# 《软件体系结构与设计模式》

# 课程设计报告

**学 院:** **遥感信息工程学院**

**班 级:**   **空信特软班2022**

**组长学号:**  **2022302191584**

**组长姓名:**   **周珏帆**

**组员学号:**  **2022302051178**

**组员姓名:**   **曹尚伟**

**实习地点: 3区1教509**

**2024年11月11日**

# 功能需求及环境限制分析。

## 1.1功能需求分析

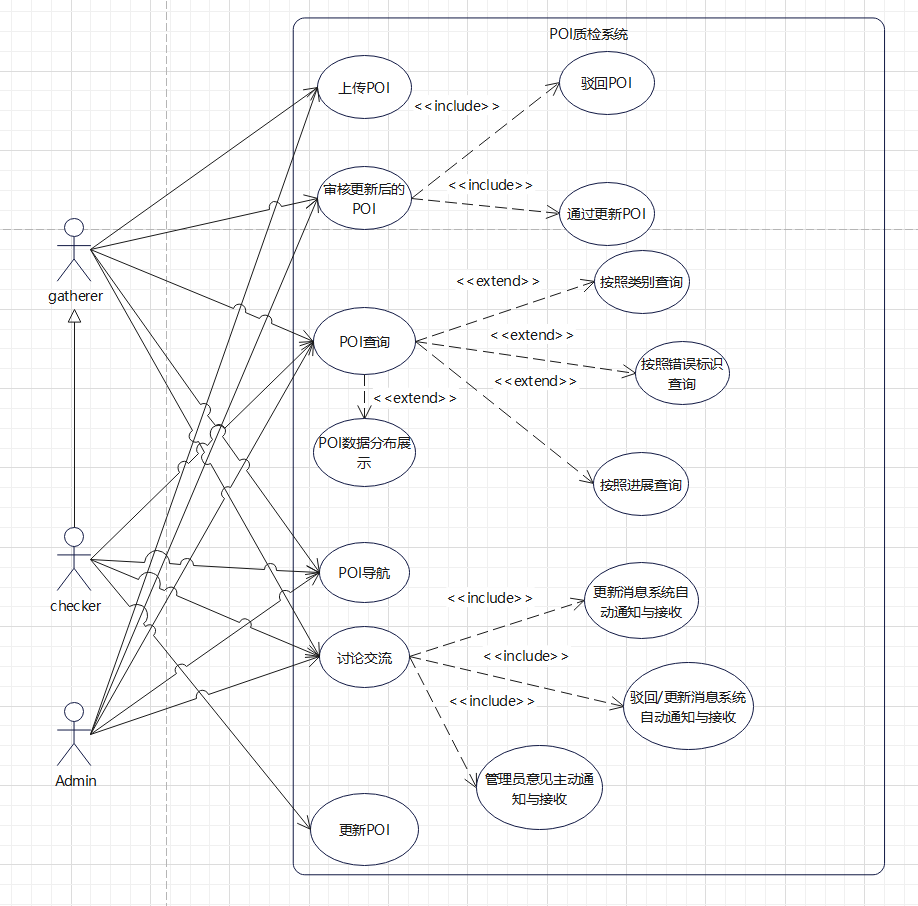
### 1.1.1脑图

经过前期的小组讨论与分析，我们通过以下脑图初步表示出需求：



### 1.1.2用例图

为更好地服务后续的开发工作，现遵循UML 2.0规范对需求进行建模：



上面的用例图体现了项目的综合需求：

地理信息平台支持（地图查询、显示、路径计算、导航）

采集者：上传POI照片、拍照位置、 POI分类及基本描述信息

核验者：发现错误后，标识出错POI,上传更新后的POI照片、拍照位置、 POI分类及基本描述信息

系统向采集者发消息，通知有错误数据需订正，采集者复核后可选择更新或驳回

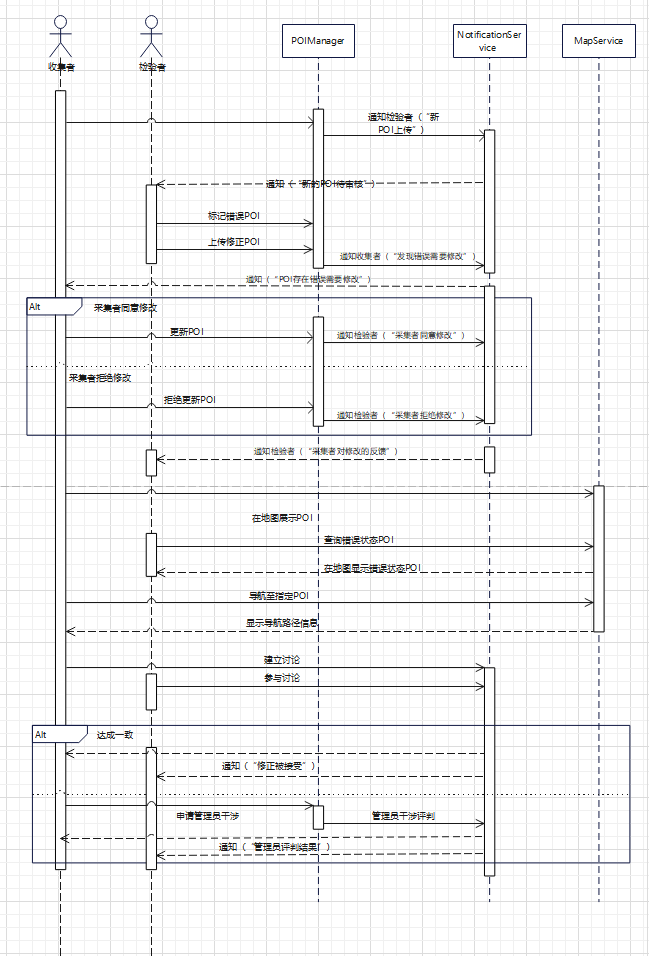
系统向核验者者发消息，通知采集者处理结果

可在地图上按类别查询显示POI数据分布、错误标识及进展进展

可使用地图平台导航功能，导航至指定POI位置

支持采集者与核验者建立讨论，交流错误标识详情，无法达成共识时由后台决定支持哪一方

### 1.1.3顺序图



该时序图主要体现了项目的动态交互过程和主要业务流程，其中角色分为收集者和检验者；实体/对象为3个模块（POIManager。NotificationService，MapService）

采集者（Gatherer）主要负责上传POI数据，包括照片、位置、分类和描述信息，并在收到核验者（Checker）的错误标记后对数据进行修正或拒绝，此外还可参与讨论以解决数据争议。核验者的职责是审核采集者上传的POI数据，标记错误并上传修正数据，同时接收采集者的反馈并参与讨论。系统的核心对象包括POIManager，作为数据管理模块，负责处理POI数据的上传、核验和更新；NotificationService，用于实现采集者和核验者之间的异步消息通知；以及MapService，提供POI数据的地图分布展示、错误状态可视化和导航功能。这些角色和对象通过协作，完成数据的闭环管理和交互，确保系统流程高效、可靠。

## 1.2环境限制分析

### 1.2.1开发环境概述

开发框架：使用 Vue 和 SpringBoot，采用 UniApp 技术框架开发。

运行环境：微信开发者工具模拟器，云端部署后同时支持在 iOS 和 Android 的移动端环境下运行和测试。

移动端设备：最终小程序可部署到 Android 移动端（例如 HONOR），并通过微信小程序运行。

服务器环境：提供两种形式的后台服务环境（本地和云端部署）。

开发工具：前端开发采用UniApp官方IDE HBuilderX，后端开发采用IntelliJ IDEA,数据库采用MySQL 8.0CE，缓冲数据库采用Redis.

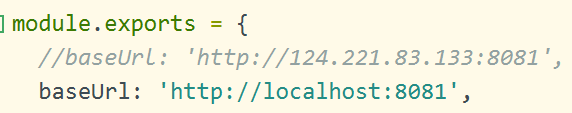
### 1.2.2本地部署环境限制

**在本地端运行服务器时，系统主要依赖以下组件：**

SpringBoot 作为本地应用后端，提供 RESTful API 接口支持。**此模式下需要启动本地后台和本地Redis、MySQL.**

Redis 用作缓存服务。

MySQL 用作数据存储服务。



本地部署需访问localhost，端口为8081

**兼容性和网络限制**

操作系统兼容性：本地部署的服务器与移动端设备无直接操作系统兼容性问题，网络连接可通过局域网配置。

端口配置：本地服务器需开放相关端口（默认 Spring Boot 为 8080、Redis 为 6379、MySQL 为 3306）以便于移动端小程序访问。

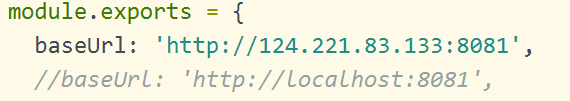
移动端兼容性：微信小程序兼容 iOS 和 Android，但由于测试绑定设备限制为 Android，需优先在 Android 设备上测试并确认运行。

网络依赖：本地部署依赖于内网或本地网络环境，若在外网访问则需进行公网配置并确保外部 IP 可用。

### 1.2.3腾讯云服务器部署环境限制

若选择腾讯云服务器进行部署，则所有后台服务（包括 SpringBoot、Redis、MySQL）都在云端环境中运行，且通过公网 IP 访问。**此模式下不需要启动本地后台和本地Redis、MySQL.**

云服务器公网IP：124.221.83.133

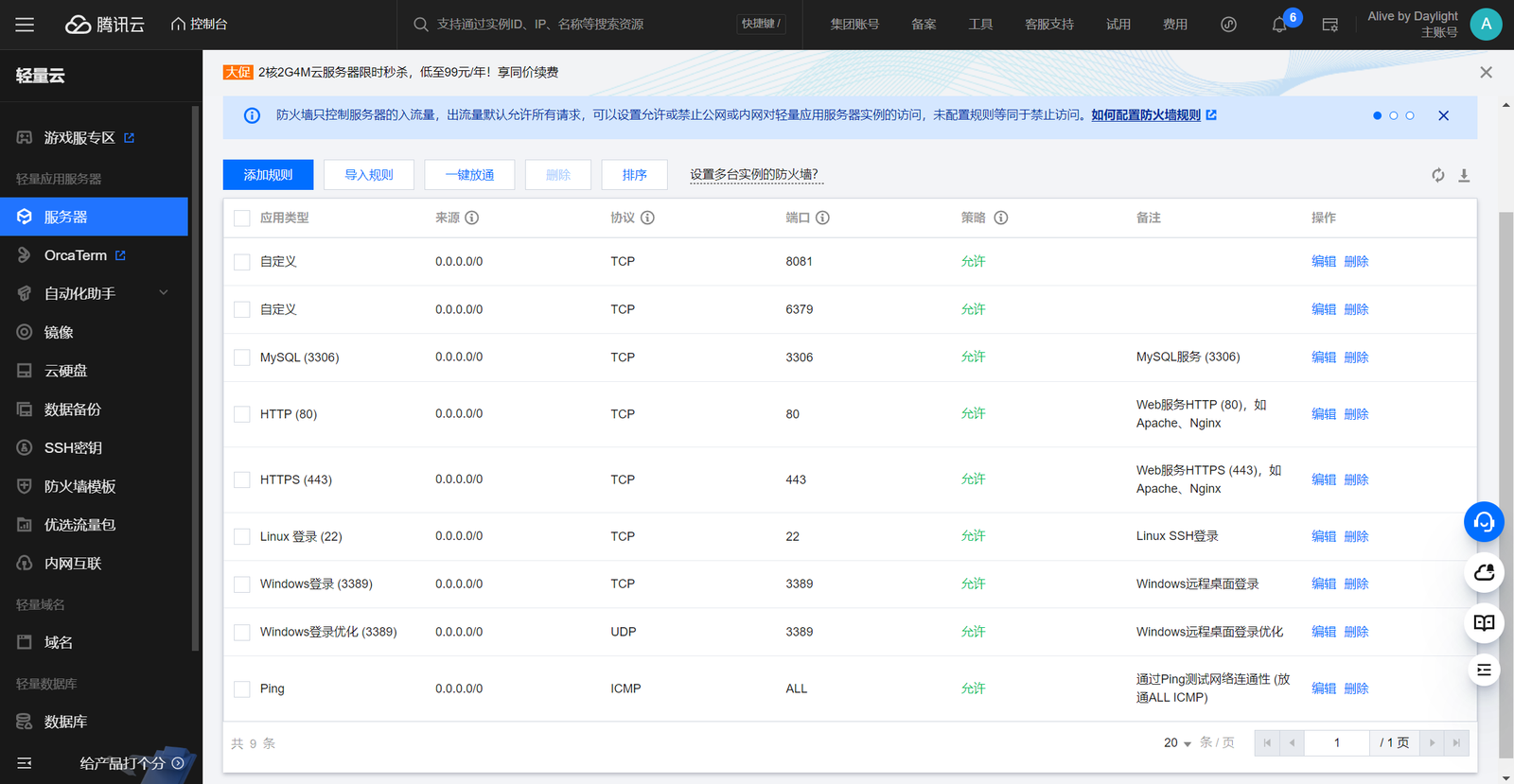


云端部署需访问云服务器公网ip，端口为8081

**腾讯云环境配置要求**

部署要求：需要在云服务器上按照教程完成部署、环境配置、端口开放等操作，确保服务在腾讯云上正常启动。

端口开放：需在腾讯云控制台中开放所有必需端口，例如8081、6379 和 3306，以支持前端访问数据库和缓存服务。



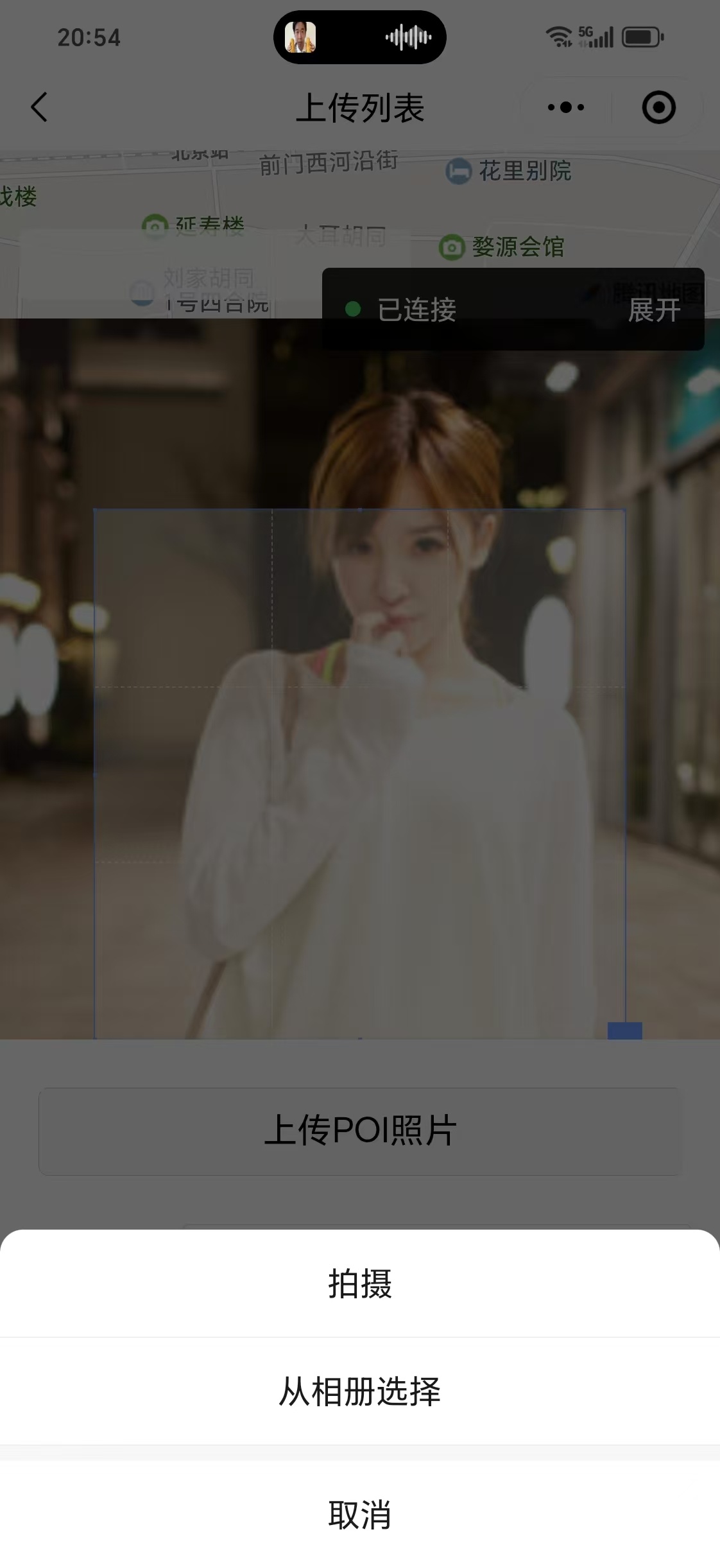
网络依赖：使用公网 IP 访问，小程序可通过微信服务器直接连接至腾讯云服务器提供的 API。

