**软件工程——动态作业管理系统设计文档**

**小组成员：姚扬鑫、徐昊博**

**目录**

[1. 引言 4](#_Toc7610)

[1.1 项目背景 4](#_Toc20134)

[1.1.1项目概述 4](#_Toc4875)

[1.1.2目标与范围 4](#_Toc3281)

[1.2 目标读者 5](#_Toc9546)

[1.3 文档概述 6](#_Toc31342)

[2. 总体架构设计 6](#_Toc20360)

[2.1 系统架构概述 6](#_Toc539)

[2.1.1移动端架构概述 6](#_Toc1647)

[2.1.2网页端架构概述 8](#_Toc20537)

[2.2 模块划分 11](#_Toc11705)

[2.2.1各模块功能概述 11](#_Toc26384)

[2.2.2模块间关系图 15](#_Toc32312)

[2.3 技术选型 15](#_Toc1323)

[2.3.1移动端技术栈 15](#_Toc2372)

[2.3.2网页端技术栈 17](#_Toc30608)

[（1） 网页：HTML + CSS 17](#_Toc11413)

[（2） 框架：Django 17](#_Toc32447)

[（3）数据库：MySQL 18](#_Toc8643)

[（4）界面设计：Mockplus 18](#_Toc25757)

[（5）版本控制：Git 18](#_Toc28430)

[3.移动端设计 19](#_Toc4785)

[3.1 组件设计 19](#_Toc27416)

[3.2 数据绑定 21](#_Toc13154)

[3.3 网络通信 21](#_Toc21428)

[3.4 生命周期管理 22](#_Toc32283)

[4. 网页端设计 22](#_Toc14336)

[4.1 架构概述 22](#_Toc3444)

[4.2 URL 路由设计 23](#_Toc14424)

[4.3 模型设计 23](#_Toc22494)

[4.4 视图设计 24](#_Toc29420)

[4.5 模板设计 24](#_Toc26201)

[5. 接口设计 25](#_Toc27132)

[5.1 接口概述 25](#_Toc579)

[5.1.1接口设计原则 25](#_Toc22796)

[5.1.2接口命名规范 26](#_Toc6727)

[5.2 接口列表 27](#_Toc22663)

[5.3 接口示例 27](#_Toc24546)

[6. 数据库设计 28](#_Toc27564)

[6.1 数据库概述 28](#_Toc27181)

[6.2 数据表设计 28](#_Toc12989)

[6.3 数据库关系图 30](#_Toc20419)

[6.4数据库触发器 31](#_Toc3011)

# 1. 引言

## 1.1 项目背景

### 1.1.1项目概述

动态作业管理系统是一种创新的教育管理平台，旨在通过作业拼机制优化学生的作业体验。系统允许学生发起和参与作业拼，与同学共同完成作业任务，提高协作能力和学习效果。教师可以通过系统动态发布作业、跟踪学生进度并进行个性化指导。系统主要包含以下功能模块：

用户管理：用户注册、登录、权限管理。

作业管理：教师发布作业、学生查看和提交作业、作业批改与反馈。

拼作业机制：学生发起和参与作业拼单，协作完成作业任务。

统计与分析：作业完成情况统计、学生作业拼情况分析。

### 1.1.2目标与范围

本项目的主要目标是通过引入拼作业机制，增强作业管理系统的互动性和灵活性，提升学生的学习兴趣和参与度，同时减轻教师的工作负担，提高作业管理的效率和效果。具体目标包括：

(1)增强学生互动：通过拼作业机制，促进学生之间的互动与合作，提高学习效果。

(2)提升教师管理效率：提供便捷的作业发布、批改和反馈功能，减轻教师的工作负担。

(3)个性化学习支持：根据学生的学习进度和表现，提供个性化的作业和指导建议。

本项目的实施范围包括：

(1)系统功能开发：完成用户管理、作业管理、拼作业机制、通知与提醒、统计与分析等模块的开发。

(2)系统集成与部署：将系统集成并部署到指定的服务器环境，确保系统的稳定运行。

(3)用户培训与支持：为系统用户（包括教师和学生）提供培训和技术支持，确保其能够熟练使用系统。

(4)系统维护与升级：在系统上线后，提供持续的维护和升级服务，及时修复系统中的问题并增加新功能。

## 1.2 目标读者

本设计文档的目标读者包括以下角色及其职责：

(1)项目经理

职责：项目的总体规划与管理，确保项目按时完成并符合预期质量。

作用：使用本设计文档了解项目的技术架构和设计思路，以便有效地进行项目管理和决策。

(2)软件架构师

职责：设计和定义系统的整体架构，确保系统的可扩展性和可维护性。

作用：使用本设计文档来确定和评审系统的架构设计，确保符合最佳实践和项目需求。

(3)开发人员

职责：根据设计文档实现系统功能，编写和维护代码。

作用：使用本设计文档作为编码参考，确保实现的功能符合设计要求。

(4)测试人员

职责：设计和执行测试用例，确保系统的质量和稳定性。

作用：使用本设计文档来理解系统的功能和架构，以便设计有效的测试用例和测试计划。

(5)运维人员

职责：部署和维护系统，确保系统的正常运行和性能优化。

作用：使用本设计文档来了解系统的部署要求和配置方法，确保系统的稳定运行。

(6)产品经理

职责：定义和管理产品需求，确保产品满足用户需求和市场需求。

作用：使用本设计文档来确认系统的功能设计是否符合产品需求。

## 1.3 文档概述

# 2. 总体架构设计

## 2.1 系统架构概述

### 2.1.1移动端架构概述

(1)架构介绍

MVVM（Model-View-ViewModel）是当前Android开发中最正式的一种架构模式。它通过将用户界面（UI）与业务逻辑和数据逻辑分离，提升代码的模块化、可维护性和可测试性。MVVM 架构的核心在于它将UI更新和数据管理分离开来，通过数据绑定和观察者模式实现View和ViewModel之间的通信。

(2)组件介绍

①Model

职责：处理管理应用的数据，包括从网络或本地数据库获取数据、业务逻辑处理等。

角色：通常由数据模型类和数据管理类（如Repository）组成，负责数据的获取、存储和更新。

②View

职责：显示UI并与用户交互，包括布局(XML)、活动（Activity）、片段（Fragment）及各种UI控件。

角色：View监听ViewModel中的数据变化，并根据变化更新界面。

③ViewModel

职责：作为View和Model之间的桥梁，处理UI相关的逻辑，将数据从Model传递给View。

角色：ViewModel包含LiveData或其他观察者模式，用于在数据变化时通知View。

(3)架构优势

分离关注点：将UI逻辑、业务逻辑和数据逻辑分离，使代码更易于维护和扩展。

双向数据绑定：通过数据绑定机制，View和ViewModel之间的数据同步更简洁高效。

提升可测试性：业务逻辑集中在ViewModel和Model中，使单元测试更加方便。

更好的状态管理：ViewModel负责管理UI状态，避免Activity和Fragment中的复杂状态管理逻辑。

(4)数据流

用户交互：用户在View上进行操作（如点击按钮、输入文本）。

事件处理：View将用户操作事件传递给ViewModel。

业务逻辑处理：ViewModel调用Model层处理业务逻辑，获取或更新数据。

数据更新：Model层通过LiveData或其他机制将数据变化通知ViewModel。

UI更新：ViewModel更新LiveData，View监听到数据变化后自动更新UI。

(5)具体实现细节

①Model层

数据源管理：Model层负责管理数据源，可以是网络API、数据库或者文件系统。它定义了获取、存储和更新数据的接口。

业务逻辑处理：在Model层进行数据的业务逻辑处理，确保数据的一致性和完整性。

②ViewModel层

状态管理：ViewModel持有LiveData对象，用于管理和暴露UI状态。LiveData是一个可观察的数据持有者，ViewModel通过它来向View发送数据更新通知。

