**《分布式旅行系统》**

**姓 名：许玉壮**

**学 号：SY1406123**

**日 期：2015-07-01**

**目 录**

[1 题目描述 3](#_Toc423555372)

[1.1 系统功能 3](#_Toc423555373)

[1.2 系统对象 3](#_Toc423555374)

[2 CORBA实验 5](#_Toc423555375)

[2.1 应用系统设计 5](#_Toc423555376)

[2.1.1 用户代理 5](#_Toc423555377)

[2.1.2 酒店服务器 6](#_Toc423555378)

[2.1.3 车票服务器 6](#_Toc423555379)

[2.2 应用系统实现 7](#_Toc423555380)

[2.2.1 物理拓扑 7](#_Toc423555381)

[2.2.2 酒店服务器 8](#_Toc423555382)

[2.2.3 车票服务器 11](#_Toc423555383)

[2.2.4 用户代理 13](#_Toc423555384)

[2.3 实验环境及结果 15](#_Toc423555385)

[2.3.1 实验环境 15](#_Toc423555386)

[2.3.2 实验结果 15](#_Toc423555387)

[3 J2EE&Web Service实验 20](#_Toc423555388)

[3.1 应用系统设计 20](#_Toc423555389)

[3.2 应用系统实现 20](#_Toc423555390)

[3.2.1 价格服务(PriceService) 20](#_Toc423555391)

[3.2.2 余量服务(RemainsService) 22](#_Toc423555392)

[3.2.3 订单服务(OrderService) 23](#_Toc423555393)

[3.2.4 业务流程 25](#_Toc423555394)

[3.2.5 客户端程序 27](#_Toc423555395)

[3.3 实验环境及结果 29](#_Toc423555396)

[3.3.1 实验环境 29](#_Toc423555397)

[3.3.2 实验结果 29](#_Toc423555398)

[4 总结 32](#_Toc423555399)

# 题目描述

## 系统功能

本实验实现一个小型的分布式旅行系统，其功能包括酒店信息处理，车票信息处理和订单处理。酒店信息处理模块包括查看酒店信息，预订酒店；车票信息处理模块包括查看酒店信息，预订车票；订单处理模块包括查看酒店订单和查看车票订单。系统用例图如图1‑1所示。

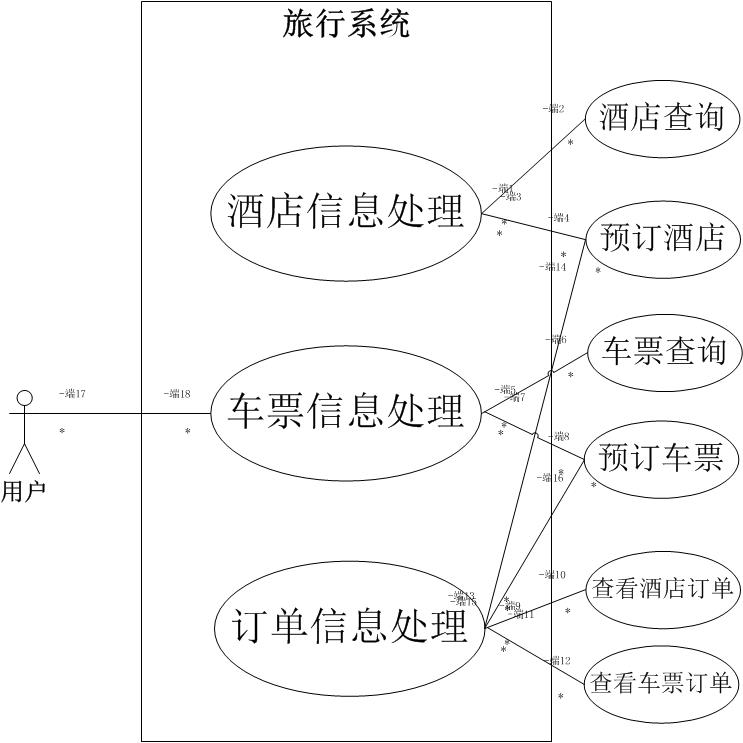


图1‑1 旅行系统用例图

## 系统对象

本旅行系统包括三个分布式对象——用户代理、酒店服务器和车票服务器。三个对象分别分布在独立的空间上，它们通过多种中间件技术进行信息交互。

用户代理(作为Client)为用户提供操作界面，包括输入酒店信息查询条件、选择酒店、预订酒店，输入车票查询条件、选择车票、预订车票，进行查询酒店订单、查询车票订单。

酒店服务器(作为Server)操作酒店信息数据库，接受用户代理的请求，对酒店信息数据库的酒店表和订单表进行查询、增加、修改操作。

车票服务器(作为Server)操作车票信息数据库，接受用户代理的请求，对车票信息数据库的车票表和订单表进行查询、增加和修改操作。

系统结构如图1‑2所示。

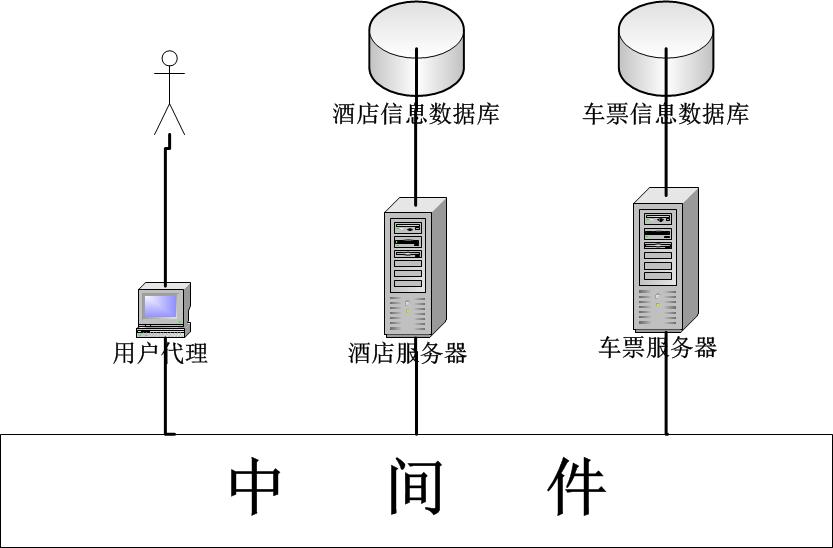


图1‑2 系统结构图

# CORBA实验

## 应用系统设计

CORBA实验的系统结构与1.2节的系统结构(如图1‑2所示)相同，Client和Server所实现的功能也与1.2节描述的功能相同。只是Client与Server之间的通信选择CORBA技术。