移动端兼容性：同样支持 iOS 和 Android 的小程序访问和使用。

### 1.2.4操作系统兼容性结论

基于 Vue 和 SpringBoot 的 UniApp 小程序框架，生成的微信小程序在微信开发者工具模拟器中成功测试，验证了对 iOS 和 Android 系统的兼容性。两种操作系统均可支持该小程序的运行。

**iOS 和 Android 均兼容：**基于微信小程序框架的 iOS 和 Android 环境兼容性得以保证。



小程序在移动端（Android）成功运行

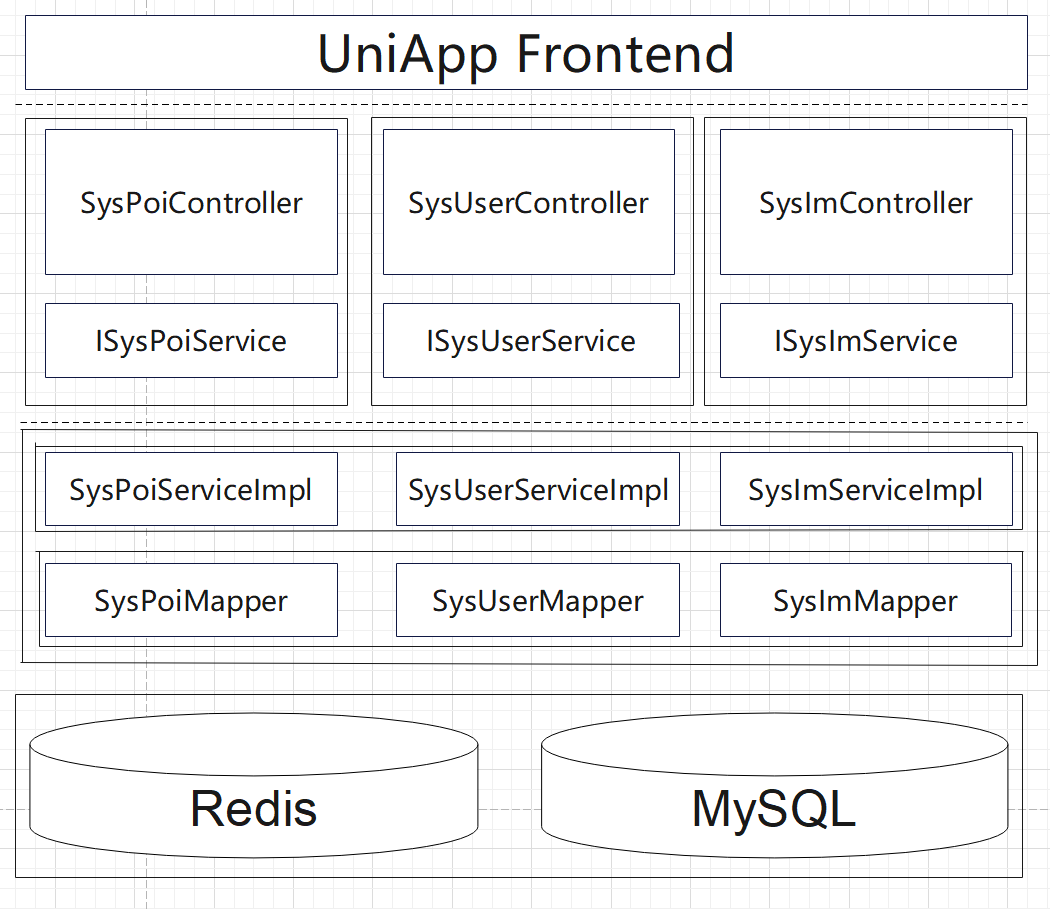
**开发者注意事项：**需针对 Android 设备优先测试和优化，因为绑定的微信 APP ID 机型为 Android。

本次环境限制分析得出，小程序支持在 iOS 和 Android 移动端设备上运行，并且通过本地服务器或腾讯云部署均可提供稳定的后台服务支持。

# *体系结构设计、体系结构风格分析与选择、设计模式应用。*

## 2.1体系结构：

### 2.1.1 N层体系结构架构图及其说明



该N层架构的各层功能划分如下：

**Interface层：**由`UniApp Frontend`组成，用于用户与系统的直接交互，负责前端展示和输入处理。用户通过这个层发送请求，系统将响应结果反馈给用户。

**Interface Control层：**包含`SysPoiController`等控制器类，作为请求的入口，将前端请求转发至业务层。控制器主要负责请求的路由和参数处理，不包含具体的业务逻辑，而是依赖业务接口进行逻辑处理。

**Business层：**包括`ISysPoiService`等业务接口声明及其实现类`SysPoiServiceImpl`等。接口定义了业务操作的方法，具体的实现类则负责实际的业务逻辑处理，例如数据校验、逻辑计算等。

**Data Access层：**由`SysPoiMapper`等Mapper类及对应的`SysPoiMapper.xml`文件组成，负责与数据库的交互。Mapper类包含数据库访问的方法接口，而Mapper XML文件定义了实际的SQL语句，实现了对MySQL数据库的CRUD操作。

**数据存储和管理层：**包括`MySQL`和`Redis`，其中MySQL作为主要数据存储，持久化存储数据；Redis作为缓存层，提升数据访问效率，减少数据库负载。

### 2.1.2 N层体系结构优缺点分析

N层架构是一种将应用程序划分为多个独立层次的架构模式，其主要优点和缺点如下：

**优点**

1. 职责分离：各层负责不同的功能，代码更加模块化、易于维护。UI层负责展示，业务逻辑层处理业务规则，数据访问层与数据库交互。

2. 增强可测试性：由于各层相对独立，可以更方便地进行单元测试和集成测试。例如，可以在不依赖数据库的情况下测试业务逻辑层。

3. 复用性高：数据访问层和业务逻辑层可以被不同的应用程序或不同的前端界面（如Web端、移动端）重用。

4. 增强可扩展性：每一层可以独立扩展或优化。例如，可以在不影响业务逻辑层的情况下更换数据访问层的数据库。

5. 便于团队协作：不同层的开发可以由不同的团队进行，接口定义清晰，有利于并行开发。

6. 支持分布式部署：在高并发场景中，可以将不同层级部署在不同的服务器上，提升系统的性能和可靠性。

**缺点**

1. 性能开销：由于层次之间存在较多的接口调用和数据传递，N层架构比简单的单层架构有更高的性能开销，尤其在层次多、接口复杂的情况下。

2. 开发成本高：N层架构的设计和实现相对复杂，需要更多的代码、接口和文档定义，因此会增加项目的开发时间和成本。

3. 增加系统复杂度：多个层次的划分使得系统的整体复杂度增加，新手开发者可能不容易掌握，项目管理和架构设计的要求较高。

4. 容易过度设计：在小型项目或简单的业务场景中，采用N层架构可能会显得繁琐和冗余，反而增加了不必要的开发工作。

5. 调试困难：由于各层相互依赖，调试可能需要跨层级进行，某些问题的定位和排查较为复杂。

**总结**

N层架构适用于复杂的大型系统或需要高扩展性和维护性的项目，而对于小型项目，N层架构的复杂性可能带来额外负担。

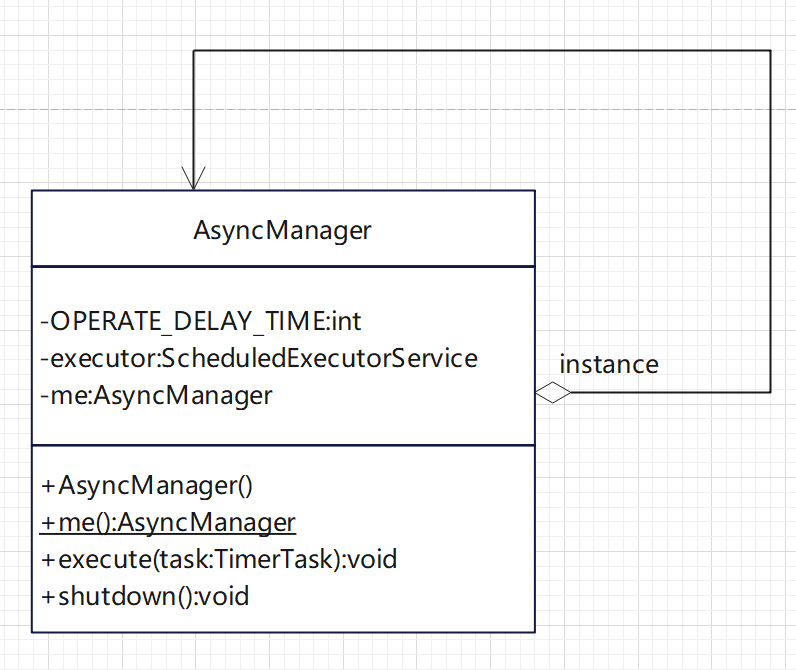
## 2.2设计模式：

作图规范：UML 2.0 作图平台：亿图图示 VIP版

### 2.2.1创建型

#### 2.2.1.1单例模式





在上述类图中，`AsyncManager`采用了单例模式，即确保该类在应用程序中只有一个实例。这个模式通过以下方式实现：首先，`AsyncManager`类有一个私有的构造函数`AsyncManager()`，禁止外部直接创建对象。然后，类内部通过一个私有的静态变量`me`来持有该类的唯一实例。`static me()`方法提供对该实例的访问，保证每次调用都返回相同的对象，从而实现单例特性。在类图中，通过一个聚合箭头指向`AsyncManager`自身，表示类通过静态方法`me()`来获取其唯一实例。这种设计使得类的实例化与外部代码解耦，确保了系统中只有一个`AsyncManager`实例，避免了资源浪费和线程安全问题。

#### 2.2.1.2建造者模式

‌ 建造者模式(Builder Pattern)‌又称生成器模式是一种创建型对象设计模式，它允许将一个复杂对象的构建过程与其表示分离，使得同样的构建过程可以创建不同的表示。



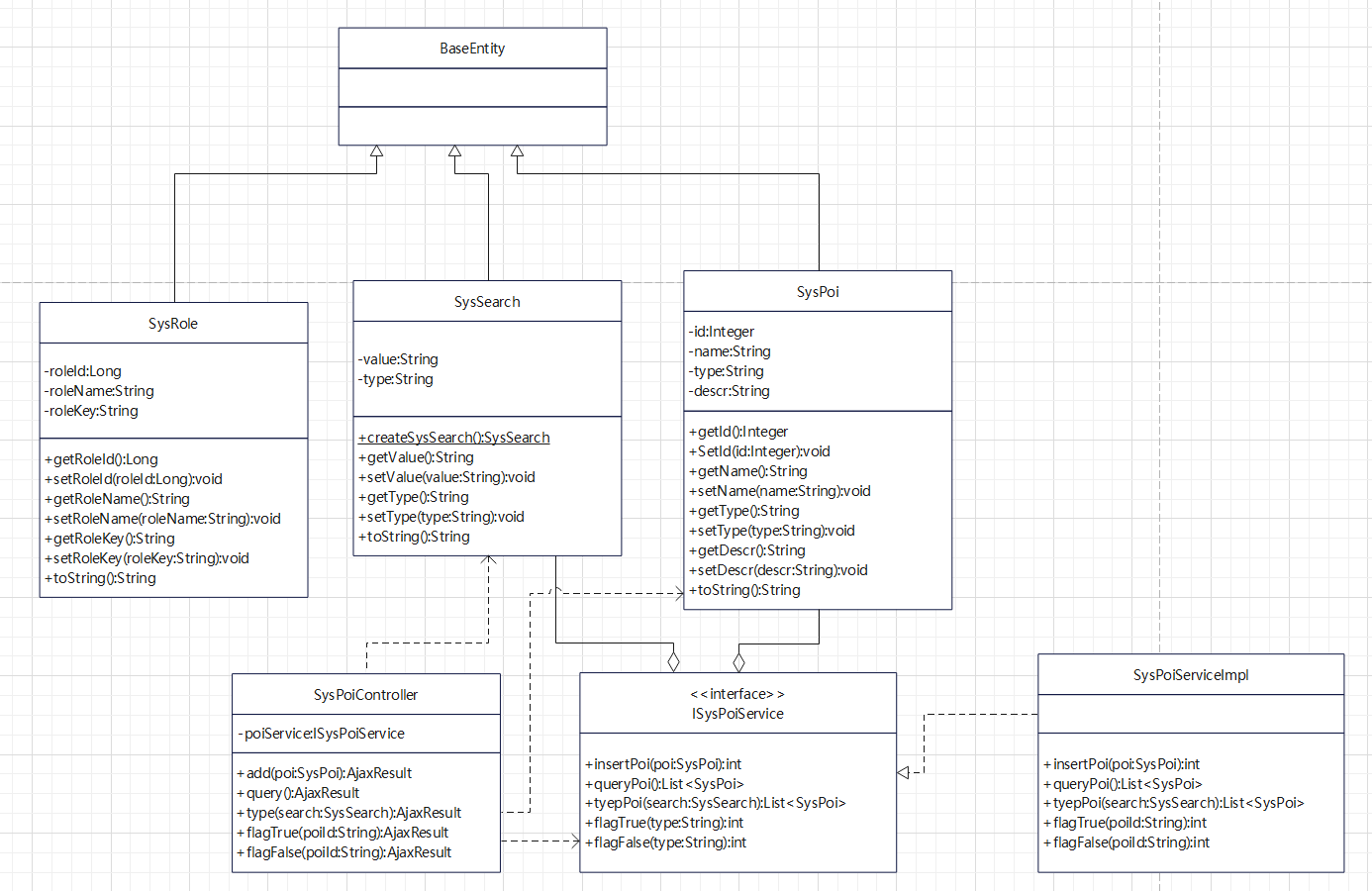
在这个代码片段中：

**逐步构建对象的字符串表示**：SysPoi 类的 toString() 方法利用 ToStringBuilder 类，以链式调用的方式逐步添加对象的各个属性值（例如id、name、type等）。每个 .append() 方法调用都是在构建对象的不同部分，最终生成完整的字符串表示。

**清晰的构建流程**：ToStringBuilder 类允许将属性值逐步添加到字符串表示中，避免了手动拼接字符串带来的易读性差和易错性。每个 .append() 方法都清楚地显示了属性名和属性值的匹配关系，使代码更易读。

**与原始对象解耦**：ToStringBuilder 的使用使得 toString() 方法的构建过程与 SysPoi 类本身解耦。若未来需要改变字符串的格式，开发者可以调整 ToStringBuilder 的设置，而不必修改 SysPoi 类的实现逻辑。