无状态管理：ViewModel本身不持有对View的引用，避免了内存泄漏和UI状态不一致的问题。

异步操作：ViewModel可以通过协程或其他异步机制来处理耗时操作，确保UI的流畅性。

③View层

数据绑定：View监听ViewModel中LiveData对象的数据变化，并根据变化自动更新UI。数据绑定使用Android官方提供的Data Binding库。

生命周期感知：View（Activity/Fragment）在其生命周期内观察ViewModel中的LiveData，确保在View销毁时不再更新UI，避免内存泄漏。

(6)通信机制

数据绑定：通过Data Binding库，View和ViewModel之间可以实现双向数据绑定。数据绑定简化了UI组件和数据之间的同步操作。

观察者模式：LiveData采用观察者模式，View观察ViewModel中的LiveData，当数据发生变化时，View会自动收到通知并更新UI。

(7)实现流程

定义数据模型（Model）：设计数据结构和业务逻辑，定义数据获取和存储的接口。

创建ViewModel：在ViewModel中管理和处理数据，暴露LiveData供View观察。

设计UI（View）：创建Activity或Fragment，设置数据绑定，观察ViewModel中的LiveData。

实现数据绑定和观察：在View中绑定LiveData，当数据变化时自动更新UI。

测试和优化：通过单元测试和UI测试确保业务逻辑和UI的正确性，并优化性能

### 2.1.2网页端架构概述

Django项目遵循典型的MVC（模型-视图-控制器）模式，虽然在Django中习惯上称为MTV（模型-模板-视图），其基本架构如下：

（1） MVC架构

①模型（Model）：

功能：模型负责定义数据的结构和数据库操作的方式，使用Django的ORM进行数据库交互。

实现：在models.py文件中定义模型类，每个类对应数据库中的一个表，类的属性对应表中的字段。

常用操作\*\*：数据创建、读取、更新和删除（CRUD），使用Django提供的ORM方法如 filter()、get()、save()和 delete()。

② 视图（View）：

功能：视图包含业务逻辑，负责处理用户请求、与模型交互并返回响应。视图决定了显示给用户的内容和格式。

实现：在 views.py文件中定义视图函数或类视图。视图函数接受请求并返回响应。

常用操作：渲染模板、重定向、处理表单提交、验证数据等。

③. 控制器（Controller）：

功能：控制器负责处理用户输入，调用模型和视图来处理请求并返回最终的响应。在Django中，控制器的角色由URL路由系统和视图共同承担。

实现：在urls.py文件中定义URL模式，将URL路径映射到相应的视图函数。

常用操作：URL路由，URL与视图函数的映射。

（2）组件介绍

① URL路由：

功能：URL路由系统将用户请求的URL路径映射到相应的视图函数。它是控制器的一部分，负责分配请求到正确的视图。

实现：在 urls.py 文件中定义URL模式，通过path()函数将URL路径与视图函数关联。

优势：使得项目结构清晰，URL配置灵活，便于管理不同的视图和功能。

②模型（Model）：

功能：模型定义了数据的结构和数据库操作的方式。它使用Django的ORM进行数据库交互，使得开发者可以使用面向对象的方式操作数据库。

实现：在models.py文件中定义模型类，每个类对应数据库中的一个表，类的属性对应表中的字段。

优势：通过ORM简化数据库操作，支持多种数据库后端，提供了查询集API，易于扩展和维护。

③视图（View）：

功能：视图包含业务逻辑，负责处理请求、与模型交互并返回响应。视图决定了显示给用户的内容和格式。

实现：在 views.py文件中定义视图函数或类视图，处理请求并返回响应。可以使用Django提供的通用视图类简化常见任务。

优势：分离业务逻辑和表现层，支持函数视图和类视图，便于复用和测试。

④模板（Template）：

功能：模板用于定义网页的结构和内容，通过占位符和模板标签将动态数据插入到HTML中。模板引擎解析模板文件并生成最终的HTML页面。

实现：在 templates目录中存放HTML模板文件，通过视图函数渲染并返回给客户端。

优势：支持模板继承和复用，提供内置的模板标签和过滤器，便于动态生成内容。

（3）架构优势

关注点分离：模型、视图和模板分别处理数据、业务逻辑和用户界面，使得代码更易于维护和扩展。

高效开发：Django提供了大量内置功能和通用视图类，减少了重复代码，提高了开发效率。

灵活性和扩展性：Django的URL路由系统和ORM提供了灵活的配置和扩展能力，支持多种数据库和复杂的查询。

安全性：Django内置了多种安全功能，如防止SQL注入、跨站脚本攻击（XSS）和跨站请求伪造（CSRF）。

社区和文档支持：Django拥有一个活跃的社区和详尽的文档，提供了丰富的资源和支持。

（4） 具体实现细节

URL路由：在 urls.py文件中定义URL模式，通过 `path()` 函数将URL路径与视图函数关联。

模型定义：在 models.py文件中定义模型类，每个类对应数据库中的一个表，类的属性对应表中的字段。

视图函数：在views.py文件中定义视图函数或类视图，处理请求并返回响应。

模板渲染：在templates目录中存放HTML模板文件，通过视图函数渲染并返回给客户端。

## 2.2 模块划分

### 2.2.1各模块功能概述

#### 2.2.1.1移动端模块功能概述

(1) 用户界面模块（UI Module）

功能概述：负责显示应用的用户界面，并处理用户的交互。包括Activity、Fragment、RecyclerView、ViewPager等UI组件。使用Data Binding库实现数据绑定，以简化UI更新过程。监听ViewModel中的LiveData，自动响应数据变化并更新UI。

子模块如下：

LoginActivity:APP的LOGO界面以及登录界面

CourseCenterFragment：课程中心界面模块，内部还有课程信息、课程作业等子模块

HopeSquareFragment：希望广场界面模块，内部还有广场中心、我的发布等子模块

PersonCenterFragment：个人中心界面模块，内部还有个人资料、切换账号等子模块

(2) 视图模型模块（ViewModel Module）

功能概述：作为View和Model之间的桥梁，处理UI相关的逻辑和数据。通过LiveData或StateFlow持有和管理UI状态。响应用户操作并更新数据，将结果通过LiveData通知View。无状态管理，避免对View的直接引用，防止内存泄漏。

子模块如下：

LoginViewModel：处理用户登录相关的业务逻辑

CourseCenterViewModel：处理用户课程相关的业务逻辑，内部还有处理课程信息、课程作业等业务逻辑的子模块

SquareCenterViewModel：处理希望广场相关的业务逻辑，内部还有处理我的发布、作业拼详情等业务逻辑的子模块

PersonCenterViewModel：处理个人中心相关的业务逻辑，内部还有处理账号切换等业务逻辑的子模块

(3) 数据模型模块（Model Module）

功能概述：负责处理和管理应用的数据，包括从网络或本地数据库获取数据。包括数据实体类和业务逻辑类。采用Repository模式，集中管理数据操作，提供统一的数据接口给ViewModel。

