对应的业务也要改为空。

5、历史记录查询

管理员可以点开历史纪录查询，去搜索某一段时间里的业务办理的历史纪录，通过历史里记录可以了解过去某一段时间里业务的处理情况。

6、图形统计

图形统计的功能主要是通过统计过去的业务数量，业务的成功、失败的比率来反映某一段时间里的市场对业务的需求量、业务的热度和办理的难易程度等等，管理员可以通过查看这些统计图形来帮助写分析报告。

后台管理系统是一个web系统，我们通过网页上的一些操作，然后向后台发送请求，后台接收到请求之后通过向对应的逻辑去操作数据库，并返回相应的结果给页面，有页面来展示给管理员看。具体过程如图3.14。

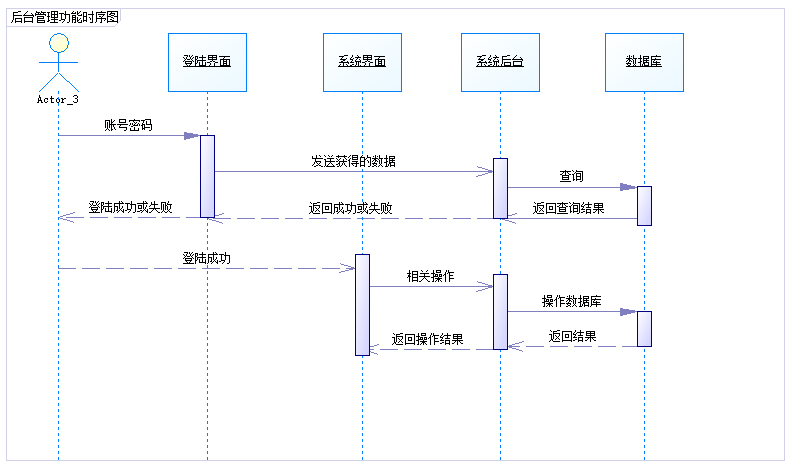


图3.14后台管理功能时序图

# 系统实现

## 4.1取号端实现

取号端一共有三个自己的包(net、service、view),以及一个共用的enrity包。其四个包之间关系如图4.1所示

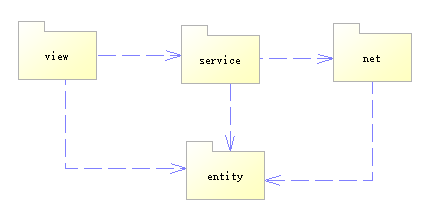


图4.1取号端包图

取号端的界面如图4.2所示。



图4.2取号端界面

当你选择其中一个业务取号时，该按钮绑定的事件将会将目前的时间和业务信息传到service层，在service层将信息封装在指令消息类当中，然后通过net层将指令消息发给服务端，并接受服务端返回的消息，依次传到service层和view层，然后由获得的消息打印出票号的信息。票号的显示界面如图4.2所示。



图4.3票号信息界面

如上过程的详细实现如下：

1. 消息指令类的封装
2. Net层实现与服务端的通信
3. 展示所有业务和打印票号信息。

### 4.1.1消息指令类的封装

消息指令分为两种，一种是客户端发送给服务端的，一种是服务端返回给客户端的，都是用来存放指令信息和数据的。

客户端发送给服务端指令信息主要有action、subAction和client三个属性，分别表示动作和子动作和客户端，例如动作是获取业务类型，客户端为取票端。数据主要有ip和data，分别表示客户端IP和具体的数据。

服务端返回给客户端的指令信息有success属性，表示是否成功，数据有msg和data两个属性，表示处理的结果信息和返回的数据。

public class ActionMessage<T> implements Serializable{

private static final long serialVersionUID = -2296930195311270937L;

/\*\*动作\*/

private String action;

/\*\*子动作\*/

private String subAction;

/\*\*客户端\*/

private String client;

/\*\*客户端ip\*/

private String ip;

/\*\*具体的数据\*/

private T data;

}//省略get()/set()方法

public class HandleMessage<T> implements Serializable{

private static final long serialVersionUID = -6881641318143937323L;

/\*\*是否成功\*/

private boolean success;

/\*\*处理的结果信息\*/

private String msg;

/\*\*处理成功返回的数据\*/

private T data;

}//省略get()/set()方法

### 4.1.2 Net层实现与服务端的通信

1、连接服务器

将服务器的IP和端口放在server.properties文件里面，通过文件流去获得，然后通过socket和服务端连接。

Config cfg = new Config("serverConfig/server.properties");

String ip = cfg.get("server.ip");

int port = Integer.parseInt(cfg.get("server.port"));

Socket socket=new Socket(ip,port);

2、发送消息给服务器

通过流的形式将消息指令发给服务器

try {

ObjectOutputStream out = new ObjectOutputStream( socket.getOutputStream() );

out.writeObject(message);

out.flush();

return true;

} catch (Exception e) {

e.printStackTrace();

}

3、接收服务器的消息

通过流的形式接受服务器返回消息指令

try {

ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

Object o = in.readObject();

if(o instanceof HandleMessage){

return (HandleMessage)o;

}

} catch (Exception e) {

e.printStackTrace();

