TONGJI UNIVERSITY

毕业设计（论文）

课题名称	利用云计算平台设计 ERP 物流管理服务
副 标 题	
学　　院	电子与信息工程学院
专　　业	计算机科学与技术
学生姓名	肖鹏飞
学　　号	1953072
指导教师	曾国荪
日　　期	2023 年 6 月 6 日

利用云计算平台设计 ERP 物流管理服务

装 订 线

摘 要

ERP（企业资源计划）是工业软件的核心内容，通过仓储和供应链等模块的统筹，对信息流、物流、数据流、人流等进行集成一体化的管理。而随着云计算技术的兴起，与云计算高度密切相关的应用被广泛运用于实际生产环境当中。以 Kubernetes 为代表的容器化微服务架构下的开发技术，是目前云计算应用开发趋势和主流，对于解决软件即服务应用具有普及和支撑作用。对物流和供应链的集成管理在企业的资源管理中起到了举足轻重的作用。

本文以 Kubernetes 集群和容器化微服务架构为主题，研究了云原生技术和 ERP 工业软件与物流管理系统，以及物流管理系统和云计算的融合形态等内容，设计了 Kubernetes 集群的架构设计以及集群在云平台上的部署形态，包括集群抽象层级，集群内外通信架构，集群上云部署形态等方面的设计，并依据研究和相关设计在阿里云计算平台上搭建了一个完整的 Kubernetes 集群，将各项容器化微服务部署在集群中。本文立足于容器化微服务架构下的软件开发，通过使用开源 Kubernetes 工具，在云计算平台下进行相关软件的开发和运行，对 Kubernetes 的应用领域、功能、管理和部署等方面进行了相关的研究与实践，严格按照软件工程的指导思想进行软件系统的建模和相关应用软件的开发。系统使用 C++作为主要的编程语言，使用 gRPC 分布式框架和 Protocol Buffers 通信协议，基于容器化微服务技术开发了一个物流管理系统，并通过对微服务的编排、封装、部署和发布，将该系统以微服务的形式部署在阿里云计算平台上，提供了基于云原生开发的企业物流管理应用。在 Windows 平台上基于 QT GUI 系统开发了 Client 端，实现了从客户端到服务端到数据存储端的全链路打通。并使用 google/benchmark、火焰图等相关工具对系统的正确性、可靠性、性能等指标进行了量化测试。

关键词：ERP，物流管理，云计算，云原生，Kubernetes

Design ERP logistics management services using cloud computing platforms

装 订 线

ABSTRACT

Enterprise Resource Planning (ERP) is the core content of industrial software, which integrates and manages information flow, logistics flow, data flow, and personnel flow through the coordination of modules such as warehousing and supply chain. The development technologies based on containerized microservices architecture, represented by Kubernetes, are currently the trend and mainstream in cloud computing application development, playing a popularizing and supportive role in Software-as-a-Service applications. The integrated management of logistics and supply chain plays a crucial role in enterprise resource management.

This paper focuses on Kubernetes clusters and containerized microservices architecture, researching topics such as cloud-native technology, ERP industrial software, and logistics management systems, as well as the integration of logistics management systems and cloud computing. It designs the architecture of Kubernetes clusters and their deployment on cloud platforms, including cluster abstraction hierarchy, intra-cluster communication architecture, and deployment forms on the cloud. Based on the research and relevant designs, a complete Kubernetes cluster is built on the Alibaba Cloud computing platform, and various containerized microservices are deployed within the cluster. This paper is based on software development under the containerized microservices architecture, utilizing the open-source Kubernetes tool to develop and run relevant software on cloud computing platforms. It conducts research and practical work on the application areas, functionalities, management, and deployment of Kubernetes, strictly following the guiding principles of software engineering for system modeling and application software development. The system uses C++ as the main programming language, the gRPC distributed framework, and Protocol Buffers communication protocol. Based on containerized microservices technology, a logistics management system is developed, and through the orchestration, encapsulation, deployment, and release of microservices, the system is deployed on the Alibaba Cloud computing platform in the form of microservices, providing an enterprise logistics management application based on cloud-native development. A client-side application is developed on the Windows platform using the QT GUI system, achieving end-to-end connectivity from the client to the server to the data storage. Quantitative tests on the correctness, reliability, and performance of the system are conducted using tools such as google/benchmark and flame graphs.

Key words： ERP, logistics management, cloud computing, cloud native, Kubernetes

目 录

1 引 言 1
1.1 企业资源计划 ERP 与工业软件 1
1.2 研究利用云计算平台设计 ERP 物流管理服务的意义 1
1.3 ERP 物流管理系统当前发展现状 2
1.4 本文所做的工作 2
1.5 论文的组织架构 3
2 云平台上的 Kubernetes 集群架构 4
2.1 云原生架构的概述 4
2.1.1 云原生的历史 4
2.1.2 云原生的技术体系 4
2.2 容器技术与微服务架构 4
2.2.1 容器技术 4
2.2.2 微服务架构 5
2.2.3 容器化微服务架构 5
2.3 Kubernetes 集群系统架构 5
2.3.1 Kubernetes 的基本概述 5
2.3.2 Kubernetes 抽象层级 6
2.3.3 Kubernetes 集群的调度算法 7
2.3.4 集群化部署的意义 7
2.4 Kubernetes 集群通信原理及通信架构模型 8
2.4.1 Kubernetes 集群通信原理 8
2.4.2 Kubernetes 集群内部通信 9
2.4.3 Kubernetes 集群对外通信 9
2.5 云计算平台上的 Kubernetes 集群部署形态 9
2.5.1 云计算平台上 Kubernetes 集群本体 10
2.5.2 云计算平台上的 Kubernetes 集群的附属服务 10
2.6 Kubernetes 集群运行的分布式系统通信框架与协议 11
2.6.1 Kubernetes 集群的分布式通信框架 gRPC 11
2.6.2 Protocol Buffers 序列化通信协议 11
3 物流管理系统的软件架构设计 13
3.1 物流管理系统的软件工程设计 13
3.1.1 物流管理系统的需求分析 13
3.1.2 物流管理系统的数据库与数据表设计 15
3.1.3 物流管理系统的类图设计 19
3.2 物流管理系统的应用架构设计 21
3.2.1 Server 端微服务部署设计 21
3.2.2 Client 端视图切换设计 22
3.2.3 Protocol Buffers 通信协议设计 23
4 物流管理系统的系统搭建与集群部署 26
4.1 本地编译与服务端开发流程 26
4.1.1 本地环境与软件清单 26
4.1.2 本地编译机编译环境配置 26
4.1.3 Server 端软件代码的编写 27
4.1.4 Server 端应用镜像的制作 29

装 订 线

4.2 在云计算平台上进行集群部署 30
4.2.1 云计算平台服务清单 30
4.2.2 在阿里云计算平台上搭建 Kubernetes 集群与部署微服务 30
4.2.3 在阿里云计算平台上搭建附属服务 32
4.3 Windows 下客户端的开发与使用 33
4.3.1 本地环境与软件清单 33
4.3.2 客户端软件代码的编写 34
4.3.3 客户端软件的编译及打包发布 34
4.4 物流管理系统的运行及使用说明 35
4.5 物流管理系统的测试 39
4.5.1 Server 端微服务的 Google Logging 日志系统 39
4.5.2 google/benchmark 单元测试 40
4.5.3 使用 perf 命令和火焰图进行系统级性能测试 42
4.5.4 集群高可用测试 44
5 结论和展望 46
5.1 结论 46
5.2 展望 46
参考文献 47
谢 辞 49

装 订 线

1 引言

1.1 企业资源计划 ERP 与工业软件

ERP（Enterprise Resource Planning，企业资源计划），是通过仓储和供应链等，对信息流、物流、财务流、人流等进行集成一体化的企业管理软件。一个成熟的 ERP 系统主要包括库存、采购、营销、BOM、车间任务管理、工艺、MRP、成本、人力资源、质量管理、经营决策、总账、自动分录、应收应付、固定资产等功能模块。

ERP 这个概念最早于 1990 年，由美国 Gartner Group 公司提出。然而，对企业的各项资源进行信息化管理的概念最早要追溯到上世纪 60 年代，由最初的 MRP（物料需求计划）发展而来。出于接近企业内部资源配置的需要，MRP 被广泛应用于制造业确定产品的加工进度和订货日程。在此之后，根据实际需求，MRP 不断演进，路径为：MRP—闭环 MRP—MRPII（企业制造资源计划）—ERP—ERPII，涵盖的需求范围逐渐扩大，信息化程度也随着增加。

ERP 系统在类别划分上属于工业软件，研究工业软件 ERP 对我国当前的制造业转型升级发展意义重大。工业软件由于其特殊性，需要与具体的工业门类密切结合，具有高度的专业融合特性。我国的工业软件发展水平受到时代的限制，发展时间与发达国家相比较短，与世界先进水平相比尚处于发展的初级阶段，在 CAD（建筑行业）、EDA（芯片设计行业）等各个专业领域基本没有软件自主权，每年都需要因此向其他公司缴纳一笔不菲的软件使用授权费用，并且受国际形势影响，这些软件的授权本身是不稳定的，因此为了摆脱这种受制于人的局面，大力发展 ERP 工业软件可以说是必然要求，与国家的科技发展战略大方向相符合。对此，我国有关部门也制订了相关的 ERP 行业标准[1]。

1.2 研究利用云计算平台设计 ERP 物流管理服务的意义

在物流和供应链管理的环节中，物流管理应用在当前的信息化社会中起到了核心作用。而对传统软件开发形态的革新就在于容器化微服务在应用开发与部署的应用。云计算技术是大规模数据应用的支柱和后盾，应用上云是当前国内外软件开发的大趋势，借用平台和算力实现弹性资源调度，是利用云计算技术为传统需求赋能的必由之路。

物流管理对于实体企业来说具有重要的地位。随着信息技术的快速发展，现代物流和供应链管理也逐渐向着信息化、智能化的方向进行转型升级。目前，基于容器化微服务架构技术的软件已经在许多工业产业门类中得到了广泛应用，如电网[2]、厨电[3]、财务管理[4]等。通过将信息化技术与传统产业相结合，是我国未来实现产业转型升级的必由之路。而在这一产业升级的过程中，使用互联网、云计算、大数据等技术为物流管理这样的传统产业赋能，已经成为了一个重要的举措和手段[5-7]。

以 Kubernetes 为代表的容器化微服务架构是当前工业级软件开发的一个趋势，依托于大规模公有/私有的云计算平台，在其上部署应用可以有效的保证应用的安全性和可靠性。而对于计算资源的调度核心要素在于弹性，通过实时监控资源的使用情况对应用进行资源调度、按需扩容，与传统固定服务器相比更加灵活，对资源的利用率更高。

ERP 工业软件虽然与大部分非专业用户无关，但其存在本身就是一个具有不可替代的刚需的庞大市场，具有较高的开发价值。然而，对于这样一个高度成熟的产业结构来说，打破巨头垄断可以说是极为困难的。仅就我国来说，工业软件的发展与国内市场的自主化标志着我国信息技术发展到了一定水平，值得国家和相关公司、从业人员对相关方面进行大力研究与投入。

1.3 ERP 物流管理系统当前发展现状

当前，随着互联网等相关技术的发展，物流管理在国内飞速发展，在较短的实践内完成了从全人工到半信息化到高度信息化、集成化的演进步骤。对于供应链的大多数环节，企业物流通过点对点的大规模集成仓储模式，形成了网状物流结构，从而在不同企业、不同实体站点之间进行实体运输的流转。而与此同时，信息也随之进行同步流动。

当前对于绝大多数企业来说，除了类似于京东物流、淘宝（菜鸟物流）等同时具有线上线下双重大规模优势的大型企业来说，其物流系统是依托于本公司集团的自有供应链，上下游全链条打通而形成在全国范围内的规模化物流网络，其他绝大多数企业的 ERP 物流管理系统都依托于集成服务供应商。行业内的服务提供商提供的通常是一体化的全套解决方案，实体物流渠道高度依赖国家公共服务或者私人物流合作商。

而在国外的相关行业现状，由于计算技术的发展，欧美发达国家的 ERP 软件更为发达，其应用服务上云的进度也更高，市场相对于国内也更加成熟。绝大部分相关领域的概念革新和标准制定都是由国外相关企业所提出，在相关领域占据了主导地位，先发优势明显。

在 ERP 工业软件领域，大部分知名的集成服务提供商都来自国外。来自德国的 SAP（思爱普）公司在国际市场上占据了较高的市场份额，是相关领域规模最大的 ERP 解决方案提供商。除了核心的企业应用套件外，其 SAP Cloud 产品线构成了其最具竞争力的部分，也是 ERP 应用与云计算平台相结合的先驱和行业风向标。

当前，我国的物流管理系统已经有了比较高的成熟度，在传统的软件行业生态下已经开发出了许多相适配的软件产品，并投入了实际使用，取得了较为良好的使用效果。大规模集成仓储模式发展而来的点对点网状物流结构成为了主要方式。

然而，在云计算高速发展的背景下，基于云原生和微服务架构，物流管理系统的去中心化成为未来的技术发展方向。将物流管理上云大大降低了系统的复杂度和进行统一管理的难度，根植于云上的微服务在应对高并发和复杂均衡等方面具有独特的优势，在最重要的可靠性方面在系统层面给予保证[8-10]。

1.4 本文所做的工作

本文按照选题要求，以利用云计算平台设计 ERP 物流管理服务为研究主题，主要做了以下工作：

（1）完成了选题的理论研究，包括以 Kubernetes 集群和容器化微服务架构为主题的云原生技术和 ERP 工业软件与物流管理系统，以及物流管理系统和云计算的融合形态等内容。

（2）完成了 Kubernetes 集群的架构设计，包括集群抽象层级，集群内外通信架构，集群上云部署形态等方面的设计。在阿里云计算平台上搭建了一个完整的 Kubernetes 集群，由一个 Master

节点和两个 Slave 节点组成。并在该云计算平台上使用 OSS 对象存储、ACR 容器镜像服务、PolarDB 云原生数据库等云平台提供的各项服务辅助集群的搭建与服务的使用。

（3）系统使用 C++作为主要的编程语言，使用 gRPC 分布式框架和 Protocol Buffers 通信协议，基于容器化微服务技术开发了一个物流管理系统，并通过对微服务的编排、封装、部署和发布，将该系统以微服务的形式部署在云计算平台上。在 Windows 平台上基于 QT GUI 系统开发了 Client 端，实现了从客户端到服务端到数据存储端的全链路打通。在软件开发过程中遵循软件工程的指导思想，进行软件系统的建模和相关应用软件的开发。

（4）使用专业的工具对系统进行验证与测试，包括基于 Google Logging 日志库的日志正确性验证与记录、C-S 端联通及功能可用性测试、使用 google/benchmark 进行核心功能的单元测试、使用 perf 命令和火焰图等工具进行系统整体的负载测试、集群负载与高可用性的测试等。

1.5 论文的组织架构

本文以利用云计算平台设计 ERP 物流管理服务为选题，论文总体的组织架构介绍如下：

第一章 引言，本章介绍了云计算平台上的 ERP 物流管理系统的研究背景和意义，对相关领域的发展及研究情况进行了说明。

第二章 云平台上的 Kubernetes 集群架构，本章中介绍了云计算平台上的 Kubernetes 集群架构，包括容器技术、集群调度算法、集群的抽象层级和通信架构，以及部署在阿里云计算平台上的集群形态和分布式通信框架。

第三章 物流管理系统的系统搭建与集群部署，本章按照软件工程的指导思想进行了物流管理系统的软件架构设计，包括软件系统、微服务架构、通信协议等模块的设计。

第四章 物流管理系统的系统搭建与集群部署，本章进行了基于前文设计的相关实验部分，详细阐述了容器化微服务架构下的服务端和客户端的开发流程，以及在云计算平台上搭建 Kubernetes 集群并部署微服务的具体实验过程，给出了构建部署完成的物流管理系统的使用说明，进行了单元测试和性能评估等测试项目。

第五章 结论和展望，本章对文中主要研究内容和所做工作进行了总结，并提出了后续可行的改进方向。

2 云平台上的 Kubernetes 集群架构

2.1 云原生架构的概述

2.1.1 云原生的历史

本选题与微服务密切相关，建立在云原生的技术体系之上。云原生是一个近年来在云计算的开发实践中被提出的概念，由美国公司 Pivotal 在 2013 年提出了“Cloud + Native”的组合，表示应用程序位于云上，而不是传统意义上的数据中心[11]。与传统应用开发在云计算时代试图“上云”，与云平台兼容不同，云原生从最开始就提出了一套根植于云计算平台的技术架构，从而充分利用云计算平台的弹性资源和分布式硬件体系。

2.1.2 云原生的技术体系

当前，工业界对于云原生的定义由以下四部分组成：微服务、DevOps、持续交付、容器化。

（1）微服务：微服务原生根植于云计算平台，与传统的大规模集成式软件不同，微服务将具体功能碎片化，降低不同函数之间的耦合程度，最终实现高内聚、低耦合的开发目标[12-13]。而以 Kubernetes 为代表的容器化微服务架构下的开发技术，是目前云计算应用开发趋势和主流，对于解决软件即服务应用具有普及和支撑作用[14-15]。相关的软件架构在大规模并行计算（MPP）和高性能计算（HPC）等领域上得到了广泛使用以充分发挥硬件效能[16]。