子模块如下：

UserRepository：管理用户数据的获取和存储。

CourseRepository：管理课程相关数据的获取和存储。

HelpRepository：管理作业拼相关数据的获取和存储。

(4) 网络模块（Network Module）

功能概述：负责处理所有与网络相关的操作，如API请求、响应处理等。包括网络请求接口的定义和实现，使用Retrofit等库进行网络通信。提供统一的API接口，供Repository调用以获取网络数据。

子模块如下：

RetrofitClient：初始化和管理网络请求库Retrofit的实例。

ApiService：定义网络请求接口。

(5) 数据库模块（Database Module）

功能概述：负责处理应用的数据存储和检索。定义DAO接口和数据库管理类。提供数据的存储和查询接口，供ApiService调用。

子模块如下：

dao\_android：定义数据库的增删改查操作。

noviews\_android：处理数据请求，提供数据接口。

(6) 工具模块（Tool Module）

功能概述：包含应用中通用的工具类、帮助类和常量定义。提供通用的功能和服务，供其他模块使用。

子模块如下：

Tool：包含通用的工具方法，如字符串处理、日期处理等。

(7) 资源模块（Resources Module）

功能概述：管理应用的资源文件，包括布局文件（XML）、字符串资源、图像资源、样式和主题等。提供统一的资源管理和访问方式。

子模块如下：

Layout：定义应用的布局文件。

Values：定义字符串资源、颜色资源、样式和主题等。

#### 2.2.1.2网页端模块功能概述

（1） 静态文件static

功能：存放项目中的静态文件资源，如CSS、JavaScript和图片等。

作用：用于前端页面的样式和脚本处理，使页面具备良好的用户体验和交互效果。

实现细节：静态文件通常位于 static目录中，开发过程中使用 {% static 'path/to/file' %}模板标签来引用这些资源。

（2）模版文件templates

功能：存放HTML模板文件，用于定义网页的结构和内容。

作用：通过Django模板语言（DTL）将动态数据嵌入到HTML中，生成用户所看到的网页。

实现细节：模板文件位于 templates目录中，可以使用模板继承和包含机制提高代码复用性和维护性。在视图中使用 `render()` 函数来渲染模板并返回响应。

子模块：

course\_detail.html：显示课程详情的模板。

help.html：求助页面的模板。

homework.html：作业详细页面的模板。

hope.html：希望广场页面的模板。

login.html：登录页面的模板。

main.html：主页面的模板。

（3） 数据模型（数据库）模块models

功能：定义数据模型，使用Django的ORM进行数据库操作。

作用：描述应用中的数据结构以及数据之间的关系，通过ORM简化数据库的操作和查询。

实现细节：在 models.py文件中定义模型类，每个类对应数据库中的一个表，包含数据字段和关系定义。模型类提供了CRUD操作的方法，如save()、delete()和 filter()。

（4）视图路由 urls

功能：定义URL路由，将请求路径映射到相应的视图函数。

作用：管理和控制应用的URL配置，使得用户请求能够正确地被处理。

实现细节：在 urls.py文件中使用path()和re\_path()函数定义URL模式，将URL路径与视图函数或类视图关联。可以通过include()函数将应用的URL配置导入到项目的主URL配置中。

（5）视图模块 views

功能：包含视图函数或类视图，处理请求、与模型交互并返回响应。

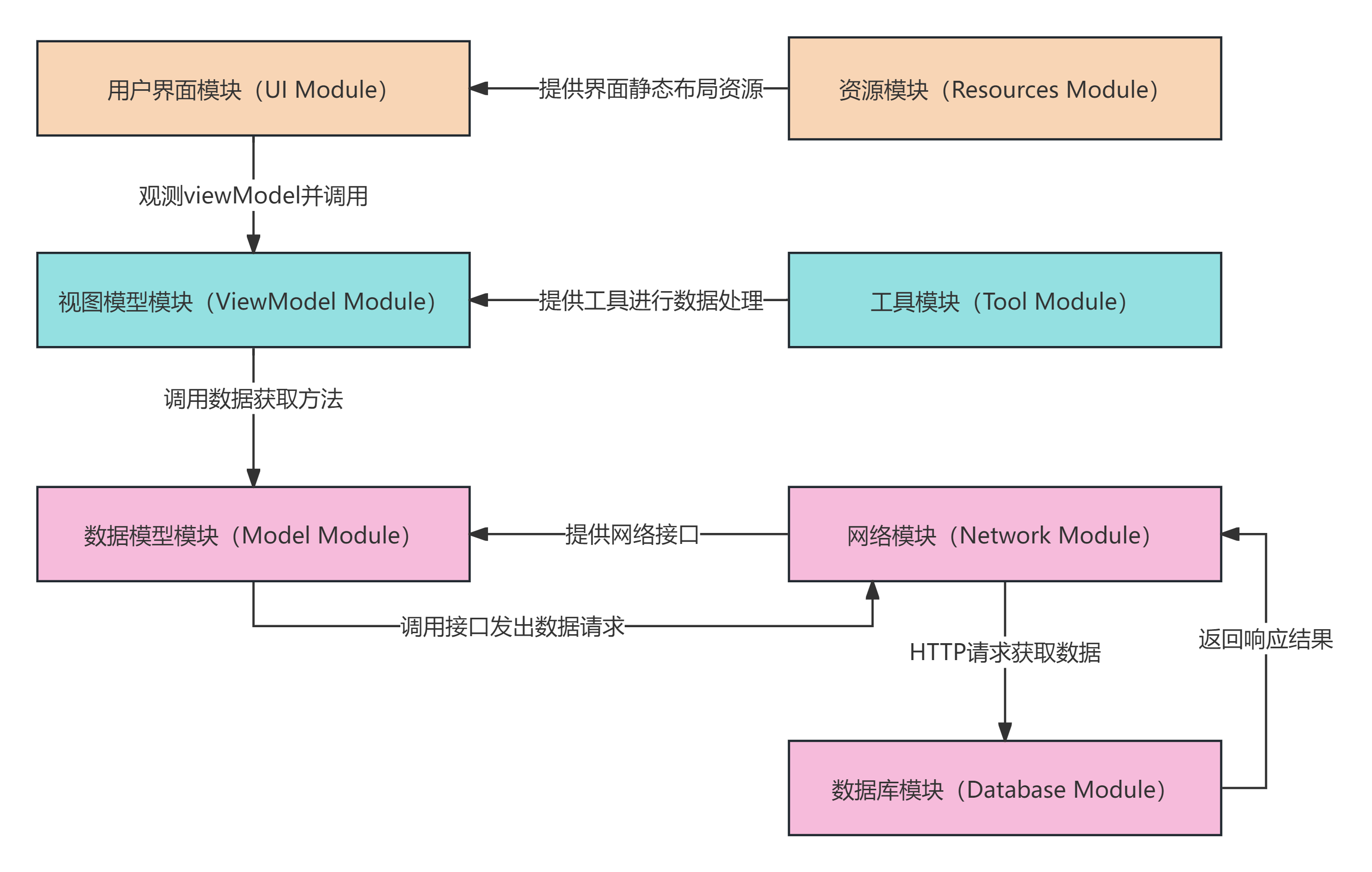
作用：处理业务逻辑，决定返回给用户的内容和格式。

实现细节：在views.py文件中定义视图函数或类视图，接收请求并处理业务逻辑。视图函数可以返回渲染的模板、JSON数据或重定向到其他视图。视图与模型交互获取数据，并将数据传递给模板以生成最终的HTML页面。

