

**本科毕业设计（论文）**

**题 目：** 微服务项目设计与实现

**学生姓名：** 黄志鹏

**学 院：** 计算机学院

**专 业：** 软件工程

**学 号：** 2019120310511

**班 级：** 软件工程5班

**指导教师：** 王乌云

**提交日期：** yyyy年 mm月dd日

**年 月 日**



本科毕业设计（论文）诚信声明

本人郑重声明:所呈交的本科毕业设计（论文）,是本人在指导老师的指导下，独立进行研究工作所取得的成果，成果不存在知识产权争议，除文中已经注明引用的内容外，本设计（论文）不含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学生签名： 时间： 年 月 日



关于毕业设计（论文）使用授权的说明

本人完全了解广东工商职业技术大学关于收集、保存、使用学位论文的规定，即:

1. 按照学校要求提交学位论文的印刷本和电子版本;
2. 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务;
3. 学校可以采用影印、缩印、数字化或其它复制手段保存论文;本人同意上述规定。

学生签名： 时间： 年 月 日

摘 要

随着互联网技术的快速发展，微服务架构成为了现代软件开发中的热门话题。微服务架构通过将大型应用系统拆分成多个独立的、相互协作的服务模块，以实现系统的高可扩展性、高可维护性和高可用性等特点，因此在企业应用开发中得到了广泛的应用。然而，当前互联网领域中微服务架构还面临着诸多挑战和困难，例如如何进行服务拆分、服务治理、服务监控、服务安全等方面的问题，这些都需要得到深入研究和解决。

本文旨在通过对微服务项目的架构设计和实现进行研究，帮助开发人员更好地理解和掌握现代软件架构的设计理念和方法。本文通过采用项目实战、文献、书籍等方法，分析和探讨微服务架构的设计和实现，探索微服务架构在提高软件开发效率、系统性能、可维护性和可扩展性等方面的作用，从而实现对互联网企业的发展和竞争力提升。

本文主要涉及的研究内容包括：微服务、SpringCloud Alibaba组件、项目实战。通过这些内容的探索和实践，本文得出了一套可行的微服务架构设计和实现方案，能够提高应用系统的可扩展性、可维护性和可用性等特点，满足互联网应用开发的需求。

在未来，微服务架构还将继续在互联网领域发挥重要作用。未来的研究将集中于解决微服务架构中的瓶颈问题，例如服务治理、服务监控、服务安全等方面的问题，提高微服务架构的可用性、可靠性和稳定性。

**关键词：**微服务架构、服务拆分、服务治理、服务监控、服务安全、SpringCloud Alibaba

**目 录**

[摘 要 4](#_Toc20041)

[第一章 绪论 7](#_Toc32337)

[1.1 背景 7](#_Toc19542)

[1.2 目的 7](#_Toc8991)

[1.3 内容 7](#_Toc2729)

[第二章 微服务 9](#_Toc17674)

[2.1单体应用架构 9](#_Toc25817)

[2.2微服务架构 10](#_Toc29385)

[2.2.1微服务架构定义 11](#_Toc12880)

[2.2.2微服务架构优缺点 12](#_Toc25447)

[第三章 微服务组件 13](#_Toc15891)

[3.1 服务注册和配置中心 14](#_Toc28013)

[3.1.1 Nacos简介 15](#_Toc17844)

[3.1.2 Nacos的快速启动 17](#_Toc23695)

[3.2 服务间通信框架 18](#_Toc218)

[3.2.1 Dubbo简介 18](#_Toc16788)

[3.2.2 Dubbo基本工作流程 19](#_Toc32261)

[3.2.3 Dubbo快速入门Demo 21](#_Toc30342)

[第四章 项目实战 23](#_Toc18039)

[4.1 需求分析 23](#_Toc8567)

[4.2 技术选型 24](#_Toc2669)

[4.3 架构设计 25](#_Toc4700)

[4.3.1 表设计 25](#_Toc24584)

[4.3.2 模块设计 27](#_Toc31948)

[4.4 开发文档 28](#_Toc12951)

[4.5 功能实现和测试 29](#_Toc21668)

[4.6 部署调试 40](#_Toc11555)

[附录 42](#_Toc3296)

[参考文献 42](#_Toc622)

[致谢 42](#_Toc12840)

第一章 绪论

**1.1 背景**

分布式的微服务架构在当前互联网领域已经广泛应用，并取得了显著的成果和效益。其应用背景和意义主要表现在以下几个方面：

首先，随着互联网行业的快速发展，传统的单体式应用架构已经无法满足当前复杂的业务需求和大规模的用户量，因此需要一种更加灵活和高效的应用架构。微服务架构正是针对这一需求而产生的，通过将单体应用拆分成多个小的服务单元，实现了更好的解耦、更灵活的部署和更高的可扩展性。

其次，分布式的微服务架构还能够提高互联网应用的可靠性和容错性。由于服务之间的解耦和隔离，一个服务出现问题不会影响其他服务的正常运行，从而降低了整个系统的风险和故障率。

然而，分布式的微服务架构在实际应用过程中仍然面临着一些挑战和需求：

1. 微服务架构带来了服务间的复杂性和管理难度，需要通过相应的工具和平台来支持和管理。
2. 微服务架构要求服务的部署和运行环境必须保持一致，需要解决版本控制和依赖管理的问题。
3. 服务之间的通信和数据传输需要保证安全和可靠性，需要建立相应的安全机制和协议。
4. 微服务架构的设计和实现需要结合具体的业务需求和技术特点，不能简单地追随潮流或者模仿他人，需要进行深入的分析和实践。

**1.2 目的**

我们通过上一节，也明白了微服务其应用场景和意义，所以本文的研究目的是让广大开发人员更好地理解和掌握现代软件架构的设计理念和方法，以及从实操上实践架构微服务项目;在未来研究打算以解决微服务架构在实际应用过程中仍然存在的挑战和需求。

**1.3 内容**

本文介绍微服务架构的背景和为什么在分布式系统开发中微服务架构将逐渐替代单体架构。同时介绍了Spring Cloud Alibaba提供的微服务组件和解决方案，帮助学者系统地了解Spring Cloud Alibaba的相关知识，并掌握微服务架构应用的设计、开发、部署和运维等知识。最后，通过实战让学者彻底理解现代软件架构的设计理念和方法。

本文主要章节只有3章：

1. 微服务：
2. Springcloud alibaba组件：
3. 项目实战：
4. 微服务

**2.1****单体应用架构**

单体应用架构是一种常见的应用程序架构，它由单独的代码库和运行时环境组成，包含应用程序的所有组件。在Java开发中，一个典型的单体架构应用通常将应用的所有功能打包在一个WAR文件中，并在应用服务器（如Tomcat）中部署和运行。如图1所示。

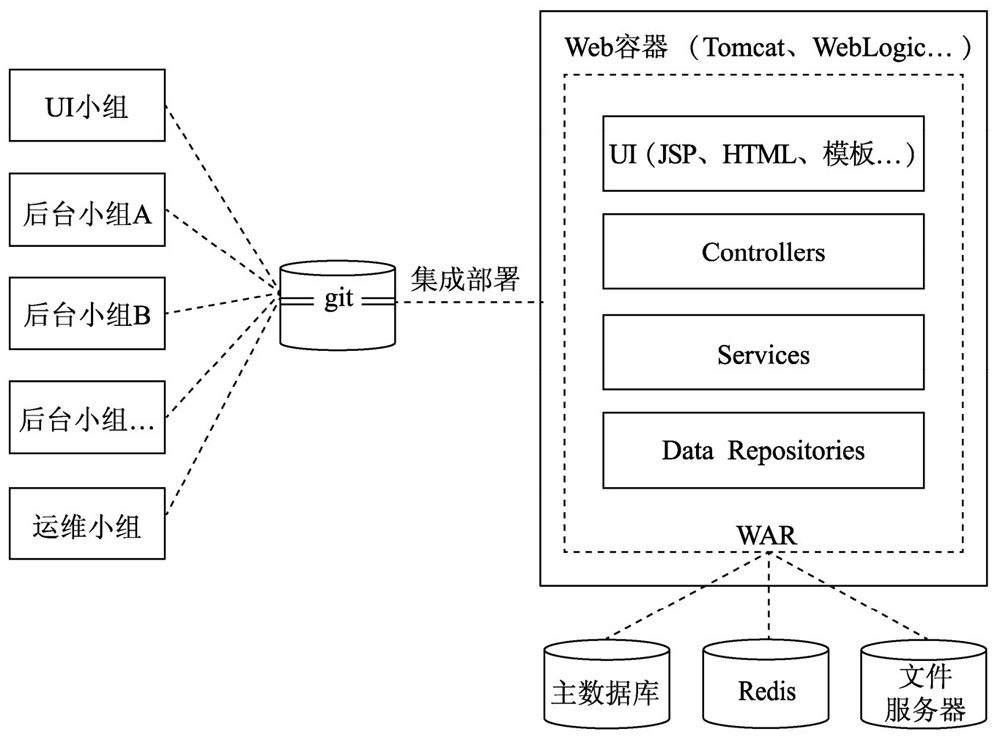


图 1 单体应用的架构图

单体应用架构是一种将整个应用程序封装在一个单一的可执行文件或单个代码库中的架构。在单体应用架构中，整个应用程序运行在一个单一的运行时环境中，这包括应用程序的所有组件和依赖项。虽然单体应用程序可以具有许多不同的组件和功能，但是它们都共享相同的代码库和运行时环境。