}

### 4.1.3展示所有业务和打印票号信息

通过socket的通信，获得了服务端返回的信息，其中包含业务信息和票号信息，将这些信息返回到view层，通过swing将其展现出来即可。

## 4.2叫号端实现

和取号端类似，一共有三个自己的包(net、service、view),以及一个共用的enrity包。其四个包之间关系如图4.1所示。当柜员叫号的时候需要先登录到叫号系统，然后在进行叫号。登陆的时候需要选择窗口号，窗口号是与业务类型相对应的，登陆之后就办理该业务的票号。登录界面如图4.4所示。

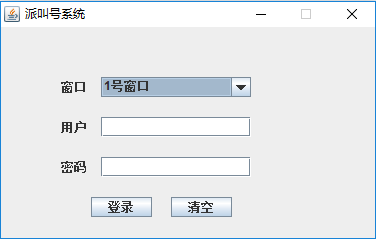


图4.4柜员登陆界面

成功登陆之后，来到叫号界面，如图4.5所示。我们可以点击叫下一个号，重复叫号，暂停服务，办理完成之后可以按办理完成，失败的话填写失败原因。

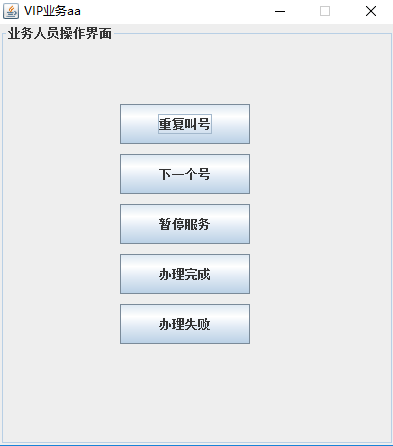


图4.5柜员登陆界面

### 4.2.1 net和service层的实现

叫号端和取号端的消息指令类是同一个，这里就不详细说明了。网络层的实现也和取号端类似，主要有连接服务器，发送消息给服务器以及从服务器接收返回的消息等方法。

Service层主要有8个方法，主要是处理登陆的方法、叫下一个号、重叫、暂停服务、办理完成、办理失败、获得窗口列表、退出系统等方法。八个方法的主要实现相似，只是对于数据的处理有区别，这里详细说明叫号方法的实现。

这里我们必须要为指令消息类ActionMessage设定客户端和动作，因为在服务端那边需要解析这些来执行相应的逻辑。同时，数据也需要通过指令消息类传给服务端。HandleMessage是服务端返回的数据，我们需要对它进行处理并传给view层。

public Ticket CallTicket(String businessCode, int windowNo,String account)

throws Exception {

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

ActionMessage am = new ActionMessage();

//设置客户端为CALL\_TICKET\_CLIENT

am.setClient(Constants.CALL\_TICKET\_CLIENT);

//设置动作为CALL\_TICKET\_ACTION

am.setAction(Constants.CALL\_TICKET\_ACTION);

Ticket t = new Ticket();

t.setBusinessCode(businessCode);

t.setWindowNo(windowNo);

t.setAccount(account);

am.setData(t);

// 发送动作指令

callTicketNet.send(am);

// 接收处理结果

HandleMessage hm = callTicketNet.receive();

if (hm.isSuccess()) {

// 返回取得的号

return (Ticket) hm.getData();

} else {

throw new Exception("叫号失败:"+hm.getMsg());

}

}

其中Constants是一个常量类，主要定义一些常用的常量。

### 4.2.2 叫号的语音实现

叫号的语音实现实在服务端实现的，在服务端通过叫号端发送过去的消息指令，调用实现语音的方法。详细见服务端设计。

## 4.3服务端实现

服务端和取号端、叫号端一样拥有三个自己的包(net、service、view),以及一个共用的enrity包，另外还有一个dao包，这里是与数据库相连的持久层，无论是叫号端还是取号端，都要来到这里才能通过服务器操作数据库。其五个包之间关系如图4.6所示。

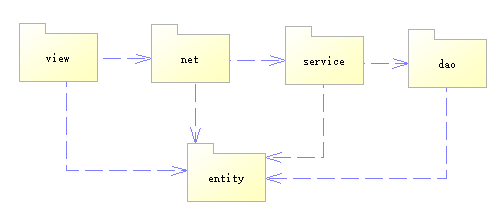


图4.6服务端包图

服务端的界面如图4.7所示。



图4.7服务端界面

### 4.3.1 net层实现

与叫号端还有取号端一样，服务器的net层也有发送指令信息和接收指令信息两个方法，以下就不详细说明。

服务器需要接收不同客户端的连接请求，为了保证数据安全，不使数据在被同时访问时发生冲突，我们需要给服务端接收和发送信息构建一个线程。

/\*\*

\* 该类需要继承Thread

\* 重写Runnable的方法，目的是为了构建线程

\* \*/

public void run() {

try {

while (true) {

// 接收客户端发来的指令消息

ActionMessage<Object> am = receive();

// 根据指令动作获得相应的处理器

IHandleAction handleAction = HandleActionFactory.getHandleAction(

am.getClient(), am.getAction());