### 用户代理

用户代理向用户提供界面。用户可以通过用户代理实现查询酒店、预订酒店，查询车票、预订车票，查询酒店订单、查询车票订单等功能。用户功能流程图如图2‑1所示。

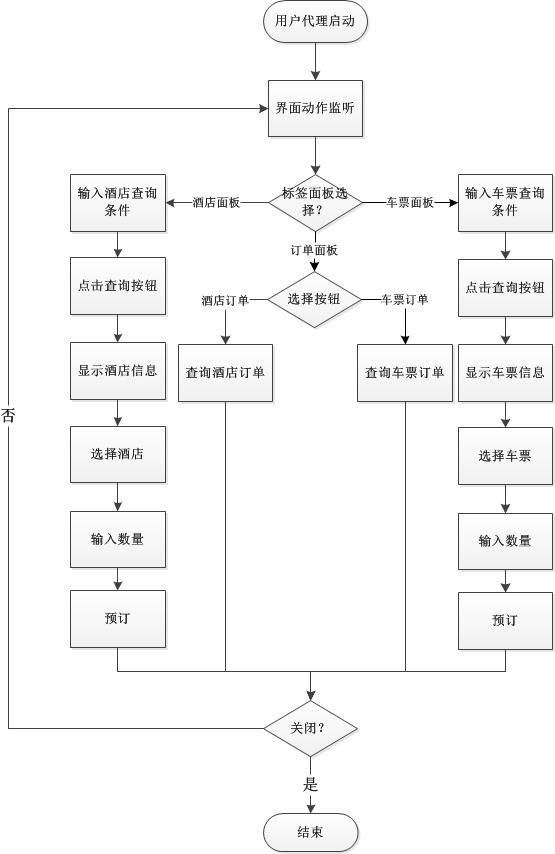


图2‑1 用户功能流程图

### 酒店服务器

酒店服务器接受用户代理的请求，对酒店信息数据库的酒店表和订单表进行操作，包括查询酒店信息表，增加订单，修改酒店房间余量等。其功能流程如图2‑2所示。

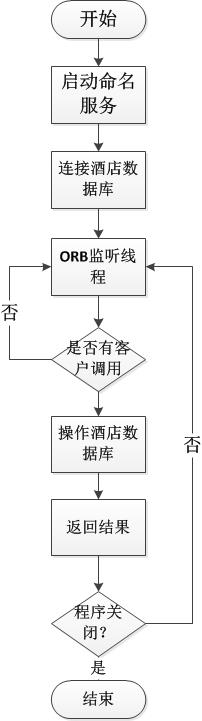


图2‑2 酒店服务器功能流程图

### 车票服务器

车票服务器(作为Server)操作车票信息数据库，接受用户代理的请求，对车票信息数据库的车票表和订单表进行操作，包括查询车票信息表，增加订单，修改车票余量等。器功能流程如图2‑3所示。

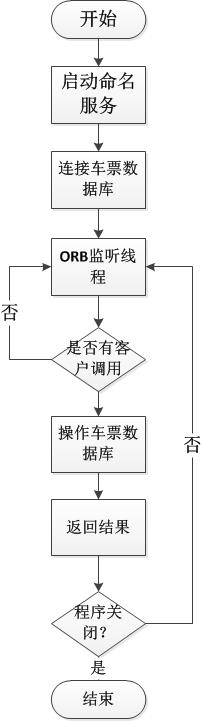


图2‑3 车票服务器功能流程图

## 应用系统实现

### 物理拓扑

由于实验环境有限，实验的服务器端和客户代理端均在一台机器上。ORB的调用端口为1050，主机名为localhost，这两个参数写在程序中，无法更改。如图2‑4和图2‑5所示。

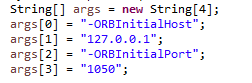


图2‑4 用户代理端连接酒店服务器参数

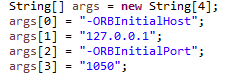


图2‑5 用户代理端连接车票服务器参数

### 酒店服务器

* + - 1. IDL文件——HotelInfo.idl

IDL文件定义了远程对象的接口。HotelInfo.idl文件内容如下：

module HotelInfoApp{

interface HotelInfo{

//酒店信息条目

struct HotelItem{

long Id;

string Destination;

string Name;

string Type;

long price;

long Remains;

string date;

};

typedef sequence<HotelItem> HotelSeq;//返回类型，映射在Java中为无固定

//长度的数组

HotelSeq getHotelInfo(in string dest, in string inData, in string outData, in string type,in string user);//获取酒店信息

long orderHotel(in long id, in string dest,in string name, in string Date, in string type, in long num,in string user);//预订酒店

HotelSeq getOrderInfo(in string user);//查询酒店订单

};

};