单体应用架构的优点在于易于理解和开发、更容易部署和维护、更少的复杂性和更好的性能。这些优点使得单体应用架构在应用程序规模小时是一种很好的选择。

然而，随着应用程序的规模和复杂性的增加，单体应用架构的缺点也开始暴露出来。单体应用架构的可扩展性是受限的，当负载增加时，无法水平扩展应用程序。单体应用架构的可维护性差，部署难度高，代码库会变得越来越庞大，每次需要修改、增加新的功能时都需要更多的协调和测试，应用程序部署的时间和成本也会增加。单体应用架构的可靠性也是不高的，只要有一个不注意，单体应用程序中的一个组件出现故障或者不常用的模块因为存在内存泄漏而造成整个服务无法正常提供服务，都会促使整个应用程序崩溃。

因此，随着应用程序的规模和复杂性的增加，许多组织和开发者正在使用新型架构，如微服务架构和分布式架构，来代替单体应用架构。这些新型架构可以更好地解决单体应用架构的缺点，并提供更高的可扩展性、可维护性和可靠性。

**2.2微服务架构**

单体应用架构的缺陷和变化需要，促使了微服务架构的诞生。在传统单体应用中，所有功能和组件打包在一起，导致难以扩展、维护和部署，且升级、维护或增加新功能需要重新部署整个应用。此外，单体应用也难以实现团队分工协作。

微服务架构是将应用拆分成小自治服务的架构模式，每个服务运行在自己的进程中，通过轻量级通信机制相互通信。每个服务有自己的代码库、数据库、部署环境和团队，服务之间耦合度低，开发团队可以分别开发和部署各自的服务，不同服务之间可以独立升级、扩展和维护。

微服务架构（如图2所示）最早是由Martin Fowler在2005年提出的，随着云计算、容器化、DevOps等技术的兴起，微服务架构逐渐流行，并在许多大型互联网公司中得到广泛应用。

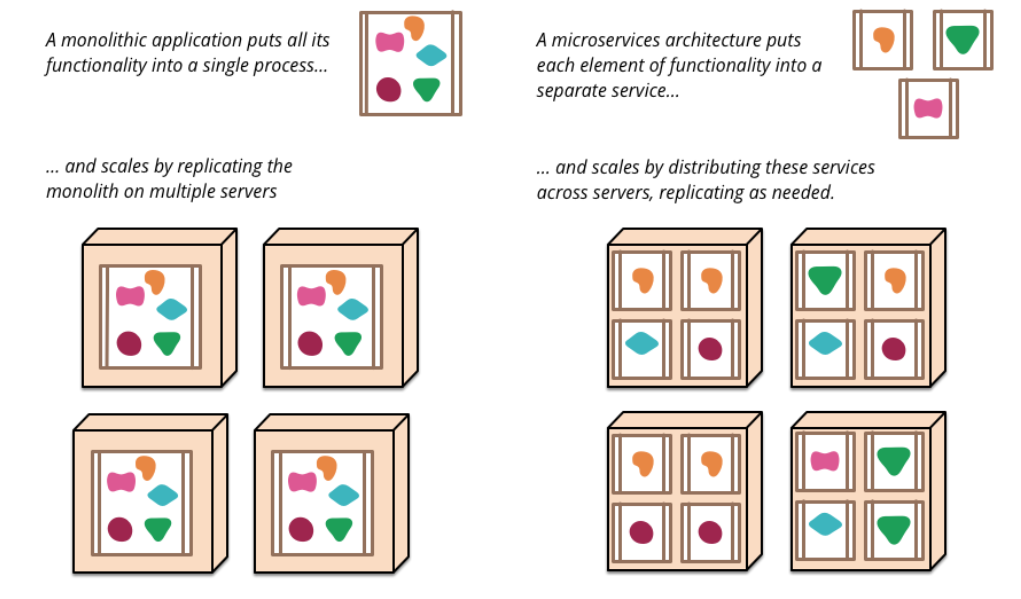


图 2 微服务架构图

**2.2.1微服务架构定义**

微服务是一种软件架构模式，它将一个应用程序拆分成多个小的服务，每个服务都可以独立部署、扩展和维护。每个服务都围绕特定的业务领域构建，并通过轻量级的通信机制（例如REST API或消息队列）进行通信。这种架构风格被称为微服务架构，其核心思想是**分而治之**的思想。将大型复杂的系统分解为多个小型服务，以便更好地管理和维护应用程序。

微服务架构是一个松散耦合的分布式系统，每个服务都可以独立开发、部署和运行。每个服务都使用自己的数据库，因此不会对整个应用程序的可用性产生影响。此外，每个服务都可以使用不同的技术栈，因此可以使用最适合该服务的技术。

在微服务架构中，开发人员可以更快地开发和部署新功能，因为每个服务都是独立的。此外，由于每个服务都是独立的，因此系统的可扩展性也更好。

要理解微服务，需要理解其核心概念和架构原则。这包括：

1. 单一职责原则：每个服务应该只处理一个具体的业务领域。
2. 松耦合：每个服务都应该是独立的，不会影响其他服务。
3. 自治性：每个服务都应该具有自己的数据库和代码库，因此不会影响整个应用程序。
4. 分布式系统：每个服务都应该是一个分布式系统，可以独立部署和运行。
5. API通信：每个服务都应该通过轻量级的通信机制进行通信，例如REST API或消息队列。

微服务是一种分布式系统架构模式，将应用程序拆分成多个小的服务，每个服务都可以独立开发、部署和运行。这种架构风格可以提高开发速度、应用程序的可维护性和可扩展性。

在《架构即未来》一书中，提出了一个可扩展模型，即AFK扩展立方体（Scalability Cube）。该模型将应用程序抽象为三个可扩展的维度，即产品、流程和团队。根据该模型，可以按照这三个维度对系统进行扩展，从而实现无限扩展的目标。如图3所示。

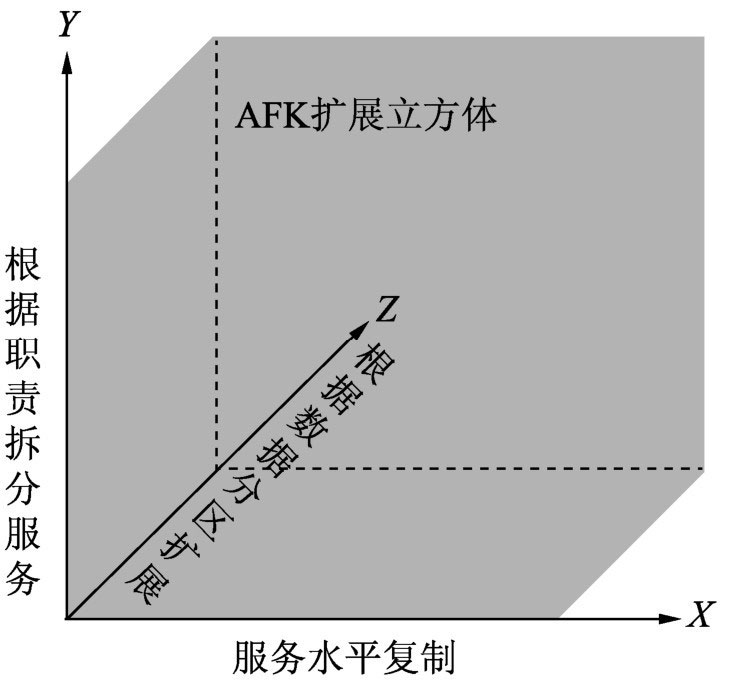


图 3 AFK扩展立方体模型

Scalability Cube模型的三个维度如下：

* X轴：服务水平复制，即多个应用实例的运行并集群负载均衡，是单体应用中常用的扩展方式。
* Z轴：基于数据分区的扩展，即每个应用实例只处理部分数据，并根据请求参数或属性路由请求到不同的应用实例进行处理。
* Y轴：微服务的功能拆分模式，即在业务层面上将单体应用拆分为多个微服务。

虽然X轴和Z轴的扩展可以提升应用的容量和可用性，但无法解决随着业务发展而增加的开发和运维复杂性。相比之下，Y轴的扩展通过从业务功能的角度对庞大的单体应用进行分解，降低了业务开发和运维的复杂性，同时也实现了服务故障的隔离并提高了系统响应时间。此外，功能拆分后可以控制每个团队的规模，让团队更加聚焦并促进团队成长。

**2.2.2微服务架构优缺点**

微服务架构它已成为现代软件开发中的重要思想。微服务架构旨在实现服务的解耦、弹性、可伸缩性和灵活性，但同时也带来了一些挑战。本文将探讨微服务架构的优缺点，分析其在技术和业务层面的影响。

微服务架构具有以下优点：

1. 服务解耦：微服务架构将复杂的应用程序拆分成一组小的、独立的服务，每个服务都可以独立部署和扩展。这种解耦性使得服务的修改和升级更加容易，同时也提高了应用程序的可维护性。
2. 弹性：微服务架构使得服务可以独立部署和扩展，可以根据业务需求调整服务的数量和规模。在服务负载增加时，可以通过增加服务实例来处理负载；当服务负载减少时，可以缩小服务规模，以减少资源消耗。这种弹性使得服务更加健壮，具有更高的可用性。
3. 可伸缩性：微服务架构将应用程序拆分成一组小的服务，可以对每个服务进行独立的扩展。这种可伸缩性使得应用程序可以根据需要快速扩展或缩小。

