移动互联网技术及应用

大作业报告

题目： PotChat—多角色大模型聊天系统的设计与实现

类型：应用系统设计实现

姓名	班级	学号
郭晨旭	**2021211308**	**2021211184**

2024.4

目录

目录 2
1. 相关技术 4
1.1. Spring Boot 4
1.2. Sa-Token 4
1.3. Spring Mail 5
1.4. Flutter 5
1.5. Getx 6
1.6. Dio 6
1.7. Redis 6
1.8. MySQL 7
1.9. MongoDB 7
1.10. Server-Send Events 8
1.11. 百度智能云人脸识别 API 8
1.12. 星火认知大模型 API 9
1.13. 七牛云存储 API 10
2. 系统功能需求 11
2.1. 需求分析 11
3.1.1. 注册功能 11
3.1.2. 登录功能 11
3.1.3. 用户主页功能 11
3.1.4. 添加人脸功能 11
3.1.5. 多角色对话功能 12
3.1.6. 技术实现 12
2.2. 系统功能描述 12
3. 系统设计与实现 13
3.1. 总体设计 13
3.1.1. 总体架构 13
3.1.2. 功能设计 13
3.1.3. 界面设计 14
3.2. 系统组成 19
3.3. 各层模块设计 19
3.3.1. 用户模块 19
3.3.2. 人脸模块 23
3.3.3. 七牛云模块 24
3.3.4. 邮箱模块 25
3.3.5. 验证码模块 25
3.3.6. 对话模块 26
3.3.7. 缓存模块 30
3.4. 项目亮点 30
4. 系统可能的扩展 32
5. 总结体会 33

1. 相关技术

1.1. Spring Boot

本项目采用前后端分离的架构进行开发，后端使用 Spring Boot 框架。

Spring Boot 是一个开源的 Java 基础框架，由 Pivotal 团队（现为 VMware 旗下）开发，旨在简化 Spring 应用的创建和开发过程。Spring Boot 通过提供默认配置和一系列启动器（starters）来减少开发人员需要手动配置的工作量，使得开发者能够快速搭建和部署独立的、生产级别的基于 Spring 框架的应用。Spring Boot 具有以下这些良好的特性：

1. 独立运行: Spring Boot 应用可以打包成一个独立的 JAR 文件，这个 JAR 文件包含了所有必要的依赖项，可以直接运行，无需部署到外部的应用服务器。

2. 自动配置: Spring Boot 提供了大量的自动配置，这些配置会根据项目中的类路径依赖自动生效。例如，如果你的项目中包含了 spring-boot-starter-web，那么 Spring Boot 会自动配置所有需要的 Web 组件，如 Tomcat 和 Spring MVC。

3. 内嵌服务器: 内嵌了 Tomcat、Jetty 等 Web 容器，无需单独部署 Web 应用到外部服务器。

4. 生产就绪: 提供了一系列的生产级特性，如健康检查、度量信息收集、外部化配置等

5. 无代码生成和 XML 配置: Spring Boot 不需要生成大量的样板代码，也不需要复杂的 XML 配置文件，大多数情况下使用注解和一些属性文件即可完成配置。

1.2. Sa-Token

本项目使用 Sa-Token 完成用户鉴权操作。

Sa-Token 是一个轻量级的 Java 权限认证框架，它主要用于解决登录认证、权限认证、Session 会话、单点登录、OAuth2.0、微服务网关鉴权等一系列权限相关问题。这个框架的设计目标是简化权限认证的复杂性，使得开发者能够更简单、更快速地实现安全认证功能。其核心特性如下：

1. 登录认证：Sa-Token 提供了简单的登录认证方法，开发者可以通过几行代码实现用户的登录和登出功能。

2. 权限认证：框架支持权限和角色的认证，可以轻松地对用户的权限进行控制。

3. Session 会话：Sa-Token 管理用户的会话信息，支持全端共享 Session 和单端独享 Session。

4. 单点登录：支持多种单点登录模式，无论是否跨域、是否共享 Redis，都可以实现单点登录。

5. OAuth2.0：基于 RFC-6749 标准编写，实现了 OAuth2.0 标准流程的授权认证，支持 openid 模式。

1.3. Spring Mail

在注册和添加人脸校验时，本项目使用 Spring Mail 向用户邮箱发送验证码进行身份校验。

Spring Mail 是 Spring 框架提供的一个用于电子邮件发送的模块，它简化了基于 Java 的应用程序发送电子邮件的过程。通过使用 Spring Mail，开发者可以轻松地集成邮件发送功能，无论是简单的文本邮件、HTML 格式邮件，还是带有附件和内嵌图片的复杂邮件。其核心特性如下：

1. 简化邮件发送：Spring Mail 提供了一个简单的 API，用于发送邮件，无需深入了解底层的邮件协议，如 SMTP、POP3 和 IMAP。

2. 支持多种邮件格式：可以发送纯文本邮件、HTML 邮件以及带有 MIME 类型的复杂邮件。

3. 邮件模板：支持邮件模板的使用，可以轻松地将动态内容集成到邮件中。

4. 附件和内嵌图片：可以添加文件作为附件发送，或者将图片等资源内嵌到邮件正文中。

5. 邮件验证：支持邮件发送前的验证，如 SMTP 服务器的 SSL/TLS 配置，以及发送者和接收者地址的验证。

6. 异步发送：支持异步发送邮件，提高应用程序性能，特别是在发送大量邮件时。

7. 集成测试：提供了测试邮件发送的功能，方便开发者进行单元测试和集成测试。

1.4. Flutter

本项目前端使用 Flutter 框架及进行开发。

Flutter 是由 Google 开发的一个开源 UI 软件开发工具包，用于创建跨平台的高性能、高保真的移动应用、Web 应用和桌面应用。它首次发布于 2018 年，迅速获得了开发者社区的广泛关注和使用。Flutter 的主要特点是使用了 Dart 语言进行编程，这是一门由 Google 开发的现代编程语言，具有结构化、强类型和垃圾回收等特点。其核心特性如下：

1. 跨平台开发：Flutter 允许开发者使用单一代码库创建 Android 和 iOS 应用程序，这意味着可以节省时间和资源，同时保持两个平台的一致性。

2. 富有表现力的 UI：Flutter 提供了丰富的预制组件和强大的布局系统，使得创建美观、流畅的用户界面变得简单快捷。

3. 高性能：Flutter 应用编译为原生 ARM 代码，这意味着它们可以在设备上高效运行，提供接近原生应用的性能。

4. 热重载：在开发过程中，Flutter 支持热重载，这意味着开发者可以在应用程序运行时实时看到更改，无需重新启动应用程序。

5. Dart 语言：Flutter 使用 Dart 语言，它是一种面向对象的语言，支持类、接口、混合、泛型和扩展等特性，同时具有出色的工具支持。

6. 强大的社区和生态系统：Flutter 拥有一个活跃的开发者社区和不断增长的生态系统，提供了大量的插件和库，可以帮助开发者快速实现各种功能。

7. 可访问性：Flutter 支持辅助功能，如屏幕阅读器，这使得应用程序对所有用户都更加友好。

1.5. Getx

在使用 Flutter 开发时，我使用 Getx 框架进行状态管理、页面跳转和依赖注入。

GetX 是一个用于 Flutter 的开源状态管理库，它提供了一种新颖的方式来处理应用程序的状态管理。GetX 的核心思想是通过简化状态更新和 UI 渲染之间的连接，来提高开发效率和性能。它不仅适用于状态管理，还可以用于路由管理、依赖注入等多种场景。其核心特性如下：

1. 响应式状态管理：GetX 使用响应式编程的概念，当状态发生变化时，会自动更新 UI，而无需手动调用 setState()方法。

2. 简洁的 API：GetX 的 API 设计简洁直观，易于理解和使用，可以快速构建和维护状态管理逻辑。

3. 性能优化：通过优化渲染逻辑，GetX 减少了不必要的 UI 更新，从而提高了应用的性能。

4. 无锁状态管理：GetX 的状态管理是无锁的，这意味着在多线程环境下，状态的更新和访问更加安全和高效。

5. 路由管理：GetX 内置了强大的路由管理功能，可以轻松实现页面跳转和导航。

6. 依赖注入：通过依赖注入，GetX 允许开发者将应用程序的各个部分解耦，使得代码更加模块化和可测试。

7. 易于测试：GetX 的状态管理模型使得编写单元测试变得更加简单，有助于提高代码质量。

1.6. Dio

本项目中，前端使用 Dio 库进行网络请求。

Dio 是一个强大的 HTTP 客户端库，用于在 Flutter、Dart 和其他支持 HTTP 请求的应用程序中进行网络请求。它由一款流行的 Flutter 网络请求库演变而来，旨在提供一种简单、快速且高效的方式来处理 HTTP 请求和响应。其核心特性如下：

1. 支持 FormData：Dio 支持上传文件和数据，可以轻松地创建多部分表单数据请求。

2. 拦截器：Dio 允许开发者使用拦截器来统一处理请求和响应，例如添加请求头、处理错误等。

3. 请求取消：Dio 提供了请求取消的功能，可以取消不再需要的请求。

4. 超时和重试：Dio 支持设置请求超时时间，并可以配置重试策略。

5. 响应类型灵活：Dio 支持多种响应类型，包括 JSON、字符串、文件等。

6. 支持 HTTPS：Dio 支持 HTTPS 请求，包括自签名证书的处理。

7. 拦截器链：Dio 的拦截器链允许开发者自定义请求和响应的处理流程。

8. 易于使用：Dio 的 API 设计简洁，使得创建和发送请求变得非常简单。

1.7. Redis

为提高系统性能，本项目使用了 Redis 作为缓存，缓存用户的个人信息和聊天记录。Redis（Remote Dictionary Server）是一个开源的，基于内存的高性能键值对数据库，支持多种数据结构，如字符串（strings）、列表（lists）、集合（sets）、有序集合（sorted sets）、散列（hashes）、

位图（bitmaps）、超日志（hyperloglogs）和地理空间（geospatial）索引半径查询。Redis 最初由 Salvatore Sanfilippo 在 2009 年发布，并由 Redis Labs 维护至今。其核心特性如下：

1. 基于内存：Redis 将大部分数据存储在内存中，从而提供极高的读写速度。

2. 数据持久化：尽管 Redis 是基于内存的，但它也提供了数据持久化机制，可以将内存中的数据保存到磁盘中，确保数据的安全性。

3. 支持多种数据结构：Redis 不仅仅是一个简单的键值存储，它支持丰富的数据结构，使得开发者可以根据不同的应用场景选择合适的数据结构。

4. 原子操作：Redis 的操作是原子性的，这意味着在执行复杂的命令时，要么全部执行，要么全部不执行，保证了数据的一致性。

5. 发布/订阅模式：Redis 支持发布/订阅模式，这允许客户端之间通过 Redis 进行消息的发布和订阅，非常适合构建实时消息系统。

6. 高可用性和分布式：通过 Redis Sentinel 和 Redis Cluster，Redis 提供了故障转移和数据分片的机制，确保了高可用性和水平扩展能力。

