**软件系统详细设计文档**

**快递物流管理系统**

**第45组**

**陈之威**

**刘璇琳**

**王子安**

**张诗琦**

# 更新历史

|  |  |  |  |
| --- | --- | --- | --- |
| 修改人员 | 日期 | 变更原因 | 版本号 |
| 张诗琦 | 2015-10-30 | 创建文档框架 | V0.1 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[更新历史 2](#_Toc434059398)

[1. 引言 4](#_Toc434059399)

[1.1. 编制目的 4](#_Toc434059400)

[2. 产品概述 4](#_Toc434059401)

[3. 体系结构设计概述 4](#_Toc434059402)

[4. 结构视角 5](#_Toc434059403)

[4.1. 业务逻辑层的分解 5](#_Toc434059404)

[5. 依赖视角 5](#_Toc434059405)

# 引言

## 编制目的

本报告详细完成快递物流行业快递物流管理系统的详细设计，达到指导后续软件构造的目的，同时实现和测试人员及用户的沟通。

本报告面向开发人员、测试人员及最终用户而编写，是了解系统的导航。

# 产品概述

参考快递物流行业快递物流管理系统用例文档和快递物流行业快递物流管理系统软件规格说明中对产品的概括描述。

# 体系结构设计概述

请参考快递物流管理系统需求文档对体系结构设计的概述。

# 结构视角

## 业务逻辑层的分解

业务逻辑层的开发包图详见体系结构文档图3。

**4.1.10.Basebl 模块**

（1）模块概述

Basebl负责实现单据审批的功能性需求和非功能性需求，可参见快递物流系统需求规格说明文档和软件体系结构设计文档。

（2）整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如在展示曾和业务逻辑层之间添加businesslogicservice.base blservice.BaseBLService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了BaseController，这样BaseConroller会将对用户管理的业务逻辑处理委托给Base对象。BasePO是作为的成本常量的持久化对象被添加到设计模型中去的。

Basebl的设计模式如图：



Basebl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| BaseController | 负责实现公司运营成本管理界面所需要的服务 |
| Base | 成本常量的领域模型对象，拥有成本常量的信息，可以帮助完成成本常量管理界面所需要的服务 |

（3）模块内类接口规范

**BaseController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| BaseController.addBase | 语法 | public ResultMessage addBase(BaseVO vo) |
| 前置条件 | 已创建一个Base领域对象，添加的Base不重复 |
| 后置条件 | 调用Base领域的addBase方法 |
| BaseController.deletBase | 语法 | public ResultMessage deleteBase(String ID) |
| 前置条件 | 已创建一个Base领域对象，id符合输入规则 |
| 后置条件 | 调用Base领域的deleteBase方法 |
| BaseController.updateBase | 语法 | public ResultMessage updateBase (BaseVO base) |
| 前置条件 | 已创建一个Base领域对象 |
| 后置条件 | 调用Base领域的updateBase方法 |
| BaseController.getID() | 语法 | public String getID() |
| 前置条件 | 已经创建一个Base领域对象，界面提出添加base请求 |
| 后置条件 | 调用Base领域的getID方法 |
| BaseController.show(FindTypeBase baseType) | 语法 | public ArrayList<BaseVO vo>show(FindTypeBase baseType) |
| 前置条件 | 已经创建一个Base领域对象 |
| 后置条件 | 显示运营常量列表 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| BaseController.addBase | 添加一个常量 | |
| BaseController.deletBase | 删除一个常量 | |
| BaseController.updateBase | 更新一个常量 | |
| BaseController.show | 显示常量列表 | |
| BaseController.getID() | 获得一个常量编号 | |

**Base接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Base.addBase | 语法 | public ResultMessage addBase(BaseVO vo) |
| 前置条件 | 添加的Base不重复 |
| 后置条件 | 添加Base |
| Base.deleteBase | 语法 | public ResultMessage deleteBase  (String id) |
| 前置条件 | id符合输入规则 |
| 后置条件 | 查找相应的Base，根据输入的id删除该常量 |
| Base.getID | 语法 | public String getID() |
| 前置条件 | 界面提出添加Base申请 |
| 后置条件 | 返回一个新增Base的ID |
| Base.show | 语法 | public ArrayList<BaseVO>show(FindTypeBase baseType) |
| 前置条件 | 有可以显示在的常量 |
| 后置条件 | 显示当前类型的常量列表 |
| Base.updateBase | 语法 | public ResultMessage updateBase(BaseVO base) |
| 前置条件 | Base信息符合输入规则 |
| 后置条件 | 查找相应的Base，更改Base的信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFacory.getBaseDataBase | 得到Base的服务的引用 | |
| BaseDateService.find  (BasePO po) | 查找单一持久化对象 | |
| BaseDataService.insert  (BasePO po) | 插入单一持久化对象 | |
| BaseDataService.delete  (BasePO po) | 删除单一持久化对象 | |
| BaseDataService.update  (UserPO po) | 更新单一持久化对象 | |

