本文的设计是基于C/S结构，利用hibernate持久化框架、使用JAVA编程技术在Eclipse集成开发环境下开发的航空订票系统。该系统实现了航班管理、订票管理、营业网点管理以及业务统计等功能。

系统在客户端内能够完成营业员登录，查询航班，出票等功能。在服务器航空公司内部能够完成添加或删除航班，添加或删除营业网点等功能。系统还能统计业务量，并将这些信息保存在服务器端，并供业务员随时查询。

关键词**：** JAVA hibernate 订票系统

目 录

[1 引言 1](#_Toc196908288)

[1.1 民航发展概况 1](#_Toc196908289)

[1.2 技术选择 1](#_Toc196908290)

[1.2.1 CS结构 1](#_Toc196908291)

[1.2.2 Java技术 2](#_Toc196908292)

[1.3 开发工具 2](#_Toc196908293)

[2 需求分析 3](#_Toc196908294)

[2.1软件分层与部署 3](#_Toc196908295)

[2.2 系统功能模块 3](#_Toc196908296)

[2.3 系统运行流程 4](#_Toc196908297)

[3 系统设计 7](#_Toc196908298)

[3.1 业务模型 7](#_Toc196908299)

[3.2 数据库设计 8](#_Toc196908300)

[3.2.1飞机机型表（AIRPLANE\_MODEL） 8](#_Toc196908301)

[3.2.2 航班计划表（FLIGHT\_SCHEDULER） 8](#_Toc196908302)

[3.2.3 航班表（FLIGHT） 9](#_Toc196908303)

[3.2.4 营业网点表（BRANCH） 9](#_Toc196908304)

[3.2.5 出票记录表（TICKET\_ORDER） 10](#_Toc196908305)

[3.3 DAO 接口设计 10](#_Toc196908306)

[3.3.1 FlightDAO 11](#_Toc196908307)

[3.3.2 BranchDAO 11](#_Toc196908308)

[3.3.3 TicketOrderDAO 12](#_Toc196908309)

[4 关健代码分析 14](#_Toc196908310)

[4.1流程代码分析 14](#_Toc196908311)

[4.2用表格显示表单 15](#_Toc196908312)

[4.3 Hibernate映射关系 19](#_Toc196908313)

[5 总结 20](#_Toc196908314)

# 1 引言

## 1.1 民航发展概况

2007年，民用航空行业完成旅客总周转量2776.3亿人公里、旅客运输量19452.1万人、货物周转量115.6亿吨公里，民用航空货运量394.9万吨，比2006年同期分别增长18.7%、1.7%和14.3%。国内航线（含港澳）三项指标分别完成232.5亿吨公里、1952.1万人和272万吨，同比分别增长17.1%、16.1%和12.3%；国际航线三项指标分别完成129.1亿吨公里、1884.1万人和119.4万吨，同比分别增长26.4%、20.8%和26.2%。在人民币持续升值预期明确、燃油成本波动相对较平和以及2008年奥运会、2010年世博会、亚运会等相对积极因素推动下，行业正处于景气度不断提升时期。根据研究院的预测，2008年全球航空运输业的表现将不及2007年。预计今年全行业的毛收入大约为5140亿美元，净利润是50亿美元，盈利率为1%。

    鉴于民用航空业的飞速发展，飞机运输量的不断加大，以及人们对航空公司服务和质量的要求提高，本文为某机票代理公司开发一套机票预定系统，来满足航空公司的系统管理和定票管理。

该系统要求通过分布在每一个代理点的客户端机器完成查询航班和出票的功能。该软件系统由两部分组成：安装在代理点机器上的程序是该系统的客户端软件；另外还有安装在代理公司机器上的服务器端软件。客户端软件能够完成与服务器的通信，根据给定的出发地和目的地以及出发日期向服务器发送航班查询请求和出票的请求。服务器端软件能够接受客户端请求，完成航班查询和出票功能，并向客户端返回结果。同时，服务器端软件还能进行代理商的管理，记录并统计每个代理点的业务量并以日志文件的方式保存下来。