世间万物，均有阴阳两面。微服务架构也是如此，从其本身来说具有以下缺点：

1. 分布式系统的复杂性：在微服务架构中，系统被分割成多个服务单元，这些服务单元需要通过网络相互通信，因此系统的复杂度增加了许多。
2. 运维难度加大：由于微服务架构中的服务单元数量众多，这使得系统的运维难度加大，需要对每个服务单元进行独立的部署和维护。
3. 数据一致性问题：由于系统被分割成多个服务单元，数据可能会被分割到不同的服务单元中，这就可能会导致数据一致性问题。因此，需要采用分布式事务等技术来保证数据的一致性。
4. 分布式系统的性能问题：在微服务架构中，服务单元需要通过网络相互通信，这就可能会导致网络延迟和性能问题。
5. 代码冗余：在微服务架构中，服务单元需要独立部署和维护，这就意味着每个服务单元都需要包含自己的运行环境和库，这可能会导致代码冗余问题。
6. 服务之间的依赖关系：在微服务架构中，服务之间的依赖关系变得更加复杂，可能会导致服务之间的耦合度增加，这就可能会导致系统的可维护性和可扩展性降低。
7. 重构的成本：如果要将一个单体应用拆分成多个服务单元，需要进行大量的重构工作，这就可能会导致重构的成本非常高。

尽管微服务架构存在一些缺点，但其优点仍然使得它成为当前最流行的架构之一。而缺点也可以通过技术手段得到缓解或解决。

1. 微服务组件

随着现代应用的复杂性不断增加，微服务架构已经成为了许多企业和组织在构建分布式应用程序时的首选。微服务架构的核心是将一个大型的应用程序分解为一组较小、相互独立的服务，从而使开发、部署和维护更加容易。在微服务架构中，服务不仅仅是代码库，还需要包括一些组件和工具，它们协同工作以支持应用程序的构建和运行。以下是一些常见的微服务组件。

1. 服务注册和配置中心：Nacos、ZooKeeper、Spring Cloud Config、Eureka 和 Consul 等
2. 服务间通信框架：openFeign、dubbo
3. Api网关：Spring Cloud Gateway、Zuul
4. 流量治理：sentinel、hystrix

**3.1** **服务注册和配置中心**

服务注册和配置中心是微服务架构中重要的组件之一，主要用于服务的注册、发现和配置。在传统的单体应用架构中，服务之间的调用是通过直接引用代码中的方法来完成的，而在微服务架构中，每个服务都是独立部署的，需要通过注册和发现机制来实现服务之间的通信。

服务注册和发现的概念在SOA(Service-Oriented Architecture，面向服务的架构)中已经被广泛使用。在SOA中，服务的注册和发现是通过UDDI(Universal Description, Discovery and Integration)来实现的。然而，UDDI本身存在诸多问题，如复杂性、效率低等，导致其在实际应用中并不普及。微服务架构中采用了更为简单、轻量的服务注册和发现方式，即通过服务注册中心来实现。

服务配置中心则是用于统一管理微服务中各个服务的配置信息。在微服务架构中，服务的配置信息通常包括服务的端口号、数据源信息、缓存策略等。在传统的单体应用架构中，这些配置信息通常是写在配置文件中，然而，在微服务架构中，由于每个服务都是独立部署的，因此需要一个统一的配置中心来管理服务的配置信息，以便于快速修改和更新。

服务注册和配置中心在微服务架构中的应用场景非常广泛。例如，当一个新的服务被部署时，它需要将自己注册到服务注册中心中，以便其他服务可以通过服务注册中心来发现和调用它。又如，当服务的配置信息发生变化时，只需要在服务配置中心中进行修改即可，所有使用这个配置信息的服务都可以自动更新，无需手动修改每个服务的配置文件。

服务注册和配置中心的意义在于，它们提供了微服务架构中服务治理的基础设施，为微服务架构提供了更好的可扩展性和可维护性。通过服务注册和发现机制，服务之间的调用变得更为简单和灵活；通过配置中心，服务的配置信息变得更加集中和可控。同时，服务注册和配置中心也为微服务架构中的监控、负载均衡等其他组件提供了基础。

其中国内最具有代表性的服务注册和配置中心：Nacos

**3.1.1 Nacos简介**

服务（Service）是 Nacos 世界的一等公民。Nacos 支持几乎所有主流类型的“服务”的发现、配置和管理：

1. [Kubernetes Service](https://kubernetes.io/docs/concepts/services-networking/service/)
2. [gRPC](https://grpc.io/docs/guides/concepts.html" \l "service-definition) & [Dubbo RPC Service](https://dubbo.apache.org)
3. [Spring Cloud RESTful Service](https://spring.io/projects/spring-cloud)

**Nacos的关键特性包括：**服务发现和服务健康监测、动态配置服务、动态DNS服务、以及服务及其元数据管理。

* 服务发现和服务健康监测是Nacos的核心功能之一，它支持基于DNS和基于RPC的服务发现，提供实时的健康检查，防止向不健康的主机或服务实例发送请求，还支持传输层和应用层的健康检查。对于复杂的云环境和网络拓扑环境中的服务的健康检查，Nacos提供了两种健康检查模式：agent上报模式和服务端主动检测。
* 动态配置服务是Nacos的另一个重要功能，它让您以中心化、外部化和动态化的方式管理所有环境的应用配置和服务配置。动态配置消除了配置变更时重新部署应用和服务的需要，让配置管理变得更加高效和敏捷，帮助您更安全地在生产环境中管理配置变更和降低配置变更带来的风险。
* 动态DNS服务支持权重路由，让您更容易地实现中间层负载均衡、更灵活的路由策略、流量控制以及数据中心内网的简单DNS解析服务。动态DNS服务还能让您更容易地实现以DNS协议为基础的服务发现，以帮助您消除耦合到厂商私有服务发现API上的风险。
* 服务及其元数据管理可以让您从微服务平台建设的视角管理数据中心的所有服务及元数据，包括管理服务的描述、生命周期、服务的静态依赖分析、服务的健康状态、服务的流量管理、路由及安全策略、服务的SLA以及最首要的metrics统计数据。

