海 南 大 学

分布式与云计算技术课程设计报告

题　　目：基于微服务架构的校园网的设计与实现

学　　号：20203105988

姓　　名：王涛

年　　级：2020 级

学　　院：应用科技学院

学　　部：工学部

专　　业：网络工程

指导教师：刘金羽

完成日期：2023 年　6 月　24 日

目录

摘要

使用 JDK 17、Spring Boot 3.1 和 Spring Cloud 构建和部署基于分布式微服务架构模式的校园网系统能够带来诸多益处。首先，JDK 17 作为最新的 Java 开发工具包版本，提供了卓越的性能和安全性，使系统能够更高效地执行任务并保障数据的安全。

Spring Boot 3.1 是一个快速开发框架，它采用约定优于配置的理念，简化了 Spring 应用程序的配置和部署过程。借助 Spring Boot 的自动配置和开箱即用特性，开发人员可以更专注于业务逻辑的实现，节省了大量的开发时间和工作量。同时，Spring Boot 还提供了丰富的开发工具和便捷的调试方式，提升了开发效率和质量。

使用 Spring Cloud 构建校园网系统可以将其转变为一个由多个独立微服务组成的分布式系统。每个微服务负责特定的功能模块，通过服务注册与发现、负载均衡、断路器等组件实现相互之间的通信和协作。这种架构将系统解耦并降低了服务之间的依赖性，使系统更加灵活、易于扩展和维护。

关键词：Java；Spring Boot；微服务；分布式

一、小组成员及分工

组长：闫宏基

组员：赵才博、庄梦淮、冯骥、王涛、陈哲宇

工作分工：

赵才博、闫宏基负责实现以及测试；

陈哲宇、庄梦淮负责系统架构设计；

冯骥、王涛负责需求实现分析；

二、关键技术

使用 JDK 17、Spring Boot 3.1 和 Spring Cloud 构建和部署基于分布式微服务架构模式的校园网系统能够提供更好的性能、开发效率和系统的可伸缩性、弹性和可管理性。

1. JDK 17：它带来了许多新特性和改进，包括性能优化、安全增强、新的语言功能和 API 等。使用最新的 JDK 版本可以提供更好的性能和稳定性，并且能够享受到 Java 语言技术的好处。

2. Spring Boot：Spring Boot 是一个用于构建 Java 应用程序的开源框架，它简化了应用程序的创建、配置和部署过程。使用 Spring Boot 可以快速搭建微服务应用程序，并具有自动配置、内嵌容器、可管理的依赖关系等特性，使得开发变得更加高效和便捷。

3. Spring Cloud：Spring Cloud 是一个用于构建分布式系统和微服务架构的框架。它提供了诸如服务注册与发现、负载均衡、断路器、配置中心、消息总线等功能，可以帮助开发者构建弹性、可伸缩的分布式应用程序。使用 Spring Cloud 可以简化微服务架构的开发和管理，提供了丰富的工具和组件来解决分布式系统所面临的问题。

4. 分布式微服务架构模式：基于分布式微服务架构模式可以将系统拆分成多个小型、自治的服务，每个服务都专注于完成特定的业务功能。这种架构模式具有高内聚、低耦合的特点，并且易于扩展和维护。通过合理划分和组织微

服务，可以实现校园网系统的模块化开发、部署和运维，提高系统的可伸缩性和灵活性。

5. 可伸缩性和弹性：借助 Spring Cloud 提供的服务注册与发现、负载均衡等功能，校园网系统可以灵活地添加、删除和水平扩展各个微服务实例，以应对不同规模和负载压力的需求。同时，使用容错机制可以提高系统的弹性，确保即使某个微服务出现故障，也不会影响整个系统的可用性和稳定性。

6. 高度可管理性：Spring Boot 和 Spring Cloud 提供了丰富的监控和管理工具，可以实时监控系统的健康状况、指标数据等，从而及时发现和解决问题，提高系统的可管理性和运维效率。

在前端方面

Vue 3：JavaScript 前端框架，它提供了用于构建用户界面的工具和组件。Vue 3 相对于早期版本有许多改进，包括更好的性能、更小的包大小、更强大的响应式系统以及更好的开发体验。它支持虚拟 DOM 技术，使得页面更新更加高效，并提供了丰富的 API 和生态系统，使开发人员可以轻松构建交互性强、响应迅速的现代 Web 应用程序。

TypeScript 4：是 JavaScript 的超集，并添加了静态类型检查和更多的编程工具。TypeScript 能够让开发者在开发过程中尽早地检测到潜在的错误，增加代码的可靠性和可维护性。它还提供了面向对象编程的特性，包括类、接口、泛型等，使得代码结构更清晰，易于理解和扩展。TypeScript 的编译器将 TypeScript 代码转换为纯 JavaScript 代码，从而可以在任何支持 JavaScript 的环境中运行。

Pinia 2：基于 Vue 的状态管理库，它提供了一种简洁而优雅的方式来管理应用程序的状态。Pinia 2 借鉴了 Vuex 的概念，但更加轻量、直观和易于使用。它通过定义 store 来管理应用程序的状态，并提供了一些 API 来访问和修改这些状态。Pinia 2 还支持模块化和热重载，使得状态管理变得更加灵活和高效。

Axios 1：基于 Promise 的 HTTP 客户端库，用于在浏览器和 Node.js 中发送 HTTP 请求。它提供了简洁而强大的 API，使得在应用程序中进行网络请求变得更加便捷和可靠。Axios 支持各种请求类型，并提供了丰富的配置选项，如

请求拦截、响应拦截、错误处理等。它还支持异步操作、取消请求等功能，可以轻松地与后端 API 进行交互，并处理常见的网络请求场景。

关于后端方面

WebFlux：它基于 Reactive Streams 规范，使用异步非阻塞的方式处理请求和响应。WebFlux 支持函数式编程风格和注解式编程风格，可以处理大量并发请求，并具有更好的性能和可伸缩性。通过使用 WebFlux，可以构建高性能、响应式的 Web 应用程序。

JWT：它使用 JSON 格式对信息进行编码，可以包含用户身份验证和授权等相关数据。JWT 通常由三部分组成：头部、载荷和签名。头部包含描述令牌类型和使用的加密算法的信息，载荷包含实际传输的数据，签名用于验证令牌的合法性。JWT 具有无状态、可扩展性和跨平台等特性，广泛应用于身份验证和授权场景。

Spring Security：它提供了一套强大且可扩展的认证和授权机制，可以轻松集成到 Spring 项目中。Spring Security 可以用于管理用户身份验证、访问控制、会话管理、密码加密等安全相关功能。同时提供了各种安全策略和过滤器来保护应用程序免受攻击。

架构技术方面

Spring Cloud Config：它支持将配置存储在版本控制系统中，并通过 REST 接口提供给各个微服务。使用 Spring Cloud Config，可以实现配置的集中化管理、动态刷新和版本控制，方便快速调整和部署应用程序。

Spring Eureka：服务提供者通过向 Eureka 服务器注册自己的服务信息，而服务消费者可以从 Eureka 服务器获取可用的服务列表。Spring Eureka 提供了服务治理、负载均衡和容错等能力，使得微服务之间的通信更加简单可靠。

Resilience4j：它提供诸如断路器、失败重试、限流和超时控制等功能，帮助开发人员构建可靠的服务，并处理外部依赖的故障。

Spring Cloud Gateway：用于构建和管理微服务架构中的统一访问入口。它提供路由、负载均衡、安全认证等功能，用于将外部请求转发到适当的微服务上。

Spring Cloud Stream：用于构建分布式消息驱动的应用程序。它提供了统

一的编程模型和抽象层，使得开发人员可以更方便地使用消息中间件进行消息的发送和接收。

数据库技术方面

使用 MongoDB 技术进行

多文档事务：可以在一个事务中同时处理多个文档的读写操作，确保了跨多个集群节点的数据一致性。

数据隔离级别：MongoDB 4 定义了四个数据隔离级别，包括读未提交、读已提交、可重复读和串行化。根据应用程序的需求选择适当的隔离级别，确保并发操作之间的数据一致性和正确性。

