2018 年春季

图书馆选座系统

软件设计文档

指导老师：王青

组长：　黄敏怡　15331116

组员：　胡子昂　15331111

李沁航　15331159

林彬彬　15331194

汪睿琪　15331293

联系方式：sysuhmy@163.com

目录

一、 软件设计技术

1 图书馆选座系统概述

1.1 问题陈述

高校图书馆有着良好的学习环境，但座位却是有限。而在期中期末等时候，随着对座位的需求增大，甚至出现了占座等浪费资源的行为，导致图书馆中座位供不应求的恶行循环，扰乱图书馆管理秩序。本项目以微信小程序为载体，采用C/S 系统，与学生信息对应，为高校图书馆设计了一个图书馆选座系统，实现图书馆自习资源的有效共享，实现预约，选座等功能，能够提高座位的使用效率。本系统为小组成员在系统分析与设计课程上的作业项目。

1.2 设计原则

（1） 易于拓展复用，通过封装实现细节，降低耦合度。
（2） 按职责分配类的功能，一个类只负责一项职责。
（3） 保证图书馆座位信息的实时和预定的可靠性，并提供不间断的服务
（4） 系统需具有易用性，界面简洁易懂，操作逻辑简单合理，操作效率高。
（5） 用户的操作记录以及个人信息不被非法获取，或丢失，确保学生与管理员只能在各自允许的权限范围内操作，保证系统安全性。
（6） 考虑多个学校使用系统情况，实现可拓展性。

1.3 功能设计

- **学生：**

（1）注册登录：学生用户可以在系统上注册一个账号，帐号信息包括学号、姓名、学校，预约历史纪录，违规计数次数。注册帐号后学生可登录进入系统。

（2）查看座位预约情况：登录帐号后学生能够查看图书馆当前或未来空余座位信息。

（3）预约座位：学生可以预约未来某一时间段的空余图书馆座位。一个学生一天内最多可预约 4 次。

（4）查看个人预约记录：学生可以查看个人所有预约记录以及履约情况。但学生无法查看其他用户信息。

（5）修改或删除预约：学生在预约开始时间前可以修改或删除预约。修改或删除预约需要提前 15 分钟，否则将被记录违规一次。

（6）签到：系统通过二维码登录信息来记录用户履约情况。学生在当天预约时间进入图书馆扫二维码，视为签到完成，预约成功。若在预约开始时间后 10

分钟，学生仍未签到，则预约失败，无法履约学生将会被记录违规一次。

（7）签退：学生学习完毕，可在预约结束时间前或预约时间到达时进行签退。一旦超过预约时间，学生需再次预约申请空余座位。

- **管理员：**

（1）管理座位信息：管理员能够查看图书馆所有座位的预约情况。同时，管理员能够对图书馆内可用座位资源进行修改与更新。

（2）管理学生信息：管理员能够查看所有学生的帐号信息。

1.4 术语定义

名词术语	定义
学生用户	已经在系统中注册了并标记为本校学生的用户，具有一般用户权限。
管理员	已经在系统中注册了并标记为管理员的用户，具有管理员权限。
帐号信息	用户在注册时填入的信息，学生用户帐号信息包括 NetID、密码、学校、个人预约记录、违规记录次数。管理员用户帐号信息包括工号和密码。
违规计录次数	学生用户帐号信息之一，记录学生违规使用系统次数。
个人预约记录	学生用户帐号信息之一，记录学生预约图书馆座位的所有历史情况，包括预约时间，预约座位，是否履约。
座位信息	图书管所有座位的相关信息，包括所有时间段每个座位预约记录。
空余座位信息	图书馆座位信息之一，记录所有时间段内空余的座位信息。
预约座位信息	图书馆座位信息之一，记录所有时间段内被预约的座位信息。
惩罚状态	当学生违规记录次数达到三次，则进入惩罚状态，一旦进入惩罚状态一周内该学生用户无法预约图书馆座位。一周后违规记录次数清零，跳出惩罚状态。

表 1 术语表

2 开发环境与工具

后端开发语言：Go 语言

后端开发工具：Visual Studio Code

数据库：MongoDB

前端页面开发：微信开发者工具

3 软件设计技术

本系统中使用的技术如下：

(1) Service Oriented Programming 面向服务的编程

(2) 工厂模式

(3) 适配器模式

(4) 单例模式

3.1 技术选型理由

- **面向服务的编程(SOA)**

  为了规范前后端接口，后端进行了面向服务的编程设计，规定接口如下：

  i. 通过 web 中的 get，post 请求访问，访问限制如下：

  get 请求：请求数据放在 url 中。

  post 请求：请求数据放在 body 中，格式为 json 格式。

  ii. 返回值为 json 格式。

  因此，使用 SOA 技术具有以下优势：

  (1) 通过 SOA 技术，可以事先规定好前后端的接口，使前后端可以分开开发，最后进行整合测试。

  (2) 使用 SOA 技术，可以降低项目中耦合度，当需求发生变化或者增加需求时，可以只修改原项目的其中一层服务，或者原封不动原项目，增加新的接口。增加程序的可扩展性。

  (3) 通过 SOA 技术，可以使前端没必要等待后端开发完毕后再进行测试，可先构建桩模块进行测试。

- **工厂模式**

  通过工厂模式管理对象，所有非空的对象都由工厂创建，其他类负责使用。因此，使用工厂模式理由如下：

  (1) 工厂模式可以只产生指定类型的对象，防止对象出现副作用。

  (2) 工厂模式可以分离对象的产生和使用，降低使用类和对象类的耦合度。

- **适配器模式**

  使用适配器模式编写数据库驱动（dao 层）。当数据库进行变更时（使用

不同数据库时），只需要修改数据库驱动，不影响项目的其他部分。因此，使用适配器模式理由如下：

(1) 适配器模式可以在换数据库时，减少代码的修改量。
(2) 适配器模式可以增加项目的扩展性，方便项目移植到其他平台或者系统中使用。

- **单例模式**

使用单例模式编写后台项目中的每一层。使用单例模式的理由如下：

(1) go 语言的包实际上是单例的。
(2) dao 层和数据库接轨，使用单例模式可以方便的管理数据库锁，防止数据库出现错误。
(3) 层与层之间只有接口的调用和数据的传递，使用单例模式更合适。