因此，这段代码体现了建造者模式的核心特性：**通过链式方法逐步构建**，**使构建过程简洁清晰**，**并与最终构建的字符串表示解耦**。



在这个类图中，BaseEntity作为Builder，定义了各个基础实体的通用属性和方法。它的三个子类 (SysRole、SysSearch和SysPoi) 是ConcreteBuilder，各自扩展了 BaseEntity 的属性和方法，用于构建不同类型的 POI 对象及其相关数据结构。Director是SysPoiController和ISysPoiService的结合体，其中 Controller 发起构建请求，Service 执行具体的构建逻辑，生成 POI 对象或执行相关的业务操作。这种设计通过建造者模式使数据的构建逻辑更具灵活性和可扩展性，同时将控制与构建实现相分离。

#### 2.2.1.3工厂模式

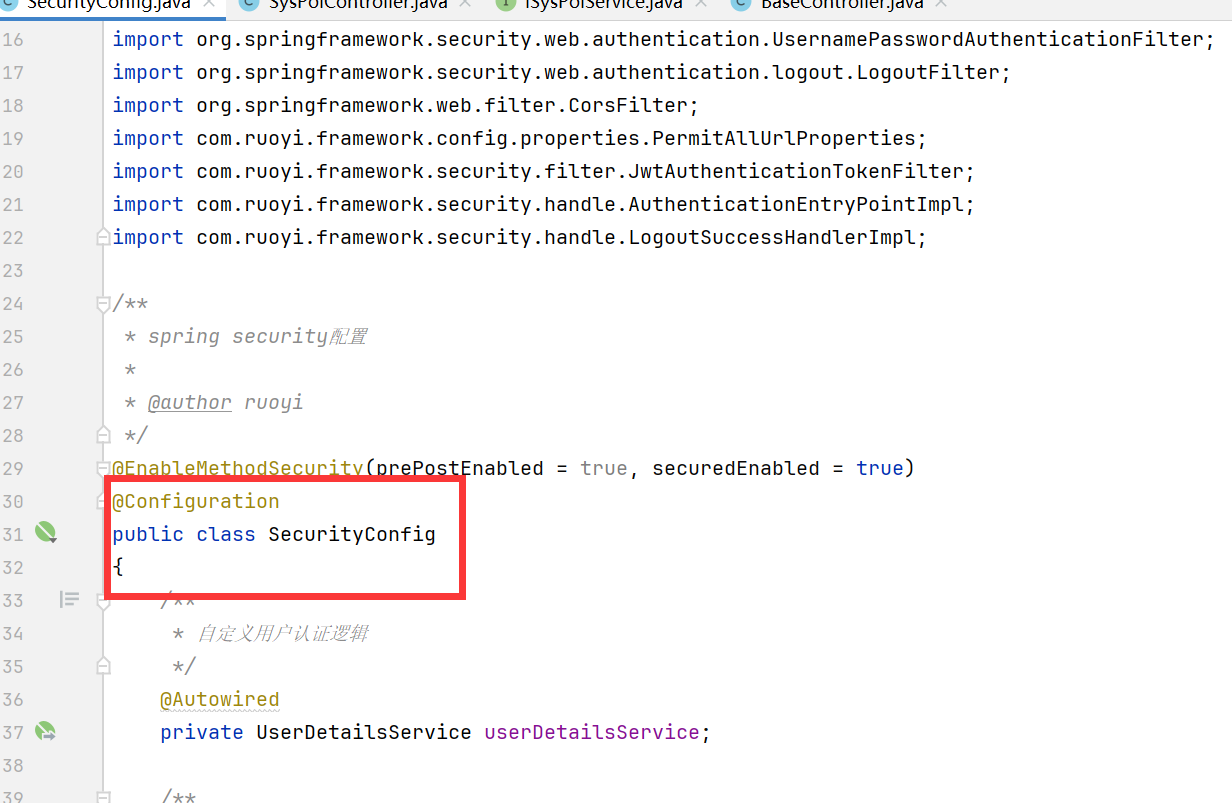
工厂模式是一种创建型设计模式，它通过定义一个创建对象的接口，允许子类决定实例化哪个类，从而将对象的创建和使用分离，提高代码的模块化和可维护性。

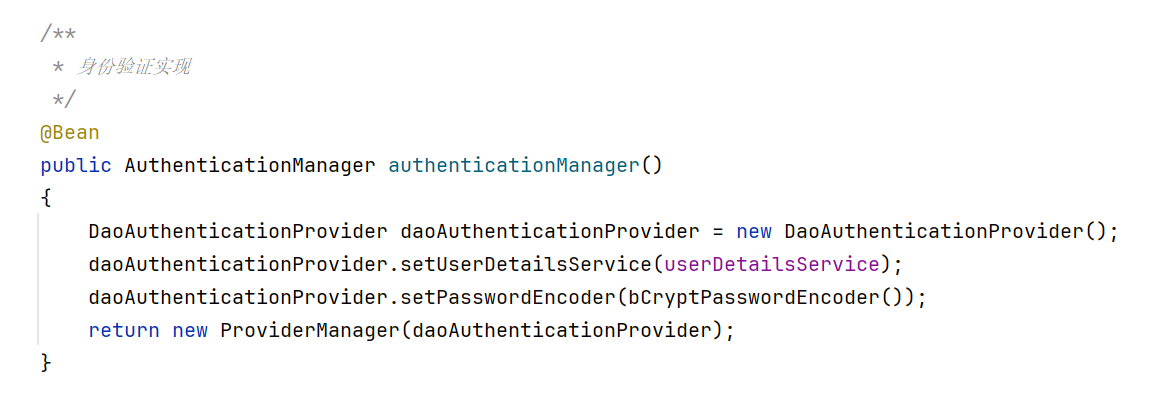
SpringBoot中工厂模式的实现通常涉及到以下几个方面：

1.使用@Configuration注解定义配置类：这个类包含了配置信息，可以定义bean的创建逻辑。

2.使用@Bean注解定义方法：这些方法会返回一个bean实例，Spring容器会调用这些方法来创建和管理bean。

1. 使用@Autowired注解注入依赖：如果工厂方法需要依赖其他bean，可以使用@Autowired进行自动装配。



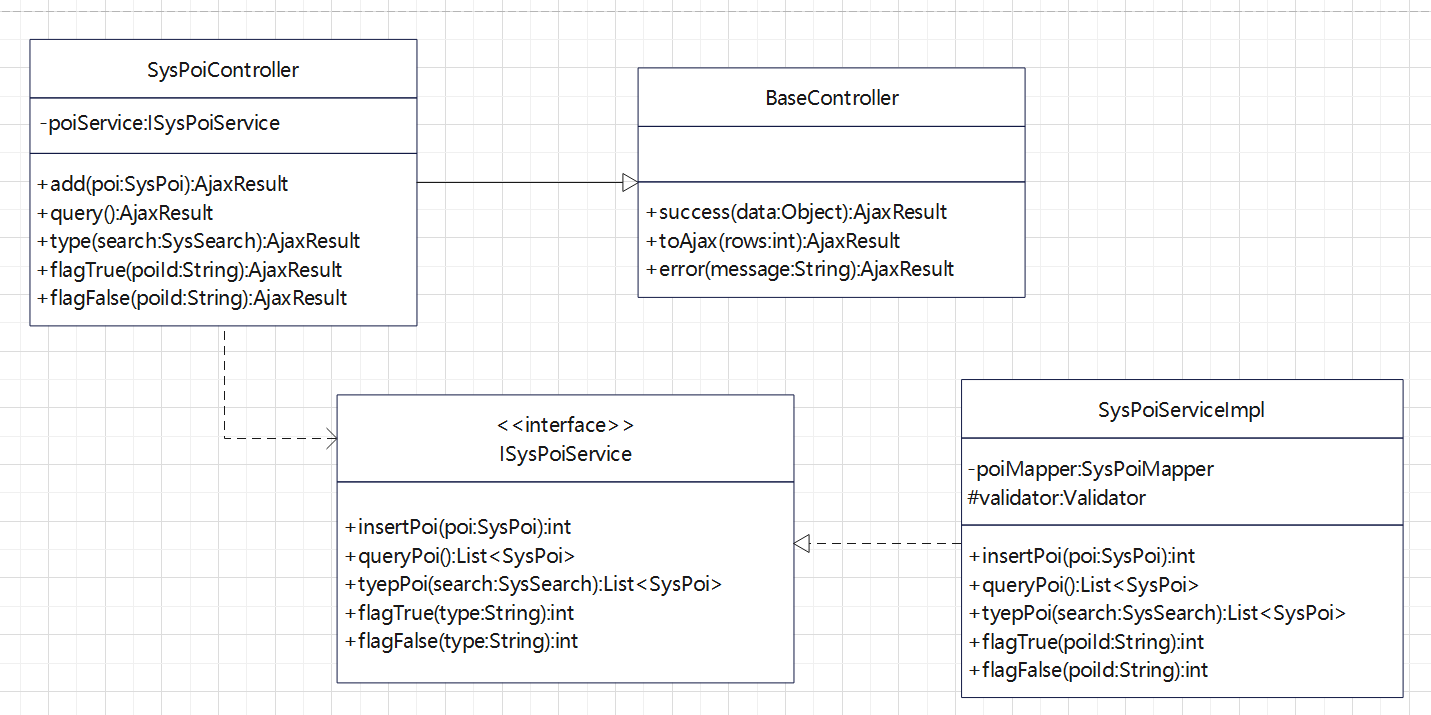


这里创建了一个 AuthenticationManager 实例，具体过程如下：

* DaoAuthenticationProvider 是一个认证提供者，它用于实现基于数据库的认证。在这段代码中，它被创建并配置了 UserDetailsService 和 PasswordEncoder（这两个对象的创建也涉及工厂模式）。
* 然后，将 DaoAuthenticationProvider 传入到 ProviderManager 的构造函数中，生成最终的 AuthenticationManager 对象。

这部分代码的工厂模式表现在通过 ProviderManager 构造器创建 AuthenticationManager 实例时，隐藏了创建过程中的具体细节，并提供了一个工厂方法（authenticationManager()）来返回已配置好的对象。





上图展示了基于接口的工厂模式，工厂模式的实现主要体现在使用 @Autowired 注解注入的 ISysPoiService 服务接口。

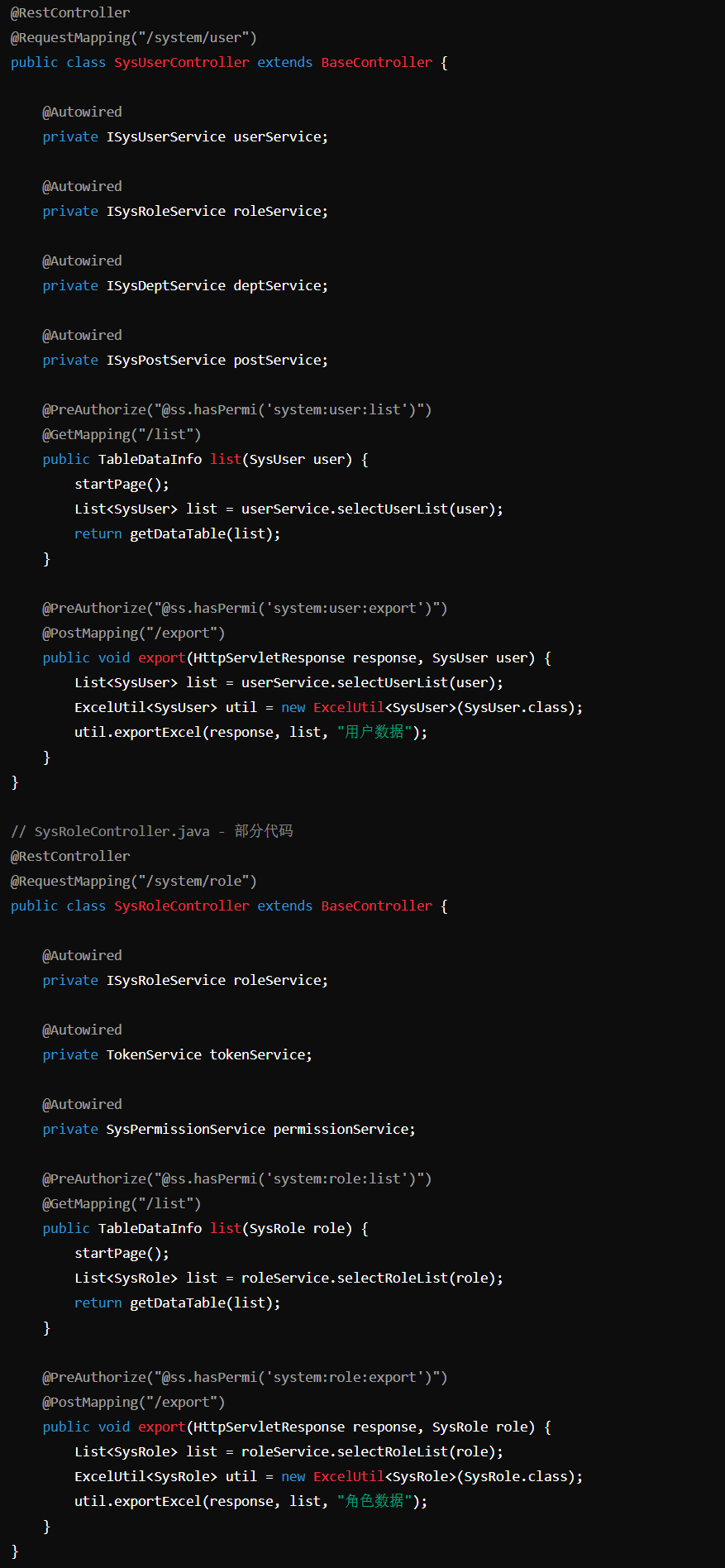
**@Autowired 注解**：ISysPoiService 作为一个接口，被 @Autowired 注解自动注入 SysPoiController 类中。Spring 框架通过其内部的工厂方法，根据依赖注入的配置提供 ISysPoiService 的具体实现对象（如 SysPoiServiceImpl）。

**解耦性**：通过接口 ISysPoiService 来定义 POI 相关服务的操作，使 SysPoiController 与具体实现类解耦。这种解耦性符合工厂模式的思想，由工厂（Spring容器）决定具体实现，以便轻松替换或扩展服务实现。

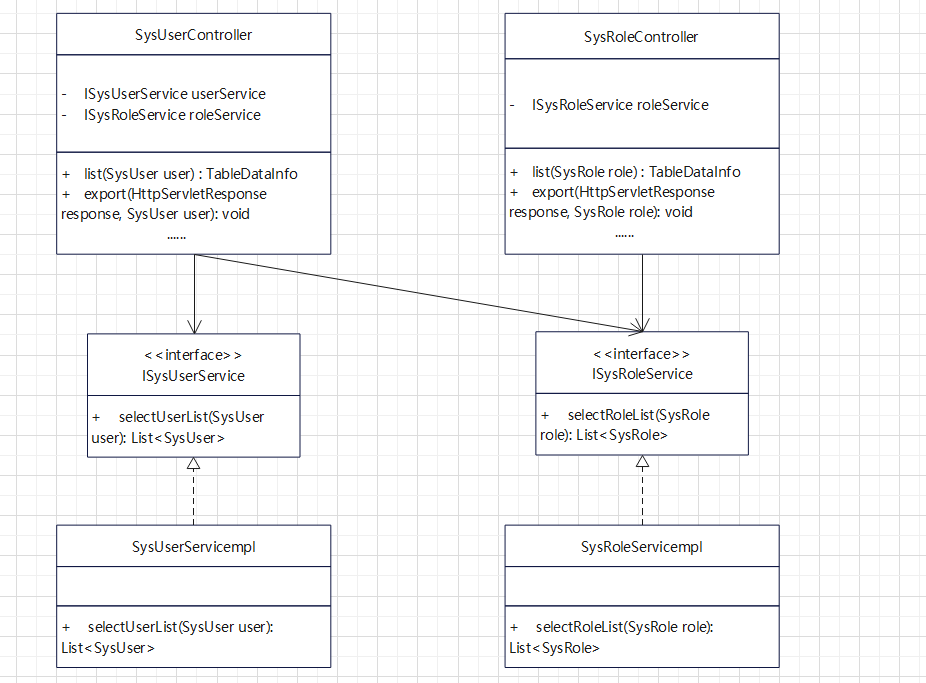
**接口工厂模式的好处**：接口工厂模式使得代码的耦合性降低，SysPoiController 不需要知道 ISysPoiService 的具体实现类是哪个，方便更换实现或进行单元测试。

### 2.2.2结构型

#### 2.2.2.1外观模式



SysUserController和SysRoleController通过注入多个服务类（如 ISysUserService 和 ISysRoleService），将复杂的用户和角色业务逻辑封装在各自的控制器方法中。前端调用这些控制器的 API 时，不需要了解具体的业务实现逻辑，控制器作为外观接口简化了业务调用的复杂性。



(草图，之后修改sysUserService->sysUserServicempl ?)