数据库加密：它使用透明的数据加密方式，开发者可以轻松配置和管理数据库的数据加密设置。

自适应查询：根据查询工作负载的变化自动调整索引和执行计划，提供更高效的查询性能。

分片负载均衡：通过更智能的路由策略和数据迁移算法，提升了分片集群的可扩展性和性能。

部署技术方面

Docker 是将应用程序及其依赖项打包成一个轻量、独立且可移植的容器。以下是 Docker 的一些关键概念和技术

容器化：将应用程序及其所需的环境（操作系统、库、配置文件等）打包进一个容器中，形成一个隔离的运行环境。这使得应用程序在不同的计算机或云环境中能够以相同的方式运行，减少了与环境相关的问题。

镜像：包含了运行应用程序所需的全部内容。通过基于镜像创建容器，可以实现快速部署和扩展应用程序。

容器管理：用于管理容器的生命周期，包括创建、启动、停止、删除等操作。它还支持容器间的网络通信和数据共享，方便应用程序之间的交互和协作。

缩放和编排：通过容器编排工具来方便地进行应用程序的横向扩展和管理多个容器实例。

Nginx 具有出色的并发处理能力和低系统资源消耗

反向代理：作为反向代理服务器，将客户端请求转发给后端的多个服务器，分担服务器负载，提高系统的稳定性和性能。

静态文件服务：将这些文件直接返回给客户端，减轻了动态请求的服务器压力。

负载均衡：可以将请求分发给后端的多个服务器，均衡各个服务器的负载，提高系统的可靠性和性能。

动态内容缓存：通过缓存动态生成的内容来提高性能，并且可以根据缓存规则进行内容刷新和更新。

三、系统分析

添加统计功能的目的主要是为了提供系统管理员对校园网各个板块的阅读次数进行实时监控和统计分析。

1. 追踪和评估内容受欢迎程度：通过统计功能，系统管理员可以了解校园网中每个板块的阅读次数，从而判断哪些板块或内容受到用户的关注和兴趣。这能够帮助管理员更好地了解用户需求，并据此优化校园网的内容策略。管理员可以发现受欢迎的板块并加强它们的更新与推广，同时对于不受欢迎的板块，可以考虑优化或删除相关内容。

2. 精确的用户行为分析：通过统计功能，管理员可以准确获取特定板块的阅读次数以及用户在该板块的行为数据。这可以帮助管理员了解用户对特定内容的喜好和参与程度，如文章点击率、评论数量等。通过对用户行为的综合分析，管理员可以为用户提供更个性化的推荐和服务，改善用户体验，并为决策提供依据。

3. 内容管理和调整：统计功能使管理员能够清晰了解每个板块的阅读情况，如哪些板块的阅读次数较高、较低，以及潜在的原因。基于这些数据，管理员可以作出相应的调整和决策，如重新定位和整理板块内容，优化页面布局和导航结构，提升用户体验和参与度。

4. 评估广告效果：如果校园网有广告投放或合作推广，统计功能可以为管理员提供展示页的阅读次数数据。管理员可以根据广告展示与点击之间的转化率

来评估广告效果，并根据统计结果进行广告优化和投放调整，提高广告投放的精确性和效果。

要实现其需求需要设计以下步骤：

1. 定义数据模型：首先，您需要为每个板块设计一个数据模型，该模型可以包含板块名称、阅读次数等字段。

2. 统计访问数量：在每个板块页面加载完成时，您可以使用前端技术发送请求到服务器，并在服务器端记录该板块的访问次数。

3. 存储访问信息：在服务器端接收到请求后，将板块的阅读次数加一，并将更新后的信息保存到数据库中或其他持久存储介质中。可以使用数据库来存储不同板块的阅读次数。

4. 建立管理控制台：创建一个系统管理员可以访问的管理控制台界面。在控制台上，您可以提供一个信息统计菜单，用于显示各个板块的阅读次数。

5. 查询数据库并展示数据：当管理员选择信息统计菜单时，从数据库中查询各个板块的阅读次数，并将结果展示给管理员。并且实现处理数据库查询和生成统计报告的功能。

用户界面：

使用 HTML5、CSS3 和 JavaScript 等前端技术构建网站的用户界面。采用响应式设计，确保在不同设备上都能提供良好的用户体验。

设计简洁、直观的导航和布局，使用户能够轻松找到所需的信息和功能。

通过使用富媒体、动画和视觉效果等元素，提升用户界面的吸引力和互动性。

后端架构：

使用 Spring Boot 作为后端框架，以快速搭建和开发网站的后台服务。

使用 MongoDB 作为主要的数据库存储系统，存储网站的动态内容和用户数据。

功能模块：

用户认证与授权：实现用户注册、登录、密码重置等功能，并确保用户信息的安全性。

个人信息管理：允许用户编辑个人资料、修改密码等。

课程管理：提供课程信息的查询、选课、成绩查询等功能。

校园新闻与公告：发布校内新闻、通知和公告，并提供浏览和搜索功能。

社团活动与校园活动：展示校内社团活动、校园活动的信息和报名功能。

图书馆资源：提供图书馆资源的查询、借阅和归还功能。

数据存储与管理：

使用 MongoDB 存储动态内容和用户数据，并根据需求设计合适的数据模型和集合结构。

考虑使用 MongoDB 的分片和副本集等功能，以提高数据的可伸缩性和高可用性。

安全与性能优化：

使用 HTTPS 协议保护用户数据的传输安全。

实施用户身份认证和权限控制，确保只有授权用户可以访问敏感信息。

使用前端和后端的性能优化技术，如压缩和合并资源文件、缓存静态文件等，以提高网站的加载速度和性能。

部署和扩展：

使用云服务提供商（如 AWS、Azure 或阿里云）的托管服务，将网站部署到云平台上，以便于弹性扩展和高可用性的实现。

考虑使用容器化技术，如 Docker 和 Kubernetes，以实现网站的快速部署和管理。

以上是一个基本的设计思路，您可以根据具体需求和业务场景进行进一步的详细设计和实现。同时，还需要考虑网站的可维护性、可扩展性和安全性等方面的因素，以确保网站的稳定运行和持续发展。

四、系统设计

1. 系统架构设计

微服务是一种软件开发架构风格，它将应用程序拆分成一系列小型、独立的服务，每个服务都运行在自己的进程中，并通过轻量级的通信机制进行交互。每个微服务都专注于完成特定的业务功能，并且可以独立部署、扩展和维护。

微服务架构的主要优势之一是它的灵活性和可伸缩性。由于每个微服务都是独立的，开发团队可以使用不同的编程语言、技术栈和数据库来构建它们。这使得团队可以根据需求选择最适合的工具和技术，而不会受到整个应用程序的限制。

此外，微服务架构还提供了更高的可靠性和可维护性。当一个微服务发生故障时，只会影响到该服务，而不会对整个系统产生影响。这使得故障隔离和故障恢复更加容易。此外，由于每个微服务都可以独立部署，开发团队可以更快地进行迭代

和发布新功能，而不会影响其他服务。

API 网关是一个在客户端和后端服务之间充当中间层的服务器。它充当了一个单一的入口点，用于管理和控制对后端服务的访问。

API 网关的主要功能包括：

路由和转发：API 网关接收来自客户端的请求，并将其转发到适当的后端服务。这样，客户端只需与 API 网关通信，而无需直接与后端服务进行交互。

认证和授权：API 网关可以处理认证和授权，确保只有经过身份验证和授权的用户才能访问后端服务。这可以通过使用令牌、API 密钥或其他身份验证机制来实现。

请求转换：API 网关可以对来自客户端的请求进行转换和重写，以适应后端服务的要求。例如，可以进行请求参数的重命名、合并或拆分，以及请求格式的转换。

缓存和限流：API 网关可以缓存后端服务的响应，以减轻后端服务的负载并提高响应速度。此外，它还可以实施限流策略，以防止过多的请求对后端服务造成压力。

监控和分析：API 网关可以收集和记录与请求相关的指标和日志，以便进行监控和分析。这可以帮助开发团队了解系统的性能、可用性和安全性，并进行故障排除和优化。

通过使用 API 网关，企业可以实现对后端服务的集中管理和控制。它提供了一种简化和保护后端服务的方式，并且可以提供更好的性能、安全性和可伸缩性。许多互联网公司和企业都在他们的架构中使用 API 网关来管理他们的 APIs。