HotelInfo.idl文件通过JDK自带的idlj编译工具进行编译，命令如下：

idlj –fall HotelInfo.idl

生成HotelInfoApp包，目录结构如图2‑6所示。

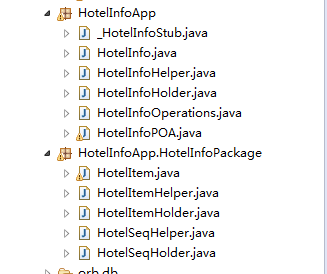


图2‑6 HotelInfoApp包结构

* + - 1. 酒店服务器ORB服务监听代码

**public** HotelServer() **throws** Exception {

// **TODO** Auto-generated constructor stub

String args[] = **new** String[2];

args[0] = "-ORBInitialPort";

args[1] = "1050";

java.util.Properties p = **new** java.util.Properties();

p.setProperty("com.sun.CORBA.codeset.charsets", "0x05010001, 0x00010109"); // UTF-8, UTF-16

p.setProperty("com.sun.CORBA.codeset.wcharsets", "0x00010109, 0x05010001"); // UTF-16, UTF-8

ORB orb = ORB.*init*(args, p);

System.***out***.println("开始初始化ORB!");

org.omg.CORBA.Object obj = orb.resolve\_initial\_references("RootPOA");

System.***out***.println("取根POA引用");

POA rootPoa = POAHelper.*narrow*(obj);

rootPoa.the\_POAManager().activate();

HotelInfoImpl hotelInfoImpl = **new** HotelInfoImpl();

System.***out***.println("创建一个接口的实例!");

org.omg.CORBA.Object ref = rootPoa.servant\_to\_reference(hotelInfoImpl);

HotelInfo href = HotelInfoHelper.*narrow*(ref);

System.***out***.println("从服务中提取对象的引用");

org.omg.CORBA.Object objRef = orb.resolve\_initial\_references("NameService");

NamingContextExt ncRef = NamingContextExtHelper.*narrow*(objRef);

System.***out***.println("得到根名称的上下文!");

String name = "HotelInfo";

NameComponent path[] = ncRef.to\_name(name);

ncRef.rebind(path, href);

System.***out***.println("在根名称的上下文绑定实例对象!");

System.***out***.println("启动线程服务等待客户端调用!");

orb.run();

}

* + - 1. 远程接口实现类——HotelInfoImpl.java

**public class HotelInfoImpl extends HotelInfoPOA**

(1) 连接数据库

**public** HotelInfoImpl() {

// **TODO** Auto-generated constructor stub

String driver = "com.mysql.jdbc.Driver";

String url = "jdbc:mysql://localhost:3306/hotelinfo?characterEncoding=utf8";

String user = "root";

String passwd = "123456";

**try** {

Class.*forName*(driver);

conn = DriverManager.*getConnection*(url, user, passwd);

**if**(!conn.isClosed()){

System.***out***.println("Success Connect!");

}

statement = conn.createStatement();

statement2 = conn.createStatement();

} **catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

System.***out***.println("Connection failed!");

}

}

(2) 远程方法实现

实现了三个方法：

**public** HotelItem[] getHotelInfo(String dest, String inData, String outData, String type,String user);//查询酒店信息

**public** **int** orderHotel(**int** id, String dest, String name, String Date,String type, **int** num,String user);//预订酒店

**public** HotelItem[] getOrderInfo(String user);//查询酒店订单

### 车票服务器

* + - 1. IDL文件——TicketInfo.idl

module TicketInfoApp{

interface TicketInfo{

//车票信息条目

struct TicketItem{

long Id;

string Name;

string Departure;

string Destination;

string StartTime;

string ArriveTime;

long Price;

long Remains;

};

typedef sequence<TicketItem> TicketSeq;//返回类型，映射在Java中为无

//固定长度数组

TicketSeq getTicketInfo(in string dept,in string dest, in string startTime,in string user);

long orderTicket(in long id,in long num,in string user);

TicketSeq getTicketOrderInfo(in string user);

};

};

HotelInfo.idl文件通过JDK自带的idlj编译工具进行编译，命令如下：

idlj –fall TicketInfo.idl

生成TicketInfoApp包，目录结构如所示。

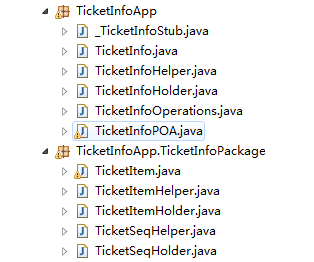


图2‑7 TicketInfoApp包结构

* + - 1. 车票服务器ORB服务监听代码

与2.2.2.2节类似，此处略。

* + - 1. 远程接口实现类—TicketInfoImpl.java

**public class TicketInfoImpl extends TicketInfoPOA**

(1) 连接数据库

**public** TicketInfoImpl() {

// **TODO** Auto-generated constructor stub

String driver = "com.mysql.jdbc.Driver";

String url = "jdbc:mysql://localhost:3306/ticketinfo?characterEncoding=utf8";

String user = "root";

String passwd = "123456";

**try** {

Class.*forName*(driver);

conn = DriverManager.*getConnection*(url, user, passwd);

**if**(!conn.isClosed()){

System.***out***.println("Success Connect!");

}

statement = conn.createStatement();

statement2 = conn.createStatement();

} **catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

System.***out***.println("Connection failed!");

