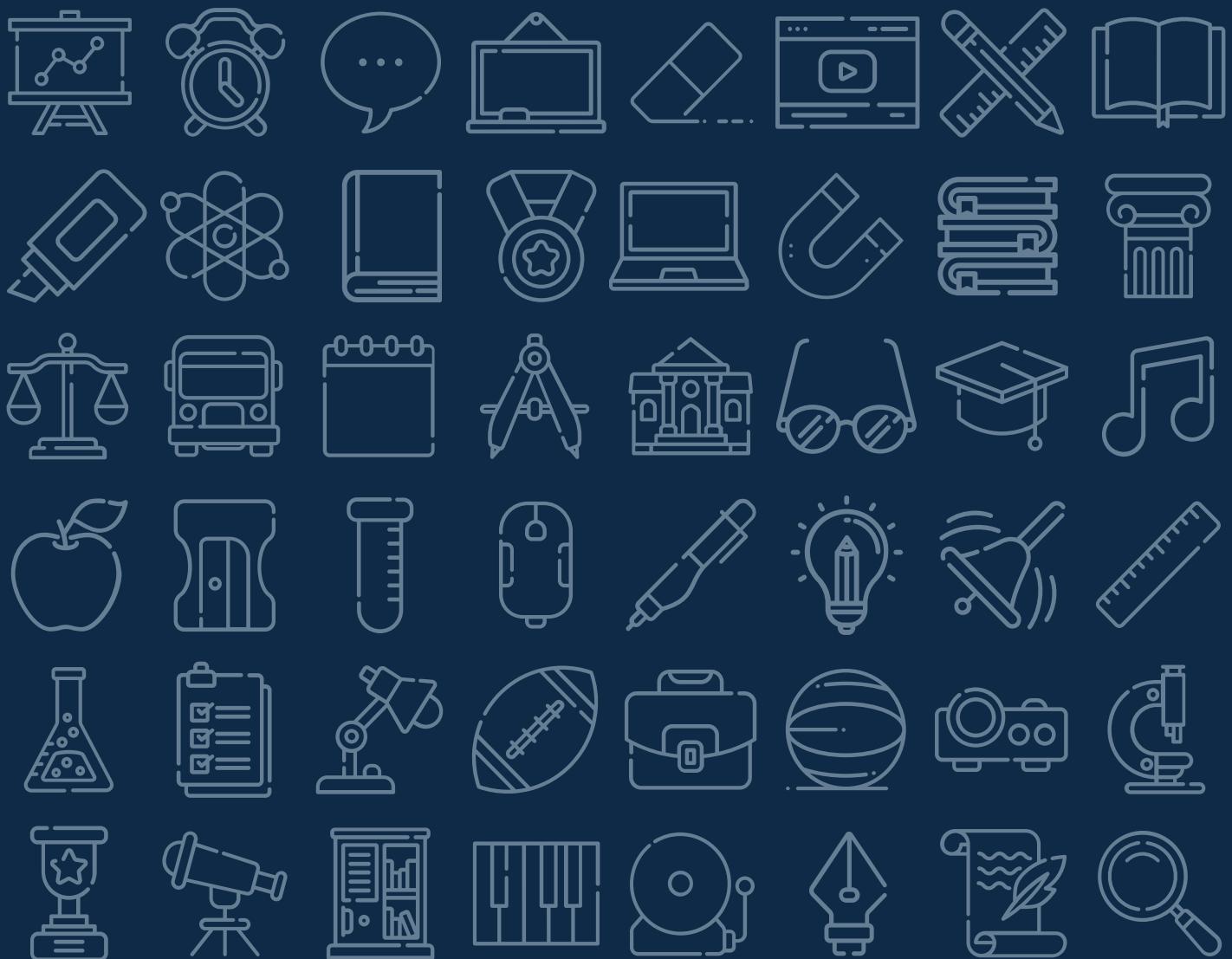


SRS Document

9min

System Analysis and Design

Spring 2022



1. 介绍

- 1.1 目的
- 1.2 项目范围
- 1.3 术语表
- 1.4 系统设计的改进
- 1.5 实现平台与框架
- 1.6 架构风格
- 1.7 设计模式
 - 单例模式
 - 职责链模式
 - 观察者模式
 - 命令模式
 - 代理模式
 - 策略模式
 - 解释器模式
- 1.8 关键的设计抉择

2. 架构完善

- 2.1 平台相关架构
- 2.2 子系统和接口
 - 用户服务系统**
 - 需求服务系统**
 - 仲裁服务系统**
 - 评价服务系统**
 - 协作服务系统**
- 2.3 接口规范(外部)
- 2.4 接口样例
 - 管理员获取用户列表
 - 用户注册
 - 用户获取订单列表
 - 用户获取推荐列表

3 设计机制

- 3.1 数据存储与访问
- 3.2 权限管理

4. 用例的实现

- 4.1 用户登录
 - 时序图
 - 类图
 - 通信图

- 4.2 发布需求
 - 时序图
 - 类图
 - 通信图

5. 产品原型进展

- 5.1 前端原型
- 5.2 后端原型
- 6 自我反馈
 - 李乐天:
 - 杨淳屹:
 - 姜文渊:
 - 杨鑫:
 - 孟宇:
- 7.Contribution

1. 介绍

1.1 目的

该文件的目的是为了说明JIYU的架构分析和分析模型。更重要的是，系统用户界面的快照随着我们工作的进展而更新，所以我们也包括更新的快照和我们所学到的参考资料。

1.2 项目范围

9min 是一个基于 Web 的学术支援平台，主要目标是为大学生及科研人员提供便利的在线解决学术问题的求助平台和一个帮助他人解决问题以赚取一定报酬的平台。用户可以通过 PC 或移动设备通过浏览器使用我们的网页端平台，或者以手机 App 的形式进行访问。这个平台中的潜在场景包括检索和查看系统推荐的他人发布的请求、浏览需求信息、发布需求、接取需求等等。用户在登录平台后可以作为访客浏览广场，也可以成为甲方发布自己的需求等待接单，或是作为乙方接取他人的需求赚取报酬。对于遇到了学术问题的用户，该平台使得他们的问题能得到有经验的同龄人的优质快速的解决，对于有能力帮助他人的用户，我们为其提供了一个很好结合自身专业知识的便利外快平台。



此外，我们的在线平台有着以下特点：

- 1.运行在网络环境下。
- 2.进行用户认证以确保安全。
- 3.拥有集中式数据库等。

1.3 术语表

中文名词	英文名词	术语解释
------	------	------

用户/访客	User/Visitor	每个用户在登录后就能够作为访客进入本平台，访客能够浏览平台提供的信息，包括已经完结的学术需求订单信息、已经发布而尚未被接取的需求信息、用户信息等
甲方/需求方	First Party	甲方是发布需求的人，可以编辑需求，和乙方一同推动需求，完成后需要支付报酬

甲方/需求方	英文名词 Second Party	术语解释 乙方是接受需求的人，在接受需求以后，需要和甲方进行沟通，完成甲方诉求，完成需求后可以得到报酬
需求/订单	Post	甲方所发布的学术相关的，需要一定帮助来解决的问题
仲裁方/管理员	Arbitrator/Admin	仲裁方是具备一定资格而被选出的特殊用户，他们可以在用户提起的仲裁中进行投票；管理员是平台的管理者，他们审核和处理平台的各项事务，有最高的权限

1.4 系统设计的改进

在之前的文档中，我们已经设计了一个9min学术支援平台的分析模型。我们在先前的部分里提供了项目的具体用户界面、架构和类图。

在之前的工作基础上，这次我们确定并进一步完善了项目的微服务架构。我们将提供具体的技术架构和逻辑架构模型，并为项目设计具体的接口，详细描述部分第三方API、子系统接口、分析机制、用例实现和系统原型设计。我们注重项目的具体设计，本次文档说明项目实施所需的主要技术，并确定项目的主要实现方式。

我的平台设计将主要有如下的改进：

1. 通过分析平台机制和功能，对逻辑结构等内容进行了重新配置。
2. 除了逻辑架构外，我们进一步增加了技术和物理架构的设计。提供了每一层所需的技术栈和在物理层面上的部署方式。

3. 增加了具体的接口设计。我们根据我们的新子系统提供了具体的接口。其中包含了更多的细节，如参数类型和返回值类型等，考虑了更细节的部分。我们还为第三方和一些子系统的接口提供了详细的说明。
4. 我们使用了API网关、数据持久化和一系列的设计机制来提高项目的安全性和可靠性、高容错性和灵活性。
5. 我们对模型中的机制进行分析，并选择合适的设计机制。
6. 在系统的分析设计过程中应用了设计模式。
7. 我们亲自实践了部分项目内容的建设，进行了一部分系统的开发工作，并提出了原型设计。
8. 将用户用例与平台的系统结构结合起来以分析系统的实现过程和方式。

1.5 实现平台与框架

在实际开发中，我们将使用微服务架构。在微服务架构中，系统中的每个微服务都可以独立部署，而且每个微服务都是松散耦合的。每个微服务只专注于一项任务，并对此进行最好的实现。这样可以做到低耦合，更灵活，有针对性地解决问题，更容易独立开发，以及能够带来高可用性和稳定性。为此，我们需要一系列成熟的技术和框架来构建系统。对于整个系统中的每一个子系统和组件，实际实现的概况大致如下：

web 应用：通过 Flutter、Material-UI、Panache、Supernova、Sylph、Codemagic 以及一些其他工具构建前端框架。使用AJAX来更新页面上的信息而不刷新页面。

API 网关：通过应用Spring Cloud Alibaba，将API网关作为一个统一的外部接口来提供微服务。Spring Cloud Alibaba包含了开发分布式应用微服务的必要组件，这样我们就可以通过Spring Cloud编程模型轻松地使用这些组件来开发分布式应用服务。

安全：通过Sa-Token进行授权认证，并基于这个轻量级的授权认证框架进行安全测试，以此满足 9min 学术支持平台系统的开发要求。

数据存储：数据存储方面我们将使用关系型数据库 MySQL，文档型数据库 MongoDB，对象存储MinIO，使用 Redis 来满足高并发情况下数据的安全性和一致性。

服务端框架：Spring boot作为一个强大的网络开发框架，具有高扩展性和出色的性能，我们将使用 Java 语言进行后端开发，并应用 Spring boot 作为后端框架。

1.6 架构风格

系统整体采用微服务架构风格，实现了设计的去中心化，提高了系统的容错性、可扩展性和可维护性。微服务架构的理念不是开发一个巨大的单体应用，而是将应用分解成小型的、相互联系的微服务。一个微服务完成一个特定的功能，这个系统将整个系统划分为五个广泛的微服务子系统部分。

由于 9min 提供的服务是相对完整和独立的，可以打包成不同的微服务集群用于其他系统，因此本系统可以设计成微服务架构系统，具有高内聚、低耦合、不同服务独立部署的特点，同时微服务开发具有弹性，能快速迭代，支持快速更新，可以满足当前系统的需求。

在微服务架构中单个应用的设计过程中，我们的分析将在下面展示。

Garlan和Shaw将软件架构风格分为五类，数据流风格、调用/返回风格、独立组件风格、虚拟机风格和仓库风格。其中：

- 数据流风格包括：批处理序列、管道/过滤器；
- 调用/返回风包括：主程序/子程序、面向对象风格、层次结构；
- 独立构件风格包括：进程通讯、事件系统；
- 虚拟机风格有：解释器、基于规则的系统；
- 仓库风格有：数据库系统、超文本系统、黑板系统。

我们的系统主要采用调用/返回的方式来设计整体系统。在整体架构方面，基于数据抽象和面向对象的架构，层次结构方面，由小到大进行设计。

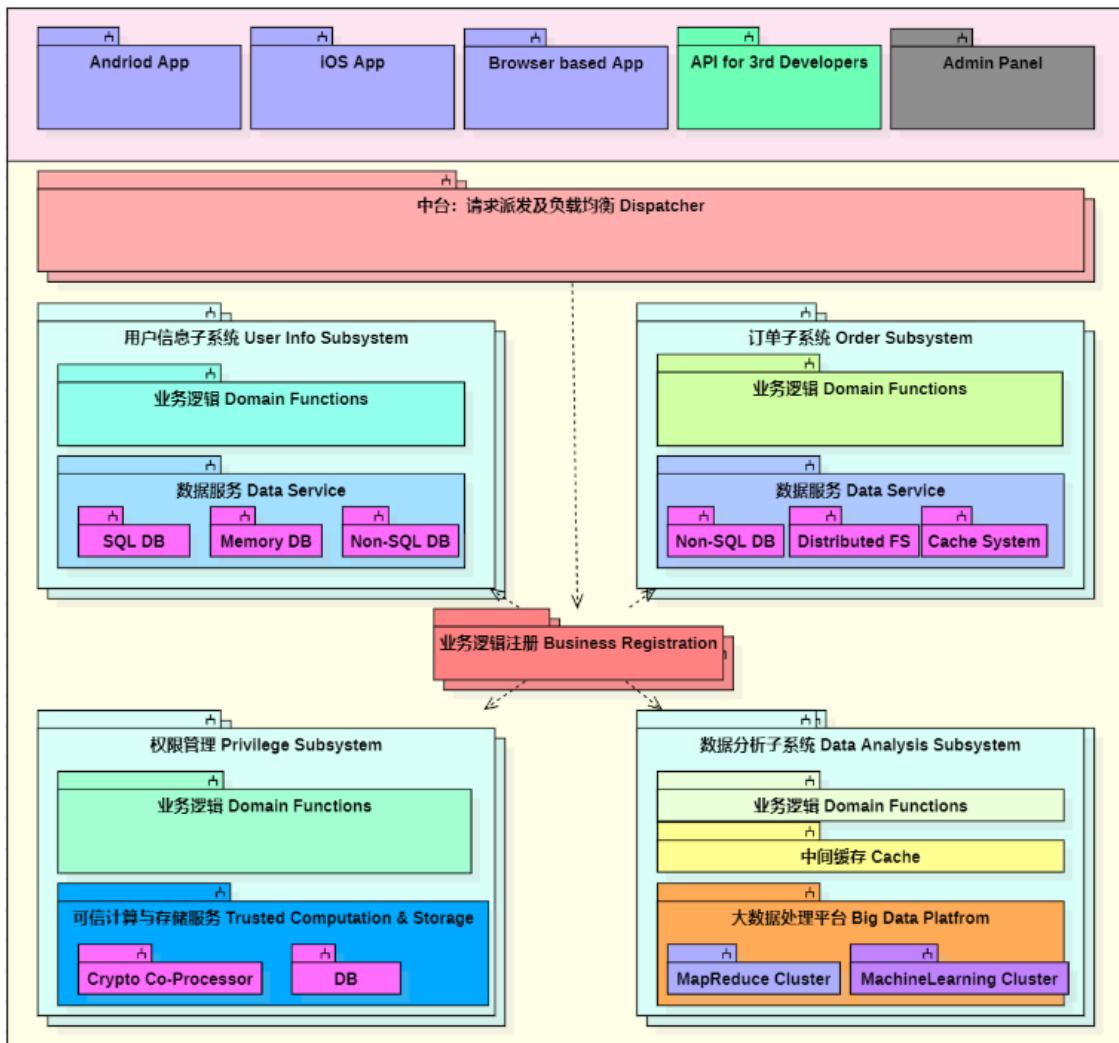
1. 数据抽象和面向对象架构

数据抽象和面向对象架构风格的组成部分是对象，对象是抽象数据类型的实例。在抽象数据类型中，数据的表示和相应的操作被封装起来。对象的行为体现在它的接受和请求动作中。连接器是对象之间互动的方式。对象通过函数和程序的调用进行交互。对象是被封

