在线购物平台

面向对象技术课程设计

任俊州 S201507114

张鹏 S201507110

白涵 S201507098

目录

面向对象技术课程设计 1
在线购物平台 3
1 题目描述 3
2 需求分析 3
3 静态建模 6
4 动态建模 10
5 系统实现 13
6 面向对象技术分析 14
7 总结提高 14
附录：源码 15

在线购物平台

1 题目描述

在基于 web 的“在线购物系统中”，客户可以像供应商请求购买一件或多件商品。客户提供个人信息，例如地址和信用卡信息。这些信息被存储在客户账户中。如果信用卡是有效的，那么系统创建一个配送订单并且发送给供应商。供应商检查可用的库存，确认订单，并且输入一个计划好的配送日期。当订单完成配送后，系统通知客户并且向客户的信用卡账户收费。

2 需求分析

2.1 用例图

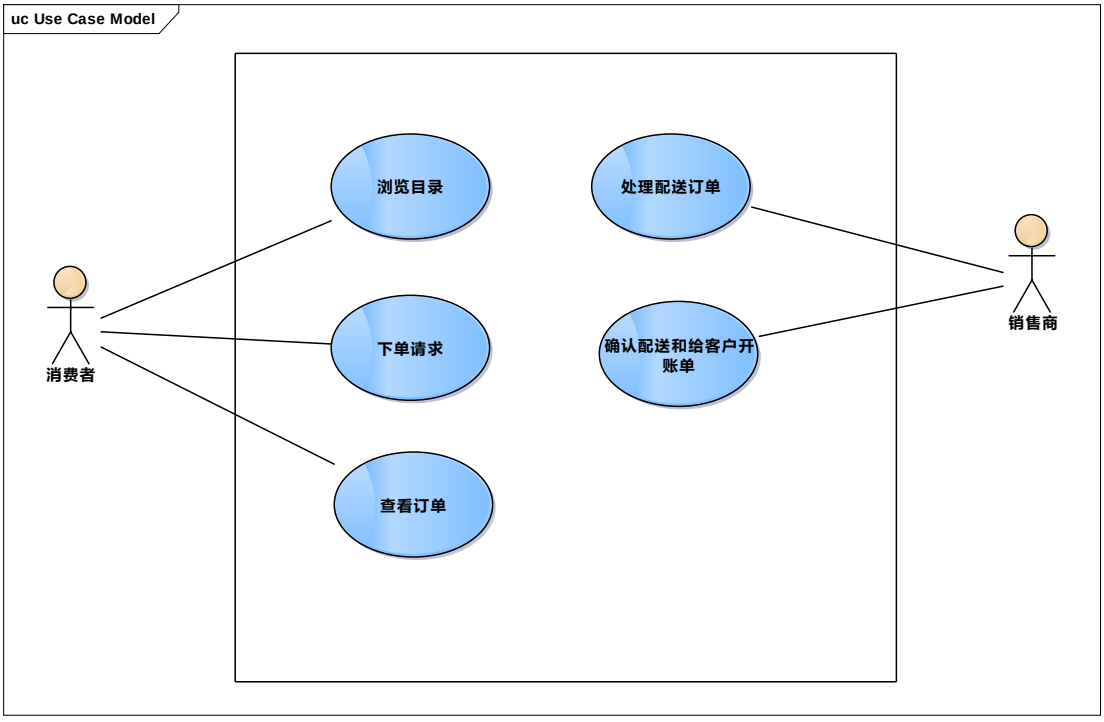


图 2.1 用例分析图

2.2 用例描述

表 2.1 浏览目录用例描述表

用例名称:浏览目录
概述：客户浏览万维网目录，从供应商的目录中查看各种各样的商品项，并且从目录中选择商品。

参与者：客户
前置条件：客户的浏览器链接到供应商的目录网站。
主序列： 1.客户请求浏览目录。 2.系统向客户显示目录信息。 3.客户从目录中选择商品。 4.系统显示商品列表，包含商品描述，价格以及总价。
可替换序列：步骤 3：客户没有选择商品并且退出。
后置条件：系统显示了所选择的商品列表。

表 2.2 下单请求用例描述表

用例名称：下单请求
概述：客户输入一个订单请求来购买目录商品。客户的银行卡被检查其有效性和是否有足够的额度来支付请求的目录商品。
参与者：客户
前置条件：客户选择了一个或者多个目录商品。
主序列： 1.客户提供订单请求和客户账户 ID 来支付购买。 2.系统检索客户账户信息，包括客户的银行卡详细信息。 3.系统根据购买总额检查客户银行卡，如果通过，创建一个银行卡购买授权号码。 4.系统创建一个配送订单，包括订单细节、客户 ID 和银行卡授权号码。 5.系统确认批准购买，并且向客户显示订单信息。
可替换序列： 步骤 2：如果客户没有账户，系统提示客户提供信息来创建一个新账户。客户可输入账户信息或者取消订单。 步骤 3：如果客户的银行卡授权被拒绝（例如，无效的信用卡或者客户的银行卡账户资金不足），系统提示客户输入一个不同的银行卡号码。客户可输入一个不同的信用卡号码或者取消订单。
后置条件：系统为配送订单保留了库存商品。

表 2.3 处理配送订单用例描述表

用例名称：处理配送订单
概述：供应商请求一个配送订单；系统确定库存对于满足订单是可用的，并且显示订单。
参与者：供应商
前置条件：供应商需要处理一个配送订单并且一个配送订单存在。
主序列： 1.供应商请求下一个配送订单。 2.系统检索并且显示配送订单。 3.供应商为配送订单请求商品库存检查。 4.系统确定库存中的商品对于满足订单是可用的，并且保留这些商品。 5.系统给供应商显示库存信息，并且确认商品被保留。
可替换序列：步骤 4：如果商品库存不足，系统显示警告信息。
后置条件：系统提交了库存，向客户收费，并且发送了确认信息。