异步消息队列是一种用于在分布式系统中进行异步通信的机制。它允许不同的组件或服务之间通过发送和接收消息来进行解耦和协作。

异步消息队列的主要特点包括：

异步通信：发送方将消息发送到队列中，而不需要等待接收方的响应。这种异步通信模式可以提高系统的吞吐量和响应速度。

解耦合：发送方和接收方之间通过消息队列进行通信，它们不需要直接知道彼此的存在。这种解耦合的设计使得系统的组件可以独立地进行开发、部署和扩展。

可靠性：消息队列通常具备持久化机制，可以确保消息在发送和接收过程中不会丢失。即使接收方暂时不可用，消息也会被保存在队列中，等待接收方重新上线后进行处理。

扩展性：由于消息队列是分布式的，可以通过增加队列的消费者来实现系统的横向扩展。这样可以提高系统的处理能力和负载均衡。

可靠性保证：消息队列通常提供一些高级特性，如消息重试、消息顺序保证和事务支持。这些特性可以确保消息的可靠传递和处理。

异步消息队列在许多场景中都有广泛的应用，例如解耦微服务之间的通信、处理大量的异步任务、实现事件驱动架构等。常见的消息队列系统包括 RabbitMQ、Apache Kafka 和 Amazon SQS 等。

异步消息队列的设计和使用需要根据具体的业务需求和系统架构进行合理的选择和配置。

面向文档的数据模型：MongoDB 使用文档来组织和表示数据，文档是一种类似于 JSON 的结构，可以嵌套和包含各种类型的数据。这种灵活的数据模型使得 MongoDB 适用于各种类型的应用程序和数据模式。

可扩展性：MongoDB 支持水平扩展，可以通过添加更多的服务器节点来增加系统的处理能力和存储容量。它还提供了分片（sharding）功能，可以将数据分布到多个节点上，以实现更高的并发和负载均衡。

高性能：MongoDB 具有快速的读写性能，它使用了内存映射技术来加速数据的读取。此外，MongoDB 还支持索引和查询优化，以提供高效的数据访问。

强大的查询功能：MongoDB 提供了丰富的查询语言和功能，包括范围查询、正则表达式、聚合管道等。它还支持全文搜索和地理空间查询，使得开发人员可以轻松地进行复杂的数据查询和分析。

可用性和可靠性：MongoDB 具有复制和故障恢复机制，可以在节点故障时自动切换到备用节点。它还支持数据备份和恢复，以确保数据的安全性和可靠性。

MongoDB 适用于许多场景，包括 Web 应用程序、实时分析、日志存储等。但在选择和使用 MongoDB 时，需要根据具体的业务需求和数据特点进行评估和规划。

容器化部署在分布式微服务中扮演着重要的角色，它可以提供以下作用和优势：

隔离性：容器化部署可以将每个微服务封装在自己的容器中，实现隔离性。这意味着每个微服务运行在自己的运行环境中，不会相互影响，从而提高了系统的稳定性和安全性。

可移植性：容器化部署使用容器技术，将应用程序及其所有依赖项打包成一个可移植的镜像。这样，可以在不同的环境中轻松部署和运行微服务，无需担心环境差异和依赖问题。

弹性伸缩：容器化部署可以根据实际需求进行弹性伸缩。通过容器编排工具，可以自动扩展或缩减微服务的实例数量，以适应负载的变化，提高系统的可用性和性能。

简化部署和管理：容器化部署提供了一种简化的部署和管理方式。通过容器编排工具，可以自动化地部署、更新和管理微服务，减少了手动操作和人为错误的风险。

容器化部署的原理主要基于容器技术，如 Docker。Docker 使用 Linux 容器技术，通过隔离和资源管理机制，将应用程序及其依赖项打包成一个镜像。镜像可以在任何支持 Docker 的环境中运行，提供了一致的运行环境。

在分布式微服务架构中，通常使用容器编排工具，如 Kubernetes，来管理和编排多个容器实例。容器编排工具可以自动化地创建、启动、停止和监控容器，以及处理容器之间的网络通信和负载均衡。

通过容器编排工具，可以定义和管理微服务的部署配置、扩缩容策略、服务发现和负载均衡规则等。这样可以实现分布式微服务的高效部署、弹性伸缩和自动化管理。

2. 数据库设计

MongoDB 4 在基于分布式微服务设计中扮演着重要的角色，它提供了一些功能和特性，有助于构建可靠、可扩展的分布式系统。

以下是 MongoDB 4 在基于分布式微服务设计中的作用和原理：

分片（Sharding）：MongoDB 4 引入了更强大的分片功能，可以将数据水平划分为多个分片，存储在不同的节点上。这样可以实现数据的分布式存储和处理，提高系统的可扩展性和性能。

副本集（Replica Set）：MongoDB 4 继续支持副本集，通过在多个节点上复制数据，提供数据的冗余和高可用性。副本集可以自动进行故障检测和故障转移，确保系统在节点故障时仍然可用。

分布式事务：MongoDB 4 引入了多文档事务的支持，可以在分布式环境中实现跨多个文档的原子操作。这对于需要保持数据的一致性和完整性的应用程序非常重要，例如在微服务架构中处理复杂的业务逻辑。

查询路由（Query Routing）：MongoDB 4 的分片功能还提供了查询路由的支持，可以自动将查询请求路由到正确的分片上。这样可以减少网络传输和数据处理的开销，提高查询的效率和性能。

数据分布和负载均衡：MongoDB 4 的分片功能可以根据数据的分布情况，自动将数据均匀地分布在不同的分片上，实现负载均衡。这样可以避免某些分片上的数据过载，提高系统的整体性能。

3. 类设计

在基于分布式微服务的架构中，类（Class）仍然扮演着重要的角色，它们在不同的微服务中用于定义和实现各自的功能和业务逻辑。以下是类在基于分布式微服务中的作用和原理：

微服务实现：类用于实现微服务中的具体功能和业务逻辑。每个微服务通常都有自己的代码库和部署单元，其中包含了多个类。这些类负责处理请求、执行业务逻辑、访问数据库等操作，从而实现微服务的具体功能。

服务接口：类可以作为服务接口的实现类。在基于分布式微服务的架构中，服务接口用于定义不同微服务之间的通信协议和数据交互方式。类实现了这些接口，提供了具体的服务功能，并通过网络进行调用和交互。

数据传输对象（DTO）：类可以用作数据传输对象，用于在不同微服务之间传递数据。DTO 类通常只包含属性和相关的 getter 和 setter 方法，用于在不同服务之间传递数据的载体。通过序列化和反序列化，DTO 类可以在不同的服务之间进行数据传输。

领域模型：类可以用于定义领域模型，即业务领域中的核心概念和对象。领域模型类用于表示业务实体、值对象、聚合根等，它们包含了业务逻辑和行为，用于实现业务规则和操作。

在基于分布式微服务的架构中，类的原理与传统的面向对象编程相似。每个微服务都有自己的代码库和运行环境，其中包含了多个类。这些类通过网络进行通信和交互，实现各自的功能和业务逻辑。

通过服务注册与发现、负载均衡、消息队列等技术，不同微服务之间可以进行通信和协作。类通过接口和协议定义了服务之间的通信方式，通过序列化和反序列化实现数据的传输。每个微服务独立运行，可以独立部署和扩展，从而实现分布式系统的高可用性和可伸缩性。

