Departmentbl模块的静态结构和动态行为

(1)模块概述

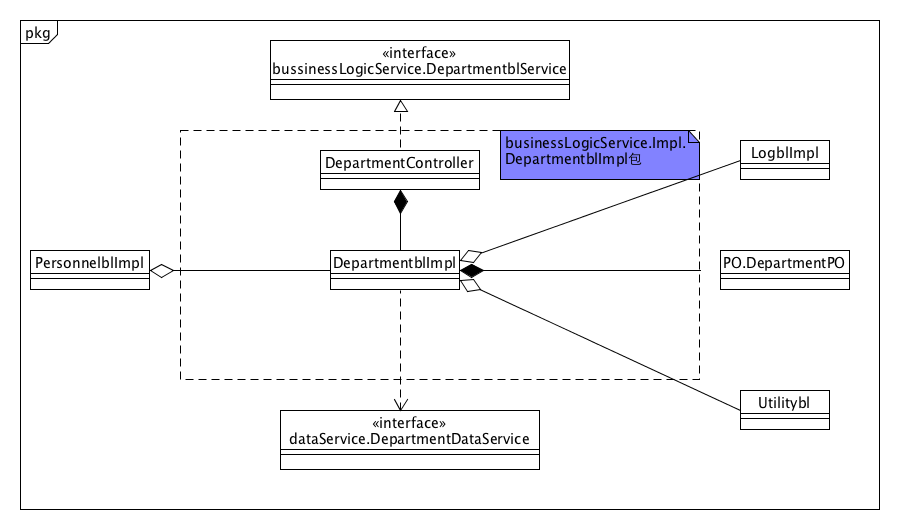
Departmentbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Departmentbl模块的职责和接口参见软件体系结构描述文档。

(2)整体结构

根据体系结构的设计，系统分为展示层、业务逻辑层和数据层，并在展示层和业务逻辑层之间添加bussinssLogicServi ce.DepartmentblService接口、在业务逻辑层和数据层之间添加dataService.DepartmentDataService接口，从而增加了系统的灵活性。我们增加了DepartmentController从而隔离了业务逻辑职责和逻辑控制职责，这样DepartmentController会将业务逻辑处理委托给Departmentbl对象。DepartmentPO作为用户信息的持久化对象被添加到设计模型中去的。LogImpl用来进行对管理用户操作时产生的系统日志的记录。

Departmentbl模块的设计如图所示。



Departmentbl模块的各个类的职责如下表所示。

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应于登陆界面所需要的服务 |
| DepartmentController | 负责实现用户管理界面所需要的服务 |
| DepartmentblImpl | 用户管理领域的模型对象，可以帮助完成用户管理界面所需要的服务 |