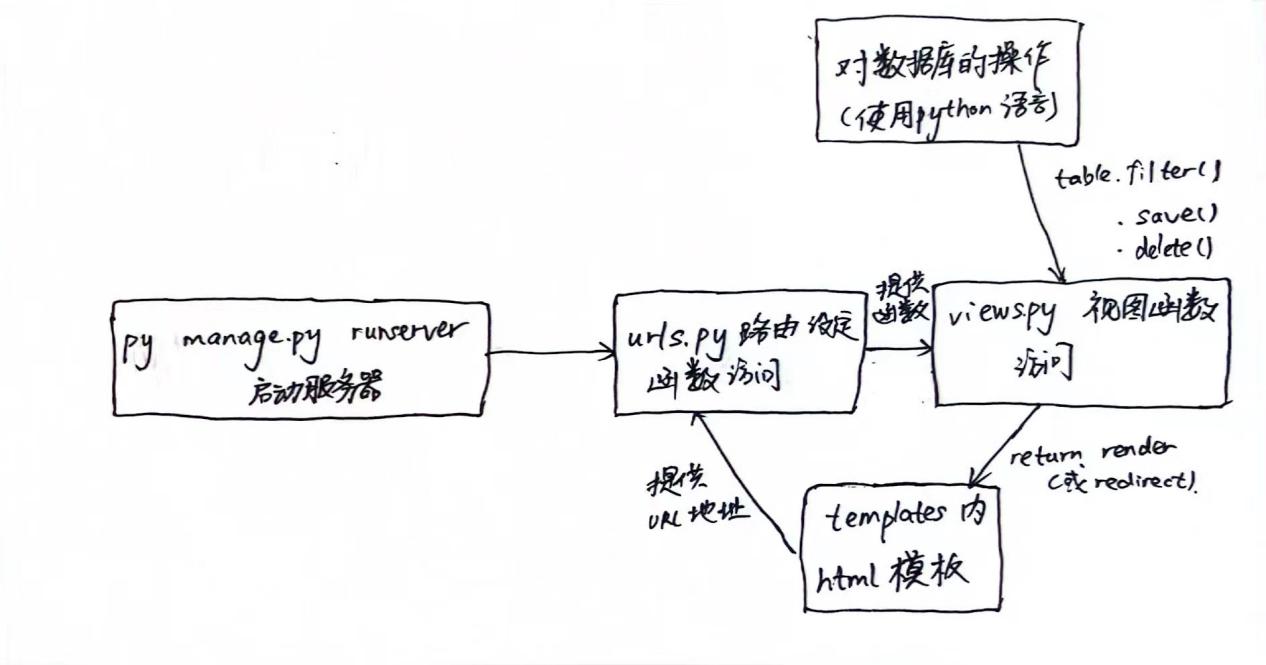
通过这些模块的协同工作，Django项目能够实现高效、灵活和可扩展的Web应用开发。各模块分工明确，职责清晰，有助于项目的维护和扩展。

### 2.2.2模块间关系图

#### 2.2.2.1移动端模块关系图



#### 2.2.2.2网页端模块关系图



## 2.3 技术选型

### 2.3.1移动端技术栈

(1)开发语言和框架

语言选择：Kotlin

理由：Kotlin是一种现代化的、静态类型的编程语言，它与Java兼容并且在Android开发中有着较高的采用率。Kotlin相比Java具有更简洁、安全和表达力强的特性，可以提升开发效率和代码质量。

框架选择：Android Jetpack

理由：Android Jetpack是由Google提供的一组现代化的工具和库，旨在帮助开发者更轻松地构建健壮、高效的Android应用。它包括ViewModel、LiveData、Room、Navigation等组件，支持MVVM架构，提升应用的可维护性和可测试性。

(2)架构模式

MVVM（Model-View-ViewModel）

理由：MVVM是一种现代化的架构模式，能够有效地分离用户界面的逻辑和业务逻辑。通过ViewModel与View的分离，可以使代码更加清晰和易于测试，同时利用数据绑定机制简化UI更新流程，提升开发效率和代码质量。

(3)网络通信

库选择：Retrofit

理由：Retrofit是一个强大且易于使用的HTTP客户端库，专门用于Android和Java开发。它简化了与RESTful API的交互，并提供了强大的功能如请求的自动化序列化和反序列化、请求拦截、文件上传等，适合移动端应用的网络通信需求。

(4)数据库选择

远程数据库：MySQL（存储在阿里云服务器上）

理由：MySQL是一种稳定、可靠且广泛使用的关系型数据库管理系统（RDBMS），适合存储结构化数据。将数据库部署在阿里云服务器上，可以保证数据的安全性、可用性和持久性，同时通过云服务可以实现高可扩展性和灵活性。

(5)本地存储选择

本地存储：SharedPreferences

理由：SharedPreferences是Android平台提供的一种轻量级的本地存储解决方案，适合存储简单的键值对数据。它主要用于存储应用的配置信息、用户偏好设置等，并且操作简单、易于使用。对于移动端应用而言，SharedPreferences适合存储一些临时性的、不需要复杂结构或复杂查询的数据。

(6)依赖注入

库选择：Dagger Hilt

理由：Dagger Hilt是Google推出的依赖注入库，专门设计用于Android应用的依赖注入。它通过简化依赖注入的实现、提供编译时依赖检查和自动化依赖注入，帮助开发者更好地管理应用的依赖关系，提升应用的可维护性和扩展性。

(7)界面设计工具

界面设计工具：Mockplus

理由：Mockplus是一款专业的快速原型设计工具，适合于移动端应用的UI/UX设计。它提供了丰富的组件库、易于使用的拖拽式界面设计，能够快速创建和演示应用的交互原型。Mockplus支持多平台导出和共享，有助于团队成员之间的协作和沟通，提升设计效率和最终产品的用户体验。