7. 丰富的客户端库：Redis 拥有多种编程语言的客户端库，方便开发者在不同的环境下使用 Redis。

1.8. MySQL

为了持久化用户个人信息，本项目采用 MySQL 数据库持久化用户信息。

MySQL 是一个流行的开源关系型数据库管理系统（RDBMS），由瑞典 MySQL AB 公司开发，后被 Sun Microsystems 收购，最终在 2012 年被甲骨文（Oracle）公司收购。MySQL 使用结构化查询语言（SQL）进行数据库管理和操作，是目前最广泛使用的数据库之一。其核心特性如下：

1. 开源和免费：MySQL 社区版是免费的，对于大多数用户来说，这是一个很大的优势，尤其是对于初创公司和小型企业。

2. 跨平台：MySQL 支持多种操作系统，包括 Windows、Linux、macOS 等。

3. 高性能：MySQL 提供了高性能的数据处理能力，尤其是在 Web 应用程序中，它能够处理大量的数据和高并发的请求。

4. 可扩展性：MySQL 可以运行在各种硬件上，从小型个人服务器到大型企业级服务器。

5. 安全性：MySQL 提供了强大的数据安全特性，包括加密、访问控制和安全认证。

6. 灵活性：MySQL 支持多种存储引擎，如 InnoDB、MyISAM 等，每种引擎都有其特定的功能和性能特点。

7. 复制和集群：MySQL 支持数据复制和集群，这有助于提高数据的可靠性和可用性。

8. 事务支持：MySQL 支持事务处理，确保数据的完整性和一致性。

1.9. MongoDB

为了避免存储过大数据影响 MySQL 性能，本项目使用文档性数据库 MongoDB 来存储用户每次的对话记录。

MongoDB 是一个开源的 NoSQL 文档型数据库，使用文档来存储数据，这些文档类似于 JSON 对象，由键值对组成。MongoDB 以其高性能、高可用性和易扩展性而闻名，特别适合处理大量的数据和复杂的查询操作。它由 MongoDB Inc.开发和维护，并提供了一个名为 MongoDB Atlas 的云数据库服务，允许用户在云上部署和运行 MongoDB。其核心特性如下：

1. 文档导向：MongoDB 存储数据的方式是灵活的 BSON（类似于 JSON）文档，这种格式使得数据结构更加自然和易于理解。

2. 高性能：MongoDB 提供高速的数据读写操作，特别适合处理大量的数据和高并发请求。

3. 高可用性：通过副本集（Replica Sets）提供数据的高可用性和自动故障转移。

4. 扩展性：MongoDB 支持水平扩展，可以通过分片（Sharding）将数据分布在多个服务器上，从而处理大规模的数据集。

5. 强大的查询语言：MongoDB 提供了一个强大的查询语言，支持丰富的查询操作，包括对文档的各个字段进行查询、更新和删除。

6. 索引：MongoDB 支持多种类型的索引，包括文本索引、地理空间索引等，以优化查询性能。

7. 聚合框架：MongoDB 提供了一个强大的聚合框架，用于处理复杂的数据处理和分析任务。

8. 灵活的数据模型：MongoDB 不需要预定义的模式，文档的字段可以动态添加，非常适合快速开发和迭代。

1.10. Server-Send Events

在对话时，只需要服务端向客户端单向传输消息，所以我使用了 Server-Send Events 来推送数据。

Server-Sent Events（简称 SSE）是一种允许服务器主动向客户端发送事件的技术。与传统的 HTTP 请求不同，SSE 在客户端建立一个到服务器的单向连接，服务器可以通过这个连接发送消息给客户端，而无需客户端重新发起请求。其核心特性如下：

1. 单向通信：SSE 是单向的，只能由服务器向客户端发送消息，客户端不能通过这个连接发送消息到服务器。

2. 实时性：SSE 提供了一种实时通信的手段，非常适合需要服务器推送最新数据到客户端的应用场景，如股票行情、新闻更新、社交媒体动态等。

3. 轻量级：与 WebSocket 相比，SSE 更轻量级，因为它基于标准的 HTTP 协议，不需要额外的协议支持。

4. 自动重连：如果连接断开，浏览器会自动尝试重新连接，这对于保持客户端与服务器的持续通信非常有用。

5. 简单易用：在客户端使用 SSE 非常简单，只需要创建一个 EventSource 对象，并指定服务器端点即可开始接收事件。

1.11. 百度智能云人脸识别 API

在本项目中，我增加了人脸登录的功能，因此我使用了百度智能云的人脸检测和人脸对比的 API。人脸检测的 API 用于在添加人脸时进行校验，人脸对比的 API 用于在人脸登录时进行校验。

百度智能云的人脸识别技术主要基于深度学习算法，这些算法经过大量真实人脸数据的训练，具备高度的识别和分析能力。以下是百度智能云人脸识别技术的一些核心技术和优势：

核心技术

1. 深度学习算法：利用深度神经网络进行人脸的特征提取和识别，能够处理和分析复

杂的人脸图像数据。

2. 多模态融合：结合人脸、语音、动作等多种生物特征信息进行综合分析，提高对假体攻击的识别率。

3. 活体检测技术：通过分析人脸图像中的微小动作和纹理变化，有效区分真实人脸和照片、视频等伪造攻击。

4. 人脸对比和搜索：通过高效的比对算法，实现在大规模人脸数据库中快速准确地搜索和匹配目标人脸。

技术优势

1. 高准确性：深度学习算法使得人脸识别具有高准确率，即使在复杂多变的环境下也能保持稳定的识别效果。

2. 实时响应：优化的算法和强大的计算资源确保了人脸识别的快速响应，适用于实时监控和快速身份验证场景。

3. 安全性：活体检测技术和多模态融合有效抵御各种攻击手段，保障了人脸识别系统的安全性。

4. 易用性：提供简单易用的 API 接口，开发者可以快速集成人脸识别功能，无需深入了解底层算法。

5. 可扩展性：支持大规模的人脸数据库管理，能够应对不同规模的应用需求。

6. 权威认证：百度智能云的人脸识别系统通过了公安部一所的安全性能认证，证明了其在安全性能方面的卓越表现。

1.12. 星火认知大模型 API

本项目的聊天功能主要是使用星火认知大模型的 API。

星火聊天 API 使用了先进的深度学习技术和自然语言处理（NLP）算法，这些技术使得 API 具备了强大的语言理解和生成能力。以下是星火聊天 API 所使用的主要技术和技术优势：

主要技术

1. 深度学习算法：星火聊天 API 基于深度神经网络模型，这些模型通过大量的数据训练，能够理解和生成自然语言。

2. 大语言模型：API 利用大语言模型来处理和生成文本，这种模型能够捕捉语言的复杂性和多样性。

3. 多模态理解：除了文本处理，API 还能够结合语音识别和合成技术，提供跨模态的交互体验。

4. 内置插件：星火聊天 API 支持多种内置插件，如搜索、天气、日期、诗词等，这些插件扩展了 API 的功能，使其能够处理特定类型的请求。

5. Function Call 功能：在高级版本中，API 支持用户自定义 function，使得 API 能够根据用户的描述执行特定的任务或生成特定的内容。

技术优势

1. 高度的灵活性：星火聊天 API 能够适应多种对话场景，生成个性化的回复，满足不同用户的需求。

2. 强大的理解能力：通过深度学习算法，API 具备了高度的语言理解能力，能够准确把握用户的意图和上下文。

3. 高效的交互体验：API 支持快速响应，能够在短时间内处理大量请求，提供流畅的用户体验。

4. 安全性和稳定性：使用 wss 协议和其他安全措施确保数据传输的安全性，同时 API

的稳定性保证了服务的可靠性。

5. 易于集成：API 的设计使得开发者可以轻松地将其集成到现有的系统和应用中，无需复杂的配置和改动。

6. 持续学习和进化：API 不断从新的数据中学习，以适应不断变化的语言使用和用户需求，保证了长期的适应性和有效性。

星火聊天 API 的这些技术和优势使其成为了构建智能对话系统和增强现有应用智能性的强大工具。无论是在提供客户支持、内容创作还是其他需要自然语言交互的场合，星火聊天 API 都能够提供高效、智能的解决方案。

1.13. 七牛云存储 API

用户在使用本系统注册时可以自定义用户头像,这个地方本项目使用了七牛云存储 API。

七牛云存储提供了多种 API 接口用于文件的上传，这些 API 使用了 HTTP 协议和 multipart/form-data 格式来处理文件上传请求，确保了高效、稳定和安全的文件传输。七牛云存储 API 主要有以下技术优势：

1. 简单易用：通过 HTML 表单或者直接构造 HTTP 请求，开发者可以轻松实现文件的上传，无需复杂的配置或额外的库。

2. 安全性：上传凭证机制确保了上传过程的安全性，防止了未授权的访问和数据泄露。

3. 灵活性：支持自定义变量和元数据，使得开发者可以在文件上传过程中添加额外的信息，为后续的处理提供了更多的灵活性和便利。

4. 高效性：断点续传和并发上传功能使得大文件的上传更加高效，尤其是在网络条件不佳的情况下，可以显著减少上传时间和失败率。

5. 稳定性：七牛云存储的服务器设计用于处理大量的并发请求，保证了上传服务的稳定性和可靠性。

6. 跨平台支持：七牛云存储的 API 可以在多种编程语言和平台上使用，包括但不限于 Java、Python、Go 等，为开发者提供了广泛的选择。

综上所述，七牛云存储的文件上传 API 通过使用标准协议、安全机制、自定义功能和高效的上传策略，为开发者提供了一个强大、灵活且易于使用的文件上传解决方案。

2. 系统功能需求

2.1. 需求分析

在开发软件时，进行详细的需求分析是确保项目成功的关键，下面是我对本项目的需求分析：

3.1.1. 注册功能

1. 用户可以通过注册功能创建新账户，需要提供至少头像、昵称、电子邮箱和密码字段。
2. 用户注册时，系统应验证邮箱的有效性，通过发送验证邮件来激活账户。
3. 系统应采用加密算法（如 bcrypt）对用户密码进行存储，确保即使数据泄露，密码也不会轻易被破解。

3.1.2. 登录功能

1. 用户在登录时需输入正确的邮箱和密码。
2. 登录还提供人脸登录方式，已经添加过人脸的用户可以使用人脸登录。
3. 系统应使用先进的人脸识别算法来验证用户身份，并与数据库中存储的人脸数据进行匹配。

3.1.3. 用户主页功能

1. 应提供用户主页功能，在此页面用户可以查看自己头像、昵称、登录的邮箱。
2. 用户可以在此页面添加人脸。
3. 用户可以在此页面退出登录

3.1.4. 添加人脸功能

1. 用户添加人脸时必须先使用登录邮箱接收验证码，校验操作人员的合法性。
2. 用户需通过前置摄像头进行人脸数据采集，系统应提供清晰的指导和足够的时间进行操作。
3. 用户上传人脸后应当校验人脸是否合法（图片内有且仅有一张人脸）。
4. 为了保护用户的安全，服务端不应该直接存储人脸图片，应该提取人脸的特征向量进行存储。

3.1.5. 多角色对话功能

1. 软件应支持多个具有不同对话风格和知识库的角色。
2. 用户应能够与这些角色进行自然语言对话，并根据上下文进行互动。
3. 系统应能够根据用户的输入和对话历史提供合适的回应。
4. 系统应提供角色选择界面，用户可以轻松切换对话角色。
5. 支持自定义角色，允许用户根据特定需求创建和训练新的角色。