在上面的类图中可以看出：

**SysUserController 和 SysRoleController：**作为外观类，它们封装了用户和角色 的管理逻辑，通过调用 ISysUserService 和 ISysRoleService 的方法来处理业务， 隐藏了具体的业务实现细节，前端只需与控制器交互。

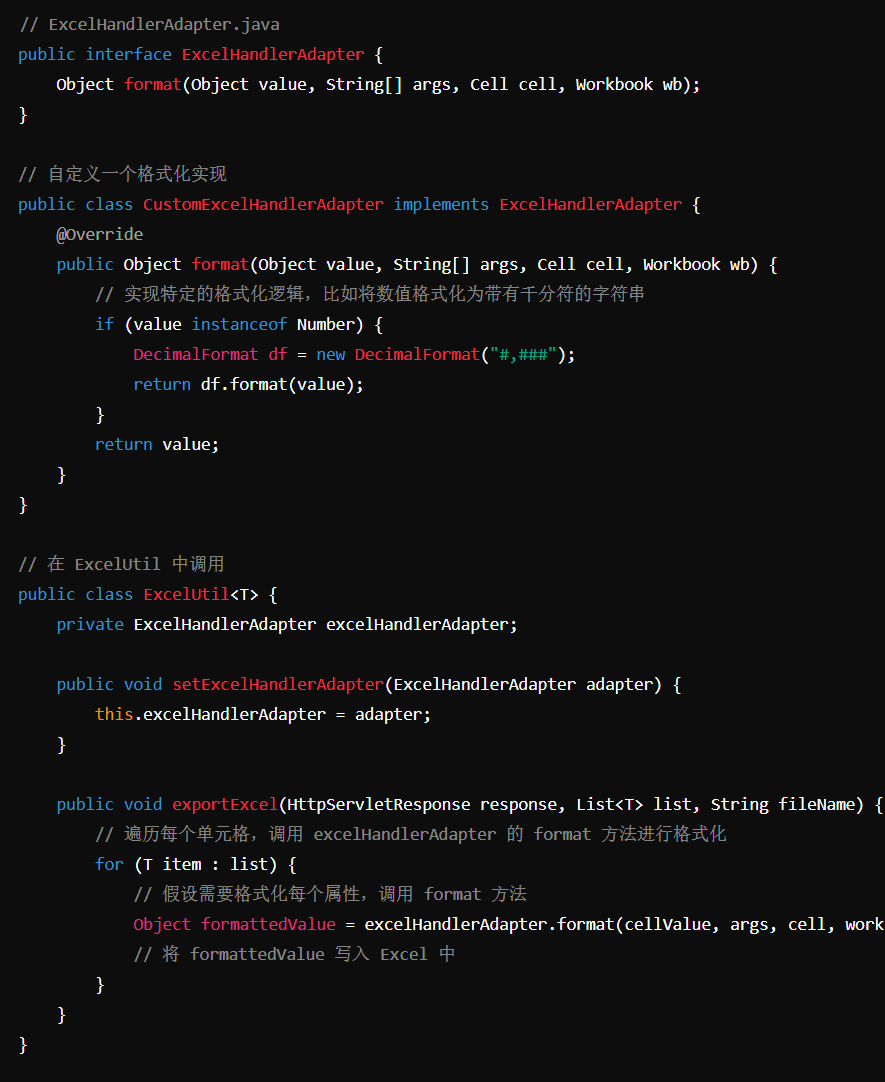
**ISysUserService 和 ISysRoleService：**作为服务接口，定义了用户和角色业务 的核心操作。

**SysUserServiceImpl 和 SysRoleServiceImpl：**具体实现类，提供用户和角色的 业务逻辑。

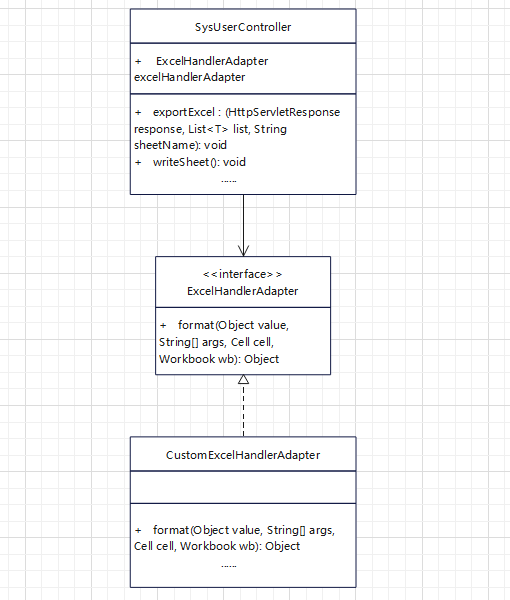
**调用关系：**SysUserController 调用了 ISysUserService 和 ISysRoleService，而 SysRoleController 仅调用了 ISysRoleService，控制器与具体业务实现隔离，符合 外观模式的设计原则。

#### 2.2.2.2适配器模式

在本项目中，适配器模式的使用主要体现在 ExcelHandlerAdapter 接口的设计。它为 Excel 数据的格式化和处理提供了一个统一的接口，以便不同的格式化需求可以适配到 ExcelUtil 中进行处理，以下为示例代码：



代码中ExcelHadnlerAdapter作为适配器模式中的 **Target**，而 CustomExcelHandlerAdapter 是其具体的适配器实现（**Adapter**）。通过实现 format 方法，CustomExcelHandlerAdapter 可以将不同的格式化逻辑适配到 ExcelUtil 中进行处理。



由这个类图可知道：

1. **ExcelHandlerAdapter**：定义了 format 方法，作为适配器模式中的目标接口（Target）。
2. **CustomExcelHandlerAdapter**：具体的适配器实现类，将特定的格式化逻辑封装起来，以适应 ExcelUtil 的格式化需求。
3. **ExcelUtil**：通过适配器模式，ExcelUtil 可以适配不同的格式化实现，从而灵活处理 Excel 导出中的数据格式化需求。

这种设计在不修改 ExcelUtil 的前提下，可以实现多种不同的格式化逻辑，体现了适配器模式的灵活性。

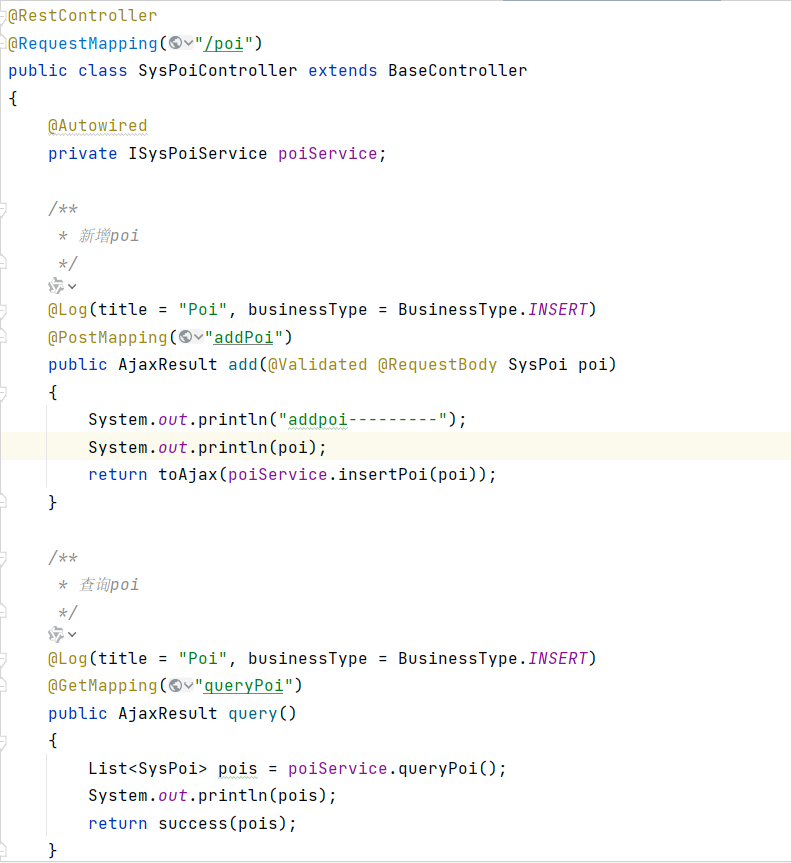
### 2.2.3行为型

#### 2.2.3.1模板方法模式

模板方法模式的核心思想是定义一个操作的骨架，而将一些具体的步骤延迟到子类中去实现。在这个项目结构中，BaseController 定义了一些通用的操作模板，比如返回成功或失败的结果、分页处理、数据表处理等。其他控制器（如 SysPoiController、SysUserController 等）可以直接继承 BaseController，从而复用这些基础方法，而无需重复定义相同的代码逻辑。

在 BaseController 中，定义了一些通用的模板方法，便于子类使用。例如：

1. success() 和 error() 方法：提供了统一的响应结果构建方法。子类控制器可以直接调用 success()、error()、warn() 等方法来简化代码逻辑。
2. toAjax() 方法：统一了返回成功或失败的判断逻辑，子类可以调用 toAjax(int rows) 和 toAjax(boolean result) 来返回 Ajax 格式的结果。
3. 分页和排序模板方法：startPage()、startOrderBy() 和 clearPage() 方法提供了分页和排序的基本处理逻辑，子类控制器在需要分页的查询方法中可以直接调用这些方法。

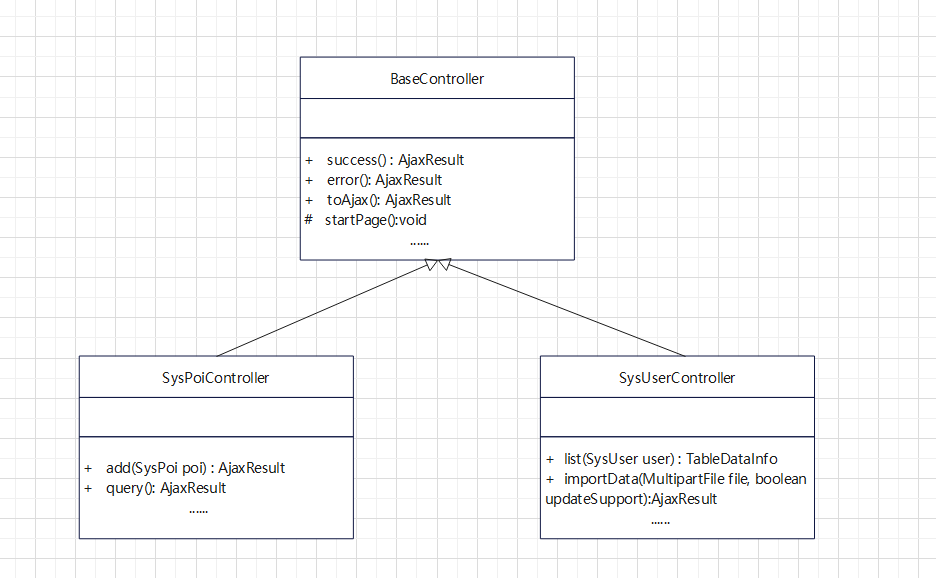


而在 SysPoiController 中，就有一些方法直接使用了 BaseController 中的模板方法

在上面的代码片段中：

1. add 方法调用了 toAjax(poiService.insertPoi(poi))。toAjax() 是 BaseController 中的模板方法，直接返回标准化的 AjaxResult 结果。
2. query 方法调用了 success(pois)，这也是 BaseController 中的模板方法，统一构建成功响应的返回格式。

通过继承 BaseController，SysPoiController 复用了响应结果处理的逻辑，简化了代码的实现，体现了模板方法模式的优势。

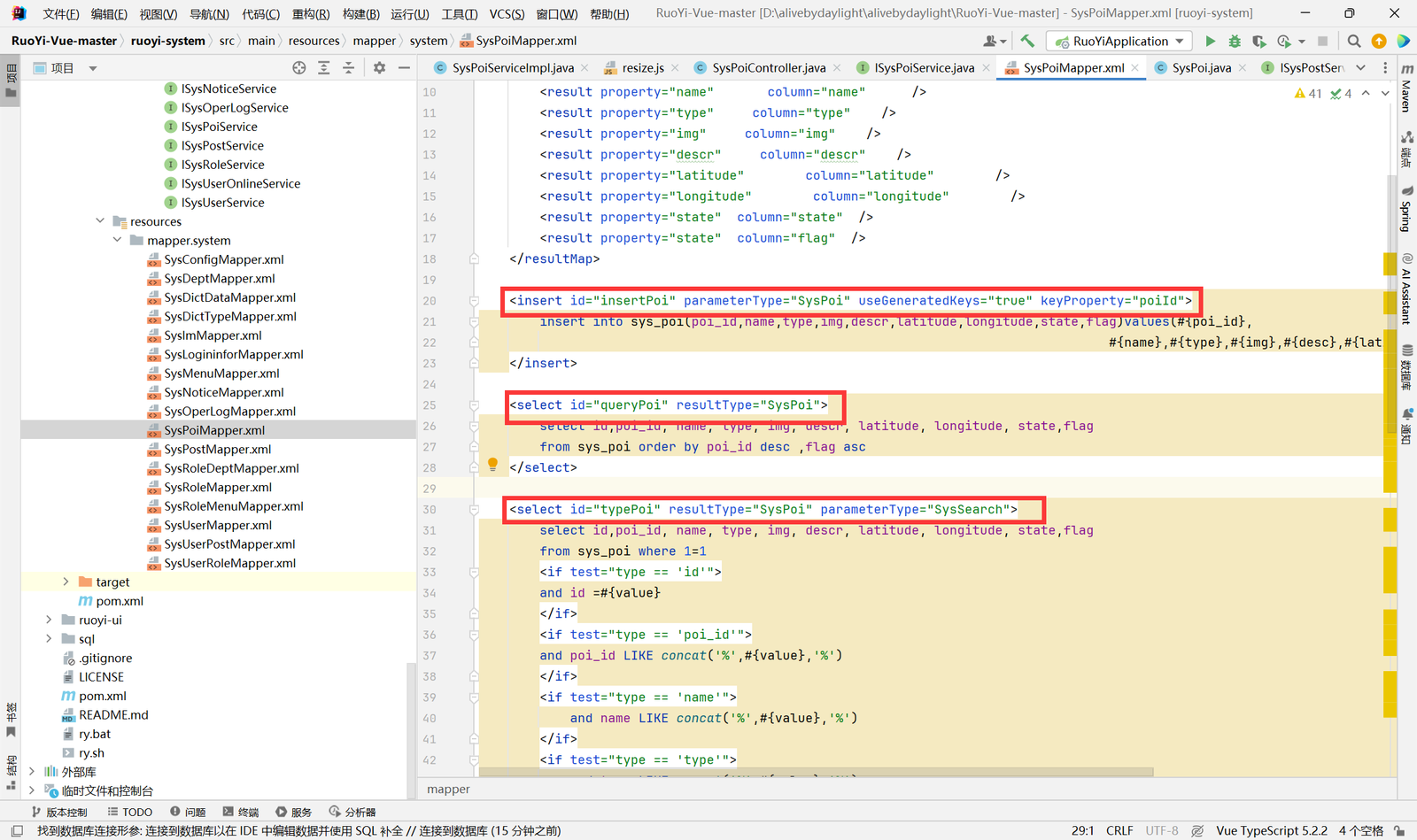


在上述UML类图中

**BaseController：**定义了模板方法 success()、error()、toAjax() 等，以及分页和排 序的基础方法 startPage()、startOrderBy() 等。子类控制器可以通过继承 BaseController 来使用这些方法，复用其通用逻辑。