(8)版本控制和团队协作

版本控制工具：Git

理由：Git是目前最流行的版本控制系统，能够有效地管理代码的版本、协调团队开发、处理分支和合并操作，是团队协作和代码管理的基础工具。

### 2.3.2网页端技术栈

### （1） 网页：HTML + CSS

**标准化**：HTML 和 CSS 是构建网页的基本标准，得到广泛支持和使用。

**易学习和使用**：语法简单，易于上手，对于初学者非常友好。

**跨平台兼容性**：可以在各种浏览器和设备上运行，确保网站的普适性。

**可访问性**：通过语义化的 HTML 标签和合适的 CSS，可以提高网页的可访问性，使得更多用户能够使用网站，包括残障用户。

**可扩展性和可维护性**：HTML 和 CSS 的结构化和分离性使得代码更易于管理和维护，同时也方便未来的扩展和修改。

### （2） 框架：Django

**快速开发**：Django 提供了一系列内置工具和功能，如用户认证、管理后台等，减少了开发时间。

**安全性**：Django 在设计时考虑了安全问题，内置了防范 SQL 注入、跨站请求伪造、跨站脚本攻击等安全机制。

**强大的 ORM**：Django 的 ORM（对象关系映射）简化了数据库操作，使得开发者可以使用 Python 代码进行数据库查询和操作，而无需编写 SQL 语句。

**灵活性**：Django 的模块化设计允许开发者根据需求选择使用其功能组件，支持快速原型开发和大型项目的扩展。

**社区支持**：Django 拥有一个活跃的社区和丰富的文档资源，使得开发过程中遇到的问题更容易解决。

### （3）数据库：MySQL

**性能优秀**：MySQL 以其高性能和高可靠性而著称，能够高效地处理大量数据和高并发访问。

**开源免费**：MySQL 是开源软件，无需支付任何费用，适用于各种规模的项目。

**跨平台支持**：MySQL 可以在各种操作系统上运行，如 Windows、Linux 和 macOS，具有很好的跨平台兼容性。

**易用性**：MySQL 提供了直观的管理工具和简单的配置选项，使得数据库管理更加便捷。

**丰富的生态系统**：MySQL 拥有大量的第三方工具和扩展，能够满足不同的业务需求和技术场景。

### （4）界面设计：Mockplus

**快速原型设计**：Mockplus 提供了丰富的 UI 元素和模板，使得设计师能够快速创建高保真原型。

**协作功能**：支持团队协作设计，允许多位设计师同时在同一个项目中工作，提高了工作效率。

**交互设计**：Mockplus 支持创建复杂的交互效果，帮助设计师更直观地展示用户体验。

**简洁易用**：界面友好，操作简便，即使是初学者也能快速上手。

**跨平台支持**：Mockplus 支持 Windows 和 macOS 系统，设计文件可以在不同平台间无缝切换。

### （5）版本控制：Git

**分布式版本控制**：每个开发者都有一个完整的代码库副本，支持离线工作，并能够轻松合并代码和管理版本。

**高效的分支管理**：Git 的分支功能强大且灵活，允许开发者在独立的分支上进行开发、测试和集成，而不影响主干代码。

**回退和恢复**：Git 可以方便地回退到历史版本，恢复误操作，保证代码的安全性和完整性。

**协作开发**：Git 提供了丰富的协作工具和流程，如 pull request 和 code review，促进团队间的协作和代码质量的提高。

**开源和免费**：Git 是开源软件，广泛使用于各种项目，无论是个人小项目还是大型企业项目。

# 3.移动端设计

## 3.1 组件设计

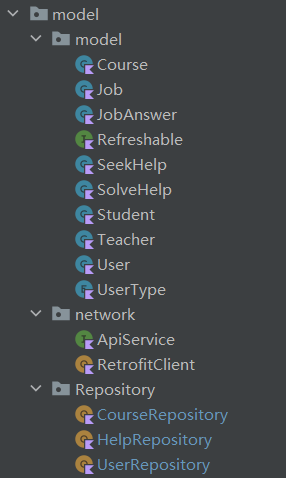
(1)Model 层设计

功能：处理数据的获取和存储，管理业务逻辑。

实现：包括Repository（数据仓库）、网络请求接口、数据模型等组件。

描述：数据模型中定义所需数据结构，使用Repository模式统一管理数据的获取和存储操作，封装数据访问接口给ViewModel使用，网络请求接口提供请求数据功能。

实际结构如下



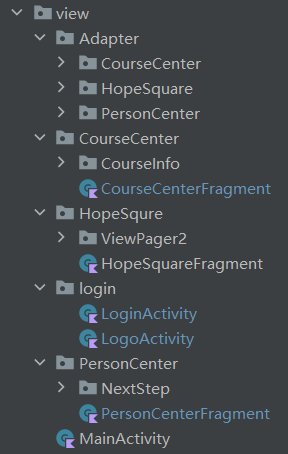
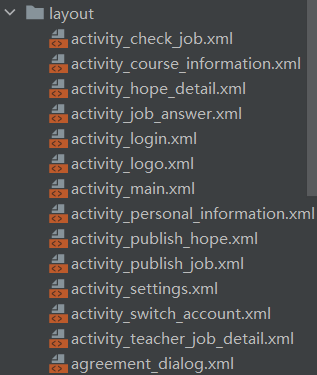
(2)View 层设计

功能：负责用户界面的展示和用户输入的响应。

实现：Activity、Fragment等Android组件，通过布局文件定义界面结构。

描述：使用XML布局文件定义静态界面，使用Activity、Fragment进行动态界面变化，通过数据绑定绑定ViewModel中的数据。

实际结构如下



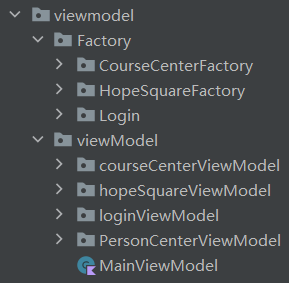
(3)ViewModel 层设计

功能：连接View和Model，处理业务逻辑和UI状态管理。

实现：持有和管理UI相关的数据（如LiveData、StateFlow），响应View的用户操作。

描述：使用LiveData实现数据的观察和通知机制，使得View能够自动更新UI。

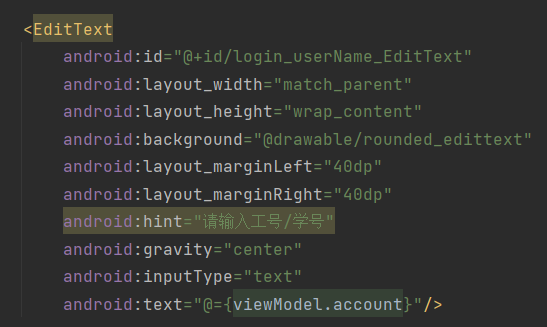
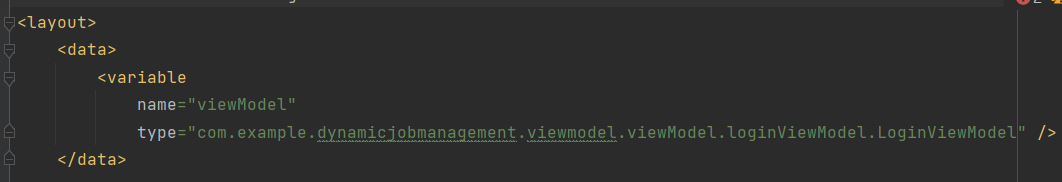
实际结构如下



## 3.2 数据绑定

Android中的数据绑定技术（Data Binding）允许将UI组件直接绑定到应用程序的数据源。这种技术大大简化了代码，减少了手动更新UI的需要，提高了开发效率和代码可维护性。