说了这么多我们还是从一张图来看懂Nacos吧！如图4所示

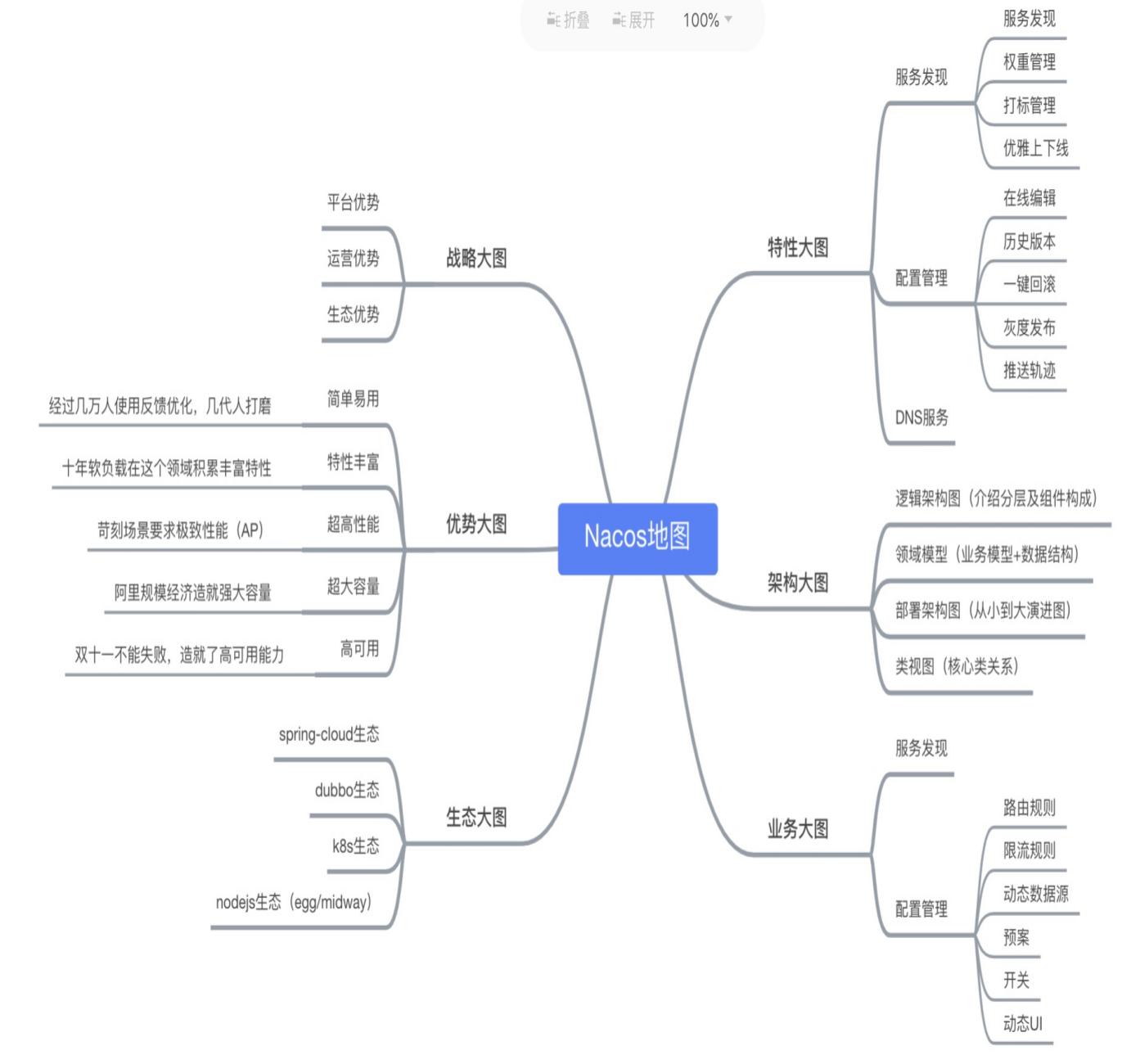


图 4 Nacos地图

* 特性大图：要从功能特性，非功能特性，全面介绍我们要解的问题域的特性诉求
* 架构大图：通过清晰架构，让您快速进入 Nacos 世界
* 业务大图：利用当前特性可以支持的业务场景，及其最佳实践
* 生态大图：系统梳理 Nacos 和主流技术生态的关系
* 优势大图：展示 Nacos 核心竞争力
* 战略大图：要从战略到战术层面讲 Nacos 的宏观优势

**3.1.2 Nacos的快速启动**

Nacos快速启动方式有很多种，比如：

1. 安装Nacos服务软件,通过脚本来快速启动
2. 通过docker容器技术，以命令的方式快速启动Nacos服务
3. 通过Kubernetes容器编排引擎来自动化部署Nacos服务

我们将介绍一下docker方式来快速启动Nacos服务。

**操作步骤：**

1. 下载Nacos镜像

|  |
| --- |
| docker pull nacos/nacos-server:v2.1.1 |

1. 启动Nacos镜像

|  |
| --- |
| # 单机方式启动Nacos  docker run --name nacos \  -e MODE=standalone -e NACOS\_APPLICATION\_PORT=8850 \  -e SPRING\_DATASOURCE\_PLATFORM=mysql  -e MYSQL\_SERVICE\_HOST=192.168.7.161 \  -e MYSQL\_SERVICE\_DB\_NAME=nacos  -e MYSQL\_SERVICE\_USER=root  -e MYSQL\_SERVICE\_PASSWORD=123123 \  -e MYSQL\_SERVICE\_DB\_PARAM="serverTimezone=Asia/Shanghai&characterEncoding=utf8&connectTimeout=1000&socketTimeout=3000&autoReconnect=true&allowPublicKeyRetrieval=true" \  -e NACOS\_AUTH\_ENABLE=true  -e NACOS\_AUTH\_TOKEN\_EXPIRE\_SECONDS=18000 \  --network="host"  -p 8850:8850  -d nacos/nacos-server:v2.1.1 |

**3.2 服务间通信框架**

在微服务架构中，服务间通信组件扮演着至关重要的角色。由于微服务架构中各个服务都是独立的、自治的，因此服务间通信必不可少。这些服务之间的通信可以是同步的，也可以是异步的。

服务间通信组件的意义在于，它提供了一种可靠的方式来处理服务之间的通信，以确保系统的高可用性和可靠性。例如，在同步通信中，服务之间需要相互协作，而在异步通信中，一个服务可以发送消息给另一个服务，然后继续执行自己的任务。

服务间通信组件可以采用多种协议，例如REST，RPC和消息传递。REST是一种基于HTTP的通信协议(使用REST方式的框架有OpenFeign)，RPC是一种更高效的通信协议(使用RPC方式的框架有Dubbo)，而消息传递则可以使用消息队列来实现异步通信。这些不同的协议可以在不同的场景下使用，以满足不同的需求。

在微服务架构中，服务间通信组件的另一个重要意义在于，它有助于将系统中的各个服务解耦。这样一来，系统中的每个服务都可以独立地进行开发、部署、升级和维护，而不会对其他服务产生影响。

由此可见，服务间通信组件在微服务架构中扮演着至关重要的角色。接下来我们来接一下RPC通信方式的框架-Dubbo。

**3.2.1 Dubbo简介**

Apache Dubbo是一种用于解决微服务架构下服务治理和通信问题的RPC服务开发框架，它提供了Java、Golang等多语言SDK实现。Dubbo原生具备远程地址发现和通信能力，利用其服务治理特性，可以实现服务发现、负载均衡、流量调度等需求。Dubbo还具有高度可扩展性，可以方便地实现各种定制逻辑。

Dubbo3是面向云原生的下一代RPC服务框架，基于Dubbo 2.x演进而来。在保持原有功能特性的同时，Dubbo3在易用性、超大规模微服务实践、云原生基础设施适配、安全性等方面进行了全面升级。

Apache Dubbo 最初在 2008 年由 Alibaba 捐献开源，很快成为了国内开源服务框架选型的事实标准框架 ，得到了各行各业的广泛应用。在 2017 年，Dubbo 正式捐献到 Apache 软件基金会并成为 Apache 顶级项目，目前 Dubbo3 已经是一站式的微服务解决方案提供：

* 基于 HTTP/2 的 [Triple 协议](https://cn.dubbo.apache.org/zh-cn/docs3-v2/java-sdk/concepts-and-architecture/triple/)以及面向代理 API 的编程体验。
* 强大的[流量治理能力](https://cn.dubbo.apache.org/zh-cn/overview/tasks/traffic-management)，如地址发现、负载均衡、路由选址、动态配置等。
* [多语言 SDK 实现](https://cn.dubbo.apache.org/zh-cn/overview/mannual/)，涵盖 Java、Golang、Javascript 等，更多语言实现将会陆续发布。
* 灵活的适配与扩展能力，可轻松与微服务体系其他组件如 Tracing、Transaction 等适配。
* [Dubbo Mesh 解决方案](https://cn.dubbo.apache.org/zh-cn/docs3-v2/java-sdk/concepts-and-architecture/mesh/)，同时支持 Sidecar、Proxyless 等灵活的 Mesh 部署方案。

Apache Dubbo 是为了解决超大规模微服务集群实践问题而设计的，能够很好地满足企业的大规模微服务实践需求。阿里巴巴、工商银行、中国平安、携程等社区用户多年来通过大规模生产环境流量对 Dubbo 的稳定性与性能进行了充分验证。因此，Dubbo 在解决业务落地与规模化实践方面有着无可比拟的优势：

* 开箱即用
  + 易用性高，如 Java 版本的面向接口代理特性能实现本地透明调用
  + 功能丰富，基于原生库或轻量扩展即可实现绝大多数的微服务治理能力
* 面向超大规模微服务集群设计
  + 极致性能，高性能的 RPC 通信协议设计与实现
  + 横向可扩展，轻松支持百万规模集群实例的地址发现与流量治理
* [高度可扩展](https://cn.dubbo.apache.org/zh-cn/overview/what/extensibility)
  + 调用过程中对流量及协议的拦截扩展，如 Filter、Router、LB 等
  + 微服务治理组件扩展，如 Registry、Config Center、Metadata Center 等
* 企业级微服务治理能力
  + 国内公有云厂商支持的事实标准服务框架
  + 多年企业实践经验考验，参考[用户实践案例](https://cn.dubbo.apache.org/zh-cn/users)

**3.2.2 Dubbo基本工作流程**

Dubbo 首先是一款 RPC 框架，它定义了自己的 RPC 通信协议与编程方式。如上图所示，用户在使用 Dubbo 时首先需要定义好 Dubbo 服务；其次，是在将 Dubbo 服务部署上线之后，依赖 Dubbo 的应用层通信协议实现数据交换，Dubbo 所传输的数据都要经过序列化，而这里的序列化协议是完全可扩展的。

使用 Dubbo 的第一步就是定义 Dubbo 服务，服务在 Dubbo 中的定义就是完成业务功能的一组方法的集合，可以选择使用与某种语言绑定的方式定义，如在 Java 中 Dubbo 服务就是有一组方法的 Interface 接口，也可以使用语言中立的 Protobuf Buffers [IDL 定义服务](https://cn.dubbo.apache.org/zh-cn/overview/tasks/triple/idl/)。定义好服务之后，服务端（Provider）需要提供服务的具体实现，并将其声明为 Dubbo 服务，而站在服务消费方（Consumer）的视角，通过调用 Dubbo 框架提供的 API 可以获得一个服务代理（stub）对象，然后就可以像使用本地服务一样对服务方法发起调用了。 在消费端对服务方法发起调用后，Dubbo 框架负责将请求发送到部署在远端机器上的服务提供方，提供方收到请求后会调用服务的实现类，之后将处理结果返回给消费端，这样就完成了一次完整的服务调用。如图5中的 Request、Response 数据流程所示。