// 处理客户端的动作指令，并得到处理结果

HandleMessage hm = handleAction.handle(am);

// 发送处理结果客户端

send(hm);

}

} catch (NetException e) {

e.printStackTrace();

} finally {

if (null != socket) {

try {

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

由于服务端需要与客户端进行通信，我们在通信前需要先启动服务器，等待客户端的连接。

public void startServer() throws StartException {

try {

//服务器启动的标志

isRun = true;

DisplayMessage.displayActionMessage("服务器启动成功，在端口" + port + "监听...");

while (isRun) {

// 监听接收客户端的连接

Socket s = serverSocket.accept();

// 每当一个客户端连接，就开一个线程为它服务

new ServerHandleThread(s).start();

}

} catch (Exception e) {

e.printStackTrace();

throw new StartException("服务器出现问题", e);

} finally {

if (null != serverSocket) {

try {

serverSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

### 4.3.2 service层实现

在服务端的service层，我们需要做很多事情，例如我们上一节提到的调用声音，解析客户端发来的指令，对将要返回给客户端的数据进行处理等等。这里介绍几个比较核心的也比较主要的实现。

#### 4.3.2.1解析指令实现

在4.3.1节中，我们看到一个类HandleActionFactory，这个类就是用来解析客户端发来的指令的一个类。这里就是根据封装在消息指令当中的客户端和动作两个属性，来判断该消息指令需要执行哪个实现类。

public class HandleActionFactory {

/\*\*

\* 根据客户端和动作不同返回不同的实现类对象

\* @param client

\* @param action

\* @return

\*/

public static IHandleAction getHandleAction(String client, String action) {

if (Constants.TAKE\_TICKET\_CLIENT.equals(client)

&& Constants.TAKE\_TICKET\_ACTION.equals(action)) {

return new TakeTicketActionImpl();

} else if (Constants.CALL\_TICKET\_CLIENT.equals(client)

&& Constants.CALL\_TICKET\_ACTION.equals(action)) {

return new CallTicketHandleImpl();

} else if (Constants.CALL\_TICKET\_CLIENT.equals(client)

&& Constants.STOP\_SERVE\_ACTION.equals(action)) {

return new StopServeActionImpl();

} else if (Constants.CALL\_TICKET\_CLIENT.equals(client)

&& Constants.LOGIN\_ACTION.equals(action)) {

return new LoginActionImpl();

} else if (Constants.CALL\_TICKET\_CLIENT.equals(client)

&& Constants.FINISH\_SERVE\_ACTION.equals(action)) {

return new FinishServiceImpl();

} else if (Constants.CALL\_TICKET\_CLIENT.equals(client)

&& Constants.AGAINCALL\_TICKET\_ACTION.equals(action)) {

return new CallAgainActionImpl();

} else if (Constants.CALL\_TICKET\_CLIENT.equals(client)

&& Constants.FAILCALL\_TICKET\_ACTION.equals(action)) {

return new FailCallActionImpl();

} else if (Constants.TAKE\_TICKET.equals(client)

&& "getBusinessTypes".equals(action)) {

return new GetBusinessTypesActionImpl();

} else if (Constants.TAKE\_TICKET.equals(client)

&& "takeTicket".equals(action)) {

return new TakeTicketActionImpl();

}else if (Constants.CALL\_TICKET\_CLIENT.equals(client)

&& Constants.CALL\_LIST\_ALL\_WINDOWS.equals(action)) {

return new CallListAllWindowsActionImpl();

}else if (Constants.CALL\_TICKET\_CLIENT.equals(client)

&& Constants.CALL\_EXITE.equals(action)) {

return new ExitSystemActionImpl();

} else {

return null;

}

}

}

例如，如果客户端属性为 ” TAKE\_TICKET\_CLIENT ”，动作属性为 “TAKE\_TICKET\_ACTION ”，那么我们将执行的是new TakeTicketActionImpl()。那么TakeTicketActionImpl类中有什么呢？

/\*\*

\* 处理动作指令

\* @param am - 动作指令

\* @return 处理结果

\*/

public HandleMessage handle(ActionMessage am) {

HandleMessage hm = new HandleMessage();

String businessCode = am.getData().toString();

String clientIp = am.getIp().toString();

Ticket t;

try {

//调用业务逻辑类取号

t = takeTicketService.takeTicket(businessCode,clientIp);

hm.setSuccess(true);//处理成功

hm.setData(t);//处理成功后的数据

return hm;

} catch (Exception e) {

e.printStackTrace();

hm.setSuccess(false);//取号失败

hm.setMsg(e.getMessage());//取号失败的原因

return hm;

}

}

该类中有一个处理动作指令的方法，它处理的是取号这个动作指令，我们可以看到在这里我们获得客户端传过来的数据，并将数据传到取号的业务中去，并且将返回的结果封装到HandleMessage当中，传给客户端。取票的业务逻辑一共有三步：首先查询该业务代码当天的办理量是否达到最大办理量，如果没有，就在最大号基础上加1，最后根据信息保存该票号。具体实现比较简单，就不详细说明了。

