**学生考勤系统设计描述**

1. **UML** **构建**
2. **用例图**

本考勤系统分为系统和用户两个部分，系统一共包含了五个子系统，分别是数据库系统、 后端系统、用户管理系统、管理员系统和前端系统，用户包含了学生、老师和管理员三个子 用户。本项目用例图如下所示：

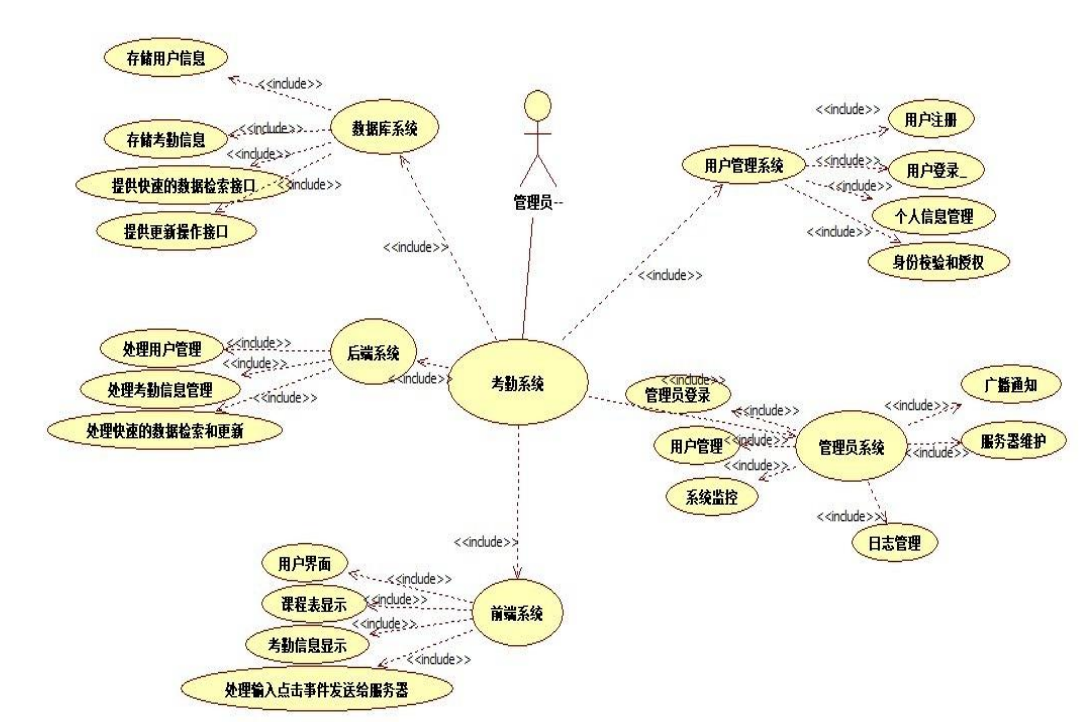


图 1 考勤系统的各个子系统

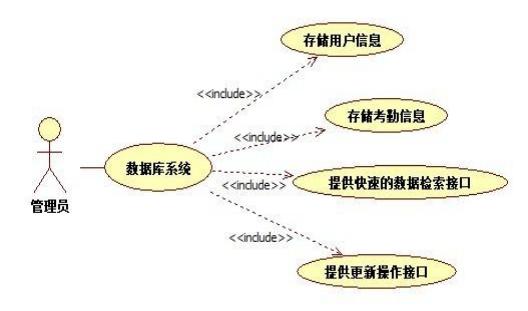


图 2 数据库系统

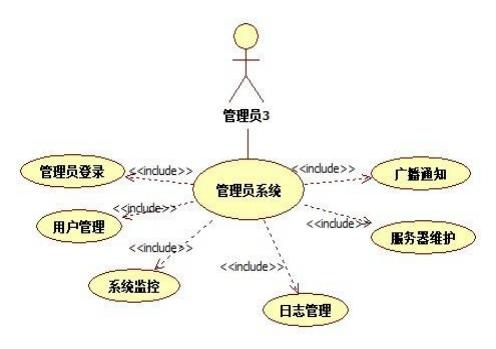


图 3 管理员系统

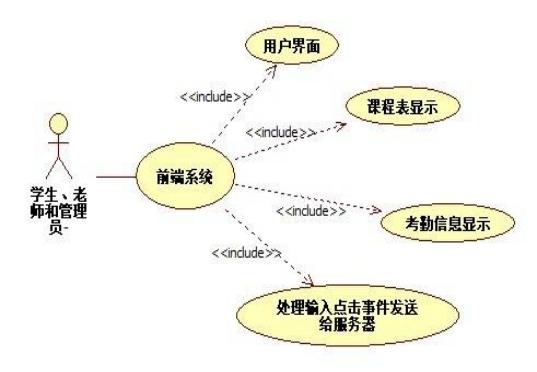


图 4 前端系统

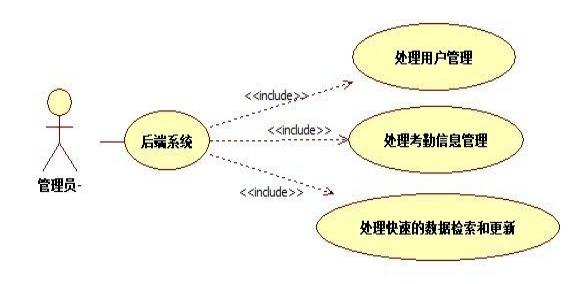


图 5 后端系统

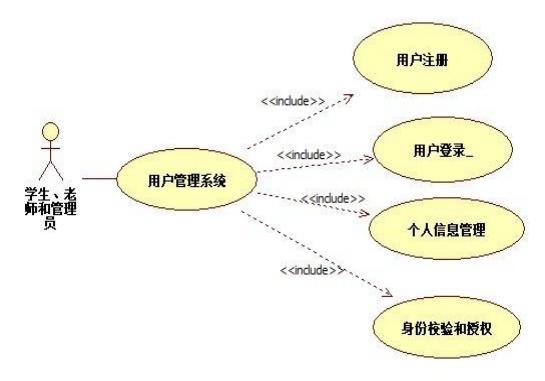


图 6 用户管理系统

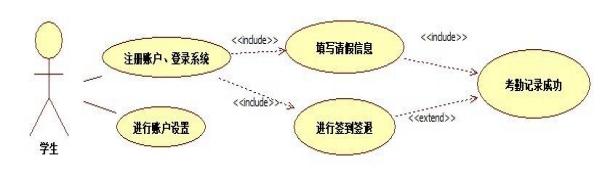


图 7 学生用例图

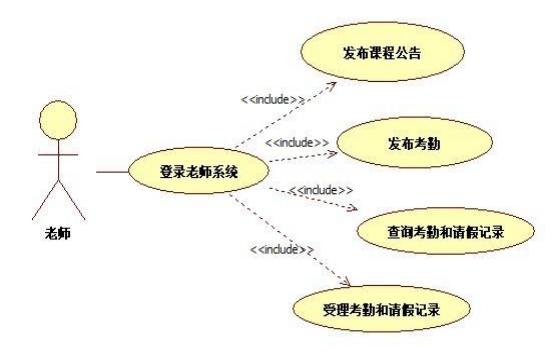