3.1.6. 技术实现

1. 后端架构：采用 Spring Boot 主流框架开发后端服务。
2. 前端开发：使用跨平台的 Flutter 框架构建用户友好的界面，确保良好的兼容性和响应性。
3. 数据库设计：使用关系型数据库存储用户信息，使用 NoSQL 数据库存储大量的非结构化对话内容。
4. 人脸识别技术：集成高性能的人脸识别服务，百度智能云人脸识别 API。
5. 聊天功能实现：对接星火大模型 API。

2.2. 系统功能描述

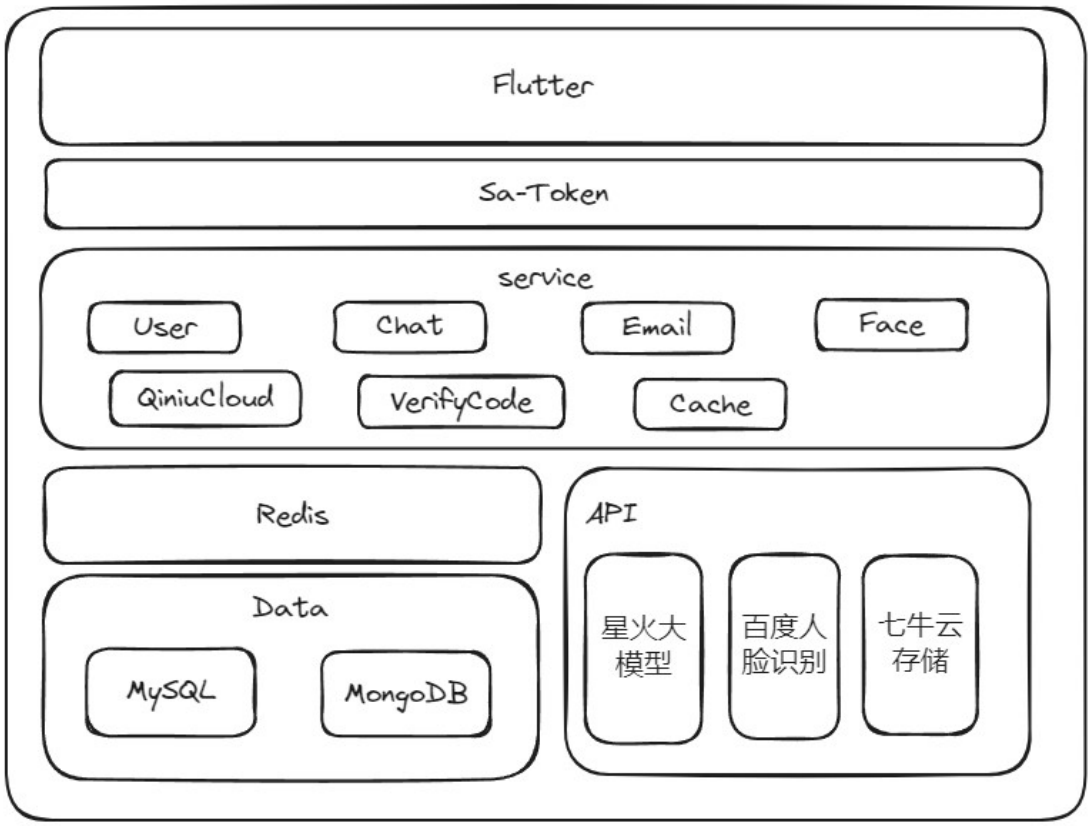
功能名称	功能描述
注册功能	用户提供头像、昵称、邮箱、密码和验证码注册本聊天系统
邮箱密码登录	用户输入邮箱密码，经过校验后登录本系统
人脸登录	用户输入邮箱后选择人脸登录，上传人脸后登录本系统
添加人脸	用户在校验已登录邮箱的邮箱后可以添加人脸
查看主页	用户可以查看自己的头像、昵称和注册邮箱
退出登录	用户可以退出登录，退出登录后返回登录页
管理对话	用户可以选择不同的角色进行对话，也可以对已有的对话历史记录进行管理（查看/删除/接着对话）

3. 系统设计与实现

3.1. 总体设计

3.1.1. 总体架构

本系统使用前后端分离的架构，总体的架构设计图如下：



前端：前端使用 Flutter 负责页面 UI 的开发。

后端：后端使用主流的 Spring Boot 框架进行接口开发和业务逻辑编写，使用 Sa-Token 框架进行鉴权，业务逻辑划分为七个模块。

Redis：使用 Redis 缓存使用过程中的数据，提高系统性能。

MySQL：用于存储结构化的信息，如用户的个人信息。

MongoDB：用于存储每次对话的问答数据。

外接 API：通过对接其他服务 API 实现聊天、人脸识别、文件上传等功能。

3.1.2. 功能设计

本项目主要主要有用户服务、聊天服务、邮箱服务、人脸服务、七牛云服务、验证码服务以及缓存服务这七大服务功能，每个模块实现的功能如下：

服务名称	功能名称	功能描述
用户服务	注册	用户提供头像、昵称、邮箱以及密码注册本系统
	发送验证码	向用户邮箱发送验证码
	邮箱密码登录	用户提供邮箱和对应密码登录系统
	人脸登录	用户提供密码和人脸登录
	添加人脸	用户添加人脸用于人脸登录
	查看用户主页	用户查看自己的头像、昵称以及注册邮箱
	退出登录	用户推出登录
聊天服务	查看历史聊天记录	查看在本系统中的历史对话记录
	查看聊天内容	选择某一聊天记录查看其详细内容
	聊天	和本系统聊天对话
	删除聊天记录	删除指定聊天记录
	新建聊天	选取合适的角色，新建一个聊天
邮箱服务	给邮箱发送信息	向指定邮箱发送邮件
人脸服务	校验人脸是否合法	校验上传人脸图片中有且仅有一张人脸
	人脸匹配	将两张人脸进行匹配
七牛云服务	上传文件	上传文件到七牛云
验证码服务	发送验证码	生成验证码并发送
	校验验证码是否正确	从 redis 中获取对应验证码并校验是否正确
缓存服务	管理缓存	对缓存的增删改查以及续期等操作

3.1.3. 界面设计

1. 首页

首页上方是项目 logo，中间用户可以选择注册或者登录

2. 注册页面

注册页面使用 ListView 罗列出文本框，用户需要填写响应的信息，验证码点击发送后会向指定邮箱发送验证码，并开始 60s 倒计时。

如果用户已经有系统账号，则可以直接跳转至登录页面登录，注册成功后也会自动跳转至登录页。

3. 登录页面

登录页面用户可以选择邮箱密码登录或者人脸登录，邮箱密码登录需要填写邮箱和密码，人脸登录则只需要填写邮箱。

如果用户没有系统账号，可以直接跳转至注册页注册账号。

登录成功后会跳转至历史会话页面。

4. 历史记录

用户在该页面可以在该页面查看历史对话记录，每条记录将展示对话描述和对话时间。

点击新建对话可以新建新的一次聊天对话。

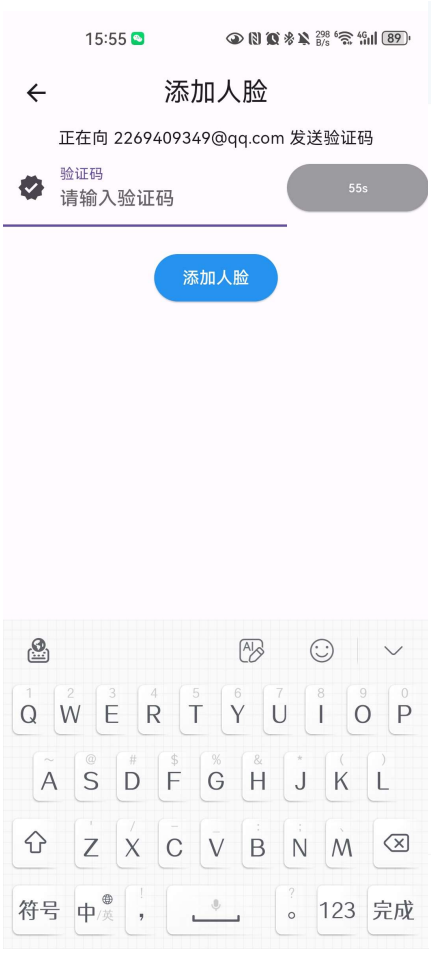
点击每条记录的删除，可以删除该条记录。

点击左上角可以查看用户个人信息。

5. 用户信息

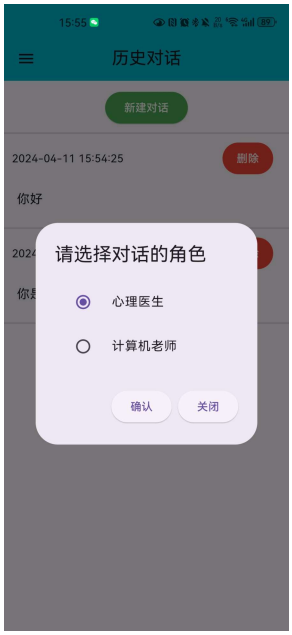
用户在该页面可以查看自己的头像、昵称和注册用的邮箱，点击下面各按钮可以执行对应操作。