装的，一个对象的变化不会影响其他对象。对象有状态和操作，也负责维护状态。这种结构风格包括封装、交互、多态性、集成和重用等特征。为了实现低耦合、高度可重用和可维护的系统，易于升级和保护内部完整性，9min 平台使用IOC（反转控制）来构建容器，以实现对象之间的解耦和AOP(Tangent Oriented Programming)来提取被封装为公共服务的公共行为，以减少系统的冗余和模块间的耦合，并提高系统的可操作性和可维护性。

2. 层次结构

分层系统被组织成一个分不同层级的结构。组件在一些层中实现了一个虚拟机。连接器是由决定各层如何互动的协议来定义的。拓扑约束包括对相邻层之间互动的约束。每一层都为上一层提供服务，当使用下一层的服务时，你只能看到与之相邻的一层。大问题被分解成许多渐进的小问题，这些问题被一步步解决，隐藏了很多复杂性。当一个层被修改时，最多影响两个层，通常只有上层能受到影响。上层必须知道下层的身份，不能调整层与层之间的顺序。在本系统以前的设计中，我们采用了如图所示的层次结构。前端部分对应传统三层架构的展示层，而后端架构则是基于微服务的架构。每个微服务都拥有自己的架构，各自通过内网进行通信。其中间引入了中间派发和负载均衡，在底层进行子系统的划分，通过业务逻辑注册控制进行解耦。在本文档涉及的内容中，我们基本保持了这一架构设计思想，并进行了一定的细节完善和改进。



1.7 设计模式

我们的平台开发过程将涉及如下的设计模式：

单例模式

在各个子系统的控制类中，我们将应用单例模式，因为在整个系统中，各个部分的控制器实例都是有一个且仅有一个的。出于线程安全和降低启动延迟的考量，我们将使用带锁的懒汉模式，将控制器的实例化延迟到控制方法调用时进行。

以登录控制为例，在 LoginManager 中保存 instance 成员指针指向自身类型对象，并在初始化时暂时指向 null，并将控制类的构造方法私有化。在调用 public static 方法 getInstance 时，若 instance 指向 null，则实例化控制类并返回，否则直接返回。如此，LoginManager().getInstance 将总是能够返回全局唯一的登录控制器实例。此外，注意到初次访问 getInstance 时同时进入的可能有

超过一个线程，故需要使用两层判断，其中使用 synchronized 锁保证只有一个线程进入 instance 的实例化。

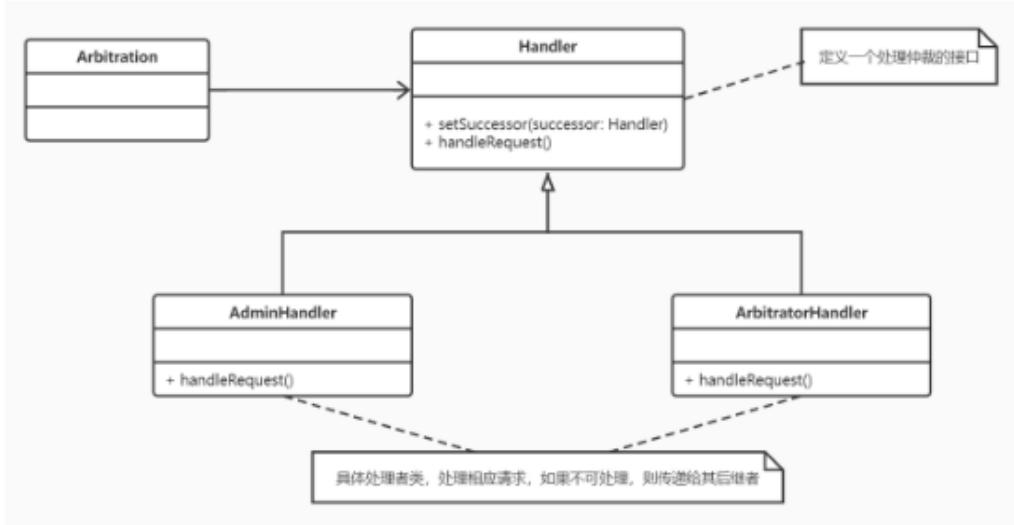
API 实例描述

API	LoginManager().getInstance
描述	获取登录控制器实例
参数	无
返回值	LoginManager.instance 对象

职责链模式

在简单处理流程中，通过职责链模式使得多个处理逻辑不同的对象能够处理请求，将这些对象连成一条链，并沿该链式结构传递请求，进行请求的处理。

在本平台中，以仲裁子系统为例，仲裁者与管理员均会对仲裁请求实例进行处理，而处理逻辑有所不同。通过引入 Handler 类定义处理请示结构，在管理员与仲裁者中设置继承 Handler 类的具体请求处理类的实例来分别以各自的逻辑对请求进行处理，而避免请求的发送者与处理者发生耦合。



观察者模式

我们在平台系统内定义某些对象间的一种一对多的依赖关系，当一个对象的状态发生变化时，所有依赖于它的对象都得到通知并被自动更新，以此来保证涉及到的对象的高度协作。

在我们的平台中，当订单状态发生变化时，该订单实例中所涉及的各个对象（包括甲方、乙方等），都应该收到相应的通知和进行更新。我们对订单进行抽象，在抽象类中为订单对象设置观察者的 List，建立 Observer 抽象类和 Subject 类，Subject 对象带有绑定观察者到订单对象和从订单对象解绑观察者的方法，在订单的观察者 List 中存放 Observer 的扩展类实例对象，使得这些扩展实例都能随着订单的更新而更新。

命令模式

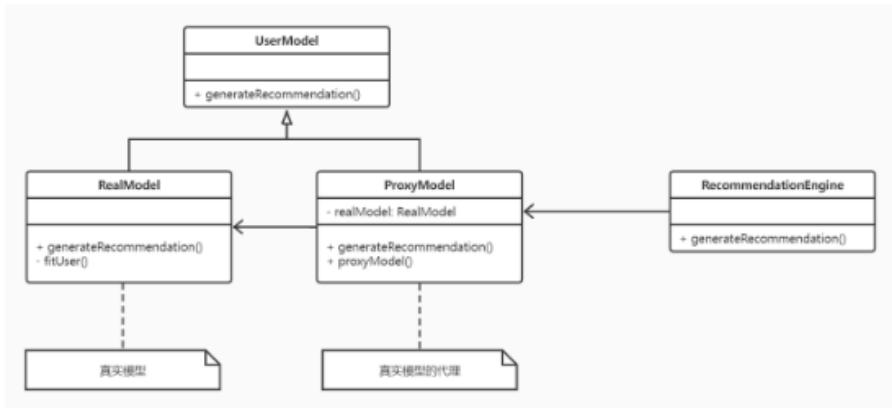
命令模式将一个请求封装为一个对象，从而使得系统可用不同的请求对客户进行参数化；可以对请求排队或记录请求日志，以及支持可撤销的操作。

在我们的平台用户搜索子系统中，用户搜索行为涉及到对行为进行记录、撤销或重做、事务等处理，因此搜索的请求者和搜索的处理者间产生耦合是不合适的，因此引入命令模式，将“搜索”这一组行为抽象出来成为 Query 类，用户在进行搜索时生成一个 Query 实例交由系统内处理。

代理模式

在代理模式中，我们为其他对象提供一种代理以控制程序其他部分对这个对象的访问。在某些情况下，一个对象不适合或者不能直接引用另一个对象，而代理对象可以在客户端和目标对象之间起到中介的作用。

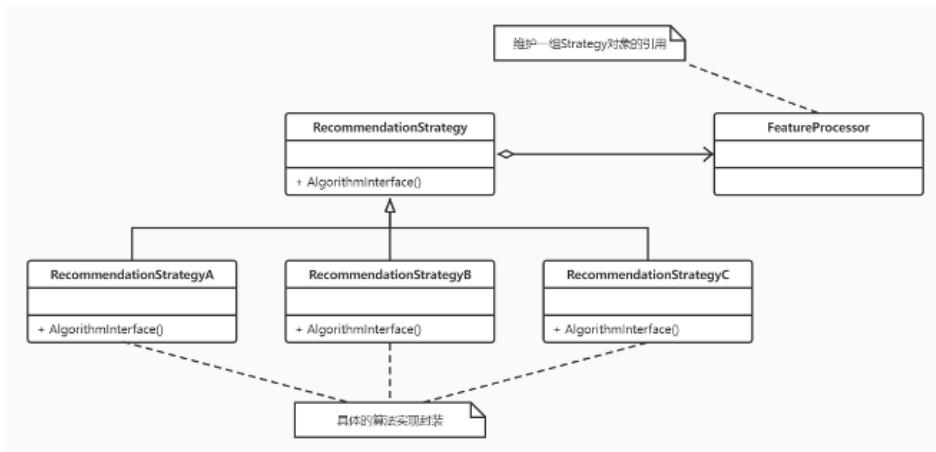
在我们平台的检索推荐系统中，对用户进行首页内容推荐和检索结果推荐的数据通常来自深度神经网络模块的计算结果，这些模块部署与整个系统相对独立，直接创建的开销可能较大，且在调用时通常需要对返回值进行一定的处理，因此我们应用代理模式，设置 FeatureProcessor(ProxyModel) 类作为 RecommendationEngine 访问 UserModel 时的代理，使得访问职责清晰，带来了扩展性和智能化。



策略模式

在策略模式中，一个类的行为或其算法可以在运行时更改。这种类型的设计模式属于行为型模式。在策略模式中，我们创建表示各种策略的对象和一个行为随着策略对象改变而改变的 context 对象。策略对象改变 context 对象的执行算法。

在本平台的检索与推荐系统中，对搜索结果进行分类筛选检索推荐的算法可能根据情况在运行时有不同的选择，因此我们应用策略模式，设置 RecommendationStrategy 抽象类引入一组接口，将不同算法继承该类并封装实现，从而让推荐引擎在运行时动态选择推荐算法策略。



解释器模式

当用户浏览首页推荐订单时，他们往往不知道最近有多少订单被发起。因此在这种情况下，我们根据用户的喜好来设置当前订单的顺序，只要在用户查看时为其提供下一个订单就可以了。在迭代器设计模式中，提供了一种方法来按顺序访问聚合对象中的元素，而不暴露对象的内部表示。所以我们抽象出：SuggestedOrder接口、Orders接口和OrderName类。OrderName类存储推荐的浏览顺序。