（2）DevOps：这个词是“Development + Operations”的缩写，意思是软件开发和系统运维，强调的是将软件开发和系统运维的界限模糊化，达到快速定位，快速响应的目的[17]。DevOps 的核心是自动化构建，通过一系列的自动测试流程进而达到快速部署，这个过程也被称为 CI/CD，即持续集成、持续交付和持续部署。目前普遍使用 Jenkins 作为 CI/CD 的工具，而 Jenkins 和 Kubernetes 的结合也是云原生开发的重要工具[18]。

（3）持续交付：这个概念强调的是保持软件产品时刻处于一个可以发布的状态，类似于在软件工程中的敏捷开发和快速迭代，对应用在上线之后持续在原有的基础上发布新功能。而与传统的人工检测和部署不同，持续交付将这一过程通过流水线的方式，在云上自动化完成。在这个过程中需要快速的资源分配和释放，从而需求虚拟化技术将这一过程轻量化。

（4）容器化：以 Docker 为代表的容器化技术在开发过程中的优点在于：可以跨平台，无需对开发、编译、运行环境进行复杂的配置。概括来说，就是与底层具体系统架构解耦合。云计算的基础就是建立在虚拟化的容器集群之上的，由管控统一进行资源的分配调度。

2.2 容器技术与微服务架构

2.2.1 容器技术

容器技术，在本质上来说，是一种对操作系统内核进行轻量级虚拟化的技术。有两大核心机制组成：Namespace 和 Cgroup[19]。

Namespace（命名空间）是在 Linux 上的逻辑资源隔离机制，负责的是对运行在系统中的进程资源进行隔离，每一个 Namespace 都有其对应的内核资源。同一个 Namespace 下的进程视作在

同一个实体系统下运行，而不同 Namespace 下的进程在逻辑上则没有任何关系，被认为分属于两个不同的系统，运行于两台不同的计算节点上。

Cgroup 全名为 Control Groups，中文译名为控制组群，是 Linux 系统官方提供的物理资源隔离机制。该机制是相对于 Namespace 更深一层的，实现在内核这个层面对进程组的资源进行精细化控制的机制，如计算资源的限制、优先级的控制、进程组的状态控制和隔离等。

2.2.2 微服务架构

微服务架构由 SOA 架构发展而来，与传统的面向服务的 SOA 架构相比，取消了企业服务总线 ESB 的架构设定，用户直接访问到微服务[20-21]。微服务架构避免了传统集中式架构的不同应用之间互相耦合的问题，极大程度的降低了系统复杂性和开发难度。

分布式部署的不同微服务之间需要进行相关的通信，目前主流是通过 RPC（Remote Procedure Call，远程过程调用）框架来进行相关的数据传输和通信，此外 HTTP 等通信协议在某些情况下也被使用。

微服务架构核心的特征是去中心化，微服务之间互不干扰，拥有自己的进程、数据库等系统资源，被分布式的部署在不同的计算节点上，互相之间没有耦合性。不同微服务之间可以使用不同的语言进行开发，每个微服务可以被独立的部署、测试、升级、扩展、更新而不受其他微服务的影响。这种可扩展性是在数据层面和业务逻辑层面共同存在的。

2.2.3 容器化微服务架构

基于容器技术的微服务架构是当前云原生开发的主流，目前普遍采用的技术路线是 Docker + Kubernetes，将微服务部署在 Kubernetes 创建的容器组 Pod 中[22-23]。Kubernetes 作为开源的容器集群管理系统，将应用部署、节点监控等各个环节的流程简化，且具有较高的性能表现，是优秀的集群管理工具。此外，Kubernetes 具有智能调度、负载均衡等特性，能够有效应对容器组失效和节点负载过高等情况[24]。在默认的调度策略之外，还可以根据实际需求自定义调度算法，具有高度客制化的特点[25-26]。而具体到对整体程序的性能评估和把控，目前业界广泛使用的一种方式是通过火焰图（FlameGraph）对程序嵌套调用的函数堆栈在 CPU 上的运行时间进行查看[27-28]。

2.3 Kubernetes 集群系统架构

2.3.1 Kubernetes 的基本概述

Kubernetes 是一个容器化部署并对应用进行更新维护的应用，缩写为 k8s。k8s 的应用场景通常为在云计算平台上对应用进行容器化管理。

对应用进行容器化部署的优点很明显：每个容器之间互相隔离，从而使得应用与底层基础设施解耦，便于在不同规格的机器之间进行迁移。且容器相对于虚拟机，占用资源较少的同时部署快，在使用上更加轻量级。容器是进程级操作，资源消耗相对较小。而虚拟机是系统级行为，资源消耗相对较大。且与传统的在操作系统上安装应用相比，将应用打包成镜像运行在容器中更为方便。

2.3.2 Kubernetes 抽象层级

一个完整的 Kubernetes 集群，可以自上而下分解成几个抽象的层次结构，集群的抽象层次结构见图 3.1。

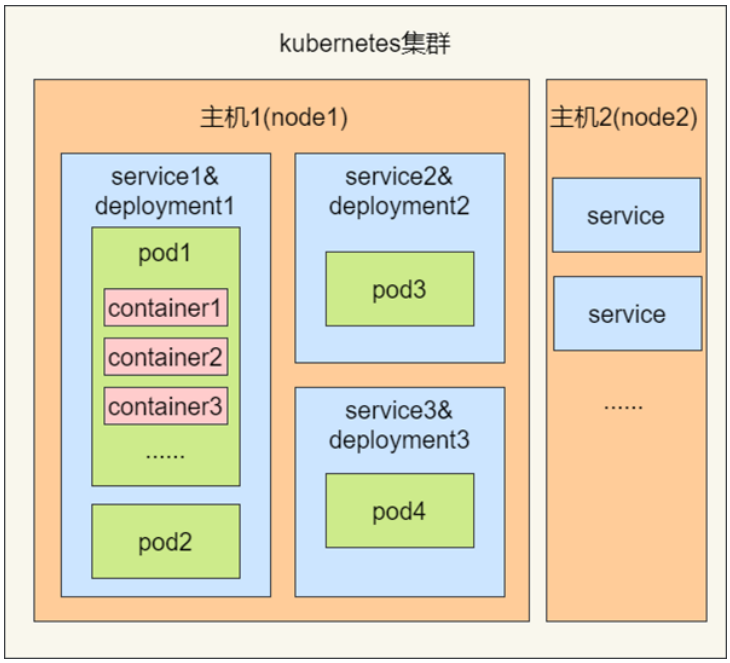

图 2.1 Kubernetes 集群的抽象层级

从图 2.1 可见，Kubernetes 集群是一个分布式部署的多机集群，针对的是分布式集群场景的核心：如何去管理底层的资源，从而方便的去帮助管理员去部署应用。而 Kubernetes 本身也是一个 C-S 架构的模型。 云计算的基础就是建立在虚拟化的容器集群之上的，由管控统一进行资源的分配调度。

（1）Node（节点）：一个具体的物理机或者虚拟机（一个工作节点），作为集群中最大的可以被调度的资源。Node 在本地多机集群上通常指实体机，在云计算平台上则通常指可以进行弹性资源调度的云服务器。一个功能完整、正常部署且状态良好的 Kubernetes 集群通常有一个 Master 节点和多个 Slave 节点。普遍的实现方式是 Master 节点上做 taint 标识，不用做应用 Pod 的调度节点，而是起到运行整体调度的作用。在 Master 节点上运行的是 api-Server、proxy、scheduler 等集群整体的调度 Pod，而 Slave 节点上通常运行实际应用的 Pod。

（2）Deployment（部署）：定义并管理一组 Pod，Pod 之间的地位是对等的。与 Pod 属于逻辑上的上下级关系，主要作用是对 Pod 中的应用进行统一管理维护，如在线升级、版本回滚等操作。其下属的 ReplicaSet 起到定义 Pod 的个数，并自动拉起状态异常的 Pod 等作用。

（3）Service（服务）：Service 在抽象层级上是与 Deployment 并行的操作，在实际的部署场景中，通常也是将 Service 与 Deployment 一同注册。Service 主要是对 Pod 内的具体服务进行访问和代理上的抽象。因为 Pod 是会随着集群的调度而动态变化的，随时可能被销毁与重建，其绑定的 Pod ip 也是由集群分配的，因此需要由 Service 来对外暴露固定的 ip 和端口，从而由 Service 将

请求转发到对应的 Pod 上。

（4）Pod（容器组）：是一个 k8s 可以去部署的最小单元，一个 Pod 里可以有多个容器，容器之间有一些资源可以共享（如网络），独立资源（进程 pid，文件系统等）。作为 Kubernetes 集群的原子调度单位，Pod 在操作系统层面上不存在，是 k8s 集群上的一个抽象化的逻辑概念。应用以容器化的方式运行在 Pod 中，并接受 Kubernetes 集群的调度。

（5）Container（容器）：在 Pod 中实际运行的应用，以容器的形式被部署在相应的容器组中。容器的本质是一个系统被隔离、资源受限的进程，基于云原生的理念，容器与应用是高度绑定的，即“应用就是容器，容器就是应用”。较为普遍的做法是除非应用之间有亲密的关系（如相互间的文件调用与本地通信等），否则在一个 Pod 中仅部署一个应用容器，以便于以最小粒度对应用进行调度。

2.3.3 Kubernetes 集群的调度算法

Kubernetes 集群的调度算法是 K8s Scheduling：有一个新创建的 Pod，寻找一个最合适的节点去运行这个 Pod，若节点均不可用则结束调度。

集群的调度算法为评分机制，具体流程为：filtering（过滤、预选调度）->scoring（评价算法、优选调度）[29]。调度算法策略均可以自己定制，常见的选择节点的方式：节点的内存、CPU 占用率等资源满足要求，或者对对应特殊资源的需求打上标签（如 GPU 计算能力）。当节点计算资源紧张的时候，也会打对应的标签，Kubernetes 集群也会逐渐将 Pod 从对应节点中取出，防止该节点挂掉。

一种典型的调度方法是 bin packing 调度方法：向一个 Node 调度到满为止，之后再调度到其他节点，该方法最主要的好处是节约节点资源。也存在其他的定制化调度方法，集群中国的调度框架为插件的形式，通过设计 Scheduling + binding 的流水线可以自定义调度方法。

作为调度框架的一部分，Kubernetes 集群中存在着对 Pod 的 kill 机制，即 Eviction 机制，该机制根据资源请求、服务优先级、资源请求的相对占比等指标来进行判断。类似于 Linux 操作系统中对进程的通过 OOM 进行删除的机制。

2.3.4 集群化部署的意义

集群的本质是在分布式的场景下对服务做负载均衡，以达到搭建分布式集群的根本目标：使得服务性能随着计算资源而线性增长。在这个过程中，通过尽可能少的降低因负载不均衡导致的资源消耗，从而尽可能提高计算资源的利用率。

集群规模会很大程度上影响集群中每个节点的性能表现，这一点在大规模、高负载的计算集群上表现的尤为明显。通常来说，对于多个理论计算性能相同的集群，通常是单个节点性能较高、实际节点数较少的集群性能表现更好，即“10CPU*1 > 1CPU*10”。这是由于在集群中部署的节点越多，集群越大，调度开销就越大，并且在具有庞大规模的集群中，对资源的分配调度是一个极为困难的工作。

运行在集群上的服务挂掉的情况：当系统长时间负载过高时，由 OOM 机制（out of memory，Linux 操作系统上通行的内存管理机制）去将负载过高的进程 kill 掉，释放计算资源以降低系统

负载。在这种情况下，依据进程优先级，占用大量计算资源（CPU 时间、内存等）的所部署的微服务进程是很有可能被优先 kill 掉的对象。这时候，k8s 集群检测到服务处于（挂起）状态，就需要重新部署该服务，生成新的 Pod，部署在集群中。

2.4 Kubernetes 集群通信原理及通信架构模型

2.4.1 Kubernetes 集群通信原理

在搭建 Kubernetes 集群的过程中，需要将对应的 Node 节点加入集群，在集群中构建虚拟化的容器通信网络，从而使 Node 节点可以互相连接。通过在 Master 节点上安装网络插件，使得 Node 节点之间可以互相通信，其他的 Slave 节点可以加入该集群，集群通过网络插件进行跨主机容器的通信和调度管理。

网络插件基于网络容器接口 CNI（container network interface）所对应的标准，目前比较流行的的网络插件有 flannel、calico 等。在本例中，选用 flannel 作为搭建 Kubernetes 集群的网络插件，这是一款轻量级的网络插件，默认使用 flannel Vxlan 构建通信隧道实现虚拟网络（Overlay），在性能和复杂度上取得了很好的平衡。

在 Kubernetes 集群中有一些特殊的 IP 地址与 port 端口，对应关系

（1）Node IP：物理节点的 IP 地址。对应的 nodePort 是对集群外暴露的 IP 地址，完成的是集群与外网的通信。

（2）Cluster IP：Service 对应的虚拟 IP 地址。对应的 port 是 Service 对外暴露的端口，集群内部通信所访问的端口。

（3）Pod IP：Pod 对应的虚拟 IP 地址。对应的 targetPort 和 containerPort 是 Pod 对外暴露的端口，来自 nodePort 和 port 的数据包均被转发到该端口（即为制作镜像时 EXPOSE 的端口）。在部署服务时将 targetPort 映射到 containerPort。

在 Kubernetes 集群中，通信方式分为集群内部相互通信和集群对外通信，集群的整体通信架构见图 2.2。

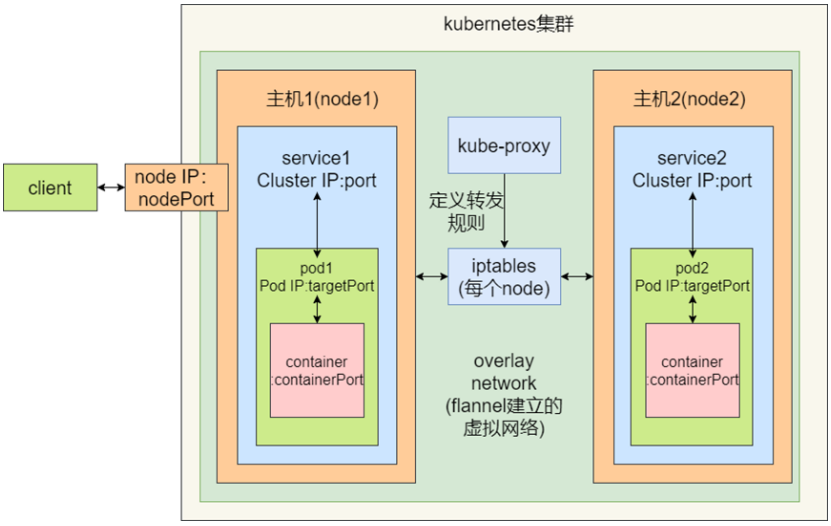

图 2.2 Kubernetes 集群网络通信架构

2.4.2 Kubernetes 集群内部通信

Kubernetes 集群内部的相互通信是通过 kube-proxy 来完成的。kube-proxy 在 Linux 上使用 iptables 的方式来实现网络的代理和转发（forward），在 iptables 中注册一系列的转发与 NAT 规则，从而实现对应的数据包的正确转发。

Pod 在发送信息的过程中，数据包经过 iptables，通过 SNAT 的方式，将 source IP 由 Pod IP 修改为 Service 提供的 Cluster IP，从而使得对 Pod 的动态调度不影响服务的对外暴露。而 service 提供的 Cluster IP 收到数据包时，同样由 iptables 将 Destination IP 由 Cluster IP 修改为对应的 Pod IP，使得 Pod 可以实际接收到数据包。

2.4.3 Kubernetes 集群对外通信

在制作镜像时，指定服务的通信类型为 NodePort，同时在部署该服务时需要指定集群内服务对外暴露的 nodePort 端口。集群内的所有节点都监听该端口，请求到任意节点的 Node IP:nodePort 均可访问到集群内对应服务。因此，只需要得到一个固定的静态公网 IP 即可实现外网对集群内已部署服务的稳定访问。

因此，Client 通过公网访问集群内服务的流程是：Client 请求访问 Master 或 Slave 中的任意 Node 的 nodePort 端口，当流量进入宿主机对外暴露的 nodePort 端口后，会被转发给 Service 的 Cluster IP，然后通过 iptables 转发到对应的 Pod。

2.5 云计算平台上的 Kubernetes 集群部署形态

在公有云计算平台上，Kubernetes 集群通常部署在内网，通过建立在内网上的虚拟网络进行节点之间的相互连接。与公网 IP 相比，部署在云计算平台上的集群、节点、微服务、数据库、容器镜像服务、存储服务等通过相同的内网进行连接，在连接过程中极大的提高了安全性。除了部署在内网的由多台 ECS 云服务器组成的 Kubernetes 集群以外，还需要额外部署一些辅助服务，以使得部署的集群可以调用这些服务。而部署在云计算平台内网环境下的集群，想要对外部通过公网进行访问，具体流程见图 2.3。