图 8 教师用例图

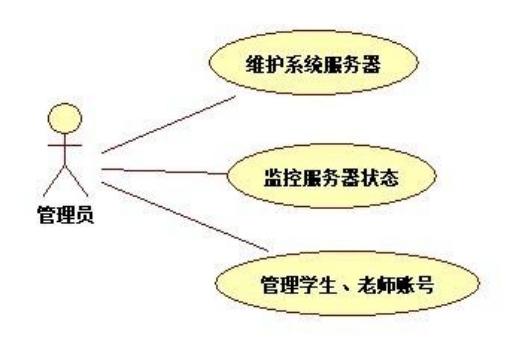


图 9 管理员用例图

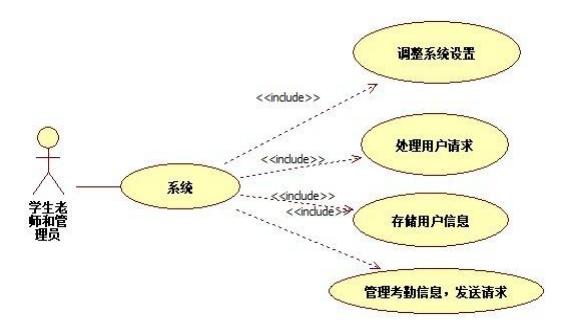


图 10 系统用例图

1. **类图**

项目有 16 个类，分别是 ClassScheduleManager（管理课程表），ClassSchedule（课程表类），CourseSelectionManager（管理课程选择）， CourseSelectionRecord （记录课程选择）， StudentManager（管理学生） ， Student （学生类）， PostAttendanceManager （管理考勤发布）， PostAttendanceRecord （记录考勤发布） ， CourseManager （管理课程） ， Course （课程类） ， AttendanceManager （管理考勤） ， AttendanceRecord（记录考勤），Teacher（教师类），TeacherManager（管理教师），StudentRoutes（学生界面指引），TeacherRoutes（教师界面指引）。