(3)模块内部类的接口规范

DepartmentController和Departmentbl的接口规范如下表所示。

DepartmentController的接口规范

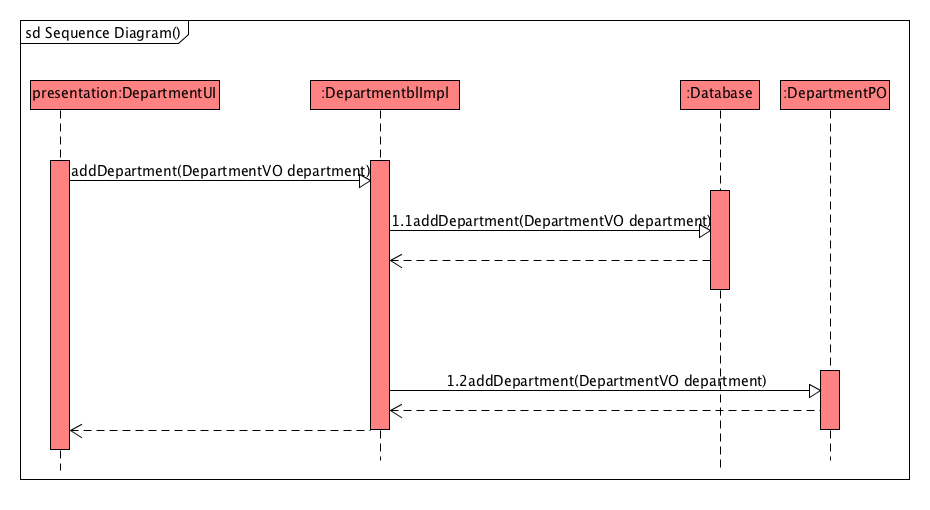
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| DepartmentController.getDepartmentInfo | 语法 | Public DepartmentPO getDepartInfo(long id) |
| 前置条件 | 无 |
| 后置条件 | 查找是否存在相应的Department，如果存在返回相应的Department信息 |
| DepartmentController.deleteDepartment | 语法 | Public Boolean deleteDepartment(long id) |
| 前置条件 | 无 |
| 后置条件 | 查找是否存在相应的Department， 如果删除成功返回true，否则返回false |
| DepartmentController.changeDepartment | 语法 | Public Boolean changeDepartment(DepartmentVO Department) |
| 前置条件 | 无 |
| 后置条件 | 根据id查找Department，并修改相应的数据，如果修改成功返回true，否则返回false |
| DepartmentController.addDepartment | 语法 | Public Boolean addDepartment(DepartmentVO Department) |
| 前置条件 | 无 |
| 后置条件 | 如果增加成功返回true，否则返回false |
| DepartmentController. addCity | 语法 | Public Boolean addCity(CityVO city); |
| 前置条件 | 传入的城市信息符合规范 |
| 后置条件 | 如果增加成功返回true，否则返回false |
| DepartmentController. findCity | 语法 | Public CityVO findCity(long id); |
| 前置条件 | 系统存在相应的id |
| 后置条件 | 返回城市信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Department.getDepartmentInfo(long id) | 根据id得到一个机构对象的信息 | |
| Department.deleteDepartment(long id) | 根据id删除一个机构对象 | |
| Department.changeDepartment(DepartmentVO Department) | 修改一个机构对象 | |
| Department.addDepartment(DepartmentVO Department) | 新增一个机构对象 | |
| Department. addCity(CityVO city) | 新增一个城市对象 | |
| Department. findCity(long id) | 根据id得到一个城市对象的信息 | |

Departmentbl的接口规范

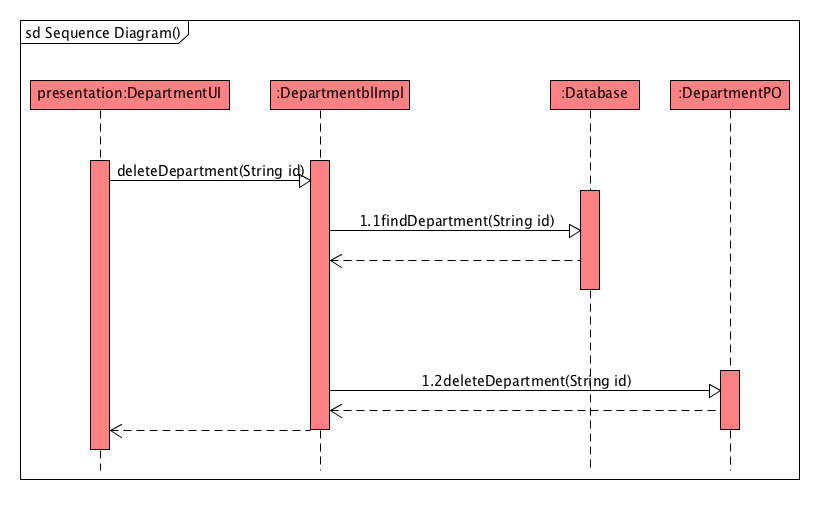
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Department.getDepartmentInfo | 语法 | Public DepartmentPO getDepartInfo(long id) |
| 前置条件 | 无 |
| 后置条件 | 查找是否存在相应的Department，如果存在返回相应的Department信息 |
| Department.deleteDepartment | 语法 | Public Boolean deleteDepartment(long id) |
| 前置条件 | 无 |
| 后置条件 | 查找是否存在相应的Department， 如果删除成功返回true，否则返回false |
| Department.changeDepartment | 语法 | Public Boolean changeDepartment(DepartmentVO Department) |
| 前置条件 | 无 |
| 后置条件 | 根据id查找Department，并修改相应的数据，如果修改成功返回true，否则返回false |
| Department.addDepartment | 语法 | Public Boolean addDepartment(DepartmentVO Department) |
| 前置条件 | 无 |
| 后置条件 | 如果增加成功返回true，否则返回false |
| Department. addCity | 语法 | Public Boolean addCity(CityVO city); |
| 前置条件 | 传入的城市信息符合规范 |
| 后置条件 | 如果增加成功返回true，否则返回false |
| Department. findCity | 语法 | Public CityVO findCity(long id); |
| 前置条件 | 系统存在相应的id |
| 后置条件 | 返回城市信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DepartmentDataService  .addDepartment(DepartmentPo department)：boolean | 增加机构的单一持久化对象 | |
| DepartmentDataService  .findDepartment(long id):DepartmentPo | 根据id查找机构的单一持久化对象并返回 | |
| DepartmentDataService  .deleteDepartment(long id):boolean | 根据id删除机构的单一持久化对象 | |
| DepartmentDataService  .updateDepartment(DepartmentPo department):boolean | 查找相应的机构持久化对象并替换成传入的机构持久化对象 | |
| DepartmentDataService  .addCity(CityPo city):boolean | 增加城市持久化对象 | |
| DepartmentDataService  .findCity(long id):CityPO | 根据id返回城市持久化对象 | |