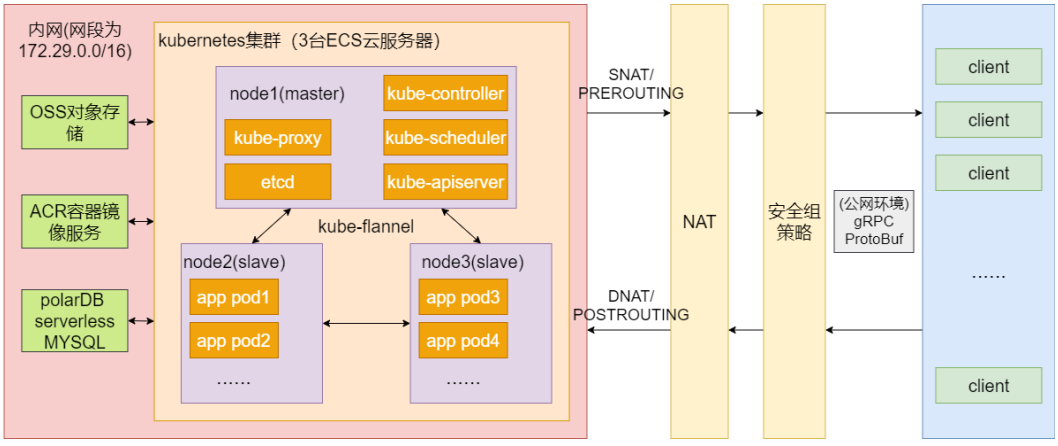

图 2.3 Kubernetes 集群在云计算平台上的整体网络拓扑结构

2.5.1 云计算平台上 Kubernetes 集群本体

（1）安全组策略

在内网的最外层，由 ECS 云服务器的安全组策略进行处理。对于外部公网来的流量，由组策略对数据包进行第一层清洗，保证只有来自指定 IP 和通向对应开放端口的数据包才能进入。安全组默认的部署策略是对本机发送流量不做清洗，仅过滤外部流量。因此需要根据需要对外开放对应的 IP 和端口。

（2）NAT（网络地址转换）

在安全组策略的内层，为了使位于内网的 ECS 云服务器可以与外网相连接，云平台使用了 NAT（Network Address Translation，网络地址转换）技术，基于 iptables，对于从公网进入的数据包，使用 DNAT（目标地址转换），在 PREROUTING 链上将目的地址从公网替换为内网地址；而对于从内网节点向公网发送的数据包，通过 SANT（源地址转换），在 POSTROUTING 链上将源地址转换为对外暴露的公网地址。NAT 层存在的意义是将内网环境和公网环境进行隔离，使得在保留内网服务之间的高效率通信的同时，实现内网和公网之间的通信。不需要在服务器上部署实际的物理网卡，提高了整体云服务的虚拟性。

（3）ECS 云服务器

每一台云服务器都可以被看作一台独立的节点，被部署在 Kubernetes 集群中。服务器的实际运行环境是在内网，以华东（上海）地区为例，其网段为 172.29.0.0/16，这种特性便于在内网的节点和服务间进行高速的网络通信，而不需要经过一次“内网——公网——内网”的通信转发。在节点服务器上没有部署实际的公网物理网卡，而是通过 NAT 与公网通信，提高了整体云服务的虚拟性。

2.5.2 云计算平台上的 Kubernetes 集群的附属服务

（1）OSS 对象存储

对象存储（Object Storage Service）是阿里云提供的针对具体的文件对象进行存储的服务。服务器上实际的存储资源是系统盘和额外挂载的数据盘，在计算集群上开辟大量空间用于对象的存储是不经济的。较为优秀的方法是将大对象存储在专用的存储集群上，这样使得集群内的每个节点均能够以相同流程上传和下载对象，如配置文件、库文件、可执行文件等，为分布式的开发和部署提供了便利。

（2）ACR 容器镜像服务

容器镜像服务（Alibaba Cloud Container Registry）是阿里云提供的基于云原生理念，进行镜像的分发与托管的服务。通过使用容器镜像服务，对仓库中的容器镜像进行统一管理。在开发阶段的核心思想是编译环境和部署环境分离，保证部署环境的纯净性。编译环境可以是任何合法的环境，在本地或云平台上均可。在编译环境中编译可执行文件和制作镜像之后，将制作完成的容器镜像打上对应的 tag 标签，上传到 ACR 容器镜像服务提供的内网仓中。这样就能够在 Kubernetes 集群中直接从内网仓获取到所需的容器镜像，从而可以直接在集群中部署。

（3）云原生数据库 PolarDB Serverless MySQL

PolarDB 是阿里云提供的基于云原生的数据库，与服务形式部署在 Kubernetes 集群中的数据库服务相比，PolarDB 不受集群的调度，使得 MySQL 服务不占用 Kubernetes 集群内部的虚拟网络和计算资源，提高了集群的性能。且该服务自动提供主从备份、容灾和快照等技术，与个人部署的数据库服务相比极大程度上提高了数据存储的可靠性。

2.6 Kubernetes 集群运行的分布式系统通信框架与协议

2.6.1 Kubernetes 集群的分布式通信框架 gRPC

在分布式场景下，基于容器化微服务和云原生架构进行应用开发需要解决的一个重要问题是将原本的单个应用程序分解为一组并行的微服务，而微服务和客户端及相互间进行通信的过程关键技术就是进程间的通信。这种通信可能是跨域乃至跨平台的，因此是在分布式场景下开发微服务的技术难点。

RPC（Remote Procedure Call），中文译名为远程过程调用，是分布式系统常用的通信技术。系统设计阶段要求构建系统通信模型，基于以上容器化微服务的开发场景，本选题使用了 gRPC 框架进行云侧微服务和端侧客户端之间的通信。gRPC 是由 Google 公司开发的开源、跨语言的 RPC 框架，基于 HTTP/2 协议和 Protocol Buffers 序列化协议进行开发。gRPC 框架将语义定义、网络传输、编码解码等 RPC 的核心功能集成在该框架下，并在此基础上极大程度的提高了分布式服务的开发效率[30]。

具体方法是在服务端实现 Protocol Buffers 协议接口中所定义的 RPC，而在客户端通过 RPC 方式使用类似调用本地对象的操作跨平台调用服务端的微服务接口。并且数据编码/解码的过程是跨平台的，易于创建异构分布式应用和服务，分布式框架示意图如图 2.4 所示。

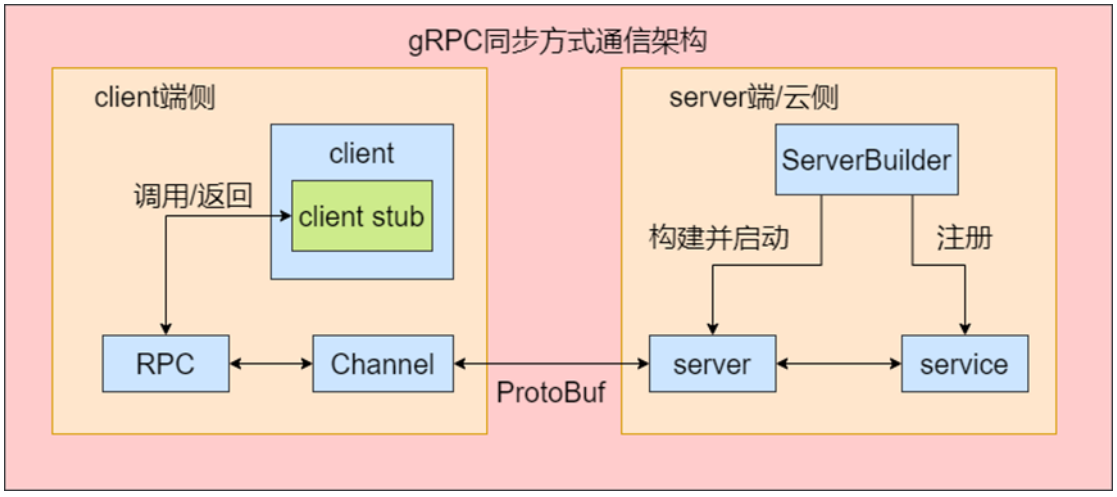

图 2.4 gRPC 分布式框架同步方式通信架构

2.6.2 Protocol Buffers 序列化通信协议

Protocol Buffers 是由 Google 公司开发的序列化通信协议。在本例中，gRPC 框架与 Protocol

Buffers 配合使用，对通信过程中的数据进行序列化/反序列化，将数据转化为二进制格式传输，并在对应的 Server 端和 Client 端完成编码与解码的对应工作。二进制序列化编码与 JSON 等文本格式的内容编码方式相比性能较高，单位信息长度仅为 JSON 的 1/3 左右，且该协议使用对称加密的方式确保数据在网络传输过程中的安全性。

仅需要通过编写对应格式的.proto 文件即可通过 Protocol Buffers 自动生成对应语言版本的可调用双端通信协议，且这种协议是无视语言和跨平台的。在这个过程中生成 Client 端使用的客户端存根 Stub，由 Stub 向 Client 端提供对应的 RPC 调用接口，具体通信流程如图 2.5 中所示。

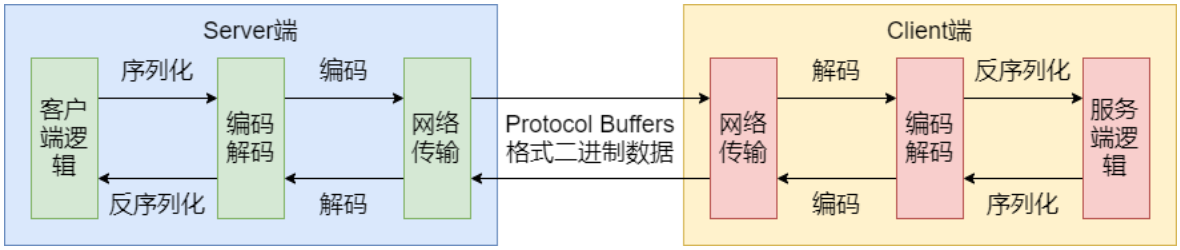

图 2.5 Protocol Buffers 序列化协议通信流程图

装 订 线

3 物流管理系统的软件架构设计

3.1 物流管理系统的软件工程设计

3.1.1 物流管理系统的需求分析

通过 UML 用例建模技术，可以成结构化的描述系统的功能需求，在宏观上构建出系统的功能模型。通过对系统中典型用例的分析，可以有效的将用户的需求与开发者的功能实现联系起来。

（1）游客

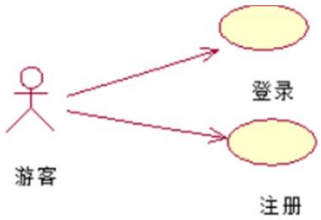

图 3.1 游客用例图

如图 3.1 所示，游客有以下主要用例：

注册：模块包括普通用户注册、站点工作人员注册和管理员注册。其中站点工作人员注册需要验证密钥和站点 id，管理员注册需要验证特有密钥。

登录：模块包括普通用户登录、站点工作人员登录和管理员登录。

（2）普通用户

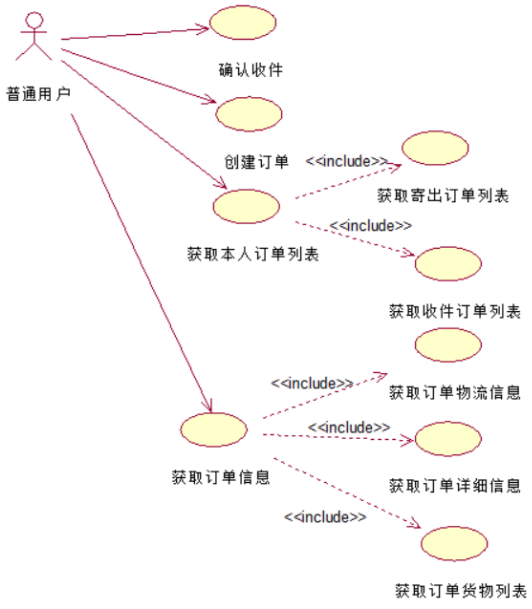

图 3.2 普通用户用例图

如图 3.2 所示，普通用户有以下主要用例：

获取本人订单列表：包括获取本人的寄件列表和收件列表两个部分，普通用户的权限只能查看自己的订单。

获取订单信息：根据订单号获取订单的信息，包括全部的详细信息，订单物流信息和货物列表等。用户只有查看本人相关订单信息的权限。

创建订单（寄件）：用户填写订单信息，选择货物加入列表，将填写完成的订单信息上传。

确认收件：用户确认收货，修改对应订单状态为“已完成”。

（3）站点工作人员

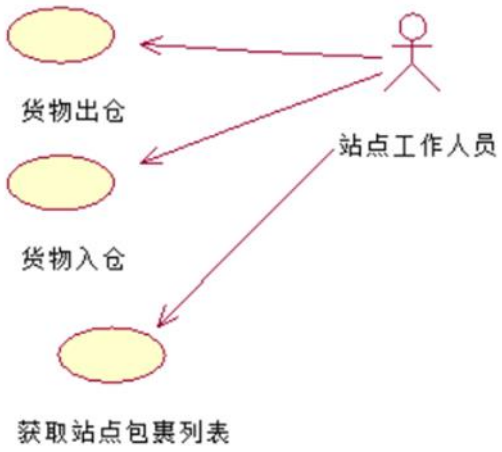

图 3.3 站点工作人员用例图

如图 3.3 所示，站点工作人员有以下主要用例：

货物入仓：将某订单经过查验进入仓库，记录在本地仓库存储数据中，同时更新物流信息。

货物出仓：将本地仓库中某订单发往别处，从本地仓库存储数据中移除，同时更新物流信息。

获取站点包裹列表：查看本仓库中所有积压订单，站点工作人员只有查看本仓库中订单的权限。

（4）系统管理员

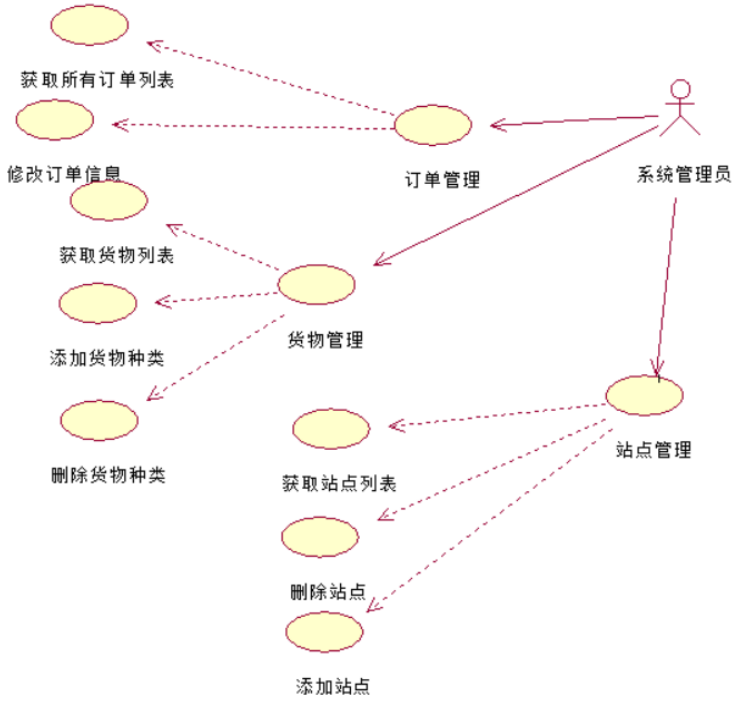

图 3.4 系统管理员用例图

如图 3.4 所示，系统管理员有以下主要用例：

订单管理：管理员可以获取所有订单列表，并手动修改订单的各项信息。

货物管理：管理员可以查看系统中存储的所有货物列表，并添加或删除对应的货物。

站点管理：管理员可以查看系统中接入的所有站点列表，并添加或删除对应的站点。

3.1.2 物流管理系统的数据库与数据表设计

本系统采用了 MySQL 数据库，这是一款兼顾轻量级和高性能的关系型数据库，并实际部署在阿里云云计算平台上的 PolarDB Serverless MySQL 云原生数据库中。

根据上述系统需求，创建名称为 Logistics_System 的数据库，并在该数据库实例中创建 9 张数据表，建立数据库的 E-R 图（实体-关系图）如图 3.5 所示。