6. 添加人脸



添加人脸前需要先发送验证码进行校验，右图为验证码格式。

7. 新建对话



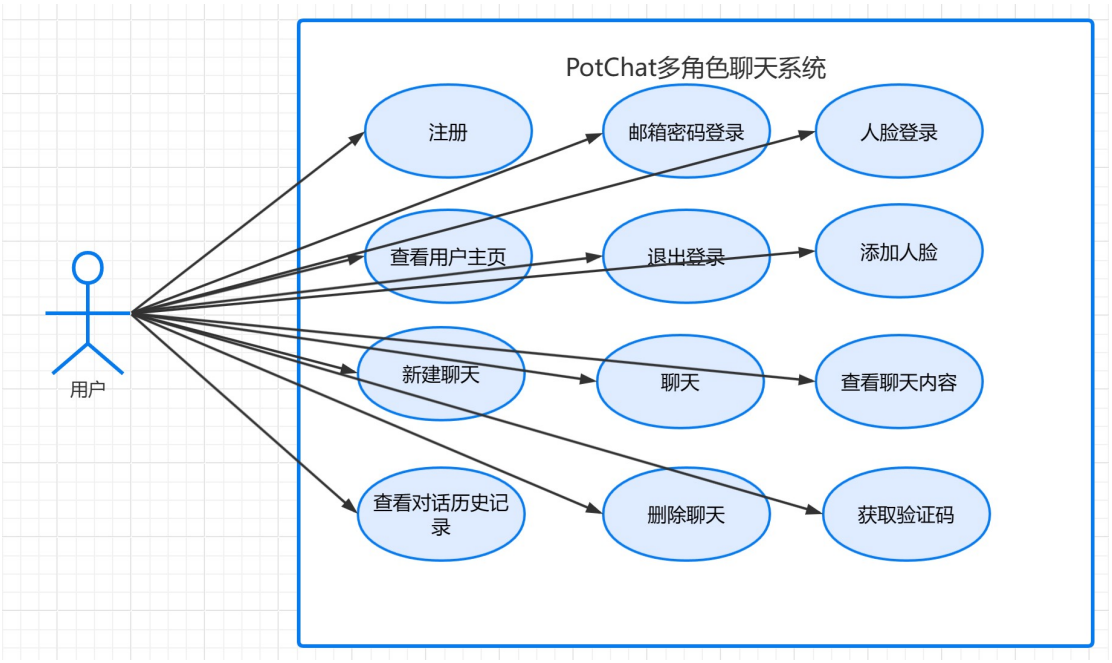
新建对话时会弹出对话框让用户选择角色，确认后即可创建新的对话。

8. 聊天页面

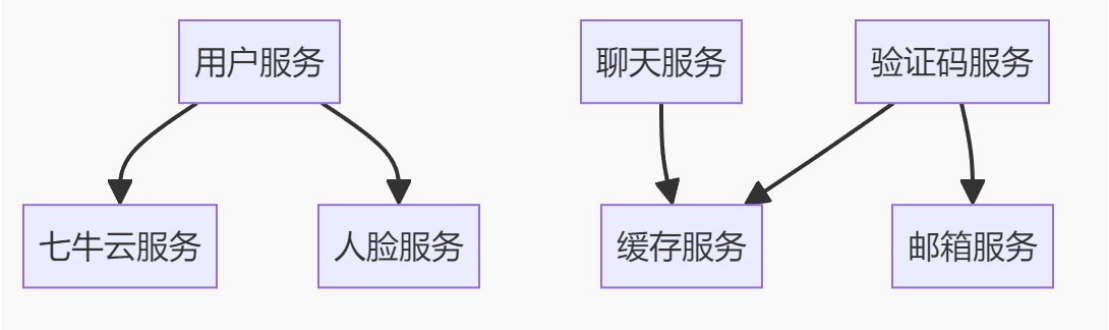
聊天页面上方是聊天描述；中间是历史记录，右边式用户输入，左边是系统回答，这里使用 ListView.builder()配合 Getx 实现记录动态刷新和每轮对话的打印机式输出；页面下方式输入框，用户可以输入问题与系统聊天。