1.8 关键的设计抉择

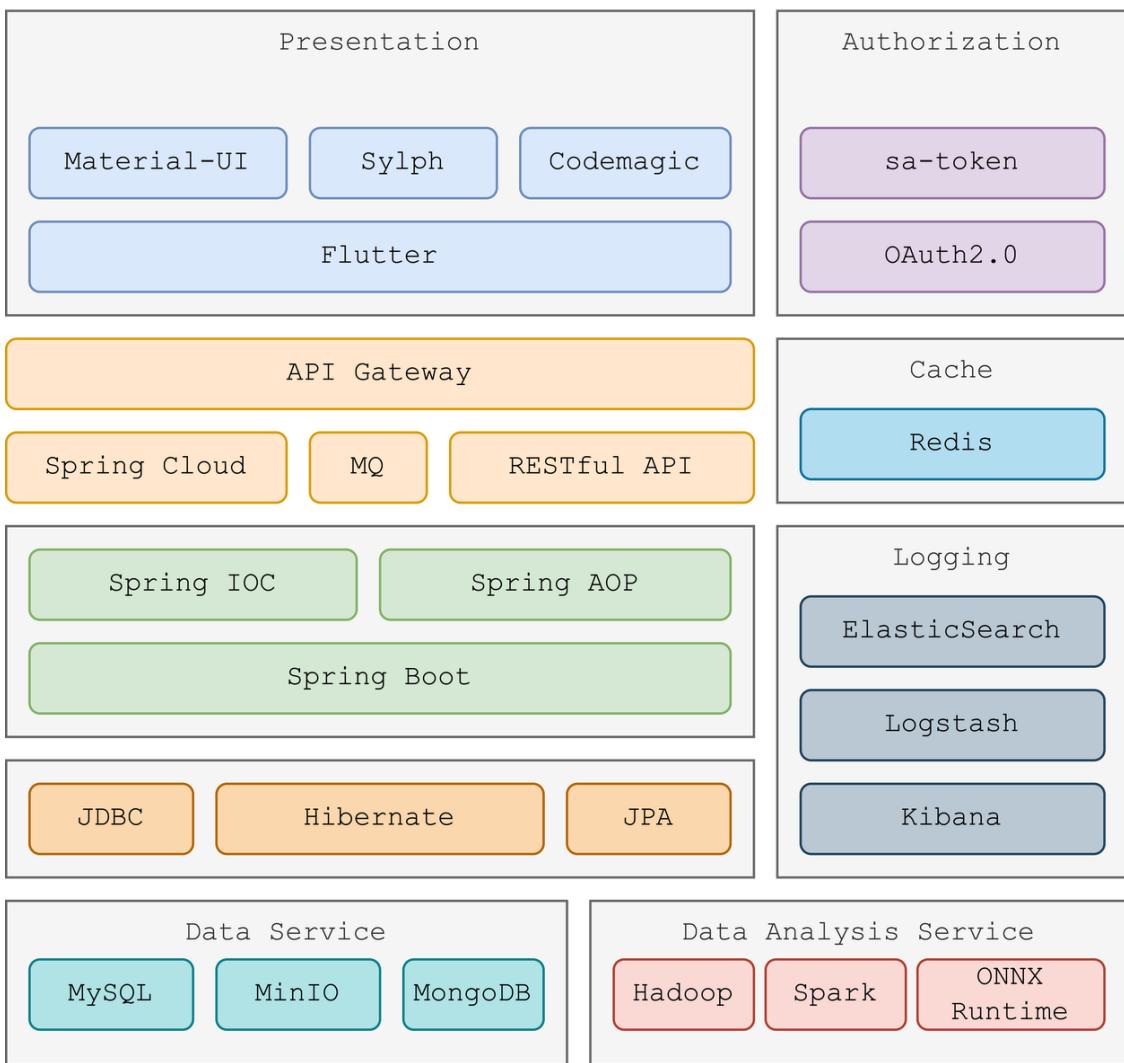
1. 使用Flutter和MaterialUI构建我们的UI：Flutter作为新一代的跨平台前端框架，支持以原生的方式在各平台上进行渲染；这样在提高了前端开发的效率的同时，也使得用户具有更加流畅的体验。MaterialUI的引入使得前端设计更加规范合理。
2. 使用微服务架构：微服务架构具有较大的弹性，可以方便地与敏捷开发过程相结合，在重构和持续集成时具有传统架构不可比拟的优势。此外，微服务架构易于部署和扩展的特性使得我们在运维的过程中可以利用云平台有效控制成本。
3. 尊重用户隐私：我们在构建系统的各个部分时都将保护用户隐私作为首要任务，除了基于sa-token的权限控制外，我们也考虑使用数据库层面的加密和磁盘级别的硬件加密，以充分保护用户数据。此外，我们在进行数据采集和分析的过程中会使用已经脱敏后的数据，以保障用户的隐私安全。
4. 以OOP风格为基础，兼用其它领域特定语言：我们的项目的大部逻辑是使用Spring技术栈开发的，但在一些专用的服务上（例如数据库，数据分析等），得益于微服务框架，我们也采用对应领域的特定技术栈，从而提高开发效率。

2. 架构完善

2.1 平台相关架构

基于我们前文的分析，我们为我们架构使用的技术栈进行了选择和细化，大致如下图所示：

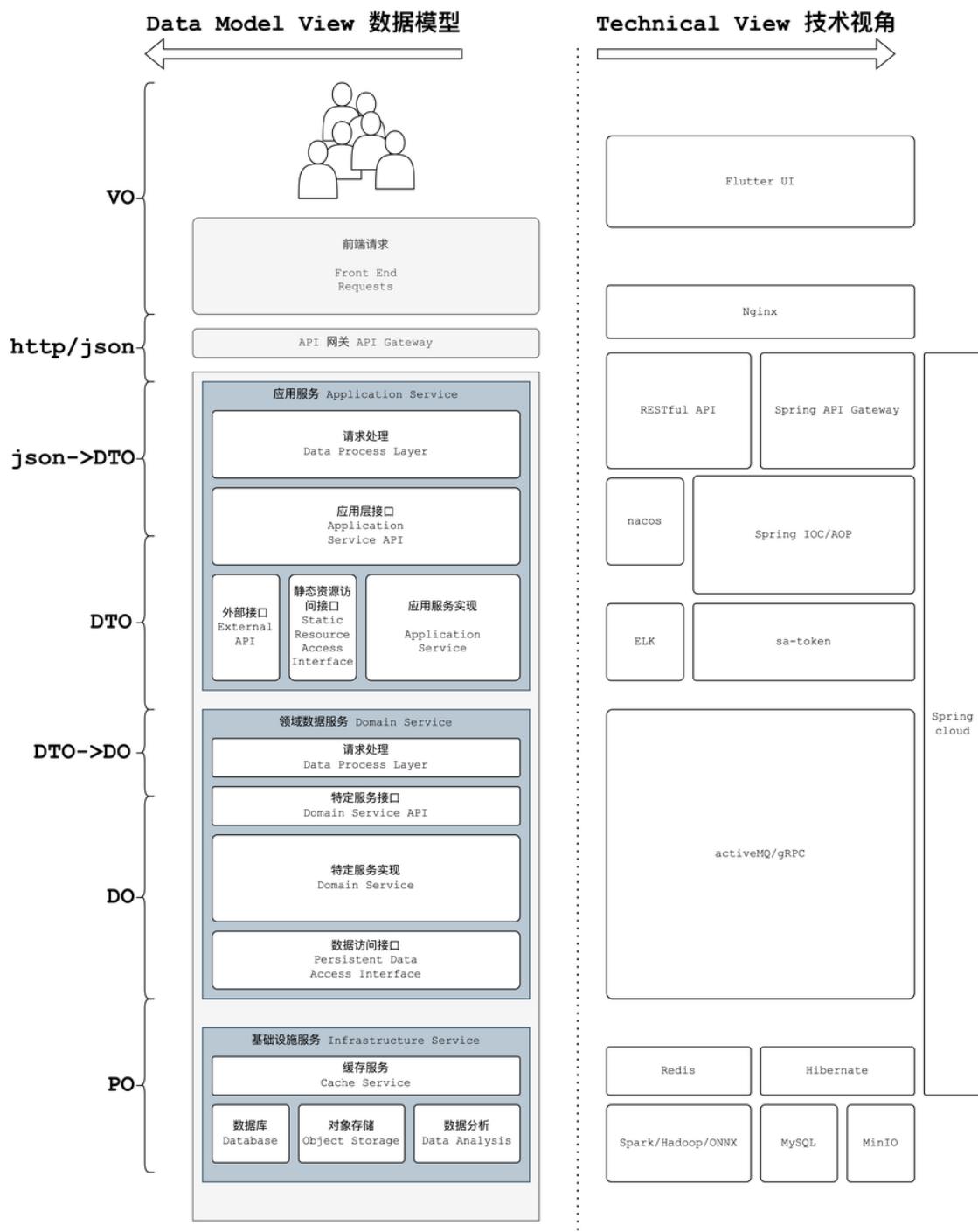
9min Platform-dependent Architecture Overview



我们仍然使用分离的前端和后端开发，前端使用以Flutter为中心的技术栈进行开发，后端一般使用 SpringBoot 和 SpringCloud 进行开发。前端和后端通过 REST 风格的 API 调用和 json 数据文件进行通信，各层使用不同的数据模型（例如，VO、DTO等）。系统通过sa-token提供授权和认证，通过ELK完成日志采集和数据分析可视化，并为了有效解决异步消息和流量裁剪，activeMQ消息队列用于消息传递和存储。后端与关系数据库的数据交换通过 Hibernate 的持久化机制进行数据库，并使用 Redis 进行缓存；一些较大的多媒体资源则使用 MinIO 进行对象存储。

此外，为了分析用户习惯，进行订单的精准推送，我们使用 Hadoop 和 Spark 为大数据技术栈构建统计模型，对于用于自动监测异常内容等功能的更加复杂的机器学习模型，我们则使用 ONNX Runtime 进行部署。

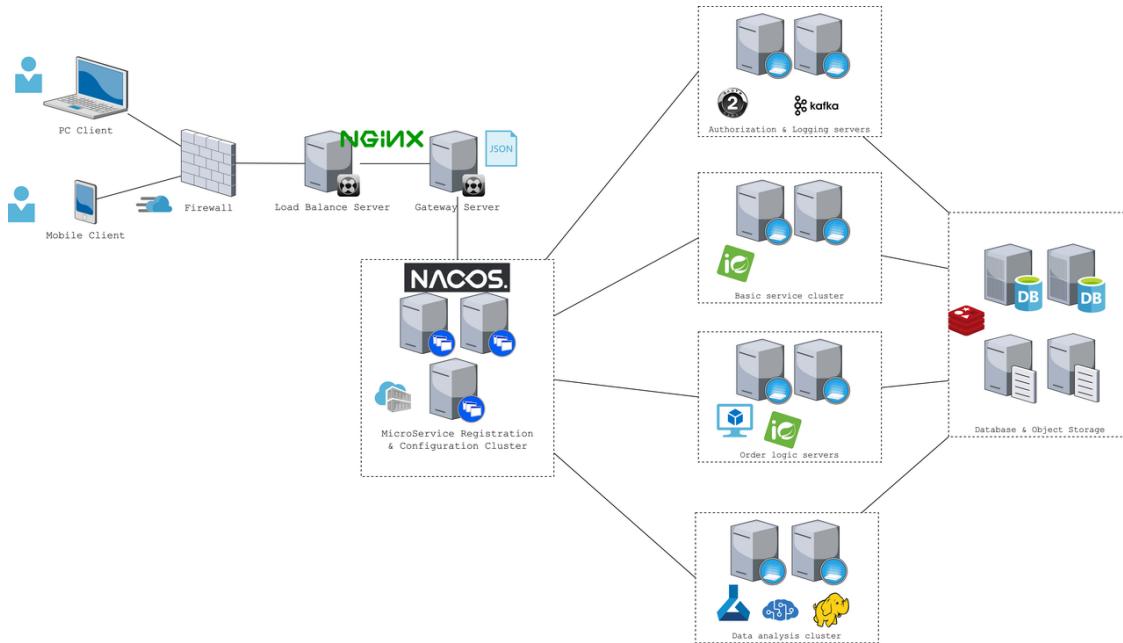
从数据流动和技术栈的视角重新审视我们的架构，如下图所示：



图示中，数据的流动与技术栈的选择相对应，各种平台相关的服务器服务于各层次的不同组织方式的数据，从而实现整个系统的高效与安全的运作。

为了实现逻辑架构和物理部署架构之间的对应，系统的物理实现通过一系列的“服务器”实现（不仅仅只常规的机架服务器，也可以是虚拟机和云服务器）。基于微服务架构的特点，一个服务集群注册中心和配置中心服务器用于管理四个不同功能的服务器集群用于实现微服务之间的通信。此外，防火墙和负载均衡服务器也被引入以

保障系统的可用性。数据持久化部分，各数据库和缓存服务被尽可能部署在较为集中的环境下（例如低功耗的国产arm存储服务器），在方便维护的同时也能降低运维成本。具体的部署方案大致如下图所示：



2.2 子系统和接口

根据领域驱动设计的理念，为了更加合理的设计相应的领域服务，我们将系统涉及的业务边界按一定逻辑划分为以下五个领域：

- 用户领域
- 需求领域
- 协作领域
- 评分领域
- 仲裁领域

基于上述领域设计，为了在系统设计过程中使得服务的粒度比较适中而不至于太冗杂，我们将微服务根据功能进一步划分为以下五个子系统：

- 用户服务系统
- 需求服务系统
- 仲裁服务系统
- 评价服务系统

- 协作服务系统

用户服务系统

A	B	C
序号	REST API	接口解释
1 1	POST /api/user/	该接口实现用户账号注册功能
3 2	GET /api/user/?userID=id & userPassword = password/	该接口实现用户登录系统的功能
4 3	PUT /api/user/?userID=id/	用户通过该接口发起修改个人信息的请求
5 4	GET /api/users/	管理员通过该接口获取所有用户信息，返回包含所有用户及相关信息的列表
6 5	DELETE /api/user/?userID=id/	管理员通过该接口实现注销用户的功能，返回是否注销成功的信息

需求服务系统

A	B	C
序号	REST API	接口解释
2 6	POST api/order/orderID/	创建新需求的接口，返回是否创建成功的信息
3 7	PUT api/order/orderID/	用户通过该接口修改需求信息，返回是否修改成功的信息
4 8	DELETE api/order/orderID/	用户请求删除某需求的接口，返回是否删除成功的信息
5 9	GET api/orders/?userID=id/	管理员通过该接口获取用户所有需求信息，返回包括所有需求及相关信息的列表
6 10	GET api/order/orderID/	该接口用于获取某订单的具体信息
7 11	GET api/recommandations/?userID=id/	该接口用于根据用户id发送推荐需求的功能，返回一个对应的推荐需求列表

仲裁服务系统

A	B	C
序号	REST API	接口解释
2 12	GET api/judge/orderID/	该接口获取某订单的仲裁结果，返回仲裁的相关信息
3 13	POST api/judge/orderID/	该接口用于发起某订单的仲裁申请，返回是否发起成功的信息
4 14	PUT api/judge/orderID/	管理员通过该接口对仲裁申请做出决策，并修改相应的仲裁信息
5 15	GET api/judges/	该接口用于获取所有的仲裁信息，返回包含仲裁的订单及相关信息
6 16	DELETE api/judge/orderID/	用户通过该接口以撤销某订单的仲裁申请，返回是否撤销成功的信息

评价服务系统

A	B	C
序号	REST API	接口解释
17	GET api/comment/orderID/	该接口用于获取某订单的所有评价，返回相应的评价信息列表
18	POST api/comment/orderID/commentID/	该接口用于向某订单发送评价，返回是否评价成功的信息
19	DELETE api/comment/orderID/commentID/	该接口用于删除某订单下面的一条评论，返回是否删除成功的信息
20	PUT api/comment/orderID/commentID/	该接口用于修改某订单下面的一条评论，返回是否修改成功的信息
21	GET api/comment/orderID/commentID/	该接口用于获取某订单下面的一条评论的信息，返回其具体信息
22	POST api/comment/orderID/commentID/sub-commentID/	该接口用于为一条评论添加评论，返回是否添加成功的信息
23	POST api/comment/orderID/commentID/like/	该接口用于实现用户对某评论的点赞功能，返回是否点赞成功的信息
24	DELETE api/comment/orderID/commentID/like/	该接口用于实现用户对某评论取消点赞的功能，返回是否取消成功的信息