3.2 对应模块与代码

- **面向服务的编程(SOA)**

(1) 定义的接口文档：（使用 blueapi 定义）
https://librarybookseatsystem.docs.apiary.io/#

(2) 层的定义：

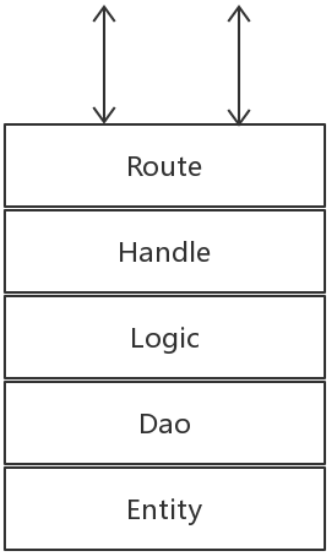
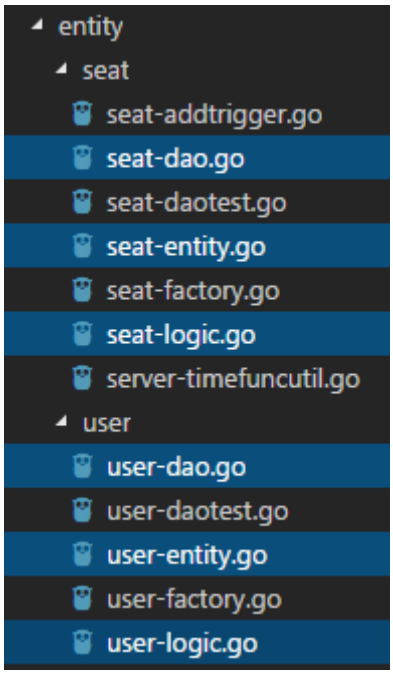


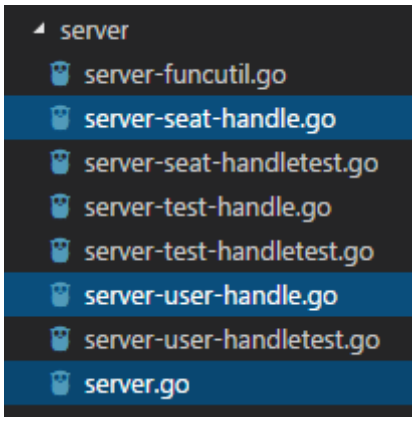
图 1.1 层的定义

(3) 层在项目中的实现

i. Entity 层，Dao 层，Logic 层。每类对象都有属于自己的 Entity 层，Dao 层，Logic 层，分成不同文件夹实现。如图所示：



ii. Handle 层，Route 层。每类对象都有属于自己的 Handle 层，Route 层将所有 Handle 层统一。如图所示：

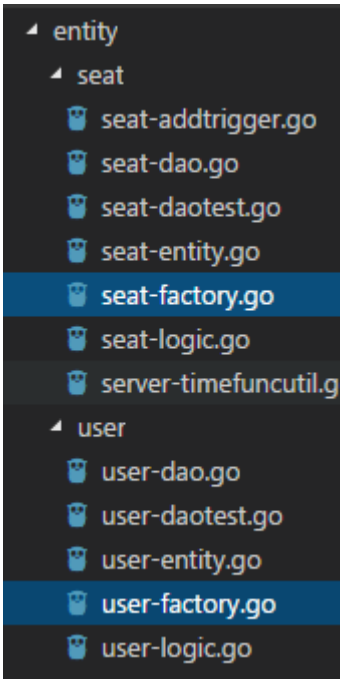


其中，图中的其他文件一部分是测试模块，一部分是 util 模块，包含 util 函数封装。

- **工厂模式**

相应实现如下：

(1) 两个 factory 位置



(2) 具体实现

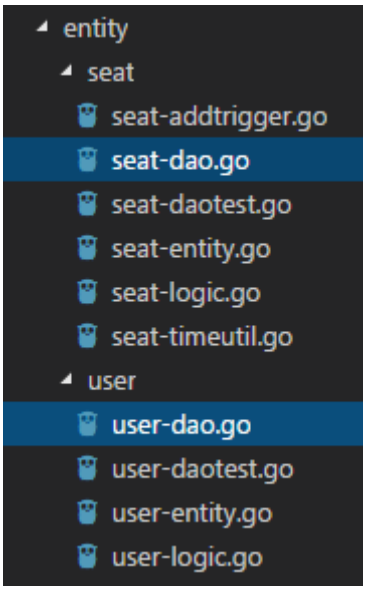
```
/*************************************************************************
Copyright(C) 2018
Author: huziang
Description: 工厂包，用于生产对象
Date: 2018年7月8日 星期日 下午1:50
*************************************************************************/

package seat

// newItems 生成一个Item数组，id从0开始
func newItems(seatnumber int) []Item {…
}

// newSeatInfo 生成一个新的SeatInfo
func newSeatInfo(timeinterval TimeInterval, item Item) *SeatInfo {…
}

// newTItems 生成一个TItem数组，timeinterval从当前时间段开始，数组数量从配置文件读取
func newTItems(seatnumber int) []TItem {…
}

// newSTItem 生成一个STItem
func newSTItem(school string, seatnumber int) *STItem {…
}
```

其中，在工厂中，固定只能生产以创建时间为起始日期的座位。

- **适配器模式**

相应实现如下：

(1) 两个 dao 层位置



(2) 数据库驱动的部分接口

```
/*************************************************
Function: Insert
Description: 插入新项
InputParameter:
    student: 新的student
Return: none
*************************************************/
func (*ItemAtomicService) Insert(student *Item) {…
}

/*************************************************
Function: Update
Description: 更新旧项
InputParameter:
    student: 要更新的student
Return: none
*************************************************/
func (*ItemAtomicService) Update(student *Item) {…
}

/*************************************************
Function: FindByID
Description: 通过主键ID查询数据
InputParameter:
    ID: 学生的ID
Return: 查询到的学生结果，包含所有的字段
*************************************************/
func (*ItemAtomicService) FindByID(ID string) *Item {…
}
```