（4）业务逻辑层的动态模型

下列图片表明了快递物流管理系统中，添加、删除、修改一个成本常量时时，常量业务逻辑处理的相关对象之间的协作。然后是它们各自的状态图。

   （5）公司运营常量制定逻辑层的设计原理

利用委托式的控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

**4.1.11. Receiptbl模块**

（1）模块概述

Receiptbl负责实现单据审批的功能性需求和非功能性需求，可参见快递物流系统需求规格说明文档和软件体系结构设计文档。

（2）整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如在展示曾和业务逻辑层之间添加businesslogicservice.receiptblservice.ReceiptBLService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ReceiptController，这样ReceiptConroller会将对用户管理的业务逻辑处理委托给User对象。ReceiptPO是作为用户的持久化对象被添加到设计模型中去的。

Receiptbl的设计模式如图：



Receiptbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| ReceiptblController | 负责实现审批管理界面所需要的服务 |
| Receiptbl | 审批的领域模型对象，拥有审批单子的信息，可以帮助完成审批管理界面所需要的服务 |

1. 模块内部类的接口规范

**ReceiptblController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| ReceiptblController.checkReceipt | 语法 | public ResultMessage checkReceipt (ReceiptVO vos) |
| 前置条件 | 已经创建一个Receipt领域对象,有需要审核的单据 |
| 后置条件 | 调用Receipt领域对象的checkReceipt方法 |
| ReceiptblController.updateReceipt | 语法 | public ResultMessage updateReceipt  (ReceiptVO receiptVO,ReceiptType receiptType) |
| 前置条件 | 已经创建一个Receipt领域对象,已经查看到单据内容 |
| 后置条件 | 调用Receipt领域对象的updateReceipt方法 |
| ReceiptBLController.passReceipt | 语法 | public ResultMessage passReceipt  (ArrayList<ReceiptVO>VOs,ReceiptType receiptTypes) |
| 前置条件 | 已经创建一个Receipt领域对象,确认单据内容没有错误 |
| 后置条件 | 调用Receipt领域对象的passReceipt方法 |
| ReceiptBLController.dontPassReceipt | 语法 | public ResultMessage dontPassReceipt  (ArrayList<ReceiptVO>VOs,ReceiptType receiptTypes) |
| 前置条件 | 已经创建一个Receipt领域对象,确认单据内容有错误 |
| 后置条件 | 调用Receipt领域对象的dontPassReceipt方法 |
| ReceiptBLController.showReceipt | 语法 | public ArrayList<ReceiptVO> showReceipt(ReceiptState receiptStates); |
| 前置条件 | 已经创建一个Receipt领域对象,有可以显示的单据 |
| 后置条件 | 调用Receipt领域对象的showReceipt方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Receipt. checkReceipt | 查看单据内容 | |
| Receipt.updateReceipt | 修改单据内容 | |
| Receipt.passReceipt | 确认单据内容没有错误，通过单据审批 | |
| Receipt.dontPassReceipt | 确认单据内容有错误，不通过单据审批 | |
| Receipt.showReceipt | 显示单据列表 | |