Service层的其他类的实现比较类似，只是业务逻辑不同而已，这里就不加以说明了。

#### 4.3.2.2叫号的语音实现

这里的语音是使用java中的AudioClip接口。使用该接口的play()方法来播放语音。首先，我们的语音模型是：请××号到××窗口办理业务，我们需要分别把每一个字的读音截取下来，放到一个文件里面，如图4.8所示

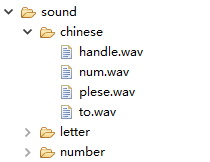


图4.8声音文件

其中chinese包中存放中文字的读音，letter包中存放英文字的读音，number包中存放的是数字的读音。

首先我们写一个处理声音的类，里面有一个播放声音的方法。

public class InvokingSound {

public static void invokSound(String Filename) {

URL url;

try {

url = new URL("file:" + Filename);

AudioClip ac = Applet.newAudioClip(url);

ac.play();

} catch (MalformedURLException e) {

e.printStackTrace();

}

}

}

然后我们就可以写一个工具类，调用播放声音的方法，区实现我们的声音模型。

public static void addCallingSound(String businessCode, int number,int windowNo) {

try {

// 调用声音“请”

InvokingSound.invokSound("sound/chinese/plese.wav");

Thread.sleep(500);

// 调用声音 叫号业务类型，如 A,B,C等

System.out.println(judgeBusinessType(businessCode));

InvokingSound.invokSound(judgeBusinessType(businessCode));

Thread.sleep(500);

// 调用声音 排队号数， 比如 14 相应的声音

invokSoundOfNumber(number);

// 调用声音 “号"

InvokingSound.invokSound("sound/chinese/num.wav");

Thread.sleep(500);

//调用声音“到”

InvokingSound.invokSound("sound/chinese/to.wav");

Thread.sleep(500);

// 调用声音 窗口号数 如 1，2 等

InvokingSound.invokSound(judgeNo(windowNo));

Thread.sleep(500);

// 调用声音“到窗口办理”

InvokingSound.invokSound("sound/chinese/handle.wav");

Thread.sleep(2500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

在这里我们需要对英文字母和数字做特殊处理。对英文的处理比较简单。

public static String judgeBusinessType(String businessCode) {

String url = "sound/letter/"+businessCode.toLowerCase()+".wav";

return url;

}

接下来是对数字进行处理。

public static void invokSoundOfNumber(int number){

int thousandsDigit = number / 1000;

int hundredsDigit = number % 1000 / 100;

int tensDigit = number % 100 / 10;

int unitsDigit = number % 10 / 1;

try {

if(thousandsDigit == 0){

}else{

InvokingSound.invokSound(judgeNo(thousandsDigit));

Thread.sleep(500);

}

if(thousandsDigit == 0 && hundredsDigit == 0){

}else{

InvokingSound.invokSound(judgeNo(hundredsDigit));

Thread.sleep(500);

}

if(thousandsDigit == 0 && hundredsDigit == 0 && tensDigit == 0){

}else{

InvokingSound.invokSound(judgeNo(tensDigit));

Thread.sleep(500);

}

InvokingSound.invokSound(judgeNo(unitsDigit));

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public static String judgeNo(int No) {

String url = null;

if (0 == No) {

url = "sound/number/0.wav";

}

if (1 == No) {

url = "sound/number/1.wav";

}

……………………………………//2-9省略

return url;

}

### 4.3.3 dao层实现

服务器的dao层主要是对数据库进行操作，这里使用的是JDBC。在dao层里面，主要有两个类，一个是叫号端发送过来的指令消息操作数据库的类，一个是取号端发送过来的指令消息操作数据库的类。其结构如图4.9所示。

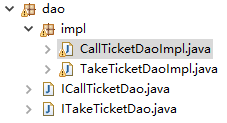


图4.9 dao层结构

dao层实现的主要主要连接数据库，根据逻辑操作数据库，这里就不详细介绍了。

## 4.4后台管理端实现

后台管理系统主要是使用Struts + Spring + Hibernate + Maven + EasyUI的框架实现的。主要的功能模块有登陆，业务管理，柜员管理，窗口管理，数据分析，办理情况等功能。如图4.10所示。



图4.10 后台管理功能模块

### 4.4.1 登陆功能实现

后台管理的登陆只能是管理员才能登陆，持有管理员账号和密码的可以登陆系统，管理后台数据。登陆界面如图4.11所示



图4.11 登陆界面

输入账号密码之后，通过后台匹配数据库，成功则登陆到系统，并且会在右下角提示成功，失败则提示登录失败。如图4.12所示。

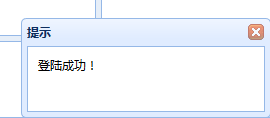


图4.12 登陆提示界面

### 4.4.2 业务管理功能实现

业务管理主要有展示数据，增删改查等功能。如图4.13所示



图4.13 业务管理界面

点击增加按钮，会弹出一个增加框，提供填写增加数据。如图4.14所示。这里单个输入框都提供了校验，业务代码和业务名称两行不能为空，业务限制量必选填写0~100之间的整数。