协作服务系统

A	B	C
序号	REST API	接口解释
25	GET api/order/orderID/	该接口用于获取某订单的全部协作信息，结果以列表形式返回
26	POST api/order/orderID/push/	该接口用于用户向某次进行中的订单发起推进请求，返回是否发起成功的信息
27	DELETE api/order/orderID/push/	该接口用于用户向某次进行中的订单撤销推进请求，返回是否撤销成功的信息
28	POST api/order/orderID/chat/	该接口用于用户向需求另一方发起聊天请求
29	POST api/order/orderID/files/	该接口用于用户在需求进行中上传文件等，返回是否上传成功的信息
30	POST api/order/orderID/finish/	该接口用于甲方发起完结需求请求，返回请求是否成功的信息
31	POST api/order/orderID/end/	该接口用于双方发起终止需求请求
32	POST api/order/orderID/pay/	该接口用于甲方发起支付酬金的请求，返回是否请求成功的信息

2.3 接口规范(外部)

为了提高开发效率，降低开发成本，在一些特定的功能上，我们使用了一系列第三方提供的 API 以构建我们的服务。在选取外部的 API 时，我们主要考量以下几个方面：

1. 综合成本：使用第三方 API 的综合成本不应当高于自行开发；
2. 服务的可靠性与稳定性：第三方 API 的接口功能稳定，服务可用性稳定；
3. 回退计划：如果第三方 API 临时不可用，应当有相应的回退计划；
4. 引入竞品：不同服务提供商的相似功能的第三方 API 应当引入至少两个，以增加系统的可靠性；

基于上面的分析，我们在如下领域引入了第三方 API，并对其进行包装以方便复用。

1. 支付与退款：阿里，微信等提供的聚合支付平台
2. 内容自动审核：阿里云、腾讯云等提供的智能审核（Intelligent Auditing）
3. 第三方平台登录支持：微信，QQ等平台的接口
4. 二次验证：短信接口，Google 2FA等
5. 视频会议接口：zoom，VooV Meeting等

下面以下单支付接口为例，详细介绍其包装方式。

请求样例

```
1 {
2   "merchant_order_no": "1234235365663",
3   "pay_request_no": "232425366",
4   "payer": {
5     "type": "USER_ID",
6     "issuer": "ALIPAY",
7     "identity": "2088102146225135"
8 }
```

返回样例

```
1  {
2      "alipay_business_paymenthub_pay_response": {
3          "code": "10000",
4          "msg": "Success",
5          "pay_request_no": "232425366",
6          "payment_id":
7              "2020021601502300700001090000026033",
8          "channel": "ICBC_BANK_TRANSFER",
9          "pay_amount": 100,
10         "pay_status": "WAIT"
11     },
12     "sign": "ERITJKEIJKJHKKKKKKHJEREEEEEEEEEE"
13 }
```

异常样例

```
1  {
2      "alipay_business_paymenthub_pay_response": {
3          "code": "20000",
4          "msg": "Service Currently Unavailable",
5          "sub_code": "isp.unknow-error",
6          "sub_msg": "系统繁忙"
7      },
8      "sign": "ERITJKEIJKJHKKKKKKHJEREEEEEEEEEE"
9  }
```

从上面的样例中可以看出，我们将一些API的复杂参数留给后端处理，暴露给我们应用的API遵循最简的原则；此外，API的安全性除了由授权系统保障外，签名等技术也被应用其中。API的返回异常也遵循极简的原则，以尽可能少地暴露内部信息。

同时我们也向第三方应用提供一些接口，这些RESTful接口的通用调用参数如下：

A	B	C	D
字段	必选	类型	说明
2 type	FALSE	string (json/xml/jsonp)	返回数据方式 默认json
3 appkey	TRUE	string	应用appkey
4 ts	TRUE	int	当前时间戳 (UNIX TIMESTAMP)
5 sign	TRUE	string	应用校验密匙
6 callback	FALSE	string	JSONP调用方式时回调函数名称
7 access_key	FALSE	string	应用在用户申请登陆后获取到的access_key 可以此access_key访问需要用户权限的操作
8 platform	FALSE	string (android/ios)	客户端平台适配及统计用

一些通用的返回值如下表：

A	B
代码	说明
2 -1	应用程序不存在或已被封禁
3 -2	Access key错误
4 -3	API校验密匙错误
5 -101	帐号未登陆
6 -102	帐号被封停
7 -105	验证码错误
8 -106	帐号未激活
9 -108	应用沒有存取相应功能的权限
10 -400	请求有误
11 -403	权限不足
12 -404	文档不存在
13 -500	服务器内部错误
14 -503	调用速度过快

调用这些 API 时，需要进行签名，签名方式如下：

把接口所需所有参数拼接，如utk=xx&time=xx，按参数名称排序，最后再拼接上密钥App-Secret，做md5加密 (callback不需要参与sign校检)

除了常规的登录、搜索等功能外，我们提供的一些常用api列表如下：

A	B	C
序号	REST API	接口解释
2 1	GET /eapi/order/recommend	返回推荐给该用户的订单内容
3 2	GET /eapi/user/recommend	返回用户可能感兴趣的用户信息
4 3	GET /eapi/user/fans	返回关注该用户的用户信息
5 4	GET /eapi/user/attention	返回用户的关注信息
6 5	GET /eapi/index	获取当前服务状态
7 6	GET /eapi/trend	获取当前各订单的热度等统计信息
8 7	GET /eapi/tag/ranking	获取某标签下订单热度排名

后续计划逐步放开 API 并提供接入的 SDK，从而借助开源社区发展我们的项目。

2.4 接口样例

我们用管理员获取用户列表、用户注册、用户获取订单列表、用户获取推荐列表四个接口为例，展示我们具体的接口设计与实现。

管理员获取用户列表

该请求为get请求，REST API为api/nine-min/users，无参数，返回的JSON如下：

```
1  [
2      {
3          "id": "id",
4          "name": "李乐天",
5          "gender": "男"
6      },
7      {
8          "id": "id",
9          "name": "姜文渊",
10         "gender": "男"
11     },
12     {
13         "id": "id",
14         "name": "杨鑫",
15         "gender": "男"
16     },
17     {
18         "id": "id",
19         "name": "孟宇",
20         "gender": "男"
21     },
22     {
23         "id": "id",
24         "name": "杨孟臻",
25         "gender": "男"
26     },
27     {
28         "id": "id",
29         "name": "戴仁杰",
30         "gender": "男"
31     },
32     {
```

```
33         "id": "id",
34         "name": "杨淳屹",
35         "gender": "男"
36     }
37 ]
```

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar is visible with a collection named '9min / users / users'. Under this collection, there are several API endpoints: 'GET users', 'POST user', 'GET orders', and 'GET recommendations'. The 'GET users' endpoint is currently selected. In the main workspace, a GET request is defined with the URL 'localhost:8080/api/nine-min/users'. The 'Body' tab of the request details shows a JSON object:

```
1 [
2   {
3     "id": "id",
4     "name": "李乐天",
5     "gender": "男"
6   },
7   {
8     "id": "id",
9     "name": "姜文渊",
10    "gender": "男"
11 },
12   {
13     "id": "id",
14     "name": "杨鑫",
15     "gender": "男"
16 },
17   {
18     "id": "id",
19     "name": "孟宇",
20     "gender": "男"
21 },
22   {
23     "id": "id",
24   }
]
```

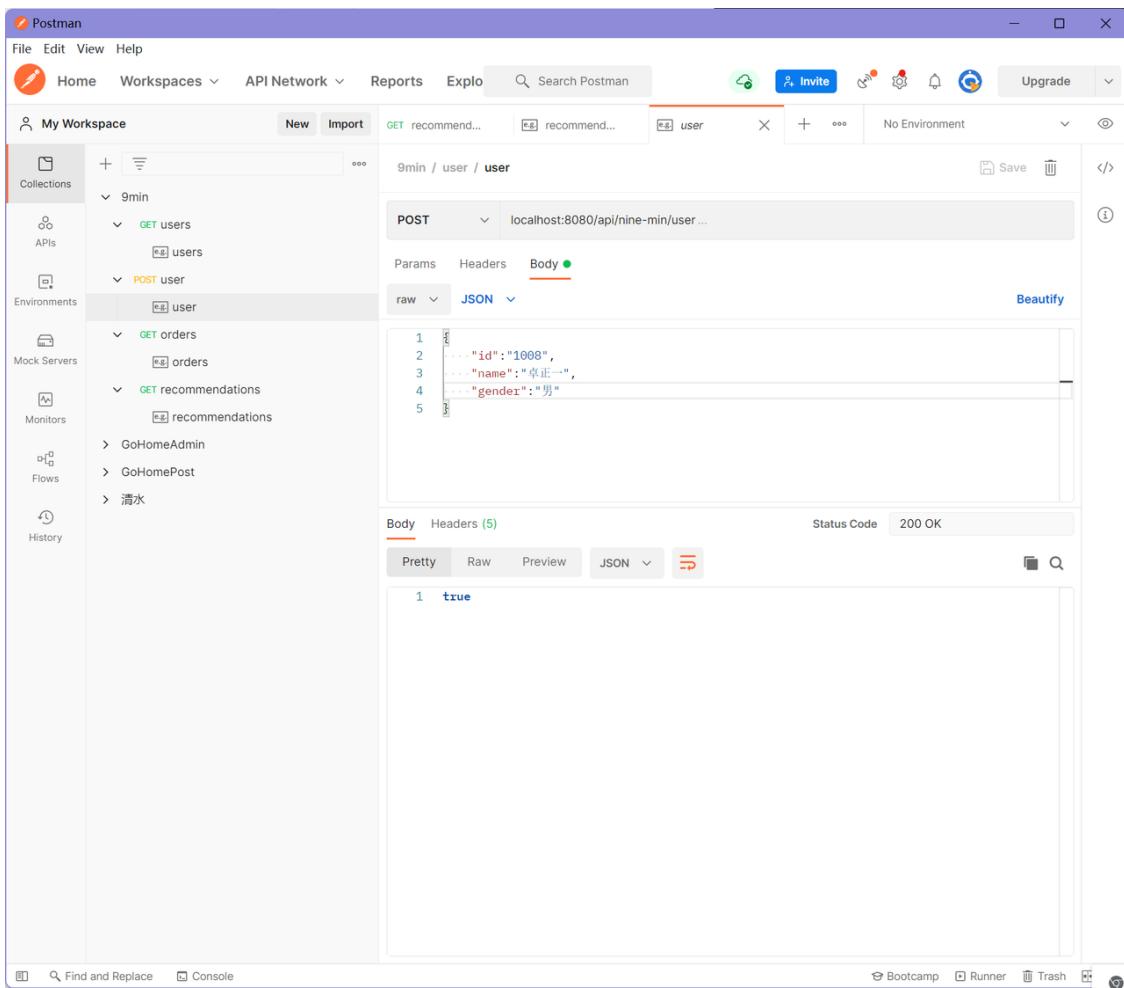
The response status code is 200 OK. The response body is displayed in a JSONpretty format, showing a list of user objects.

用户注册

该请求为post请求， REST API为api/nine-min/user， 需要以JSON格式传递请求body， 格式如下：

```
1 {
2   "id": "1008",
3   "name": "卓正一",
4   "gender": "男"
5 }
```

返回布尔值true或者false



用户获取订单列表

该请求为get请求， REST API为api/nine-min/orders， 请求参数需要
字段requireId， 返回JSON格式如下：

```
1 [  
2 {  
3     "id": "100001",  
4     "datetime": "2000-01-01 00:00:00",  
5     "deadline": "2000-01-01 00:00:00",  
6     "require": "1001",  
7     "supply": "1002",  
8     "theme": "大型机问题解决"  
9 },  
10 {  
11     "id": "100003",  
12     "datetime": "2000-01-01 00:00:00",  
13     "deadline": "2000-01-01 00:00:00",  
14     "require": "1001",  
15     "supply": "1003",
```

```

16         "theme": "中型机问题解决"
17     },
18     {
19         "id": "100005",
20         "datetime": "2000-01-01 00:00:00",
21         "deadline": "2000-01-01 00:00:00",
22         "require": "1001",
23         "supply": "1005",
24         "theme": "极小型机问题解决"
25     }
26 ]

```

The screenshot shows the Postman application interface. On the left, the sidebar displays 'My Workspace' with collections like '9min' containing 'GET users' and 'POST user', and 'GET orders' which is currently selected. The main workspace shows a 'GET' request to 'localhost:8080/api/nine-min/orders?requireId=1001...'. In the 'Params' tab, there is a single parameter 'requireId' with the value '1001'. The 'Body' tab shows the JSON response:

```

1 [
2   {
3     "id": "100001",
4     "datetime": "2000-01-01 00:00:00",
5     "deadline": "2000-01-01 00:00:00",
6     "require": "1001",
7     "supply": "1002",
8     "theme": "大型机问题解决"
9   },
10  {
11    "id": "100003",
12    "datetime": "2000-01-01 00:00:00",
13    "deadline": "2000-01-01 00:00:00",
14    "require": "1001",
15    "supply": "1003",
16    "theme": "中型机问题解决"
17  },
18  {
19    "id": "100005",
20    "datetime": "2000-01-01 00:00:00",
21    "deadline": "2000-01-01 00:00:00"
22  }
]