## 1.2 技术选择

### 1.2.1 CS结构

C/S结构，即Client/Server(客户机/服务器)结构，通过将任务合理分配到Client端和Server端，降低了系统的通讯开销，可以充分利用两端硬件环境的优势。C/S 一般面向相对固定的用户群, 对信息安全的控制能力很强。一般高度机密的信息系统采用C/S 结构适宜。C/S 程序可以处理用户面固定, 并且在相同区域, 安全要求高需求,与操作系统相关。应该都是相同的系统 。

由于航空定票系统升级不频繁，用户稳定，对安全性，保密性的要求较高，不需要网络顶票。而c/s结构能满足本系统的要求，所以本系统采用c/s结构设计。

### 1.2.2 Java技术

   Java语言是一个支持网络计算的面向对象程序设计语言，吸收了Smalltalk语言和C++语言的优点，并增加了其它特性，如支持并发程序设计、网络通信和多媒体数据控制等。主要特性如下：
　　1、面向对象性：Java语言是一个纯的面向对象程序设计语言。
　　2、健壮性：Java的强类型机制、异常处理、垃圾的自动收集等是Java程序健壮性的重要保证。Java的安全检查机制使得Java更具健壮性。
　　3、安全性: Java提供了一个安全机制以防恶意代码的攻击。

4、体系结构中立：Java程序在Java平台上被编译为体系结构中立的字节码格式, 然后可以在实现这个Java平台的任何系统中运行。

5、可移植：这种可移植性来源于体系结构中立性，Java系统本身具有很强的可移植性。
　　6、解释型语言：Java程序在Java平台上被编译为字节码格式， Java解释器对这些字节码进行解释执行。
　　7、多线程：Java语言支持多线程，并提供多线程之间的同步机制。
　 本系统采用java作为开发语言，来实现安全的、多线程的、可移植航空定票系统，并且降低开发难度。

## 1.3 开发工具

1、Eclipse集成开发环境

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。就其本身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境。Eclipse 附带了一个标准的插件集，包括 Java 开发工具（Java Development Tools，JDT）。 Eclipse 还包括插件开发环境（Plug-in Development Environment，PDE），这个组件主要针对希望扩展 Eclipse 的软件开发人员，因为它允许他们构建与 Eclipse 环境无缝集成的工具。

2、Hibernate

Hibernate是一个开放源代码的对象关系映射框架，它对JDBC进行了轻量级的对象封装，使得Java程序员可以使用纯的面向对象的思想来操纵数据库中的数据。Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用。Hibernate可以在应用EJB的J2EE架构中取代CMP，完成数据持久化的重任。

# 2 需求分析

## 2.1软件分层与部署

本文把软件分为三个层次：分别是表现层，业务层和持久层，部署如图1所示。

表现层分为客户端和服务器，负责用来接受和显示数据。

业务层都封装在服务器端，负责处理数据与分发数据。

持久层是数据库，负责存储和读取数据。

客户端

服务器

TCP/IP

JDBC/hibernate

数据库

持久层

业务层

表现层

图1 部署图

客户端包装不同类型的请求，利用TCP/IP协议，通过socket发放服务器。服务器通过控制器，把请求转发给不同的模块处理。模块调用相应的Dao访问数据库。软件分层实现了各模块功能之间的松散耦合，有利于功能扩展，系统维护。

## 2.2 系统功能模块

主要的系统功能模块如下：

1、客户端模块：客户端用例图如图2所示

(1)代理商登陆模块BranchDAO.getBranch()

(2)查询航班模块(A):FlightDAO.getAllFlights()

(3)订票模块(A)OrderTicketDAO.order()

(4)修改密码模块(A):AgentDAO.modifyPassword()

图2 客户端用例图

2、服务器模块：服务器用例图如图3所示

(1)添加营业网点模块(B):BranchDAO.addBranch()

(2)删除营业网点模块(B):BranchDAO.removeBranch()

(3)查询营业网点模块(B):BranchDAO.addBranch()

(4)添加飞机型号模块(B):PlaneModelDAO.addPlaneModel()

(5)添加航班计划模块(B):FlightDAO.addFlightSchedular()

(6)查询航班计划模块(B) FlightDAO.getAllFlightSchedulars()

(7)添加航班模块(B):FlightlDAO.addFlightl()

(8)统计所有出票记录模块:TicketOrderDAO.getAllTicketOrder()

(9)统计网点出票记录模块: TicketOrderDAO.getBranchTicketOrder()