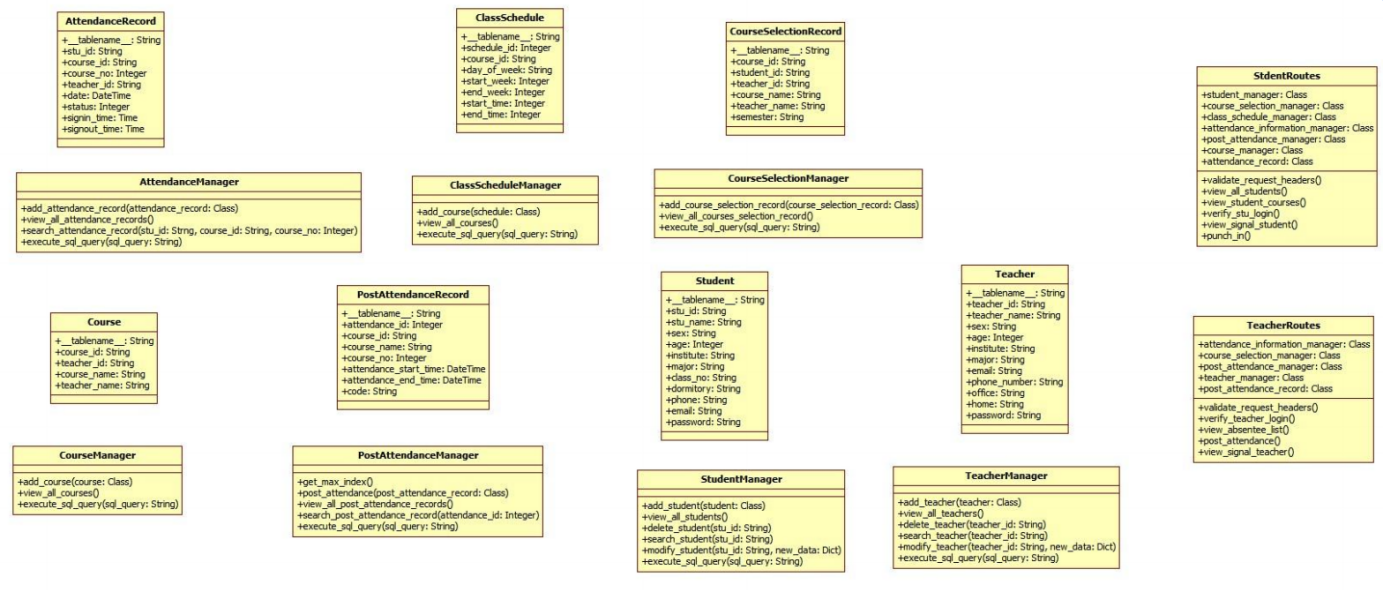


图 11 各种类的图

项目中类的关系如下

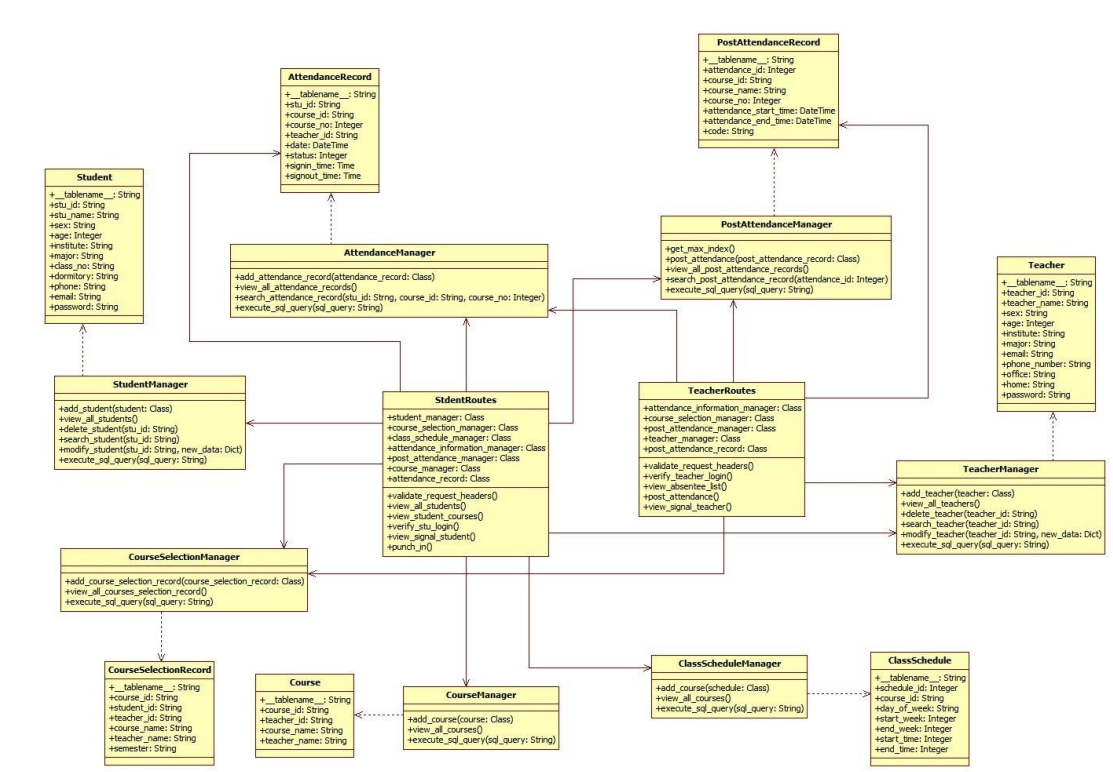


图 12 类图

1. **活动图**

本项目主要由生进行考勤及请学假、教师发布考勤及受理请假，管理员对系统进行维护， 以下是若干的基本活动图。

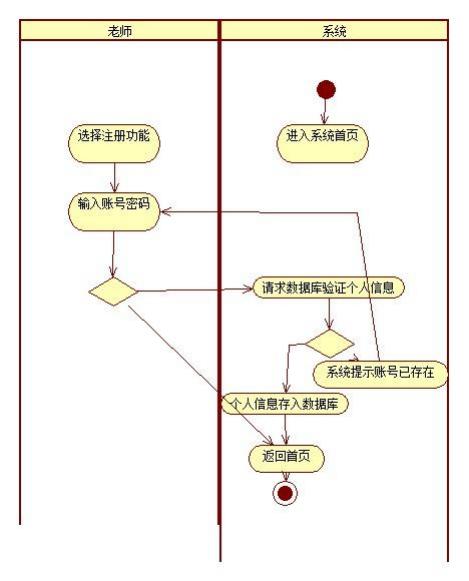


图 13 老师注册活动图

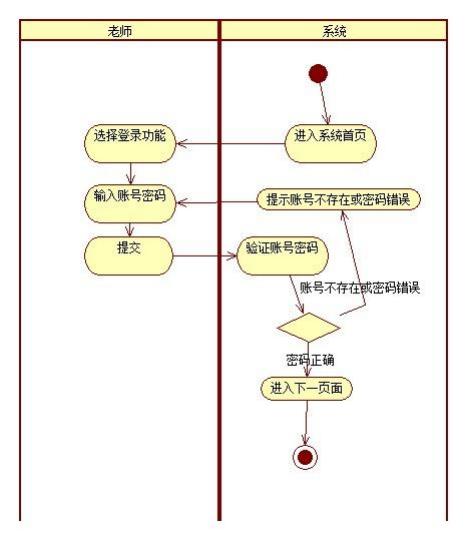


图 14 老师登录活动图