图4.14 增加业务界面

点击删除前，需要选择要删除的项，如果没有选择，会提示请勾选要删除的记录。如图4.15所示

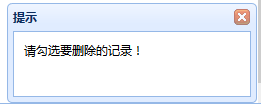


图4.15 删除勾选提示界面

选择要删除的相之后，会请求确认是否删除，如图4.16所示。

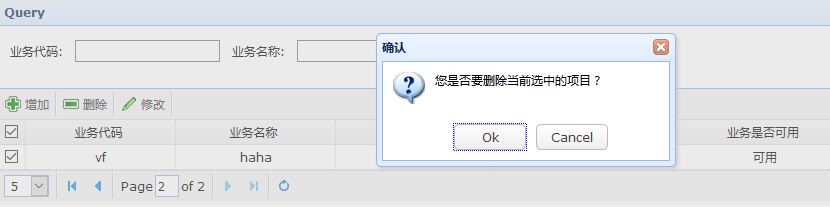


图4.16 删除确认提示界面

点击OK，就会删除这一行或者多行内容。点击取消，则不会删除内容。

回到业务管理页面，点击修改时，若没有选择要修改的项，也会提示请选择要修改的记录。如图4.17所示。

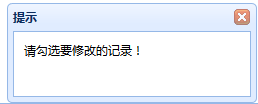


图4.17 修改勾选提示界面

选择要修改的项之后，和删除数据一样，还要确定修改，点击ok，就会弹出修改框。如图4.18所示。弹出框会自动加载选择行的数据，显示在弹出框中，修改完后，点击修改，提示修改成功，就修改完成了。如图4.19所示。