}

}

(2) 远程方法实现

实现了三个方法：

**public** TicketItem[] getTicketInfo(String dept, String dest,String startTime, String user)//查询车票信息

**public** **int** orderTicket (**int** id, **int** num,String user);//预订车票

**public** TicketItem [] getTicketOrderInfo (String user);//查询车票订单

### 用户代理

* + - 1. 目录结构

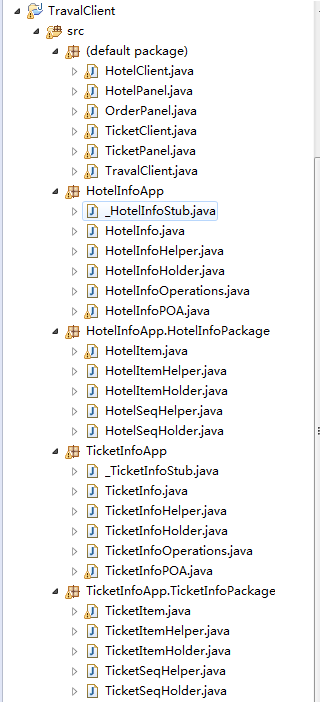


图2‑8 用户代理端目录结构

用户代理端的目录结构如所示，调入HotelInfo.idl生成的HotelInfoApp包和TicketInfoApp包。

* + - 1. 连接服务器代码

**static**{

System.***out***.println("客户端初始化配置开始！");

String[] args = **new** String[4];

args[0] = "-ORBInitialHost";

args[1] = "127.0.0.1";

args[2] = "-ORBInitialPort";

args[3] = "1050";

java.util.Properties p = **new** java.util.Properties();

p.setProperty("com.sun.CORBA.codeset.charsets", "0x05010001, 0x00010109"); // UTF-8, UTF-16

p.setProperty("com.sun.CORBA.codeset.wcharsets", "0x00010109, 0x05010001"); // UTF-16, UTF-8

ORB orb = ORB.*init*(args, p);

org.omg.CORBA.Object objRef = **null**;

**try** {

objRef = orb.resolve\_initial\_references("NameService");

} **catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

}

NamingContextExt neRef = NamingContextExtHelper.*narrow*(objRef);

String name = "HotelInfo";

**try** {

*hotelInfo* = HotelInfoHelper.*narrow*(neRef.resolve\_str(name));

} **catch** (NotFound e) {

// **TODO**: handle exception

e.printStackTrace();

} **catch** (CannotProceed e) {

// **TODO**: handle exception

e.printStackTrace();

} **catch** (org.omg.CosNaming.NamingContextPackage.InvalidName e) {

// **TODO**: handle exception

e.printStackTrace();

}

System.***out***.println("客户端初始化完毕");

