userbl模块的静态结构和动态行为

(1)模块概述

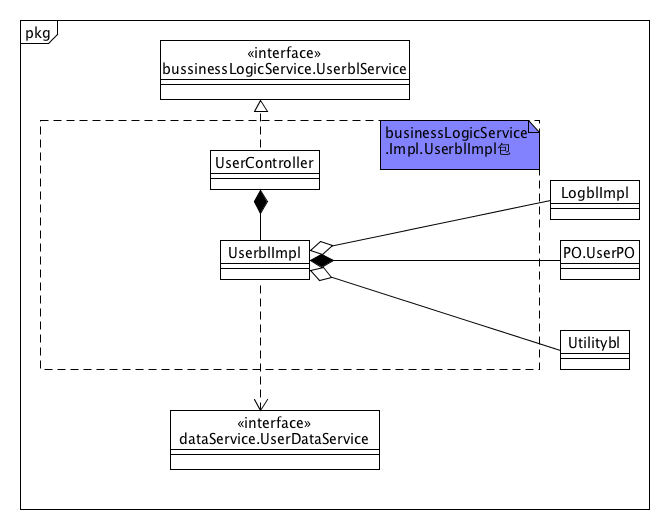
userbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

userbl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bussinssLogicSerivce.UserblService接口、在业务逻辑层和数据层之间添加dataService.UserDataService接口，从而增加了系统的灵活性。我们增加了UserController从而隔离了业务逻辑职责和逻辑控制职责，这样UserController会将业务逻辑处理委托给Userbl对象。UserPO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。

Userbl模块的设计如图所示。



Userbl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 |
| UserController | 负责实现用户管理界面所需要的服务 |
| UserblImpl | 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

UserController和Userbl的接口规范如下表所示。

UserController的接口规范

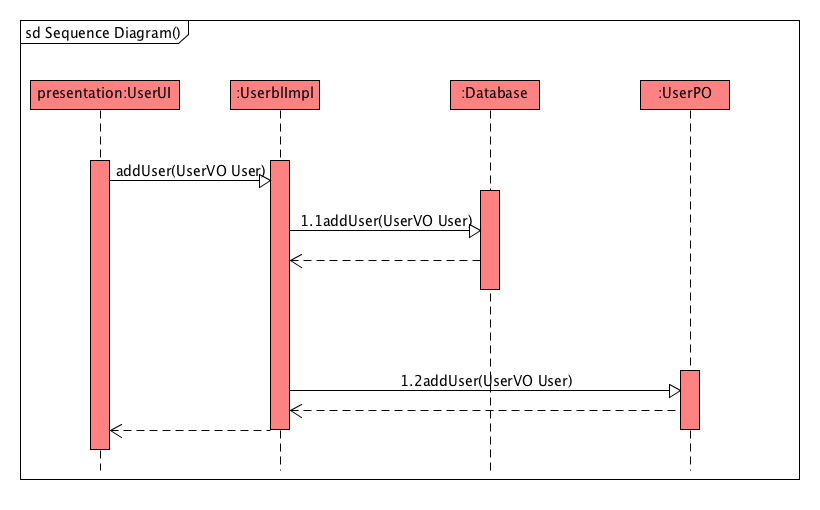
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserController.login | 语法 | Public String login(long id, String password) |
| 前置条件 | Password符合输入规则 |
| 后置条件 | 查找是否存在相应的User，根据输入的password返回输入的结果 |
| UserController.getUserInfo | 语法 | Public UserPO getUserInfo(long id) |
| 前置条件 | 无 |
| 后置条件 | 查找是否存在相应的user，如果存在返回相应的user信息 |
| UserController.deleteUser | 语法 | Public Boolean deleteUser(long id) |
| 前置条件 | 无 |
| 后置条件 | 查找是否存在相应的user， 如果删除成功返回true，否则返回false |
| UserController.changeUser | 语法 | Public Boolean changeUser(UserVO user) |
| 前置条件 | 无 |
| 后置条件 | 根据id查找user，并修改相应的数据，如果修改成功返回true，否则返回false |
| UserController.addUser | 语法 | Public Boolean addUser(UserVO user) |
| 前置条件 | 无 |
| 后置条件 | 如果增加成功返回true，否则返回false |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| User. login(long id, String password) | 登录管理用户界面 | |
| User.getUserInfo(long id) | 根据输入的id得到一个用户信息 | |
| User. deleteUser(long id) | 根据输入的id删除一个用户信息 | |
| User. changeUser(UserVO user) | 根据输入的信息修改一个用户信息 | |
| User. addUser(UserVO user) | 根据输入的信息增加一个用户信息 | |