图3 服务器用例图

## 2.3 系统运行流程

系统运行流程图如图4所示。

1、服务器启动后自动读取配置文件，之后可以进行航班管理，网点管理，统计

信息等操作。

2、客户端启动并读取配置文件，连接服务器，登陆成功后，才可以进行航班查询、修改密码、订票等操作，每一步操作都要发往服务器处理。

客户端启动

登录

有效数据？

出票

航班查询

修改密码

No

yes

输入查询条件

验证有效？

No

yes

已查航班？

No

输入乘客信息

验证有效？

No

yes

输入新密码

验证有效？

No

yes

封装request对象

将request对象写向服务器

分拣请求

调用FligthDao

调用TicketOrderDAO

调用BranchDao

调用BranchDao

出票请求

查询航班请求

修改密码请求

登陆请求

封装response对象

将response对象写往客户端

解析request对象将结果显示在客户端

图4 流程图

3、服务器为每个客户端单独开一个线程服务，线程转发给控制器，控制器分析请求类型，分发给不同模块处理。

4、模块调用相应的数据库访问对象，处理查询出的数据，将数据封装成响应对象发往客户端。

5、客户端收到响应，执行不同的操作。

# 3 系统设计

## 3.1 业务模型

通过上面的分析，建立了下列java实体类：

1、航班计划(Flight Scheduler)：用来描述由某个航空公司执行的某个定期航班的计划表。例如：由中国国际航空公司执行的每天早上8:00从北京起飞，上午10:00到达上海的CA1202就是一个航班计划。

航班计划中包含如下属性：航班号、起始日期、结束日期、出发地、目的地、离港时间、到港时间、里程、飞机机型、班期、基准票价。主要属性介绍如下：

(1) 航班号：由航空公司给航班计划定义的唯一标识号码，该号码由2位英文字符和4位数字组成，2位英文字符是航空公司编码，4位数字是航班编码。例如：CA1202就是一个航班计划的航班号，其中CA代表中国国际航空公司，1202代表航班的编号。

(2) 出发地：用来描述某个航班计划的起飞城市和机场，所有国内机场均用三个英文字母的代码表示。

(3) 目的地：用来描述某个航班计划的到达城市和机场，所有国内机场均用三个英文字母的代码表示。

(4) 离港时间：用来描述某个航班计划的从出发地机场离开的时刻，离港时间一般代表飞机舱门关闭，不再接收乘客的时刻。该时间精确到分钟。

(5) 到港时间：用来描述某个航班计划到达目的地机场的时间，到港时间一般代表飞机舱门开启，允许乘客下飞机的时刻，该时间精确到分钟。

(6) 开始日期：用来描述某个航班计划的开始日期，该日期精确到某一天。

(7) 结束日期：用来描述某个航班计划的结束日期，该日期精确到某一天。

(8) 班期：用来描述某个航班计划在一周之内哪些天有航班，哪些天没有航班。

2、航班(Flight)：用来描述在某个确定日期执行某个定期航班计划的一次航班。例如：2008年5月25日从北京飞上海的CA1202就是一个航班的实例。

属性：航班计划，出发日期，各舱剩余座位数，折扣。

3、营业网点（Branch）：用来描述航空公司的某个分支机构，该机构能使用该系统为顾客提供查询航班和出票的服务。营业网点通过若干台终端机器与航空公司的服务器相连。营业网点包含以下属性：编号、名称、密码、城市、地址。

4、舱位等级(CabinClass)：用来描述航班的不同舱位，一般分为三种：头等舱（F），公物舱（C），经济舱（Y）。不同的舱位等级具有不同的机票折扣。实际情况中航空公司定义的舱位等级可能更复杂，在本项目中为简化业务模型，只对舱位等级做上述三种划分。

5、乘客类型(TicketType)：用来描述乘坐航班的乘客的类型，一般分为三种：成人(A)（age>12），儿童(C)（2<age<=12），婴儿(I)（0<=age<=2）。不同的乘客类型具有不同的机票折扣，详情请见乘客折扣。在这三种乘客类型中，成人和儿童占用座位，而婴儿不占用座位，婴儿票不能单独出售，一张婴儿票必须凭一张成人票售出。