表 2.4 确认配送和给客户开账单用例描述表

用例名称：确认配送和给客户开账单
概述：供应商手工地准备配送并且确认配送订单已经准备好。系统通知客户订单正在配送。系统通过客户的信用卡收取购买商品的款项并且更新相关库存商品的库存。
参与者：供应商
前置条件：库存商品已经为客户的配送订单进行了预留。
主序列： 1.供应商手工地准备配送并且确认配送订单已经准备好配送。 2.系统检索客户的账户信息，包括发货单和客户的信用卡细节。 3.系统更新库存，确认购买。 4.系统通过客户账户收取购买商品的款项并且创建一个银行卡收费确认号码。 5.系统用银行卡收费确认号码更新配送订单信息。 6.系统给客户发送确认邮件。 7.系统向供应商显示确认信息来完成配送订单的配送。
可替换序列：步骤 4：如果商品库存不足，系统显示警告信息。
后置条件：系统提交了库存，向客户收费，并且发送了确认信息。

3 静态建模

3.1 系统概念静态模型

本系统以数据为基础，故由从底至上的视角进行构建，首先对系统所需的实体类进行描述。

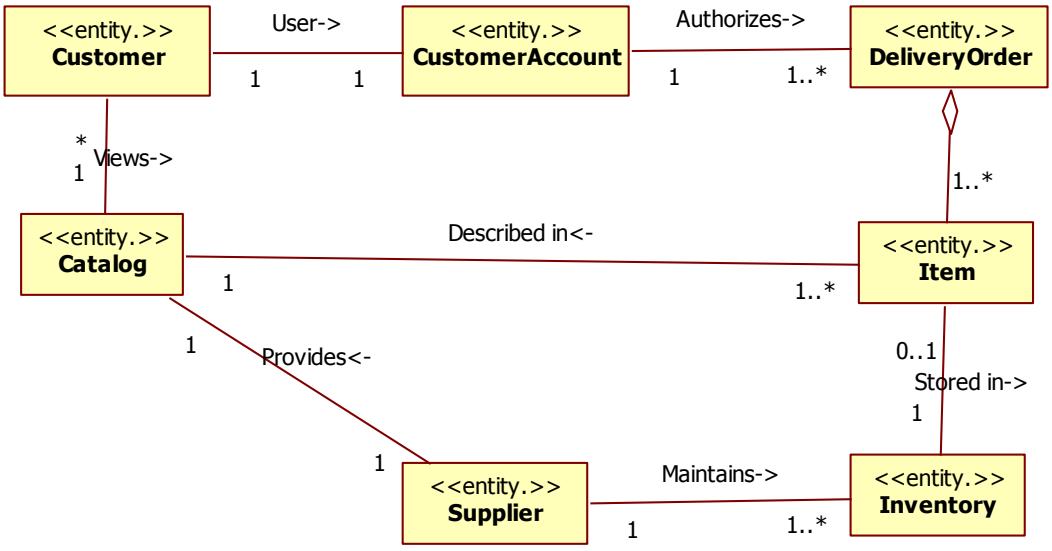