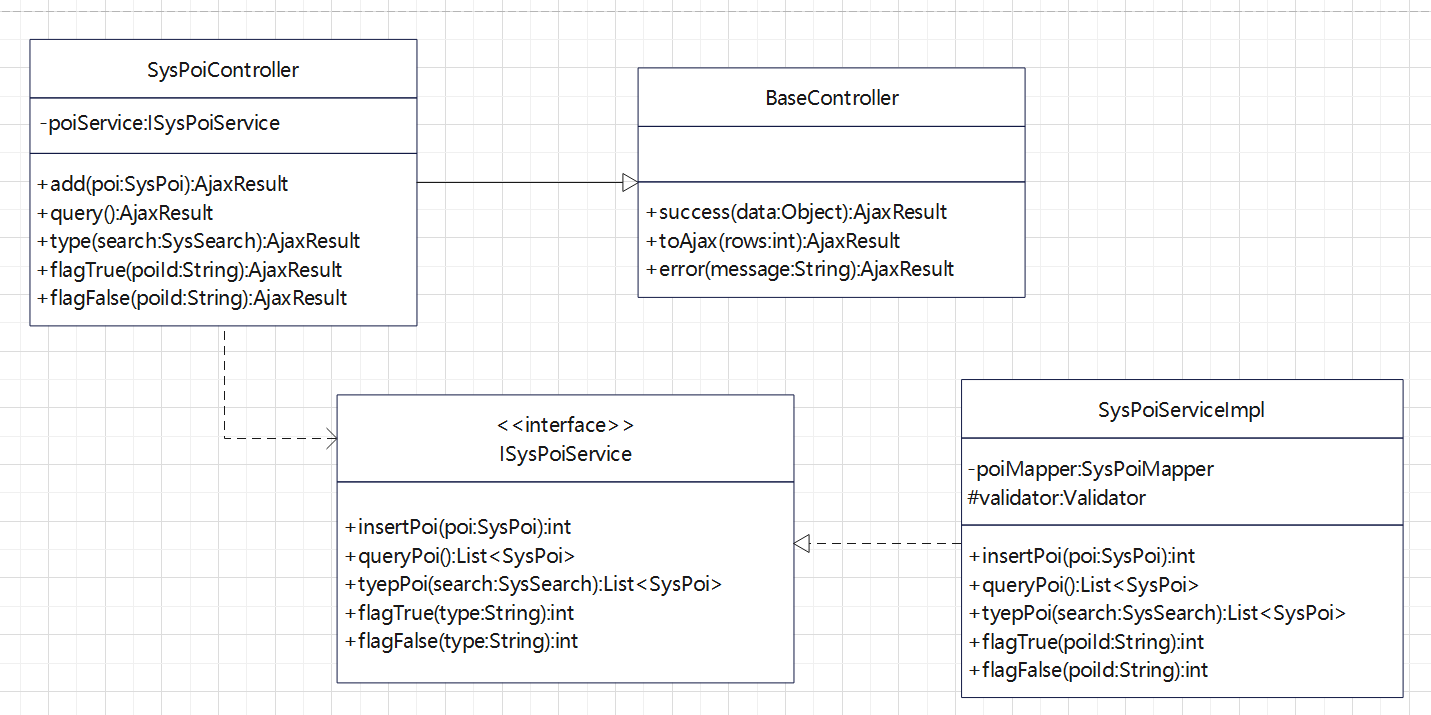
**SysPoiController：**继承了 BaseController，并在 add() 和 query() 方法中使用了 BaseController 的模板方法，实现了业务逻辑的同时保持了代码简洁和一致性。

#### 2.2.3.2策略模式



策略模式允许在运行时动态地选择算法的实现。在 SysPoiMapper.xml 中的 typePoi 查询方法，通过 <if> 标签检查不同的条件，如 type == 'id'、type == 'poi\_id' 等，根据不同的条件动态生成 SQL 查询语句。这种策略模式的应用体现了根据不同的输入（如 type）来选择不同的查询条件，从而让代码更加灵活和可扩展。

#### 2.2.3.3命令模式



命令模式将请求（方法调用）封装成一个对象，使得用户可以通过执行该对象来执行请求，从而解耦请求的发送者和接收者。在本例中，方法 insertPoi、queryPoi 等实际上代表了不同的命令，每个方法对应一个操作（插入、查询等）。通过在业务层中调用相应的命令方法来执行实际的操作。

具体体现：

* SysPoiServiceImpl 中的各个方法（如 insertPoi、queryPoi）其实可以看作是一个个独立的命令，这些方法在调用时执行不同的数据库操作。
* SysPoiMapper 中的每个 insert、select、update 方法也分别对应着数据库操作命令。

# 应用软件开发实现。

**后端代码开发概述：**

ruoyi-admin/src/main/java/com/ruoyi/web/controller/poi/SysPoiController.java:关于

poi增查的终端反馈逻辑；

ruoyi-admin:其余部分为表单、登录、人员、缓存监控相关的终端反馈逻辑；

ruoyi-common/src/main/java/com/ruoyi/common/core/domain/entity：关于POI数据、人员、菜单的实体类，其遵循构造者模式，后续ruoyi-system中声明的接口调用此实体类；

ruoyi-common其余部分：常量与工具类声明；

ruoyi-framework:线程资源、安全验证、拦截器处理逻辑

ruoyi-quartz:调度日志相关逻辑

ruoyi-system/src/main/java/com/ruoyi/system/mapper:poi增删改查、人员增删改查等接口集合

ruoyi-system/src/main/resources/mapper/system:poi增删改查、人员增删改查等接口映射的“SQL”实现集合（.xml）