}

## 实验环境及结果

### 实验环境

硬件环境：台式机一台

CPU：Intel Core2 Quad CPU Q9400 @ 2.66GHz

内存: 4G

操作系统：Windows7 64位

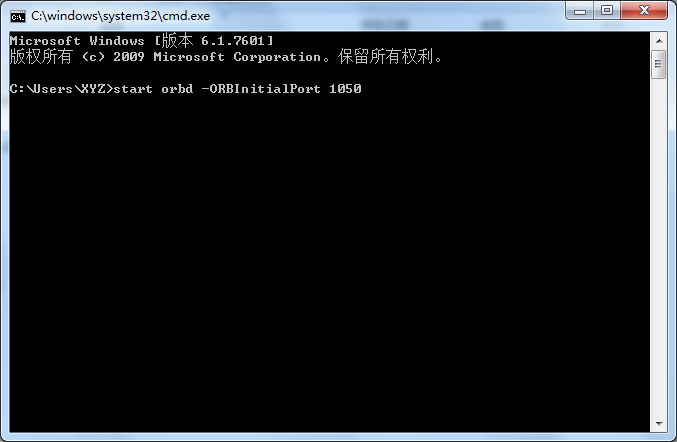
数据库：mysql 5.1

开发环境：JDK 1.8 + Eclipse

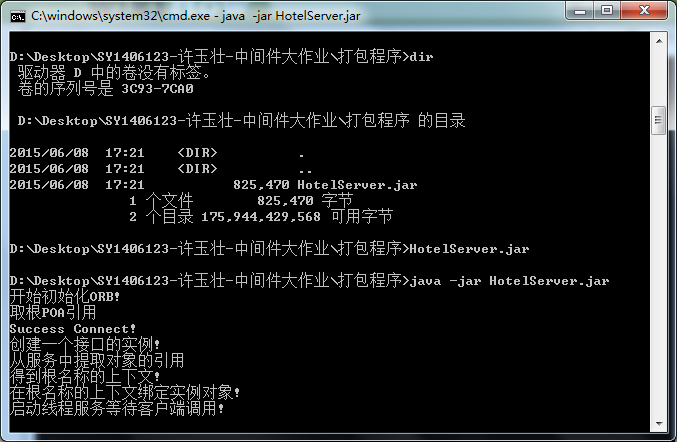
CORBA编译器：JDK自带idl编译器

### 实验结果

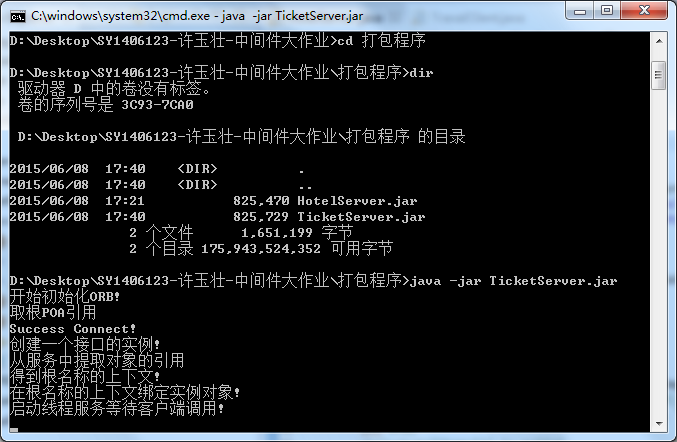
1. 启动ORB命名服务



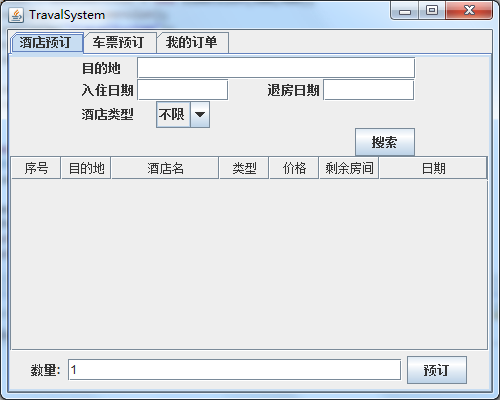
2. 启动酒店服务器



3. 启动车票服务器



4. 用户代理程序主界面

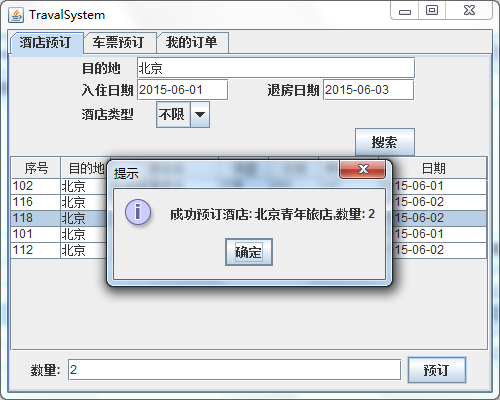


5. 查询酒店

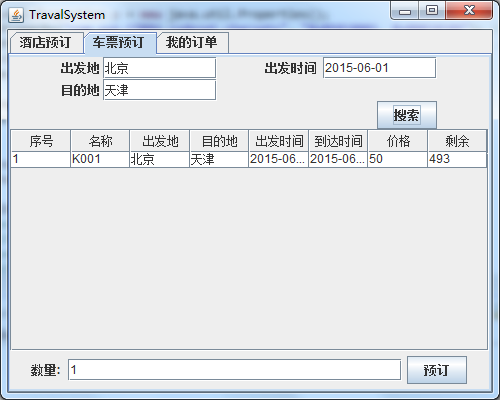


6. 预订酒店

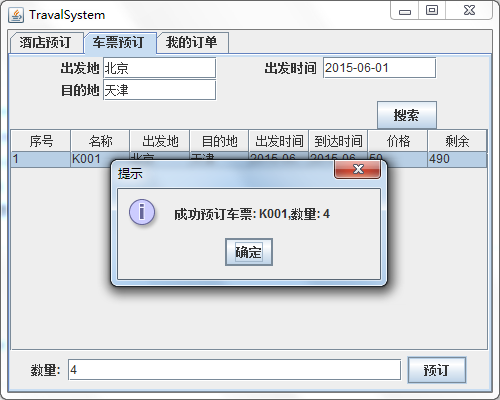




7. 查询车票



8. 预订车票



9. 查看车票订单



# J2EE&Web Service实验

## 应用系统设计

本节实验选取运行系统的一个子集，实现酒店房间预订的业务流程。EJB实现查询价格、修改余量和产生订单，并发布为Web服务；然后Web服务调用端用BEPL集成服务，实现服务流程，并将业务流程发不成Web服务；最后客户端调用业务流程Web服务，则可以预定酒店。系统结构如图3‑1所示。

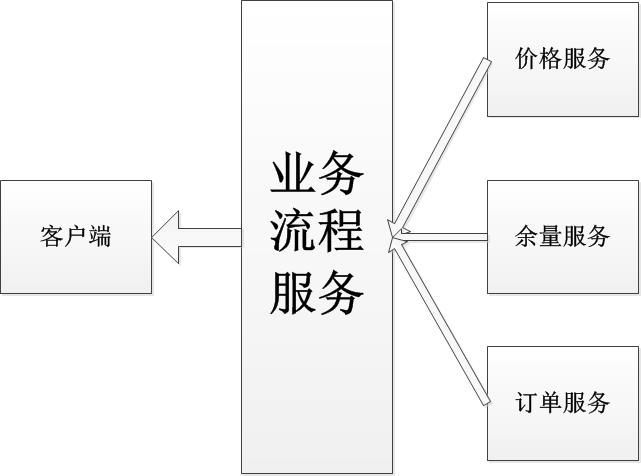


图3‑1 系统结构图

## 应用系统实现

### 价格服务(PriceService)

价格服务用于查询房间的价格，房间一共三种类型：单人房、双人房和套房，在程序中以常量表示。

创建价格Session Bean，生成接口和实现类，目录结构如图3‑2所示。



图3‑2 PriceService Bean目录

PriceServiceRemote为接口，定义了getPrice接口函数。内容如下：

**package** com.hotel.price;

**import** javax.ejb.Remote;

@Remote

**public** **interface** PriceServiceRemote {

**public** **double** getPrice(String type);

}

PriceService是接口的实现，定义了三种房间的价格

**private** **static** **final** **double** ***singlePrice*** = 150;//单人房

**private** **static** **final** **double** ***doublePrice*** = 300;//双人房

**private** **static** **final** **double** ***suitesPrice*** = 500;//套房

实现了getPrice函数。

@Override

**public** **double** getPrice(String type) {

// **TODO** Auto-generated method stub

**if**(type.equalsIgnoreCase("single"))

**return** ***singlePrice***;

**if**(type.equalsIgnoreCase("double"))

**return** ***doublePrice***;

**if**(type.equalsIgnoreCase("suites"))

**return** ***suitesPrice***;

**return** 9999;

}

并通过注释的方式将PriceService这个Session Bean发布成Web 服务。

@Stateless

@Remote({PriceServiceRemote.**class**})