```

用户获取推荐列表

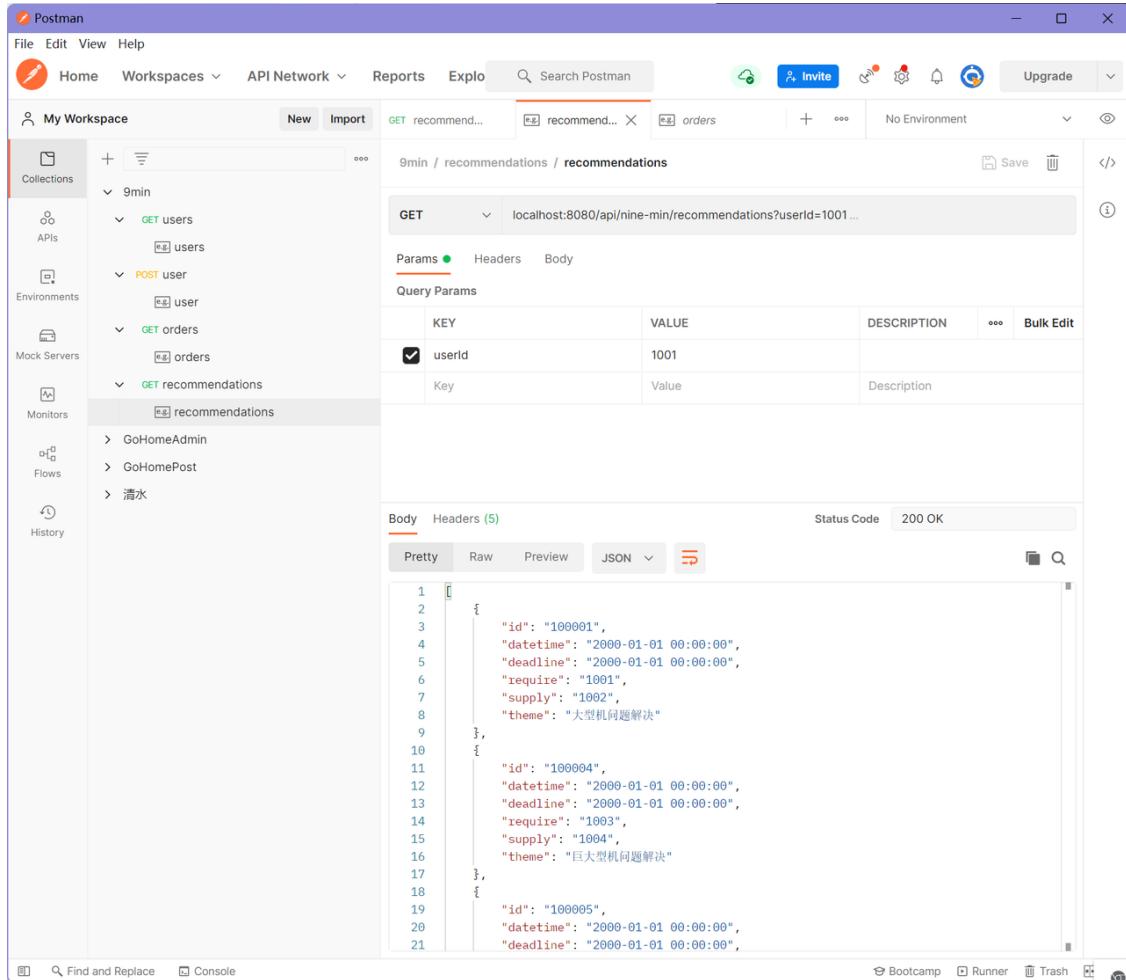
该请求为get请求， REST API为api/nine-min/recommendations，可选发送参数字段userId，返回JSON格式如下：

```

1 [
2   {
3     "id": "100001",
4     "datetime": "2000-01-01 00:00:00",
5     "deadline": "2000-01-01 00:00:00",
6     "require": "1001",
7     "supply": "1002",
8     "theme": "大型机问题解决"
9   },
10  {
11    "id": "100003",
12    "datetime": "2000-01-01 00:00:00",
13    "deadline": "2000-01-01 00:00:00",
14    "require": "1001",
15    "supply": "1003",
16    "theme": "中型机问题解决"
17  },
18  {
19    "id": "100005",
20    "datetime": "2000-01-01 00:00:00",
21    "deadline": "2000-01-01 00:00:00"
22  }
]

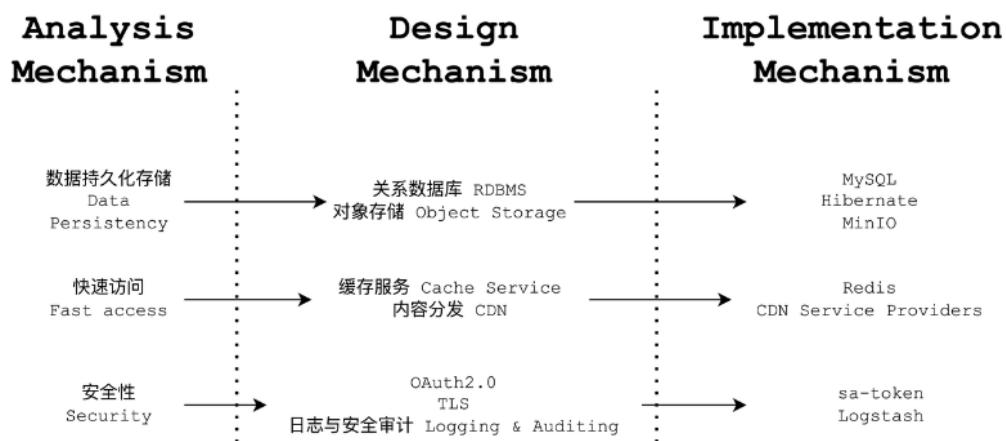
```

```
4         "datetime": "2000-01-01 00:00:00",
5         "deadline": "2000-01-01 00:00:00",
6         "require": "1001",
7         "supply": "1002",
8         "theme": "大型机问题解决"
9     },
10    {
11        "id": "100004",
12        "datetime": "2000-01-01 00:00:00",
13        "deadline": "2000-01-01 00:00:00",
14        "require": "1003",
15        "supply": "1004",
16        "theme": "巨大型机问题解决"
17    },
18    {
19        "id": "100005",
20        "datetime": "2000-01-01 00:00:00",
21        "deadline": "2000-01-01 00:00:00",
22        "require": "1001",
23        "supply": "1005",
24        "theme": "极小型机问题解决"
25    },
26    {
27        "id": "100006",
28        "datetime": "2000-01-01 00:00:00",
29        "deadline": "2000-01-01 00:00:00",
30        "require": "1004",
31        "supply": "1006",
32        "theme": "微小型机问题解决"
33    },
34    {
35        "id": "100008",
36        "datetime": "2000-01-01 00:00:00",
37        "deadline": "2000-01-01 00:00:00",
38        "require": "1006",
39        "supply": "1001",
40        "theme": "极大型机问题解决"
41    },
42    {
43        "id": "100009",
44        "datetime": "2000-01-01 00:00:00",
45        "deadline": "2000-01-01 00:00:00",
46        "require": "1007",
47        "supply": "1001",
48        "theme": "超大型机问题解决"
49    }
```



3 设计机制

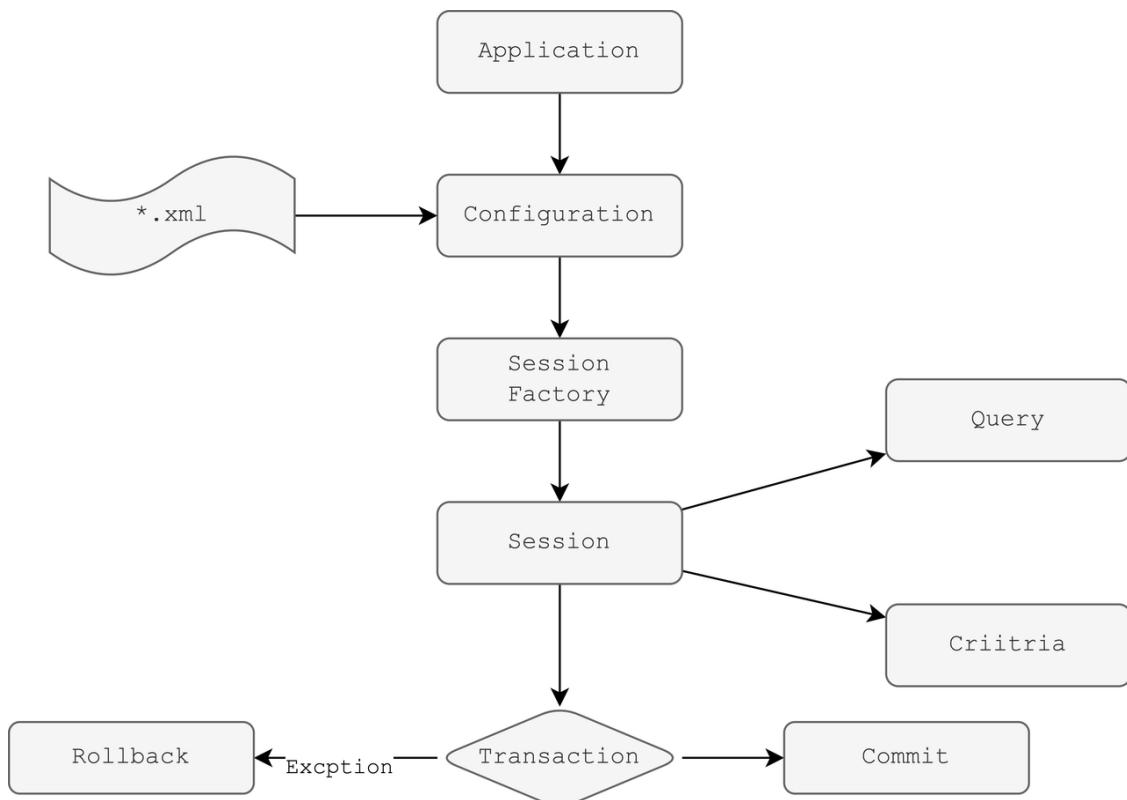
总体上，我们项目的架构设计由以下三个步骤组成：



设计机制（Design Mechanism）作为承上启下的中间环节，很大程度上决定了项目的走向。下面我们将探讨我们设计机制中的几个案例。

3.1 数据存储与访问

对于结构化数据和文本数据的存储，我们选择Hibernate作为沟通数据库和后端服务的桥梁。



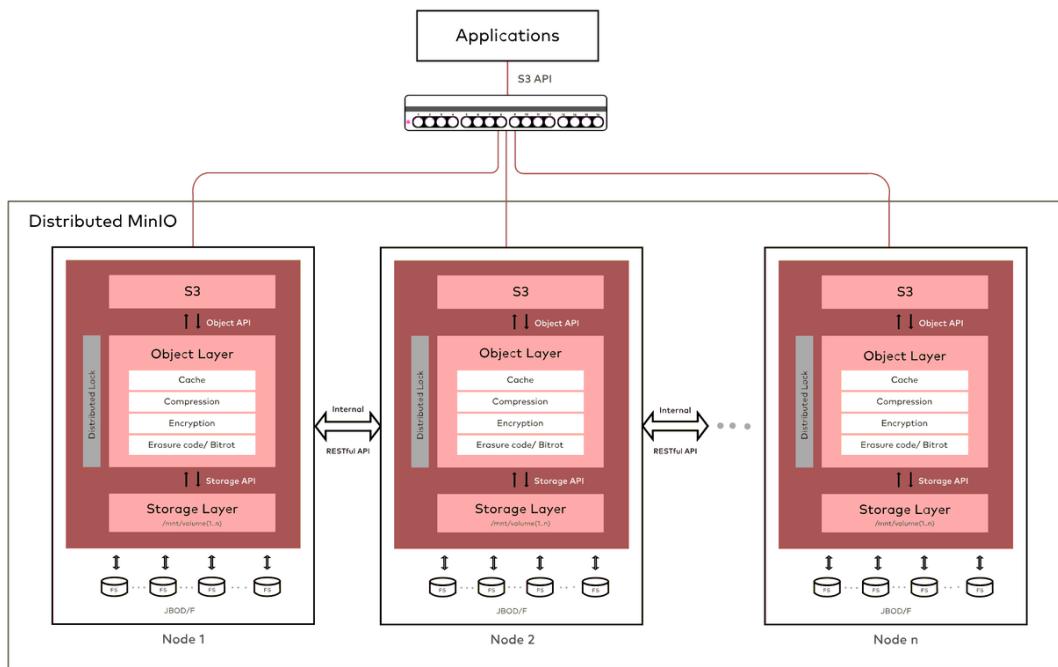
Hibernate 是一个开源的全自动的ORM框架，它封装了JDBC轻量级对象，可以生成 SQL 语句并自动执行它们。与 MyBatis 相比，Hibernate 可以自动生成 SQL 语句，减少对象与DBMS的耦合，具有完整的日志系统及更高级别可移植性。

Hibernate 的机制如上图所示。Configuration 对象是在任何 Hibernate 中创建的第一个 Hibernate 对象。它通常只在应用程序初始化期间创建一次，用于维护 Hibernate 所需的配置或属性文件。后续的 SessionFactory 对象使用 Configuration 对象的配置，并用于实例化 Session。SessionFactory 对象是线程安全的，故而可以被应用中的所有线程使用并应对并发情况下的请求。

Session 用于获取与数据库的物理连接，它提供了应用程序和数据库的接口。会话对象是轻量级的 JDBC 连接。Session 接口提供了插入、更新和删除对象的方法，并提供了 Transaction, Query 和 Criteria 的工厂方法。

Query 和 Transaction 在其它框架中都较为常见，这里就不赘述了。Criteria 可以被看成具有复杂逻辑和条件的Query，有助于实现复杂的后端数据操作。

对于多媒体数据等非格式化的数据，我们采用MinIO进行对象存储，作为关系数据库的补充。MinIO作为高性能的对象存储系统，同时也提供了很高的安全性保障，而且其逻辑结构便于服务的扩展。