**前端代码开发概述：**

pages/index.vue:首页界面

pages/login.vue:登录界面

pages/work/index.vue:工作台界面

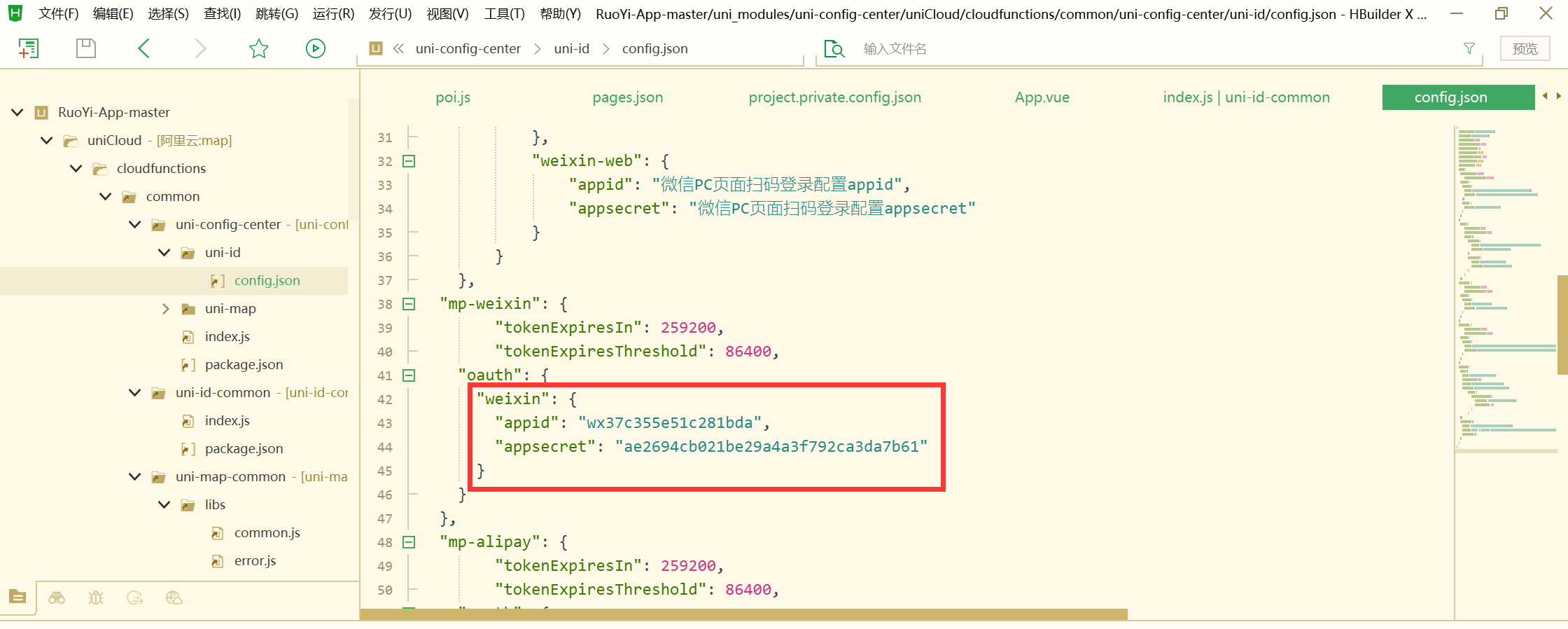
pages/route-plan:路径规划模块逻辑（腾讯地图API）

ruoyi-ui:系统后台（控制人员注册）对应的前端，需在VS Code等IDE中运行，其对应的后端与采集者、检验者一致

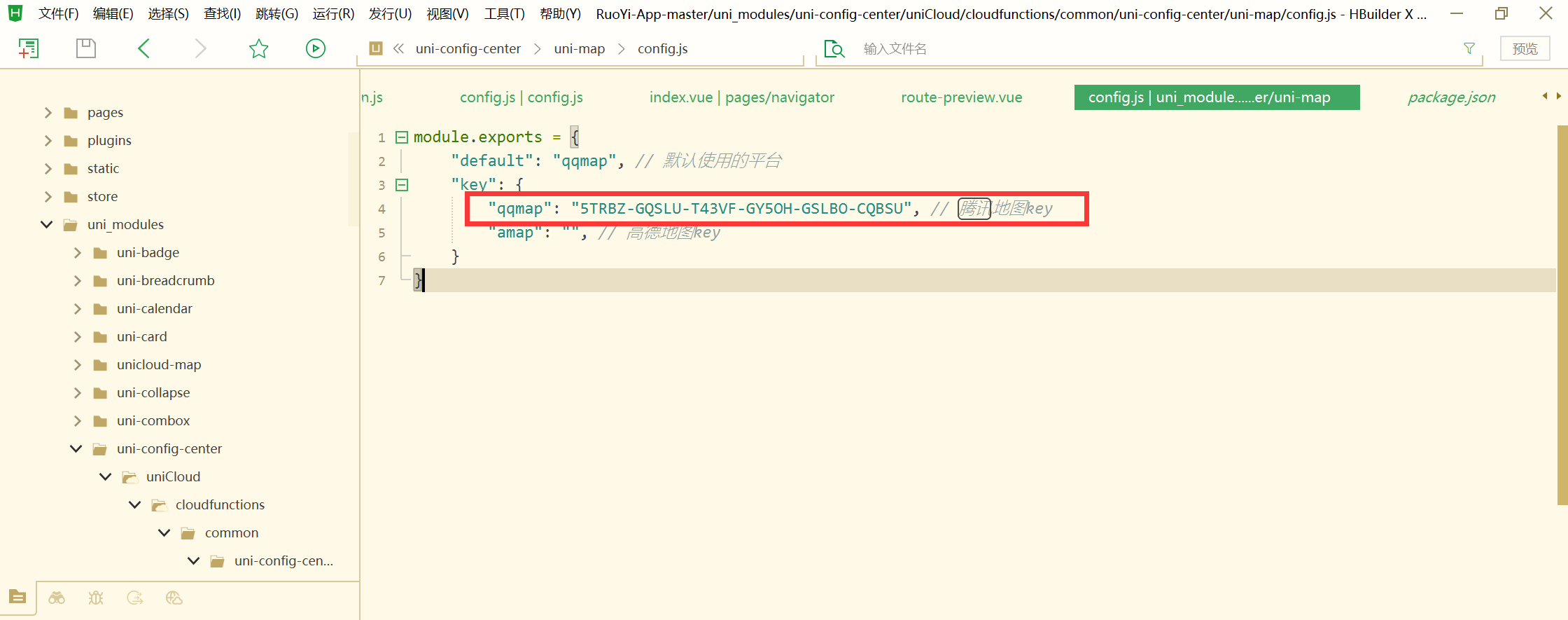
路径规划模块依赖下载的组件：腾讯地图组件



需下载的unicloud-map云端一体地图组件



地图组件配置App ID



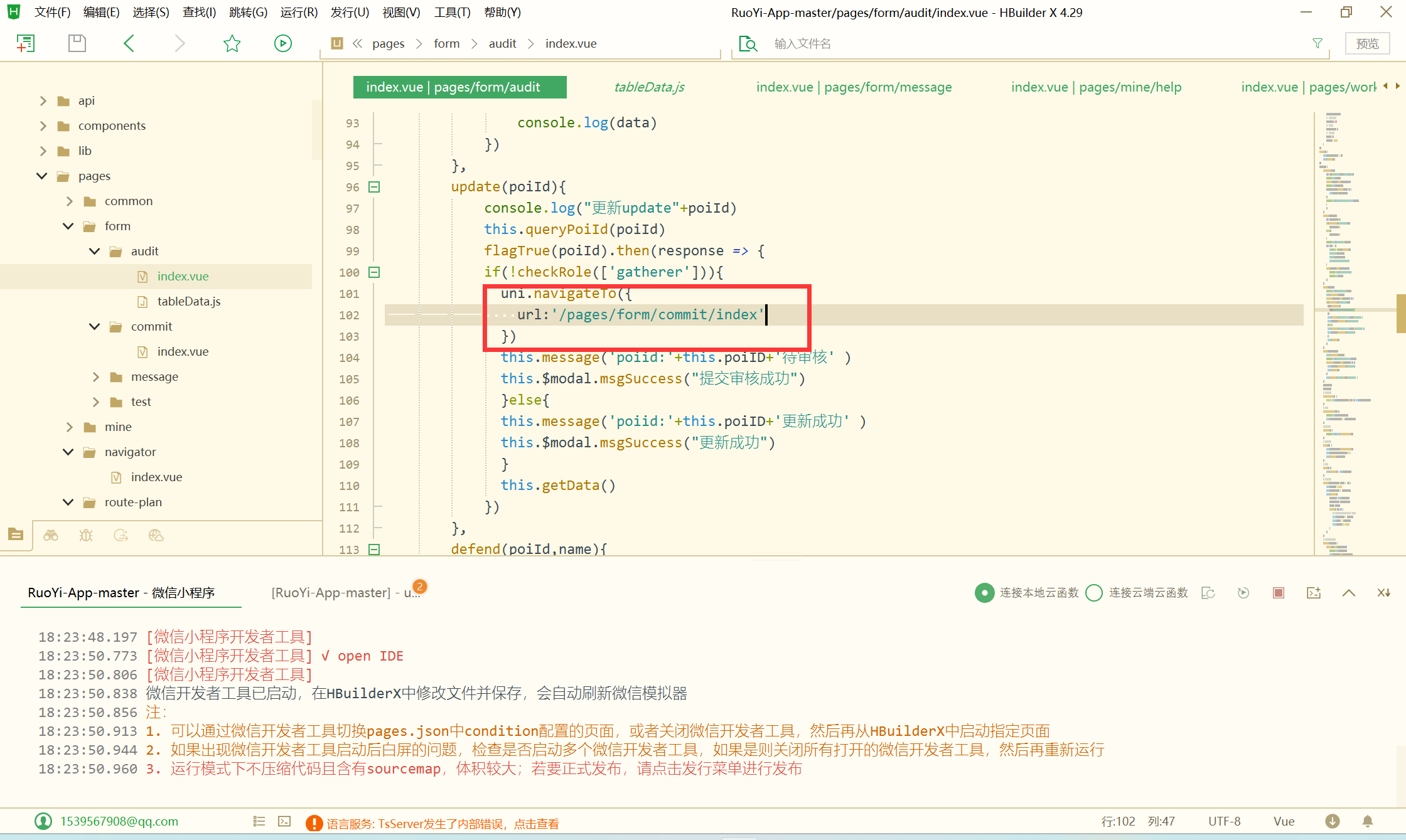
地图组件配置腾讯地图key

**uniCloud、uni\_modules:小程序中调用腾讯地图的必备组件，如若去除，腾讯地图将不能正常使用，而其他模块功能不受影响。**

pages/mine:"我的"模块

pages/form/audit/index.vue:检验者更新、采集者驳回和更新功能模块，包括数据列表和数据操作逻辑

pages/form/commit/index.vue:采集者上传数据功能模块



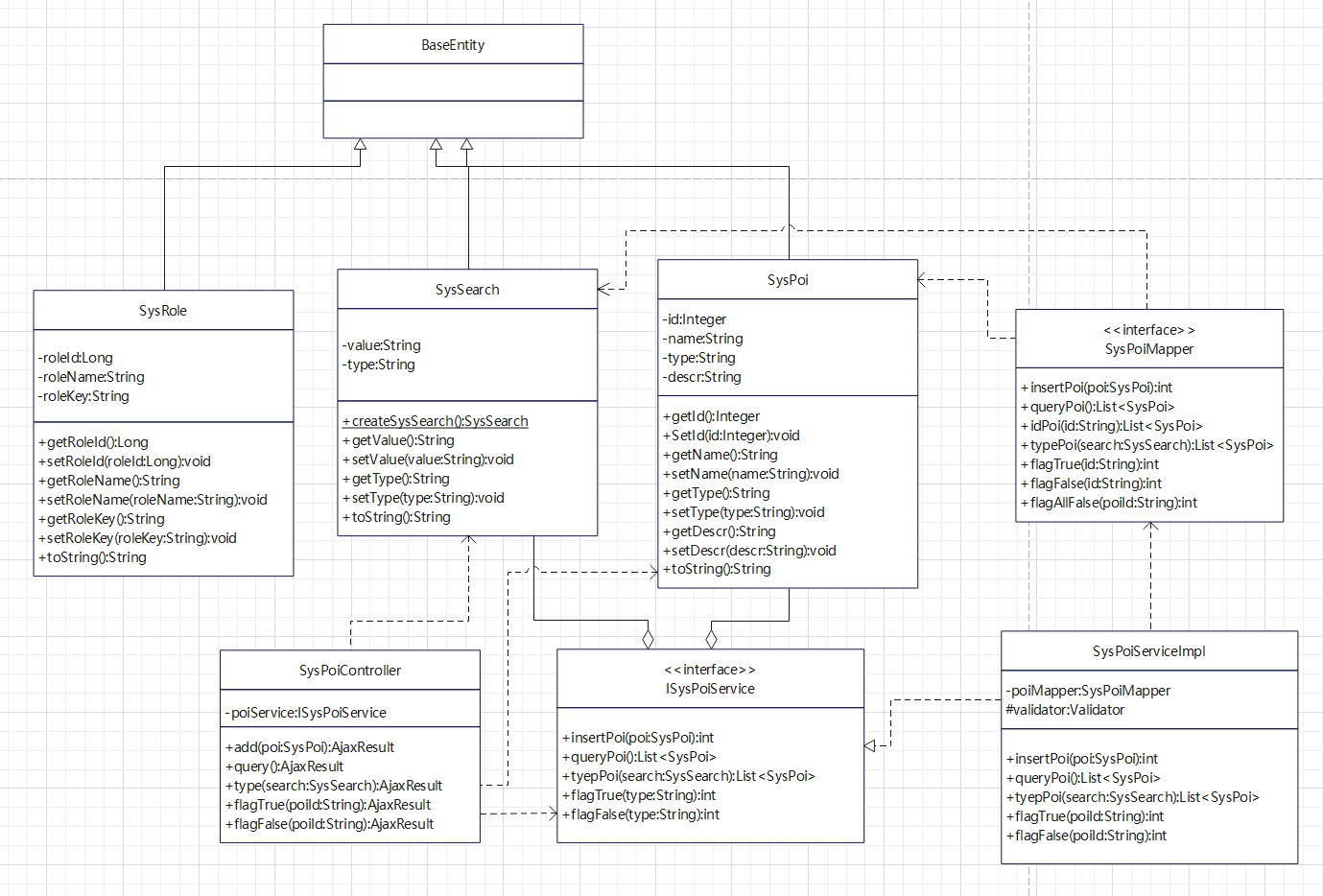
设置检验者更新数据的跳转逻辑

pages/form/message/index.vue:采集者、检验者、管理员对话讨论功能模块

## 3.1逻辑视图

逻辑视图主要关注系统提供给最终用户的功能，设计的对象模型（在使用面向对象的设计方法时）。在UML中，逻辑视图通常用类图来表示，展示系统的静态结构，包括类、接口、关联、泛化、依赖关系等。

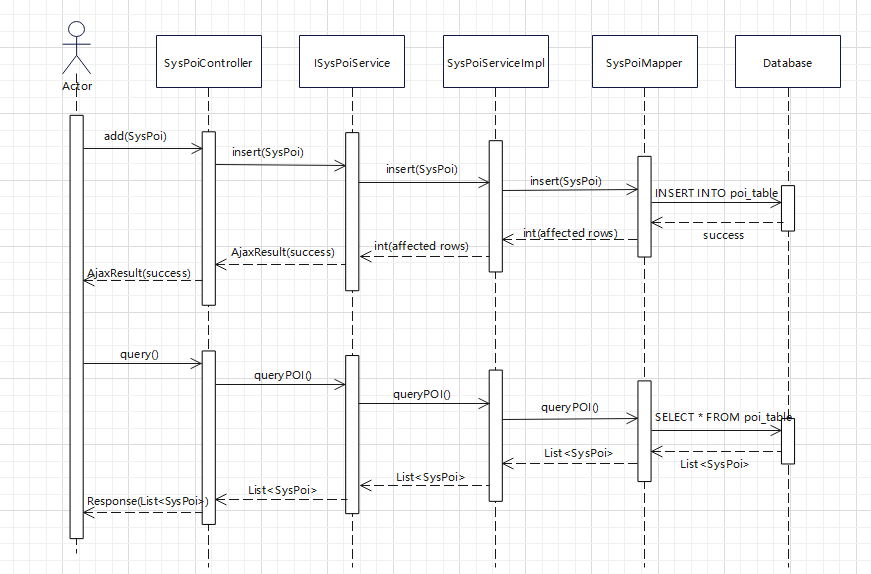
后端关键代码聚焦于“POI增加、查询”模块，此模块对应类图如下：



在 SpringBoot 应用中，`ISysPoiService` 与 `SysPoi` 和 `SysSearch` 的关系更接近于聚合关系，因为 `ISysPoiService` 作为业务层接口，处理业务逻辑并频繁操作这些对象，将它们作为核心组成部分。而 `SysPoiController` 与 `SysPoi` 和 `SysSearch` 的关系更偏向于依赖关系，因为它通过依赖注入调用 `ISysPoiService` 来处理用户请求，间接使用这些对象，这种关系通常是短暂的，并不涉及长期的组成。与此同时，`SysPoiServiceImpl` 依赖于 `SysPoiMapper` 来执行数据库操作，这也是一种依赖关系，专注于将数据访问逻辑与业务逻辑分离。

## 3.2进程视图

进程视图关注系统动态运行时，主要是进程以及相关的并发、同步、通信等问题。在UML中，进程视图可以通过状态图、活动图、时序图和协作图来表示，展示系统的动态行为和对象间的交互。在本项目中，以“POI增加与查询”功能为例，绘制其时序图并进行分析，展示用户请求从发起到数据库存储和响应的交互过程。

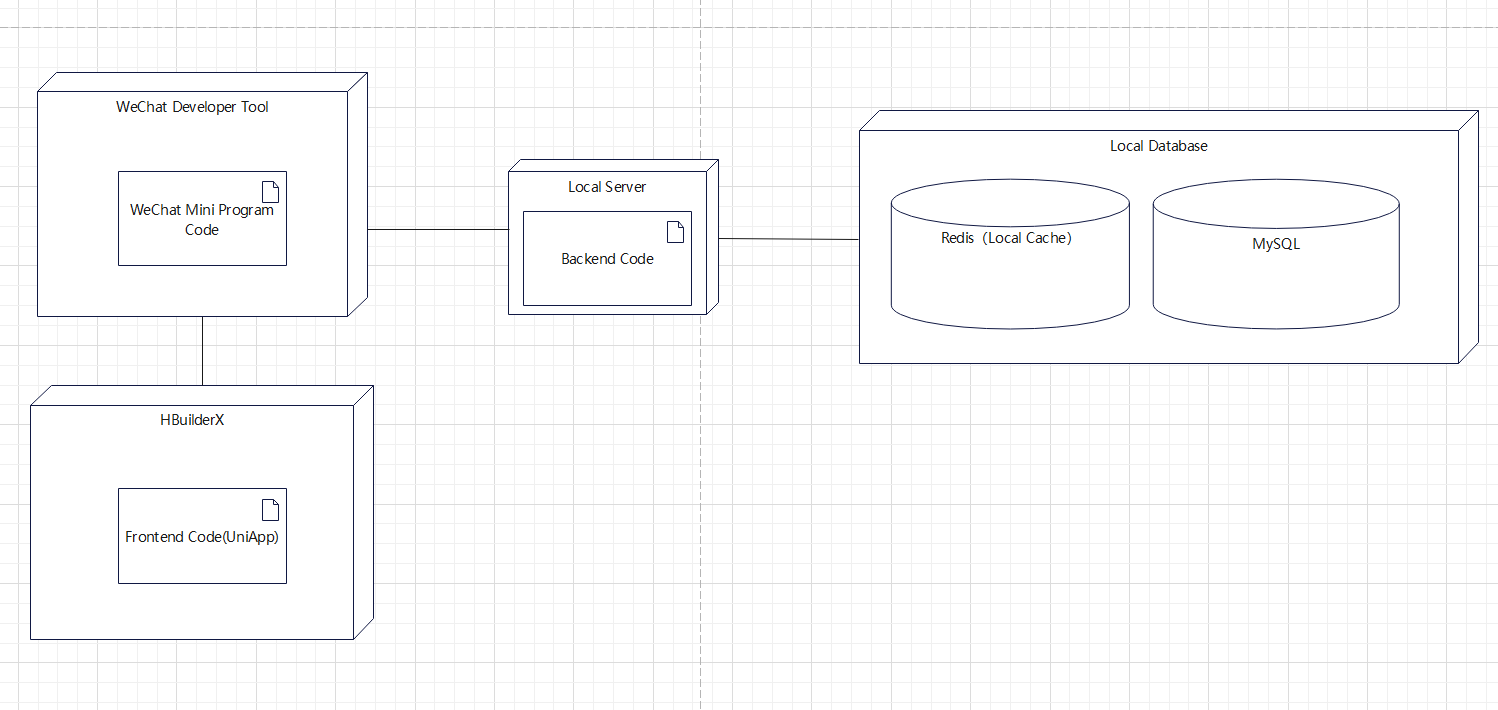


动态行为上，POI增加功能的实现是用户通过SysPoiController发起请求，控制器调用服务层接口ISysPoiService，并由其实现类SysPoiServiceImpl将数据保存至数据库。数据层SysPoiMapper负责生成SQL语句并执行插入操作，最终返回执行结果并向用户反馈操作是否成功。在POI查询功能中，用户通过SysPoiController发起查询请求，控制器调用服务层接口，通过SysPoiServiceImpl和SysPoiMapper从数据库中获取POI数据，并将结果返回给用户。这些行为体现了系统分层架构的动态调用逻辑，各组件之间职责清晰，交互有序。

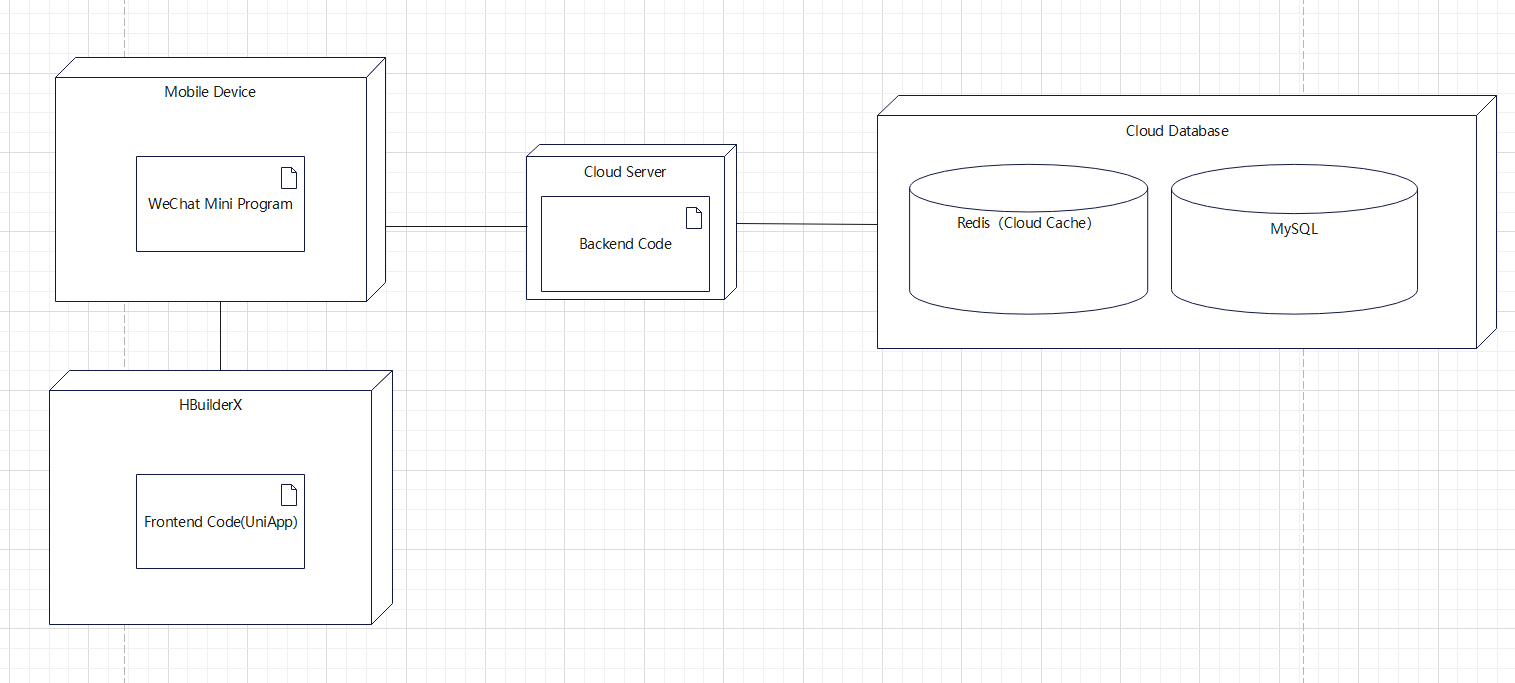
在并发与同步方面，模块采用典型的同步调用模式，用户请求以线程的形式处理，每一层都等待下游完成后再返回结果，数据库操作则通过事务管理确保数据一致性。通信机制上，各层之间通过Spring框架的依赖注入实现调用，避免直接依赖，同时控制器通过AjaxResult统一封装响应结果，确保前后端通信的一致性。整体设计基于分层架构，控制层、服务层、数据层职责分明，事务管理保障数据操作的完整性，数据流清晰直观，易于维护和扩展。