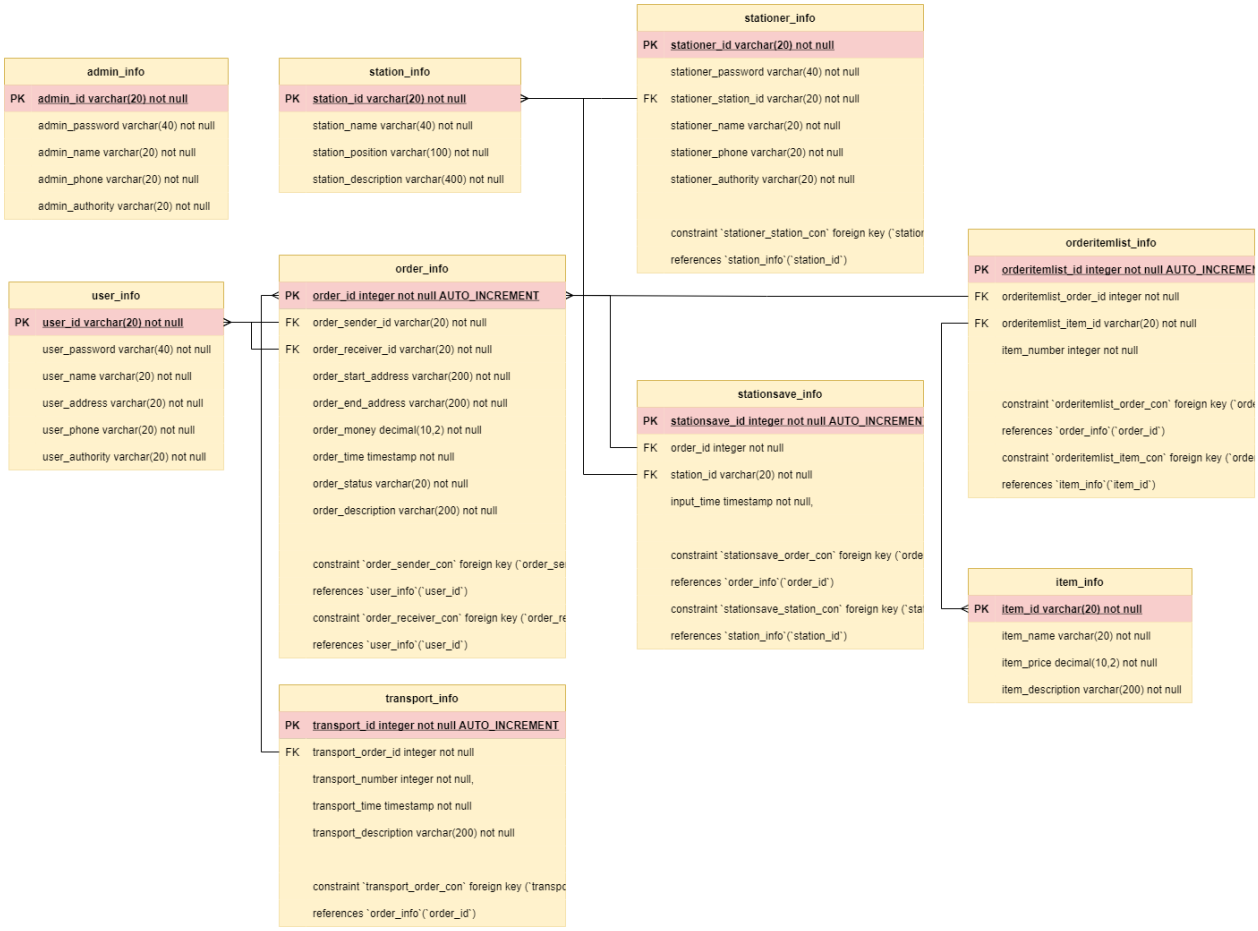

图 3.5 部署在云平台上的 MySQL 数据库的全局 E-R 图

根据图 3.5 中的实体-关系图，在数据库中建立 9 张数据表。

（1）普通用户信息表（user_info）：用于记录普通用户的个人信息，具体格式见表 3.1。

表 3.1 普通用户信息表（user_info）

字段名	类型	约束	含义
user_id	varchar(20)	主键，not null	用户 id
user_password	varchar(40)	not null	用户密码
user_name	varchar(20)	not null	用户姓名
user_address	varchar(20)	not null	用户默认地址
user_phone	varchar(20)	not null	用户手机号
user_authority	varchar(20)	not null	用户权限

（2）站点工作人员信息表（stationer_info）：用于记录站点人员的个人信息，具体格式见表 3.2。

表 3.2 站点工作人员信息表（stationer_info）

字段名	类型	约束	含义
stationer_id	varchar(20)	主键，not null	站点人员 id
stationer_password	varchar(40)	not null	站点人员密码
stationer_station_id	varchar(20)	外键依赖`station_info`(`station_id`)，not null	站点序列号
stationer_name	varchar(20)	not null	站点人员姓名
stationer_phone	varchar(20)	not null	站点人员手机号
stationer_authority	varchar(20)	not null	站点人员权限

（3）系统管理员信息表（admin_info）：用于记录系统管理员的个人信息，具体格式见表 3.3。

表 3.3 系统管理员信息表（admin_info）

字段名	类型	约束	含义
admin_id	varchar(20)	主键，not null	管理员 id
admin_password	varchar(40)	not null	管理员密码
admin_name	varchar(20)	not null	管理员姓名
admin_phone	varchar(20)	not null	管理员手机号
admin_authority	varchar(20)	not null	管理员权限

（4）订单信息表（order_info）：用于记录订单的信息，具体格式见表 3.4。

表 3.4 订单信息表（order_info）

字段名	类型	约束	含义
order_id	integer	主键，not null，AUTO_INCREMENT	订单 id
order_sender_id	varchar(20)	外键依赖`user_info`(`user_id`)，not null	寄件人 id
order_receiver_id	varchar(20)	外键依赖`user_info`(`user_id`)，not null	收件人 id
order_start_address	varchar(200)	not null	寄件地址
order_end_address	varchar(200)	not null	收件地址
order_money	decimal(10,2)	not null	订单总价
order_time	timestamp	not null	下单时间
order_status	varchar(20)	not null	订单状态
order_description	varchar(200)	not null	订单备注

（5）货物信息表（item_info）：用于记录货物信息，具体格式见表 3.5。

表 3.5 货物信息表（item_info）

字段名	类型	约束	含义
item_id	varchar(20)	主键，not null	货物 id
item_name	varchar(20)	not null	货物名
item_price	decimal(10,2)	not null	货物价格
item_description	varchar(200)	not null	货物备注

（6）订单货物信息列表（orderitemlist_info）：用于记录订单中的货物数量列表，具体格式见表 3.6。

表 3.6 订单货物信息列表（orderitemlist_info）

字段名	类型	约束	含义
orderitemlist_id	integer	主键，not null，AUTO_INCREMENT	货物 id
orderitemlist_order_id	varchar(20)	外键依赖`order_info`(`order_id`)，not null	货物名
orderitemlist_item_id	decimal(10,2)	外键依赖`item_info`(`item_id`)，not null	货物价格
item_number	integer	not null	货物备注

（7）运输链路流转列表（transport_info）：用于记录订单在链路上的物流信息，具体格式见表 3.7。

表 3.7 运输链路流转列表（transport_info）

字段名	类型	约束	含义
transport_id	integer	主键，not null	流转 id
transport_order_id	integer	外键依赖`order_info`(`order_id`)，not null	流转订单 id
transport_number	integer	not null	链路流转次数
transport_time	timestamp	not null	流转记录时间
transport_description	varchar(200)	not null	本次流转描述

（8）工作站点信息列表（station_info）：用于记录站点信息，具体格式见表 3.8。

表 3.8 工作站点信息列表（station_info）

字段名	类型	约束	含义
station_id	varchar(20)	主键，not null	站点 id
station_name	varchar(40)	not null	站点名
station_position	varchar(100)	not null	站点地址
station_description	varchar(400)	not null	站点备注

（9）站点滞留订单列表（stationsave_info）：用于记录当前存储在该站点的订单列表，具体格式见表 3.9。

表 3.9 站点滞留订单列表（stationsave_info）

字段名	类型	约束	含义
stationsave_id	integer	主键，not null，AUTO_INCREMENT	存储记录 id
order_id	integer	外键依赖` station_info`(`station_id`)，not null	订单 id
station_id	varchar(20)	外键依赖`order_info`(`order_id`)，not null	站点 id
input_time	timestamp	not null	入库时间

3.1.3 物流管理系统的类图设计

每个类具有不同的属性和方法，且类和类之间具有不同的关系。类图中不同类具有三种类型，分别为实体类（entity class）、边界类（boundary class）、控制类（control class）。每个类中具体的属性和方法在类图中展示出来。

在系统中，每个类之间具有不同的关系包括关联、组合、聚合等等。根据关联关系将不同的类连接起来，按角色和功能进行划分，形成一个个类图，系统类图的设计如下图 3.6-3.9 所示。

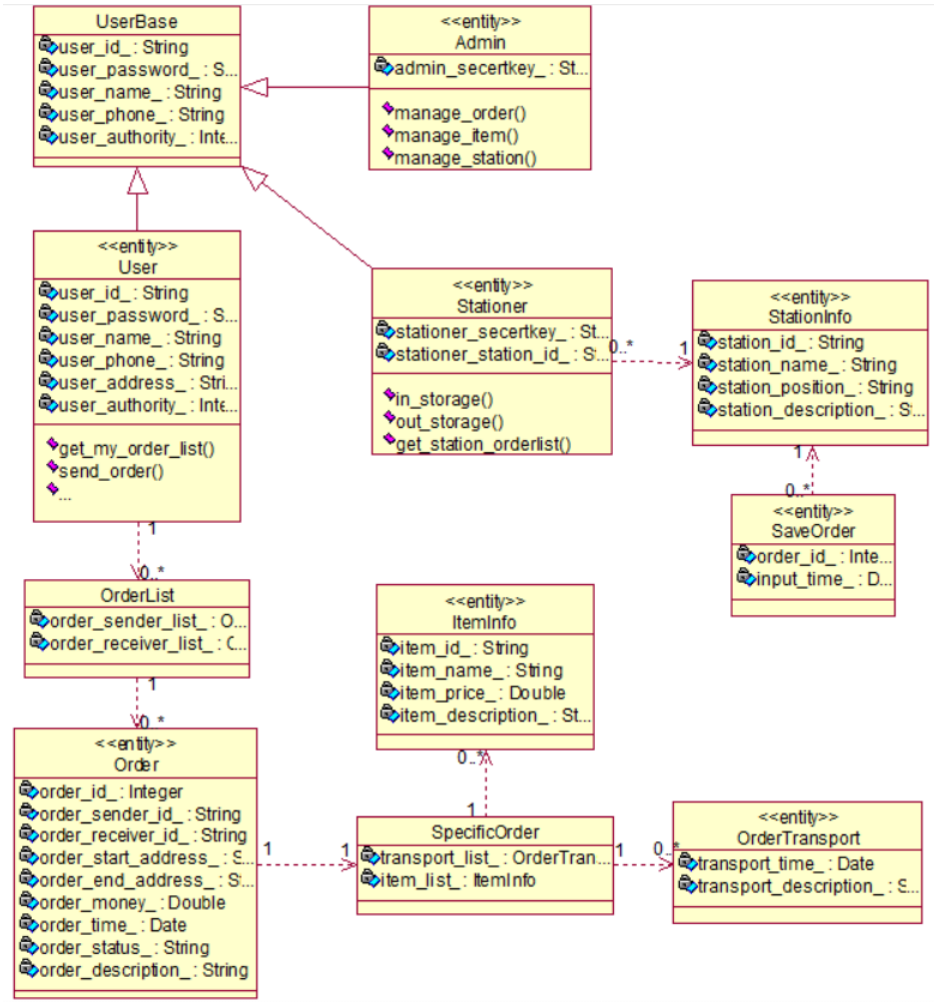

图 3.6 E-R 图映射 Server 端实体类图

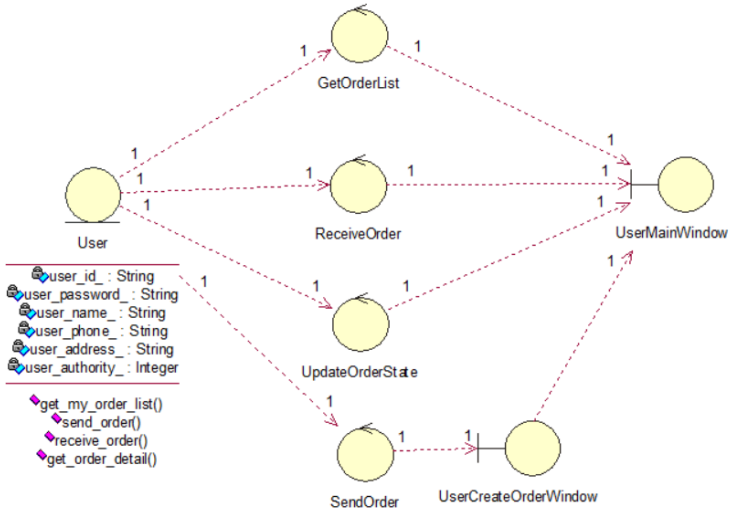

图 3.7 User Client 端交互类图

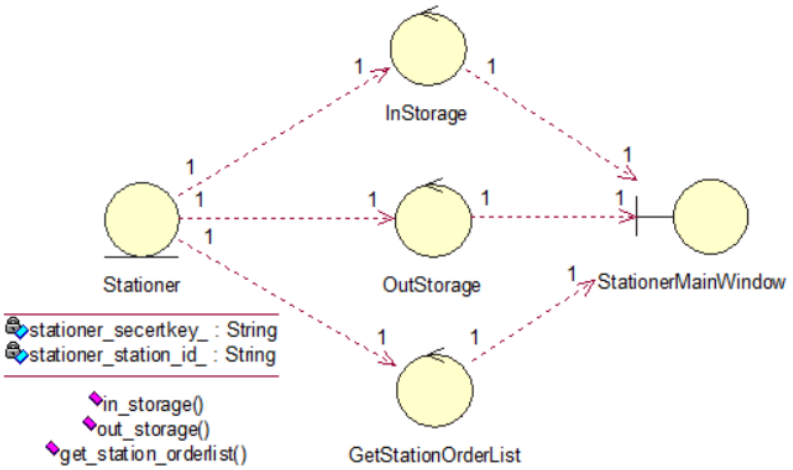

图 3.8 Stationer Client 端交互类图

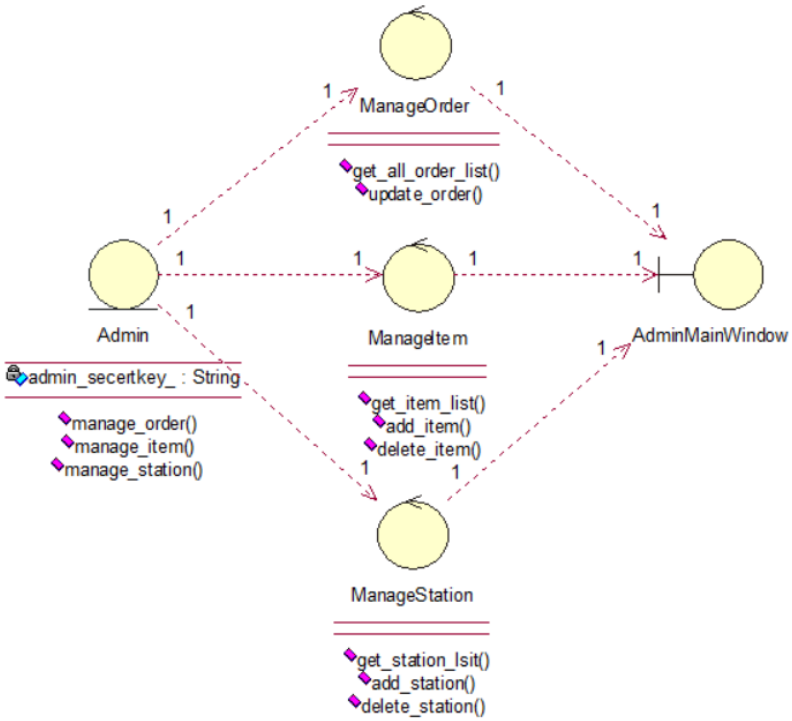

图 3.9 Admin Client 端交互类图

3.2 物流管理系统的应用架构设计

本选题的具体实践过程本质上是开发一个 C-S 架构的应用，将 Server 端部署在云计算平台的 Kubernetes 集群中，并通过本机的 Client 端对云端服务进行访问。

3.2.1 Server 端微服务部署设计

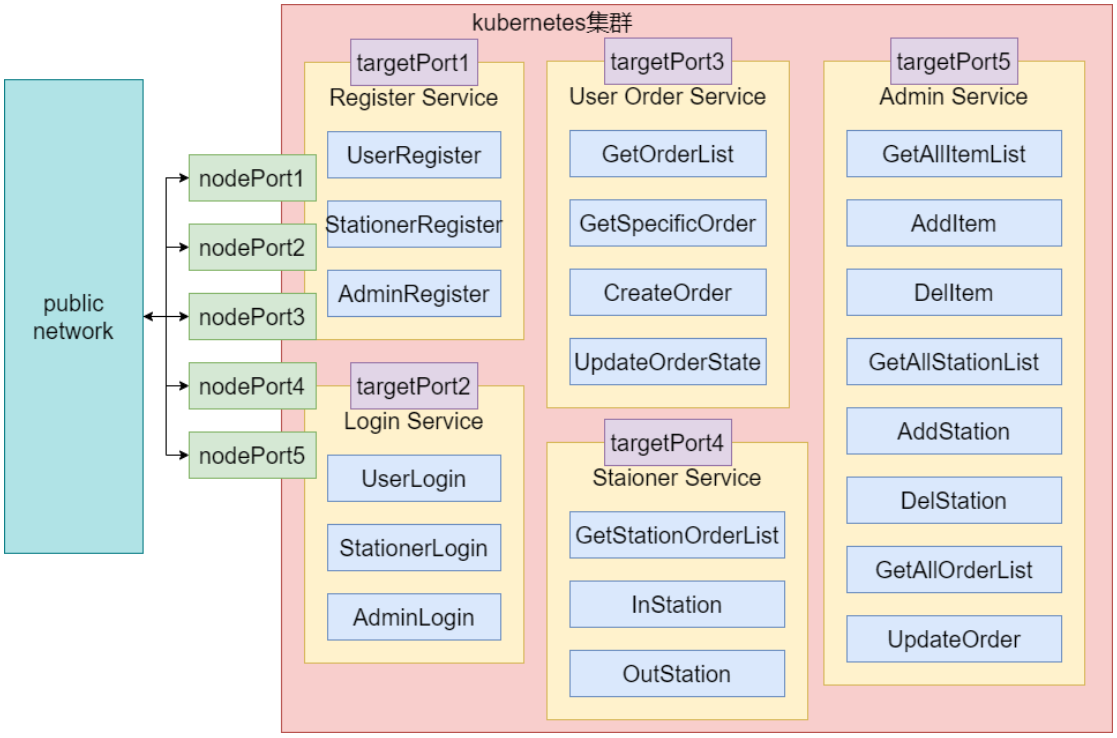

图 3.10 Server 端部署在 Kubernetes 集群中的微服务分解图

通过对软件工程方法设计阶段的需求用例进行逻辑划分，将以上用例分别部署在 5 个微服务中，具体划分方式如图 3.10 所示。对于负载较大的服务部署多个集群，以确保核心服务的高可用性。为资源占比较高的服务分配更多的计算资源，将计算密集型和 I/O 密集型服务分开，并在具体的资源分配和节点调度上有侧重的进行倾斜。