图 3.1“在线购物系统”实体类的静态模型图

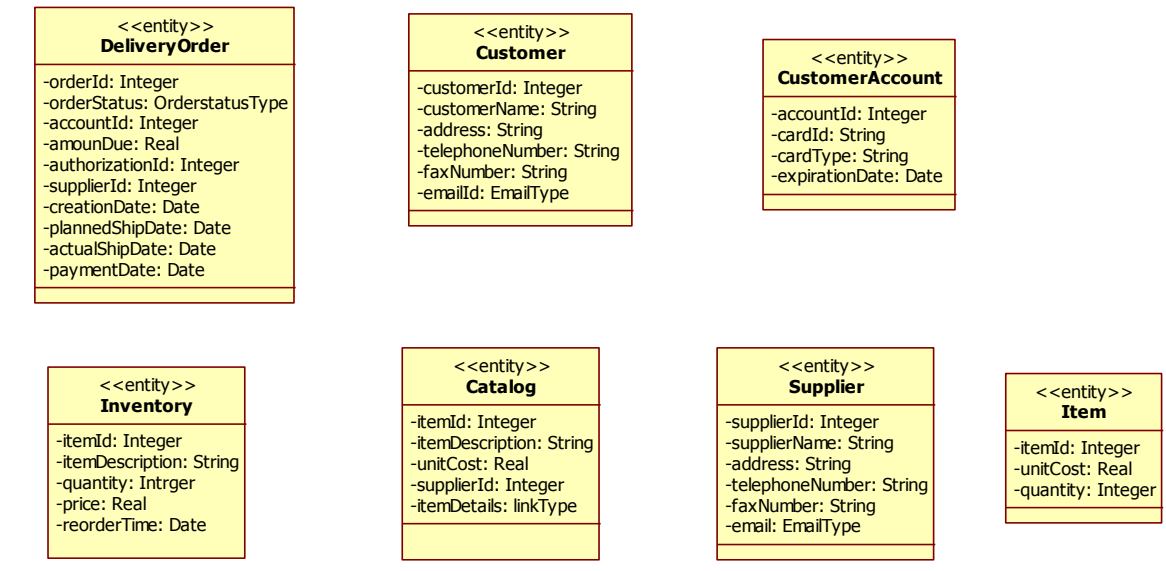


图 3.2“在线购物系统”实体类图

3.2 实体类的组织

将实体类从逻辑上进一步抽象为四个服务类，屏蔽底层的实体细节，构成系统的最下层，服务层。

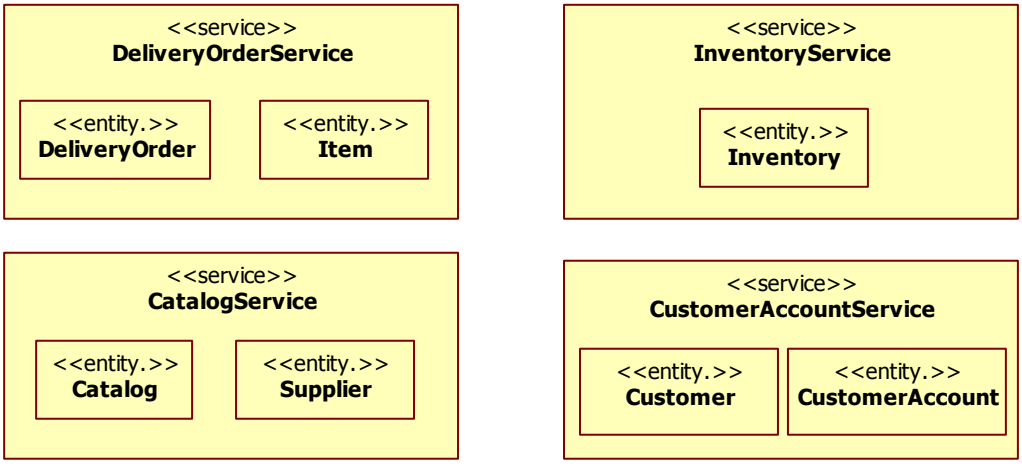
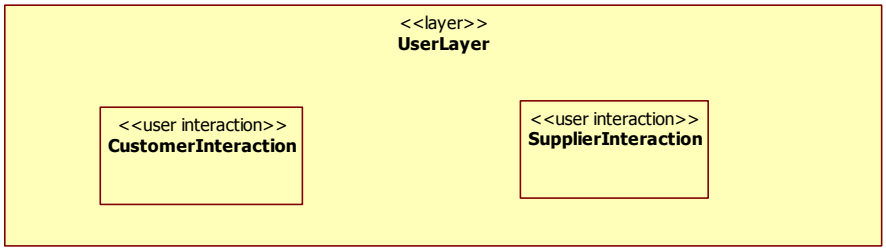
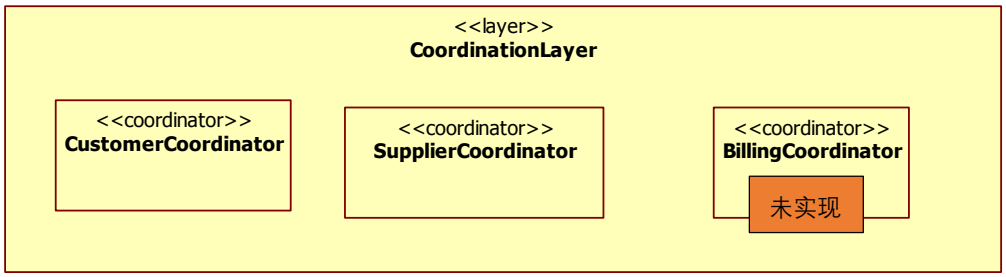


图 3.3“在线购物系统”的服务类和实体类

3.3 软件层次结构

最上层为用户层，主要提供整个系统用户交互的接口，为了使系统有较低的耦合度，并能进一步的拓展，在服务层与用户层之间加入中介者层，为系统引入中介者模式，客户可以通过中介者类来发现服务。同时避免了过多的服务模块互相交互引起模块独立性的降低。





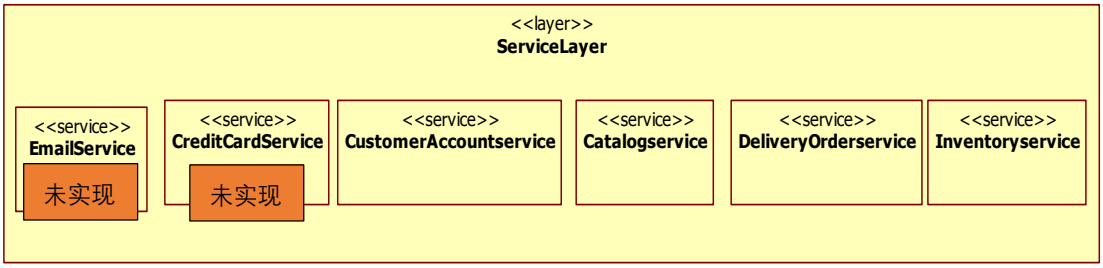
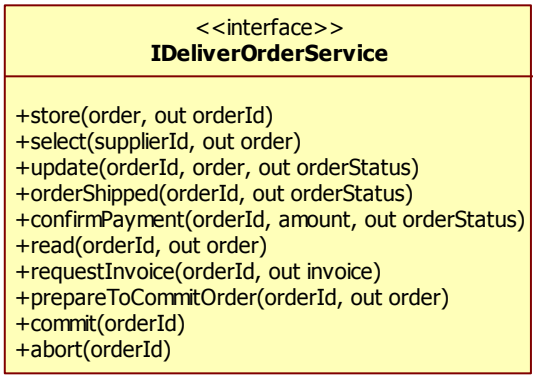