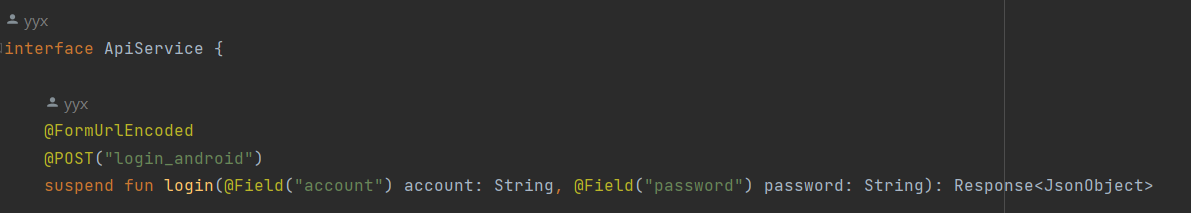
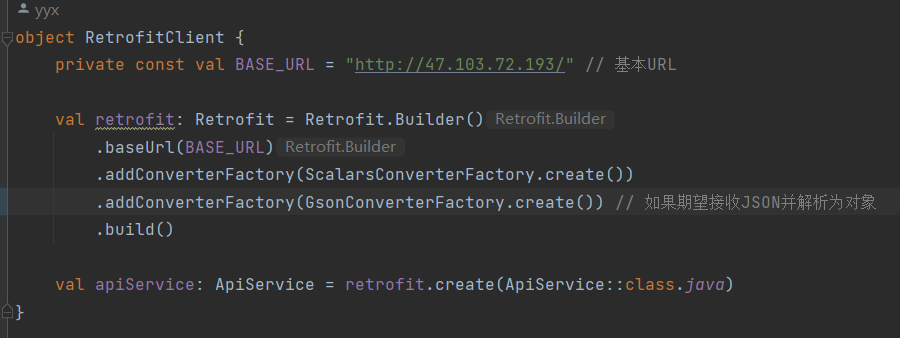
实例如下：



## 3.3 网络通信

Retrofit是一个强大且流行的网络库，适用于Android应用程序的网络通信。它简化了与RESTful API的交互，提供了强大的功能如请求的自动化序列化和反序列化、请求拦截、文件上传等。

实例如下：



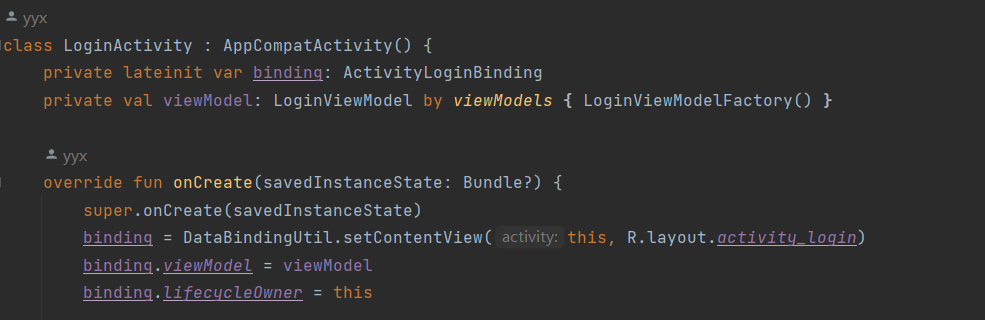
## 3.4 生命周期管理

在MVVM架构中，为了避免内存泄漏和及时释放资源，需要合理管理ViewModel的生命周期，以及确保View层能够正确地观察和监听ViewModel中的数据变化。

使用ViewModel的生命周期感知组件（Lifecycle-aware Components）：例如使用ViewModel中的LiveData与View中的Observer结合，确保数据更新仅在视图处于活动状态时进行，避免了不必要的数据更新和内存泄漏问题。

合理管理Activity和Fragment的生命周期：确保在View层中正确处理生命周期方法，如onCreate、onStart、onStop等，以便在合适的时机连接和断开ViewModel的数据绑定。

实例如下：



# 4. 网页端设计

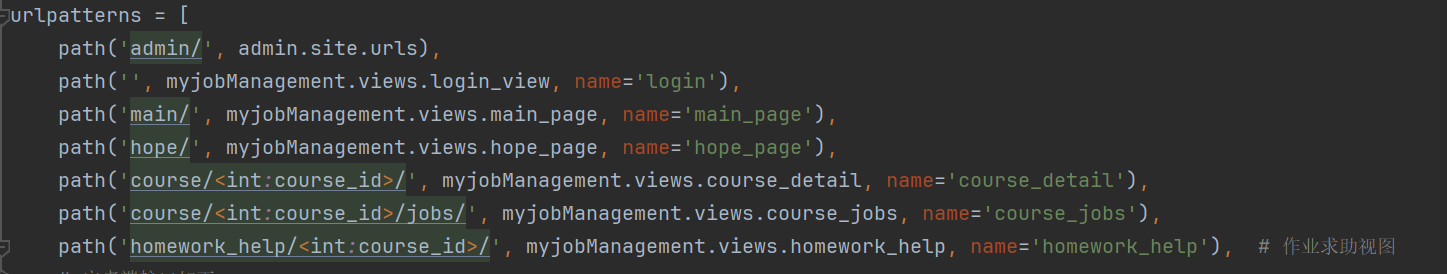
## 4.1 架构概述

Django 是一个开放源代码的 Web 应用框架，采用了“Model-Template-View” (MTV) 架构模式。MTV 架构类似于经典的 MVC (Model-View-Controller) 模式，MTV 架构模式清晰地分离了数据层、业务逻辑层和展示层，使得开发者可以专注于每个层次的具体实现，提高了代码的可读性和可维护性。Django 提供了丰富的内置功能和工具，大大简化了 Web 应用的开发过程，适用于从小型项目到大型企业应用的各种场景。

## 4.2 URL 路由设计

URL 路由在 Django 框架中起着至关重要的作用，它负责将客户端的请求映射到相应的视图函数或类视图。Django 使用一个称为 URLconf (URL configuration) 的机制来定义 URL 模式和视图之间的映射关系。使用正则表达式或路径转换器来匹配 URL 模式，从而确定哪个视图应该处理特定的请求。当服务器接收到请求时，Django 根据 URLconf 将请求分发到相应的视图。

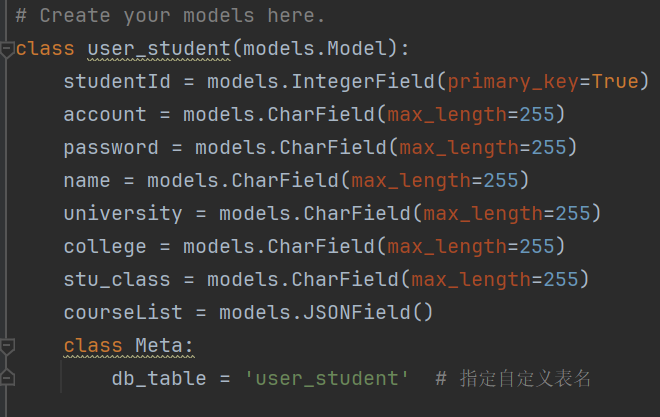
实例如下：



## 4.3 模型设计

Django ORM (Object-Relational Mapping) 是 Django 框架的重要组成部分，它允许开发者使用 Python 代码与数据库进行交互，而无需编写 SQL 语句。通过定义模型类，Django ORM 会自动将这些类映射到数据库表，并生成相应的 SQL 查询。