(3) 数据库驱动和对应数据库适配

```go
package mgdb

import (
    . "github.com/book-library-seat-system/go-server/util"
    "labix.org/v2/mgo"
)

// Mydb 数据库指针
var Mydb *mgo.Session

// 生成数据库，对数据库进行链接
func init() {
    // 链接mongodb数据库
    session, err := mgo.Dial("")
    CheckErr(err)
    session.SetMode(mgo.Monotonic, true)

    Mydb = session
}
```

- **单例模式**

由于 go 语言的特殊性，大部分包都可以体现单例模式。如：

(1) Handle 层引用 logic 层包

```go
package server

import (
    "errors"
    "fmt"
    "net/http"

    "github.com/book-library-seat-system/go-server/entity/seat"
    "github.com/book-library-seat-system/go-server/entity/user"
    . "github.com/book-library-seat-system/go-server/util"
    "github.com/unrolled/render"
)
```

(2) Handle 层只使用 Logic 层包中的接口函数

```go
// showTimeIntervalInfoHandle 返回时间段信息
func showTimeIntervalInfoHandle(formatter *render.Render) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        defer errResponse(w, formatter)
        fmt.Println("showTimeIntervalInfoHandle")
        // 解析参数
        param := parseReq(r)
        CheckUserLogin(param)
        // 从数据库获取数据
        timeintervals := seat.GetAllTimeInterval(param["school"])
        rtnjson := TimeintervalRtnJson{}
        for i := 0; i < len(timeintervals); i++ {
            rtnjson.Timeintervals = append(rtnjson.Timeintervals, TimeintervalJson{
                TimeInterval: timeintervals[i],
                Restseatsnum: len(seat.GetAllSeatinfo(param["school"], timeintervals[i])),
            })
        }
        formatter.JSON(w, http.StatusOK, rtnjson)
    }
}
```

二、 架构设计

1 架构描述

1.1 简要描述

在本系统中，使用 MVC 作为系统的架构。

1.2 设计框架图

图 2.1 设计架构图

1.3 系统架构描述

(1) View

在小程序中通过使用 POST 和 GET 的方式发出请求，使得用户通过使用移动设备借助网络进行操作。在本应用中，用户在 view 层进行操作，发送请求，并且接收来自 controller 层所反馈的信息。