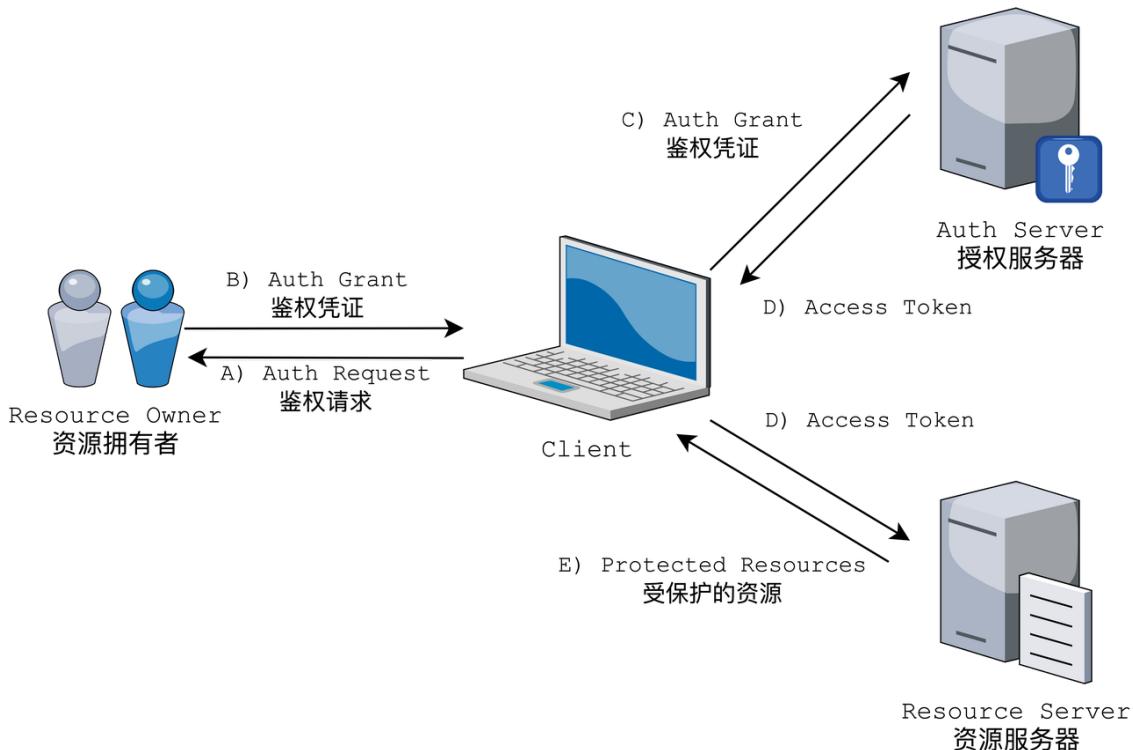
3.2 权限管理

我们的系统涉及到多个用户角色，不同的角色有不同的权限。因而，我们应该保证每个角色都能正常获取到自己的服务或资源，所以我们的系统应该提供全面、快速和友好的用户访问控制机制。

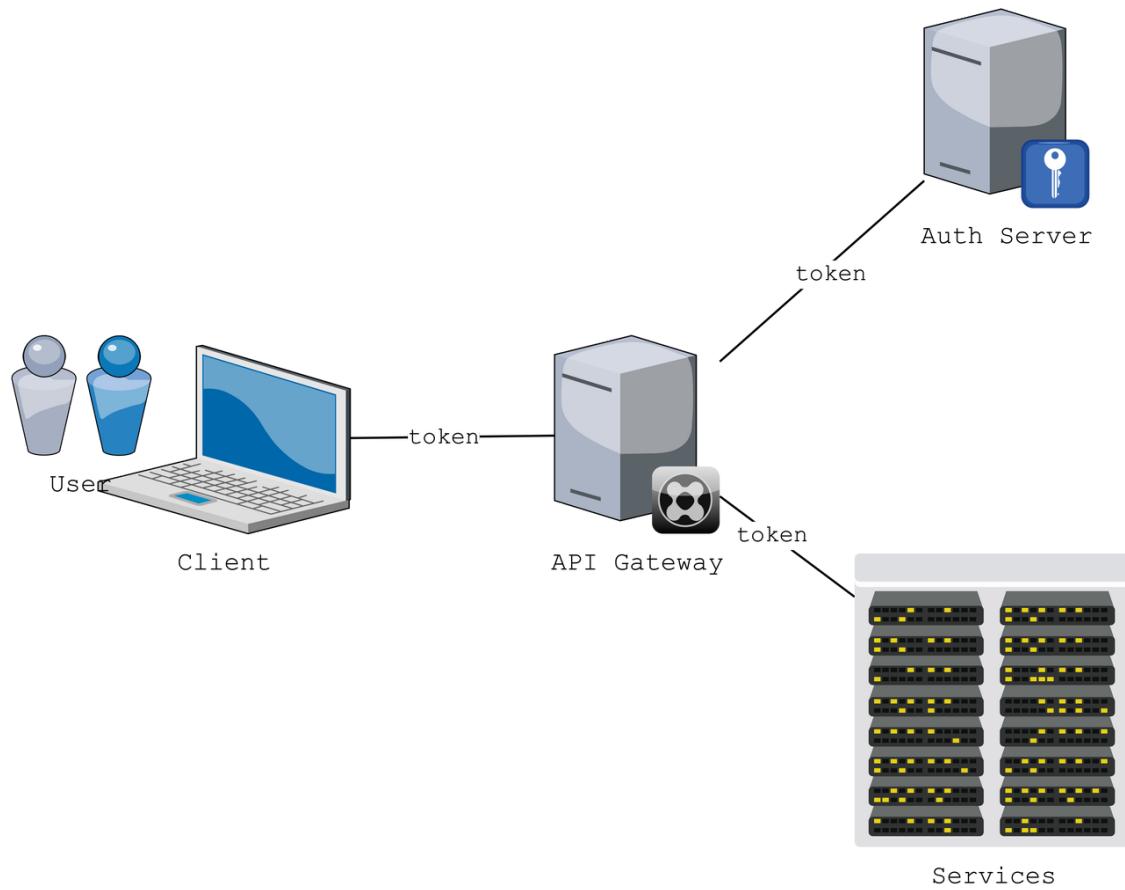
我们所使用的微服务架构与常见的权限管理实践相结合，可以提供更细粒度、更加可靠的用户访问控制机制。一方面，各微服务可以通过中心化的授权系统进行权限的鉴定，另一方面，各微服务内部可以设计更加灵活的权限控制机制。在可靠性方面，与单一应用模式相比，基于微服务的系统架构模式可以更快地提供服务。当一个微服务失败时，其余微服务不会受到影响。此外，每个微服务都可以充分利用其硬件资源，并且独立扩展不会影响其他微服务。

我们的项目中，使用 OAuth 2.0 standard protocol 进行授权管理。OAuth 2.0 是基于 token 的授权管理系统。与基于 cookie 的认证相比，基于 token 的认证是无状态的，可以防止跨站点请求伪造（CSRF），并支持多站点使用。因此，我们的系统使用基于 token 的 OAuth 2.0 协议进行权限认证。

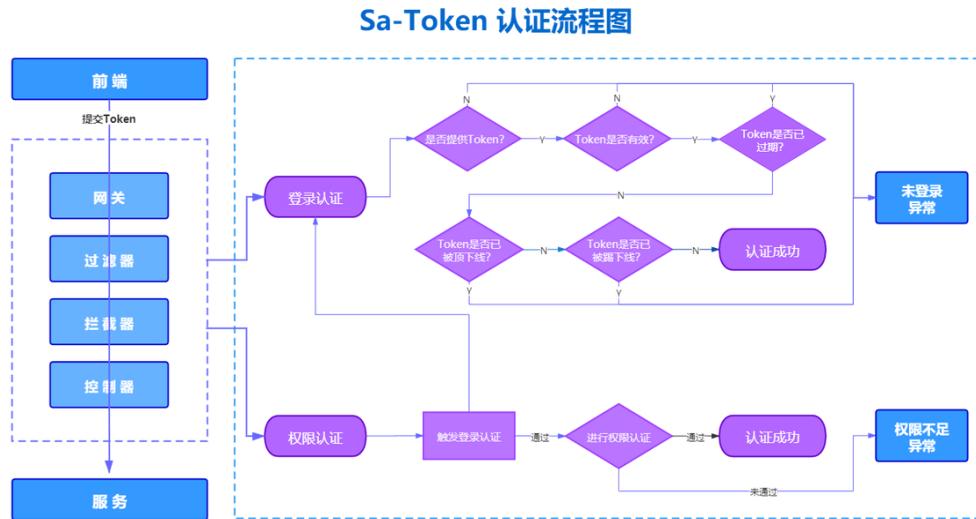
一个典型的授权过程如下图所示：



为了将 OAuth 2.0 与微服务架构相结合，我们的系统采用了常规的 API 网关的做法。API 网关是微服务架构的标准组件，它负责组织微服务架构正在处理的请求。API 网关接受来自客户端的请求，然后通过请求路由将它们路由到相应的微服务。通常，它调用多个微服务并聚合结果。因此，在 API 网关中引入权限管理，减少了客户端和后端的往返次数，从而提升了授权和其它服务的效率。下图展现了 API 网关、授权服务及各微服务之间的关系。



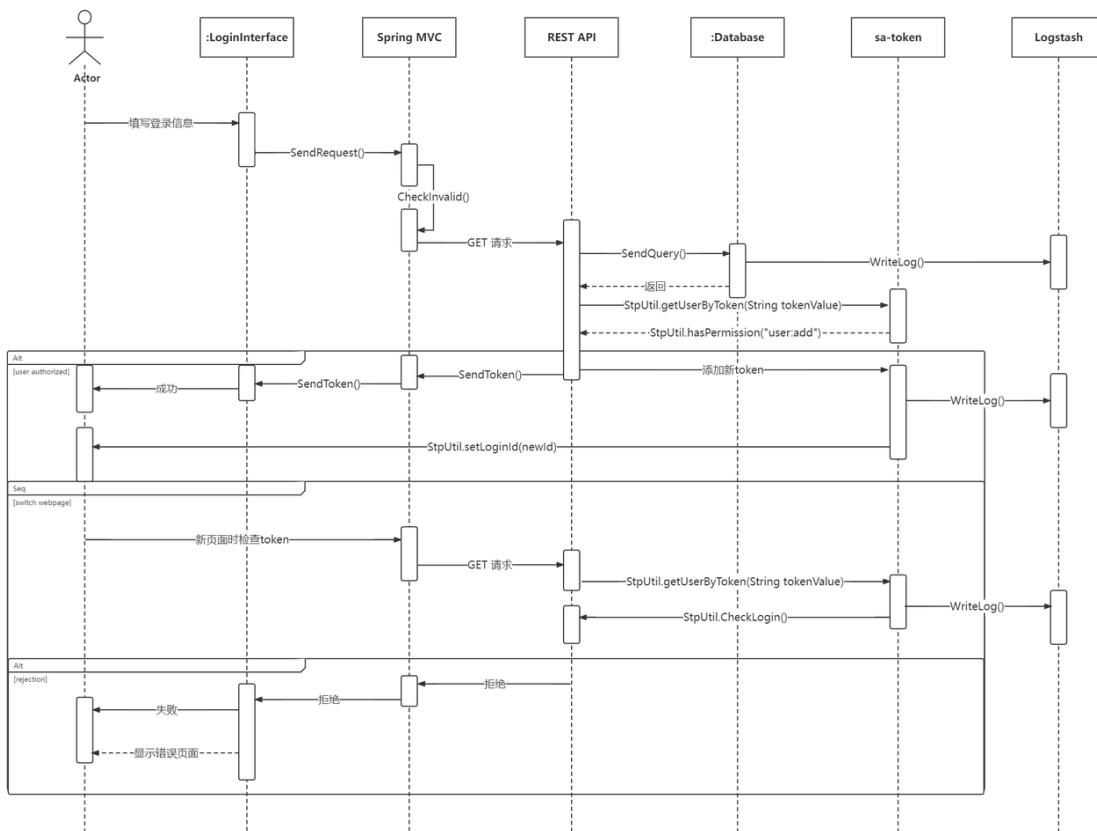
在实际开发中，我们选取 sa-token 作为我们授权服务的框架，相比于其它框架，其轻量的特性使得其在微服务架构中具有较好的灵活性，便于我们将其整合到整个系统中。Sa-OAuth2 模块基于 RFC-6749 标准编写，通过 Sa-OAuth2 可以非常轻松的实现系统的 OAuth2.0 授权认证。除了 OAuth2.0 外，sa-token 还支持常见的二级认证，这在我们的项目中也较为常见。典型的 sa-token 认证流程如下图所示。



4. 用例的实现

4.1 用户登录

时序图



每个已经注册过账号的用户都可以登录进入我们的系统，上面的时序图展示了这一过程之中的一些细节。我们使用 Spring MVC 来处理用户的登录请求；使用 REST API 将登录中的信息和请求发送到后端的数据库中，其中存储了所有用户账户的相关信息。而对于安全认证，我们则使用了 Sa-Token 进行账户权限检查和登录状态检查。

用户在登录表单中输入用户名和密码。Login Interface 会先对信息进行输入级别的检查，即系统会拒绝格式不合法的登录信息（如空用户名和空密码）。之后，信息将被解析为 JSON 格式。登录请求将被发送到 Spring MVC。同时，带有登录信息的 JSON 文件将通过 REST API 发送到数据库，同时向数据库发送相应的查询请求。

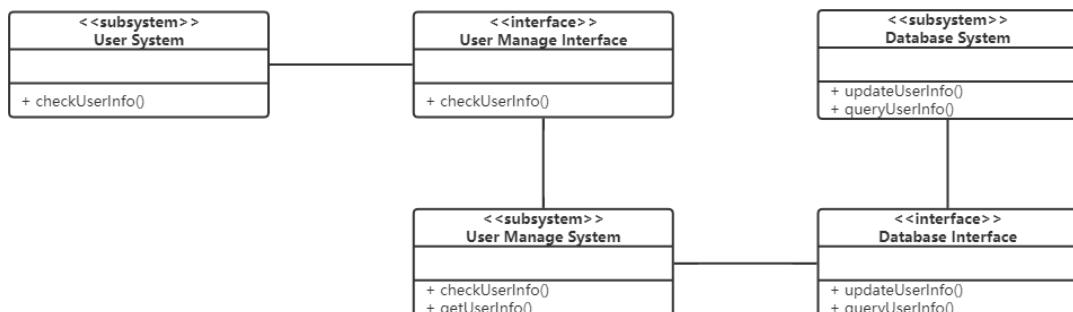
数据库中的用户表存储了用户的所有信息。数据库查询用户表，查找给定信息是否在用户表中。如果存在，则将成功消息返回给用户管理子系统，即可以在数据库中找到对应的用户名和密码。