3.2. 系统组成

本系统用例图如下：



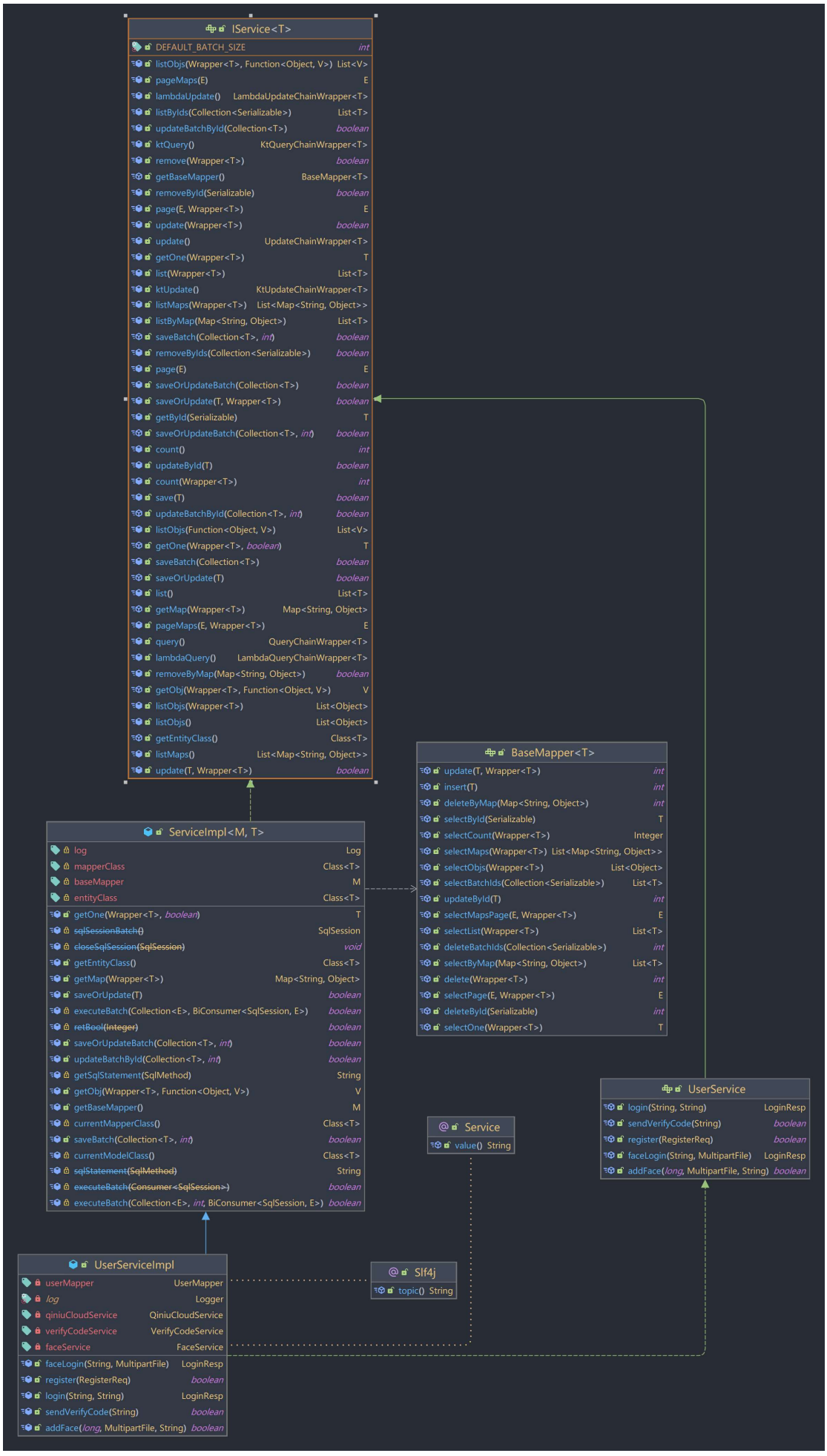
本系统的七个服务之间相互调用关系如下：



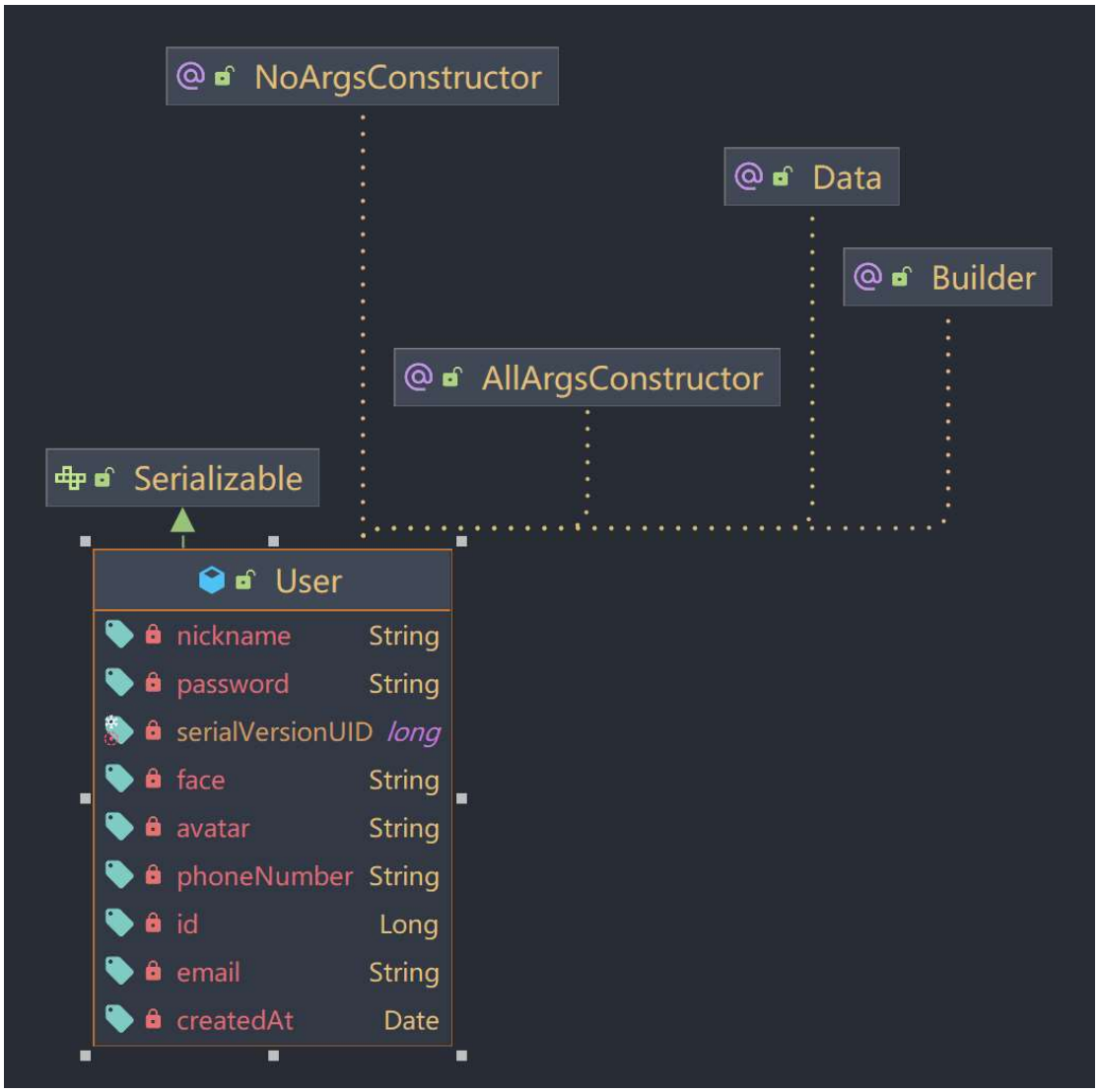
3.3. 各层模块设计

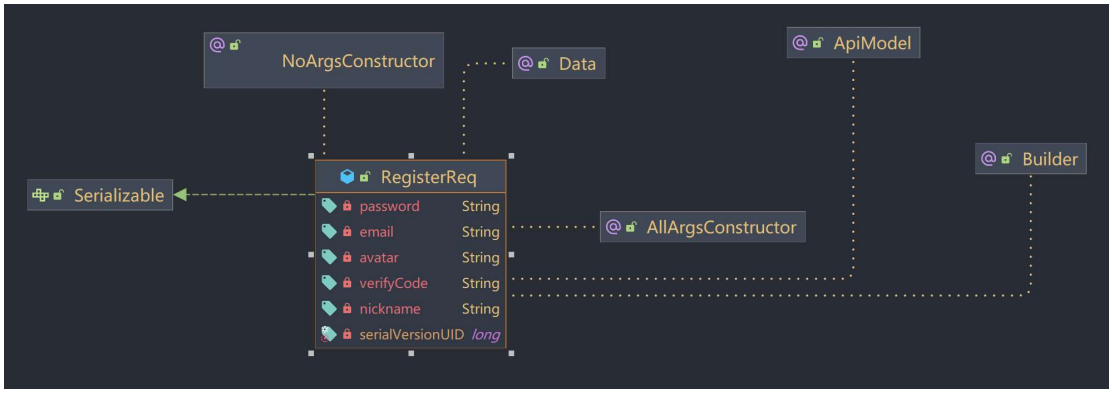
3.3.1. 用户模块

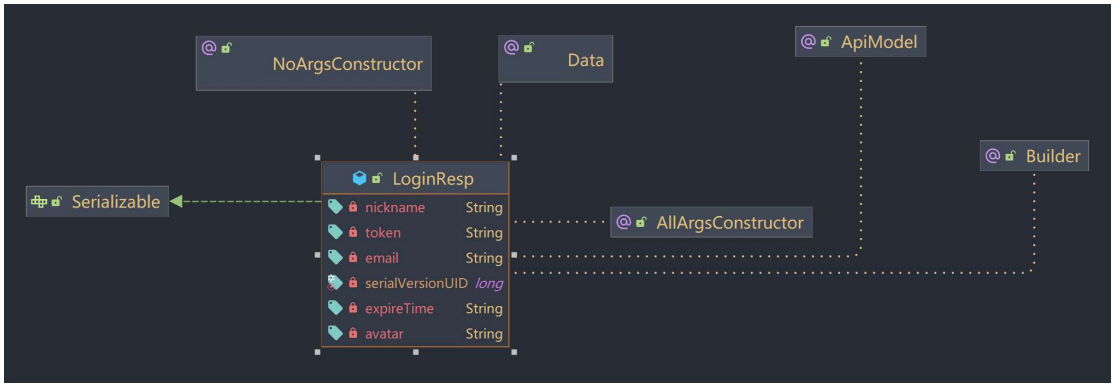
模块接口如下：


IService<T>
DEFAULT_BATCH_SIZE int
listObjs(Wrapper<T>, Function<Object, V>) List<V>
pageMaps(E) E
lambdaUpdate() LambdaUpdateChainWrapper<T>
listByIds(Collection<Serializable>) List<T>
updateBatchById(Collection<T>) boolean
ktQuery() KtQueryChainWrapper<T>
remove(Wrapper<T>) boolean
getBaseMapper() BaseMapper<T>
removeById(Serializable) boolean
page(E, Wrapper<T>) E
update(Wrapper<T>) boolean
update() UpdateChainWrapper<T>
getOne(Wrapper<T>) T
list(Wrapper<T>) List<T>
ktUpdate() KtUpdateChainWrapper<T>
listMaps(Wrapper<T>) List<Map<String, Object>>
listByMap(Map<String, Object>) List<T>
saveBatch(Collection<T>, int) boolean
removeByIds(Collection<Serializable>) boolean
page(E) E
saveOrUpdateBatch(Collection<T>) boolean
saveOrUpdate(T, Wrapper<T>) boolean
getById(Serializable) T
saveOrUpdateBatch(Collection<T>, int) boolean
count() int
updateById(T) boolean
count(Wrapper<T>) int
save(T) boolean
updateBatchById(Collection<T>, int) boolean
listObjs(Function<Object, V>) List<V>
getOne(Wrapper<T>, boolean) T
saveBatch(Collection<T>) boolean
saveOrUpdate(T) boolean
list() List<T>
getMap(Wrapper<T>) Map<String, Object>
pageMaps(E, Wrapper<T>) E
query() QueryChainWrapper<T>
lambdaQuery() LambdaQueryChainWrapper<T>
removeByMap(Map<String, Object>) boolean
getObj(Wrapper<T>, Function<Object, V>) V
listObjs(Wrapper<T>) List<Object>
listObjs() List<Object>
getEntityClass() Class<T>
listMaps() List<Map<String, Object>>
update(T, Wrapper<T>) boolean
BaseMapper<T>
update(T, Wrapper<T>) int
insert(T) int
deleteByMap(Map<String, Object>) int
selectById(Serializable) T
selectCount(Wrapper<T>) Integer
selectMaps(Wrapper<T>) List<Map<String, Object>>
selectObjs(Wrapper<T>) List<Object>
selectBatchIds(Collection<Serializable>) List<T>
updateById(T) int
selectMapsPage(E, Wrapper<T>) E
selectList(Wrapper<T>) List<T>
deleteBatchIds(Collection<Serializable>) int
selectByMap(Map<String, Object>) List<T>
delete(Wrapper<T>) int
selectPage(E, Wrapper<T>) E
deleteById(Serializable) int
selectOne(Wrapper<T>) T
ServiceImpl<M, T>
log Log
mapperClass Class<T>
baseMapper M
entityClass Class<T>
getOne(Wrapper<T>, boolean) T
sqlSessionBatch() SqlSession
closeSqlSession(SqlSession) void
getEntityClass() Class<T>
getMap(Wrapper<T>) Map<String, Object>
saveOrUpdate(T) boolean
executeBatch(Collection<E>, BiConsumer<SqlSession, E>) boolean
retBool(Integer) boolean
saveOrUpdateBatch(Collection<T>, int) boolean
updateBatchById(Collection<T>, int) boolean
getSqlStatement(SqlMethod) String
getObj(Wrapper<T>, Function<Object, V>) V
getBaseMapper() M
currentMapperClass() Class<T>
saveBatch(Collection<T>, int) boolean
currentModelClass() Class<T>
sqlStatement(SqlMethod) String
executeBatch(Consumer<SqlSession>) boolean
executeBatch(Collection<E>, int, BiConsumer<SqlSession, E>) boolean
UserService
login(String, String) LoginResp
sendVerifyCode(String) boolean
register(RegisterReq) boolean
faceLogin(String, MultipartFile) LoginResp
addFace(long, MultipartFile, String) boolean
Service
value() String
UserServiceImpl
userMapper UserMapper
log Logger
qiniuCloudService QiniuCloudService
verifyCodeService VerifyCodeService
faceService FaceService
faceLogin(String, MultipartFile) LoginResp
register(RegisterReq) boolean
login(String, String) LoginResp
sendVerifyCode(String) boolean
addFace(long, MultipartFile, String) boolean
Slf4j
topic() String

主要数据结构如下：


NoArgsConstructor
Data
Builder
AllArgsConstructor
Serializable
User
nickname String
password String
serialVersionUID long
face String
avatar String
phoneNumber String
id Long
email String
createdAt Date

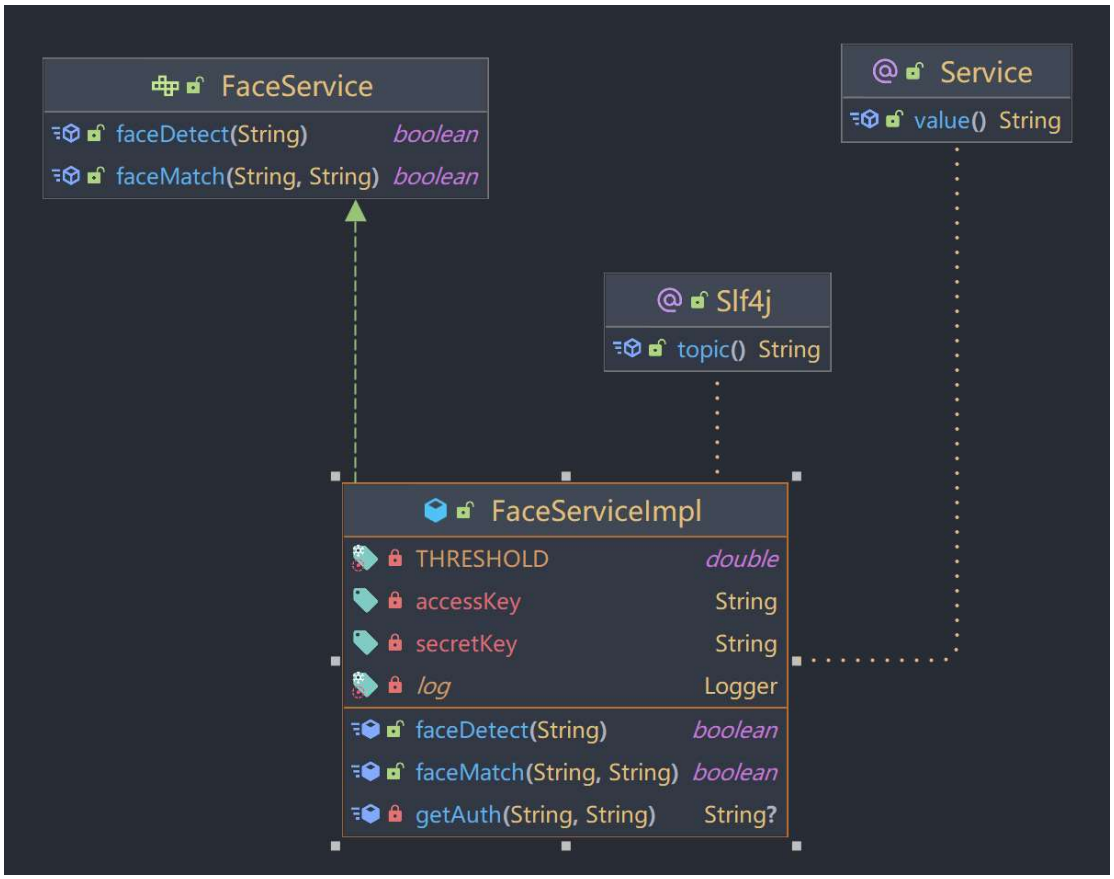

NoArgsConstructor
Data
ApiModel
Builder
Serializable
RegisterReq
password String
email String
avatar String
verifyCode String
nickname String
serialVersionUID long
AllArgsConstructor



模块中关键函数描述如下：

函数功能	函数签名	函数参数	函数返回值	函数流程描述
发送验证码	public boolean sendVerifyCode(String email);	邮箱	是否操作成功	调用验证码服务向传入邮箱发送验证码
注册	public boolean register(RegisterReq req);	注册参数，包含邮箱、昵称、密码、头像和验证码	是否操作成功	1. 校验验证码是否正确 2. 校验各项参数是否合法 3. 插入数据库并判断是否插入成功
邮箱密码登录	public LoginResp login(String email, String password);	邮箱和密码	登录返回值，包括昵称、邮箱、头像、token 和 token 过期时间	1. 查找用户 2. 校验密码是否正确 3. 用户登录，生成 token
人脸登录	public LoginResp faceLogin(String email, MultipartFile image);	邮箱和人脸照片	登录返回值，包括昵称、邮箱、头像、token 和 token 过期时间	1. 查找用户 2. 调用人脸服务校验人脸是否匹配 3. 用户登录，生成 token
添加人脸	public boolean addFace(long userId, MultipartFile image, String verifyCode);	用户 id、人脸照片和验证码	是否操作成功	1. 查找用户是否存在 2. 调用验证码服务校验验证码是否正确 3. 调用人脸服务校验人脸是否合法 4. 添加人脸并判断是否添加成功