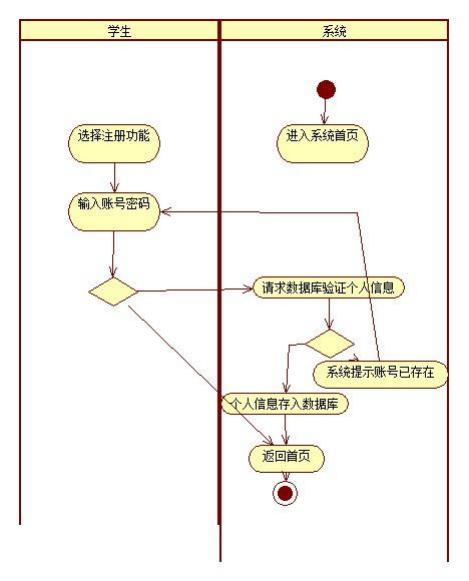


图 15 学生注册活动图

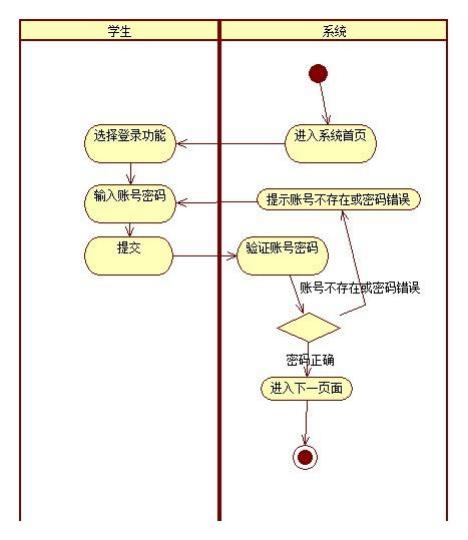


图 16 学生登录活动图

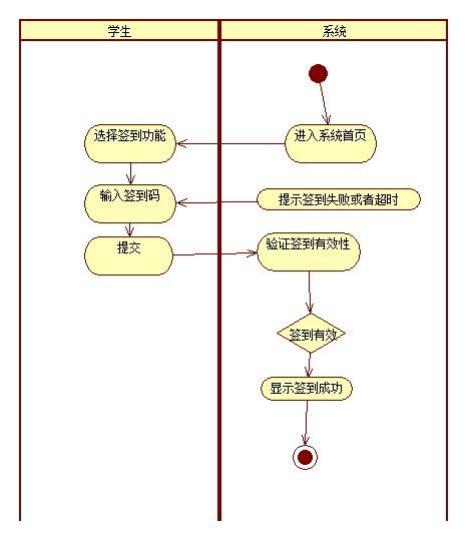


图 17 学生签到活动图

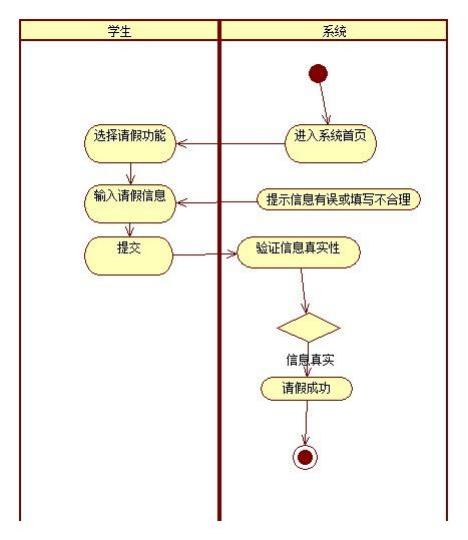


图 18 学生请假活动图

1. **状态图**

学生进行登录后，能够查看相应的考勤信息、进行考勤签到以及请假。学生进行考勤申 请，根据考勤的成功与否，会进入“考勤成功 ”或者“考勤失败 ”状态。