在得到查询结果后，我们的系统也会启动一个验证过程来检查用户帐户是否被允许登录。身份验证过程将由 Sa-Token 处理。通过调用 StpUtil.getUserByToken 方法，会返回用户的 token 是否合法。如果证明 token 是合法的（如果出现用户封禁等情况，其 token 会被设置为非法），会生成一个新的登录 token，发送到登录子系统，并保存到本地。

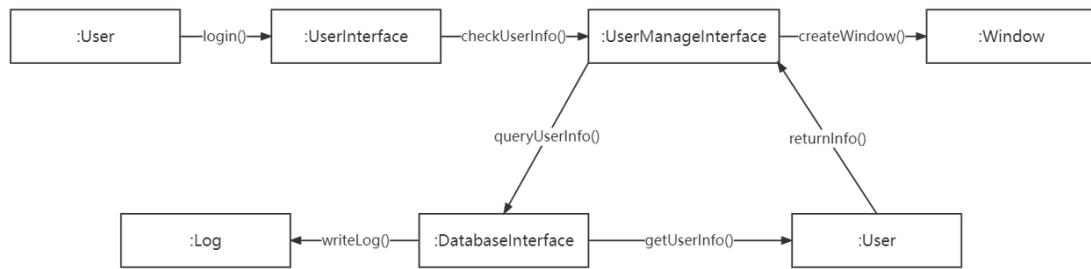
每当用户在我们系统中切换网页时，Sa-Token（通过 REST API）将在登录状态级别（操作由 StpUtil.getLoginIdByToken 方法提供支持）检查帐户的 token，以防系统崩溃或其他错误情况。具体来说，当用户的登录状态受到系统故障影响时，token 将被禁用，系统将拒绝用户继续使用该服务。

对应的子系统和接口，以及它们的通信方式，如下面的类图和通信图所示：

类图

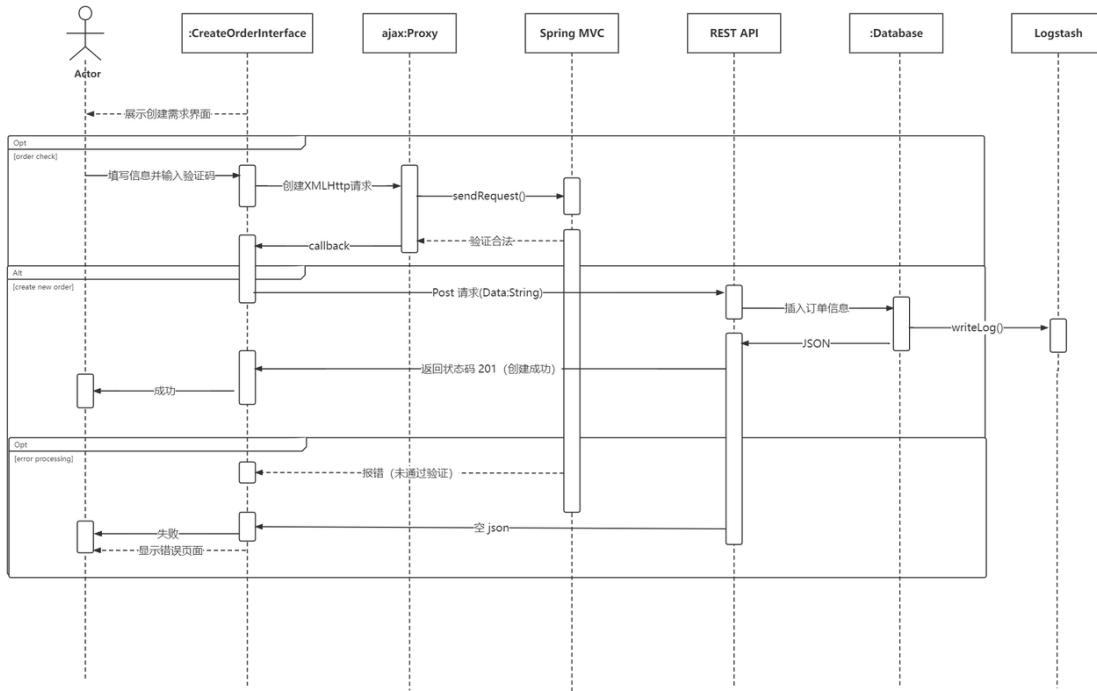


通信图



4.2 发布需求

时序图



发布需求是我们系统中的一个用例，用户可以创建并发布需求。过程中具体的细节在上面的时序图中展示出来了。

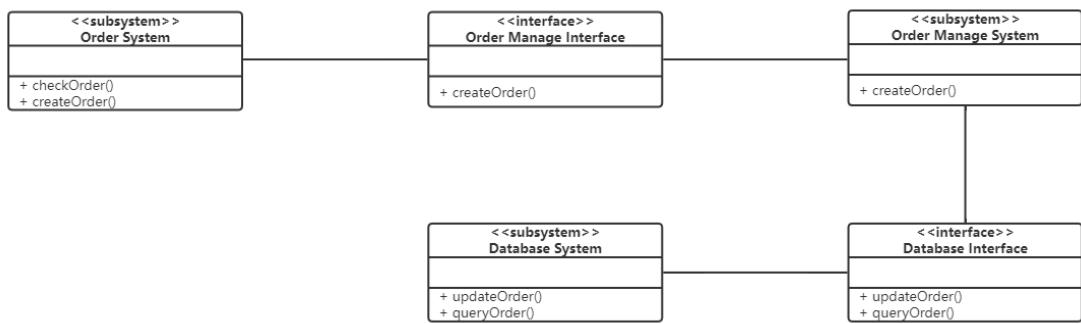
根据我们的系统设计，我们使用AJAX进行数据交互。我们之前的用例实现中使用的 Spring MVC 框架将在这里用于处理请求以及验证过程。通过验证的内容将通过 REST API 提交到数据库。

用户在创建需求页面填写相关信息后，还要输入验证码以验证是否为机器人。点击提交按钮后，会生成HTML表单数据，通过AJAX技术转化为JAVA对象，然后这个对象相对应的请求会被发送到Spring MVC中。Spring MVC首先会检查需求信息格式的正确性，然后检测验证码是否正确以确保不是机器人用户。

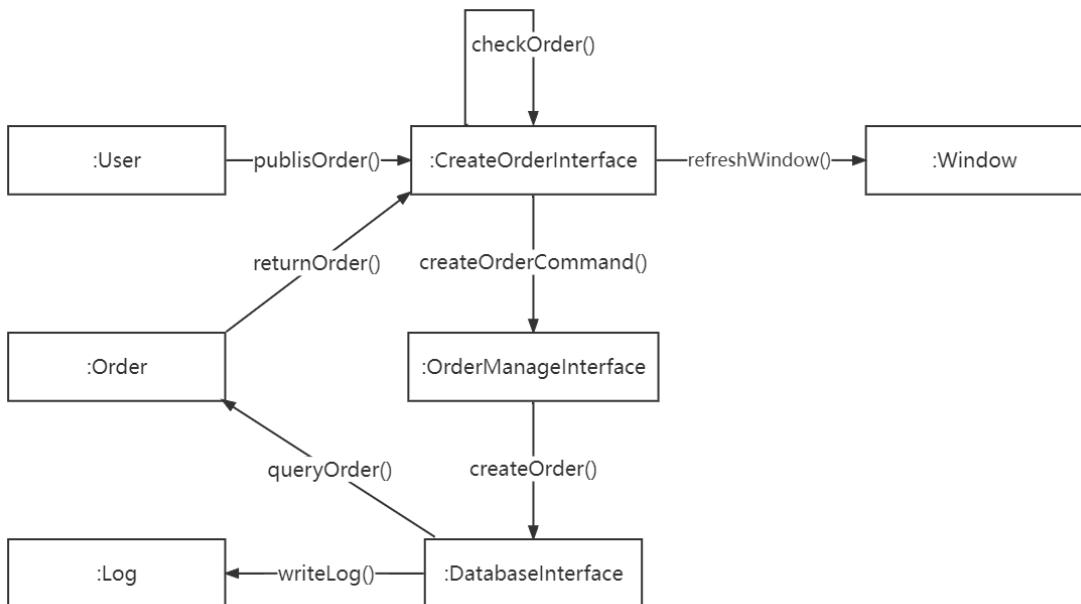
验证通过后，将要提交的内容通过对应的 POST 请求发送到数据库，数据库会以JSON 格式返回结果，表明内容已成功加载到数据库。如果操作出错，会返回一个空的JSON，并被错误处理方法捕捉到（比如用户未实名认证或网络原因等）。最后系统刷新网页，以实时显示需求创建的状态。

对应的子系统和接口，以及它们的通信方式，在下面的类图和通信图中表示：

类图



通信图



5. 产品原型进展

5.1 前端原型

我们使用了flutter以及其material design组件库进行了初步的原型开发，以下为开发环境截图：

The screenshot shows a code editor with a Flutter project open. The file being edited is `lib/main.dart`. The code defines a `MyApp` widget that extends `StatelessWidget`. It sets the title to 'smile' and uses a `MaterialApp` theme with a primarySwatch of `Colors.deepPurple`. The `home` property points to the `homPage` method, which returns a `Scaffold` with a `body` containing a `Text` widget with the text 'Hello World'.

```
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'smile',
      theme: ThemeData(
        primarySwatch: Colors.deepPurple,
      ), // ThemeData
      home: homPage(context),
    ); // MaterialApp
  }

  Widget homPage(BuildContext context) {
    return Scaffold(
      appBar: header(context) as PreferredSizeWidget,
      body: body(context),
    );
  }
}

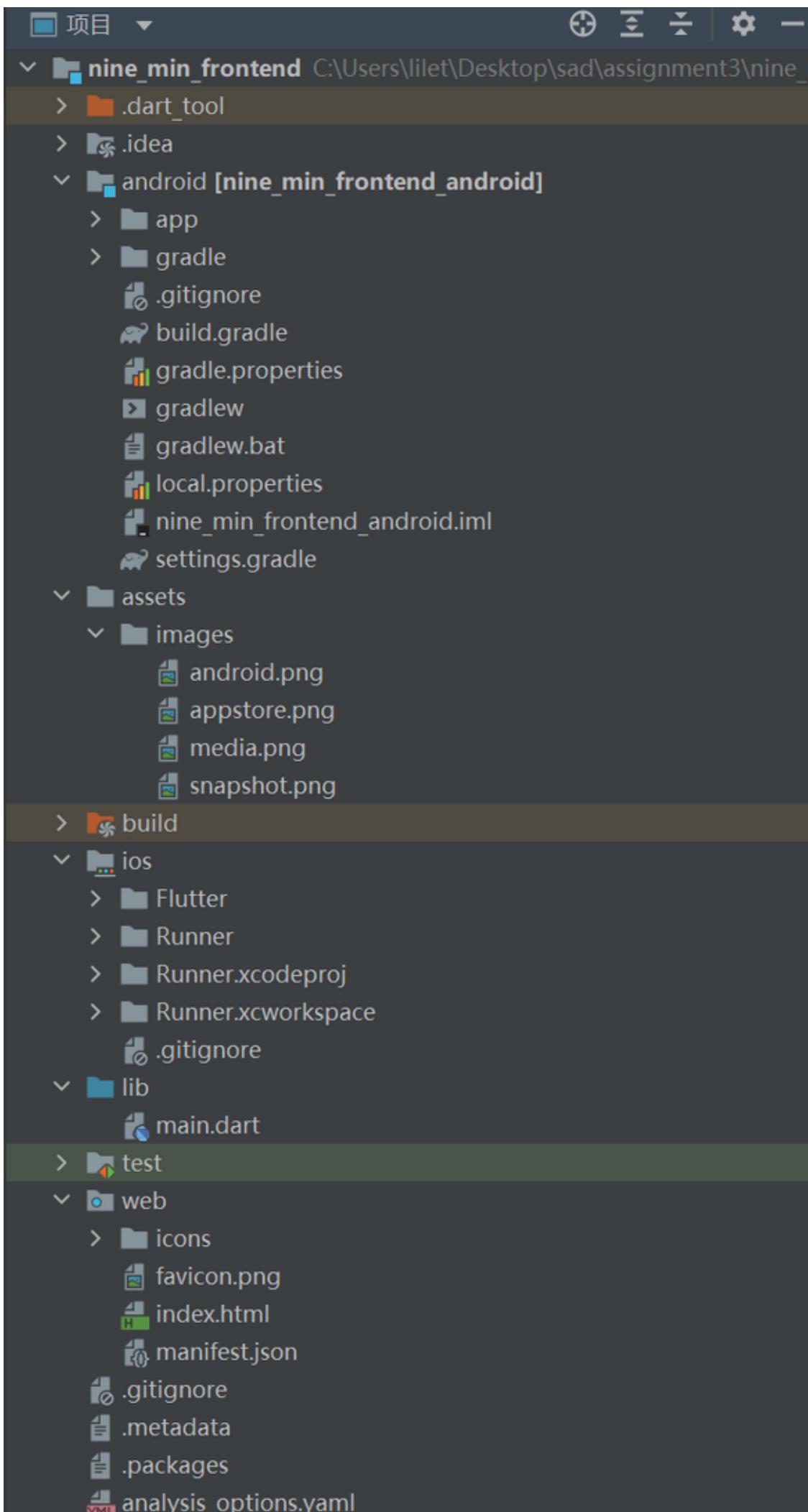
Widget header(BuildContext context) {
  return AppBar(
    title: Text('Hello World'),
  );
}

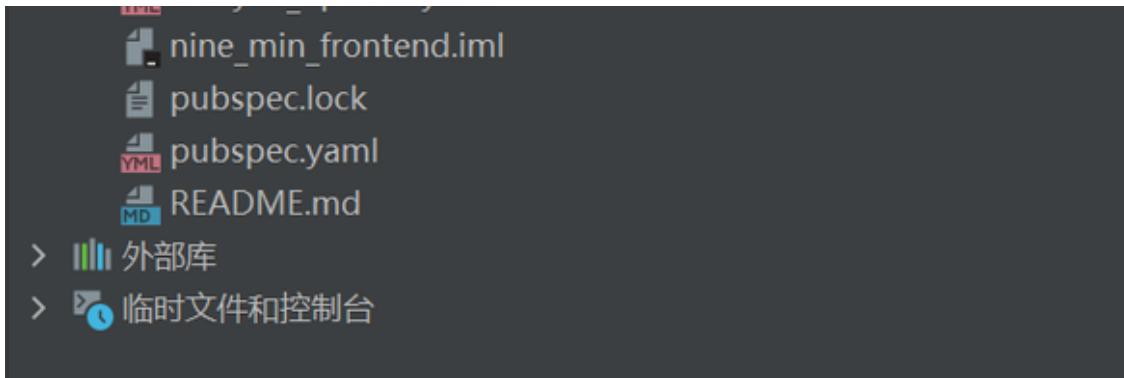
Widget body(BuildContext context) {
  return Center(
    child: Text('Hello World'),
  );
}
```