4. 界面设计

在基于分布式的微服务架构中，界面设计（UI Design）仍然扮演着重要的角色，它们用于实现用户与系统之间的交互和信息展示。以下是界面设计在基于分布式微服务中的作用和原理：

用户体验：界面设计通过合理的布局、交互方式和视觉效果，提供良好的用户体验。在分布式微服务架构中，不同的微服务可能涉及不同的功能和业务逻辑。通过统一的界面设计，可以使用户在不同的微服务之间切换时，保持一致的使用体验，提高用户满意度和系统可用性。

信息展示：界面设计用于将系统中的数据和信息以可视化的方式展示给用户。在分布式微服务架构中，不同的微服务可能存在不同的数据源和业务逻辑。通过界面设计，可以将来自不同微服务的数据整合并展示给用户，提供全局的信息视图和操作入口。

交互操作：界面设计通过按钮、表单、菜单等交互元素，实现用户与系统之间的交互操作。用户可以通过界面设计提供的交互元素，向系统发送请求、执行操作、提交数据等。界面设计需要考虑用户的操作习惯和需求，提供直观、易用的交互方式，使用户能够方便地与分布式微服务进行交互。

前端技术栈：在基于分布式微服务的架构中，界面设计需要使用前端技术栈进行实现。常见的前端技术栈包括 HTML、CSS、JavaScript 等，还可能使用一些前端框架和库，如 React、Angular、Vue.js 等。通过前端技术栈，可以实现界面的布局、样式、交互逻辑等。

在分布式微服务架构中，界面设计需要考虑以下原理：

服务边界：界面设计需要根据不同的微服务划分服务边界。不同的微服务可能涉及不同的功能和数据，界面设计需要根据微服务的边界进行划分和组织，以实现功能的模块化和解耦。

API 调用：界面设计需要通过调用微服务暴露的 API 来获取数据和执行操作。界面设计需要根据 API 的定义和约定，进行数据的获取、提交和展示。

前后端分离：在分布式微服务架构中，界面设计通常采用前后端分离的方式。前端负责界面的展示和交互，后端微服务负责数据的处理和逻辑的执行。前后端通过 API 进行通信，实现数据的传递和交互。

用户认证和授权：在分布式微服务架构中，界面设计需要考虑用户认证和授权的问题。用户需要通过界面进行登录和身份验证，系统需要根据用户的身份和权限，对不同的微服务进行访问控制。

五、系统实现

是一个统计服务类的实现，提供对菜单统计信息的增加、更新和查询功能。

在方法 addMenuNumber()中，首先通过调用 selectById()方法查询是否已存在相同 ID 的统计信息。如果存在，则将查询到的统计信息取出，并将菜单次数加 1，然后调用 updateById()方法更新统计信息；如果不存在，则直接保存新的

统计信息。

• 方法 updateById()用于更新指定 ID 的统计信息，它调用了 statisticsMapper.save()方法来执行更新操作。

StatisticsService 类是业务逻辑层，通过调用 StatisticsMapper 类的方法实现数据的增、删、改、查等操作，并提供了一些方法，如 addMenuNumber()、updateById()、listStatistics()和 selectById()，供 StatisticsController 类进行调用。

核心代码块如下：

```
@Service
public class StatisticsService {
    @Autowired
    private StatisticsMapper statisticsMapper;
    public void addMenuNumber(Statistics statistics) {
        Optional<Statistics> existingStatistics = selectById(statistics.getId());
        if (existingStatistics.isPresent()) {
            statistics = existingStatistics.get();
            statistics.setMenuNumber(statistics.getMenuNumber() + 1);
            updateById(statistics);
        } else {
            statisticsMapper.save(statistics);
        }
     }
```

```java
@Service
public class StatisticsService {

    @Autowired
    private StatisticsMapper statisticsMapper;

    /**
     * 增加菜单次数
     * 如果统计信息存在，则更新菜单次数；否则，保存新的统计信息。
     *
     * @param statistics 统计实体对象
     */
    public void addMenuNumber(Statistics statistics) {
        Optional<Statistics> existingStatistics = selectById(statistics.getId());
        if (existingStatistics.isPresent()) {
            statistics = existingStatistics.get();
            statistics.setMenuNumber(statistics.getMenuNumber() + 1);
            updateById(statistics);
        } else {
            statisticsMapper.save(statistics);
        }
    }
```

• 方法 listStatistics()返回所有的统计信息列表，通过调用 statisticsMapper.findAll()实现。

核心代码块如下：

```
public void updateById(Statistics statistics) {

    statisticsMapper.save(statistics);

}

public List<Statistics> listStatistics() {

    return statisticsMapper.findAll();

 }
```

```java
    /**
     * 根据ID更新统计信息
     *
     * @param statistics 统计实体对象
     */
    public void updateById(Statistics statistics) {
        statisticsMapper.save(statistics);
    }

    /**
     * 获取所有的统计信息
     *
     * @return 统计信息列表
     */
    public List<Statistics> listStatistics() {
        return statisticsMapper.findAll();
    }
```

• 方法 selectById()根据给定的 ID 查询统计信息，它调用了 statisticsMapper.findById()方法获取查询结果。

核心代码块如下：

```
    public Optional<Statistics> selectById(String id) {
        return statisticsMapper.findById(id);
    }
}
```

```
/**
 * 根据ID查询统计信息
 *
 * @param id 统计信息ID
 * @return 查询到的统计信息（如果存在）
 */
public Optional<Statistics> selectById(String id) {
    return statisticsMapper.findById(id);
}
```

交互过程如下：

1. 前端通过发送 HTTP 请求访问 StatisticsController 中的/listStatistics 接口。

2. statisticsController 中的 listStatistics()方法拦截该请求，并调用 statisticsService 对象的 listStatistics()方法获取统计数据列表。

3. statisticsService 对象将其业务逻辑委托给 StatisticsMapper，再返回数据给 StatisticsController。

4. StatisticsController 将统计数据列表转化为 JSON 格式然后响应给前端。

新增或更新数据的过程如下：

1.前端通过发送 HTTP 请求提交新增或更新的数据到 StatisticsController 中的/add 或/updateById 接口。

2. statisticsController 中的 add()方法或 updateById()方法拦截该请求，并调用统计服务 StatisticsService 实现对数据库的操作。

3. 统计服务 StatisticsService 根据对应的需求调用 StatisticsMapper 层

中的方法完成对数据库中数据的添加或更新。

4. 更新或添加成功之后，StatisticsController 响应给前端一个 HTTP 状态码和相关信息。

六、系统部署

按照以下步骤进行操作：

1. 安装 Docker：首先需要在目标服务器或主机上安装 Docker 引擎。具体的安装方法可以参考 Docker 官方文档或适用于你操作系统的指南。

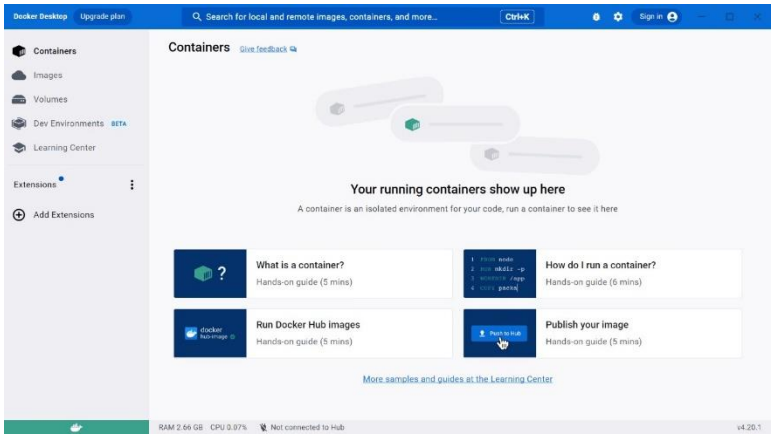
2. 创建 Docker 镜像:将你的应用程序和其依赖项打包成一个 Docker 镜像。Docker 镜像是一个可执行的文件，其中包含了运行你的应用所需的所有内容。你可以编写一个 Dockerfile 来定义创建镜像的步骤，然后使用 Docker 命令将其构建为镜像。确保在镜像中配置好应用程序的运行环境和相关设置。