当学生需要进行请 假时，学生填写信息向系统提出请假申请，由老师判断申请是否通过，分别对应“请假成功 ” 和“请假失败 ”状态。学生完成操作后，可选择退出登录状态。

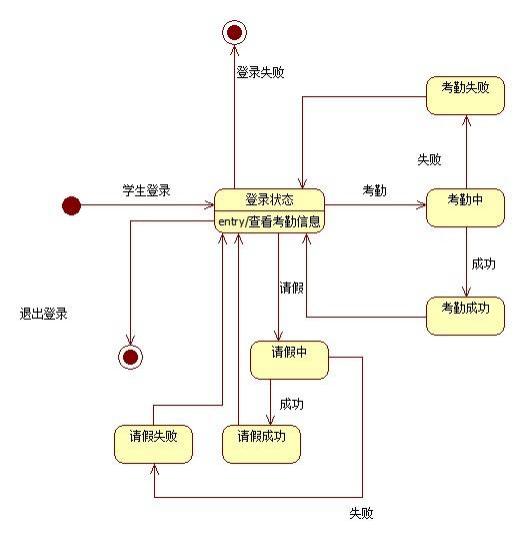


图 19 学生状态图

老师进行登录后，能够查看相应的考勤信息、发布考勤、结束考勤以及请假审批。老师 发布考勤任务，根据发布考勤的成功与否，会进入“发布考勤成功 ”或者“发布考勤失败 ” 状态，结束考勤后，会进入“考勤结束 ”状态。当学生进行请假申请时，老师在系统审批请 假申请，判断申请是否通过，分别对应“审批通过 ”和“审批不通过 ”状态。老师完成操作 后，可选择退出登录状态。