图4.18 修改界面

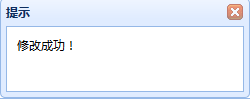


图4.19 修改成功提示界面

回到业务管理界面，可以根据业务代码和业务名称进行查询，点击清空会清空查询内容并重新加载数据。查询结果如图4.20所示。



图4.20 查询结果

### 4.4.3 柜员管理功能实现

柜员管理和业务管理一样，拥有查询，增删改查的功能，如图4.21所示。

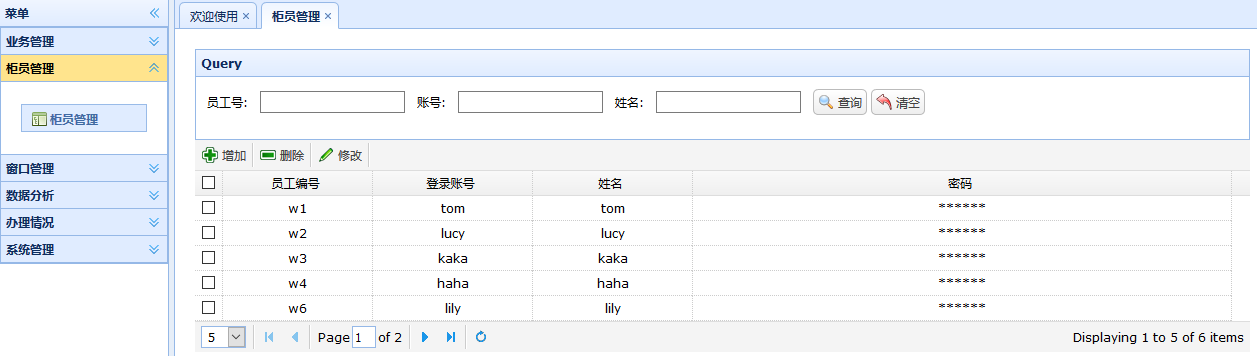


图4.21柜员管理界面

柜员管理和业务管理的增删改的弹出框和提示功能相似，就不一一列出来了，

根据输入条件的查询功能也相似，查询结果如图4.2.2所示。

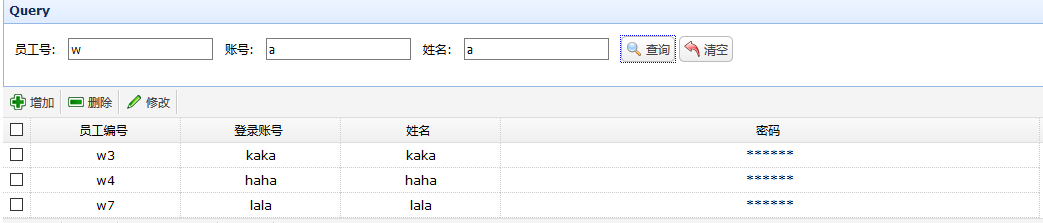


图4.22柜员查询结果界面

### 4.4.4 窗口管理功能实现

窗口管理主要是管理窗口和业务还有柜员之间的关系，如图4.23所示。

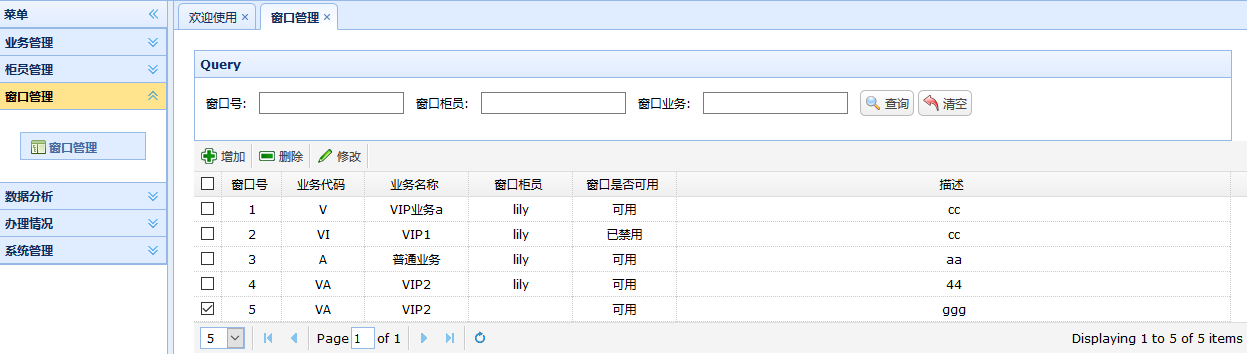


图4.23窗口管理界面

查询功能和增删改的功能以及提示很相似，就不一一列出来了。有一个不同的地方就是在增加和修改的弹出框中，业务代码这一行是使用combobox,通过ajax异步请求查询到所有业务代码，展现出来的。如图4.24所示。



图4.24窗口增加界面

### 4.4.5 数据分析功能实现

数据分析主要有三个模块，成功率和失败率统计，比较各个业务的需求量，以及业务受季度的影响。

#### 4.4.5.1成功率与失败率统计

在考虑成功率失败率的同时，还要考虑另一个因素，就是有时候会存在这样的情况，叫了号没人前来窗口办理，这样的票号我们称为流失的票号，所以在成功率和失败率以外还有一个流失率。如图4.25所示。

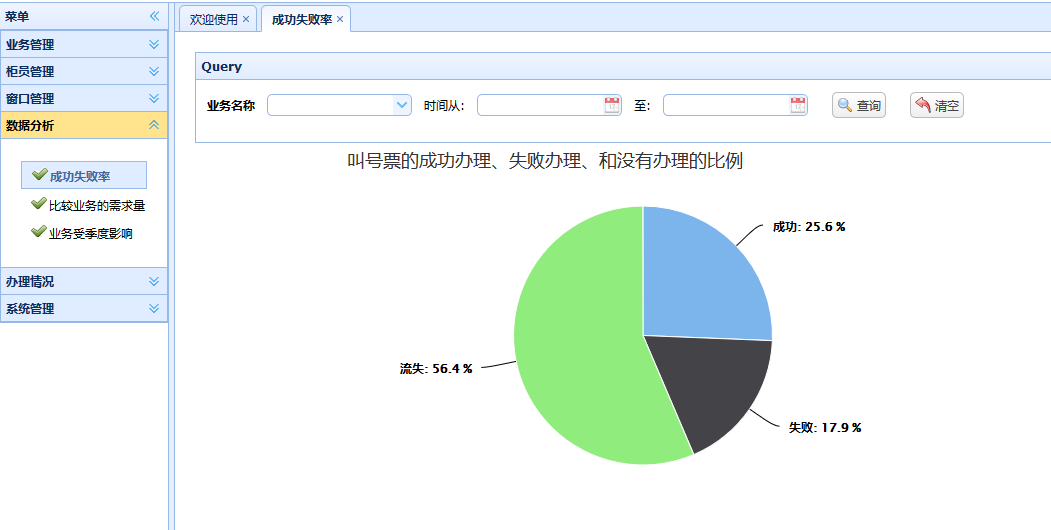


图4.25成功与失败率统计界面

同时，我们可通过我们输入的条件去查找某一个业务，在某一段时间内的成功率、失败率与流失率。

这里的饼图使用highchart实现的，初始化饼图使用的是highchart的API提供的highchart方法。

function initHighchart(data){

$('#container').highcharts({

credits:{

enabled:false // 禁用版权信息

},

chart: {

marginLeft : 200,

height : 350,

width : 700,

plotBackgroundColor: null,

plotBorderWidth: null,

plotShadow: false

},

title: {

text: '叫号票的成功办理、失败办理、和没有办理的比例'

},

tooltip: {

pointFormat: '{series.name}: <b>{point.percentage:.1f}%</b>'

},

plotOptions: {

pie: {

allowPointSelect: true,

cursor: 'pointer',

dataLabels: {

enabled: true,

color: '#000000',

connectorColor: '#000000',

format: '<b>{point.name}</b>: {point.percentage:.1f} %'

}

}

},

series: [{

type: 'pie',

name: 'Browser share',

data: data

}]

});

}

初始化饼图时的data参数是通过ajax异步请求从后台获取的。从后台获取的数据是一个由List转换过来的json数据。其中包括根据条件查询出来的成功的总数，失败的总数，和流失票的总数。

#### 4.4.5.2比较各个业务的需求量

在这里展示的每一个业务在某一段时间内办理的数量，如果没有输入查询条件，则默认为是查询所有历史业务。如图4.26所示



图4.26比较业务的需求量界面

这里的柱状图也是使用highchart实现的，初始化柱状图使用的是highchart的API提供的highchart()方法。实现的方法与饼图类似，参照API实现比较简单，这里就不详细说明了。