Below the code editor, there is a terminal window showing the build process:

```
BUILDING WITH SOUND null SAFETY
Compiling lib/main.dart for the Web... 29.5s
(Flutter) PS C:\Users\lilit\Desktop\and\assignment\smile_main_frontend> flutter build web
BUILDING WITH SOUND null SAFETY
Compiling lib/main.dart for the Web... 26.1s
(Flutter) PS C:\Users\lilit\Desktop\and\assignment\smile_main_frontend> [redacted]
```

项目目录如下所示：





初步的原型如图所示：

9min 计算机科学

大型机环境配置报错ijk23333寻求解决方案

Due:2020.4.12 23:59

\$ 300.00

接取需求

文字描述

jwy昨天晚上在给大型机作业配环境，每一次从某个卷启动都会报一个特定的错误叫ijk23333。查阅了网上所有文档，都没有关于这个错误的说明。但是问了同学之后发现同学都没有这个错误，但是助教对于这种特异性问题也无能为力。这个时候jwy发现了一个叫做9min的学术援助平台，可以为自己提供高度专业匹配的定制化学术问题解决方案。

jwy以访客的身份浏览了主页，发现里面的问题确实非常的专业，甚至有很多自己熟悉的linux运维之类的内容。抱着试一试的心态，他点了注册按钮，注册非常人性化，可以直接用微信登录，登陆之后只需要简单的设置。jwy很聪明，首先他没有直接发布需求，首先它先搜索了一下关于fei的内容。虽然没有和他完全一致的结果，但是底下推送了很多大型机相关的已完成的单。

视频描述

9min是专注于学术与技术困难付费解决方案的专业技能支援平台

关于我们 联系我们 帮助中心

©2022-2022 同济大学 版权所有

App Store

Android

关注我们：[QR code]

5.2 后端原型

后端我们初步使用了基于SpringBoot的Web服务容器的开发，开发环境与部分代码如下：

The screenshot shows the IntelliJ IDEA interface with the code editor open to the `Controller.java` file. The code implements a REST controller for users and orders.

```
package com.example.nine_min_backend;

import com.example.nine_min_backend.dto.Order;
import com.example.nine_min_backend.dto.User;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

@.RestController
@RequestMapping(value = "/api/nine-min/")
public class Controller {

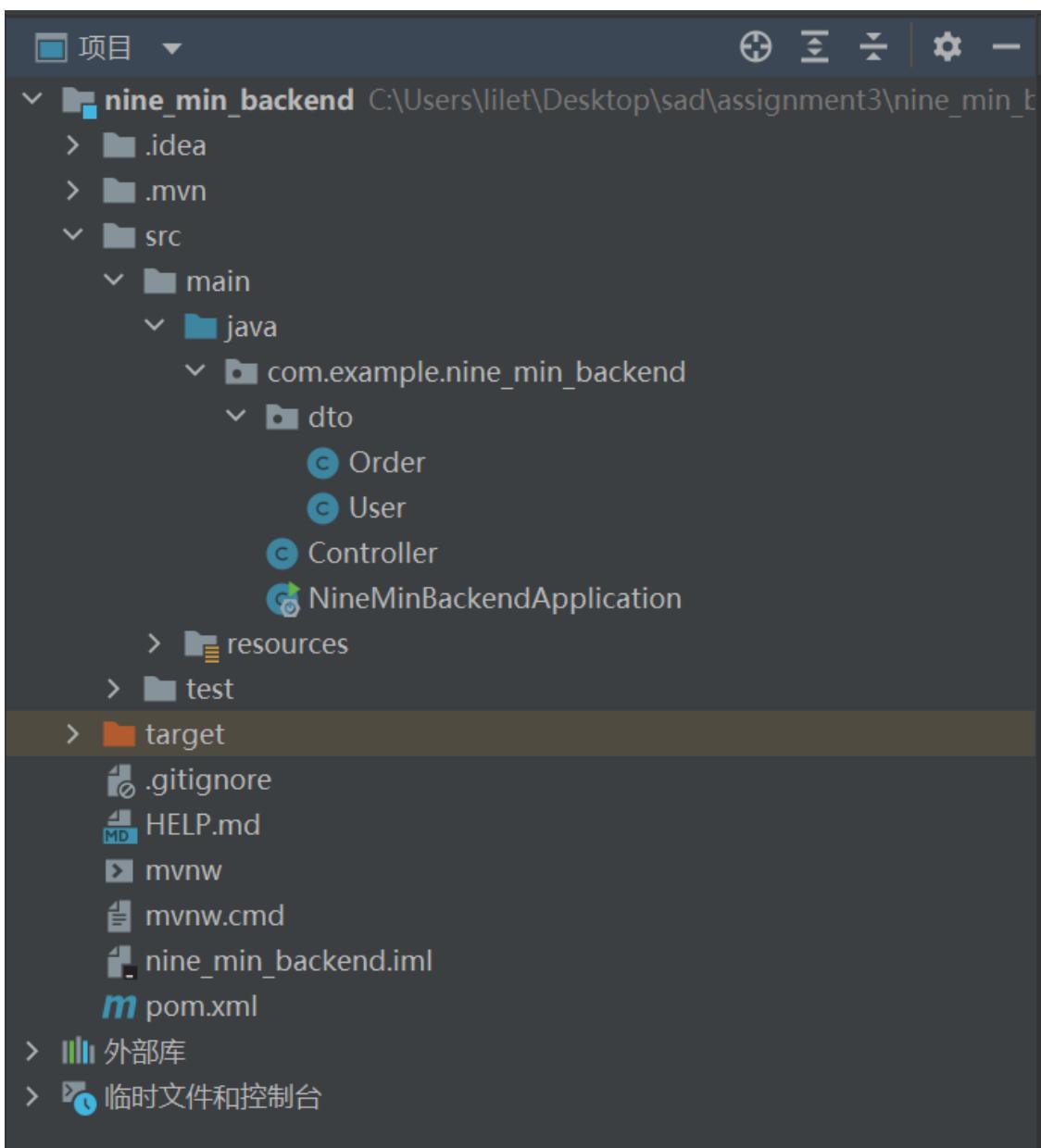
    @GetMapping(value = "/users", method = RequestMethod.GET)
    ResponseEntity<List<User>> getUsers() {
        List<User> list = new ArrayList<>();
        list.add(new User(id: "1001", name: "李华天", gender: "男"));
        list.add(new User(id: "1002", name: "张文清", gender: "男"));
        list.add(new User(id: "1003", name: "陈晓", gender: "男"));
        list.add(new User(id: "1004", name: "李亚丽", gender: "女"));
        list.add(new User(id: "1005", name: "陈立强", gender: "男"));
        list.add(new User(id: "1006", name: "陈立红", gender: "女"));
        list.add(new User(id: "1007", name: "陈立华", gender: "女"));

        return new ResponseEntity<>(list, HttpStatus.OK);
    }

    @PostMapping(value = "/user", method = RequestMethod.POST)
    ResponseEntity<Boolean> postUser(@RequestBody User user) {
        return new ResponseEntity<>(true, HttpStatus.OK);
    }

    @GetMapping(value = "/orders", method = RequestMethod.GET)
    ResponseEntity<List<Order>> getOrders() {
        return new ResponseEntity<>(
            Order.values().stream()
                .map(o -> o)
                .collect(Collectors.toList()),
            HttpStatus.OK
        );
    }
}
```

项目目录如下：



使用postman测试后端Web服务运行结果如下：

The screenshot shows the Postman interface with a collection named '9min' containing several API endpoints. The 'orders' endpoint under 'GET orders' is selected. The request method is 'GET' and the URL is 'localhost:8080/api/nine-min/orders?requireId=1001'. In the 'Query Params' section, there is a key 'requireId' with a value '1001'. The response body is displayed in a pretty-printed JSON format:

```
1 {
2   "id": "100001",
3   "datetime": "2000-01-01 00:00:00",
4   "deadline": "2000-01-01 00:00:00",
5   "require": "1001",
6   "supply": "1002",
7   "theme": "大型机问题解决"
8 },
9 {
10   "id": "100003",
11   "datetime": "2000-01-01 00:00:00",
12   "deadline": "2000-01-01 00:00:00",
13   "require": "1001",
14   "supply": "1003",
15   "theme": "中型机问题解决"
16 },
17 {
18   "id": "100005",
19   "datetime": "2000-01-01 00:00:00",
20   "deadline": "2000-01-01 00:00:00",
21 }
```

6 自我反馈

李乐天：

通过本次项目，我学到了很多关于软件工程、软件架构、系统分析方法论方面的知识，并亲身实践，获得了很多的收获。整体上我对于软件开发的流程与复杂度、面临的问题、以及各方面因素的考虑上有了较为完整、顶层的视角，并提升了决策能力与整体规划能力。在本次项目中，我也学习到了很多团队协作中的较好的实践，扩宽了自己的技术栈，并提升了与别人交换想法的能力。

杨淳屹：

经过这一学期的系统分析与设计课程的学习和作业完成以后，我学会并动手参与了一个系统从无到有的建立的完整过程。在这个过程中，我们学会了设计一个系统的用例，并以用例图和用例说明书的形式清楚地表示出来，以及如何用活动图和顺序图来完整地描述这些用例的功能。此外，还了解了系统设计过程中可能涉及到的机制。这个过程学到的知识让我们以后的项目开发更加规范系统，不会再像初学者一样面对脑海中的复杂系统无从下手。

在学习的过程中，也锻炼到了团队沟通协作能力，对以后的团队合作项目也积累了宝贵的合作经验。

姜文渊：

本学期的项目中，在知识方面，我学习到了一些关于软件架构，系统分析与设计以及软件工程的一些基本概念，常见观点以及常用工具；并且在项目的进行过程中，探索了诸多软件工程的方法论和平台相关的技术栈。在能力方面，在完成每次作业的过程中，一方面我的资料查阅的能力得到了增强，另一方面我也逐渐能使用相关领域的专业术语较为准确地与同学交流问题与想法；此外，整个团队协作推进项目的过程中，我深刻认识到软件架构的巨大的复杂性，并且能够从较为整体的视角与同学讨论各方面的取舍。诚如米开朗基罗所言“塑像本来就在石头里，我只是把不需要的部分去掉”，在当今各类软硬件商用与开源解决方案纷繁复杂的时代，软件架构的分析与设计很多时候就是在成本的限制下选取合适架构，并舍弃多余的部分。通过这一学期的学习与实践，我相信在以后在面临较为庞大的软件系统时不会无从下手，更不会如盲人摸象抑或管中窥豹一般只见实现细节而不见架构本身。

杨鑫：

在学习本课程之前，我对软件开发等相关概念不甚了解，没有过多关注到软件开发这样一个相当重要的知识体系，以前做项目也只是跟着经验走，没有成体系的开发过任何一个项目，因此也常常遇到许多困难和疑惑。

而通过这一学期课程的学习和与团队共同完成这些项目，我逐渐学会了软件开发的许多知识和概念，学会了如何分析一个系统，从功能分析到架构分析，对各种软件的架构有更深入的了解。今后在做项目的时候，要利用所学到的用例图、活动图以及时序图等工具来更加直观和深层次的剖析、分析我们的系统，这将使得我们的软件开发更加系统化，规范化，可以有效避免曾经遇到的各种因为开发方法不当而导致的错误和难题。

此外，本学期的项目还使得我在团队协作中的能力得到了锻炼。我学会了与队员进行高效沟通和协商的方式，在任务分配和协作上有了更丰富的经验。希望我在将来的团队协作中充分运用到我所学习到的知识和积累的经验，从而更加高效，愉快的完成这些任务。

孟宇：

经过本学期课程学习和项目实战，我对大型复杂软件系统的开发过程有了全新的理解。过去我对软件开发的观点认知停留较低级的工具技巧的使用和业务逻辑的实现上，通过本课程和项目的学习，我掌握了如规范UML图的语法和绘制，原型图的设计方式等很多工具的使用；了解到了如何从更高的观点上去设计一个系统，形成了对系统分析与设计过程的概念，并进行了实践；开拓视野，提高了大局观和全局规划能力。最重要的是，我从这门课中认识到了：编程的终点是改善人们的生活，而不是开发者的自娱自乐。

此外，我很幸运能与几位优秀的同学作为队友在项目中合作，小组全员分工明确，沟通积极，态度端正，工作高效，且个人能力都很强，让我从中学到了许多，积累了丰富的经验，团队合作能力得到了很大的提升。

7.Contribution

这个系统分析项目已经讨论过三次了。在完成这一任务的过程中，所有团队成员都积极参与讨论，并努力完成自己的任务。团队成员和谐合作，及时沟通和及时解决了问题。小组内部的分工平均而明确，具体如下：

Contributor	介绍	架构完善	设计机制	用例实现	产品原型
李乐天			+	+++	+++
孟宇	+++		++		
姜文渊		+++	++		
杨淳屹	++		++		
杨鑫		++	+	+++	+

成员名单及得分占比

姜文渊 1951510 100%

李乐天 1950848 100%

孟 宇 1951477 100%

杨 鑫 1950787 100%

杨淳屹 1953824 100%