6、飞机机型（Plane Model）：用来描述执行某个航班计划的飞机的型号，一般来讲，一个航班计划的各次航班都应该采用同种机型的飞机来执行。不同的飞机型号有不同的航程及各舱座位数。实际情况中，各种机型的座位数是不一样的，同一机型还分好几个系列，如B737（波音737）有B737-200、B737-300....B737-900等多个系列，每个系列座位数都不一样。即使是同一型号、同一系列的机型，所属的航空公司不同，座位数也会不同，因为航空公司在购买飞机时，会根据自己的需要要求厂家采用不同的布局，不同的布局会有不同的头等舱、公务舱、经济舱座位数。在本项目中为简化业务模型，我们认为同种型号的飞机各舱座位数是固定的。

7、机票订单(Ticket Order)：一张成功出票的机票记录。

## 3.2 数据库设计

### 3.2.1飞机机型表（AIRPLANE\_MODEL）

表1 飞机机型表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 约束 |
| model | 机型 | VARCHAR (20) | PRIMARY KEY |
| maxlength | 最大航程 | NUMBER(6) | NOT NULL |
| first\_class\_seats | 头等舱座位数 | NUMBER(3) | NOT NULL |
| business\_class\_seats | 公务舱座位数 | NUMBER(3) | NOT NULL |
| economy\_class\_seats | 经济舱座位数 | NUMBER(3) | NOT NULL |

### 3.2.2 航班计划表（FLIGHT\_SCHEDULER）

表2 航班计划表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 约束 |
| flight\_number | 航班号 | CHAR(6) | PRIMARY KEY |
| start\_date | 开始日期 | DATE | NOT NULL |
| end\_date | 结束日期 | DATE | NOT NULL |
| Fromaddress | 出发地机场 | CHAR(3) | REFERENCES airport.airport\_code |
| toaddress | 目的地机场 | CHAR(3) | REFERENCES airport.airport\_codeNOT NULL |
| starttime | 离港时间 | DATE | NOT NULL |
| endtime | 到港时间 | DATE | NOT NULL |
| airplane | 执行机型 | VARCHAR(20) | REFERENCES airplane\_model.modelNOT NULL |
| scheduler | 班期 | CHAR(7) | NOT NULL |
| sail\_length | 航程 | NUMBER(5) | NOT NULL |

### 3.2.3 航班表（FLIGHT）

表3 航班表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 约束 |
| id | 航班编号 | NUMBER(8) | PRIMARY KEY |
| flight\_number | 航班号 | CHAR (6) | REFERENCES flight\_scheduler. flight\_number |
| calendar | 出发日期 | DATE | NOT NULL |
| first\_class\_remain\_seats | 头等舱剩余座位数 | NUMBER(3) | NOT NULL |
| business\_class\_remain \_seats | 公务舱剩余座位数 | NUMBER(3) | NOT NULL |
| economy\_class\_remain \_seats | 经济舱剩余座位数 | NUMBER(3) | NOT NULL |
| priceoff | 季节折扣 | NUMBER(4,2) | NOT NULLDEFAULT 1.0 |

### 3.2.4 营业网点表（BRANCH）

表4 营业网点表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 约束 |
| id | 网点编号 | NUMBER (4) | PRIMARY KEY |
| name | 网点名称 | VARCHAR(40) | NOT NULL |
| password | 密码 | VARCHAR (15) | NOT NULL |
| address | 地址 | VARCHAR(50) | NOT NULL |
| city | 所在城市 | VARCHAR(10) | NOT NULL |

### 3.2.5 出票记录表（TICKET\_ORDER）

表5 出票记录表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 约束 |
| id | 机票编号 | NUMBER(10) | PRIMARY KEY |
| flight\_id | 航班编号 | NUMBER(8) | NOT NULLREFERENCES FLIGHT.id |
| passengername | 乘客姓名 | VARCHAR(40) | NOT NULL |
| passengerid | 证件号码 | VARCHAR(20) | NOT NULL |
| Calendar | 出票日期 | DATE | NOT NULL |
| cabinclass | 舱位等级 | CHAR(1) | NOT NULL CHECK IN(F,C,Y) |
| tickettype | 乘客类型 | CHAR(1) | NOT NULL CHECK IN(A,C,I) |
| branch\_id | 营业网点编号 | NUMBER(4) | NOT NULLREFERENCES BRANCH.id |