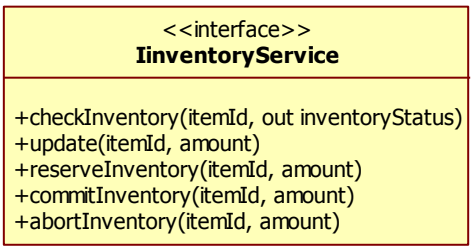
图 5.1 “在线购物系统”层次结构图

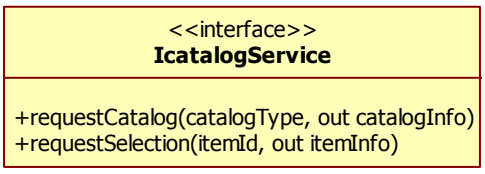
3.4 服务层与中介层接口详细设计

对系统服务层与中介层接口需要实现的方法构建详细的规约。(由于规模原因，未对服务层中的信用卡服务和邮件服务以及中介者层中的账户中间者进行构建)

3.4.1 服务接口







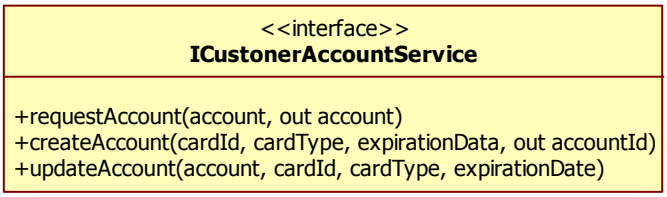
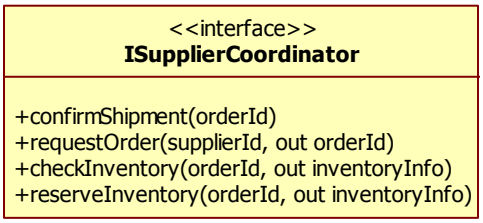


图 5.2 “在线购物系统”服务接口

3.4.2 中介者接口



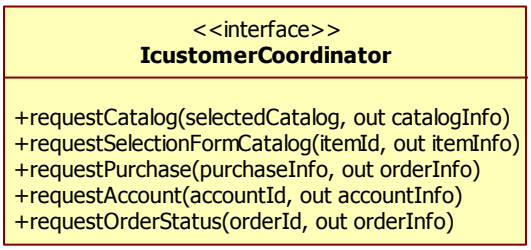


图 5.3 “在线购物系统”中介者接口

4 动态建模

对需求分析阶段提出的五个功能从系统内部模块的角度进行详尽的描述。

4.1 “浏览目录”用例的动态建模

表 4.1 浏览目录

B1：客户通过“客户交互”发出一个目录请求。 B2：“客户协调者”被实例化来帮助客户。在客户请求的基础上，“客户协调者”为客户选择一个目录来浏览。 B3：“客户协调者”向“目录服务”请求信息。 B4：“目录服务”发送目录信息给“客户协调者”。 B5：“客户协调者”把信息转发给“客户交互”。 B6：“客户交互”向客户显示目录信息。 B7：客户通过“客户交互”选择一个目录。 B8：“客户交互”传递请求给“客户协调者”。 B9：“客户协调者”向“目录服务”请求目录选择。 B10：“目录服务”确认目录商品的可用性并且发送商品价格给“客户协调者”。 B11：“客户协调者”转送信息给“客户交互”。 B12：“客户交互”向客户显示目录信息，包括商品价格和总价。

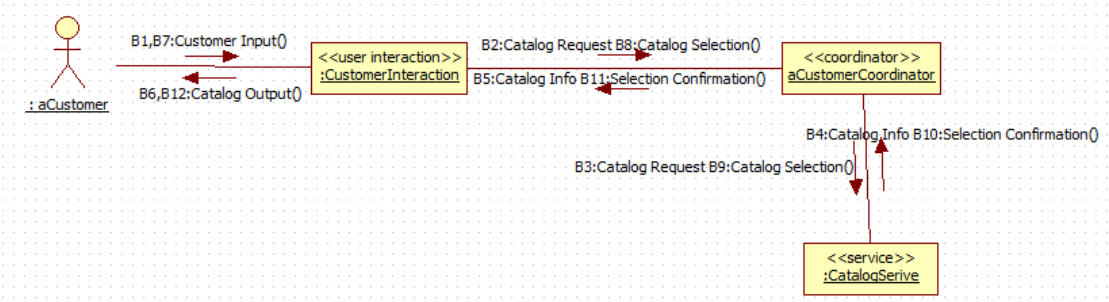


图 4.1 “浏览目录”通信图

4.2 “下单请求”用例的动态建模

表 4.2 下单请求

M1：客户向“客户交互”提出订单请求。 M2：“客户交互”将订单请求发送给“客户协调者”。 M3，M4：“客户协调者”发送账户请求给“客户账户服务”，并且接受账户信息，包括客户的信息卡详细信息。 M5：“客户协调者”向“信用卡服务”发送客户的信用卡信息和付款授权请求（这相当于一个“准备提交”的信息）。 M6：“信用卡服务”向“客户协调者”发送一个信用卡批准（这相当于“准备好提交”的消息）。

M7，M8：“客户协调者”发送订单请求给“配送订单服务”。
M9，M9a：“客户协调者”发送订单确认给“客户交互”，并且通过“电子邮件服务”向客户发送一封订单确认的邮件。
M10：“客户交互”向客户输出订单确认。