3.3.2. 人脸模块

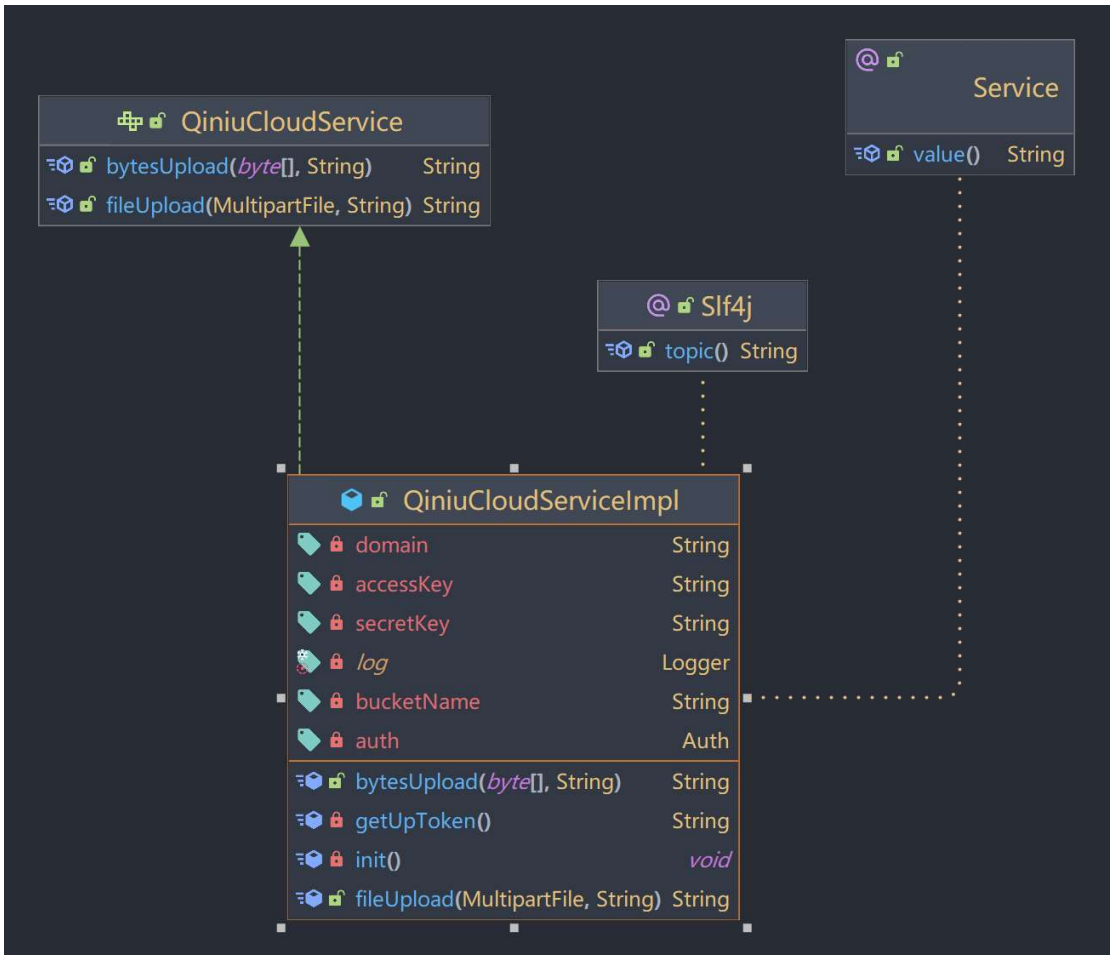


模块中关键函数描述如下：

函数功能	函数签名	函数参数	函数返回值	函数流程描述
获取鉴权 token	private void getAuth(String ak, String sk)	百度云的 access key 和 secret key	鉴权的 token	调用百度云智能鉴权 API 获取 token
人脸对比	public boolean faceMatch(String imgStr1, String imgStr2);	两张人脸照片信息	是否匹配	1. 调用 getAuth() 获取鉴权 token 2. 调用百度智能云人脸对比 API 3. 解析 API 返回结果并判断是否匹配
人脸检测	public boolean faceDetect(String imgStr);	人脸照片信息	是否成功	1. 调用 getAuth() 获取鉴权 token 2. 调用百度智能云人脸检测 API

				3. 解析 API 返回结果并判断照片中是否有且仅有一张人脸

3.3.3. 七牛云模块

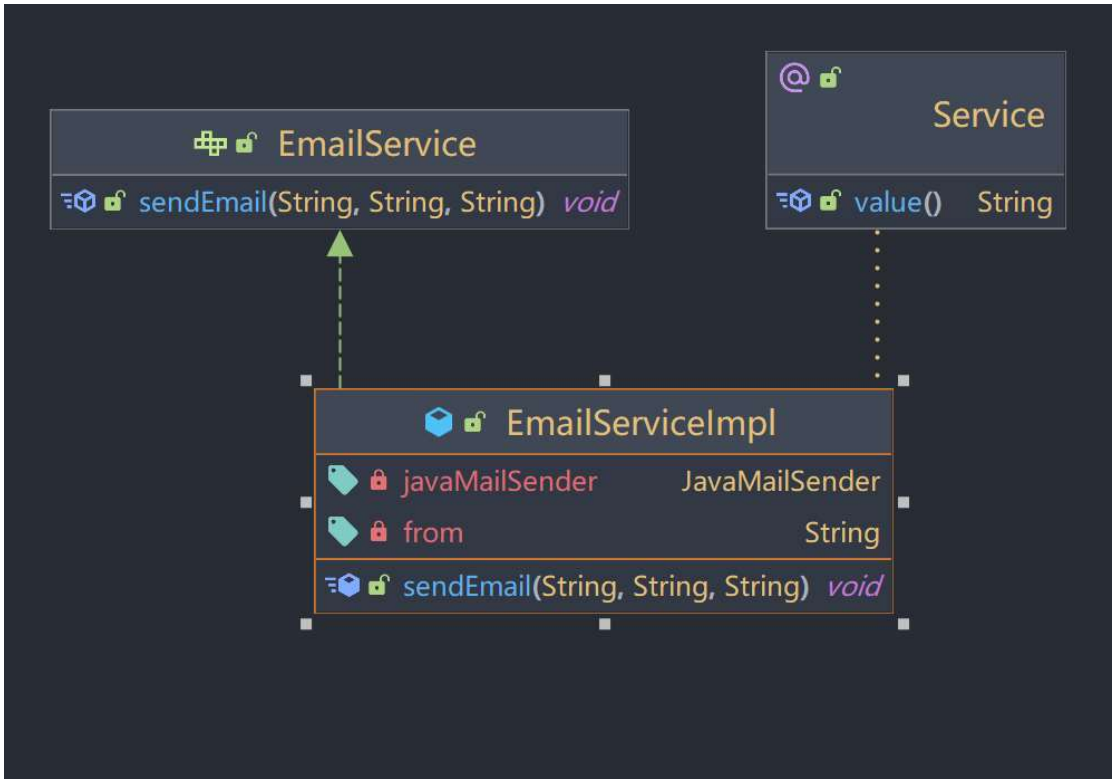


模块中关键函数描述如下：

函数功能	函数签名	函数参数	函数返回值	函数流程描述
初始化	private void init()	无	无	调用七牛云 sdk 使用 ak 和 sk 初始化鉴权信息
获取鉴权 token	private String getUpToken();	无	鉴权 token	调用七牛云 sdk，获取鉴权 token
字节数组上传	public String bytesUpload(byte[] fileBytes, String fileName);	文件字节数组和文件名	七牛云存储的 url	1. 调用 getUpToken() 获取鉴权 token 2. 组装信息，

				调用七牛云存储 API 上传文件 3. 解析返回结果，返回存储的 url
上传文件	public String fileUpload(MultipartFile file, String fileName);	文件和文件名	七牛云存储的 url	1. 将文件转换为字节数组 2. 调用并返回 bytesUpload() 结果

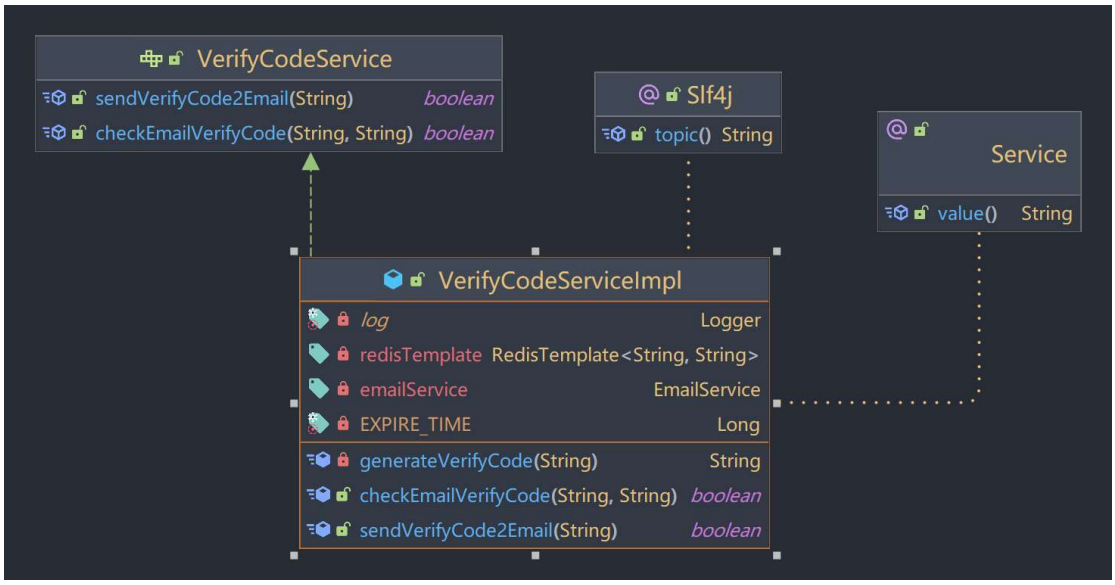
3.3.4. 邮箱模块



模块中关键函数描述如下：

函数功能	函数签名	函数参数	函数返回值	函数流程描述
发送邮件	public void sendEmail(String to, String subject, String text);	目标邮箱地址、邮件主题和邮件内容	无	1. 组装邮件信息 2. 调用 JavaMailSender 接口发送邮件