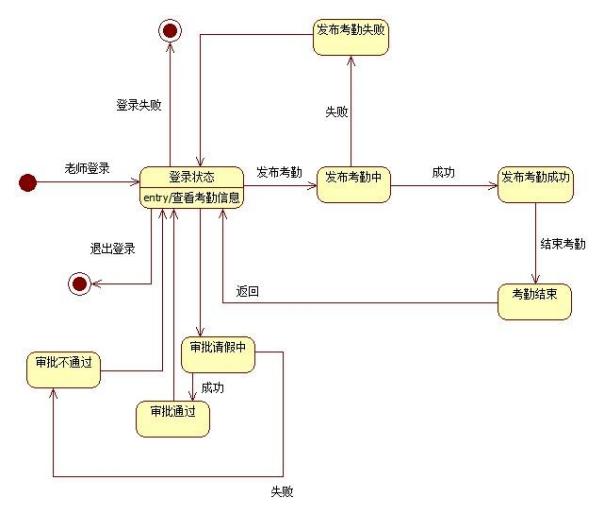


图 20 老师状态图

1. **时序图**

该项目中三个用户分别有不同的功能实现，以下是他们在系统中实现若干功能的时序图。

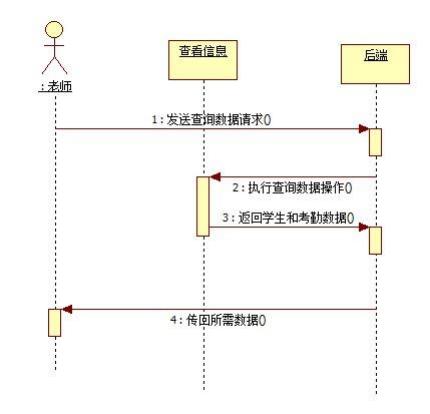


图 21 老师查询数据时序图

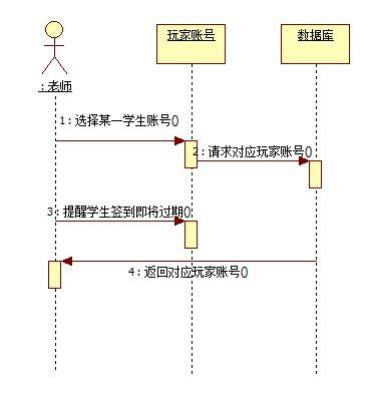


图 22 老师提醒学生签到时序图

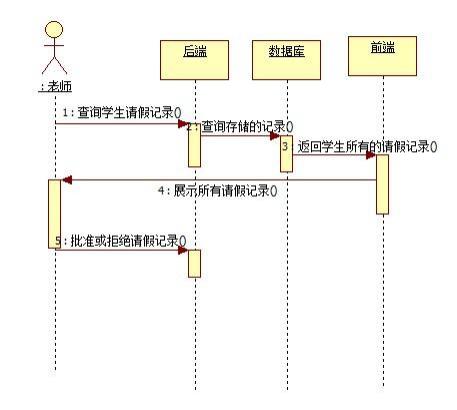


图 23 老师受理请假时序图

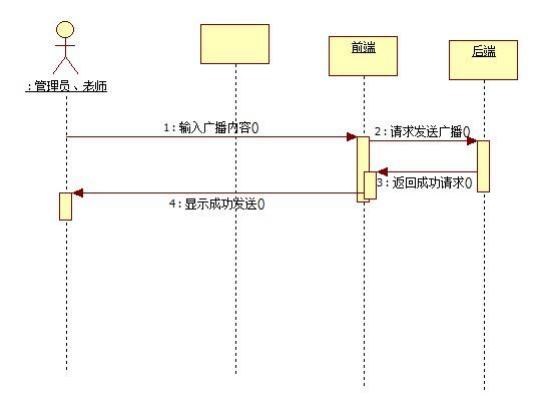


图 24 管理员、老师发布系统公告时序图

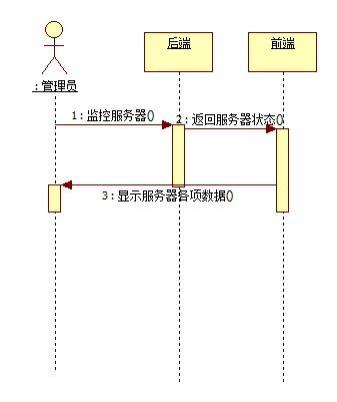


图 25 管理员监控服务器时序图

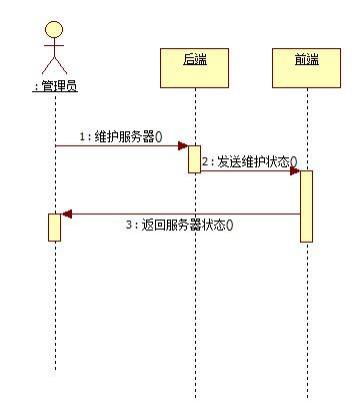


图 26 管理员维护服务器时序图

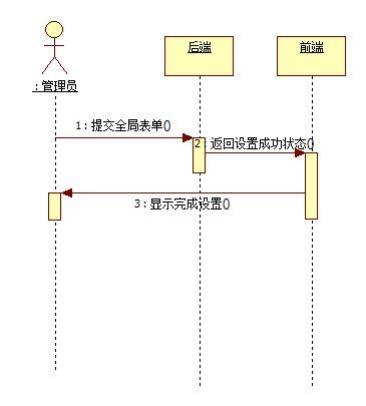


图 27 管理员对系统进行全局设置时序图

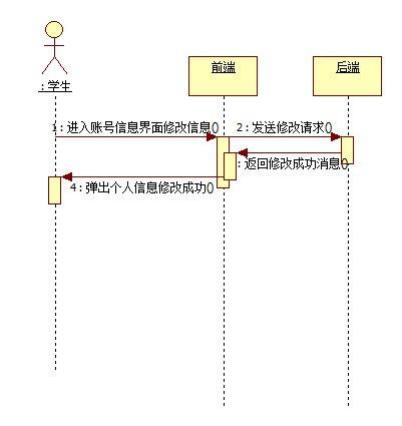


图 28 学生修改个人信息时序图

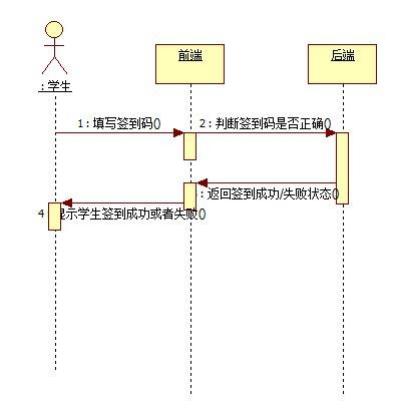


图 29 学生进行签到时序图

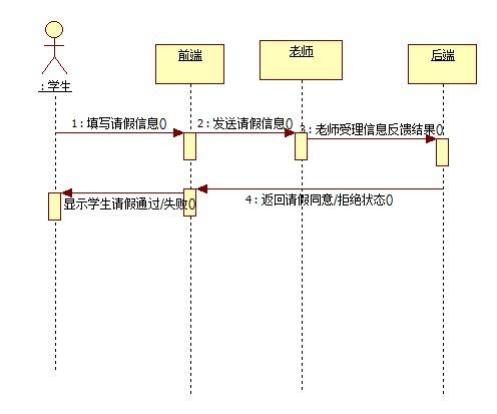


图 30 学生进行请假时序图

1. **系统架构模型**

采用前后端分离的微服务架构风格，具体阐述如下：

1. **前端（微信小程序）**
2. 视图层（View）

负责展示用户界面，包括各种页面和组件。

使用微信小程序提供的WXML（WeiXin Markup Language）和WXSS（WeiXin Style Sheets）进行页面布局和样式设计。

1. 逻辑层（Logic）

负责处理用户交互和业务逻辑。

使用微信小程序的JavaScript（或TypeScript）编写逻辑代码，包括事件处理、数据绑定、API调用等。

1. 数据层（Data）

负责与后端服务进行数据交互。