## 3.3 DAO 接口设计

设计了通用的DAO接口，而不是直接写访问数据库的实现类，这样可以创建不同的实现类来实现接口，使标准制定和标准实现分离。当换了数据库，或者换了数据库访问技术，就可以写新的实现类，不用改变原来的代码。如定义FlightDAO接口，若从文件中读取数据可以定义FlightDaoFromFile实现类，若利用JDBC访问数据可以定义FlightDaoFromJDBC实现类，若利用hibernate访问数据可以定义FlightDaoFromHibernate实现类等。

### 3.3.1 FlightDAO

flightDAO用来航班计划，航班的数据访问。

package com.tarena.abs.dao;

import com.tarena.abs.model.\*;

import java.util.\*;

//对航班和航班计划数据访问的接口。

public interface FlightDAO{

 // 根据指定出发地，目的地和出发日期在底层数据源中查找

 //得到所有的航班对象的集合。

public Set getAllFlights(String fromAddr,String toAddr,Calendar date);

 //执行出票的操作。

 public boolean order(Order ord);

 //添加指定的航班计划对象。

 public boolean addFlightSchedular(FlightSchedular fs);

 //根据给定的航班编号在底层数据源中删除该航班计划，

//以及该计划下的所有航班。

 public boolean removeFlightSchedular(String flightNumber);

 // 得到所有航班计划对象

 public Set getAllFlightSchedulars();

 //添加航班

 public boolean addFlight(Flight fl);

}

### 3.3.2 BranchDAO

BranchDAO用来访问网点数据。

package com.tarena.abs.dao;

import com.tarena.abs.model.\*;

import java.util.\*;

// 对网点数据进行访问的接口。

public interface BranchDAO{

 // 根据给定的姓名和密码在底层数据源中查找代理商的纪录，

// 若找到，则返回该代理商对象，若没找到或密码错误则返回null。

 public Branch getBranch(String name,String passwd);

 // 向底层数据中添加一个网点纪录。

 public boolean addBranch(Branch user);

 // 根据指定的网点的姓名在底层数据源中删除该网点。

 public boolean removeBranch(String name);

 //修改指定网点的密码。

 public boolean modifyPassword(String name,String oldPassword,String newPassword);

 //查询所有网点信息

 public Set getAllBranch();

}

### 3.3.3 TicketOrderDAO

TicketOrderDAO用来访问票单数据。

package com.tarena.abs.dao;

import java.util.Set;

import java.util.Calendar;

import com.tarena.abs.model.\*;

// 机票出票记录访问接口

public interface TicketOrderDAO {

 // 执行出票的操作。

 public boolean order(Order ord);

 // 执行退票操作。

 public boolean cancelOrder(int TicketNumber);

 // 查询指定营业网点在指定时间段内的出票记录

 public Set<TicketOrder> getAllTicketOrder(Branch branch,Calendar startDate,Calendar endDate);

 // 得到指定营业网点指定日期内的营业额。

 public double getAllTicketMoney(Branch branch,Calendar startDate,Calendar endDate);

 //得到所有营业网点指定日期内的营业额

 public double getAllTicketMoney(Calendar startDate,Calendar endDate);

 //查询所有出票信息

 public Set<TicketOrder> getAllTicketOrder();

}

# 4 关健代码分析

## 4.1流程代码分析

 服务器启动时读取配置文件，读取数据，并且服务器端ServerSocket 等待客户端线程访问，当收到客户端相应后，则创建一个新的服务线程 执行服务。

 ServerSocket ss=null;

 Socket s=null;

 try {

 //创建服务器socket

ss=new ServerSocket(Integer.parseInt(pro.getProperty("ServerPort")));

 while(true){

 s=ss.accept();

//转发给serverThread处理（传递socket参数）

new ServerThread(s).start()

 }

 } catch (IOException e) {

 e.printStackTrace();

 }

服务线程ServerThread通过构造函数接受socket，然后转发给Controller

 public class ServerThread extends Thread{

 Socket s;

 protected ServerThread(Socket s){ //用传入的socket初始化

 this.s=s;

 }

 public void run(){

 try {

 new Controller(s).handle(); //交给控制器处理

 } catch (Exception e) {

 e.printStackTrace();

 }finally{

 try{s.close();}catch(IOException e){}

 }

 }

}

由controller调用handle方法，分析请求对象。

if(type.equals("login")){ // 登陆处理 loginHandle(req);