(2) Controller

通过接收 HTTP 请求，并且根据请求调用相应的逻辑，并且将结果返回给客户端。

(3) Model

在逻辑层处理所有的逻辑，其中包括数据的逻辑是否符合要求等，在本层仅仅注重逻辑的处理，而不注重数据的存取的细节。

(4) DAO

作为数据源层，用于进行数据的交换，主要用于从 DB 中获取数据然后传递给 Model 层进行逻辑的处理，并且接收数据交给 DB 进行数据的修改等操作。

(5) DB 数据库

用于储存所有的用户信息，座位信息，此处注重数据的读写存储的细节方法。

2 关键抽象

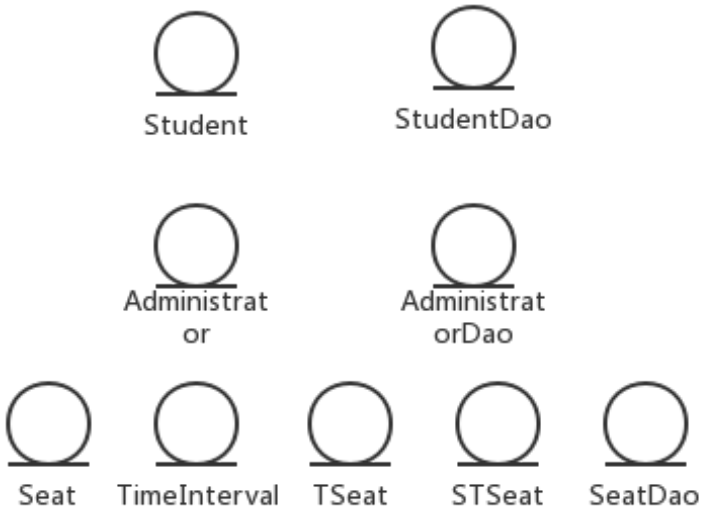


图 2.2 关键抽象

三、 模块划分

1 预约座位用例模块

➢ 类图：