**@WebService**

**public** **class** PriceService **implements** PriceServiceRemote {

……

}

PriceService服务的发布结果如图3‑3所示：

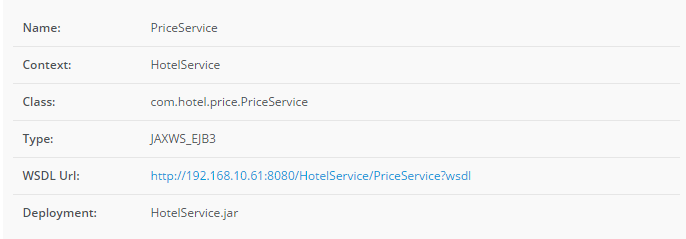


图3‑3 PriceService发布结果

### 余量服务(RemainsService)

余量服务用于查询修改房间余量，预订的房间对应房间类型的数量减去预订的数量。

余量服务Session Bean，生成接口和实现类，目录结构如图3‑4所示。

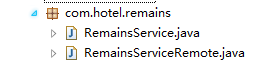


图3‑4 RemainsService Bean目录

RemainsServiceRemote为接口，定义了setRemains接口函数。内容如下：

**package** com.hotel.remains;

**import** javax.ejb.Remote;

@Remote

**public** **interface** RemainsServiceRemote {

**public** **int** getRemains(String type);

**public** **boolean** setRemains(String type, **int** num);

}

RemainsService是接口的实现，定义了三种房间的余量

**private** **static** **int** *singleRemains* = 100; //单人房

**private** **static** **int** *doubleRemains* = 80; //双人房

**private** **static** **int** *suitesRemains* = 50; //套房

实现了setRemains函数。

@Override

**public** **boolean** setRemains(String type, **int** num) {

// **TODO** Auto-generated method stub

**if**(type.equalsIgnoreCase("single")&&(*singleRemains* >= num)){

*singleRemains* -= num;

**return** **true**;

}

**if**(type.equalsIgnoreCase("double")&&(*doubleRemains* >= num)){

*doubleRemains* -= num;

**return** **true**;

}

**if**(type.equalsIgnoreCase("suites")&&(*suitesRemains* >= num)){

*suitesRemains* -= num;

**return** **true**;

}

**return** **false**;

}

并通过注释的方式将这个Session Bean发布成Web 服务。

@Stateless

@Remote({RemainsServiceRemote.**class**})

**@WebService**

**public** **class** **RemainsService** implements **RemainsServiceRemote** {

……

}

RemainsService服务的发布结果如图3‑5所示：

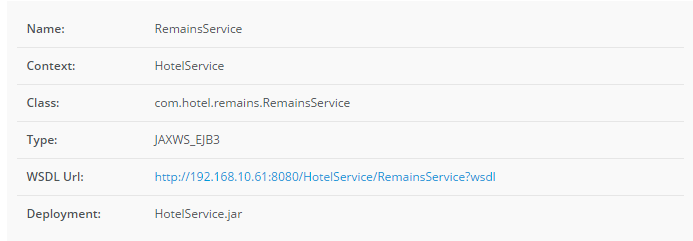


图3‑5 RemainsService发布结果

### 订单服务(OrderService)

订单服务用于生成，订单号由服务器当前时间戳组成。

订单服务Session Bean，生成接口和实现类，目录结构如图3‑6所示。



图3‑6 OrderService Bean目录

OrderServiceRemote为接口，定义了createOrder接口函数。内容如下：

**package** com.hotel.order;

**import** javax.ejb.Remote;

@Remote

**public** **interface** OrderServiceRemote {

**public** String createOrder(String type,**double** price,**int** num);

**public** String getOrder(String orderID);

}

OrderService是接口的实现，实现了createOrder函数。

@Override

**public** String createOrder(String type, **double** price, **int** num) {

// **TODO** Auto-generated method stub

SimpleDateFormat format = **new** SimpleDateFormat("yyyyMMddHHmmss");

String orderID = format.format(**new** Date());

**double** total = price\*num;

String orderItem = orderID+":"+type+":"+price+":"+num+":"+total;

**try** {

orderFile.createNewFile();

out = **new** BufferedWriter(**new** FileWriter(orderFile));

out.write(orderItem+"\n");

out.flush();

out.close();

} **catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

}

**return** orderItem;

}

并通过注释的方式将这个Session Bean发布成Web 服务。

@Stateless

@Remote({OrderServiceRemote.**class**})

**@WebService**

**public** **class** OrderService **implements** OrderServiceRemote {

……

}

OrderService服务的发布结果如图3‑7所示：

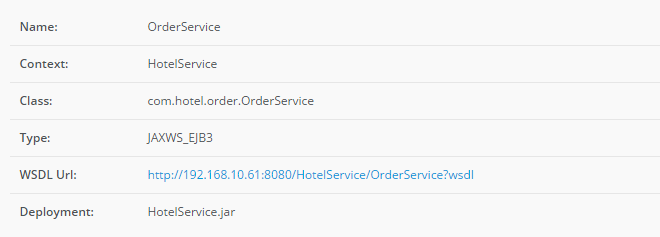


图3‑7 OrderService发布结果

### 业务流程

业务流程实现调用了PriceService、RemainsService和OrderService三个服务，进行集成，实现了预定房间业务流程。

实现过程首先导入三个服务的WSDL文件，然后建立BPEL Process文件HotelorderProcess.bpel，工具自动生成HotelorderProcess Artifacts.wsdl，最后建立BPEL Deployment Descriptor文件deploy.xml将业务流程发布到Apache ODE服务器。