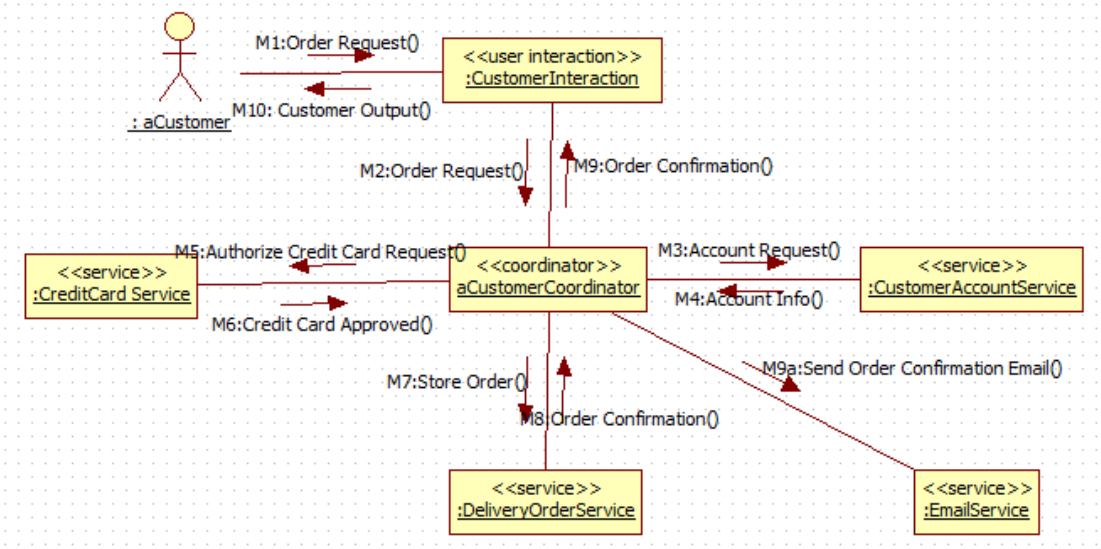


图 4.2 “下单请求”通信图

4.3 “处理配送订单”用例的动态建模

表 4.3 处理配送订单

D1：供应商请求一个新的配送订单。
D2：“供应商交互”向“供应商协调者”发送供应商的请求。
D3：“供应商协调者”请求“配送订单服务”选择一个配送订单。
D4：“配送订单服务”发送配送订单给“供应商协调者”。
D5：“供应商协调者”请求检查商品库存。
D6：“库存服务”返回商品信息。
D7：“供应商协调者”发送订单信息给“供应商交互”。
D8：“供应商交互”向供应商显示配送订单的信息。
D9：供应商请求系统在库存中保留商品。
D10：“供应商交互”发送供应商的请求给“供应商协调者”来保存库存。
D11：“供应商协调者”请求“库存服务”来保存库存中的商品（这相当于“准备提交”的信息）。
D12：“库存服务”向“供应商协调者”确认商品的保留（这相当于“准备好提交”的消息）。
D13：“供应商协调者”发送库存状态给“供应商交互”。
D14：“供应商交互”向供应商显示库存信息。

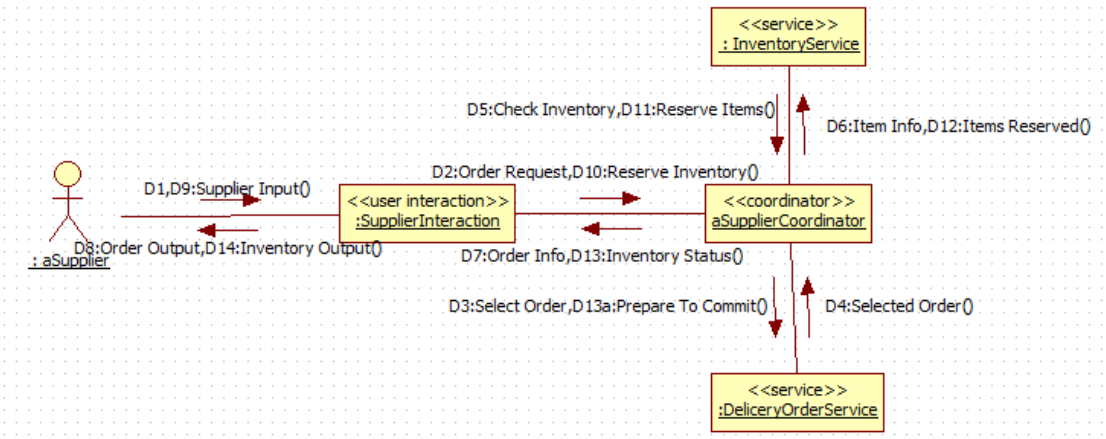


图 4.3 “处理配送订单”通信图

4.4“确认配送和给客户开账单”用例的动态建模

表 4.4 确认配送和给客户开账单

S1：供应商输入配送信息。 S2：“供应商交互”发送“准备好配送”的请求给“供应商协调者”。 S3：“供应商协调者”发送“订单准备好配送”的消息给“账单协调者”。 S4：“账单协调者”发送“准备提交”订单给“配送订单服务”。 S5：“配送订单服务”回复“准备好提交”的消息和发货单，包括订单号、账户号和总价。 S6，S7：“账单协调者”发送账户请求给“客户账户服务”，“客户账户服务”返回账户信息。 S8，S8a，S8b，S8c：“账单协调组”发送“提交收费”的消息给“信用卡服务”，发送“提交支付”的消息给“配送订单服务”，通过“电子邮件服务”发送确认邮件给客户，发送“账户已开账单”的消息给“客户协调者”。 S9，S10：“供应商协调者”发送“提交库存”的消息给“库存服务”，“库存服务”返回提交已完成。