}else if(type.equals("flightSearch")){ //航班计划查询处理

 flightSearchHandle(req);

}else if(type.equals("order")){ //订单处理

 orderHandle();

}else if(type.equals("quit")){ //退出处理

 //quitHandle();

 break;

}else if(type.equals("modifyPasswd")){

 modifyPasswd(req);

}

有不同的处理程序，调用相应的dao将查询信息封装到response中。

hs=(HashSet)(flightDao.getAllFlights(fromAddr,toAddr,cal));

Response res=new Response("flightSearch");

res.setData(hs);

## 4.2用表格显示表单

 当客户端查询到航班计划时，把flightlist航班集合传给FlightTableModel,再把FLightTableModel传给JTable，然后把JTable加入JScorllPane，就会自动显示数据。

 FlightTableModel ftm = new FlightTableModel(flightlist);

 JTable jt = new JTable(ftm);

 ClientMainClass.clientFrame.setTable(jt);

FlightTableModel实现了AbstractTableModel接口，实现了如下四个方法。

package com.tarena.abs.client;

import javax.swing.table.\*;

import java.util.\*;

import com.tarena.abs.model.\*;

public class FlightTableModel extends AbstractTableModel{

 ArrayList s;

 public FlightTableModel(ArrayList s){

 this.s=s;

 }

 //获得列名

 public String getColumnName(int arg0) {

 switch(arg0){

 case 0: return "航班号";

 case 1: return "出发地";

 case 2: return "目的地";

 case 3: return "起飞时间";

 case 4: return "到达时间";

 case 5: return "机型";

 case 6: return "票价";

 case 7: return "头等舱";

 case 8: return "公务舱";

 case 9: return "经济舱";

 default: return null;

 }

 }

 //获得行数

 public int getColumnCount() {

 return 10;

 }

 //获得列数

 public int getRowCount() {

 return s.size();

 }

 //获得指定位置的值

 public Object getValueAt(int row, int col) {

 Flight f=(Flight)s.get(row);

 if(row<0 || row>s.size())

 return null;

 switch(col){

 case 0: return f.getSch().getFlightNumber();

 case 1: return f.getSch().getFromAddress();

 case 2: return f.getSch().getToAddress();

 case 3: return f.getSch().getFromTime();

 case 4: return f.getSch().getToTime();

 case 5: return f.getSch().getPlane();

 case 6: return

(int)(f.getSch().getPrice()\*f.getPriceOff());

 case 7: return f.getFCSRemain();

 case 8: return f.getBCSRemain();

 case 9: return f.getECSRemain();

 default: return null;

 }

 }

}

 在服务器中，会有航班计划，营业网点，票单等多种类需要显示，如果为每个类写一个TableModel类，工作量大，且当改动业务模型时，维护量大。考虑如此情况，本文利用反射机制，写了一个ArrayListTableModel类，来实现多种来型的表格显示，此类如下：

package com.tarena.abs.server;

import java.lang.reflect.Field;

import java.util.ArrayList;

import java.util.\*;

import javax.swing.table.AbstractTableModel;

public class ArrayListTableModel extends AbstractTableModel{

 private ArrayList arr;

 public ArrayListTableModel(ArrayList arr){

 this.arr=arr;

 }

 //获得列名，即属性名

 public String getColumnName(int arg0) {

 Class c=arr.get(0).getClass();

 Field[] fs = c.getDeclaredFields();

 return fs[arg0].getName();

 }

 //获得列数，即属性数

 public int getColumnCount() {

 Class c=arr.get(0).getClass();

 return c.getDeclaredFields().length;

 }

 //获得行数，即元素个数

 public int getRowCount() {

 return arr.size();

 }

 //获得指定索引的值 (行，代表对象，列，代表属性)

 public Object getValueAt(int rowIndex, int columnIndex) {

 Class c=arr.get(0).getClass(); //arr类型

 Field[] fs = c.getDeclaredFields();

 //Field f = fs[columnIndex] 按索引返回属性列表中某个属性

//Object o = f.get(obj)返回指定对象(obj)上

//此 Field 表示的字段的值o

 //Object o = arr.get(rowIndex)返回数组中索引处的对象

 //设置所有访问属性

 for(int i=0;i<fs.length;i++){

 fs[i].setAccessible(true);