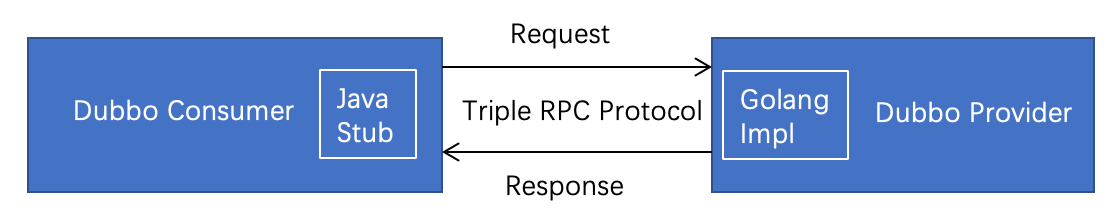


图 5 Dubbo请求流程

需要注意的是，在 Dubbo 中，我们提到服务时，通常是指 RPC 粒度的、提供某个具体业务增删改功能的接口或方法，与一些微服务概念书籍中泛指的服务并不是一个概念。

在分布式系统中，尤其是随着微服务架构的发展，应用的部署、发布、扩缩容变得极为频繁，作为 RPC 消费方，如何动态的发现服务提供方地址成为 RPC 通信的前置条件。Dubbo 提供了自动的地址发现机制，用于应对分布式场景下机器实例动态迁移的问题。如图6所示：



图 6 Dubbo结构图

通过引入注册中心来协调提供方与消费方的地址，提供者启动之后向注册中心注册自身地址，消费方通过拉取或订阅注册中心特定节点，动态的感知提供方地址列表的变化。

**3.2.3 Dubbo快速入门Demo**

由于Dubbo官方有现成的Demo，我们这里直接采用基于官方Samples项目来直接启动。

**操作步骤：**

1. 获取Samples工程项目

|  |
| --- |
| gitclone --depth=1 --branch master git@github.com:apache/dubbo-samples.git |

1. 启动一个简易的注册中心。
2. 对于一个微服务化的应用来说，注册中心是不可或缺的一个组件。只有通过注册中心，消费端才可以成功发现服务端的地址信息，进而进行调用。
3. 这里我们提供了一个基于 Apache Zookeeper 注册中心的简易启动器。

|  |
| --- |
| # Windows:  ./mvnw.cmd clean compile exec:java -pl tools/embedded-zookeeper  # Linux / MacOS:  ./mvnw clean compile exec:java -pl tools/embedded-zookeeper  # Docker:  docker run --name some-zookeeper -p 2181:2181 --restart always -d zookeeper |

1. 打包本地API模块。
2. 为了成功编译服务端、消费端模块，需要先在本地打包安装 dubbo-samples-spring-boot-interface 模块。

|  |
| --- |
| ./mvnw clean install -pl 1-basic/dubbo-samples-spring-boot  ./mvnw clean install -pl 1-basic/dubbo-samples-spring-boot/dubbo-samples-spring-boot-interface |

1. 启动服务提供者。
2. 在启动了注册中心之后，下一步是启动一个对外提供服务的服务提供者。在 dubbo-samples 中也提供了对应的示例，可以通过以下命令快速拉起。

|  |
| --- |
| # Windows:  ./mvnw.cmd clean compile exec:java -pl 1-basic/dubbo-samples-spring-boot/dubbo-samples-spring-boot-provider -Dexec.mainClass="org.apache.dubbo.springboot.demo.provider.ProviderApplication"  # Linux / MacOS:  ./mvnw clean compile exec:java -pl 1-basic/dubbo-samples-spring-boot/dubbo-samples-spring-boot-provider -Dexec.mainClass="org.apache.dubbo.springboot.demo.provider.ProviderApplication" |

1. 启动服务消费者。
2. 最后一步是启动一个服务消费者来调用服务提供者，也即是 RPC 调用的核心，为服务消费者提供调用服务提供者的桥梁。

|  |
| --- |
| # Windows:  ./mvnw.cmd clean compile exec:java -pl 1-basic/dubbo-samples-spring-boot/dubbo-samples-spring-boot-consumer -Dexec.mainClass="org.apache.dubbo.springboot.demo.consumer.ConsumerApplication"  # Linux / MacOS:  ./mvnw clean compile exec:java -pl 1-basic/dubbo-samples-spring-boot/dubbo-samples-spring-boot-consumer -Dexec.mainClass="org.apache.dubbo.springboot.demo.consumer.ConsumerApplication" |

1. 项目实战

说了那么多微服务方面的技术理论，还没缺实践，接下来将用上述知识来教你怎么架构一个分布式的微服务项目。

* 1. **需求分析**

**背景：**随着人们生活水平的提高和社会的发展，越来越多的人选择在学校西餐厅就餐。然而，由于人流量较大，顾客点餐时需要排队等候，而服务员又往往忙于为顾客送餐，导致点餐效率低下，影响了顾客的就餐体验。因此，开发一个方便快捷的学校西餐厅点餐服务平台，能够提高点餐效率、节省时间、改善就餐体验，已成为学校餐厅的刚需。

**需求分析：**

1. 用户登录与注册。
   * 1. 用户可以通过平台进行注册，注册成功后即可使用平台提供的服务；
     2. 用户可以通过手机号、邮箱或用户名进行登录，登录成功后即可浏览菜单、下单、查看订单等功能；
     3. 平台需要支持用户密码找回功能，可以通过短信或邮箱验证方式来重置密码；
     4. 平台后续会开发支持第三方登录，例如微信登录、支付宝登录等，提高用户使用便捷性。(后续集成)
2. 点餐服务：用户可通过点餐服务平台在线浏览菜单，并选择所需菜品和饮料，平台支持下单和支付功能。
   * 1. 用户可以在平台上选择西餐厅提供的餐品，并添加到购物车中。
     2. 用户可以在购物车中修改、删除餐品。
     3. 用户通过扫描桌面上的二维码来，确认订单配送位置
     4. 平台需要支持多种支付方式，例如微信支付、线下支付等。
     5. 当用户完成订单时，需要向餐厅推送订单信息。
3. 订单管理：餐厅管理员可以在订单管理模块中，查看、接单和处理订单，以保证订单及时、准确地送达给用户。
   * 1. 餐厅可以在平台上查看所有已完成的订单。
     2. 餐厅可以查看每个订单的详细信息，包括订单编号、下单时间、支付凭证、订单信息等。
     3. 餐厅可以将订单标记为“未付款”、“已付款”、“待评价”等状态，以便管理订单的进展。
4. 用户评价：平台应支持用户对菜品、服务、餐厅环境等方面进行评价，以便餐厅及时反馈和改进，提高用户的满意度。
   * 1. 用户可以在订单完成后进行评价，并给出评分和文字评价。
     2. 餐厅可以查看所有评价，并根据评价内容进行改进。
     3. 用户可以查看其他用户的评价，以便参考。
5. 西餐厅信息变更。
   * 1. 西餐厅可以通过平台对菜单进行调整、价格进行修改等操作；
     2. 西餐厅可以通过平台对菜单进行调整、价格进行修改等操作；
     3. 管理员可以对餐厅基本信息进行修改，例如餐厅名称、地址、联系方式等。
   1. **技术选型**

本项目是一个学校西餐厅点餐服务平台，需要支持用户登录与注册、在线浏览菜单并选择所需菜品和饮料、下单和支付、查看、接单和处理订单、支持用户对菜品、服务、餐厅环境等方面进行评价。网站预计每天访问量为5000人次，需要保证高性能、高可用性、可扩展性和安全性。

**我们定义了以下标准来评估技术选型：**

性能：响应时间不超过2秒，QPS达到2000以上

可扩展性：支持分布式部署，能够动态扩容和缩容

可维护性：代码结构清晰，易于维护和升级

安全性：采用HTTPS协议保证数据传输安全，防止SQL注入、XSS等攻击

我们对常用的Web框架、数据库、缓存系统等技术进行了评估和比较，最终选择了Spring Boot、MySQL和Redis作为主要技术。为了实现高可用性和可扩展性，我们采用微服务架构，使用Spring Cloud和Spring Cloud Alibaba构建服务治理和服务发现机制，并使用Nacos作为注册中心。同时，我们使用Sentinel进行服务容错和流量控制，保证系统的稳定性和安全性。

**技术版本：**

* springboot:2.7.4
* springcloud:2021.0.4
* springcloud alibaba:2021.0.4.0
* redis:7.0.5
* mysql:8.0.3
* nacos:v2.1.1
* sentinel:1.7.2
* docker:1.8.3

**可行性和风险：**我们认为这些技术成熟度高、生态环境丰富、开发成本低，风险较小，符合项目的需求。

**理念和方案：**我们采用了微服务架构，将系统拆分为用户服务、菜单服务、订单服务、评价服务等多个服务，通过RESTful API进行通信。每个服务都部署在独立的容器中，可以动态扩容和缩容，保证高可用性和可扩展性。数据库采用MySQL集群，数据缓存采用Redis集群，前端采用Vue.js框架实现。同时，我们使用Docker进行容器化部署，使用Jenkins进行自动化构建和部署，提高了系统的部署效率和可维护性。