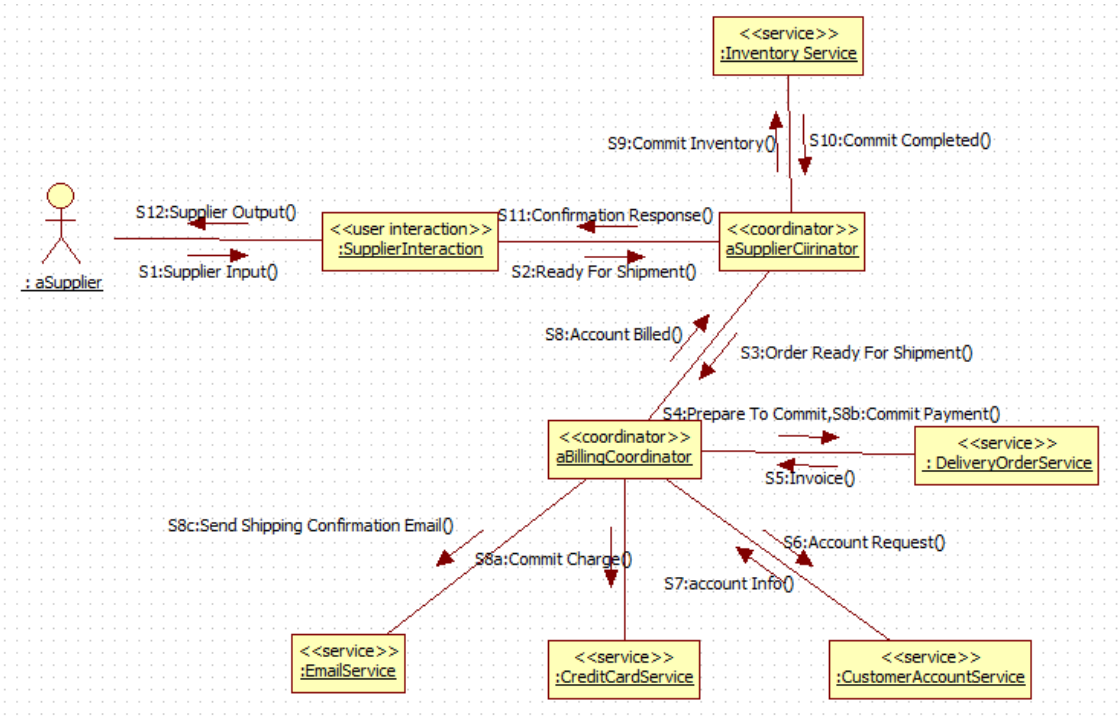


图 4.4 “确认配送和给客户开账单”通信图

4.5 “查看订单”用例的动态建模

表 4.5 查看订单

V1，V2：客户通过“客户交互”发出一个订单发货单的请求。 V3：“客户协调者”向“配送订单服务”发出一个订单请求。 V4：“配送订单服务”发送订单发货单信息给“客户协调者”。 V5：“客户协调者”转送信息给“客户交互”。 V6：“客户交互”向客户显示订单信息。

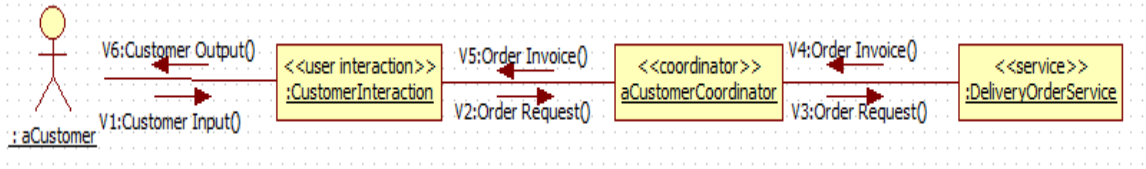


图 4.5 “查看订单”通信图

5 系统实现

（略）详情见附录部分

6 面向对象技术分析

1. **使用的面向对象原则**
   - 单一职责原则：除中介者类较为复杂，每一个类都具有且只有完全独立的符合自己定义的功能，都只有一个引起它变化的原因。同时为了进一步降低耦合，系统底层将属性和方法也进行了分离，即首先构造了基础的只含属性的实体类，并在逻辑上将其依照系统功能的需求封装在了四个服务中，而四个服务接口围绕自己需要实现的功能的各自构建了各自的方法。
   - 开放封闭原则：为了保障系统的扩展性，将系统的底层实体封装在服务类中，并引入了中介者模式，当新的功能需要被添加时，只需要实现对应的服务接口，进行拓展，并向中介者注册自己便可，底层的实体类与客户端的用户接口不必修改（甚至完全不会察觉系统的变化）。
   - 依赖倒置原则：可以看到，加入了中介者模式的系统上层模块减少了对下层的依赖，对下层的“服务接口”拓展并通过向中介者注册自己，甚至可以使用户层“发现”新的服务。
   - 接口隔离原则：使用了多个专门的服务接口构建系统的主体。

2. **使用的设计模式**
   - 中介者模式

7 总结提高

7.1 课程设计总结

这次设计首先共同商讨构建了初步的整体设计，之后是在搜索了大量资料，并参考了多本书籍后不断改进系统的情况下完成的。在这个资料收集与改进的过程中，原先怎么也记不住，理解不深刻的面向对象设计过程中的理念与工具都变的十分清晰，特别是依照改善之后的设计完成分工编码的过程中，深刻的体会了不同设计对之后工作的巨大影响，在参照了多份资料并通过书籍了解为什么这么做之后，编码过程变得极易分工，且各个独立编写的模块非常容易整合进系统，所以在很短时间就完成了这个近 40 个文件构成的系统的主体功能的编码工作。