**Receipt的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| Receipt.checkReceipt | | 语法 | public ResultMessage checkReceipt (PersistentObject record) |
| 前置条件 | 有需要审核的单据 |
| 后置条件 | 查看单据内容 |
| Receipt. updateReceipt | | 语法 | public ResultMessage updateReceipt  (ArrayList<ReceiptVO>VOs,ReceiptType receiptTypes) |
| 前置条件 | 已经查看到单据内容 |
| 后置条件 | 修改单据内容 |
| Receipt. passReceipt | | 语法 | public ResultMessage passReceipt  (ArrayList<ReceiptVO>VOs,ReceiptType receiptTypes) |
| 前置条件 | 确认单据内容没有错误 |
| 后置条件 | 通过单据，持久化更新涉及的对象的数据 |
| Receipt.dontPassReceipt | 语法 | | public ResultMessage dontPassReceipt  (ArrayList<ReceiptVO>VOs,ReceiptType receiptTypes) |
| 前置条件 | | 单据内容有错误 |
| 后置条件 | | 不通过单据，持久化更新涉及的对象的数据 |
| Receipt.showReceipt | 语法 | | public ArrayList<ReceiptVO> showReceipt(ReceiptState receiptStates); |
| 前置条件 | | 有可以进行显示的单据 |
| 后置条件 | | 显示单据列表 |
| 需要的服务（需接口） | | | |
| 服务名 | | 服务 | |
| ReceiptDataService.  Insert(ReceiptPO po) | | 插入单一持久化对象 | |
| ReceiptDataService.update(ReceiptPO po) | | 修改单一持久化对象 | |
| ReceiptDataService.find(String id) | | 查找单一持久化对象 | |
| ReceiptDataService.show(ReceiptState receiptState) | | 显示单一持久化对象 | |
| ReceiptDataService.delete(String id) | | 删除单一持久化对象 | |
| ReceiptDataService.init() | | 初始化单一持久对象 | |

1. 业务逻辑层的动态模型

下图为单据处理的顺序图和状态图。

   

（5）公司运营常量制定逻辑层的设计原理

利用委托式的控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

**4.1.12. Userbl模块**

（1）模块概述

Userbl负责实现用户管理的功能性需求和非功能性需求，可参见快递物流系统需求规格说明文档和软件体系结构设计文档。

（2）整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如在展示曾和业务逻辑层之间添加businesslogicservice.userblservice.UserBLService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了UserController，这样UserConroller会将对用户管理的业务逻辑处理委托给User对象。UserPO是作为用户的持久化对象被添加到设计模型中去的。

Usebl的设计模式如图：



Userbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| UserController | 负责实现用户管理界面所需要的服务 |
| User | 用户的领域模型对象，拥有用户的信息，可以帮助完成用户管理界面所需要的服务 |

（3）模块内类接口规范

**UserController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserController.login | 语法 | public ResultMessage login(String username, String password); |
| 前置条件 | 已创建一个User领域对象，password符合输入规则 |
| 后置条件 | 调用User领域的login方法 |
| UserController.addUser | 语法 | public ResultMessage addUser  (String username, String password, UserIdentity position) |
| 前置条件 | 已创建一个User领域对象，添加的User不重复 |
| 后置条件 | 调用User领域的addUser方法 |
| UserController.deletUser | 语法 | public ResultMessage deleteUser  (String username) |
| 前置条件 | 已创建一个User领域对象，id符合输入规则 |
| 后置条件 | 调用User领域的deleteUser方法 |
| UserController.updateUser | 语法 | public ResultMessage updateUser(String username, String password, UserIdentity newPosition, UserVO user) |
| 前置条件 | 已创建一个User领域对象，用户名、密码、职位符合输入规则 |
| 后置条件 | 调用User领域的updateUser方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| UserController.login | 用户登录 | |
| UserController.addUser | 添加一个用户 | |
| UserController.deletUser | 删除一个用户 | |
| UserController.updateUser | 更新一个用户 | |

**User接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| User.login | 语法 | public ResultMessage login(String username, String password); |
| 前置条件 | password符合输入规则 |
| 后置条件 | 查找是否存在相应的User，根据输入的password返回登录验证的结果 |
| User.addUser | 语法 | public ResultMessage addUser  (String username, String password, UserIdentity position) |
| 前置条件 | 添加的User不重复 |
| 后置条件 | 添加User |
| User.deletUser | 语法 | public ResultMessage deleteUser  (String username) |
| 前置条件 | id符合输入规则 |
| 后置条件 | 查找相应的User，根据输入的id删除该用户 |
| User.updateUser | 语法 | public ResultMessage updateUser(String username, String password, UserIdentity newPosition, UserVO user) |
| 前置条件 | 用户名、密码、职位符合输入规则 |
| 后置条件 | 查找相应的User，根据输入的id，更改该用户的权限 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFacory.getUserDataBase | 得到User的服务的引用 | |
| UserDateService.find  (UserPO po) | 查找单一持久化对象 | |
| UserDataService.insert  (UserPO po) | 插入单一持久化对象 | |
| UserDataService.delete  (UserPO po) | 删除单一持久化对象 | |
| userDataService.update  (UserPO po) | 更新单一持久化对象 | |