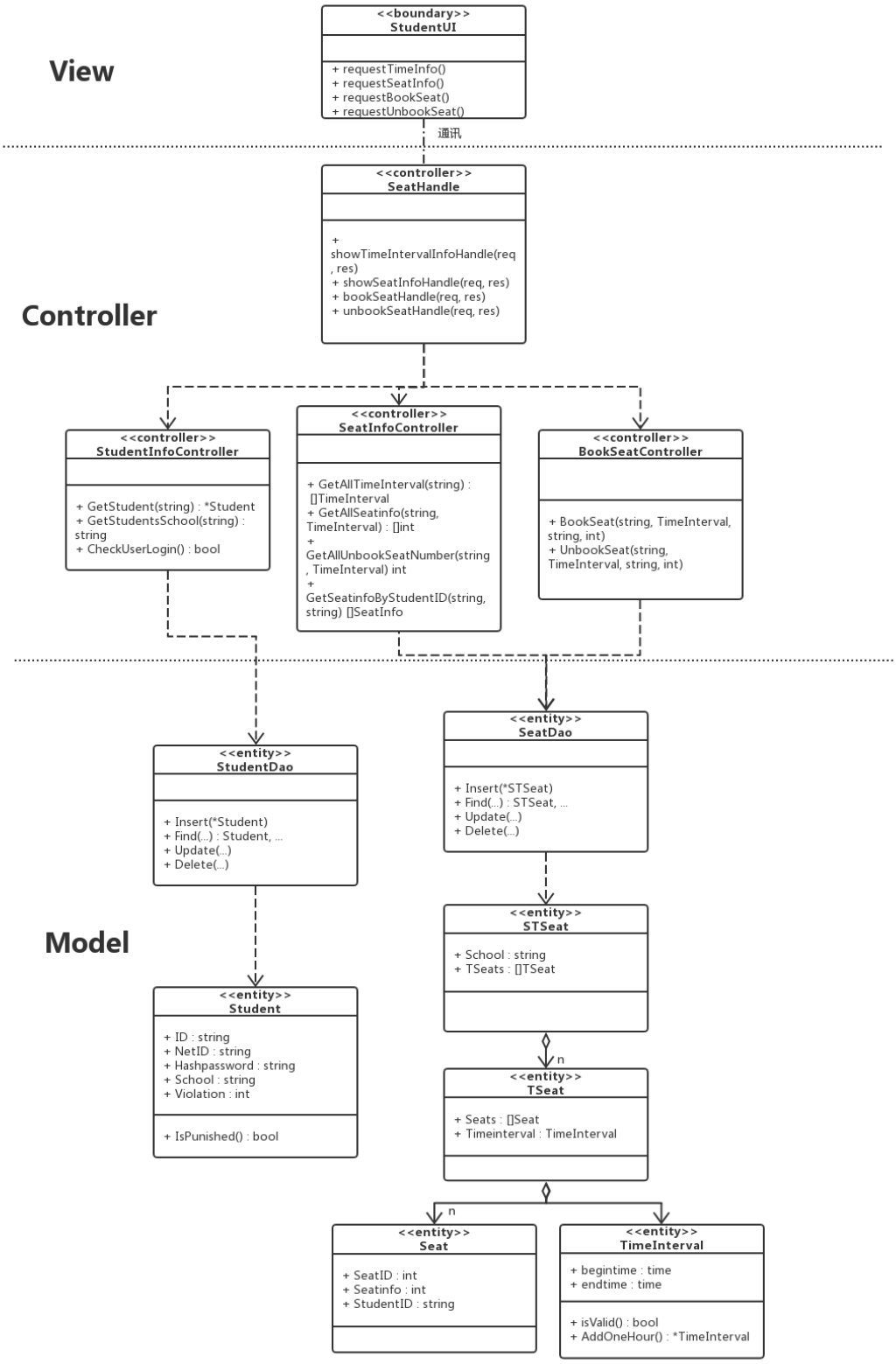


图 3.1 预约座位用例类图

➢ 时序图：

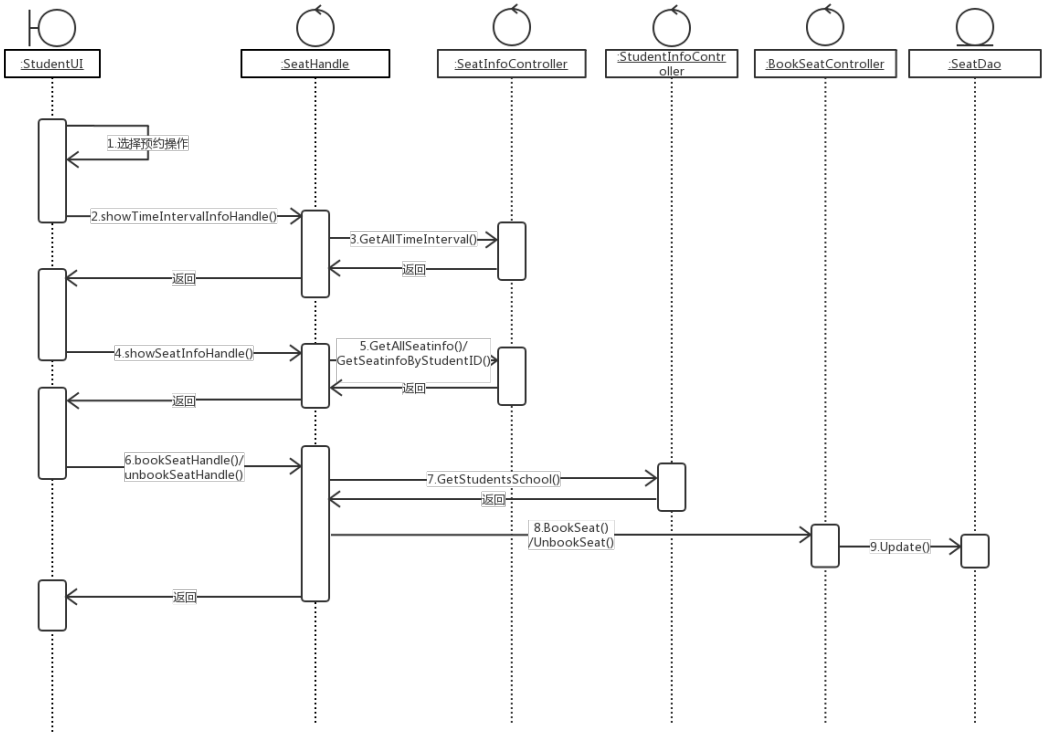


图 3.2 预约座位用例时序图

➢ 协作图：

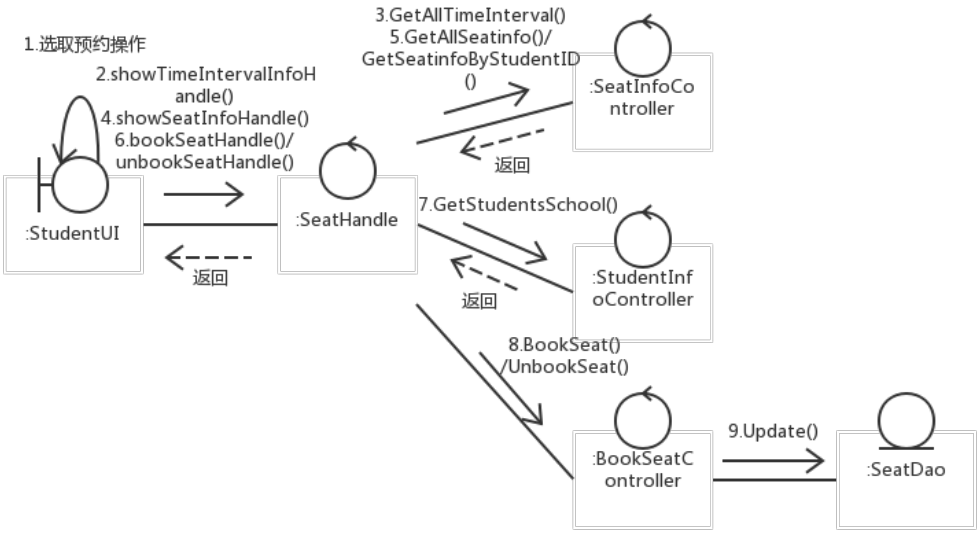


图 3.3 预约座位用例协作图

2 签到用例模块

➢ 类图：

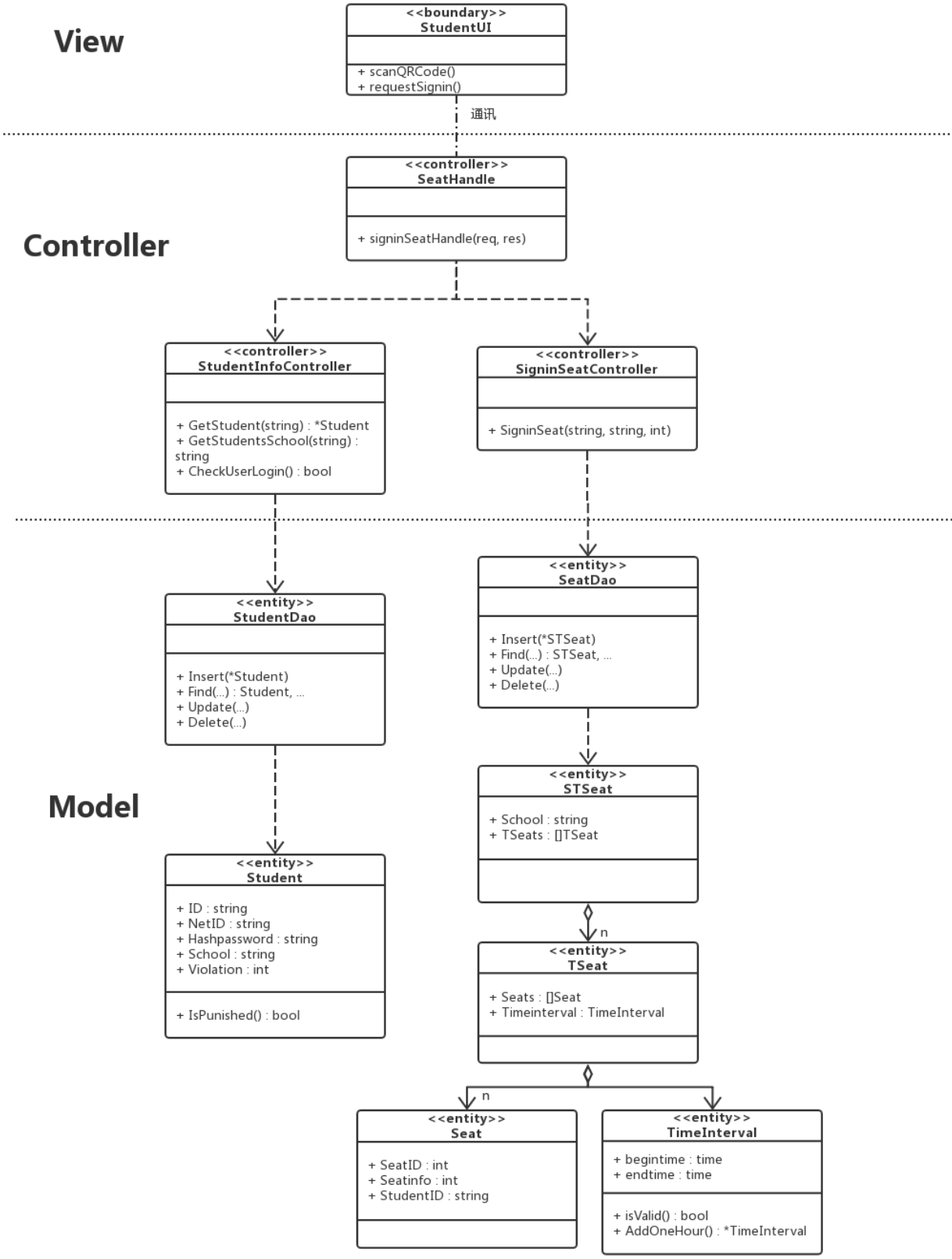


图 3.4 签到用例类图

- **时序图：**

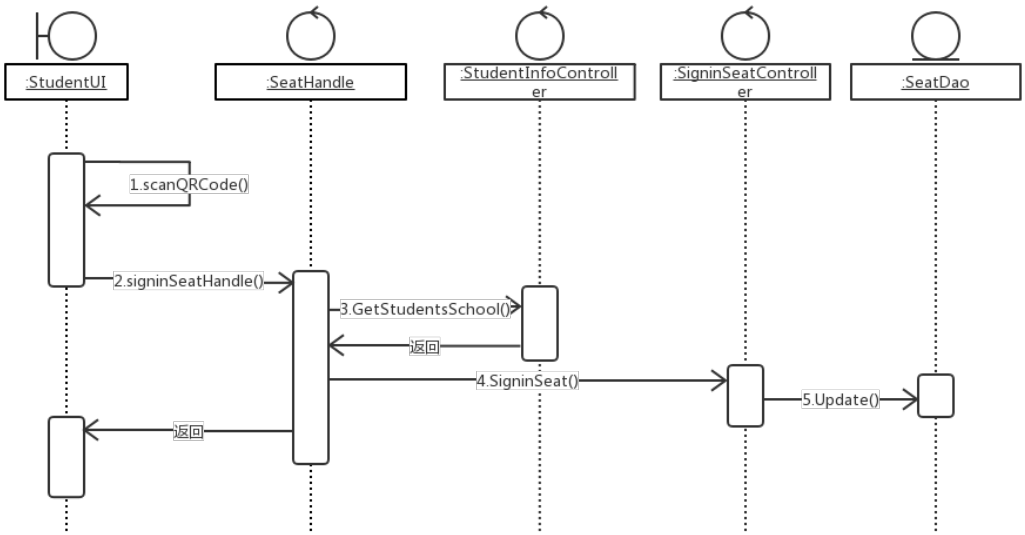


图 3.5 签到用例时序图

- **协作图：**

