一、 架构设计

2.1 架构描述

会员管理系统采用 MVVM（Model-View-ViewModel）三层架构，该架构的使用实现了应用程序的分层管理，简化后续对程序的修改和扩展，并且使程序某一部分的重复利用成为可能。模型层（Model）根据需求从数据库中获取所需要的数据；视图层（View）负责显示用户界面及相关数据并对用户输入进行反馈；视图模型层（ViewModel）是模型层与视图层的连接层，从模型层获取数据后形成视图层所需要的数据结构与视图层双向绑定。当视图层/视图模型层数据发生改变，视图模型层/视图层也会同步改变，当数据改变后，视图模型层则操作模型层对数据库进行更改。

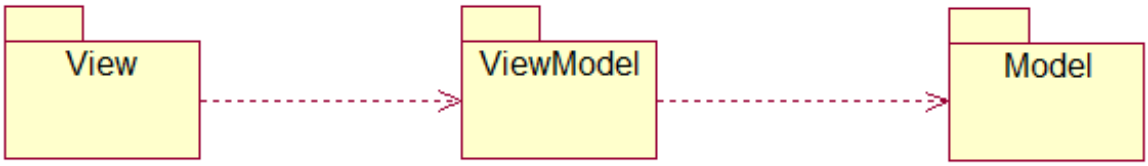

图 2.1 系统架构图

2.2 关键抽象

经过分析，本系统有 3 个实体类，分别为 users、activities、attendance_records。

users：储存用户基本信息，包括用户名、密码(加密)、邮箱、头像等个人账号信息。

activities：储存活动相关信息，包括标题、事件、地点、人数、图片等。

attendance_records：储存签到考勤相关信息，包括每个记录对应的活动以及用户。

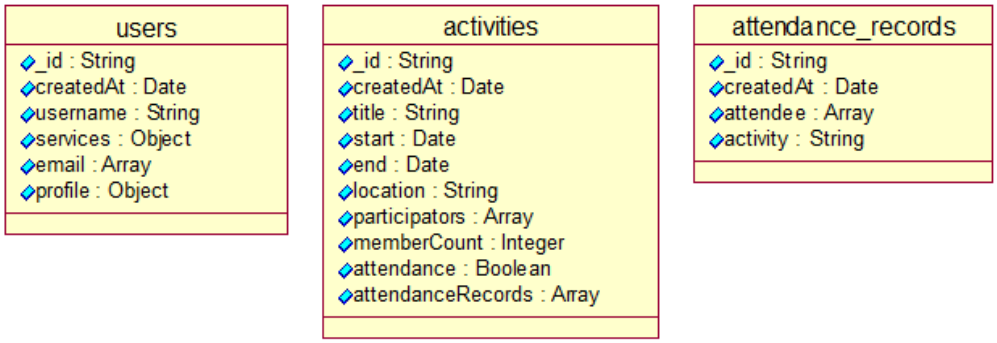

图 2.2 实体类图

二、 用例分析

3.1 注册用例分析

在注册的用例中，使用者通过“注册”按钮向 SigninPage 边界类发出注册账号的请求，Signup 控制类收到请求后，渲染注册组件，展示信息填写模块。调用 handleSubmit()方法提交用户注册信息 ,调用 checkInfo()方法对用户填写的信息如姓名、密码、学号等进行验证，若注册信息有误则提示修改，成功则通过后台 collection 类使用 checkInfo()方法对用户信息进行再次验证，并连接数据库查询是否存在用户，若不存在则执行 insert()方法将用户信息插入数据库。数据库插入成功后返回成功信息，信息传递至 SigninPage 提示用户并自动登录。