HotelorderProcess.bpel流程图如图3‑8所示。

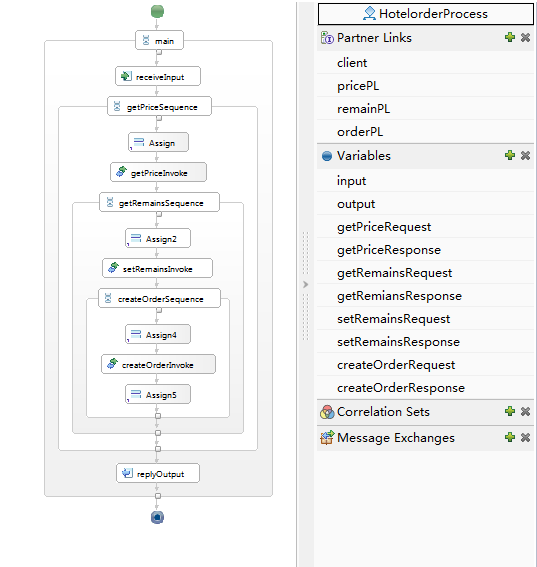


图3‑8 HotelorderProcess.bpel流程图

HotelorderProcessArtifacts.wsdl的内容如图3‑9所示

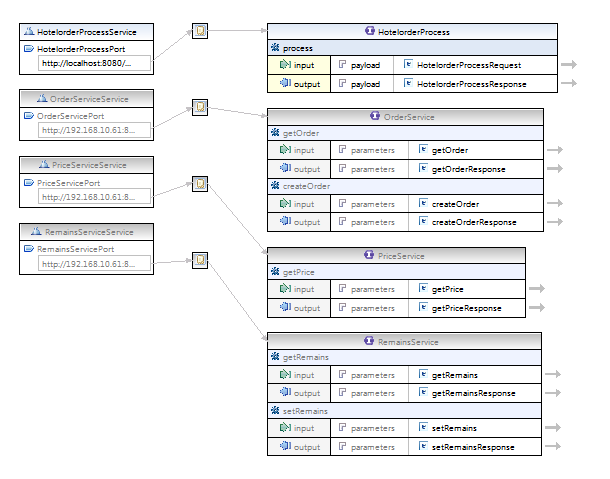


图3‑9 HotelorderProcessArtifacts.wsdl内容

deploy.xml文件内容如图3‑10所示。

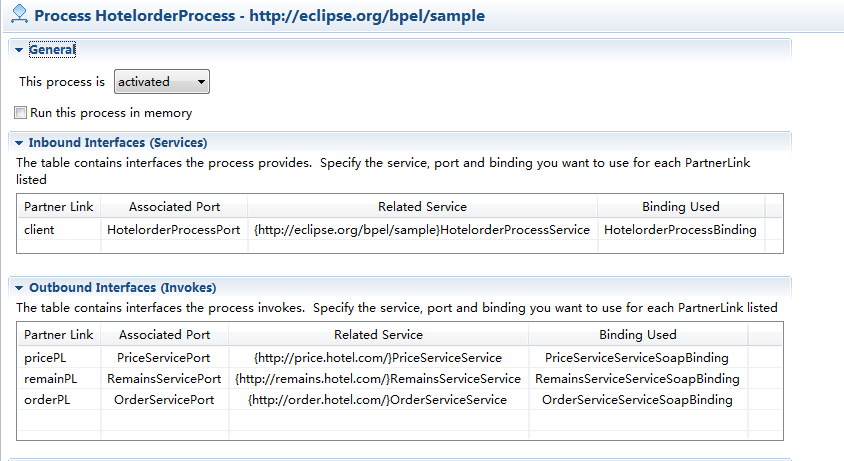


图3‑10 deploy.xml内容

将业务流程发布到Apache ODE服务器结果如图3‑11所示。

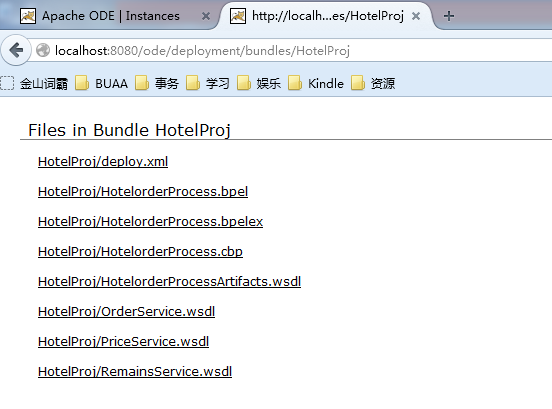


图3‑11 业务流程发布成服务图

### 客户端程序

客户端程序调用业务流程服务，进行房间预订。

利用JDK自带wsimport工具将业务流程服务HotelorderProcess Artifacts.wsdl生成客户端接口文件。

命令：wsimport –p com.hotel.orderprocess –keep HotelorderProcess Artifacts.wsdl。导入客户端工程，目录结果如图3‑12所示。

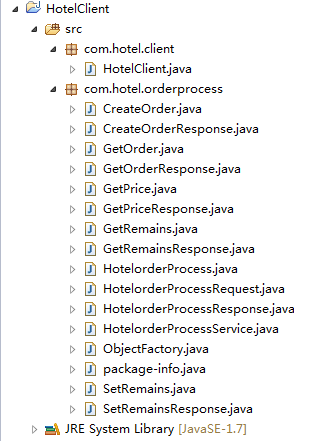


图3‑12 客户端目录结构图

客户端程序内容如下。

package com.hotel.client;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import com.hotel.orderprocess.\*;

public class HotelClient {

public static void main(String[] args) {

// TODO Auto-generated method stub

HotelorderProcessService service = new HotelorderProcessService();

HotelorderProcess hotelOrderProcess = service.getHotelorderProcessPort();

String type = null;

int number = 0;

System.out.println("Please input type of room(single OR double OR suites):");

BufferedReader br= new BufferedReader(new InputStreamReader(System.in));

try {

type = br.readLine();

//System.out.println(type);

System.out.println("Please input ordering number of room:");

number = Integer.parseInt(br.readLine());

//System.out.println(number);

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

HotelorderProcessRequest request = new HotelorderProcessRequest();

request.setType(type);

request.setNumber(number);

HotelorderProcessResponse response = hotelOrderProcess.process(request);