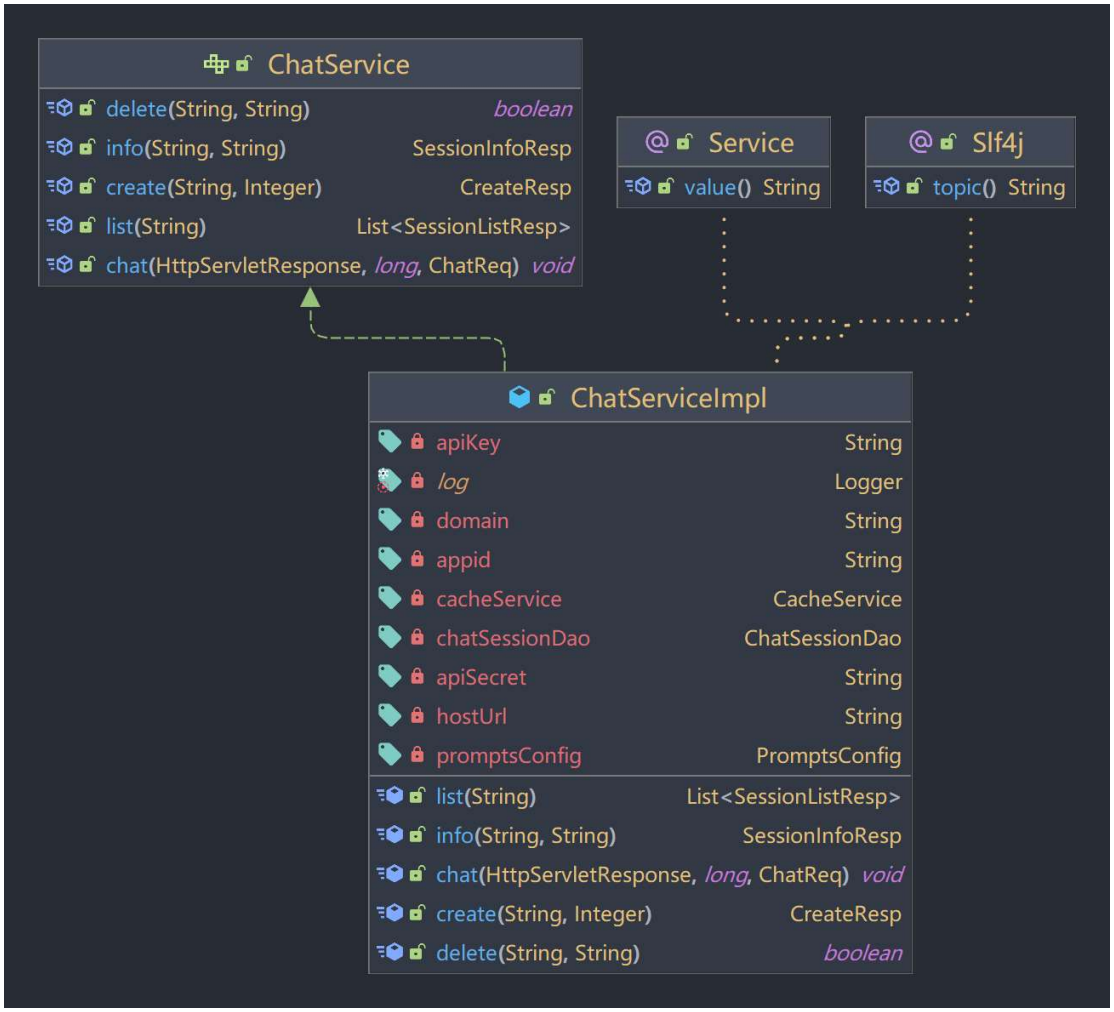
3.3.5. 验证码模块



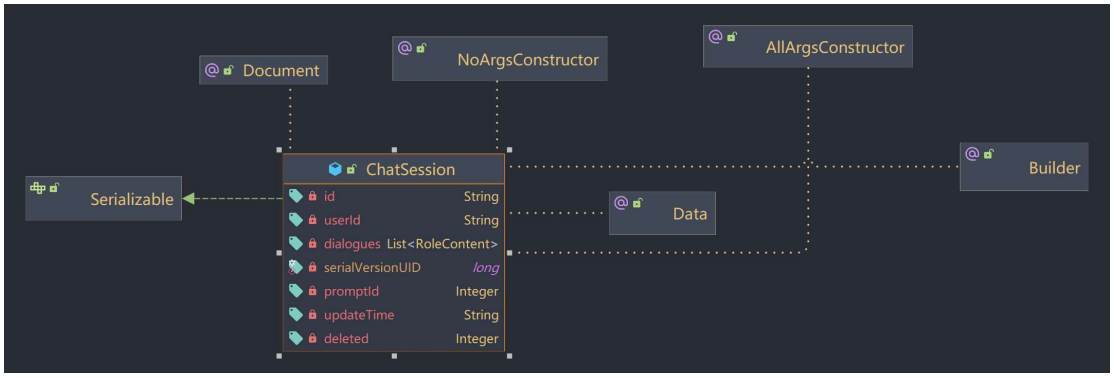
模块中关键函数描述如下：

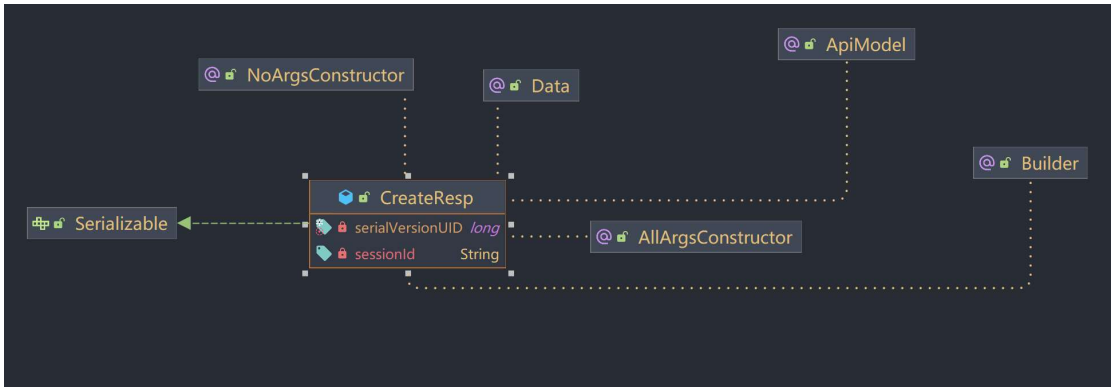
函数功能	函数签名	函数参数	函数返回值	函数流程描述
生成验证码	private String generateVerifyCode(String user);	用户信息（邮箱或者是电话等）	验证码	按照一定规则生成验证码
给邮箱发送验证码	public boolean sendVerifyCode2Email(String email);	用户邮箱	是否发送成功	1. 调用 generateVerifyCode() 生成验证码 2. 调用缓存服务将验证码写入到 redis 缓存中并设置过期时间 3. 调用邮箱服务发送验证码
校验验证码是否正确	public boolean checkEmailVerifyCode(String email, String verifyCode)	用户邮箱和验证码	是否正确	1. 调用缓存服务获取用户验证码 2. 判断是否匹配，若匹配则删除 redis 中的验证码 3. 返回结果

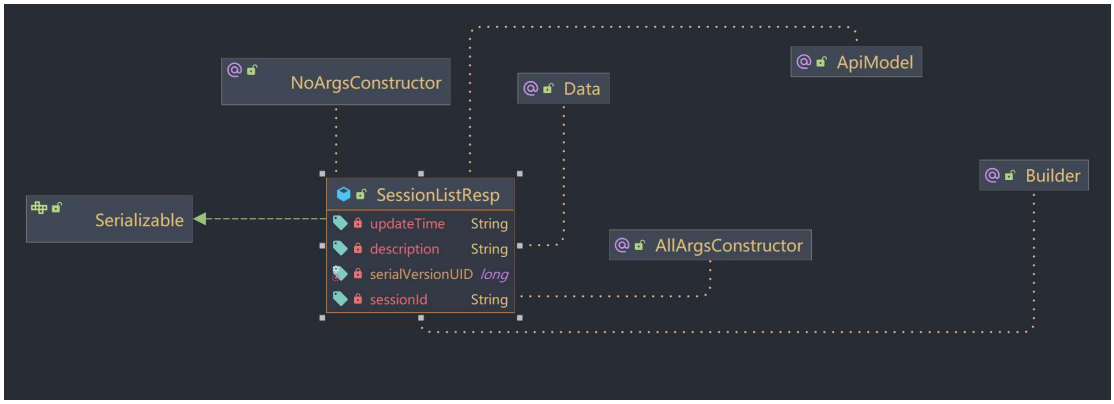
3.3.6. 对话模块

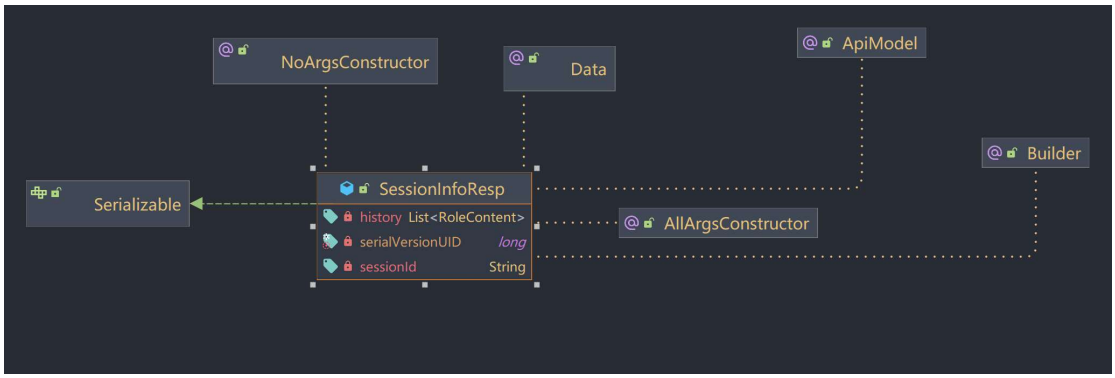

ChatService
delete(String, String) boolean
info(String, String) SessionInfoResp
create(String, Integer) CreateResp
list(String) List<SessionListResp>
chat(HttpServletResponse, long, ChatReq) void
Service
value() String
Slf4j
topic() String
ChatServiceImpl
apiKey String
log Logger
domain String
appid String
cacheService CacheService
chatSessionDao ChatSessionDao
apiSecret String
hostUrl String
promptsConfig PromptsConfig
list(String) List<SessionListResp>
info(String, String) SessionInfoResp
chat(HttpServletResponse, long, ChatReq) void
create(String, Integer) CreateResp
delete(String, String) boolean

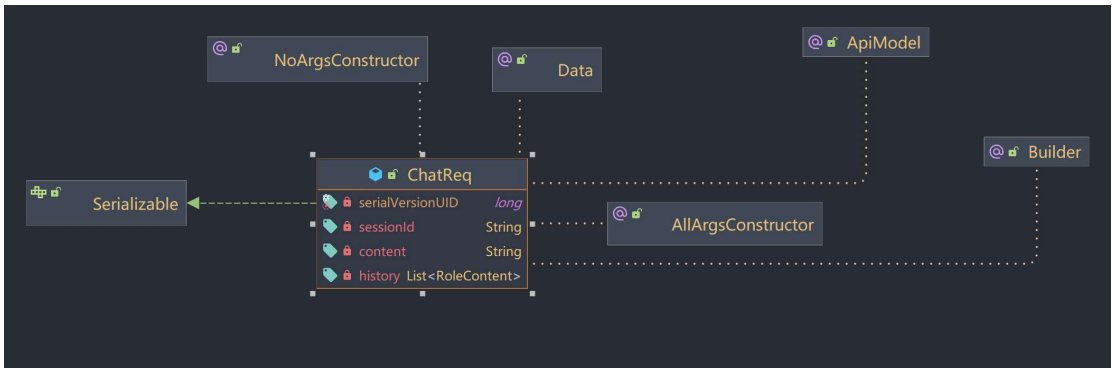
主要的数据结构如下：

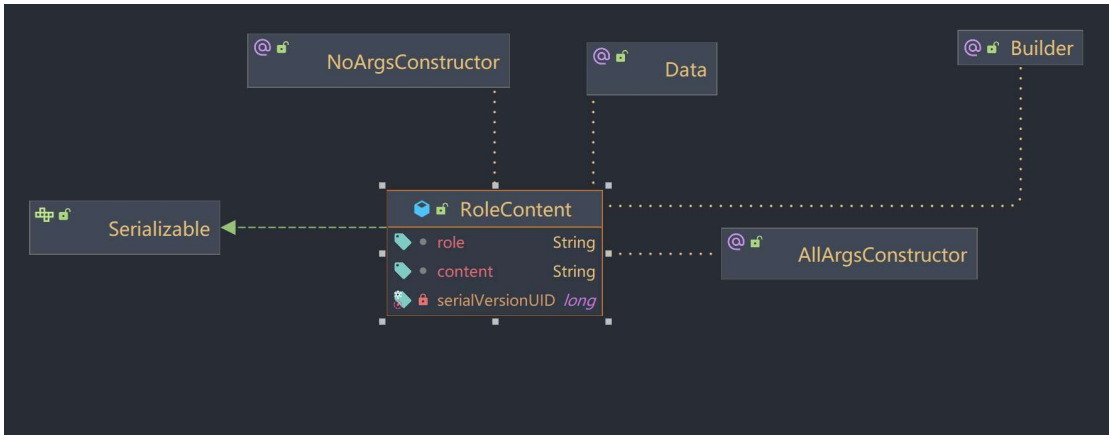

Document
NoArgsConstructor
AllArgsConstructor
Builder
Serializable
ChatSession
id String
userId String
dialogues List<RoleContent>
serialVersionUID long
promptId Integer
updateTime String
deleted Integer
Data