3.2.2 Client 端视图切换设计

Client 端视窗界面使用 QT 制作，针对不同的用户身份和用例场景制作了特定的 GUI（Graphical User Interface，图形用户界面），用户通过与 GUI 进行交互，在不同的视窗界面中完成用例在用户端的对应操作，Client 端的视窗切换界面的状态转移如图 3.11 所示。

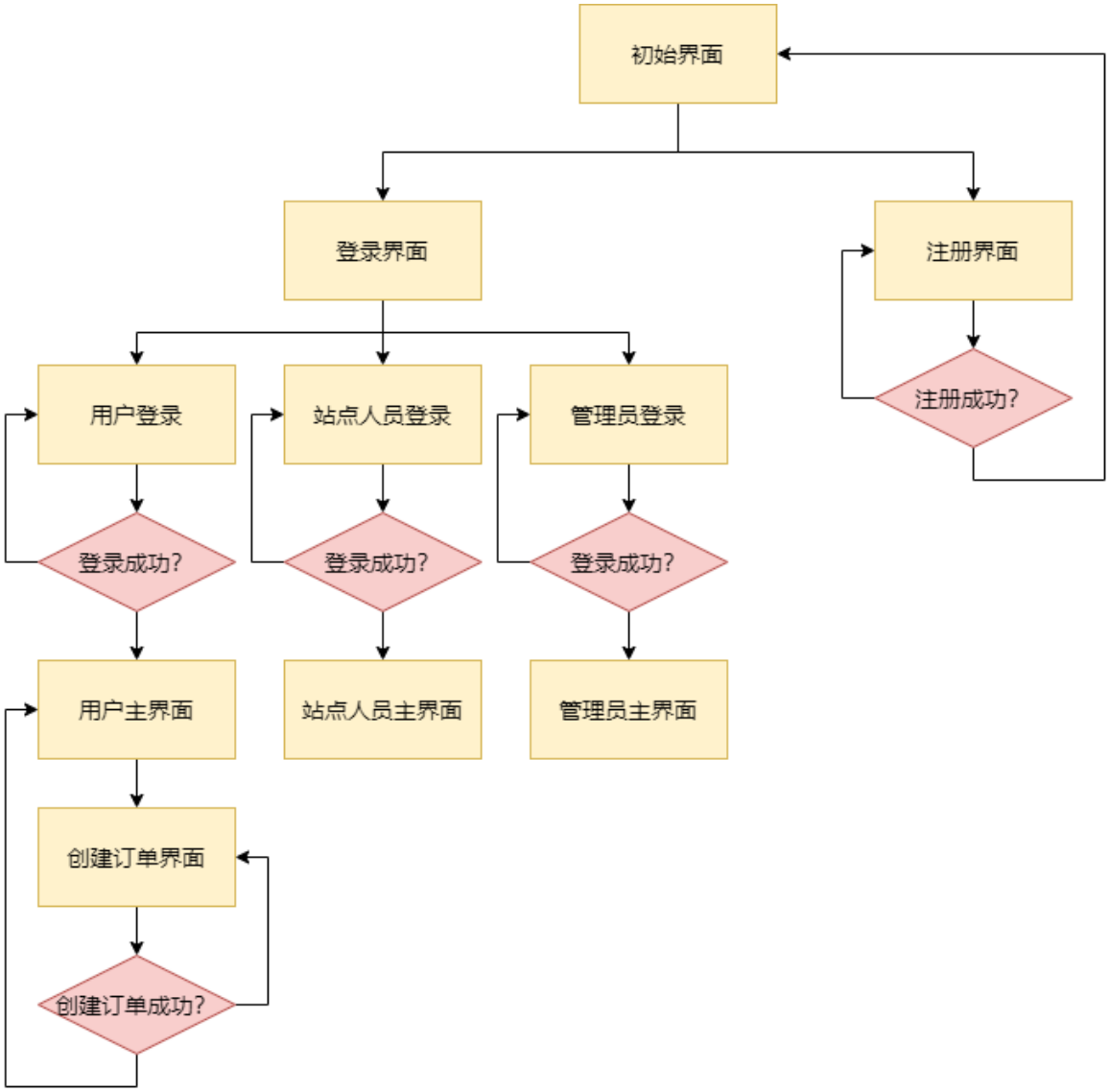

图 3.11 Client 端的视窗切换界面的状态转移图

3.2.3 Protocol Buffers 通信协议设计

用户需要使用客户端与部署在 Kubernetes 集群中的服务端进行通信，针对这个过程，需要设计专门的通信协议。本选题采用了 2.6.2 一节中所介绍的 Protocol Buffers 来进行 C-S 通信协议的设计，并配合 gRPC 分布式框架在双端进行部署，具体的协议格式及设计方案如下：

（1）MyRegister 注册服务：用于用户注册服务的通信，包括普通用户注册、站点工作人员注册和系统管理员注册，具体格式见表 3.10。

表 3.10 service MyRegister 列表

RPC 服务名	Request 请求	Reply 返回	描述
UserRegister	UserRegisterRequest	UserRegisterReply	用户注册

（2）MyLogin 登录服务：用于用户登录服务的通信，包括普通用户登录、站点工作人员登录和系统管理员登录，具体格式见表 3.11。

表 3.11 service MyLogin 列表

RPC 服务名	Request 请求	Reply 返回	描述
UserLogin	UserLoginRequest	UserLoginReply	用户登录

（3）MyUser 用户服务：用于普通用户相关服务的通信，包括获取订单列表、获取订单信息、创建订单（寄件）、收件等，具体格式见表 3.12。

表 3.12 service MyUser 列表

RPC 服务名	Request 请求	Reply 返回	描述
GetOrderList	UserId	OrderList	用户登录
GetSpecificOrder	OrderId	SpecificOrder	获取订单详细信息
CreateOrder	OrderDetail	BaseReply	寄件
UpdateOrderState	UpdateOrderStateRequest	BaseReply	确认收件

（4）MyStationer 站点工作人员服务：用于站点工作人员相关服务的通信，包括获取站点订单列表、订单入库、订单出库等，具体格式见表 3.13。

表 3.13 service MyStationer 列表

RPC 服务名	Request 请求	Reply 返回	描述
GetStationOrderList	StationId	SaveOrderList	获取仓库中订单列表
GetStationerInfo	StationerId	StationerInfo	获取站点工作人员信息
InStation	InStationRequest	BaseReply2	入库
OutStation	OutStationRequest	BaseReply2	出库

（5）MyAdmin 管理员服务：用于管理员服务的通信，包括货物管理、站点管理、订单管理等，具体格式见表 3.14。

表 3.14 service Admin 列表

RPC 服务名	Request 请求	Reply 返回	描述
GetAllItemList	AdminId	AllItemList	获取所有货物列表
GetAllStationList	AdminId	AllStationList	获取所有站点列表
GetAllOrderList	AdminId	AllOrderList	获取所有订单列表

AddItem	ItemInfo	BaseReply3	添加货物
DelItem	ItemInfo	BaseReply3	删除货物
AddStation	StationInfo	BaseReply3	添加站点
DelStation	StationInfo	BaseReply3	删除站点
UpdateOrder	OrderInfo	BaseReply3	更新订单信息

装 订 线

4 物流管理系统的系统搭建与集群部署

4.1 本地编译与服务端开发流程

4.1.1 本地环境与软件清单

（1）本地编译环境

本地编译环境为在 Windows PC 机上运行的 Linux 虚拟机，具体软件及操作系统版本如下：

本地宿主机环境：Windows 10 21H2 64 位家庭和学生版

虚拟机软件：VMware workstation 15.5 pro

虚拟机镜像版本：ubuntu-20.04.5-desktop-amd64.iso

虚拟机操作系统版本：Linux Ubuntu 20.04.5 LTS

虚拟机配置：2 核处理器、4G 内存、40G 磁盘空间

（2）安装软件清单

在本地获取并搭建对应的编译环境后，需要在 Linux 虚拟机中获取并安装一系列的软件，具体用到的软件清单及版本如表 4.1 所示。

表 4.1 本地编译环境的安装软件清单

软件名	软件版本	描述
Docker	20.10.22	镜像制作及管理工具
CMake	3.19.6	跨平台 C++编译工具
gRPC	1.12.0	Google 公司开发的开源 RPC 框架
Protocol Buffers	3.6.1	Google 公司开发的开源数据序列化协议
MySQL 组件包	8.0.32-0ubuntu0.20.04.2 for Linux on x86_64 ((Ubuntu))	数据库组件，编译过程中使用
Google Logging	0.7.0	Google 公司开发的开源日志库
perf	5.15.87	Linux 原生提供的性能分析工具，对 CPU 上的堆栈按照一定频率进行采样
Flame Graphs	/	基于 Perf 统计数据生成可视化堆栈嵌套火焰图
google/benchmark	1.7.1	Google 公司基于 googletest 框架开发的开源测试工具，统计程序运行的 CPU 时间

4.1.2 本地编译机编译环境配置

（1）初始阶段环境配置

使用 Ubuntu 自带的 apt 方法下载 perf、Docker、CMake 和 MySQL 组件包，以上文软件清

单中的版本为准。

（2）高性能开源分布式框架 gRPC 及序列化协议 Protocol Buffers 的安装与配置

从官方 GitHub 上获取源代码，克隆至本地并签出上文软件清单中的对应版本。之后进入 gRPC 源代码目录按顺序执行相关命令，安装 gRPC 及 Protocol Buffers 的依赖库。

由于安装的 gRPC 版本较低，做如下代码修改以使得当前版本的 gRPC 与 Protocol Buffers 能够兼容运行。

进入 gRPC 源代码下的 third_party/protobuf 路径，执行附录所示安装脚本，使用 CMake 从源代码开始编译并安装上述软件。如无报错提示，gRPC 框架和 Protocol Buffers 协议安装成功。

在以上分布式框架和通信协议安装完成后，进行测试，运行 example 路径下的测试程序，若能够正常运行并通信，则说明上述程序安装完成，系统可以正常运行。

（3）日志库 Google Logging（GLOG）的安装

从官方 GitHub 上获取源代码，克隆至本地并签出上文软件清单中的对应版本，具体脚本见代码附录。

执行相关代码，即可安装完成 GLOG 日志系统并使用，使用时需要在编译过程中加上编译选项：-lglog。而在对应的 C++源代码中，使用 GLOG 日志库，需要引用头文件"glog/logging.h"和"glog/raw_logging.h"。

按照以下格式，在源代码初始化阶段进行配置。需要注意的是，FLAGS 在 InitGoogleLogging 函数之前设定，在之后设定的无效。日志信息会根据源程序文件名和开始时间等在日志文件命名中反应出来，同时在日志文件头部进行记录。不用进行额外设置，使用初始设置即可。

（4）单元测试软件 google/benchmark 的安装与使用

从官方 GitHub 上获取源代码，克隆至本地并签出上文软件清单中的对应版本，具体脚本如代码附件所示。按照顺序执行相关命令，从源代码开始编译并安装 google/benchmark。

4.1.3 Server 端软件代码的编写

（1）Server 端总体架构

此处对 Server 端的源代码文件进行了相关的解析与介绍，Server 端相关代码按照功能划分，本节给出了顶层目录下的功能模块列表。

① 微服务程序模块

register：注册微服务源代码。

login：登录微服务源代码。

admin：管理员微服务源代码。

user：普通用户微服务源代码。

stationer：站点人员微服务源代码。

本系统的微服务主要分为 5 个部分分别部署，上面列出的为主目录下的子文件夹，在该路径下编写相关的业务逻辑代码并进行编译。以上 5 个微服务子文件夹下有各自的 Makefile 和 protos 文件夹，protos 文件夹存放本路径下所使用的 Protocol Buffers 协议文件，Makefile 负责主

要有系统版本自检、预处理构建、正式构建和删除模块组成。

② 测试程序模块

init_sql：自动化构建测试数据的脚本，在测试时将生成的 init_create.sql 和 init_insert.sql 文件导入 MySQL 数据库中执行。

benchmark_test：基于以上模拟的测试数据，进行大量数据单元测试。在该文件夹中同样有如上文所示的 Makefile 和 protos 文件夹，功能类型同上。

③ 通用模块

Makefile：顶层 Makefile，进入所有需要构建的文件夹，调用对应子文件夹下的 Makefile 进行构建和删除操作。

bin：存放编译完成的可执行文件，在顶层编译完成后自动将可执行文件复制到该目录下。

common：存放系统共用编译源文件（.cpp/.cc 后缀文件）。

include：存放系统共用编译头文件（.h 后缀文件）。

④ 镜像制作模块

lib：存放制作微服务镜像时需要导入的库文件。

kube_cluster：存放需要部署在 Kubernetes 集群中的.yaml 配置文件。

Dockerfile_*：共有 5 个类似格式的文件，分别对应文件名相关的微服务，通过执行该文件制作微服务镜像。

（2）Protocol Buffers 通信协议代码的编写

根据 Protocol Buffers 通信协议的设计，编写.proto 文件，通过自动化代码生成技术生成对应的协议，基本.proto 文件的格式如 User 服务所示。协议代码定义了命名空间（package）为 registerspace，之后定义了 UserRegisterRequest 和 UserRegisterReply 两个消息种类，在其中设置不同字段的数据类型。最后在名为 Register 的服务中注册一个 UserRegister 的 RPC 远程过程调用。

（3）系统主题代码的编写

在本项目中，引入了 MySQL 操作库、GLOG 日志库、gRPC 分布式框架库与 Protocol Buffers 通信接口等库，使用 C++作为主要的编程语言编写对应的系统代码。一个服务器最顶层的完整运行代码包括注册 UserOrder 服务，使用 builder 构建服务器与实例化，MysqlStart 启动 MySQL 连接、GLOG 打印工作日志、Wait 函数开始监听外部请求等功能。

封装一组 MySQL 的增删查改操作，使用 C++中的运算符重载技术，使得系统对外暴露出的 MySQL 操作接口能够被物流管理系统以较为统一且方便的方式进行调用，处理了一组封装后的 MySQL 的 C++操作，具体底层实现依赖于 C++的 MySQL Client 库。

（4）测试代码的编写

本项目使用 C++编写脚本生成测试文件和填充数据，并引入 google/benchmark 库进行单元测试代码的编写，使用条件编译的技术控制不同阶段的测试选项，针对不同的数据量级使用脚本构造不同的测试数据。使用函数调用的技术，导入需要测试的对应函数项。在 Range 的量级范围内，对 RangeMultiplier 中的递增倍数逐步增加测试的数据量级。

（5）系统的一键式自动化构建

基于以上各项系统源代码，在系统中编写自动化构建代码，一键式构建系统，编译链接生成可执行文件，使用相关的 Makefile 文件，执行 make 命令即可开始系统构建。首先启动系统自检 system-check，检查构建之前的系统环境和软件版本是否符合要求。.PRECIOUS.前缀内容代表构建前预处理，使用 protoc 工具生成 gRPC 和 Protocol Buffers 的 C++接口代码。之后使用 GCC 工具编译并链接源代码文件，将所需的库文件进行链接，形成可执行文件。执行 make clean 命令时，对编译和链接过程中生成的中间文件进行删除。

4.1.4 Server 端应用镜像的制作

云应用的部署前置形态是镜像，使用 Dockerfile 对应用镜像进行制作，Dockerfile 是一种镜像制作专用的脚本文件，关于镜像的构建过程，是一种千层饼式的构建，每执行一次命令都是一次镜像的堆叠，在原有的镜像上添加一层。本选题在本地编译环境进行手动构建，将构建完成的镜像上传到中转的内网仓库中[31]。

由于在编译过程中采用了动态编译技术，因此在制作容器镜像时，将可执行文件的依赖库拷贝到容器的镜像中，使得在容器中运行的可执行文件可直接调用依赖，而不需要在镜像中重新编译对应的库文件，降低了镜像的复杂程度。

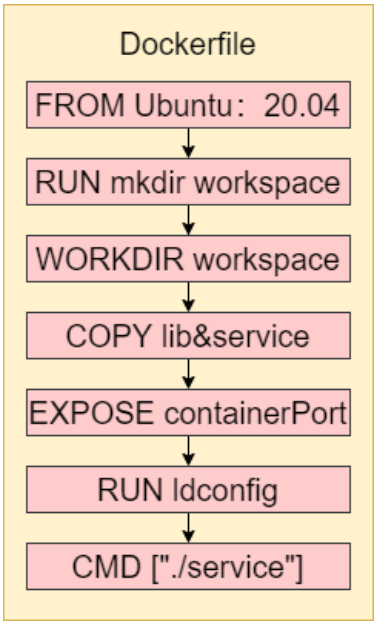

图 4.1 使用 Dockerfile 构建镜像的标准流程图

根据图 4.1 中流程所示，在制作镜像的第一步，使用 FROM 命令确定基础镜像。之后使用 WORKDIR 设定工作区并将对应的依赖库和可执行文件拷贝到对应位置。之后使用 EXPOSE 命令设定容器服务对外保留的端口（对应到具体部署在集群中时为服务的 targetPort）。之后使用 ldconfig 命令重新加载依赖库，最后在容器中运行位于工作区的可执行文件，具体 Dockerfile 编写示例如源代码中所示。