Userbl的接口规范

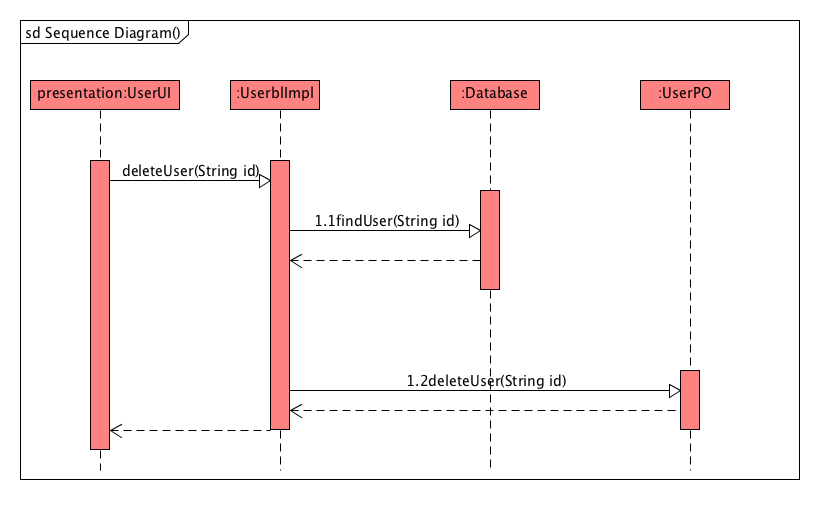
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| User.login | 语法 | Public String login(long id, String password) |
| 前置条件 | Password符合输入规则 |
| 后置条件 | 查找是否存在相应的User，根据输入的password返回输入的结果 |
| User.getUserInfo | 语法 | Public UserPO getUserInfo(long id) |
| 前置条件 | 无 |
| 后置条件 | 查找是否存在相应的user，如果存在返回相应的user信息 |
| User.deleteUser | 语法 | Public Boolean deleteUser(long id) |
| 前置条件 | 无 |
| 后置条件 | 查找是否存在相应的user， 如果删除成功返回true，否则返回false |
| User.changeUser | 语法 | Public Boolean changeUser(UserVO user) |
| 前置条件 | 无 |
| 后置条件 | 根据id查找user，并修改相应的数据，如果修改成功返回true，否则返回false |
| User.addUser | 语法 | Public Boolean addUser(UserVO user) |
| 前置条件 | 无 |
| 后置条件 | 如果增加成功返回true，否则返回false |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| UserDataService  .findUser(long id):UserPO | 根据输入的id返回一个用户持久化对象 | |
| UserDataService  .deleteUser(long id):boolean | 根据输入的id删除一个用户持久化对象 | |
| UserDataService  .updateUser(UserPo user):boolean | 根据输入的信息修改原有的用户持久化对象 | |
| UserDataService  .addUser(UserPo user):boolean | 根据输入的信息增加用户持久化对象 | |

(4)业务逻辑层的动态模型

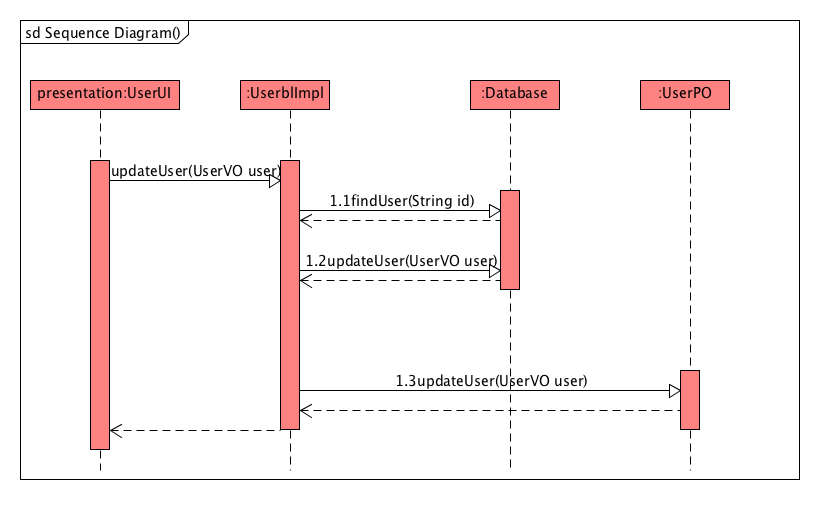
下图表示了物流管理系统中，当用户输入要增加的用户信息时，Userbl处理的相关对象的协作。



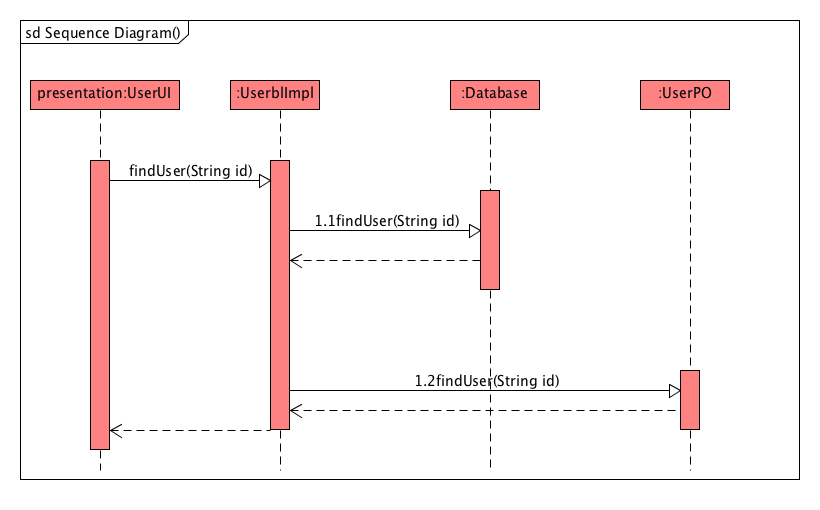
下图表示了物流管理系统中，当用户输入要删除的用户id时，Userbl处理的相关对象的协作。



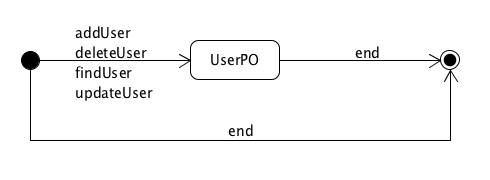
下图表示了物流管理系统中，当用户输入要修改的用户信息时，Userbl处理的相关对象的协作。



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



如下图所示的状态图描述了Userbl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。