小组成员贡献百分比：任俊州 34%，张鹏 33%，白涵 33%。

7.2 对本课程意见与建议

课程设计非常合理，结构清晰，建议可以适当尝试减少部分与高级软件工程课程重叠的纯工程部分内容。

附录：源码

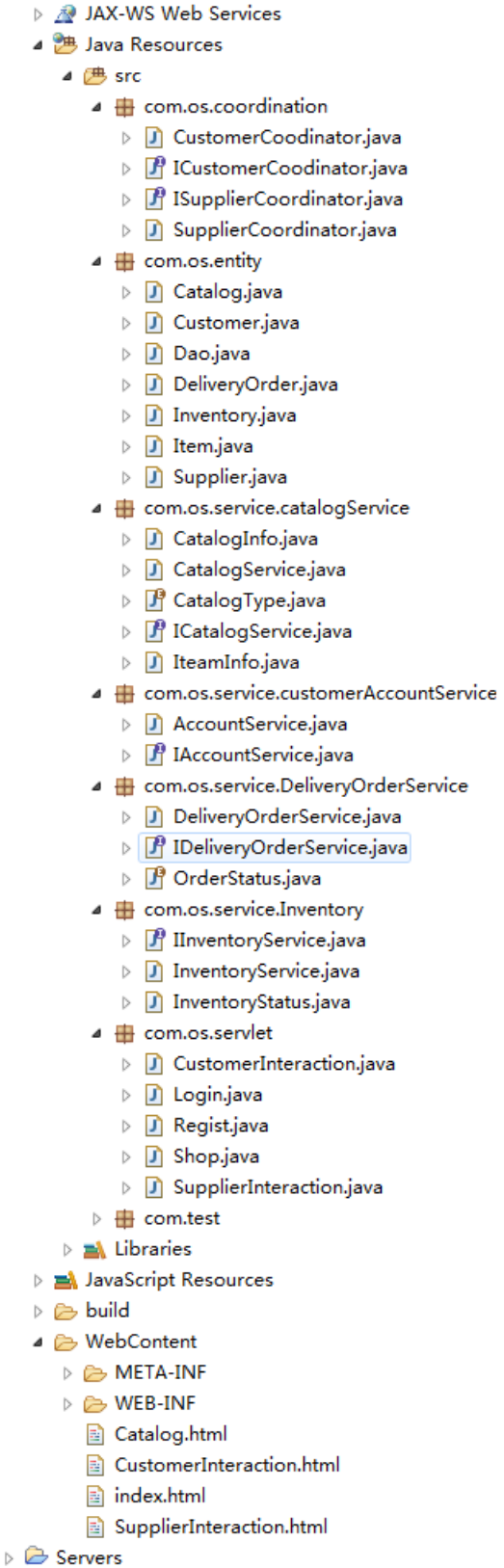


图 1 “在线购物系统” 代码结构

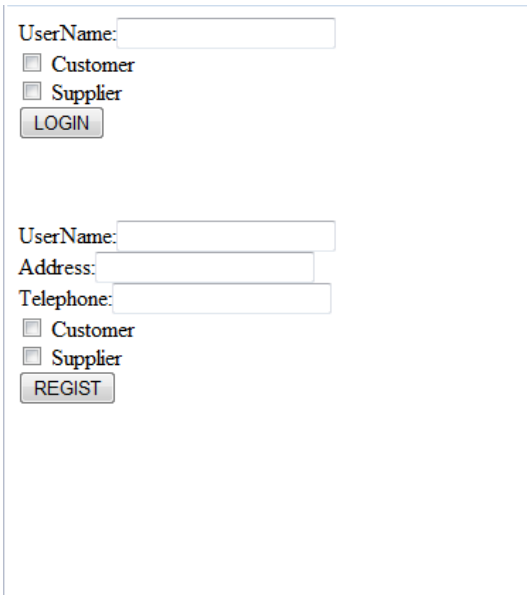


图 II “在线购物系统”登陆界面

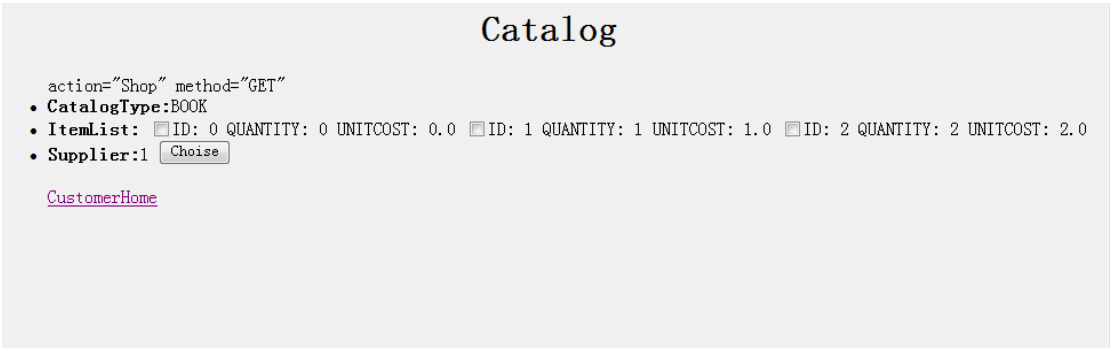


图 III “在线购物系统”商品目录界面

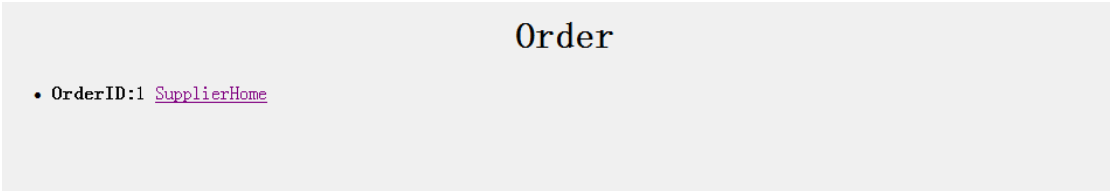


图 IV “在线购物系统”供应商订单处理页面

1. **实体类示例（消费者）**

```
public class Customer {
    int customerID;
    String customerName;
    String address;
    String telephoneNumber;

    public int getCostomerID() {
        return customerID;
    }
    public void setCostomerID(int costomerID) {
```

```
        this.customerID = costomerID;
    }
    public String getCustomerName() {
        return customerName;
    }
    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public String getTelephoneNumber() {
        return telephoneNumber;
    }
    public void setTelephoneNumber(String telephoneNumber) {
        this.telephoneNumber = telephoneNumber;
    }
}
```