在本地环境下制作完成镜像后，将制作好的镜像上传到云平台上的私有容器镜像仓库中，

在集群部署完成后即可使用该镜像在集群中部署服务[32-33]。

4.2 在云计算平台上进行集群部署

4.2.1 云计算平台服务清单

本选题要求使用云计算平台，使用阿里云计算平台。作为国内最大的云服务提供商，阿里云提供了一系列相关可供使用，在本文中具体使用内容及配置具体如表 4.2 所示（表中服务均需要在阿里云计算平台中购买）。

表 4.2 云计算平台的服务清单

云平台服务名称	数量	配置
ECS 云服务器	3	2 核 CPU、4G 内存、40G 系统盘
OSS 对象存储	1	20GB+2GB 下行流量
ACR 容器镜像服务	1	3 个命名空间+300 个镜像位
云原生数据库 PolarDB Serverless MySQL	1	MySQL 8.0 兼容版，与 OSS 共用存储空间和流量

4.2.2 在阿里云计算平台上搭建 Kubernetes 集群与部署微服务

（1）云服务器配置列表

在同一地域和内网网段下创建 3 台 ECS 云服务器，形成 1 台 Master+2 台 Slave 的集群配置。在没有购买弹性公网 IP 服务的情况下，每次开机会分配一个随机的公网 IP，需要购买固定的公网 IP 来为 Client 端提供固定的访问接口。

首先按照 4.2.1 中清单在阿里云计算平台上购买并配置对应的服务，其中 ECS 云服务器的详细信息如表 4.3 所示。

表 4.3 购买的云服务器配置列表

节点主机名	内网 IP	公网 IP	身份
Node1	172.29.1.1	随机分配（不固定）	Master
Node2	172.29.1.2	随机分配（不固定）	Slave
Node3	172.29.1.3	随机分配（不固定）	Slave

（2）Kubernetes 集群的搭建

禁用 ufw 防火墙：ufw disable，使得集群中仅由 kube-proxy 通过 iptables 的方式进行转发，不受到预料之外的防火墙的限制。

禁用 swap 交换分区：swapoff -a && sed -i '/swap/d' /etc/fstab，原因是硬盘的读写速度与内存相比相差一个甚至两个数量级，使用 swap 交换分区会极大程度上的降低集群的调度性能，增大了集群崩溃的可能性。

配置集群基础设施：使用命令安装并配置开源软件 Docker 和 Kubernetes，并从公有仓库中

拉取镜像列表。

在 Master 节点上安装网络插件：本选题选择 flannel 作为 Kubernetes 集群的网络插件，需要在 kubeadm 初始化集群的时候设定好 pod-network-cidr 字段与 flannel 的网段一致，设置集群内部 Pod 节点的通信网段与网络插件相符。

将节点加入集群：将其他的 Slave 节点逐个加入集群中，尝试在集群中进行通信，如果 Node 节点状态正常并且节点之间能够相互通信，说明内网集群配置成功，查看结果应为所有 Node 节点均为 Ready 状态。

放开安全组策略：对外开放所需端口，使得 Client 端能够通过公网访问部署在集群中的服务，具体方案如图 4.2 所示，开放全部端口便于后续阶段的微服务部署，但在某种程度上也降低了系统的安全性。

图 4.2 安全组策略开放方案

（3）在 Kubernetes 集群中通过镜像和配置文件部署服务

在 Kubernetes 集群中通过镜像和配置文件部署服务的全流程如图 4.3 所示，按照对应流程图进行服务的部署。

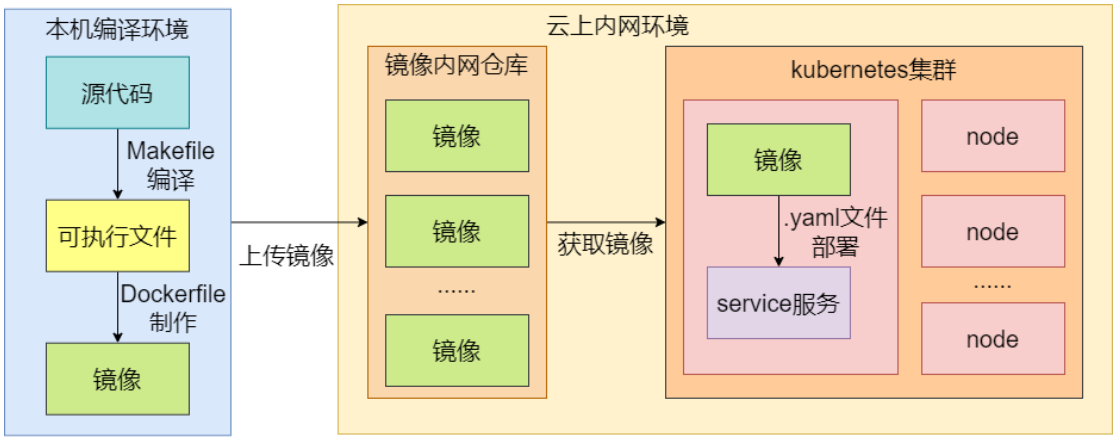

图 4.3 镜像制作及部署的全流程

在本地编译环境中制作好的应用镜像已经被上传到内网仓中，此处编写对应的.yaml 配置文件来将应用镜像部署在 Kubernetes 集群中，该配置文件大致分为两个部分：Deployment 部分在集群中部署一组 Pod，而 Service 完成对一组 Pod 的统一管理。需要注意，在部署 Service 的时候，需要选择 spec 类型为 NodePort，并设置对集群外部暴露的端口号。

在本例中，共在 Kubernetes 集群部署 5 个服务，按照从 OSS 中获取对应服务的.yaml 文件，在集群中使用 kubectl 工具进行部署。所有服务在集群中部署完成后，在 Master 节点上查看此时运行在节点中的 Pod 清单，预期结果应为运行正常。

至此，完成在阿里云计算平台上的 Kubernetes 集群搭建和容器化微服务部署，可以通过 Client 端访问部署在云上的物流管理系统。

4.2.3 在阿里云计算平台上搭建附属服务

（1）容器镜像仓库 ACR 的使用

首先需要在控制台对相应的云服务进行购买，在使用容器镜像服务作为镜像中转的内网仓时，需要做一些特定设置。最初需要设置命名空间，在对应命名空间中按照需求创建应用所需的仓库，镜像在仓库中存储的方式如图 4.4 所示。

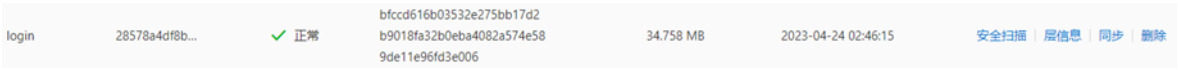

图 4.4 存储在阿里云平台上的应用镜像

此处给出了使用命令行方式进行远程操作的方法，可运行在 Linux、Windows 及其他命令行系统，基本操作节选介绍如下，可以按照如下格式连接远程镜像仓库并进行相关操作。

远程登录镜像仓库：sudo docker login --username=kazusadaisuki registry.cn-shanghai.aliyuncs.com

拉取镜像：docker pull registry.cn-hangzhou.aliyuncs.com/xpf_images/my_log_system:[镜像版本号]

推送镜像：docker push registry.cn-hangzhou.aliyuncs.com/xpf_images/my_log_system:[镜像版本号]

（2）云原生数据库 PolarDB Serverless MySQL 的部署及应用

在阿里云计算平台上购买 PolarDB Serverless MySQL 服务后，需要对其做一些基本设置。设置数据库的访问白名单，并开放公网和内网的集群地址。除此以外，还需要对数据库实例做某些特殊设置以便于同时在内网和公网环境下能够正常使用。

远程连接需要给用户赋权限，为了简单起见，此处直接给 root 用户开超级权限，针对本例的 MySQL 8.0 对应版本执行 ALTER USER 特有命令以创建高权限用户并赋予权限。

之后，需要修改对应的 MySQL 配置文件：/etc/mysql/mysql.conf.d/mysqld.cnf。在[mysqld]命名域下修改 bind-address = 0.0.0.0，查看此时 3306 端口的监听情况，发现此时监听了回环地址(127.0.0.1)。此时使用 service mysql restart 命令重启服务，发现生效，可从外部访问本机

MySQL。再次查看此时 3306 端口的监听情况，全部修改完成，此时可以连接并开始使用。

（3）对象存储 OSS 的使用

在阿里云计算平台上购买 OSS 服务后，首先需要下载并配置 OSSUtil 命令行管理工具。该工具是对云端的 OSS 对象存储进行远程操作的命令行工具。在按照相关教程配置完成后，可以开始使用 OSS，在 Linux 编译环境、Windows 开发环境、云端运行环境等不同环境进行文件对象的分发迁移。此外还可以在 Web 端使用图形化操作界面手动操作，如图 4.5 所示。

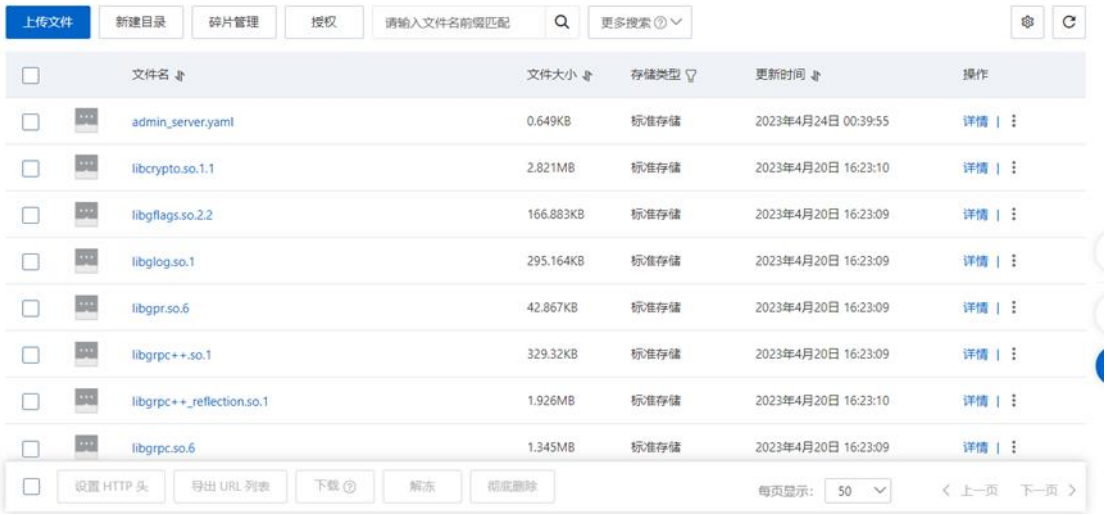

图 4.5 对象存储 OSS 的图形操作界面

4.3 Windows 下客户端的开发与使用

4.3.1 本地环境与软件清单

本地操作系统环境：Windows 10 21H2 64 位家庭和学生版。

在本地获取并搭建对应的编译环境后，需要在系统中获取并安装一系列的软件，在本例中使用 vcpkg 包管理器，对开源软件进行统一下载与安装。vcpkg 是由微软开发的 Windows 平台下的 C++包管理器，使用 vcpkg 可以有效的进行一站式的环境管理和配置。对于不同版本的依赖（不同版本号、debug/release、x86/x64/…），vcpkg 可以自动管理依赖版本，从而仅需要通过命令行的方式进行开源软件的下载与安装。具体用到的软件清单及版本如表 4.4 所示。

表 4.4 Windows 客户端编译及运行环境的安装软件清单

软件名	软件版本	描述
vcpkg	2023-03-14	微软公司开发的 Windows 平台下的 C++包管理器
CMake	vcpkg-cmake:x64-windows 2022-12-22	跨平台 C++编译工具
gRPC	x64-windows 1.51.1	Google 公司开发的开源 RPC 框架
Protocol Buffers	x64-windows 3.21.12	Google 公司开发的开源数据序列化协议

Windows SDK	8.1	Windows 软件开发工具包
Microsoft Visual Studio Community 2019	16.8.0	集成开发环境 IDE
Qt	5.13.1 (MSVC 2017 64-bit)	UI 界面开发
Qt Visual Studio Tools	version 2.4.3	VS2019 兼容 Qt 插件

在本选题中，可以使用如下命令来在系统中下载并安装开发依赖文件，仅需安装 Protocol Buffers，vcpkg 会自动编译并安装该软件所需的依赖库。

vcpkg install protobuf protobuf:x64-windows

vcpkg integrate install

如安装正常，使用 vcpkg list 命令查看列出的所有安装包列表，若显示如表 4.4 所示，则说明 gRPC 和 Protocol Buffers 及对应的依赖环境安装完成。

4.3.2 客户端软件代码的编写

（1）Protocol Buffers 通信协议代码的编写

客户端使用的协议与服务端完全相同，接口生成方法也类似，此处通过在命令行中执行代码成成文件的方式来进行相关的代码文件生成。

（2）用户界面的设计与使用

在本选题中，客户端用户界面的设计采用了 QT 客户端的方式，对用户界面进行可视化设计。采用 QT Designer 作为设计用户界面的核心工具，该工具使用拖动和手动设定的方式，生成类似于 XML（文本标记语言）的界面标记语言 QML。在编译时，通过编译器生成对应的 UI 源代码 C++文件，为业务逻辑提供接口，使用 QT 特有的信号和槽机制将用户界面与内部程序相连接，具体对应。

（3）客户端代码的编写

在本选题中，在编写 Windows 客户端的过程中也使用了 gRPC 的 Client 端框架结构，通过使用该框架特有的 Stub 结构，起到了类似于 HTTP 请求中的 Client 的作用。而在前端与用户交互的部分，使用了 QT 特有的信号和槽机制，槽函数通过监听在槽中注册的信号做出对应的操作，作为用户与可视化界面进行交互并将结果反馈到客户端成程序内部的机制。客户端 RPC 服务远程调用的代码架构可见代码。

4.3.3 客户端软件的编译及打包发布

与服务端的开发相比，客户端的编译相对简单。在 VS2019 的集成编译环境中，只需要将项目所需要的源代码文件和依赖库导入对应的集成开发环境，之后进行项目生成即可。

在完成对 Windows 客户端程序的代码编写以及编译后，需要将应用程序进行打包发布，以便于在不具有相关开发环境的电脑下能够运行该应用程序。在 Windows 命令行下打开本例所使用的 MSVC2017 64bit 编译器，执行相关打包命令，通过 windeployqt 工具即可将对应位置的.exe 可执行文件进行打包发布，运行在任何 64 位的 Windows 系统上。

4.4 物流管理系统的运行及使用说明

根据上文软件系统的设计，完成对系统 Client 端的开发之后，即可使用客户端与部署在云上的服务端连接，至此，本选题所述物流管理系统开发完成，具体界面及连通性展示如图 4.6-4.11 所示。在 4.1 一节中发布的容器化微服务服务端应用已经按照 4.2 节流程正确运行在云计算平台的 Kubernetes 集群上的前提下，软件系统已经可以正式投入使用，本节主要是面向普通用户的软件系统使用说明书。

（1）注册功能，包括普通用户注册、站点工作人员注册、管理员注册，如图 4.6 所示。

注册：在主界面中选择需要注册的用户身份，点击按钮进入注册界面。若为注册普通用户，仅需填写用户名、密码、姓名三项内容；若为注册站点人员，需填写用户名、密码、姓名、密钥、站点 id 五项内容，其中密钥在现实中进行分发；若为注册系统管理员，需填写用户名、密码、姓名、密钥四项内容，其中密钥在现实中进行分发。注册成功后，会自动跳转到初始界面，此时可以使用注册的账号登录；注册失败，服务端会返回对应的错误信息。

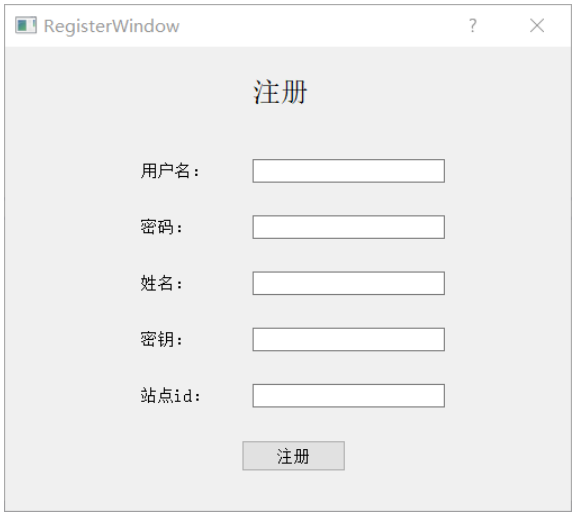

图 4.6 Windows 客户端注册界面

（2）登录功能，包括普通用户登录、站点工作人员登录、管理员登录，如图 4.7 所示。

登录：在主界面中选择需要登录的用户身份，点击按钮即可跳转进入登录界面。未注册用户默认登录失败，已注册用户填写用户名和密码两项内容，在服务端进行验证。登录成功后，跳转到对应的用户界面；若登录失败，返回对应的错误信息。