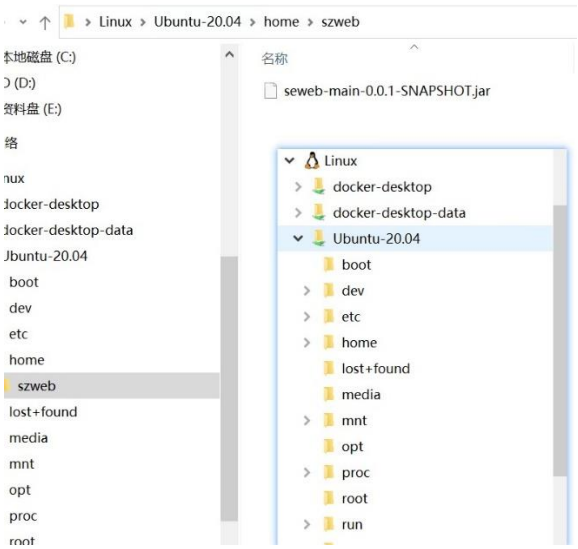
3. 运行 Docker 容器：使用 Docker 命令在主机上运行你的 Docker 容器。在运行容器时，可以通过指定端口映射、卷挂载等方式设置容器与主机之间的交互方式。确保你的容器能够监听正确的端口，并与其他容器或服务进行通信。

4. 配置 Nginx 反向代理：安装并配置 Nginx 作为反向代理服务器，用于接收来自客户端的请求并将其转发给 Docker 容器内部运行的应用程序。在 Nginx 的配置文件中，需要添加代理转发的规则，并配置负载均衡策略（如果有多个容器实例）。同时，根据需求配置 SSL/TLS 加密、访问控制等功能。

5. 启动 Nginx 服务：启动 Nginx 服务，使其开始监听指定的端口并接收来自客户端的请求。确保 Nginx 配置文件的路径正确，并且没有语法错误。可以使用命令行工具或系统服务管理工具来启动 Nginx。

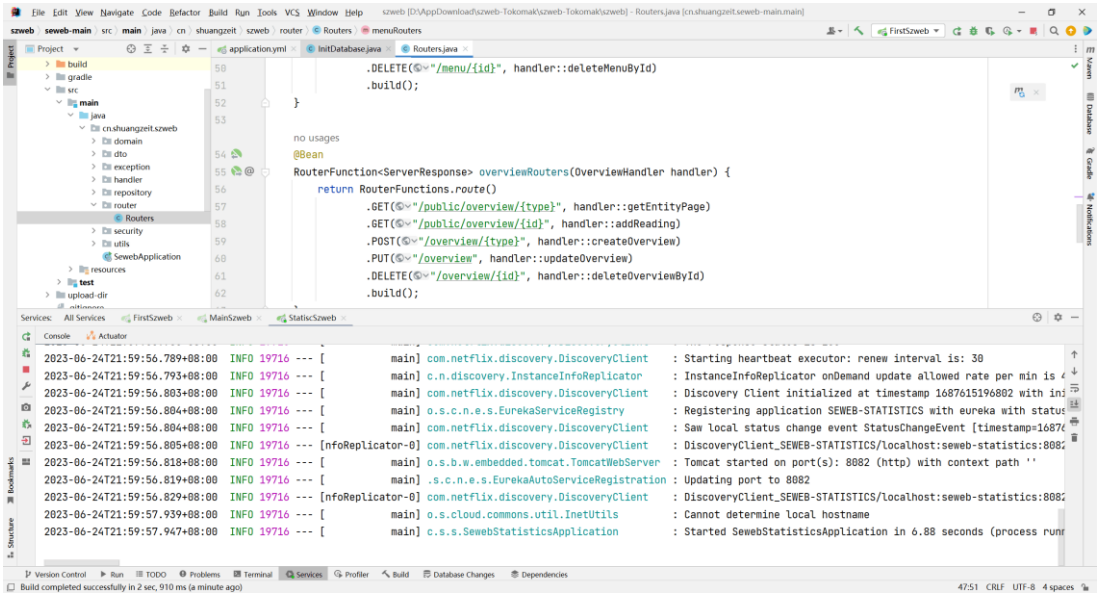
6. 测试应用程序：使用浏览器或其他 HTTP 客户端工具，发送请求到 Nginx 监听的端口。Nginx 会将请求转发给 Docker 容器内运行的应用程序，并返回响应结果。确保应用程序能够正常运行，并通过 Nginx 反向代理进行访问。