（4）业务逻辑层的动态模型

下列图片表明了快递物流管理系统中，用户登录以及添加、删除、修改一个用户时，用户业务逻辑处理的相关对象之间的协作。然后是它们各自的状态图。



  

（5）用户逻辑层的设计原理

利用委托式的控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

**4.1.13. Organizationbl模块**

（1）模块概述

Organizationbl负责实现机构管理的功能性需求和非功能性需求，可参见快递物流系统需求规格说明文档和软件体系结构设计文档。

（2）整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如在展示层和业务逻辑层之间添加businesslogicservice.organizationblservice.OrganizationBLService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了OrganizationController，这样OrganizationConroller会将对机构管理的业务逻辑处理委托给Organization对象。OrganizationPO是作为机构的持久化对象被添加到设计模型中去的。

Organizationbl的设计模式如图：



Organizationbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| OrganizationController | 负责实现机构管理界面所需要的服务 |
| Organization | 机构的领域模型对象，拥有机构的信息，可以帮助完成机构管理界面所需要的服务 |

（3）模块内类接口规范

**OrganizationController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| OrganizationController.addOrganization | 语法 | public ResultMessage addOrganization (OrganizationVO vo) |
| 前置条件 | 已创建一个Organization领域对象，添加的Organization不重复 |
| 后置条件 | 调用Organization领域的addOrganization方法 |
| OrganizationController.deletOrganization | 语法 | public ResultMessage deleteOrganization (String ID) |
| 前置条件 | 已创建一个Organization领域对象，id符合输入规则 |
| 后置条件 | 调用Organization领域的deleteOrganization方法 |
| OrganizationController.updateOrganization | 语法 | public ResultMessage updateOrganization (OrganizationVO Organization) |
| 前置条件 | 已创建一个Organization领域对象 |
| 后置条件 | 调用Organization领域的updateOrganization方法 |
| OrganizationController.getID() | 语法 | public String getID() |
| 前置条件 | 已经创建一个Organization领域对象，界面提出添加Organization请求 |
| 后置条件 | 调用Organization领域的getID方法 |
| OrganizationController.show() | 语法 | public ArrayList<OrganizationVO vo>show() |
| 前置条件 | 已经创建一个Organization领域对象 |
| 后置条件 | 显示公司机构列表 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| OrganizationController.addOrganization | 添加一个机构 | |
| OrganizationController.deleteOrganization | 删除一个机构 | |
| OrganizationController.updateOrganization | 更新一个机构 | |
| OrganizationController.show | 显示机构列表 | |
| OrganizationController.getID | 获得一个机构编号 | |

**Organization接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Organization.addOrganization | 语法 | public ResultMessage addOrganization (OrganizationVO vo) |
| 前置条件 | 添加的Organization不重复 |
| 后置条件 | 添加Organization |
| Organization.deletOrganization | 语法 | public ResultMessage deleteOrganization (String id) |
| 前置条件 | id符合输入规则 |
| 后置条件 | 查找相应的Organization，根据输入的id删除该机构 |
| Organization.getID | 语法 | public String getID() |
| 前置条件 | 界面提出添加Organization申请 |
| 后置条件 | 返回一个新增Organization的ID |
| Organization.show | 语法 | public ArrayList<OrganizationVO>show() |
| 前置条件 | 有可以显示的机构 |
| 后置条件 | 显示机构列表 |
| Organization.updateOrganization | 语法 | public ResultMessage updateBase(OrganizationVO Organization) |
| 前置条件 | Organization信息符合输入规则 |
| 后置条件 | 查找相应的Organization，更改Organization的信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFacory.getOrganizationDataBase | 得到Organization的服务的引用 | |
| OrganizationDateService.find  (OrganizationPO po) | 查找单一持久化对象 | |
| OrganizationDataService.insert(OrganizationPO po) | 插入单一持久化对象 | |
| OrganizationDataService.delete(OrganizationPO po) | 删除单一持久化对象 | |
| OrganizationDataService.update(OrganizationPO po) | 更新单一持久化对象 | |

（4）业务逻辑层的动态模型

下列图片表明了快递物流管理系统中，添加、删除、修改一个机构时，机构管理业务逻辑处理的相关对象之间的协作。然后是它们各自的状态图。

   



（5）用户逻辑层的设计原理

利用委托式的控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

# 依赖视角

下图是客户端和服务器端各自的包之间的依赖关系。