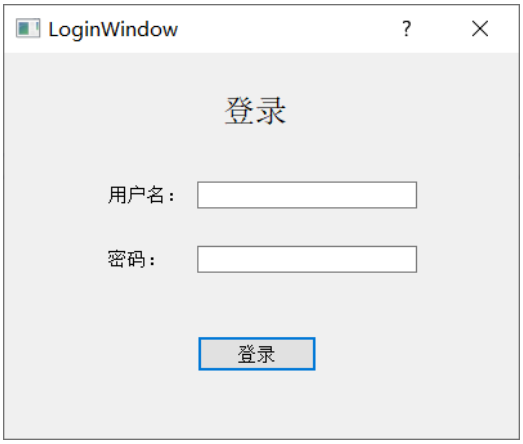

图 4.7 Windows 客户端登录界面

（3）用户界面，包括获取订单列表、获取订单详细信息、签收订单等，如图 4.8-4.9 所示。

获取订单列表：点击“刷新订单列表”按钮，在图 4.8 中的寄件订单列表和收件订单列表中分别显示作为寄件人和收件人的订单列表。

获取订单信息：单击图 4.8 所示寄件订单列表和收件订单列表中需要查看的订单信息列表，选中订单的详细信息在物流情况、货物列表、订单详细信息中显示。

签收货物：双击右侧货物列表中要签收的货物，更新选中订单的运送状态为“已收货”。

寄件（创建订单）：点击右上方寄件按钮进入图 4.9 所示操作界面。左侧栏目分别填写寄件人号码、收件人号码、寄件地址、收件地址和备注。右侧为货物列表，填写货物的 id、货物名和货物价格，选择需要寄的货物数量，点击“添加货物”按钮，将选中的货物加入下部的货物列表清单。双击货物列表清单，可以删除不需要的货物对应项。当所有货物添加完毕后，单击左下角“寄件”按钮，若寄件成功，返回图所示主界面；若寄件失败，服务端会返回对应的错误信息。

装 订 线

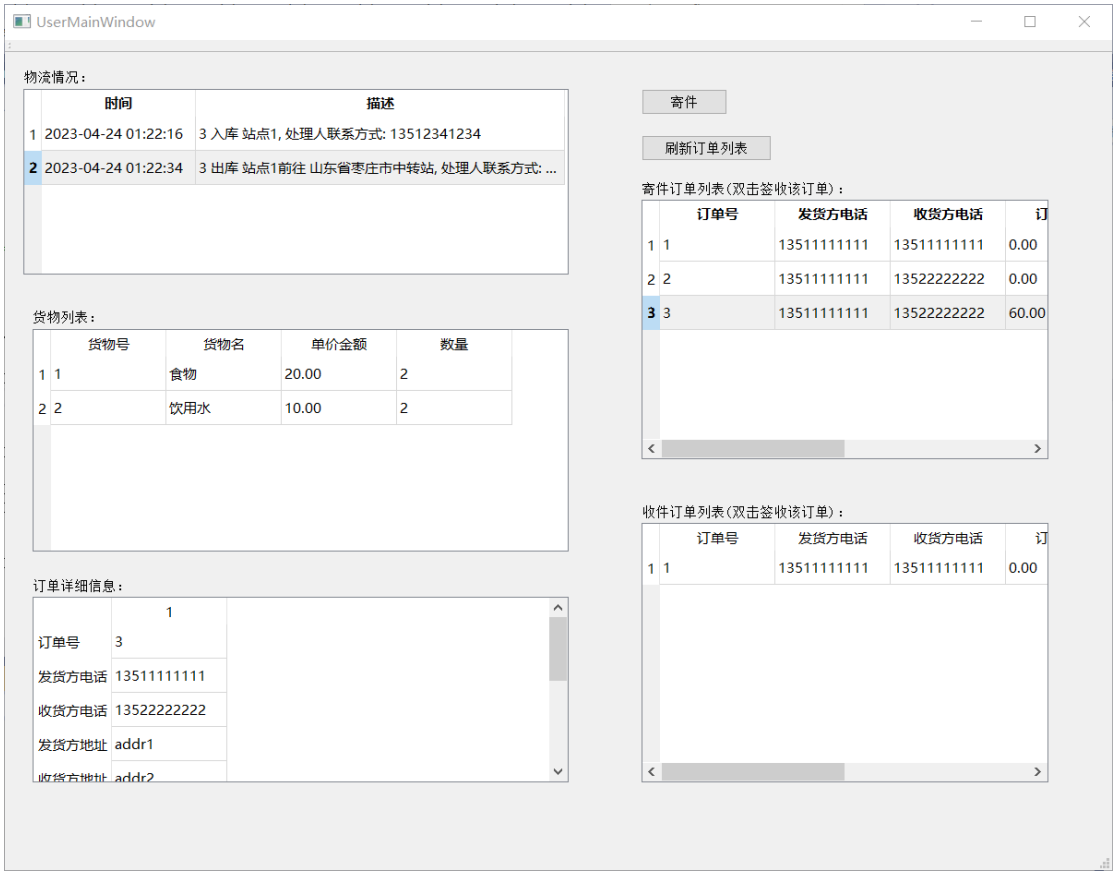

图 4.8 Windows 客户端用户主界面

图 4.9 Windows 客户端用户寄件界面

（4）站点工作人员界面，包括订单入库、订单出库、获取站点订单信息等，如图 4.10 所示。

获取站点订单信息：点击“刷新列表”按钮，在下方信息栏中显示对应的订单信息。

入库：在订单号输入栏填写入库的订单号，点击“入库”按钮，将对应订单存入该站点，并更新物流记录。

出库：填写右下角订单号、处理人手机号、出库前往地址三栏信息，之后点击“出库”按钮，将订单从该站点中送出，并更新物流记录。

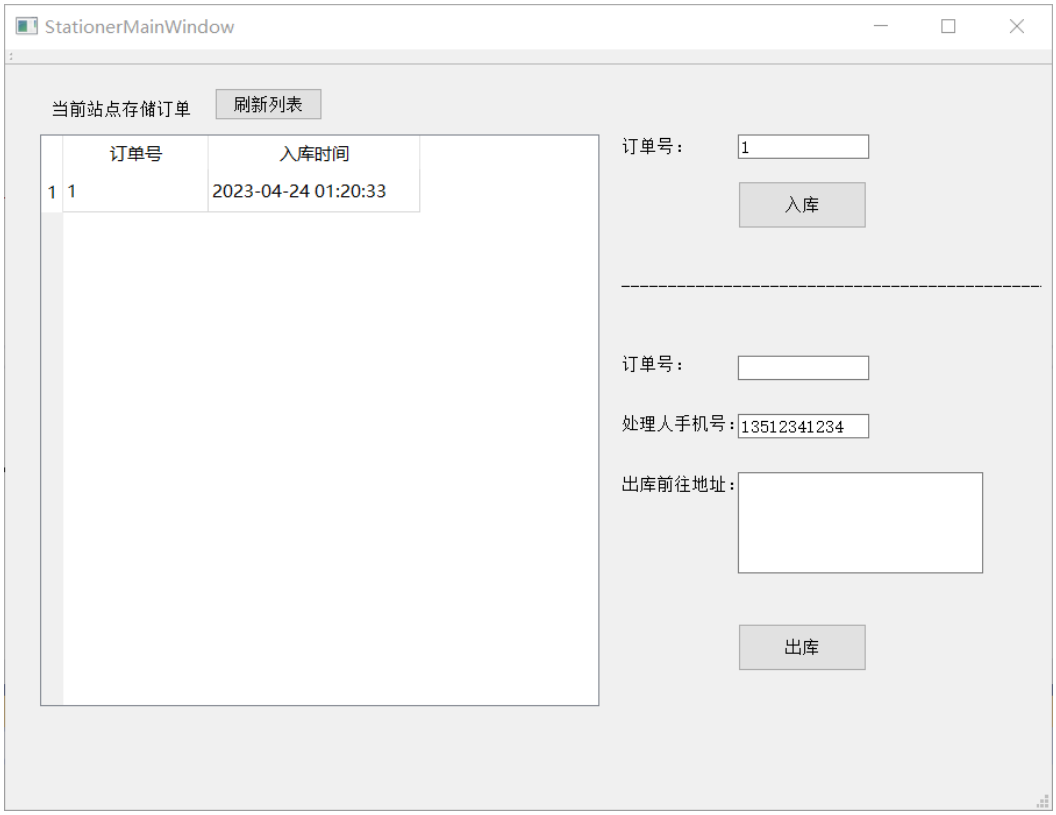

图 4.10 Windows 客户端站点工作人员界面

（5）系统管理员界面，包括货物管理、站点管理、订单管理等，如图 4.11 所示。

货物管理：在第一行填写货物相关信息，对货物进行添加、删除与更新。

站点管理：在第二行填写站点相关信息，对站点进行添加、删除与更新。

订单信息更新，在第三行填写更新后的订单信息，点击“更新订单”按钮对订单信息进行更新。

图 4.11 Windows 客户端系统管理员界面

4.5 物流管理系统的测试

4.5.1 Server 端微服务的 Google Logging 日志系统

日志系统有两个主要作用，针对开发过程中程序不符合预期的表现，通过查看日志的记录内容以进行 Debug，定位程序错误的出现位置和成因。另一个重要的功能是对关键操作进行记录，通过后台的日志数据来进行相关数据的统计、用户操作轨迹的记录和长期持有等。

根据应用的实际需要，本文搭建了一个完整的日志系统。在本选题中，使用 Google 公司开发的 Google Logging（GLOG）日志库作为日志的管理系统。该日志系统可以精准的记录日志运行时的时间、操作级别与具体信息等，粒度具体到具体到源代码文件的某一行，便于对记录信息进行精准定位。

与个人的文件记录相比，GLOG 日志系统直接封装好了一组操作，可以基于系统的严重性进行分级记录，分为 INFO（普通）、WARNING（警告）、ERROR（错误）、FATAL（致命）四个级别，此外，GLOG 日志系统还可以通过条件进行控制，且可以额外通过设置系统变量和自定义正则表达式修改日志格式。以上均为封装好的内容，不需要再进行额外的处理。日志系统的实例如图 4.12 和 4.13 所示。

```
Log file created at: 2023/04/12 11:00:46
Running on machine: ubuntu
Running duration (h:mm:ss): 0:00:00
Log line format: [IWEF]yyyymmdd hh:mm:ss.uuuuuu threadid file:line] msg
I20230412 11:00:46.108232 13955 mysql_common.cc:28] mysql_init successed
I20230412 11:00:46.111723 13955 mysql_common.cc:39] mysql_real_connect to localhost, database Logistics_System, user root successed
I20230412 11:00:46.117744 13955 admin_server.cc:255] Admin Server listening on 0.0.0.0:50055
I20230412 11:00:48.098058 13958 mysql_common.cc:169] mysql_query success
I20230412 11:00:48.098124 13958 admin_server.cc:66] GetAllItemList query succeed, admin_id_:13544444444
I20230412 11:00:50.963546 13986 mysql_common.cc:169] mysql_query success
I20230412 11:00:50.963593 13986 admin_server.cc:96] GetAllStationList query succeed, admin_id_:13544444444
I20230412 11:00:53.307695 13958 mysql_common.cc:169] mysql_query success
```

图 4.12 IFNO 级 GLOG 日志

```
Log file created at: 2023/04/04 12:41:35
Running on machine: ubuntu
Running duration (h:mm:ss): 0:00:00
Log line format: [IWEF]yyyymmdd hh:mm:ss.uuuuuu threadid file:line] msg
E20230404 12:41:35.928225 111715 mysql_common.cc:98] mysql_insert failed(You have an error in your SQL syntax; check the manual
E20230404 12:41:35.929080 111715 mysql_common.cc:99] inst:INSERT INTO orderitemlist_info VALUES (, 13, '1', 2);
```

图 4.13 ERROR 级 GLOG 日志

4.5.2 google/benchmark 单元测试

（1）测试工具和原理

本选题使用 C/C++作为主要开发语言。针对应用程序的绝对性能，使用 Google 公司开发的 google/benchmark 测试工具，该框架基于 googletest 框架，最低支持标准为 C++11，可以对程序中不同部分的代码进行模块化测试，统计不同函数模块的 CPU 时间，运行时间精确到纳秒级。

通过编写自动生成脚本，可以自动化的生成合法的测试数据。本例中通过编写 C++脚本自动化生成对应的 SQL 填充数据。本例中暂时使用两个 SQL 文件，第一个 create 文件创建对应的数据库和表结构，第二个 insert 文件创建生成的填充数据。在运行测试之前，将两个 SQL 文件在 MySQL 数据库中执行，从而将生成的测试数据填充到数据库中。

通过编写 google/benchmark 自动化测试脚本，完成对微服务系统核心功能模块的性能测试。尤其是针对高负载场景，通过不断调整存储数据和测试数据的量级，来测试集群中微服务在高负载下的性能表现。在本选题中，针对理论上工作负载较高、用户量较大的服务进行单元测试。

（2）单元测试环境

测试机 CPU 参数：

Run on (2 X 2592 MHz CPU s)

CPU Caches:

L1 Data 32 KiB (x2)

L1 Instruction 32 KiB (x2)

L2 Unified 256 KiB (x2)

L3 Unified 12288 KiB (x1)

Load Average: 0.73, 0.80, 0.89

（3）单元测试结果

GetOrderList 服务，测试结果如表 4.5 所示。

表 4.5 GetOrderList 服务 benchmark 单元测试结果

测试名	测试数量级	总时间/ns	CPU 时间/ns	测试轮数	平均 CPU 时间/ns
GetOrderList	10	7676175	1147713	644	114771.30
GetOrderList	100	73207912	11116059	63	111160.59
GetOrderList	1000	749699961	112163170	6	112163.17
GetOrderList	10000	7561785564	1135731425	1	113573.14

GetSpecificOrder 服务，测试结果如表 4.6 所示。

表 4.6 GetSpecificOrder 服务 benchmark 单元测试结果

测试名	测试数量级	总时间/ns	CPU 时间/ns	测试轮数	平均 CPU 时间/ns
GetSpecificOrder	10	6759862	969441	737	96944.10
GetSpecificOrder	100	66439151	9534888	75	95348.88
GetSpecificOrder	1000	649817656	93339614	7	93339.61
GetSpecificOrder	10000	6523920424	938471162	1	93847.12

CreateOrder 服务，测试结果如表 4.7 所示。

表 4.7 CreateOrder 服务 benchmark 单元测试结果

测试名	测试数量级	总时间/ns	CPU 时间/ns	测试轮数	平均 CPU 时间/ns
CreateOrder	10	12867879	919110	750	91911.00
CreateOrder	100	128402783	9509630	77	95096.30
CreateOrder	1000	1247433994	91321531	8	91321.53
CreateOrder	10000	12390000000	915378205	1	91537.82

GetStationerInfo 服务，测试结果如表 4.8 所示。

表 4.8 GetStationerInfo 服务 benchmark 单元测试结果

测试名	测试数量级	总时间/ns	CPU 时间/ns	测试轮数	平均 CPU 时间/ns
GetStationerInfo	10	3515992	797621	873	79762.10
GetStationerInfo	100	34903565	8101887	95	81018.87
GetStationerInfo	1000	349224844	80270466	8	80270.47
GetStationerInfo	10000	3515397699	805026912	1	80502.69

GetStationOrderList 服务，测试结果如表 4.9 所示。

表 4.9 GetStationOrderList 服务 benchmark 单元测试结果

测试名	测试数量级	总时间/ns	CPU 时间/ns	测试轮数	平均 CPU 时间/ns
GetStationOrderList	10	3647365	809417	839	80941.70
GetStationOrderList	100	36607364	8015564	84	80155.64
GetStationOrderList	1000	370216076	80265834	9	80265.83
GetStationOrderList	10000	3714798337	812837542	1	81283.75

（4）测试结论

由以上测试结果可得，在服务内横向对比，单条服务指令的平均执行时间在不同数据量级的测试环境下基本一致，没有随着数据量级的增加而大幅度降低服务性能。而在服务间纵向对比，指令的平均执行时间没有数量级上的差异，基本在 10^5 ns 这个数量级上，单个服务的执行能力约为每秒 1000 个请求。

4.5.3 使用 perf 命令和火焰图进行系统级性能测试

（1）perf 命令和火焰图原理介绍

针对微服务架构的性能监控，在云原生开发中对应用程序的相对性能进行测试，以寻找性能瓶颈并进行优化。本选题使用开源的应用程序性能监控软件 Flame Graphs，监控程序的堆栈嵌套调用以及每层堆栈函数的运行时间。

在本选题中，使用 Linux 系统原生提供的性能分析工具 perf 命令，可以在一段时间内对 CPU 上的嵌套堆栈进行一定频率的采样，返回 CPU 正在执行的函数名以及调用栈。基于以上 perf 的结果，对数据进行进一步分析，生成 SVG 格式的包含函数嵌套调用的图片，即火焰图。

可以从以下两个维度解读火焰图：首先，火焰图需要重点关注“尖峰”和“平顶”的分布，“尖峰”代表了程序在当前的函数中时间占比较少，说明该处函数组不是影响程序性能的关键因素。而“平顶”代表了程序在当前的函数层级中运行时间占比较大，往往意味着该处存在着占用时间的性能瓶颈或需要耗费大量 CPU 运行时间的重要服务。以下给出了本系统中两个对性能要求较高的高频请求服务测试实例。