## 3.3物理视图

物理视图定义软件到硬件的映射，反映架构的分布式特性。在UML中，物理视图通常由部署图来表示，展示软件组件在物理节点上的分布和通信。

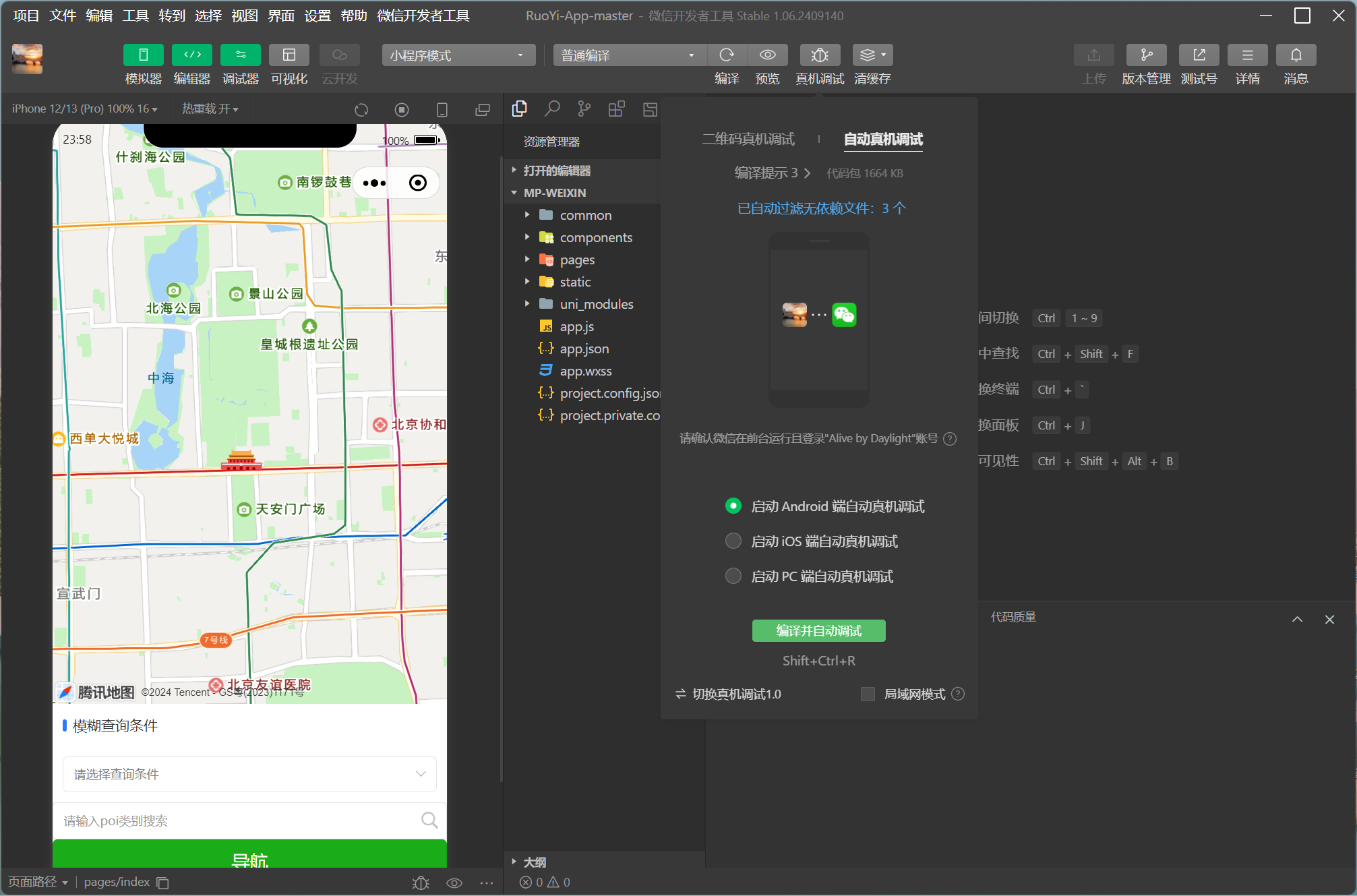


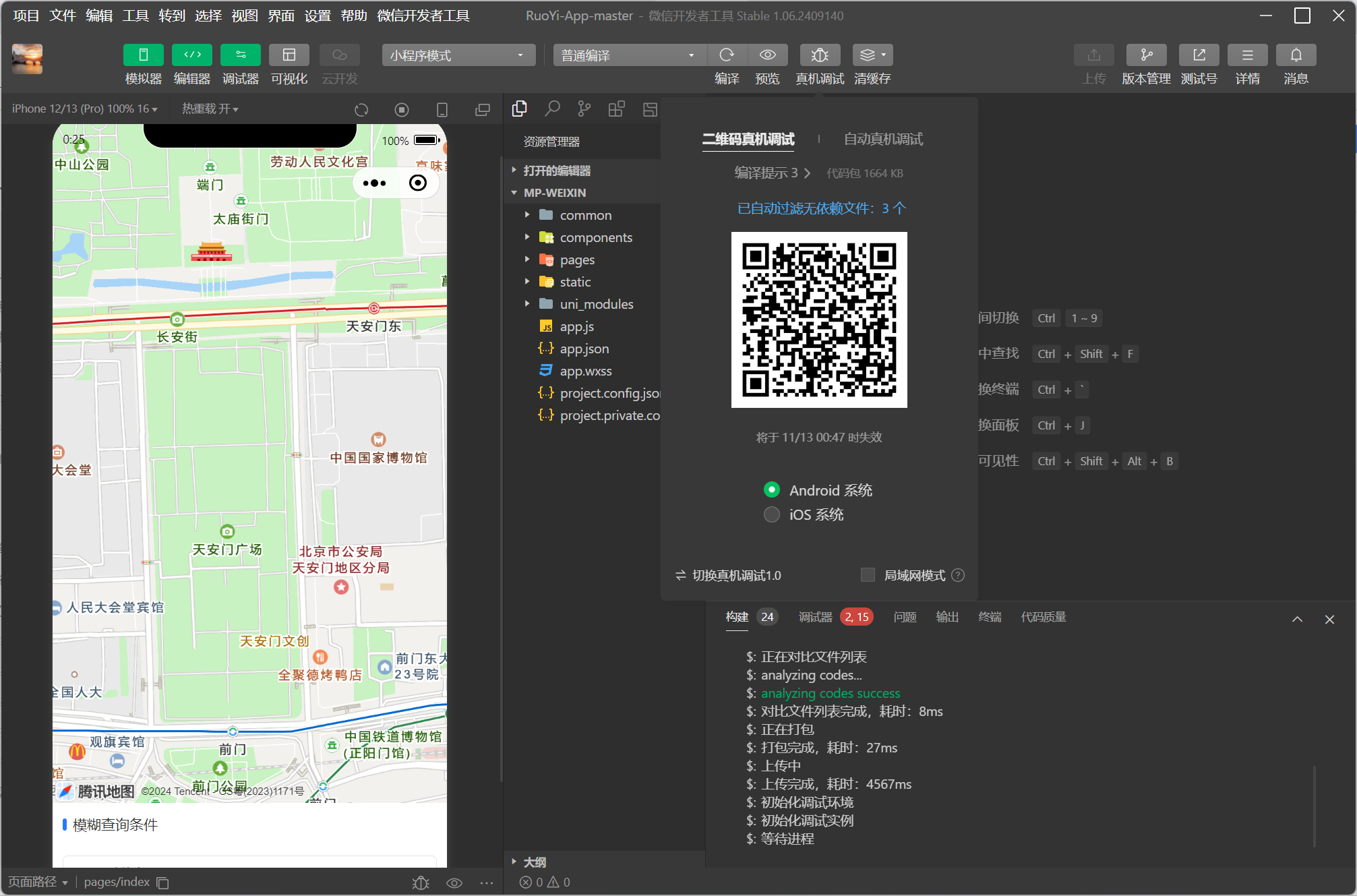
本地部署图



云端部署图

本地部署方式包括使用HBuilderX进行前端开发，并通过微信开发者工具模拟器在本地测试微信小程序代码。SpringBoot后台部署在本地服务器上，与本地托管的MySQL数据库和Redis缓存进行数据存储和缓存交互。该部署方式用于开发和本地测试。

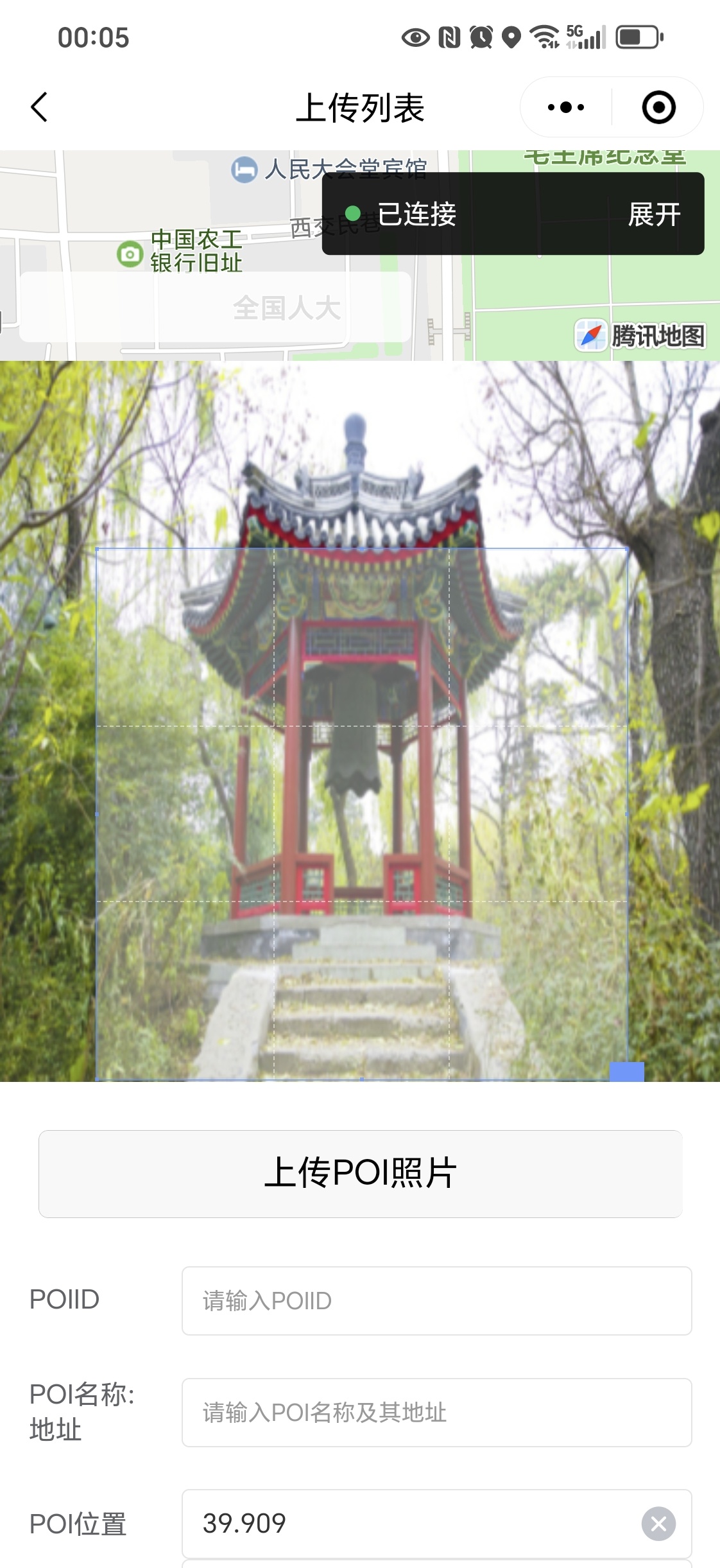
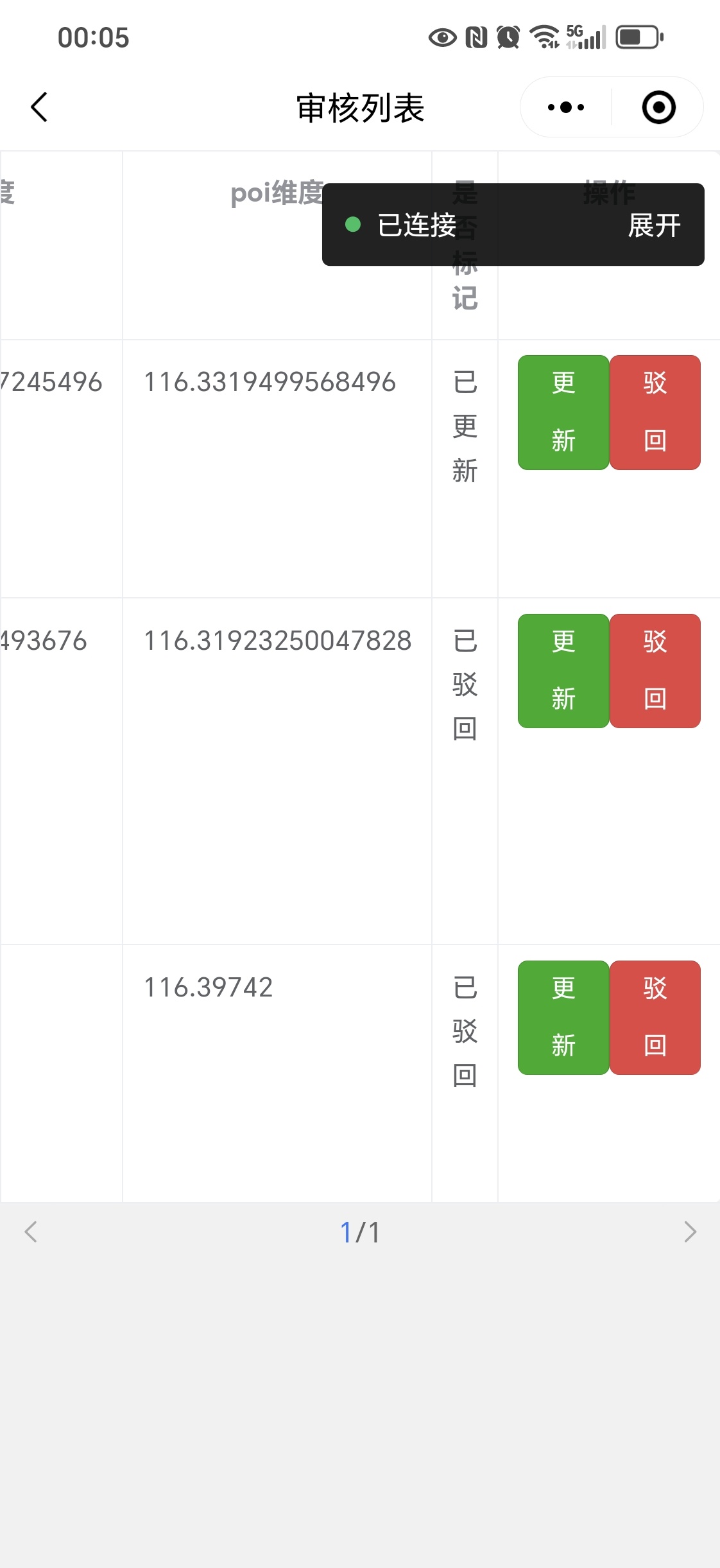