通过微信小程序的API（如wx.request）与后端服务器进行HTTP请求，获取或提交数据。

1. **后端（服务器端）**
2. API服务层

提供RESTful APIAPI，供微信小程序调用。

负责处理前端请求，执行业务逻辑，如用户认证、考勤记录查询、课程信息管理等。

1. 业务逻辑层

实现具体的业务逻辑，如考勤规则、学生选课逻辑等。

使用服务（Services）来封装业务逻辑，如AttendanceService、CourseService等。

1. 数据访问层

负责与数据库进行交互，执行数据的增删改查操作。

包含数据访问对象（DAO）或存储库（Gitee），如StudentRepository、TeacherRepository等。

1. **架构特点总结**
2. 前后端分离：前端和后端通过API进行交互，各自独立开发和部署。
3. 微服务架构：后端服务可以进一步拆分为多个微服务，每个服务负责特定的业务功能。
4. 分层设计：前端和后端都采用了分层设计，每层负责不同的职责，提高了系统的可维护性和可扩展性。
5. RESTful API：后端提供标准的API接口，便于前端调用和集成。
6. **技术使用总结**
7. 前端：微信小程序框架，使用WXML、WXSS和JavaScript（或TypeScript）。
8. 后端：python的flask框架等技术栈。
9. 数据库：MySQL。
10. **设计模式使用**
11. **单例模式（Singleton）：**

说明：为保证特定类只有一个实例并且只提供一个全局访问点，可以使用单例模式。

用例：

StudentManager和TeacherManager类使用单例模式能够确保系统中只有一个学生管理和教师管理的实例，避免数据不一致以及资源浪费。

通过getInstance方法，可以使系统的任何地方获取到唯一的实例，便于统一管理和操作。同时在类中加入私有化构造函数，防止外部实例化，确保实例在只有第一次被请求时才创建，节省资源。

1. **工厂模式（Factory）：**

说明：

为了提高代码的灵活性和可扩展性，可采用工厂模式创建对象的接口，由子类决定实例化哪一个类。工厂模式将对象的创建过程封装起来，使得客户端代码不需要直接实例化具体的类。

用例：