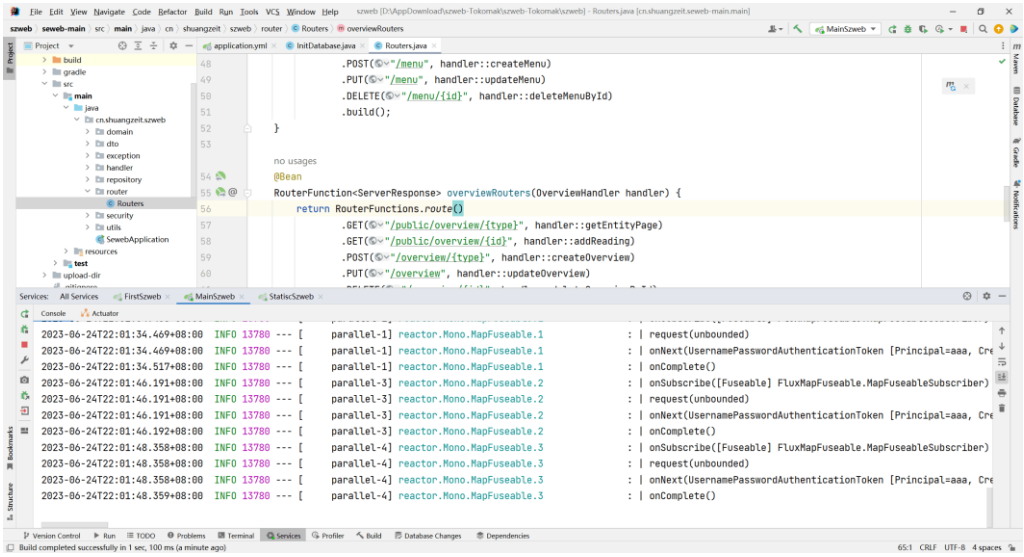


七、系统测试

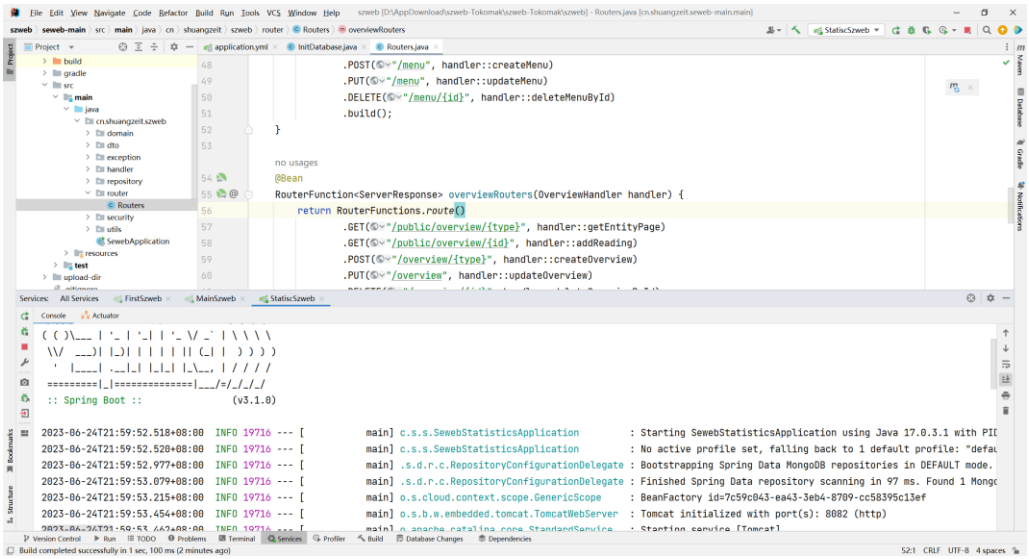
FirstSzweb 类的 spring 启动界面如下



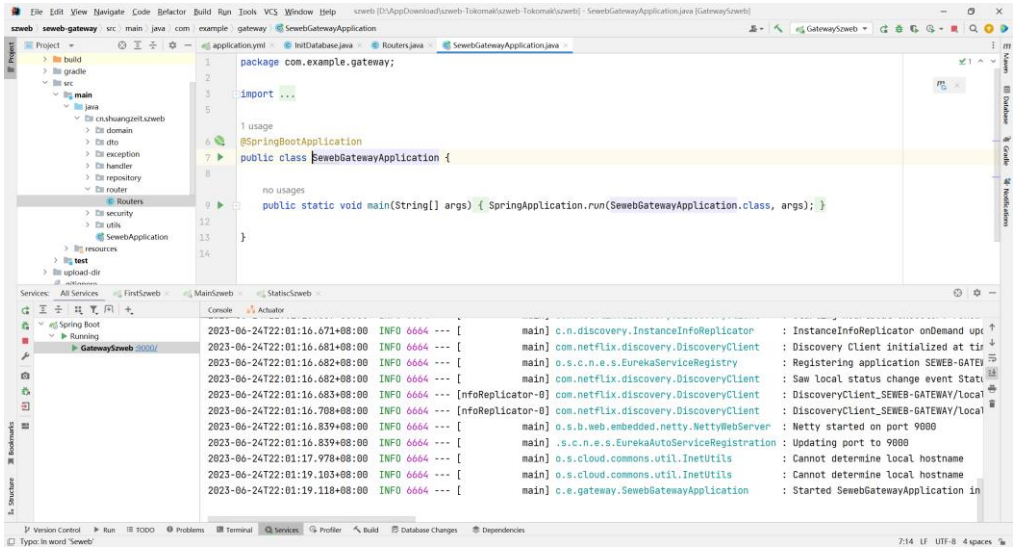
MainSzweb 类的启动界面如下



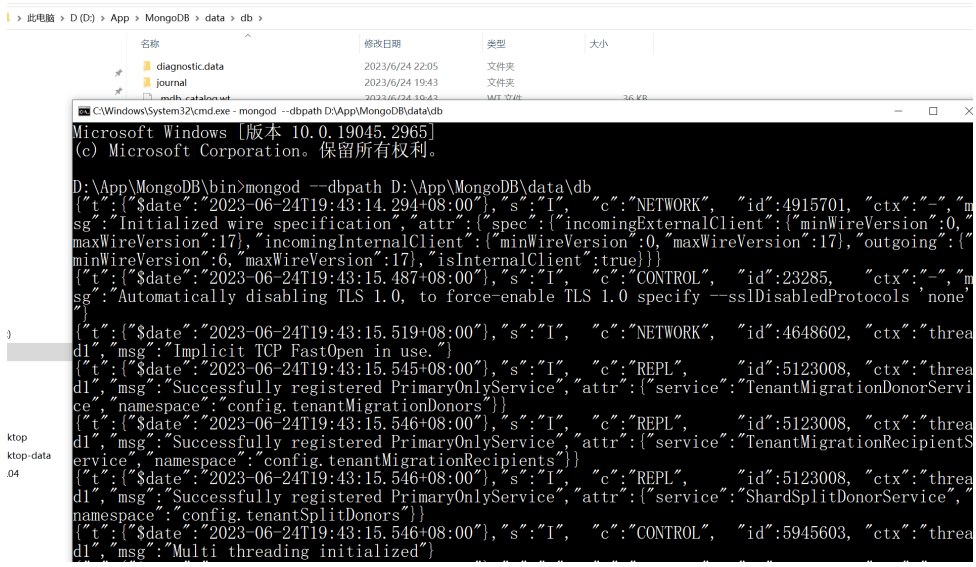
StaticSzweb 启动界面如下



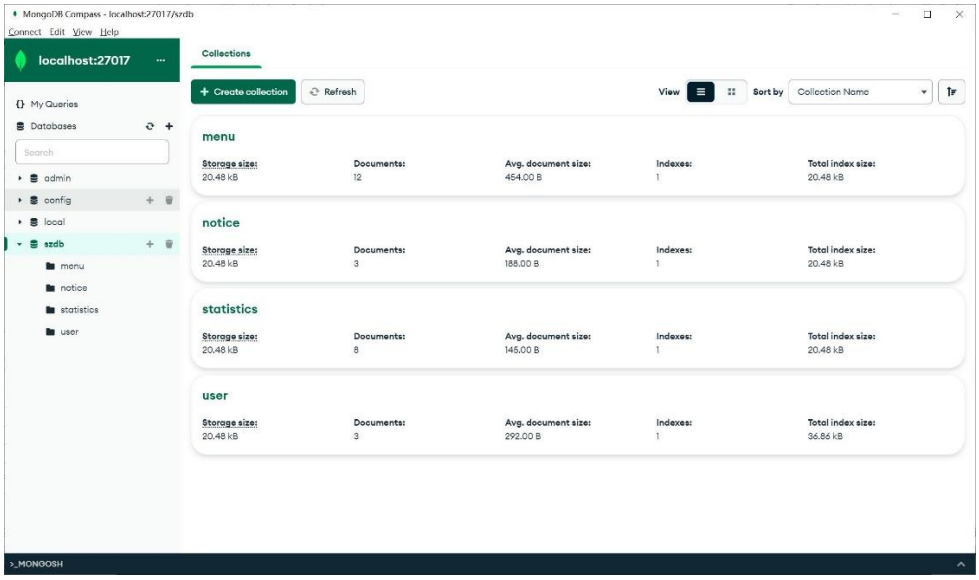
GatewaySzweb 类的启动界面如下

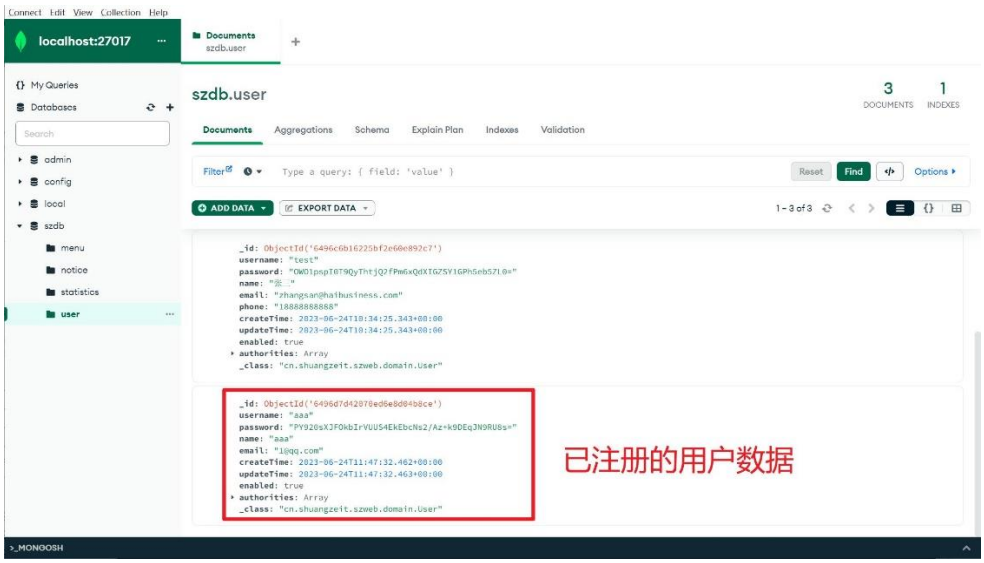


MongoDB 数据库启动界面如下

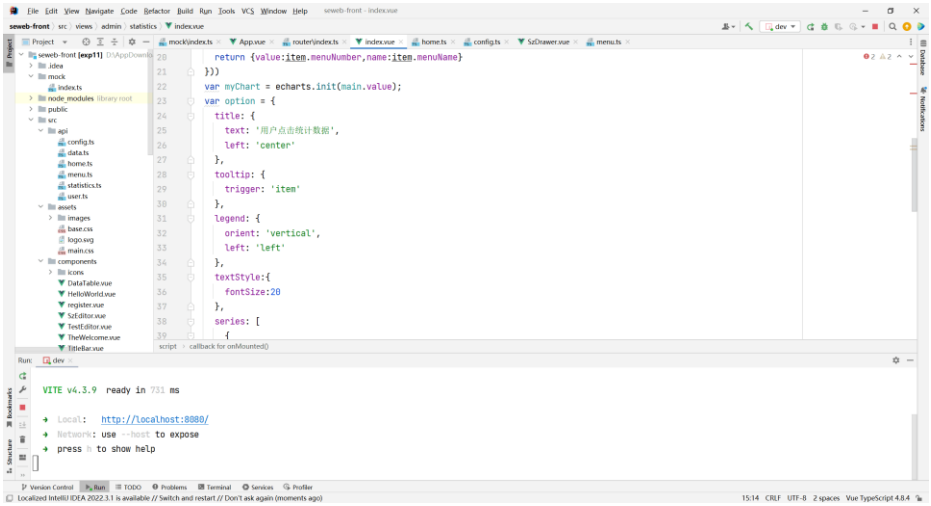


MongoDB 数据库主界面（已添加 szdb 数据库）



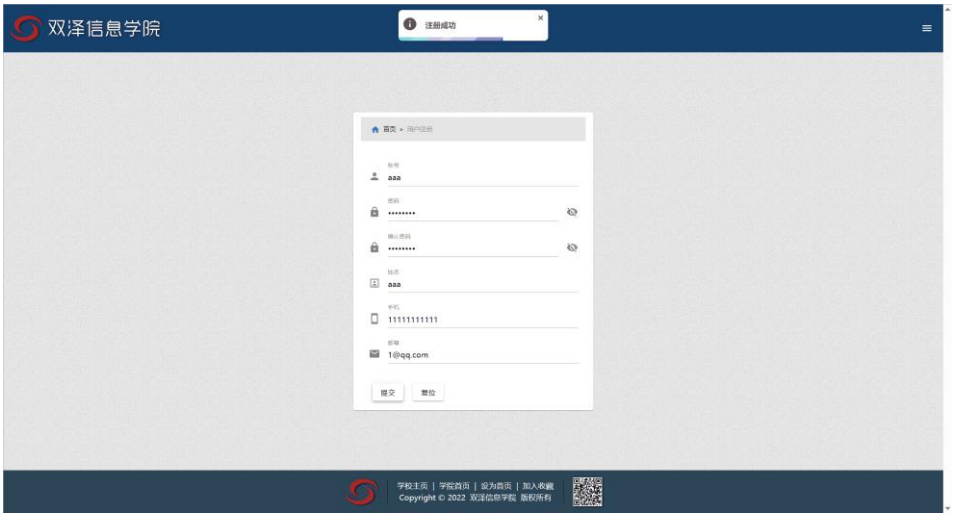


前端项目启动界面如下

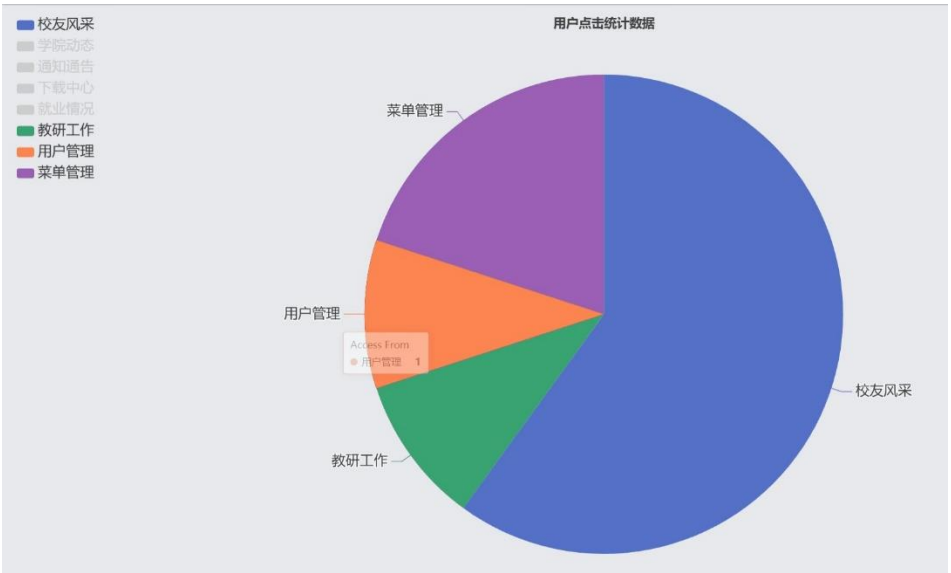