String order = response.getResult();

//System.out.println(order);

String[] result = order.split(":");

System.out.println("Room Ordered:"+

"\n\tOrderID:"+result[0]+

"\n\tType:"+result[1]+

"\n\tPrice:"+result[2]+

"\n\tNumber:"+result[3]+

"\n\tTotal:"+result[4]);

} }

}

## 实验环境及结果

### 实验环境

台式机一台

CPU：Intel Core2 Quad CPU Q9400 @ 2.66GHz

内存: 4G

操作系统：Windows7 -64bit

IP：192.168.10.61

服务器：jboss-eap-6.4

开发环境：JDK 1.8 + Eclipse

笔记本一台

CPU：IntelCore i3 CPU M370@2.40GHz

内存：6G

操作系统：Windows7 -64bit

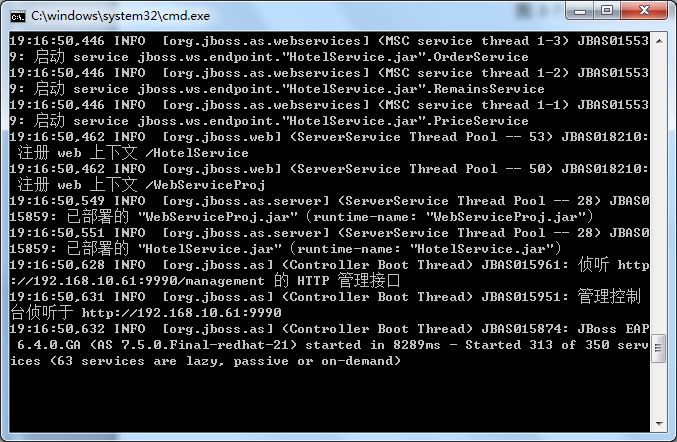
IP：192.168.10.63

服务器：apache-tomcat-8.0.23+apache-ode-war-1.3.6

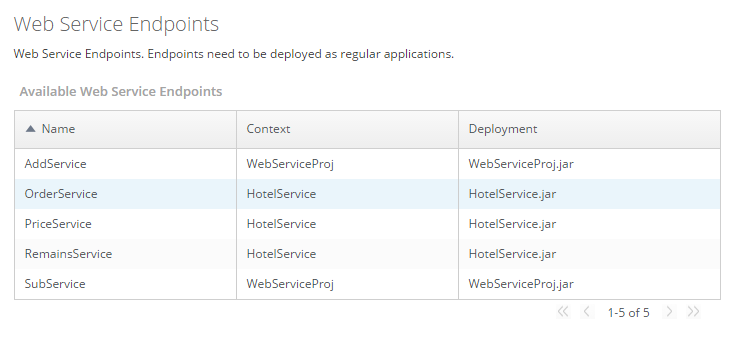
开发环境：JDK 1.7 + Eclipse+BPEL Designer

### 实验结果

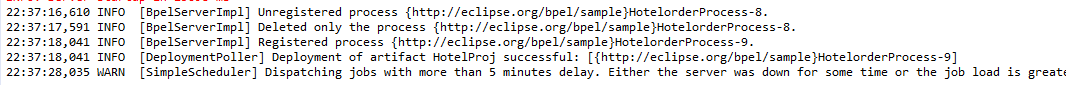
JBoss服务器启动结果：



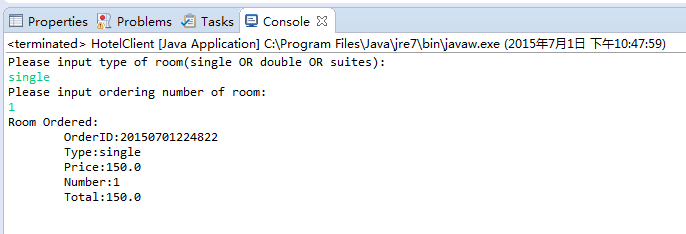
JBoss服务管理界面WebService Endpoints结果：

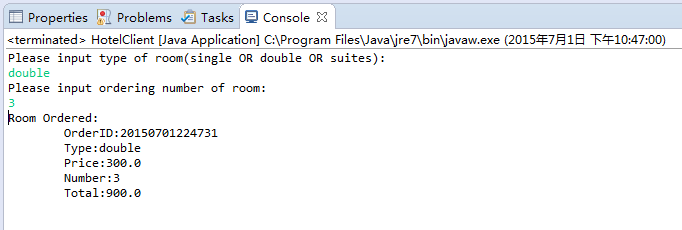


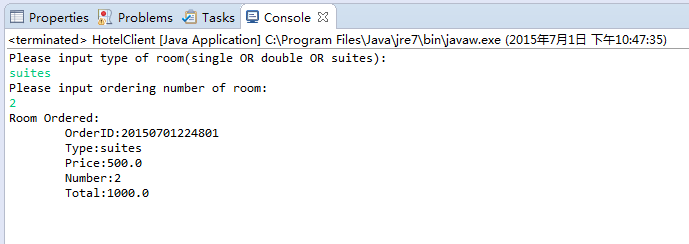
ODE服务器启动及业务流程服务部署结果：



客户端运行结果：







# 总结

通过本学期中间件课程和实验，对分布式系统常用中间件技术有了一定的了解。通过实验，对CORBA技术、J2EE技术以及Web Service技术有了一定的实践，对其了解更为深入。

但是由于时间有限，实现的系统比较简单，实验对于这几种中间件技术的深层原理没有太多涉及，尤其是J2EE&Web Service实验更是因为时间原因只实现了一个简单的功能。因此，在以后的学习和实践过程，需要对于这几种中间件技术进行进一步学习。

最后，感谢郎波老师和林学练老师在本学期的耐心讲授，让我在知识上和学习态度上都有了很大收获。