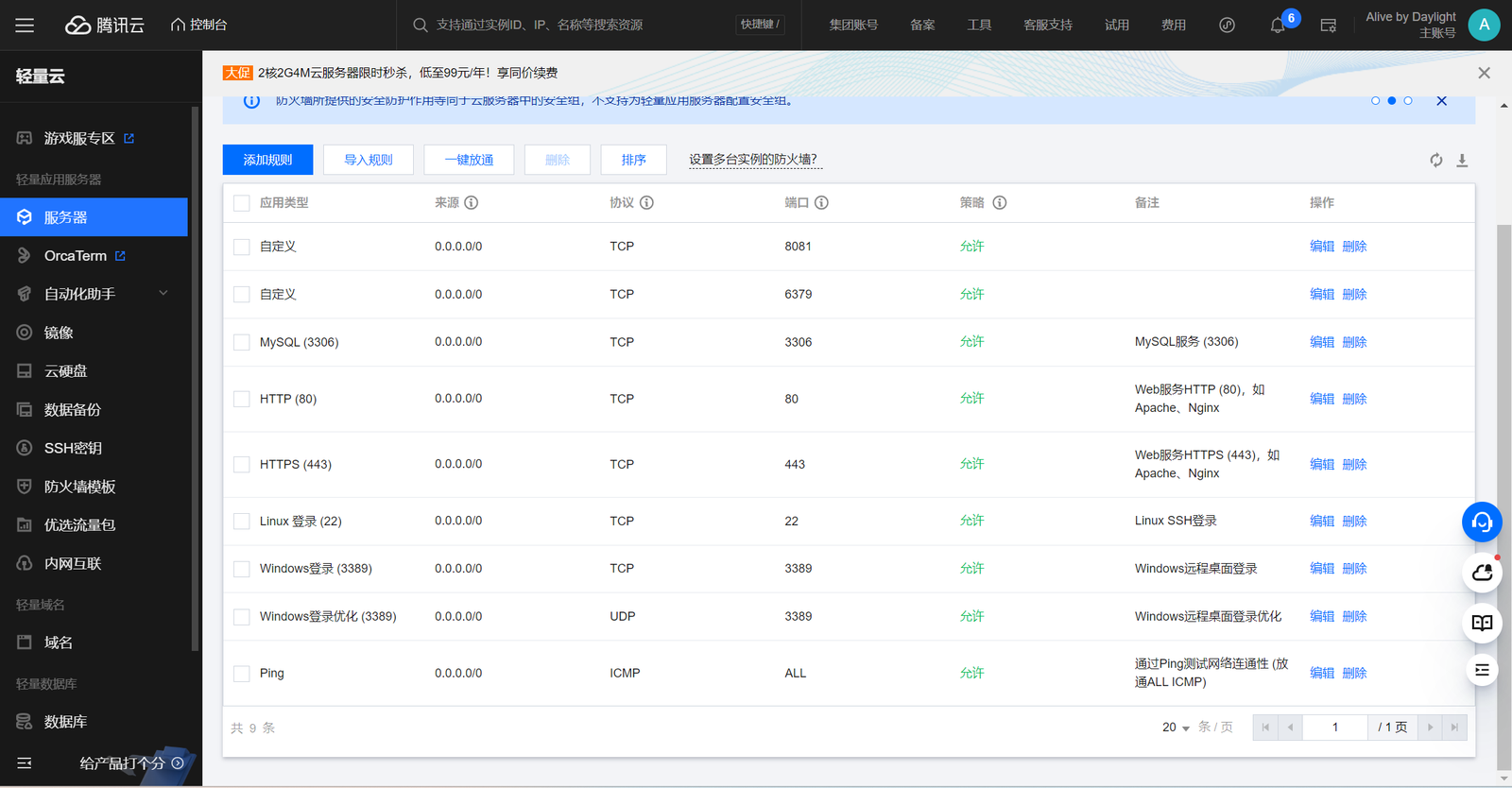
云端部署方式则是在移动设备上运行微信小程序代码，与部署在云端的Spring Boot后台进行交互。后台与云端托管的MySQL数据库和Redis缓存进行数据查询和缓存。该部署方式用于实际生产环境，用户通过移动应用访问云端资源。

如上图所示，点击“真机调试”-->”编译并自动调试“，或者扫描二维码，均可以使微信小程序的代码成功在移动端运行。

运行成果：



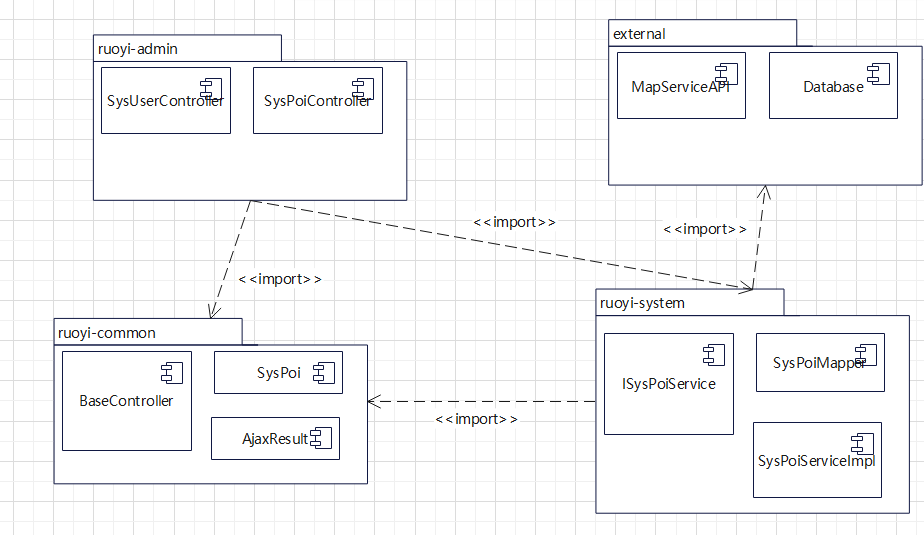
Android（移动端）真机运行成功



腾讯云服务器部署详情

## 3.4开发视图

开发视图定义在开发环境中软件的静态组织结构。开发视图关注软件模块的组织和管理，通常涉及软件的可扩展性、可移植性、可重用性等。在UML中，开发视图可以通过包图来表示，展示代码模块之间的关系。



从上面的包图可以分析得出：  
ruoyi-admin：控制层模块，包含具体的控制器类（如SysPoiController、SysUserController和SysRoleController），负责处理用户的请求，调用服务层接口，并将结果返回给用户。控制层依赖于基础控制器BaseController以复用通用逻辑。

ruoyi-common：公共模块，包含基础类和工具类，如BaseController、AjaxResult（用于封装返回结果）以及POI相关的实体类SysPoi和查询条件类SysSearch。该模块提供了业务无关的通用功能，支持其他模块复用。

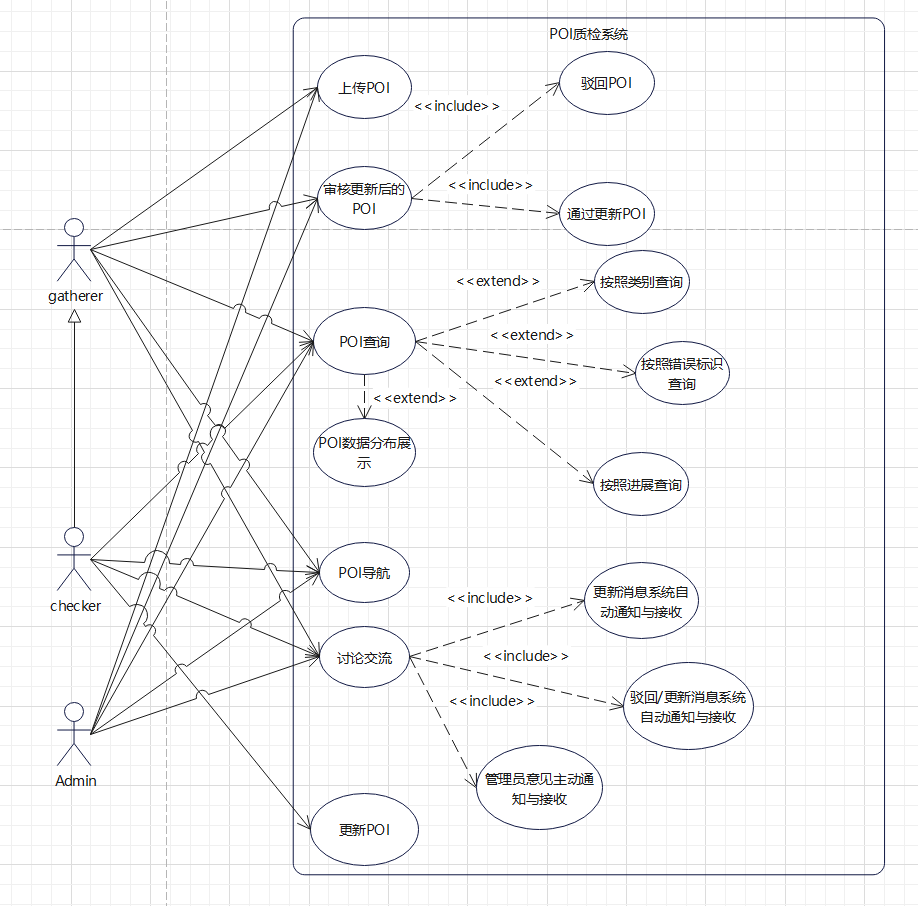
ruoyi-system：服务层和数据访问模块，包含业务接口ISysPoiService及其实现类SysPoiServiceImpl，并通过SysPoiMapper访问数据库执行SQL操作。

external：外部依赖模块，包括数据库和外部地图服务API（MapServiceAPI）。数据库用于存储和检索POI相关数据，地图服务API支持导航功能。

通过开发视图的包图，清晰展示了项目代码模块的静态组织结构及其关系。模块之间职责分明，依赖合理，通过分层架构实现了良好的可扩展性、可移植性和可重用性。这种模块化设计能够有效提升项目的维护性和扩展性。

## 3.5场景视图

场景视图作为其他四个视图的补充，通过use case（用例）将其他几个视图串联起来，对它们进行解释和说明。场景视图关注系统的参与者与功能用例间的关系，反映系统的最终需求和交互设计。在UML中，场景视图通常用用例图来表示。



上面的用例图体现出我们的项目满足了综合需求：

地理信息平台支持（地图查询、显示、路径计算、导航）

采集者：上传POI照片、拍照位置、 POI分类及基本描述信息

核验者：发现错误后，标识出错POI,上传更新后的POI照片、拍照位置、 POI分类及基本描述信息

系统向采集者发消息，通知有错误数据需订正，采集者复核后可选择更新或驳回

系统向核验者者发消息，通知采集者处理结果

可在地图上按类别查询显示POI数据分布、错误标识及进展进展

可使用地图平台导航功能，导航至指定POI位置

支持采集者与核验者建立讨论，交流错误标识详情，无法达成共识时由后台决定支持哪一方

# 团队成员任务分工、工作量比例。

周珏帆：总体分工50%.需求分析与建模、软件体系结构设计，后端设计模式设计，后端（Data access、Business、Controller及其接口）开发，前端路径规划、POI查询功能实现，POI上传功能模块开发；POI更新、驳回逻辑Debug，前端消息机制功能评审；《周报》撰写，《部署说明》撰写，《课程设计报告》脑图、用例图、环境限制分析、软件体系结构、创建型设计模式、行为型设计模式、4+1视图模型、总结部分撰写与UML作图；演示PPT的2、3部分制作。云服务器（Redis、MySQL、后端）部署、项目统筹规划与进度管理。

曹尚伟：总体分工50%.需求分析与建模、后端设计模式设计，数据库（MySQL、Redis）设计，前端POI更新、驳回逻辑开发，前端消息机制功能实现；POI上传功能、POI查询功能审核与修改；《周报》撰写，《部署说明》评审，《课程设计报告》顺序图、结构型设计模式、行为型设计模式撰写与UML作图，环境限制分析、4+1视图模型模块、创建型设计模式模块评审；演示PPT的1、4部分制作。云服务器维护。

# 总结、课程学习心得。

## 5.1总结

在这个地理信息平台项目中，我们的目标是构建一个支持地图查询、显示、路径计算与导航功能的系统。该平台不仅允许采集者上传POI数据（包括照片、位置、分类和描述信息），还支持核验者发现错误并进行数据更新。通过与采集者和核验者的互动，系统能够确保数据的准确性，同时支持实时消息通知和协作讨论功能。当采集者和核验者无法达成共识时，系统将通过后台决策支持来解决争议。

我们的开发框架选择了Vue和Spring Boot，并采用了UniApp技术框架来开发微信小程序。前端开发使用了UniApp官方IDE HBuilderX，后端开发则使用了IntelliJ IDEA，数据存储选用了MySQL 8.0CE，而缓存层则使用了Redis。

在开发过程中，我们支持两种部署方式：本地部署和云端部署。在本地开发阶段，前端通过微信开发者工具模拟器进行测试，云端部署则确保系统能够在iOS和Android设备上运行。系统的地图功能不仅可以查询POI数据的分布，还支持导航至指定的POI位置，并且支持采集者与核验者之间的讨论，确保数据修改的正确性。

通过这次项目的开发，我们深入实践了前后端分离、N层架构以及基于角色的权限管理，使得系统在设计和实现上更加高效和灵活。

## 5.2课程学习心得

在熊老师教授的《软件体系结构与设计模式》课程中，我们系统地学习了软件架构设计和设计模式的核心理念。这门课程让我们认识到，一个合理的架构设计对软件系统的长期发展至关重要，尤其是在复杂的项目中，架构的选择直接影响到系统的可扩展性、可维护性和性能。

课程中，我们学习了多种设计模式，如单例模式、工厂模式、观察者模式等，并通过案例分析了解了这些设计模式如何在实际开发中解决问题。通过实际的项目经验，我们发现，设计模式不仅能够帮助我们优化代码结构，提升模块的复用性，还能在面对复杂问题时提供清晰的解决思路。

此外，课程还强调了架构决策的重要性。在本项目中，我们通过使用N层架构和前后端分离的模式，确保了系统的灵活性和可扩展性。选择合适的架构不仅能够提高开发效率，还能在面对需求变化时更容易进行调整和优化。

通过这门课程的学习，我们不仅掌握了设计模式的理论知识，更学会了如何在实际项目中应用这些模式，解决具体的设计问题。我们也更加明白了架构设计在整个软件开发生命周期中的重要性，未来我们将更加注重在设计阶段对系统架构的合理规划。最重要的是，经过熊老师的指点，我们领悟到了“界面设计”与“功能设计的严谨性”是我们今后需要重点提升的领域。

**课程设计成绩：**

|  |  |  |
| --- | --- | --- |
| **评语：** | | |
| **得分：** |  | **时间：2024年11月 日** |