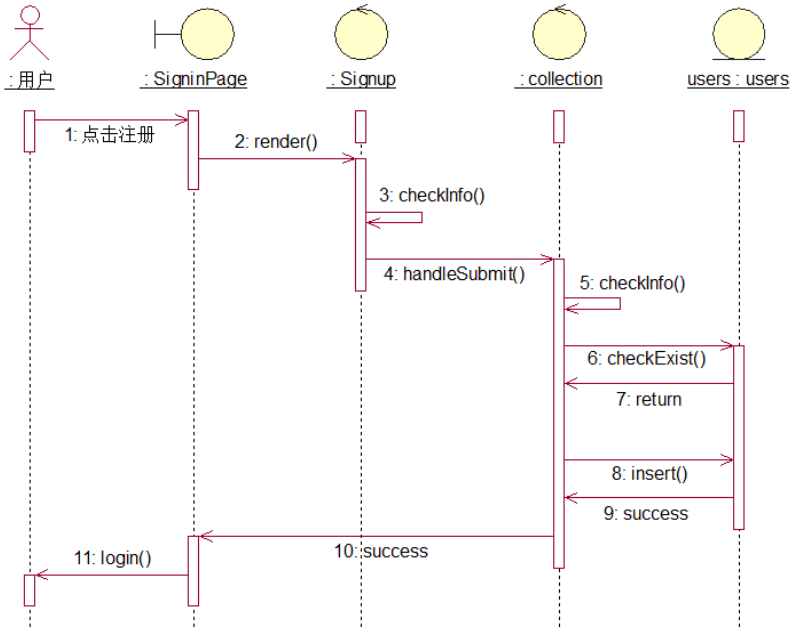

图 3.1 注册用例时序图

3.2 签到考勤用例分析

在签到考勤的用例中，用户通过扫描二维码，访问 TakeAttendance 边界类，边界类使用 checkUser()方法检测登录状态、使用 checkActivityStatus()方法检测活动状态。调用 handleSubmit()方法提交以上状态信息，调用 checkExist()方法连接数据库查询是否已经签到，若未签到则执行 insert()方法将信息插入数据库。数据库插入成功后返回成功信息，信息传递至 TakeAttendance 并提示用户签到成功。

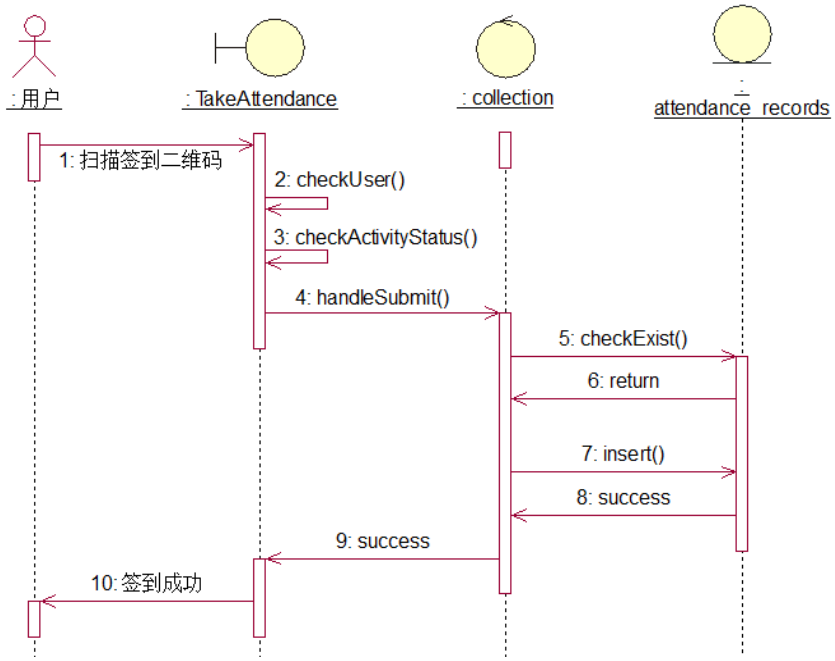

图 3.2 签到考勤用例时序图

3.3 发布活动用例分析

在发布活动的用例中，干事通过边界类 ActivityEdit 填写活动信息，ActivityEdit 调用 checkInput()方法对活动信息进行验证，再调用 handleSubmit()方法提交活动信息。collection 类调用 checkInfo()方法对活动信息进行二次验证，然后执行 insert()方法将活动信息插入数据库。数据库插入成功后返回成功信息，信息传递至 ActivityEdit 提示干事活动发布成功。