（2）stationer_service 服务的监控火焰图及解读

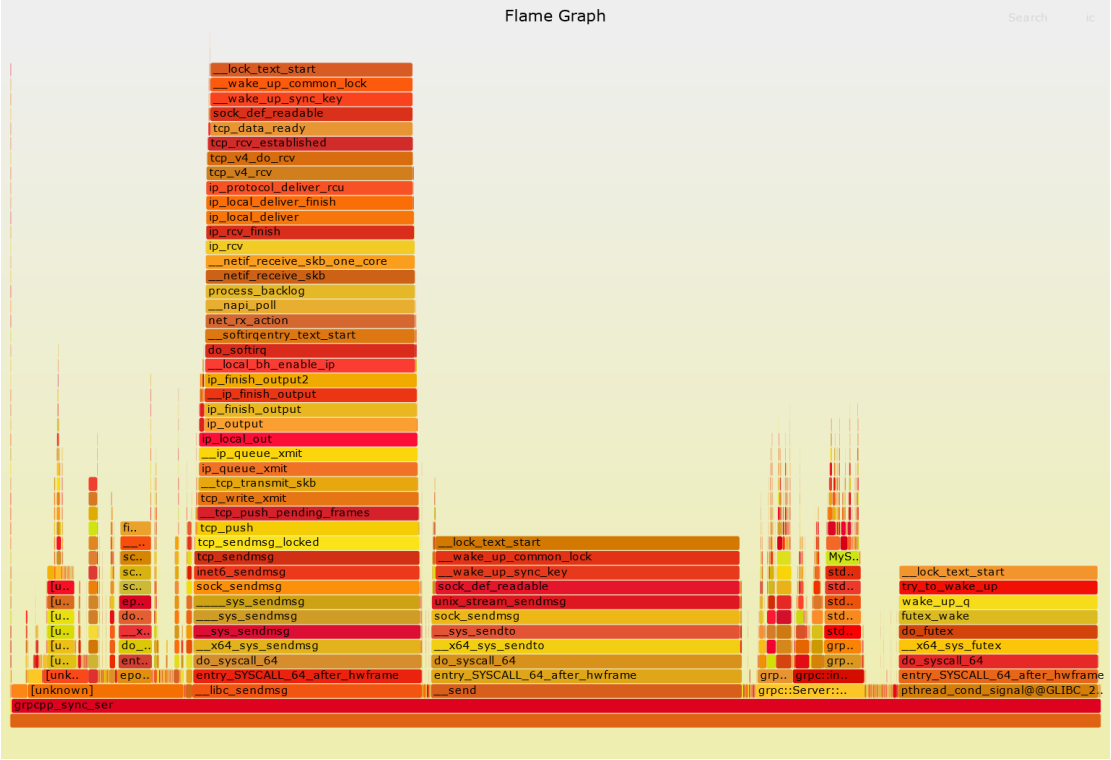

图 4.14 stationer_service 服务的火焰图

如图 4.14 所示，对于通信占比较高的 stationer 相关服务，该服务的特点是计算较简单，在业务逻辑中的占比较少，而与远端之间的通信服务占比较大。由上图火焰图中占比可见，函数 __lock_text_start 在最上层占比较大，是性能瓶颈。该函数是 Linux 内核中自旋锁的控制函数，在系统调用的较深层次被调用。该函数做的是设置一个 spinlock（自旋锁），是为了防止读写冲突，对通信内容进行上锁的机制。该图可以说明该进程大部分时间在 socket 通信的 send 阶段等锁，忙碌等待直到锁可用。

（3）user_service 服务的监控火焰图及解读

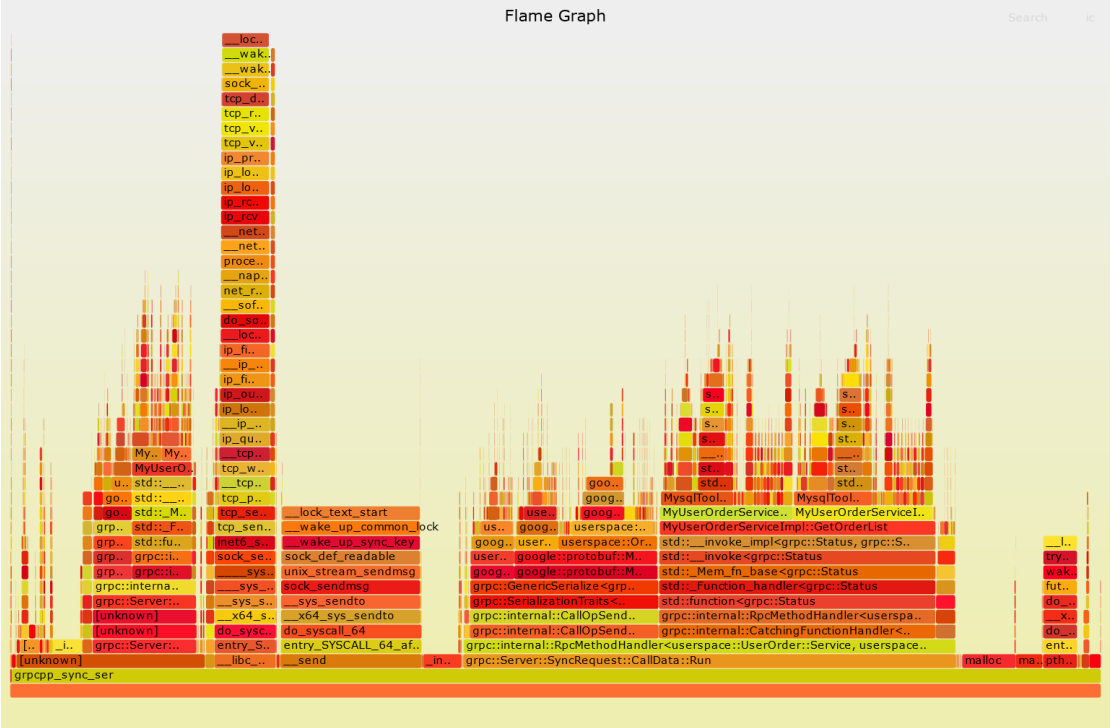

图 4.15 user_service 服务的火焰图

如图 4.15 所示，对于计算密集型的 user 相关服务，由上述火焰图可见，进程通信在该服务中资源消耗的占比明显减少，而实际的计算运行部分在服务中占据了更大的占比。这是因为 user 类型服务中完成了更多的本地计算操作，如计算数据，对象创建，文件读写等，而与 Client 端的 socket 通信占比相对较小。

4.5.4 集群高可用测试

针对 Kubernetes 集群，一个核心的评判机制就是高可用，即在服务因为负载过高或者因为自身原因崩溃的时候，能够自动拉起一个新的服务，以保证该服务始终对集群外部显示可用，这就是所谓的高可用。

本例中通过手动删除 Kubernetes 集群中正常运行 login 服务的 Pod，检测集群是否能够自动拉起 Pod 处于异常状态的服务。此处以 login 服务为例，此时该服务处于正常运行状态。

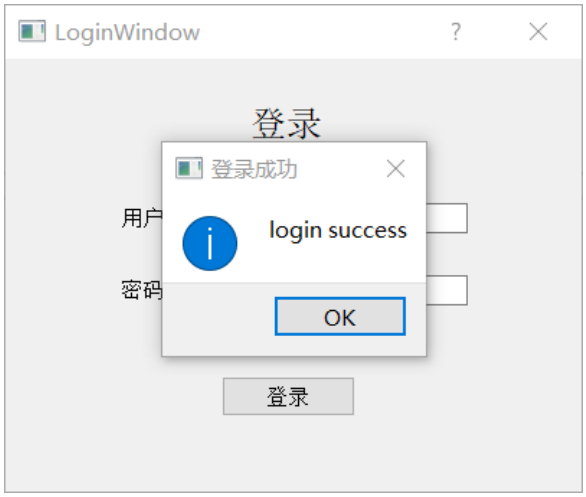

图 4.16 删除 login 服务后集群自动拉起，Client 端访问服务正常

在本例中，对集群高可用性的具体测试方法是：进入云服务器，查看部署在 Kubernetes 集群中的 Pod 状态，尝试手动将正在运行中的服务删除掉，再次查看 Kubernetes 集群中的 Pod 状态，发现原有 Pod 已经失效，Kubernetes 集群自动拉起了一个新的服务 Pod，重新拉起服务的时间<=5 秒，且 Client 端如图 4.16 所示能够正常访问，证明该 Kubernetes 具有高可用性。

5 结论和展望

5.1 结论

在本次毕业设计的过程中，以利用云计算平台设计 ERP 物流管理服务为主题，展开了对容器化微服务和工业软件 ERP 相关领域的研究。在研究过程中，首先总结了相关的研究背景和当前国内外的行业发展现状，在了解到该课题的重大意义之后，制定了切实可行的研究目标，并按照选题的要求，高质量的完成了各项工作。

在具体的工作内容中，理论研究部分，本文以 Kubernetes 集群和容器化微服务架构为主题的云原生技术和 ERP 工业软件与物流管理系统，以及物流管理系统和云计算的融合形态等内容。在设计阶段，完成了 Kubernetes 集群的架构设计以及集群在云平台上的部署形态，包括集群抽象层级，集群内外通信架构，集群上云部署形态等方面的设计，在阿里云计算平台上搭建了一个完整的 Kubernetes 集群，并将各项容器化微服务部署在集群中。

在软件设计和开发阶段，本文严格按照软件工程的指导思想，进行软件系统的建模和相关应用软件的开发。系统使用 C++作为主要的编程语言，使用 gRPC 分布式框架和 Protocol Buffers 通信协议，基于容器化微服务技术开发了一个物流管理系统，并通过对微服务的编排、封装、部署和发布，将该系统以微服务的形式部署在云计算平台上。在 Windows 平台上基于 QT GUI 系统开发了 Client 端，实现了从客户端到服务端到数据存储端的全链路打通。并使用 google/benchmark、火焰图等相关工具对系统的正确性、可靠性、性能等指标进行了量化测试。

综上所述，我认为自己完成了选题要求的各项研究实践工作，顺利完成了本次毕业设计（论文）的全部要求。

5.2 展望

至此本文已经按照要求，完成了论文选题“利用云计算平台设计 ERP 物流管理服务”所对应的各项工作，构建了一个 Kubernetes 集群以及基于容器化微服务架构的物流管理系统。

但针对本选题而言，在当前的研究和实践领域中仍有部分可以完善的点。在集群部署与配置方面，本文仅对 Kubernetes 集群的调度算法进行了调研与理论研究，可以考虑修改集群默认的节点调度算法，通过部署插件的方式定制调度算法并部署在集群中，以提高关键高负载服务的优先级，在高负载情况下通过改进调度算法提高集群的服务性能。

此外，在程序编写的过程中，本文采用了 gRPC 分布式远程过程调用框架，在服务端的处理方式是同步通信，在对该系统的架构进行进一步完善时，可以考虑将同步通信修改为异步方式，以应对更高数据量级的并发场景，达到使用异步通信对流量进行限流削峰的作用。

在现有的物流管理系统各项基本功能的基础上，在未来的进一步开发中可以考虑将系统接入官方个人信息接口和支付系统接口等，从而提高系统的安全性和可靠性，并进一步完善系统的各项功能，便于投入实际使用和提高用户体验。

装 订 线

参考文献

[1] 邓超. 企业信息化技术规范——ERP 标准开始实施[J]. 信息技术与标准化, 2003(10): 4-4.

[2] 吕志强, 孙超, 王艳蓉等. 基于 Kubernetes 的电网调度云应用容器管理系统的实现[J]. 工业控制计算机, 2022, 35(12): 130-132.

[3] 别要祥. 基于微服务的厨电企业内部供应链管理系统的设计与实现[D]. 浙江理工大学, 2022.

[4] Wan Y, Fu S. Application of microservice architecture in commodity ERP financial system[J]. International Journal of Computer Theory and Engineering, 2022, 14(4): 168-173

[5] 吕菊保. 基于云计算的快递物流管理系统的研究[J]. 中小企业管理与科技(下旬刊), 2021, (03): 174-175.

[6] 牛军. 物流管理与配送监控系统的设计与实现[D]. 电子科技大学, 2010.

[7] 齐小军, 李芳. 基于 ERP 的企业物流管理系统设计与实现[J]. 科技创新与应用, 2012(18): 47-48.

[8] Juntao B. Overview of cloud computing development[C]. Francis Academic Press, 2019: 6-6.

[9] Kakkar P. Business transformation with cloud ERP[J]. International Journal of Management IT and Engineering, 2021, 11(3): 27-31.

[10] 桂俊, 沈迎春. 基于微服务架构的企业 ERP 设计与应用[J]. 计算机系统应用, 2021, 30(08): 81-88.

[11] 刘炎火. 云原生技术的应用与研究[J]. 河南科技, 2021, 40(36): 6-9.

[12] 张墨涵, 王雪英, 沈学东等. 微服务架构技术与挑战[J]. 网络安全技术与应用, 2023, 266(02): 3-4.

[13] Chaitanya K. A systematic study of micro service architecture evolution and their deployment patterns[J]. International Journal of Computer Applications, 2018, 182(29): 18-24.

[14] 李翔. 在私有 Kubernetes 集群中实现服务的负载均衡[J]. 电子技术与软件工程, 2020, 184(14): 36-38.

[15] Haque I, Kumar R. Implementation of Kubernetes on different cloud platform[J]. Global Sci-Tech, 2020, 12(4): 181-187.

[16] Zhou N. Container orchestration on HPC systems through Kubernetes[J]. Journal of Cloud Computing, 2021, 10(1): 1-14

[17] Niño M. A microservice deployment guide[J]. Programming and Computer Software, 2022, 48(8): 632-645.

[18] Manish K A, Rajeswara R, Subrahmanyam K. Framework to deploy containers using Kubernetes and CI/CD pipeline[J]. International Journal of Advanced Computer Science and Applications, 2022, 13(4): 522-526

[19] 邢贞明, 李登辉, 潘博. 微服务架构与容器技术探析[J]. 金融科技时代, 2021, 306(02): 66-69.

[20] 董子奇, 刘淇, 高原等. 基于容器技术的微服务部署研究[J]. 信息技术与标准化, 2023, 457-

458(Z1): 93-98.
[21] 孙波. 浅谈微服务架构、Docker 和 Kubernetes[J]. 现代电视技术, 2022, 248(02): 100-103.
[22] 张悦. 基于 LSTM 的电商平台 Kubernetes 集群弹性扩缩的研究与应用[D]. 华东师范大学, 2022.
[23] 郭雷, 毛玲燕. 基于Kubernetes的企业级容器云平台设计与实践[J]. 信息技术与标准化, 2022, 453(09): 73-77.
[24] Prajval M. Load balancing using Docker and Kubernetes: a comparative study[J]. International Journal of Recent Technology and Engineering, 2020, 9(2): 782-792.
[25] 于泽川. 基于 Kubernetes 的资源调度策略研究与改进[D]. 浙江理工大学, 2022.
[26] 谭惟予. 基于大规模 kubernetes 集群多场景的任务调度优化方法研究[D]. 杭州电子科技大学, 2022.
[27] Vries S, Blaauw F, Andrikopoulos V. Cost-Profiling microservice applications using an APM stack[J]. Future Internet, 2023, 15(1): 37-37.
[28] Brendan G. The flame graph: this visualization of software execution is a new necessity for performance profiling and debugging[J]. Queue, 2016, 14(2): 91-110.
[29] 黄志成. 一种基于集群负载均衡的 Kubernetes 资源调度算法[J]. 电脑知识与技术, 2023, 19(05): 39-41.
[30] 钟建峰, 孟宇坤, 王石生等. 基于 gRPC 分布式数据通信的地铁线网指挥中心平台设计与实现[J]. 都市快轨交通, 2023, 36(02): 190-197.
[31] 陈伟, 叶宏杰, 周家宏, 魏峻. 基于领域知识的 Docker 镜像自动构建方法[J]. 大数据, 2021, 7(01): 64-75.
[32] 应毅, 刘亚军, 任凯. 基于容器云的分布式深度学习实验平台构建[J]. 实验技术与管理, 2022, 39(03): 147-152.
[33] Truyen E, Xie H, Joosen W. Vendor-Agnostic reconfiguration of Kubernetes clusters in cloud federations[J]. Future Internet, 2023, 15(2): 63-63.

装 订 线

谢 辞

回顾在同济大学学习的四年历程，我有幸遇到了许多认真负责的老师和友善的同学，与他们的交往构成了我大学本科四年的宝贵经历。

感谢各位值得尊敬的老师和教授，各位老师的专业知识和技术水平使我受益匪浅，让我成为一名合格的计算机专业的本科毕业生。感谢电信学院计算机科学与技术系的沈坚、方钰、邓蓉三位老师，他们为我撰写并提交了对外申请研究生的推荐信，感谢他们对我的认可。尤其想要感谢沈坚老师，从大一上学期的高程到大三下学期的计算机网络，是沈老师用他的耐心教导将我引入计算机世界的大门，谆谆教诲言犹在耳，学生感激不尽。

感谢曾国荪老师在毕业设计阶段提出的富有见地的选题和过程中的认真指导，在这一阶段的任务中为我指出了前进方向，使我能够顺利完成毕业设计。

感谢父母对我的关心和养育，他们为我提供了良好的生活环境，父母对我的期盼是我前进的动力。

最后，我想感谢过去付出努力的自己。在同济大学求学的这段时间是我人生中的宝贵财富，在四年的时间里我未曾懈怠，实现了个人多方面的成长与成熟。如今即将离开校园，我对自己所付出的和所得到的十分满意，希望在人生的未来旅程中，能够有着更多的期待。

装

订

线