ApiModel
NoArgsConstructor
Data
Builder
CreateResp
Serializable
serialVersionUID long
sessionId String
AllArgsConstructor









模块中关键函数描述如下：

函数功能	函数签名	函数参数	函数返回值	函数流程描述
新增会话	public CreateResp create(String userId,	用户id和选取的角色	新建会话的id	1. 校验角色选取是

	Integer promptId);	promptId		否正确 2. 新建会话入库并返回会话id
删除会话	public boolean delete(String userId, String sessionId);	用户id和会话id	是否操作成功	1. 校验用户操作是否合法 2. 删除会话
获取会话列表	public List<SessionListResp> list(String userId);	用户id	会话列表,包括有会话id、会话描述和更新时间	1. 从数据中查找用户所有会话记录 2. 组装返回值并返回
获取对话历史详情	public SessionInfoResp info(String userId, String sessionId);	用户id和会话id	会话的记录信息,包括有会话id和历史记录,每一条历史记录包含有角色和对话内容	1. 校验用户操作是否合法 2. 查找该用户该条会话的详细记录 3. 组装返回值并返回
聊天	public void chat(HttpServletResponse response, long userId, ChatReq req);	用户id和聊天请求，聊天请求包括会话id、历史记录和此次输入	无，该接口使用sse，通过HttpServletResposne获取输出输出流直接向前端输出结果	1. 校验用户操作是否合法 2. 请求星火大模型API获取鉴权url 3. 开启新线程,发送请求并监听返回结果 4. 轮询查看此轮对话是否结束 5. 结束后

				调用缓存服务记录此轮聊天记录

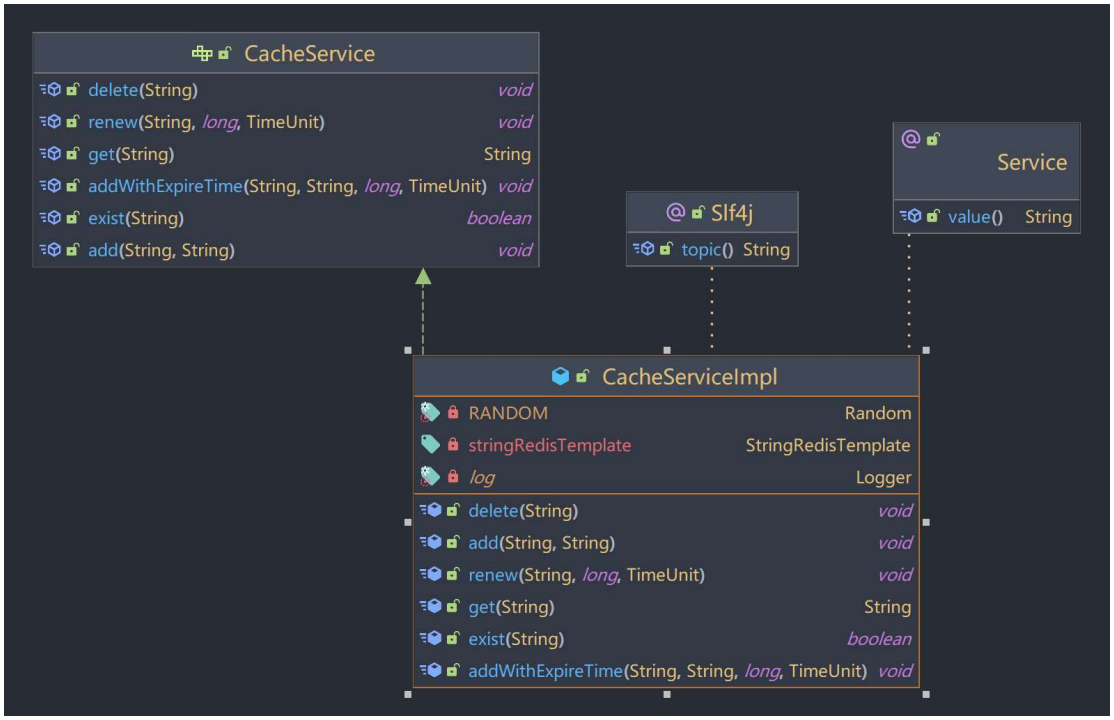
在聊天时因为涉及到了多线程操作，较为复杂，因此将聊天线程的操作详细描述如下：

1. 新线程在开启后，首先组装请求参数向星火大模型 API 发送 websocket 请求；

2. 监听 websocket 返回结果，将返回结果通过之前的 HttpServletResposne 的输出流返回给前端，并记录拼接每次答案；

3. 当此轮对话结束后，通过 redis 发送通知主线程此轮对话结束，并关闭 websocket 连接和结束此线程。

3.3.7. 缓存模块



本模块是对 StringRedisTemplate 各个接口的二次封装，完成了增删改查以及续期等各项功能，具体就不再赘述。

3.4. 项目亮点

1. 前后端分离：本项目采用前后端分离的架构进行开发，降低了系统内部的耦合性，便于开发和后去维护。

2. 模块化开发：本项目无论是客户端还是服务端均使用模块化开发的思想，具体体现为：前端将每个组件（如文本框、按钮等）均独立处理，这样增加的代码的复用性；后端将每个模块单独写成一个 service，各模块之间进行调用，切合了高内聚、低耦合的开发思想。

3. 跨平台：前端使用 Flutter 进行开发，Flutter 是一款优秀的跨平台的 UI 开发框架，本系统不仅可以在安卓系统上运行，还可以在浏览器、IOS、Windows、Linux、MacOs 多个平

台上运行，并且均有接近原生开发的性能。

4. 性能优化：本系统在开发时特地做了性能优化。前端多使用异步调用，避免线程阻塞，最大化利用资源。后端使用了缓存避免了频繁操作数据库的性能损失，在用户使用的过程中对必要的信息进行缓存，对话时数据修改均只操作缓存，采用定时任务同步缓存到数据库中，保证数据的最终一致。经过测试，每个接口的响应时间均在毫秒级。

5. 多角色对话：系统通过不同的 prompt 来提供多角色对话，用户在使用时可以选择不同的角色进行对话，提高了整个系统的可用性。

6. 对用户的隐私保护：系统前后端传输密码和人脸信息时均使用加密传输，提高隐私的安全性。在数据库的存储时，用户的所有密码也均只存储哈希值，本系统不记录任何密码原文。用户的人脸信息也是只存储提取后的特征向量，不记录任何时候用户的人脸信息。

4. 系统可能的扩展

1. 集成更多的登录方式：
   (1) 支持其他平台账号登录，如：微信登录、QQ 登录。
   (2) 支持手机验证码登录（事实上这个功能是有的，但是因为腾讯云的短信服务没充钱用不了，所以才用的邮箱登录）。
   (3) 其他生物识别方式，如：指纹识别。
2. 自定义角色：本系统预设了两个角色，后续如果需要添加角色直接在配置文件中添加即可，不需要修改代码。但是这样的方式还是有些麻烦且不自由，后续可以专门添加一个页面和接口，用户可以自定义 prompt，创建属于自己的角色。
3. 更好的无障碍体验：系统目前只支持系统的无障碍操作，没有做特别优化，后续可以实现语音输入和答案自动语音播放（同样的，星火也有这个 API，但是不免费，不充钱用不了/(ㄒ o ㄒ)/~~）。
4. 系统性能优化：目前系统使用了 Redis 和多级缓存对性能进行优化，但仍然是单机部署。后续可以实现服务的集群部署，使用网关实现负载均衡和熔断降级；Redis 分片集群；MySQL 读写分离；MongoDB 集群部署……通过这些操作来实现整套系统的高可用。

5. 总结体会

移动互联网技术是指将互联网与移动通信技术相结合，实现在移动设备上进行信息传递、数据交互和服务提供的技术体系。通过移动互联网技术，人们可以随时随地通过移动设备获取信息、进行在线交流、享受各种移动应用和服务。

在完成《移动互联网技术及应用》课程的期末作业中，我设计并开发了名为 PotChat 的多角色大模型聊天系统。通过这个过程，我对移动互联网技术有了更深入的理解和认识，并获得了一些重要的体会。

- 深入理解安卓平台

在开发过程中，我更加深入地研究了安卓操作系统的工作原理，包括其架构、组件、API 以及各种开发工具。通过实际编写代码和解决问题，我对安卓平台有了更加深刻的理解，这对于我未来的安卓开发工作是非常宝贵的。

- 掌握跨平台开发技术

虽然 Flutter 是一个跨平台的框架，但在开发 PotChat 时，我特别关注了安卓特有的功能和优化。我学习了如何使用 Flutter 与安卓系统进行交互，以及如何利用安卓的特有 API 来增强应用的功能。这种跨平台与原生开发的结合，让我能够充分利用 Flutter 的高效性，同时也保证了应用在安卓设备上的最佳表现。

- 用户界面与体验设计

在设计 PotChat 的用户界面时，我投入了大量的精力。我学习了如何使用 Flutter 创建美观、流畅且易于使用的界面。同时，我也不断地从用户的角度出发，思考如何设计出符合用户习惯和需求的功能。这个过程让我认识到了优秀的用户体验对于应用成功的重要性。

- 项目规划与管理

由于此学期大作业过多，所以为了高效地完成各门课程的大作业，我也学会了如何制定合理的开发计划，如何设置里程碑和优先级，以及如何有效地管理时间。这些项目管理的技能对于我按时完成项目并保持高质量的输出是非常关键的。

总而言之，通过这个项目，我不仅提升了自己的安卓软件开发技能，还学会了如何独立管理一个项目，如何与第三方服务进行集成，以及如何关注性能优化和用户体验。这些经验将为我未来的软件开发工作打下坚实的基础。我相信，随着我继续学习和实践，我将能够开发出更多优秀的安卓应用，满足用户的需求并为社会带来价值。