2. 服务类示例（账户服务）

```
public class AccountService implements IAccountService {
    @Override
    public int createCustomerAccount(String UserName,String Telephone,String
Address) {
        // TODO Auto-generated method stub
        Customer customer = new Customer();
        customer.setCustomerName(UserName);
        customer.setTelephoneNumber(Telephone);
        customer.setAddress(Address);
        customer.setCostomerID(Dao.customerList.size());
        Dao.customerList.add(customer);
        return customer.getCostomerID();
        }
    public int createSupplierAccount(String UserName,String Telephone,String
Address) {
        // TODO Auto-generated method stub
        Supplier supplier = new Supplier();
        supplier.setSupplierName(UserName);
        supplier.setTelephoneNember(Telephone);
        supplier.setAddress(Address);
        supplier.setSupplierId(Dao.customerList.size());
        Dao.supplierList.add(supplier);
```

```
        return supplier.getSupplierId();
        }
    @Override
    public int requestCustomerAccount(String UserName) {
        // TODO Auto-generated method stub
        for (Customer ct : Dao.customerList) {
              if (ct.getCustomerName().equals(UserName))
              return ct.getCostomerID();
        }
        return -1;
    }
    @Override
    public int requestSupplierAccount(String UserName) {
        // TODO Auto-generated method stub
        for (Supplier sl : Dao.supplierList) {
              if (sl.getSupplierName().equals(UserName))
              return sl.getSupplierId();
        }
        return -1;
    }
}
```

3. 协调者类示例（客户协调者）

```
public class CustomerCoodinator implements ICustomerCoodinator {

    ICatalogService catalogService = new CatalogService();
    /*
     * 提交目录类型，返回该类型商品生成的目录
     * (non-Javadoc)
     * @see
com.os.coordination.ICustomerCoodinator#requestCatalog(com.os.service.catal
ogService.CatalogType)
     */
    @Override
    public Catalog requestCatalog(CatalogType c) {
        // TODO Auto-generated method stub
        return catalogService.requestCatalog(c);
    }

    /*
     * 提交用户选择的商品ID 返回用户购买的商品集合和信息
     * (non-Javadoc)
     * @see
com.os.coordination.ICustomerCoodinator#requestSelectionFromCatalog(int[])
```

```
     */
    @Override
    public HashSet<Item> requestSelectionFromCatalog(int[] itemId) {
        // TODO Auto-generated method stub
        return catalogService.requestSelection(itemId);
    }

    /*
     * 提交商品购买集合
     * 生成配送订单并返回
     * (non-Javadoc)
     * @see
com.os.coordination.ICustomerCoodinator#requesPurchase(java.util.HashSet)
     */
    @Override
    public DeliveryOrder requesPurchase(HashSet<Item> iteamlist) {
        // TODO Auto-generated method stub
        DeliveryOrder deliveryOrder = new DeliveryOrder();
        IDeliveryOrderService deliveryOrderService = new
DeliveryOrderService();
        deliveryOrder.setOrderId(Dao.deliveryOrderList.size());
        deliveryOrder.setItemList(iteamlist);

        deliveryOrderService.store(deliveryOrder);
        return deliveryOrder;
    }

    @Override
    public Customer requestAccount(int coutomerId) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public DeliveryOrder requseOrderStatus(int orderId) {
        // TODO Auto-generated method stub
        return null;
    }

}
```

4. 界面示例（登录）

```
/**
 * Servlet implementation class Login
```

```
 */
@WebServlet("/Login")
public class Login extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Login() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        AccountService cas =new AccountService();

        PrintWriter out = response.getWriter();

        if(request.getParameter("lcustomer")!=null){

    if(cas.requestCustomerAccount(request.getParameter("UserName"))!=-1){

    Dao.currentUser=cas.requestCustomerAccount(request.getParameter("UserNa
me"));//当前用户为x
            /***********跳转页面**************/
            request.setAttribute("name", "haiyun");
// 为request对象添加参数
                RequestDispatcher dispatcher =
request.getRequestDispatcher("CustomerInteraction.html");    // 使用req对象
获取RequestDispatcher对象
                dispatcher.forward(request, response);
                /************************/
            }
        else
            out.print("<script>alert(\"Login fail\")</script>");
        }
        else{
```

```
    if(cas.requestSupplierAccount(request.getParameter("UserName"))!=-1){

    Dao.currentUser=cas.requestSupplierAccount(request.getParameter("UserNa
me"));//当前用户为x
                /***********跳转页面**************/
                request.setAttribute("name", "haiyun");
// 为request对象添加参数
                RequestDispatcher dispatcher =
request.getRequestDispatcher("SupplierInteraction.html");    // 使用req对象
获取RequestDispatcher对象
                dispatcher.forward(request, response);
                /************************/
            }
            else{
                out.print("<script>alert(\"Login fail\")</script>");
            }
        }
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }

}
```