初始化柱状图时也需要一个data参数，该参数也是通过ajax异步请求从后台获取的。从后台获取的数据是一个由List转换过来的json数据。其中包括X轴的业务名，以及各个业务的办理数量。

#### 4.4.5.3业务受季度的影响

这里展示每一个月份业务的办理数量，体现了了业务受季度的影响。我们可在字啊啦框中选择某一个业务，查看该业务受季度的影响，如果没有选择业务，则展示的都是所有业务的数据。如图4.27所示，其中蓝线代表总数，黑线代表成功数，绿线代表失败是，黄线代表流失的数量。

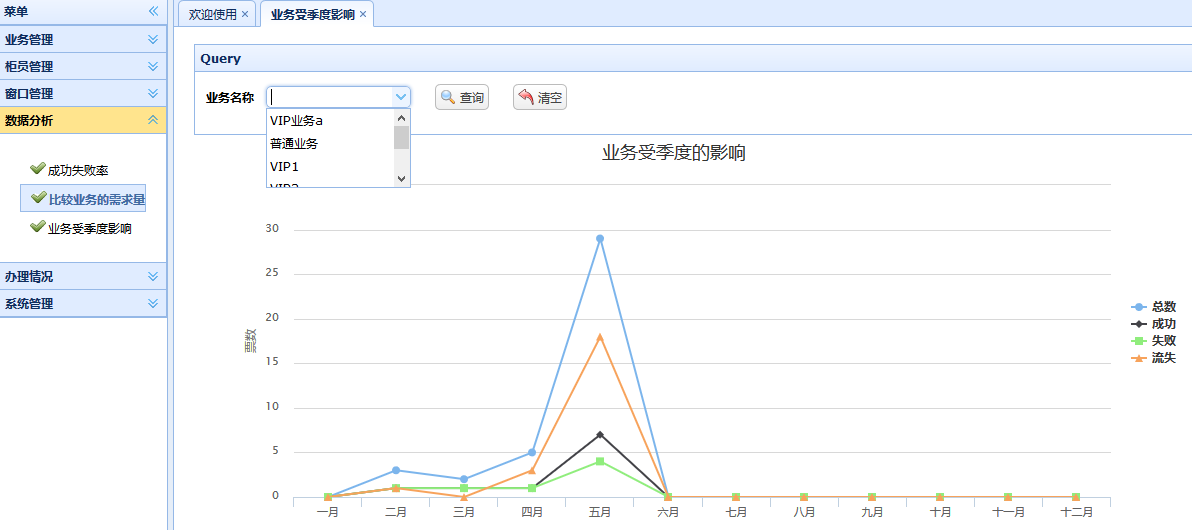


图4.27业务受季度影响界面

这里的折线图也是使用highchart实现的，初始化折线图使用的是highchart的API提供的highchart()方法。实现的方法与上面提到的柱状图和饼图相似，参照API实现比较简单，这里就不详细说明了。

初始化折线图时也需要一个data参数，该参数也是通过ajax异步请求从后台获取的。从后台获取的数据是一个由List转换过来的json数据。其中包括每个月份的业务总数，成功数，失败数和流失数。

### 4.4.6 办理情况功能实现

办理情况模块下，有两个子模块，分别是展示历史记录，和当前办理情况。历史记录是指之前办理的票的情况，当前办理情况是指当天的办理情况。

#### 4.4.6.1历史记录

历史记录只提供查询功能，这里不允许对历史记录进行修改和添加，我们可以通过业务名，窗口号，柜员账号，和时间进行查询，查看符合条件的历史记录的详细信息。如图4.28所示