(4)业务逻辑层的动态模型

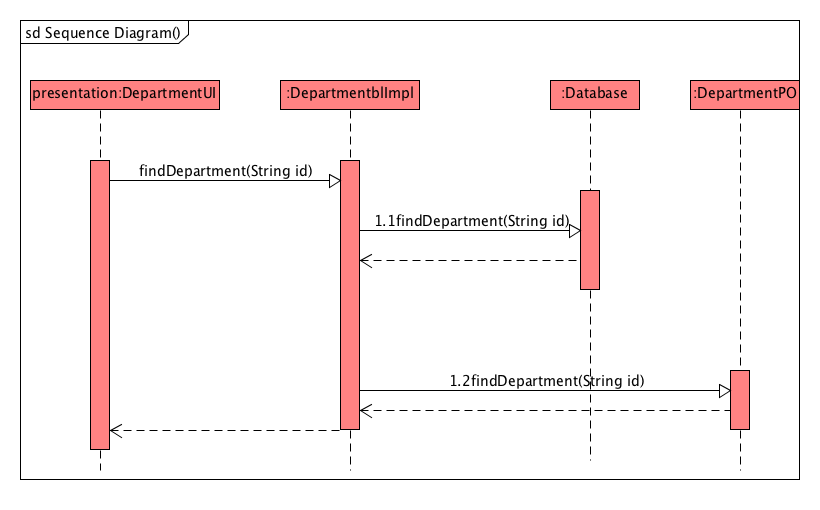
下图表示了物流管理系统中，当用户选择增加机构信息时，Departmentbl处理的相关对象的协作。



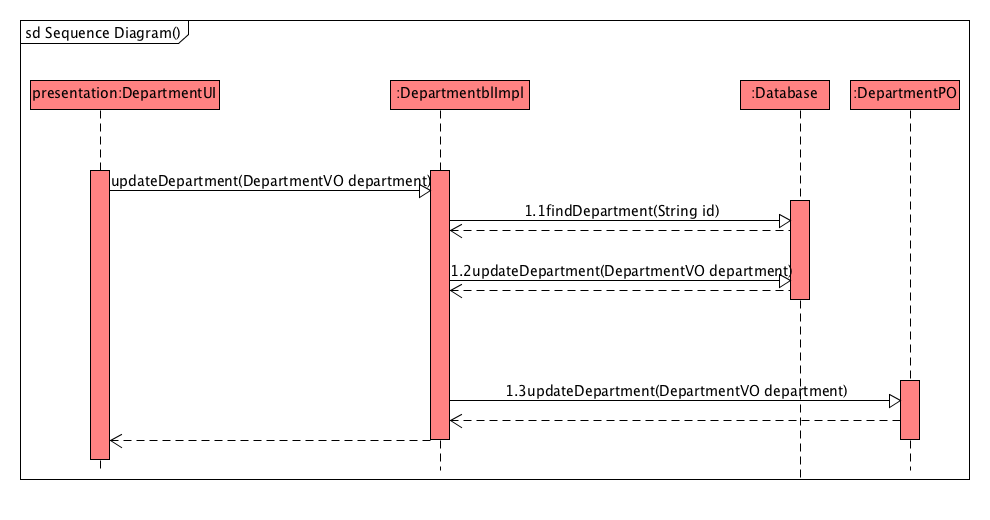
下图表示了物流管理系统中，当用户输入要删除的机构id时，Departmentbl处理的相关对象的协作。