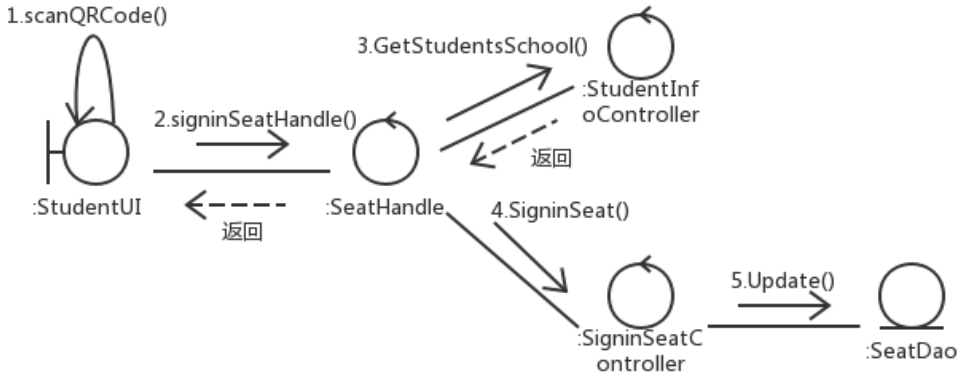


图 3.6 签到用例协作图

3 管理座位用例模块

➢ 类图：

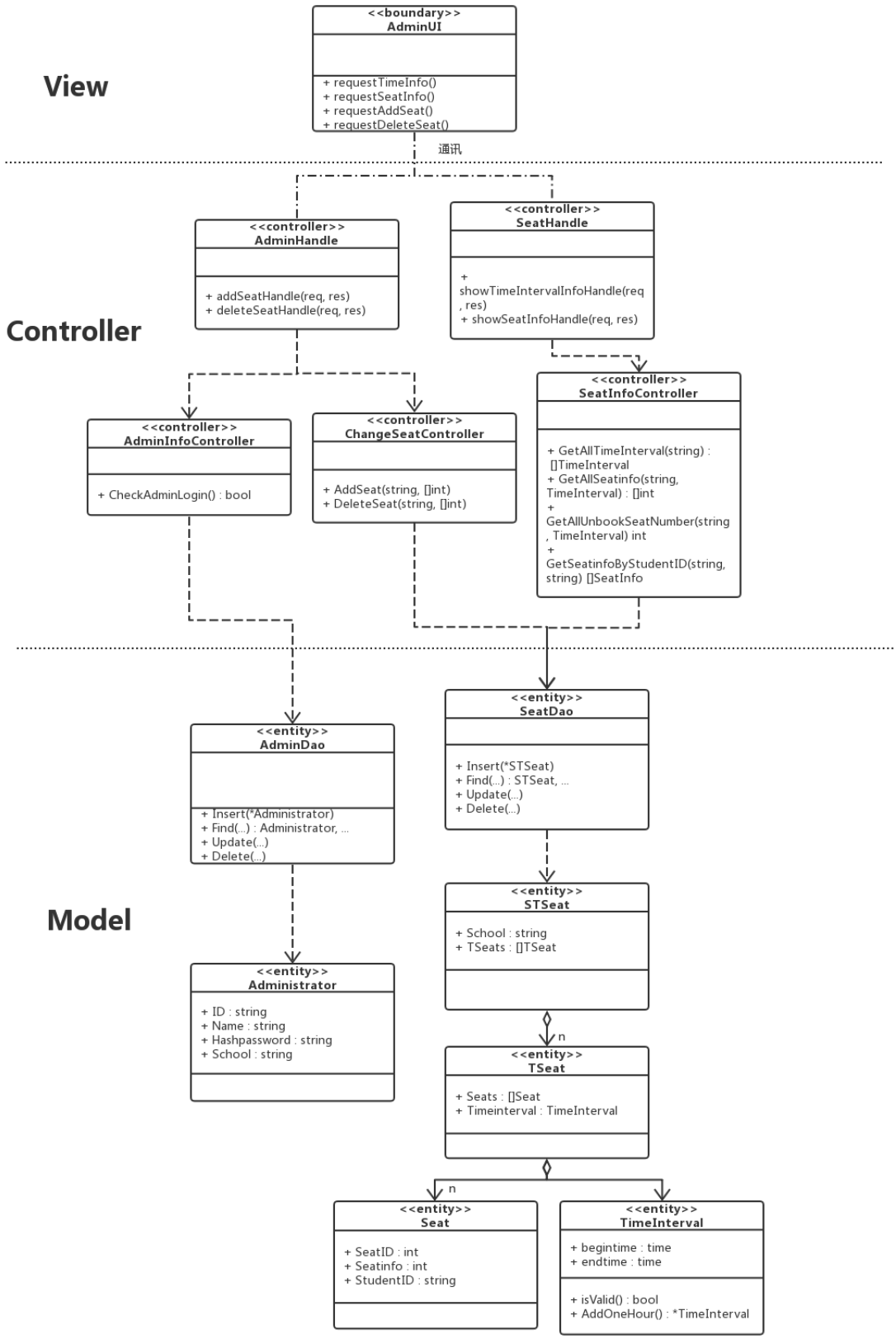


图 3.7 管理座位用例类图

- **时序图：**

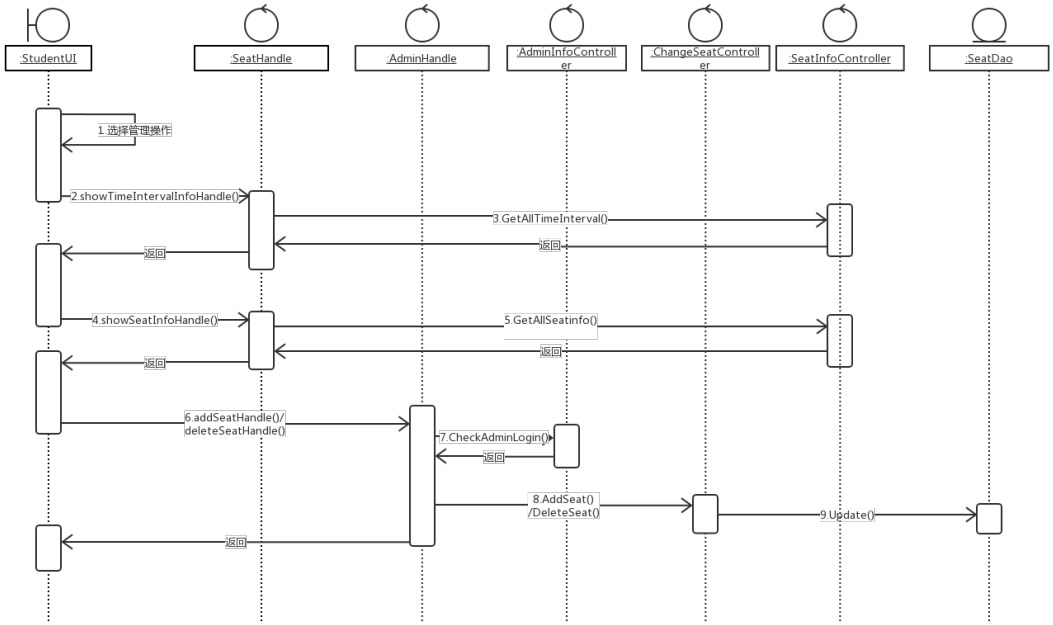


图 3.8 管理座位用例时序图

- **协作图：**

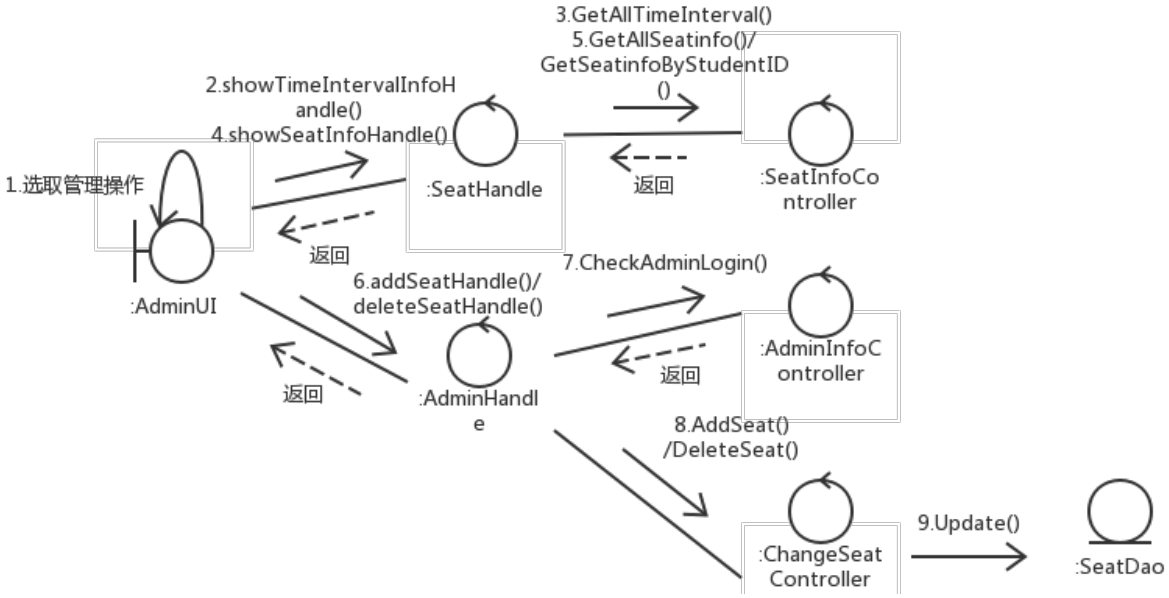


图 3.9 管理座位用例协作图

四、 子系统及其接口设计

1 分析合并类

本节将析取出来的边界类、控制类、数据类进行合并整理，得到系统的合并类图。

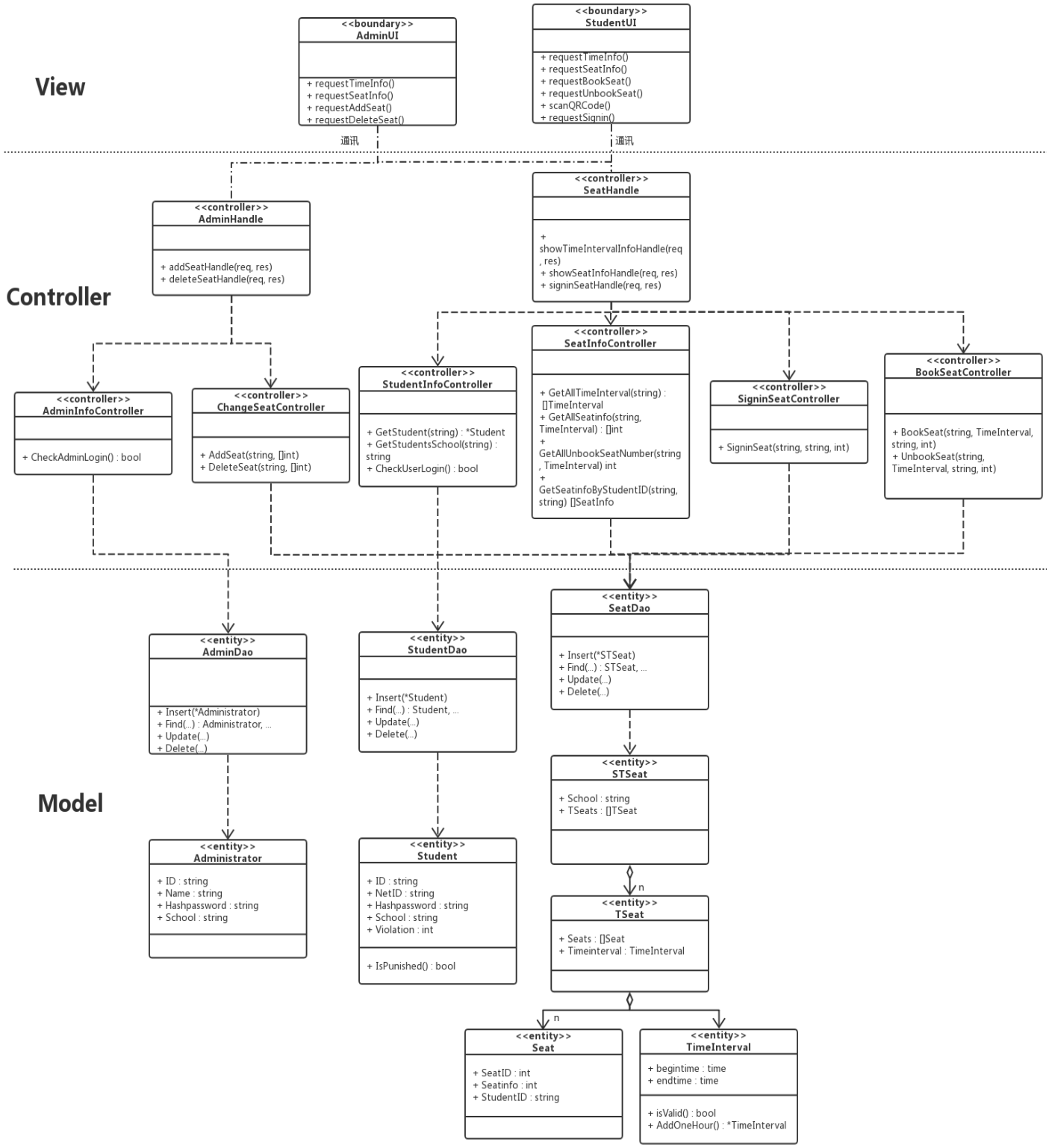