优势和挑战

采用微服务架构可以实现高可用性、可扩展性和松耦合，方便开发和维护。但同时也面临着服务调用复杂。

* 1. **架构设计**
     1. **表设计**

餐点类型表记录了不同餐点的类型，如主食、汤品、饮料等，每个餐点类型都有一个唯一的餐点类型id。

表4-1 餐点类型表

| 序号 | 字段描述 | 字段名 | 数据类型 | 主键 |
| --- | --- | --- | --- | --- |
| 1 | 餐点类型id | meal\_type\_id | int | 是 |
| 2 | 餐点类型名 | meal\_type | varchar | 否 |

菜单结构表则记录了每个餐点类型的父级类型id，这样可以形成一个菜单层次结构，比如将主食、汤品、饮料分类后再分别细分成米饭、面条、热干面等具体的菜品。

表4-2 菜单结构表

| 序号 | 字段描述 | 字段名 | 数据类型 | 主键 |
| --- | --- | --- | --- | --- |
| 1 | 餐点类型id | meal\_type\_id | int | 是 |
| 2 | 父级类型id | meal\_type\_parent\_id | int | 是 |

餐点表则记录了每个具体的餐点，包括餐点id、所属的餐点类型id、名称、销量、餐点信息、价格、库存以及创建时间和更新时间等信息。

表4 餐点表

| 序号 | 字段描述 | 字段名 | 数据类型 | 主键 |
| --- | --- | --- | --- | --- |
| 1 | 餐点id | meal\_id | int | 是 |
| 2 | 餐点类型id | meal\_type\_id | int | 否 |
| 3 | 名称 | meal\_name | varchar | 否 |
| 4 | 销量 | meal\_sales | int | 否 |
| 5 | 餐点信息 | meal\_info | varchar | 否 |
| 6 | 价格 | meal\_price | int | 否 |
| 7 | 库存 | meal\_inventory | int | 否 |
| 8 | 创建时间 | create\_time | datetime | 否 |
| 9 | 更新时间 | update\_time | datetime | 否 |

餐点图片表记录了每个餐点的图片url，可以为用户提供更直观的菜品展示。

表4-3 餐点图片表

| 序号 | 字段描述 | 字段名 | 数据类型 | 主键 |
| --- | --- | --- | --- | --- |
| 1 | 餐点id | meal\_id | int | 否 |
| 2 | 图片url | meal\_picture\_url | varchar | 是 |

这些表的设计可以帮助餐饮企业管理不同种类的菜品以及它们的相关信息，从而提高菜品管理的效率。

订单表用于存储用户下的订单信息，其中包括订单编号、用户编号、座位编号、支付凭证、订单总金额、下单需求、订单状态、下单时间、创建时间和更新时间等信息。

表4-3 订单表

| 序号 | 字段描述 | 字段名 | 数据类型 | 主键 |
| --- | --- | --- | --- | --- |
| 1 | 订单id | order\_id | varchar | 否 |
| 2 | 用户id | user\_id | bigint | 是 |
| 3 | 座位id | seat\_id | int | 否 |
| 4 | 支付凭证 | transaction\_id | varchar | 否 |
| 5 | 总金额 | total\_price | int | 否 |
| 6 | 下单需求 | order\_demand | varchar | 否 |
| 7 | 订单状态 | order\_state | tinyint | 否 |
| 8 | 下单的时间 | order\_purchase\_time | datetime | 否 |
| 8 | 创建时间 | create\_time | datetime | 否 |
| 9 | 更新时间 | update\_time | datetime | 否 |

订单子项表则用于存储订单中所包含的每个餐点的详细信息，包括订单子项编号、归属订单编号、餐点编号、餐点的单位价格和购买数量等信息。

表4-3 订单子项表

| 序号 | 字段描述 | 字段名 | 数据类型 | 主键 |
| --- | --- | --- | --- | --- |
| 1 | 订单子项id | item\_id | varchar | 否 |
| 2 | 归属的订单id | order\_id | bigint | 是 |
| 3 | 餐点id | meal\_id | int | 否 |
| 4 | 餐点的单位价格 | purchasing\_meal\_price | varchar | 否 |
| 5 | 购买数量 | purchase\_num | int | 否 |

用户可以选择餐厅中的菜品下单，系统会生成一个唯一的订单号，并记录用户的订单信息和餐点详细信息。同时，系统可以根据订单状态的不同，及时更新订单信息，以便在配送和结算等环节能够准确地进行处理。

* + 1. **模块设计**

为了简化系统的设计、开发和维护过程，提高系统的可维护性和可扩展性。

我们将系统拆分成以下模块：**（**如图7所示**）**

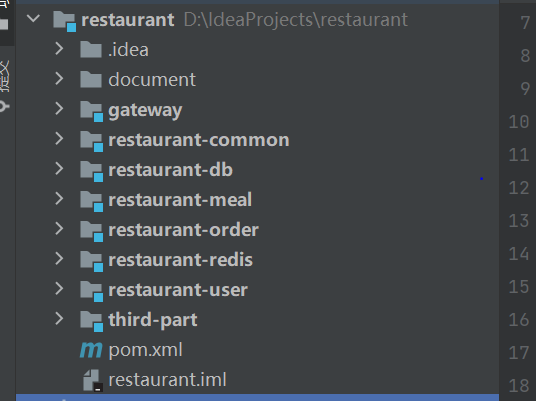
1. 用户模块：负责用户的信息管理、登录注册等功能
2. 菜单模块：负责菜单的生成展示功能。
3. 订单模块：负责订单的管理、查询、生成等功能。
4. 评论模块：负责评论的管理、查询、处理等功能。
5. 第三方接口模块：负责第三方接口的调用功能，比如：微信支付、邮件发送等
6. 

图 7 模块

* 1. **开发文档**

开发文档是指在软件开发过程中编写的各种文档，包括需求文档、设计文档、开发规范、测试文档、使用手册等等。这些文档是开发过程中记录和沉淀下来的经验和知识，是团队成员进行协作和交流的重要工具。

开发文档的好处主要体现在以下几个方面：

1. 提高团队协作效率：开发文档可以作为团队成员之间的沟通工具，帮助团队成员更好地理解项目的需求和设计，避免出现误解和沟通不畅的情况，从而提高团队的协作效率。
2. 保证开发质量：开发文档可以规范开发过程，明确开发标准和规范，避免出现漏洞和缺陷，提高软件开发质量。
3. 方便后期维护：开发文档可以记录软件开发过程中的决策和问题，对于后期维护和升级提供了很好的支持。维护人员可以根据开发文档的记录，更快速地了解系统的架构和设计，进行相关的修改和维护。
4. 降低项目风险：开发文档可以帮助项目管理者更好地了解项目的状态和进度，发现和解决问题，降低项目的风险。

因此开发文档在开发过程中极其重要，下面是本项目的开发文档。



图 8 用户模块开发文档

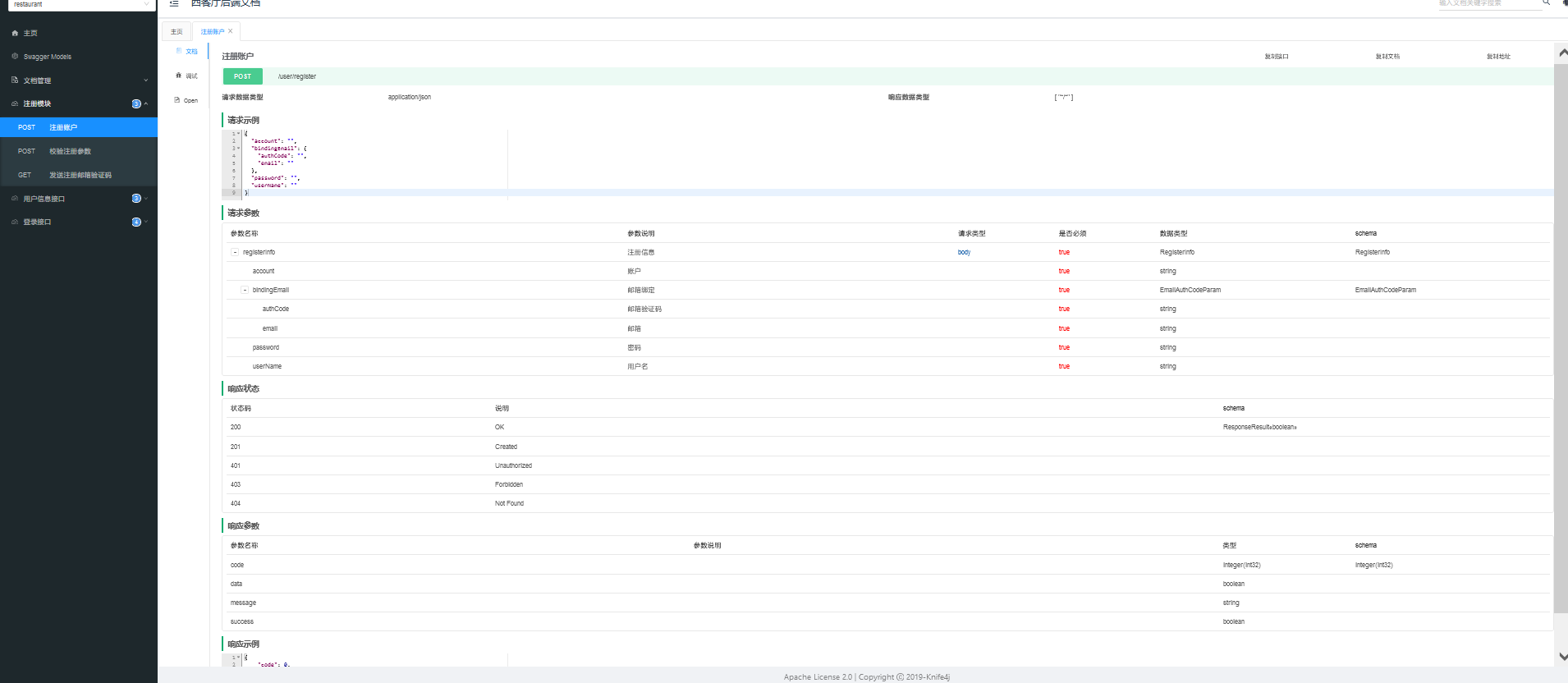


图 9 接口描述与规范

* 1. **功能实现和测试**

用户登录与注册：用户可以通过平台进行注册，如果注册的用户存在或者进行注册的邮箱存在，则不允许注册。注册成功后，会跳转到登录页面，让用户进行登录，登录可以进行邮箱登录、账户密码登录。下面来带大家走一下流程。