 }

 try {

 Field f = fs[columnIndex];

 Object o = arr.get(rowIndex);

 Object col = f.get(o); //属性字段的对象

 if(col==null)return " —— ";

 if(col.getClass()==java.util.GregorianCalendar.class){

 Calendar cal = (Calendar)col;

 StringBuffer sb = new StringBuffer();

 sb.append(cal.get(Calendar.YEAR)+"年 ");

 sb.append(cal.get(Calendar.MONTH)+"月 ");

 sb.append(cal.get(Calendar.DAY\_OF\_MONTH)+"日");

 return sb;

 }else{

 //把值转化为string

 return f.get(arr.get(rowIndex))+"";

 }

 } catch (IllegalArgumentException e) {

 e.printStackTrace();

 } catch (IllegalAccessException e) {

 e.printStackTrace();

 }

 return null;

 }

 }

## 4.3 Hibernate映射关系

1、构建pojo(Plain Old Java Objects)对象

用JDBC储存数据时，涉及多个表格的修改，查找。工作难度较大。所以考虑用hibernate实现数据储存。首先把模型改写为标准pojo。

(1) 添加Long 类型的oid属性。

(2) 为保存每个属性添加getter，setter方法。

(3) 添加无参构造函数。

(4) 写hibernate映射文件。

2、Hibernate储存原理

当储存数据时，hibernate自动调用getter方法，把属性存入相应字段。

当读取数据时，hibernate自动调用无参构造方法创建对象，然后调用setter方法给对象赋值，从而产生和原来对象相等（equals）的对象。

当增、删、改操作时，hibernate会自动把与原数据相关联的表的字段修改（必须设置级联）。

3、映射关系分析

(1) 航班计划和飞机型号是多对一关系：每个航班计划都包含飞机型号属性，各飞机型号会被多个航班计划引用。

(2) 航班和航班计划是多对一的关系：每个航班都属于某个航班计划，因此每个航班都包含一个航班计划属性，而每个航班表都有一个外间指向航班计划的id子段。

(3) 定单与航班是多对一关系：一个定单包含的航班信息有它包含的航班属性得到，通过航班属性还可以得到航班计划和飞机型号的信息。每个定单只能有一个航班，每个航班可以包含在多个定单中，因此定单与航班是多对一关系。

(4) 定单和网点是多对一的关系：原理同定单和航班的关系。

# 5 总结

 航空订票系统使用软件分层结构，利用面向对象的设计方法，并把学到的知识应用于实践。实现了稳定、可维护、可扩展性的软件，并且完成业务需求。如做以下改进会使系统更加完善：

1、初始化配置参数：

程序是直接读配置文件来读取初始化参数的，如下所示：

ServerIP=127.0.0.1

ServerPort=8888

可以将此参数放入程序中，由用户或管理员输入，来选侧不同的服务器。

客户端的参数设置及重新连接功能有待实现。

2、服务器掉线：

默认是实现是客户端先退出，通知服务器，服务器从内存中删掉此客户端，然后客户端关闭连接。

 private void quitHandle(Request req){

 String currentUserName=(String)req.getData("currentUser");

 for(Object o:onlineAgent){

 Branch a = (Branch)o;

 if(a.getName().equals(currentUserName)){

 onlineAgent.remove(a);

 }

 }

 try {

 s.close();

 } catch (IOException e) {

 e.printStackTrace();}

 }

 但是，当服务器由于网络故障，或者系统维修时，临时断开，没有通知客户端，当客户端此时向服务器发请求时，便会出现想不到的错误。 一个有效的解决方法是：利用观察者模式。在服务器推出方法中遍历在线客户端socket，发送等待信息。

Private void quit(){

 For(Socket s:Currentsockets){

 ······ //封装等待信号给客户端

}

 }

客户端受到等待信号后，执行waitServer()方法，并禁止操作。

3、请求响应对象：

现在的request对象封装了String类型的Type变量代表请求类型。Response对象包装了Object类型的Date变量作为相应。可以把请求对象作为枚举类型，更安全，不易出错。

4、数据显示的轮动更新：

服务器端显示航班信息，网点信息，出现一张表，该表不能滚动和更新。可以利用可滚动、可更新的结果集以及Swing的某些特性实现excel中的实时修改数据功能。实现此功能较为复杂。