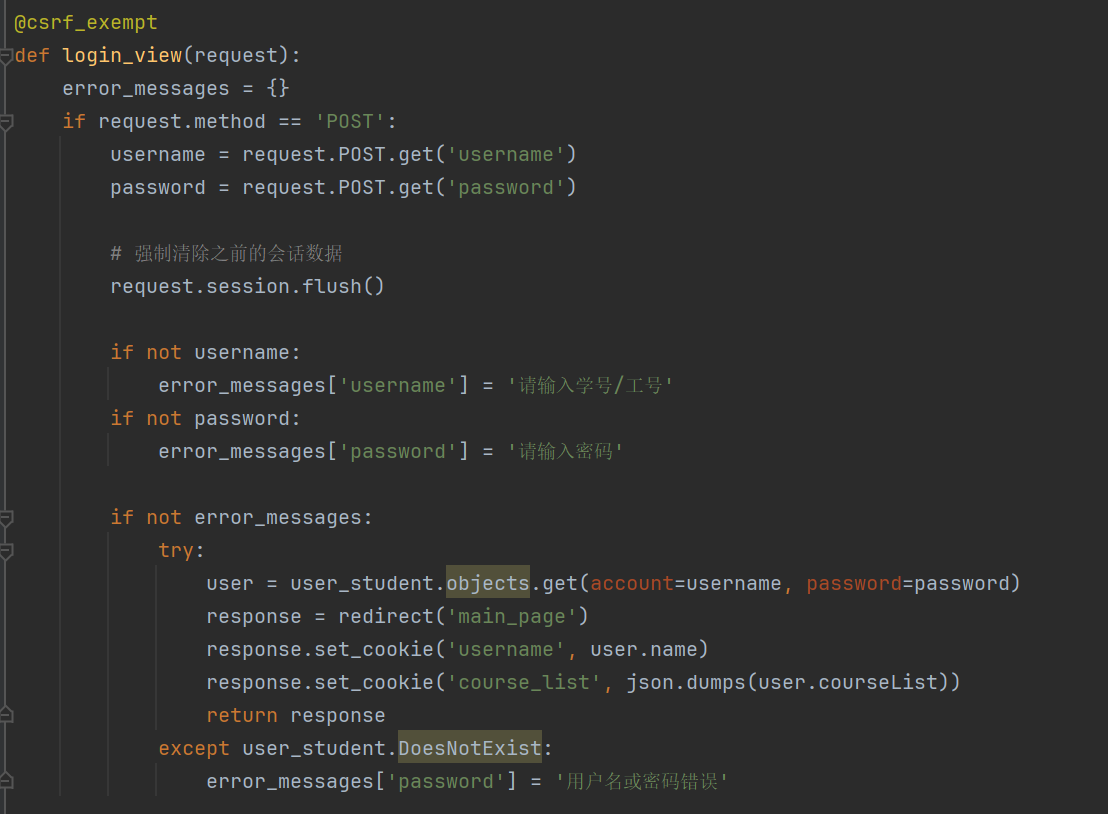
实例如下：



## 4.4 视图设计

在 Django 框架中，视图（View）是处理用户请求并返回响应的核心组件。视图包含业务逻辑，决定如何处理请求、获取数据、并生成响应。Django 提供了两种主要类型的视图：基于函数的视图（Function-Based Views, FBV）和基于类的视图（Class-Based Views, CBV）。

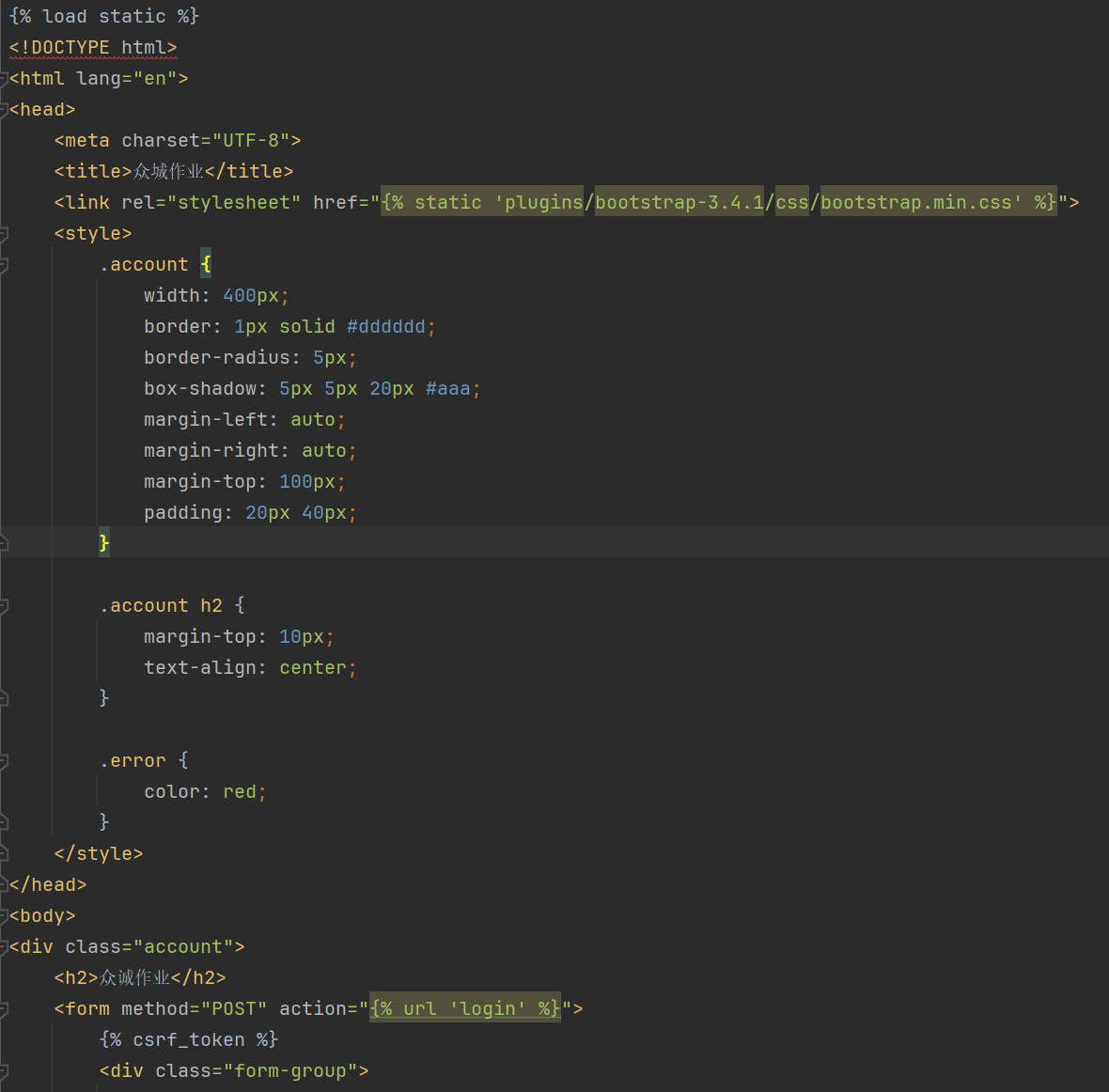
实例如下：



## 4.5 模板设计

Django 的模板系统旨在通过使用模板语言来生成动态 HTML 页面。模板语言包括变量、标签、过滤器和注释四个主要部分。这些元素允许开发者将视图中的数据传递给模板，并在模板中进行渲染，从而生成最终的 HTML 页面。变量用于显示动态数据，标签用于控制流程（如条件判断和循环），过滤器用于格式化数据，而注释用于添加开发者的备注，这些注释不会出现在最终的 HTML 页面中。