图 10 My页面

进行注册操作，填写注册信息，并点击发送验证码到注册的邮箱中。

实现思路：

1. 发送验证码到注册的邮箱中：判断注册的邮箱是否已经有用户绑定了该邮箱，有代表该用户注册过了，则注册失败！否则代表不存在注册邮箱被用户绑定的情况。所以可以发送注册邮件到注册邮箱中。
2. 根据填写的注册信息，进行校验。如果注册的账户存在，则返回用户账户邮件存在。如果注册的账户不存在，则校验验证码是否过时，如果没有过时则判断验证码是否正确匹配，如果是，则进行注册操作，即根据注册信息，生成用户，并保存到数据库中。



图 11 注册信息填写



图 12 验证码

注册成功后，进行账号密码的登录操作

实现思路：拿着登录信息到数据库里查询，看看是否存在账户，密码是否匹配，个存在并匹配，则登录成功，返回登录的用户信息，并将用户信息保存到会话缓存中，以便后面获取，提高获取的效率。



图 13 登录

登录成功后，查看个人资料。

实现思路：从缓存会话中获取登录的用户信息。



图 14 个人资料

浏览菜单，选择要点的餐点，也可以在购物车里进行增加购买数量或者减少删除餐点

实现思路：先去缓存中获取菜单各个成分数据，如果获取不到，则从数据库中获取，然后再应用层进行关联，最后响应给前端渲染。

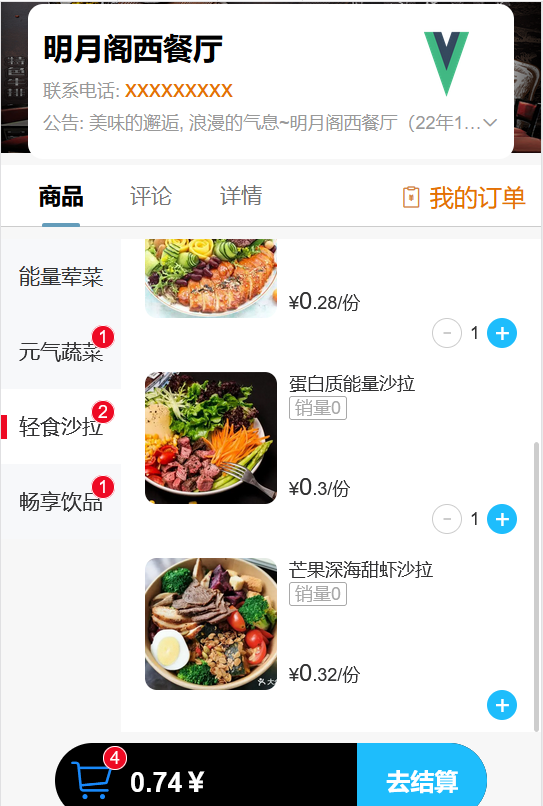


图 15 浏览菜单



图 16 餐点栏

进行下单购买操作，填写下单的要求（比如：加辣、加醋、不要葱），然后扫码支付。如果未支付，也可以去我的订单那里查看下单情况，然后进行支付。

实现思路：根据传入的下单内容，进行判断购买的餐点是否存在，库存是否充足，如果足够，则计算总价格，生成订单项和订单保存到数据库中，并去远程调用生成支付url，将生成的url字符串转换成二维码图片并压缩成base64字符串，发布支付订单事件，开启一条线程去监听订单过时，并且将支付订单数据保存到redis。



图 17 下单信息

提交订单，进行扫码支付，如果在指定时间内没有支付成功，二维码会过时，只需要点击刷新即可，重新扫码支付，支付成功后，等待就餐即可。



图 18 支付二维码

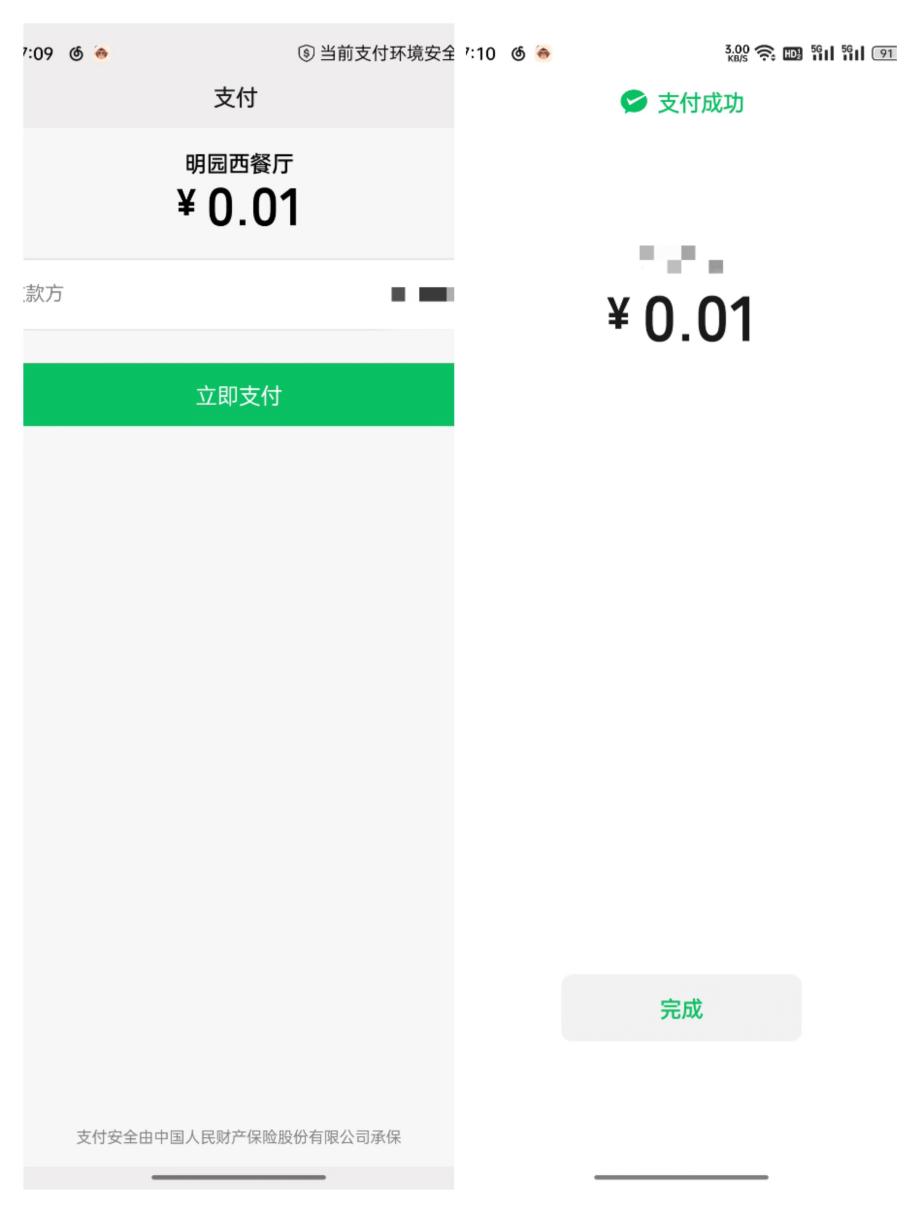


图 19 微信付款

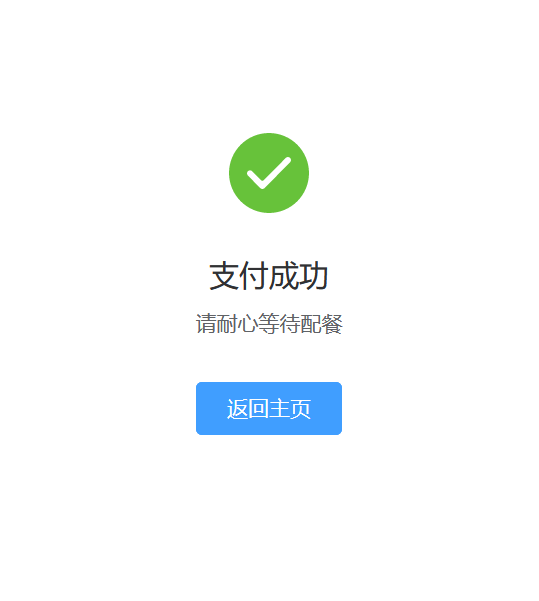


图 20 支付成功

支付成功后，我们可以在我的订单那里看到下单记录。如果我们长时间未支付，则订单会超时，即将无法再次进行支付。



图 21 订单列表

以上就是部分功能的流程和实现思路。

* 1. **部署调试**

由于项目有很多模块，环境也是诸多。因此，本项目采用docker容器引擎技术，进行部署。

部署mysql环境

|  |
| --- |
| docker run -it --name mysql -p=3306:3306 --privileged \  -e MYSQL\_ROOT\_PASSWORD=123123 \  -v ~/mysql/logs:/logs \  -v ~/mysql/conf:/etc/mysql/conf.d \  -v ~/mysql/data:/var/lib/mysql \  -d mysql:8.0.30 |