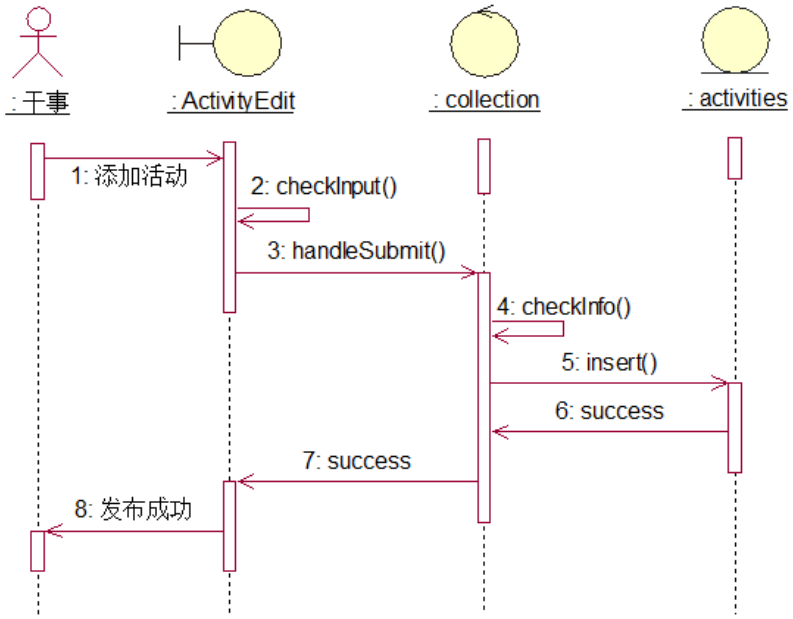

图 3.3 发布活动用例时序图

3.4 报名活动用例分析

在活动报名的用例中，用户通过向 ActivityDetail 边界类发出点击报名的请求，ActivityDetail 调用 confirmApply()确认用户请求，再调用 handleSubmit()方法提交活动报名信息。collection 类使用 checkCapacity()连接数据库查询人数容量，若有空位则执行 insert()方法将报名信息插入数据库。数据库插入成功后返回成功信息，信息传递至 ActivityDetail 提示用户报名成功。

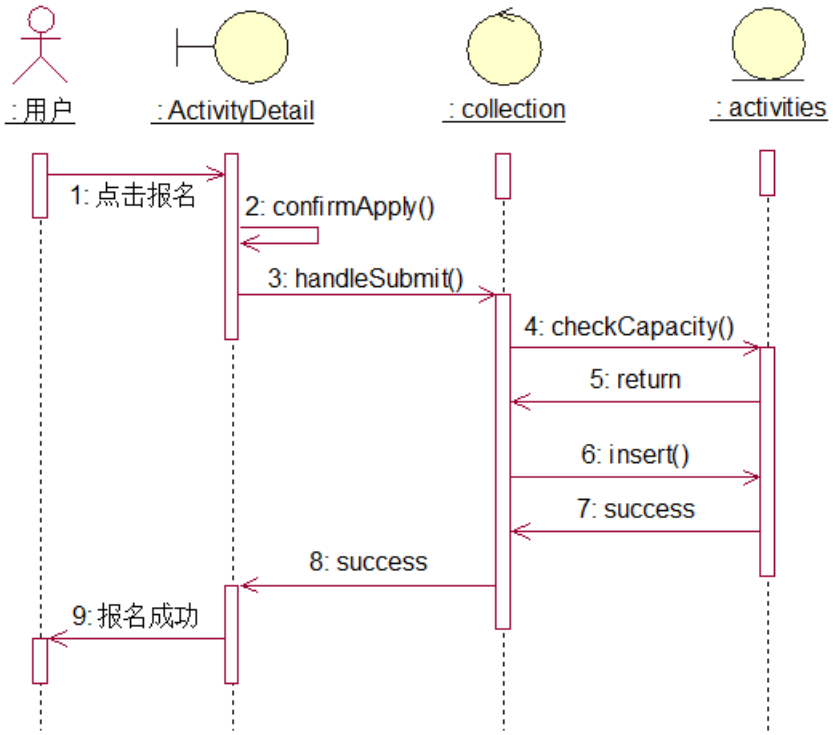

图 3.4 报名活动用例时序图

3.5 分析机制

经分析，活动管理系统中类的分析机制如下表所示：

分析类	分析机制
HomePage	持久性
ActivityPage	持久性
ProfilePage	持久性
SigninPage	持久性、安全性
ActivityDetail	持久性
ActivityEdit	持久性
TakeAttendance	持久性
collection	持久性、安全性
Signin	持久性、安全性
Signup	持久性、安全性
users	持久性、安全性
activities	持久性、安全性
attendance_records	持久性、安全性