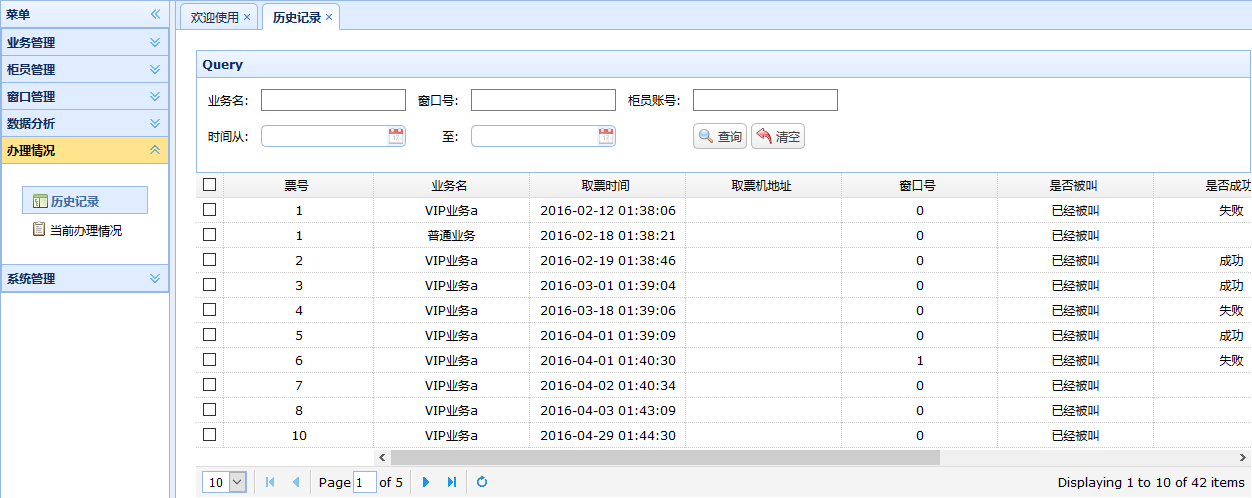


图4.28历史记录界面

#### 4.4.6.1当前办理情况

当前办理情况是通过柱状图显示的当天已办理的量，未办理的量，成功数，失败数，和流失的票数。可以通过业务名称和窗口号查询某一个业务的办理情况。如图4.29所示，这是没有数据时的情况，当有数据时，如图4.30所示。同时，还给加载数据设置了定时器，每过一分钟，就会自动加载一次数据，并刷新柱状图。

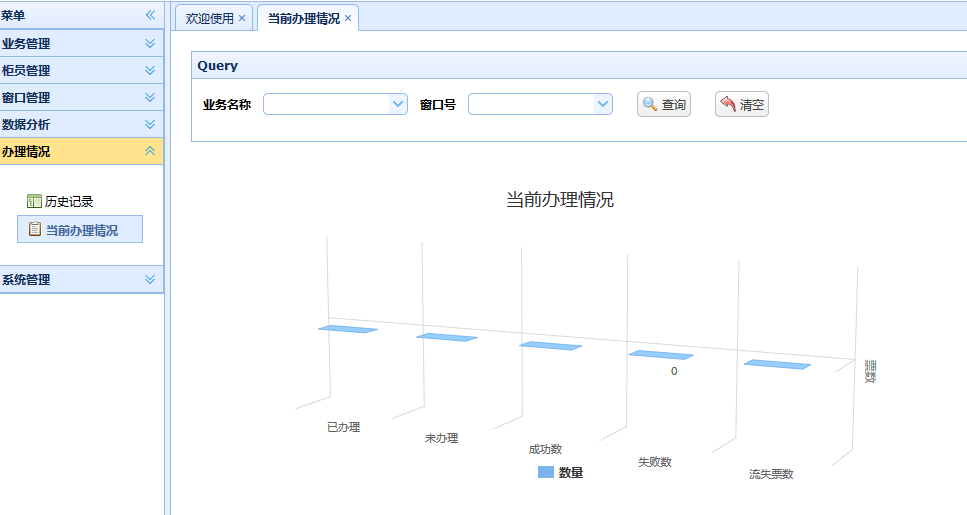


图4.29当前办理情况界面



图4.30当前办理情况界面

# 总结与展望

## 5.1总结

本论文系统与银行的工作紧密相连。在系统的开发当中，通过对于实际流程的观察，制订了系统的需求分析和功能模块。实际的银行排队叫号系统可以为银行的管理者提供详实的动态数据，方便管理者对员工进行工作分配。在本系统中，也做到了这点，不仅实现排队与叫号功能，还通过图形的展示来帮助管理员分析数据。

在实际开发当中遇到了许多问题，基本通过网上查找和与同学交流解决了。在解决问题的过程当中，学会了解决问题的方法，练就了一个细心查找问题的所在的耐心。

在整个论文和系统开发过程当中，主要的工作结论如下。

1、从需求分析，系统设计和系统实现等方面，基本理解银行排队叫号系统的研发一样，明确了系统设计的思想和过程。

2、结合银行排队叫号的流程，通过自己的需求分析，提出系统的构建思路，确定了各个功能模块，并实现了取号与叫号的功能实现。

3、结合银行的当前业务的流程和状况，收集各种业务的数据，完成后台数据管理和报表分析，有效的提升了分析数据的效率。

4、鉴于时间和技术的问题，系统还有部分功能没有达到预期的目的，还有待改善。

## 5.2展望

目前本系统只能在机器上进行取号，也就是说只能通过点对点与服务器进行连接。随着互联网的的不断发展，随着银行业务 的持续增长，需要通过更有效的方式提高银行的办事效率。在后续的开发中，将会考虑将网上预约、电话预约，多网点客户自动分配等功能加上去，进一步简化业务流程，提高办事效率，减少客户等待时间。

**致谢**

在论文和系统的开发当中，感谢滕少华老师，悉心指导和按时督促。从论文的选题、研究、设计直至完成无不凝聚着导师的心血。导师在百忙之中仍然定期听取汇报，了解课题的进展情况，把握课题的研究方向，并提出了许多重要的建设性意见。导师崇高的品德、渊博的知识、严谨的治学态度及忘我的工作精神，激励我永远奋发向上，使我终生受益。滕少华老师在我的课题研究过程中给予了极大的帮助，在此向他表示衷心的感谢。

同时也要感谢我的同学，在我课题研究的过程中遇到了各种问题向同学请教时，同学们都欣然的伸出援助之手，耐心回答我的问题并细心讲解。在课题的研究过程中，我和同一课题的同学也进行讨论并互相学习，共同进步。

最后向所有关心、鼓励、支持和帮助过我的人表示深深的谢意。