CourseManager和StudentManager类使用工厂模式来创建和管理课程和学生的实例之后，这两个类便不需要直接实例化具体的Course和Student类，而是通过工厂接口来创建对象。如果后续需要更换或添加新的Course或Student类实现，只需创建新的工厂类并替换原有的工厂类，而不需要修改客户端代码。

工厂模式还能够让系统通过配置文件或插件机制来动态选择不同的工厂实现，从而实现系统的插件化和动态扩展。

1. **观察者模式（Observer）：**

说明：定义对象间的一对多的依赖关系，当一个对象状态改变时，所有依赖它的对象都会收到通知并自动更新。

用例：AttendanceManager可以作为主题，Student和Teacher作为观察者，当考勤状态发生变化时，通知相关学生和教师。

1. **策略模式（Strategy）：**

说明：定义多个算法，将每个算法封装起来，并使它们可以互换。

用例：让AttendanceManager使用策略模式来支持不同的考勤策略，如手动考勤、自动考勤等。

1. **装饰器模式（Decorator）：**

说明：动态地给一个对象添加一些额外的职责。

用例：Student类使用装饰器模式来添加额外的属性或行为，如学生考勤记录、特殊权限等。

1. **命令模式（Command）：**

说明：将请求封装成对象，从而可以用不同的请求、队列或日志来参数化客户端，同时支持请求的撤销操作。

用例：PostAttendanceManager使用命令模式来封装考勤发布的请求，便于记录和撤销操作。

1. **中介者模式（Mediator）：**

说明：用一个中介对象来封装一系列对象的交互，使各对象不需要显式地相互引用，从而降低耦合度。

用例：ClassScheduleManager类和CourseSelectionManager类通过定义并实现中介者接口并在需要协调操作的地方调用中介者的方法来协调两个类实例中对于课程安排和选课的操作，减少直接交互。

1. **迭代器模式（Iterator）：**

说明：提供一种方法顺序访问一个聚合对象中的各个元素，而又不暴露其内部表示。

用例：StudentManager和CourseManager使用迭代器模式通过定义迭代器接口和集合接口来遍历学生和课程列表，便于统一访问。

1. **安全问题（未合并）**
2. **项目创新（未合并）**