输入字段的数据验证如下

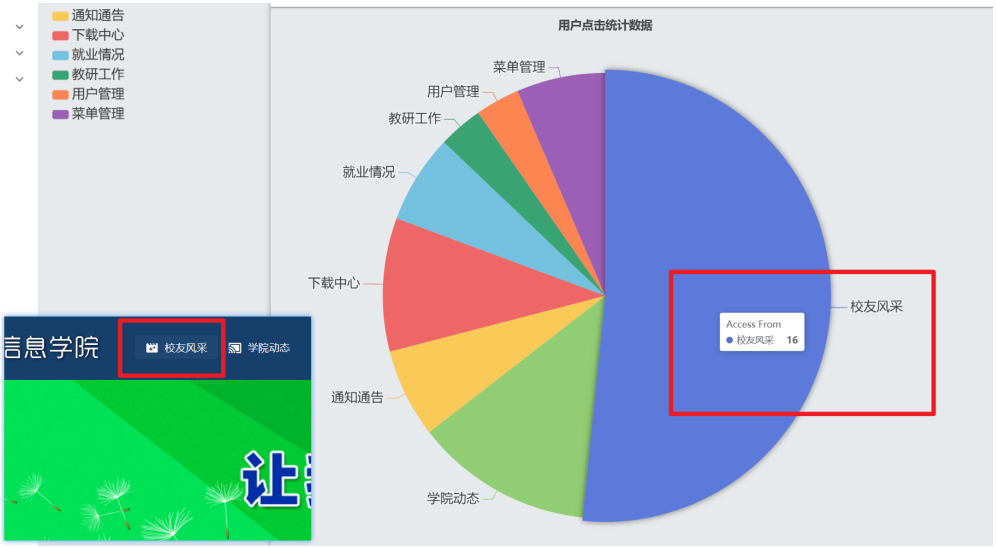
注册成功截图如下



登录成功，通过饼图展示用户点击数据



点击十次校友风采组件后，进入管理控制台，显示校友风采次数增加 10 次



八、存在问题及解决方法/方案

在设计和实现基于分布式微服务架构的校园网系统时遇到以下一些问题，例如在 Vue 中，当 URL 路径地址发生切换时，可能会遇到以下路由导航问题。

路由组件未销毁：由于组件的复用，旧的路由组件可能没有被销毁，导致数据状态混乱或内存泄漏。使用 Vue Router 提供的导航守卫功能，在路由切换前进行组件的销毁或数据重置。

异步数据加载延迟：如果需要异步加载数据，可能会导致页面内容闪烁或加载延迟。使用 Vue Router 提供的导航守卫功能，在路由切换前显示加载状态或

占位符，等待数据加载完成后再渲染页面。

路由参数传递问题：如果需要传递参数给目标路由组件，可能会遇到参数丢失或不更新的问题。使用 Vue Router 提供的路由参数功能，通过 params 或 query 传递参数。确保在路由切换时，参数能够正确传递给目标路由组件，并在组件内部及时响应参数的变化。