表 2 分析机制表

3.6 合并分析类

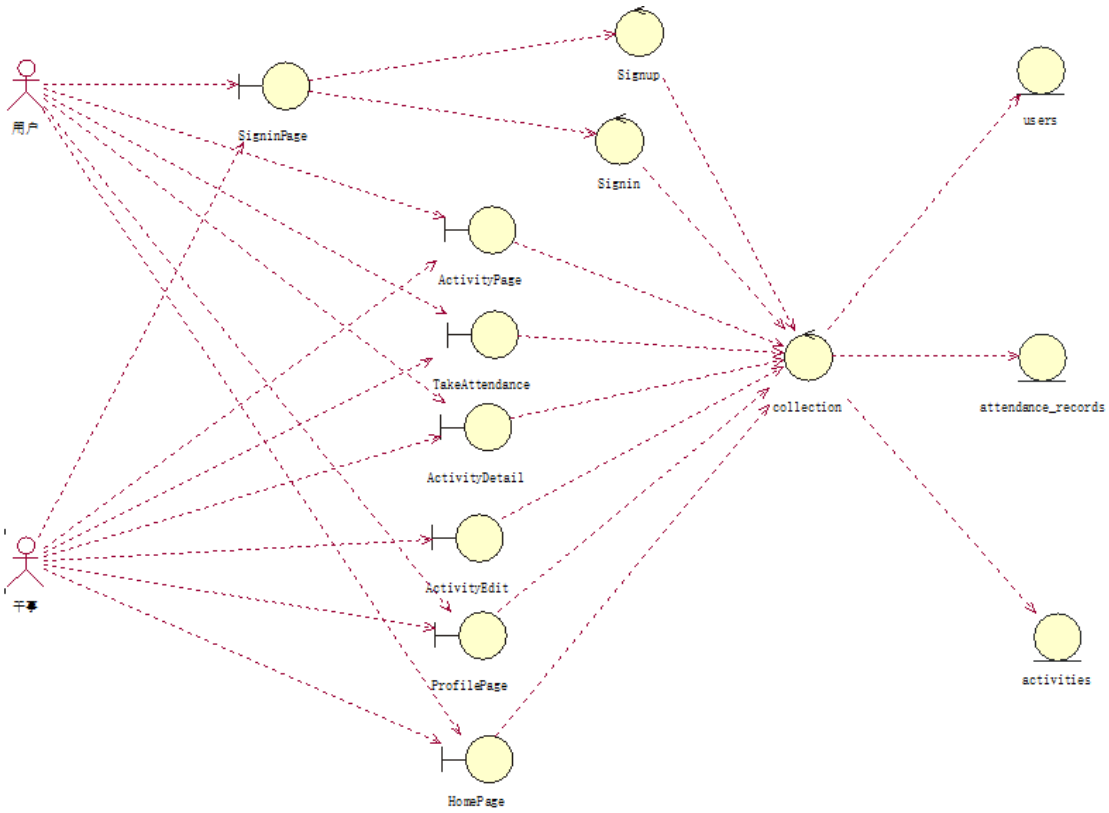

图 3.5

三、 子系统及其接口设计

4.1 合并分析类

本项目组对第三部分的分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，第三部分的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，所以设计类如第三部分所示。

4.2 定义子系统

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。因此 4.3 节、4.4 节可省略。

四、 分析系统并发需求

5.1 合并分析类

在一个系统中，并发性往往是至关重要的。因为如果有多个 CPU 可供使用，则并行地执行任务可以提高系统性能 ,同样系统的并发性也可以用来响应外部的随机发生的事件，并发性也可以增强系统的控制能力。

系统的并发需求主要来源于以下几个方面 :系统分布运行的程度、系统事件驱动的程度、关键算法的计算密度、运行环境支持的并发执行的程度。

常用的并发机制有以下几种 :多进程 ,应用程序在多个 CPU 上并行执行 ;多任务：操作系统通过间断性地执行不同的任务来实现并发。基于应用程序的并发 :应用软件在适当的时间转入到不同的分支中。

系统运行时可能产生的并发需求：同时多人使用系统并对进行 I/O 操作。

5.2 针对多人同时进行 I/O 操作的解决方案

后端使用基于 Nodejs 的框架，因此系统是单线程执行，采用非阻塞异步 I/O，提高系统运行效率。

5.3 生命周期

在本系统中，后台采用 Nodejs 搭建，Node.js 采用事件驱动和异步 I/O 的方式,实现了一个单线程、高并发的 JavaScript 运行时环境,只有在处理 I/O 操作时会创建新线程进行异步操作 ,并将该服务请求排到队尾 ,I/O 操作结束后新线程结束。