实例如下：



# 5. 接口设计

## 5.1 接口概述

### 5.1.1接口设计原则

(1)明确性和简洁性：接口应该具有清晰的功能定位，每个接口应该只做一件事情，并保持尽可能简洁。

(2)一致性：保持接口设计的一致性，包括参数顺序、命名风格等，有助于提高开发效率和降低错误率。

(3)可扩展性：设计接口时考虑未来可能的扩展需求，接口应该易于扩展和修改而不影响现有的调用方。

(4)健壮性：接口应该能够处理各种异常情况和边界条件，避免不必要的错误和中断。

(5)文档化：为每个接口提供清晰的文档和注释，包括接口的用途、输入参数、输出结果以及可能的异常情况说明。

(6)依赖倒置原则：接口应该依赖于抽象而不是具体实现，这样可以提高代码的灵活性和可测试性。

(7)安全性：确保接口设计考虑到安全性问题，包括数据加密、身份验证和授权等方面的需求。

### 5.1.2接口命名规范

(1)使用动词：接口名称应该清晰地描述其功能，使用动词开头可以直观地表达接口的操作，如 getUserInfo()。

(2)使用清晰的名词：接口名称应该使用能准确描述操作对象的名词，如 createUser()。

(3)遵循驼峰命名法：使用驼峰命名法来组织接口名称，保持统一和可读性，如 getUserInfoById()。

(4)避免缩写和简写：尽量避免使用缩写和简写，确保接口名称的完整性和清晰性，如 getCustomerInformation() 要优于 getCustInfo()。

(5)使用一致的术语和约定：在整个项目中保持一致的接口命名约定，便于团队成员之间的沟通和协作。

(6)避免过于泛化的命名：接口名称应该具体明确，避免过于泛化的命名，例如 manageData() 应该明确指出具体操作。

## 5.2 接口列表



## 5.3 接口示例

(1)请求示例

请求接口：AddLikeToSeekHelp

方法： POST

URL： /AddLikeToSeekHelp\_android

请求参数：seekHelpId：123

(2)响应示例

响应接口：AddLikeToSeekHelp

状态码： 200 OK

响应体：

{

"result": "success",

}

# 6. 数据库设计

## 6.1 数据库概述

在设计本项目的数据库时，我们选择了使用关系型数据库MySQL，存储在阿里云服务器上。以下是选择MySQL作为数据库类型的理由：

(1)成熟稳定：MySQL是一个成熟稳定的关系型数据库管理系统，被广泛应用于各种规模的应用和系统中。

(2)开源免费：MySQL是开源软件，提供免费使用许可，降低了项目的运行成本。

(3)社区支持：MySQL拥有活跃的开发者社区和用户社群，可以获得丰富的文档、教程和社区支持。

(4)跨平台兼容性：MySQL具有良好的跨平台兼容性，可以在多种操作系统上运行，包括Linux、Windows和macOS等。

(5)性能优化：MySQL提供了丰富的性能优化选项和工具，支持索引、事务和复制等功能，能够满足项目中复杂的数据处理需求。

(6)安全性：MySQL具备高级的安全功能和管理工具，支持用户权限管理、数据加密等安全措施，保障数据的安全性和完整性。

## 6.2 数据表设计









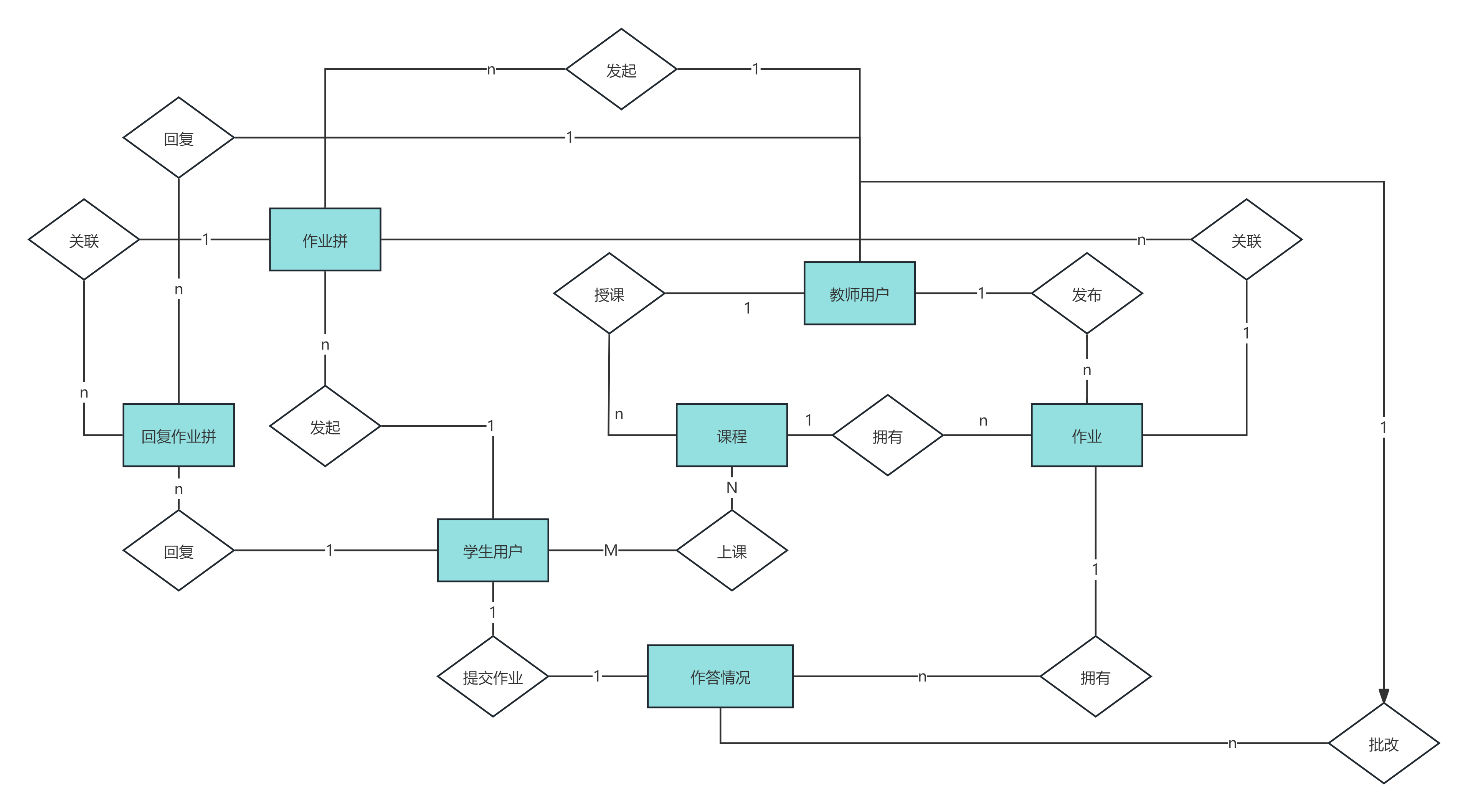






## 6.3 数据库关系图

ER 图如下



## 6.4数据库触发器