部署redis环境

|  |
| --- |
| docker run --name redis -p=6379:6379 \  -v ~/redis/config/redis.conf:/redis/config/redis.conf \  -v ~/redis/config:/redis/config \  -d redis:7.0.5 redis-server /redis/config/redis.conf |

部署nacos环境

|  |
| --- |
| docker run --name nacos \  -e MODE=standalone -e NACOS\_APPLICATION\_PORT=8848 \  -e SPRING\_DATASOURCE\_PLATFORM=mysql -e MYSQL\_SERVICE\_HOST=39.108.164.243 \  -e MYSQL\_SERVICE\_DB\_NAME=nacos -e MYSQL\_SERVICE\_USER=root -e MYSQL\_SERVICE\_PASSWORD=123123 \  -e MYSQL\_SERVICE\_DB\_PARAM="serverTimezone=Asia/Shanghai&characterEncoding=utf8&connectTimeout=1000&socketTimeout=3000&autoReconnect=true&allowPublicKeyRetrieval=true" \  -e NACOS\_AUTH\_ENABLE=true -e NACOS\_AUTH\_TOKEN\_EXPIRE\_SECONDS=18000 \  -p 8848:8848 -p 9848:9848 \  -d nacos/nacos-server:v2.1.1 |

部署sentinel环境

|  |
| --- |
| docker run --name sentinel -d -p 8858:8858 -d bladex/sentinel-dashboard |

如图所示 则部署成功。

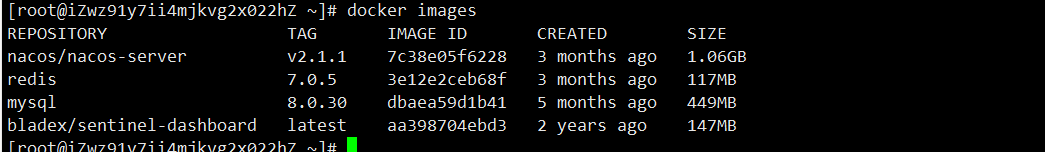


图 22 环境镜像

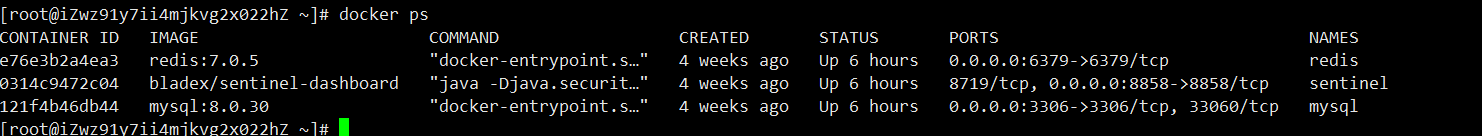


图 23 运行情况

编写模块服务的dockerfile文件，这里举例一个，比如订单模块

|  |
| --- |
| FROM java:8  MAINTAINER Hzp<2412594046@qq.com>  ENV WORK\_DIR ~/jar  ENV FILE\_JAR restaurant-order-0.0.1-SNAPSHOT.jar  ENV JAR\_POS $WORK\_DIR/$FILE\_JAR  WORKDIR $WORK\_DIR  ADD $FILE\_JAR $JAR\_POS  RUN chmod +x $JAR\_POS  EXPOSE 8083  CMD java -jar $JAR\_POS |

编写批处理脚本,批量构建模块服务镜像

|  |
| --- |
| docker build -f gateway -t gateway:0.0.1 .  docker build -f user -t user:0.0.1 .  docker build -f third-part -t third-part:0.0.1 .  docker build -f meal -t meal:0.0.1 .  docker build -f order -t order:0.0.1 .  docker build -f fastdfs -t fastdfs:0.0.1 . |

编写批处理脚本,批量启动模块服务

|  |
| --- |
| docker run --name gateway -p 81:81 -it -d gateway:0.0.1  docker run --name user -p 8079:8079 -it -d user:0.0.1  docker run --name third-part -p 8081:8081 -it -d third-part:0.0.1  docker run --name meal -p 8082:8082 -it -d meal:0.0.1  docker run --name order -p 8083:8083 -it -d order:0.0.1  docker run --name fastdfs -p 8084:8084 -it -d fastdfs:0.0.1 |

如图所示，则部署成功了。

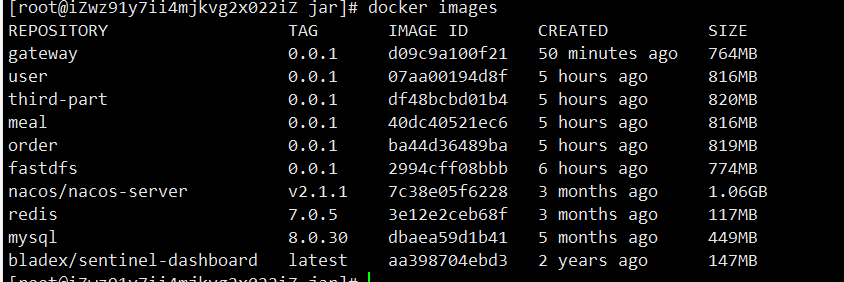


图 24 模块服务镜像

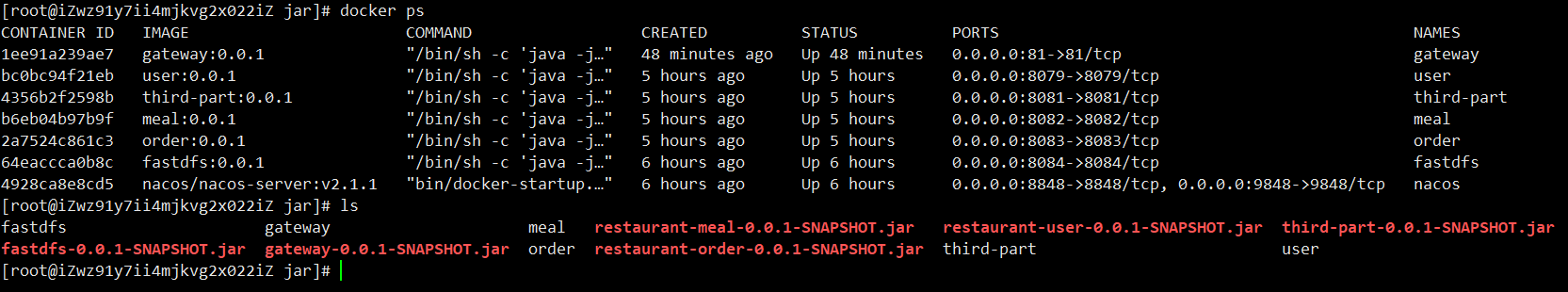


图 25 运行情况

参考文献

1. 杨开振.《Spring Cloud微服务和分布式系统实践》.人民邮电出版社 2020 - 05
2. 黄文毅.《一步一步学Spring boot:微服务项目实战 第二版》.清华大学出版社 2019 - 12
3. 董超,胡炽维.《Spring Cloud微服务架构开发实战》.机器工业出版社 2018 - 08
4. 柳伟卫.《分布式系统常用技术及案例分析 第二版》.电子工业出版社 2019 - 01
5. 王伟杰.《从程序员到架构师》.机器工业出版社 2022 - 01
6. Alibaba.《Nacos手册》.阿里巴巴
7. Apache.《Duboo手册》.Apache软件基金会
8. Martin L. Abbott /Michael T. Fisher .《架构即未来》.机器工业出版社 2016 - 4
9. Martin Fowler.《Micro Services》.2014

致谢

在此，我要向我的指导老师王乌云老师表达最真挚的感激之情。

感谢王老师在我学术研究的各个方面的悉心指导和无私支持。在我研究的过程中，王老师不仅在学术上给予了我深刻的指导和启示，还在实践中对我进行了细致的指导，使我能够迅速提高自己的研究水平。在此，我要向王老师表示深深的敬意和感激之情。同时，我还要感谢我的家人和朋友，感谢他们一直以来对我的支持和鼓励，让我在学业和生活中感受到了无限的温暖和力量。

最后，我要感谢评审专家和答辩委员会的专业评价和指导意见，感谢同行学者的启示和帮助，感谢一直支持和关注我的人们。感谢你们的陪伴和支持，让我在学术路上不断前行，取得了这次研究成果。

在此，再次向王老师和所有给予我帮助的人们致以最诚挚的感谢和祝福。