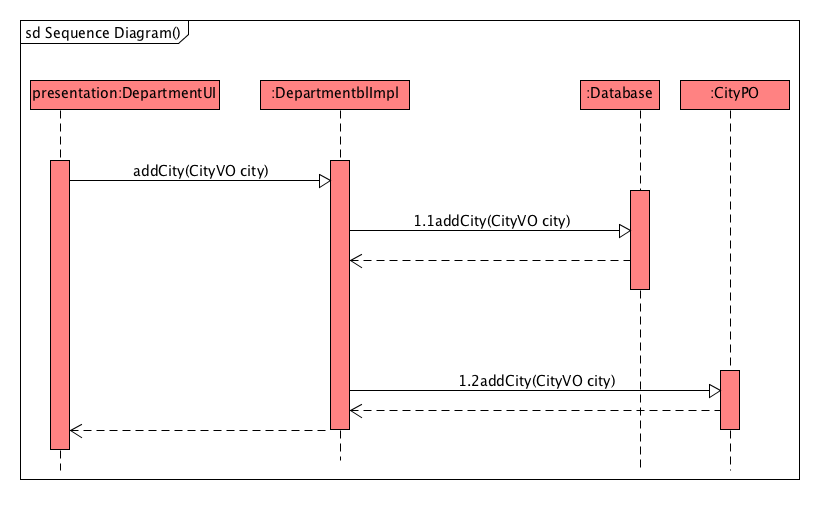
下图表示了物流管理系统中，当用户输入要查询的机构id时，Departmentbl处理的相关对象的协作。



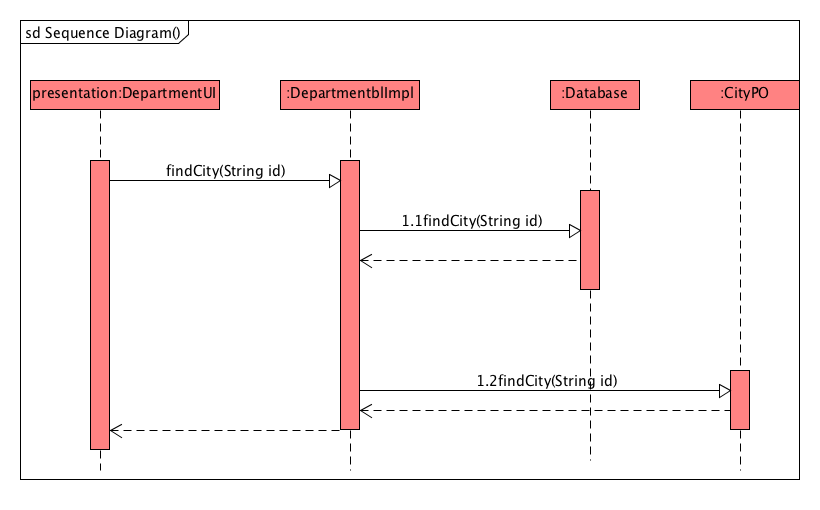
下图表示了物流管理系统中，当用户选择修改机构信息时，Departmentbl处理的相关对象的协作。



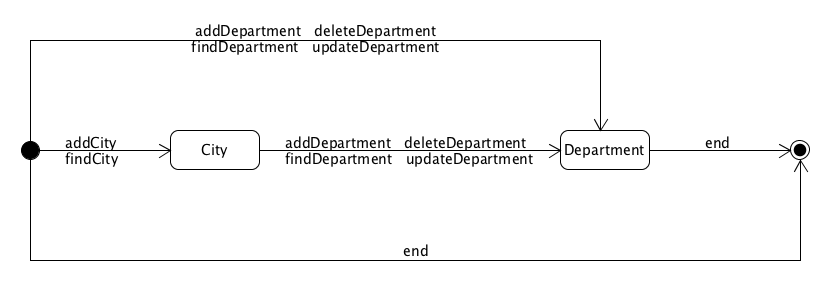
下图表示了物流管理系统中，当用户选择增加城市信息时，Departmentbl处理的相关对象的协作。



下图表示了物流管理系统中，当用户输入要查询的城市id时，Departmentbl处理的相关对象的协作。



如下图所示的状态图描述了Departmentbl对象的生存期内的状态序列、引起转移的事件以及因状态转移而伴随的动作。



(5)业务逻辑层的设计原理

利用委托式控制风格，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。