图 4.1 合并类图

2 确定设计类

本项目组对析类进行了分析与检查，以确定其是否能成为设计类。经过分析发现，所有分析类均为单逻辑，不需要进行类的分解或合并，因此不做修改。

3 划分子系统

经过本项目组分析，本系统无需进行子系统设计及其接口设计。

五、 部件设计

1 分析并发需求

该系统需要满足以下三个并发需求：

1） 同一时间，同一个学校的不同学生可能会同时发出操作请求，此时需要并发进行

2） 学校的管理员可能会在学生操作时修改座位信息和学生信息，此时需要并发进行

3） 不同学校可能会共用同一个服务器，此时需要并发进行

2 针对某个需求的设计方案

对三个并发需求，共提出两个设计方案：

1） 针对第一个和第二个需求：当收到一个新请求，主线程创造新的线程，新的线程执行收到的请求，主线程则继续等待接收。并且在进行数据库操作前，添加读写锁，保证不会出现读写冲突。

2） 针对第三个需求：多个学校请求公用一个服务器 CPU 资源。由于不同学校的数据不互通，将不同学校的数据放入不同数据库，或同一个数据库中的不同库或不同表中。

3 生命周期

新线程的周期开始于主线程收到一个新请求，结束于线程执行函数结束并回复完毕。

4 映射到现实系统

1） 使用 negroni 包进行线程创建调度。

2） 构建 mutexmanager 类，管理数据库读写锁。

3） 封装数据库接口，使不同学校的座位信息存储到同一个数据库的不同

表中。

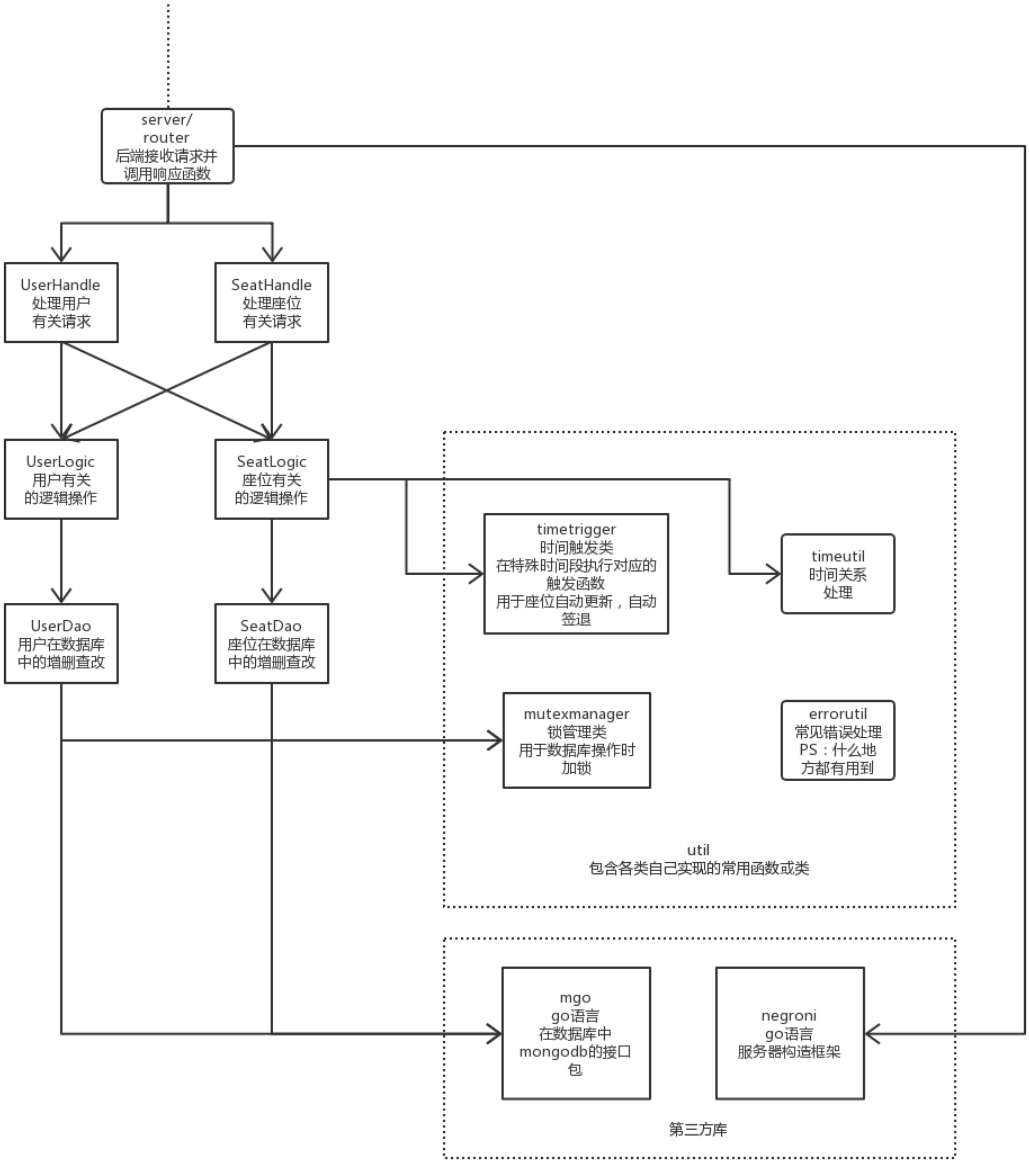


图 5.1 后端实现的逻辑结构图