路由嵌套问题：如果存在路由嵌套的情况，可能会导致嵌套路由的组件未被正确渲染或数据未更新。使用 Vue Router 提供的嵌套路由功能，确保在父子路由切换时，子路由的组件能够正确渲染和更新。

路由权限控制问题：如果需要进行路由权限控制，可能会遇到未授权访问或无法跳转的问题。使用 Vue Router 提供的导航守卫功能，在路由切换前进行权限验证。

服务拆分和划分问题：如何合理地拆分和划分微服务，进行领域驱动设计将系统按照业务边界进行拆分。同时，考虑服务的粒度和职责，避免服务过大或过小。使用微服务设计原则确保服务的独立性和可扩展性。

服务间通信问题：实现微服务之间的通信和协作，确保数据的一致性和可靠性，使用适当的通信协议和技术确保服务间的数据传输安全和可靠性。

分布式事务问题：当涉及多个微服务的操作时，保证分布式事务的一致性和可靠性，使用分布式事务管理器，来协调多个微服务的事务操作。确保分布式事务的一致性。

服务注册与发现问题：实现微服务的注册与发现，以便于服务的动态扩展和负载均衡，使用服务注册与发现组件，通过心跳机制和健康检查，确保服务的可用性和动态扩展。

高可用性和容错性问题：保证系统的高可用性和容错性，避免单点故障和服务不可用，使用负载均衡器，如 Nginx 或 Spring Cloud Gateway，将流量均衡到多个实例上。

数据一致性问题：当涉及到多个微服务的数据操作时，保证数据的一致性，将数据变更事件发布到消息队列中，由其他订阅者进行相应的操作。保证数据的一致性。

敏感数据访问问题：管理控制台通常包含敏感数据，如用户个人信息、学生

成绩、教职工工资等。未经授权的用户可以通过访问管理控制台获取这些敏感数据，导致隐私泄露。

九、心得体会

在我学习和实践分布式系统和云计算的过程中，我对这两个概念有了更深入的理解和体会。分布式系统是一种通过网络将多个计算机连接起来，共同完成一个任务的技术，而云计算则是一种基于互联网的计算模式，它将资源、服务和应用通过互联网提供给用户。

首先，我认为分布式系统是一种非常有用的技术，它可以帮助我们解决大规模数据处理的问题。具体来说，分布式系统可以将大规模的数据分成小块，然后分配给不同的计算机进行处理，最后再将处理结果合并起来。这种技术可以大大提高数据处理的速度和效率，同时也可以减少计算机的负载，提高系统的可用性和可靠性。

其次，云计算是一种非常灵活和可扩展的计算模式。云计算可以帮助我们按需分配计算资源，从而实现资源的动态管理和调度。这样，我们就可以根据实际需求灵活地调整计算资源的大小，从而避免资源的浪费和过度配置。此外，云计算还可以提供弹性的扩展能力，使我们能够更好地应对业务量的增长和变化。

在实践中，我也遇到了一些分布式系统和云计算的技术挑战。例如，分布式系统的数据一致性、负载均衡和容错等问题需要我们仔细地研究和解决。而在云计算中，如何保证计算资源的隔离性和安全性、如何提供可靠的网络连接等问题也需要我们进行深入的思考和探索。

在分布式的学习中与课程设计的制作中，web 应用技术也起到了关键的作用，给我留下了深刻的印象：

首先，良好的编程实践是至关重要的。这意味着要注重代码的可读性、可维护性和可扩展性。代码应该清晰易懂，遵循一致的命名和格式规范。要避免使用过多的复杂语法，而应优先考虑简单易懂的实现。

其次，要始终考虑安全问题。在 Web 开发中，安全是一个核心问题。必须时刻关注输入验证、防止注入攻击、使用安全的密码存储等。此外，还要遵循最小化权限原则，避免给予应用程序过多的权限。

再次，要注重性能。Web 应用程序的性能直接影响用户体验。因此，要优化关键代码路径，减少资源占用，使用高效的算法和数据结构。还可以使用缓存、压缩和负载均衡等技术来提升性能。

最后，要不断学习和探索新技术。Web 技术发展迅速，每年都有新的技术和框架涌现。为了保持竞争力，必须保持对新技术的关注，并定期评估和尝试新的工具和技术。

分布式系统和云计算为我们提供了非常有用的技术工具，可以帮助我们更好地处理大规模数据、实现资源的动态管理和调度。当然，在实际应用中，我们也需要根据具体的业务场景和需求进行选择和优化，从而更好地发挥分布式系统和云计算的优势和价值。展望未来，我认为分布式系统和云计算将会越来越普及和应用。随着技术的不断进步和发展，这两个领域将会出现更多的创新和变革，例如分布式系统中的区块链技术、云计算中的容器技术和微服务技术等。同时，我也希望能够继续深入学习和实践这两个领域的技术，并将其应用于实际业务中，实现业务价值的提升和创新。

致谢

经过三年的生活和学习，经过老师和同学的帮助，我们在各方面都有了很大的进步。首先，最需要感谢的是我们的分布式与云计算课程设计指导老师刘金羽，刘老师不但在生活、工作和学上不断给我们指导，使我们掌握了很多知识，而且老师知识渊博，思维敏捷，治学态度严谨，对学生关怀无微不至，所有的这些都给我们留下了深刻的印象，为我们今后的人生发展打下了良好的基础，在此，我们再一次对刘金羽老师表示由衷的感激。感谢老师在学习上和生活上对我的悉心指导和无微不至的关怀，感谢老师给了我锻炼的机会和良好的学习环境，感谢老师在批改课程设计时倾注的每一滴心血。同时，我还要感谢我的舍友们，感谢他们在制作课程设计过程中提供了很多指导和帮助，在此表示深深的感谢。我还要感谢我的同学，感谢他们在生活上和学习上给予我的热情帮助，使我生活增添了不少色彩。最后还要感谢我的家人，是他们给了我永恒的动力，让我认真完成每学习任务。课设的顺利完成，离不开家人的理解和支持，离不开朋友的关心，离不开老师的谆谆教诲，因此，感谢我的老师对我的鼓励和支持，感谢所有提供了各种形式支持和帮助的朋友，向他们表示我最诚挚的感谢。

分布式与云计算课程设计小组

2023 年 6 月于海大儋州

参考文献

[1] 田锐,喻婧.分布式数据库技术及应用分析[J].长江信息通信,2023,36(05):165-167.

[2] 郭强.基于分布式时分多址接入的无线网络时隙调配方法[J].电脑知识与技术,2023,19(04):80-82.DOI:10.14004/j.cnki.ckt.2023.0177.

[3] 秦健,韩斌,崔芸.分布式数据库技术在大数据中的应用[J].电脑知识与技术,2022,18(30):54-56+70.DOI:10.14004/j.cnki.ckt.2022.1961.

[4] 陈楠,赵建军,钟平,黄勇军,陈天.基于云原生的分布式物联网操作系统架构研究[J].电信科学,2022,38(07):146-156.

[5] 梁姝娜. 大数据的分布式自适应支持向量回归[D].青岛大学,2022.DOI:10.27262/d.cnki.gqdau.2022.001955.

[6] 杨浩. 基于网格的分布式 Web 服务发现技术研究[D].华北电力大学(北京),2021.DOI:10.27140/d.cnki.ghbbu.2021.000758.

[7] 杨浩. 基于网格的分布式 Web 服务发现技术研究[D].华北电力大学(北京),2021.DOI:10.27140/d.cnki.ghbbu.2021.000758.

[8] 袁一. 分布式测试系统 Web 服务封装及管理系统的设计与实现[D].电子科技大学,2020.DOI:10.27005/d.cnki.gdzku.2020.003790.