[1]迟殿委.基于Ribbon的微服务通讯及负载均衡的实现[J].电脑与信息技术,2019,27(05):25-27.

[2]林清山.云原生时代的消息与流计算平台——RocketMQ[J].软件和集成电路,2020(09):92-93.

[3]姚刚,王从镔,吴海莉.基于Docker技术的微服务架构探析[J].信息系统工程,2020(04):127-128.

[4]倪学磊.微服务架构与容器技术分析及应用[J].信息与电脑(理论版),2019,31(21):150-151.

[5]辛园园,钮俊,谢志军,张开乐,毛昕怡.微服务体系结构实现框架综述[J].计算机工程与应用,2018,54(19):10-17.

[6]Walls C. Spring Boot in action[M]. Suffolk Country, New York: Manning Publications,2016.

[7]John Carnell, Spring Microservices in action[M]. Suffolk Country, New York: Manning Publications, 2017

[8]陈文轩. 电子商务与中国零售变革[D].浙江大学,2018.

[9]耿德胜.面向微服务架构的容器级弹性资源供给方法[J].信息与电脑(理论版),2019(05):97-99.

[10]宁方美,贺雪梅,牟晋娟.SpringBoot集成Redis缓存技术在企业一卡通系统中的应用[J].电子技术与软件工程,2019(24):133-134.

[11]石建良. 基于RocketMQ框架的消息通信与监控系统[D].华中科技大学,2019.

[12]CSDB.什么是微服务[EB/OL].(2017-11-27).https://blog.csdn.net/wuxiaobingandbob/article/details/78642020

选题理由和意义

当代社会，计算机，信息技术的发展使得商品交易方式由线下搬到了线上，电子商务在中国社会经济发展中起着越来越重要的作用。然而，由于我国进行网络购物的用户规模庞大，在未来的电子商务发展的过程中，势必出现规模巨大的高并发需求，特别是互联网造就了各种节日噱头，导致购物节的密集出现，越来越多的人开始出现了集中买买买的现象，购物平台为了迎合大众，使得系统的业务逻辑越来越复杂。传统企业平台大多采用单体应用架构来开发部署系统，但是随着功能模块的增加，开发人员的增加，单体应用已经显得臃肿不堪，业务耦合性大，不利于人员维护；部署耗时严重，不利于工程的持续交付；技术栈太过单一，后续技术更新困难。急需新的设计架构解决这些问题，提升开发效率，维护效率，业务稳定性，于是单体应用开始向面向服务的架构演进，再到现在更加细粒度的微服务架构，所谓微服务架构就是通过解耦，将单体应用分成一组小型的，便于管理的服务组件，解决大型代码库传统架构带来的复杂问题。根据以上背景和相关调研，结合工作上的需求与业务特色，最终确定利用相关微服务技术实现对原有商品中心系统的改造，利用其可扩展性和快速部署的特性，应对将来不断变化的发展需求。

国内外研究现状和发展趋势

美国是互联网技术的发源地，上世纪90年代美国提出了电子商务的概念，开始诞生亚马逊，ebay的电子商务平台，这些电子商务企业将业务实现了全球化，在亚洲，电子商务市场，更是发展得如火如荼。如今我国的电子商务已经位居世界第二，像阿里巴巴，京东商城等是我国优秀电子商务企业的代表。

早期的电子商务系统，仅通过HTML静态页面向用户展示物品，用户查看完后通过线下方式和商家交互，完成交易行为。后来，随着网页技术的发展，网站由静态页面转向动态页面，数据持久层开始使用MySQL，Oracle等数据库，由于早期系统业务流量较小，单体应用可以满足人们购物需要。随着电商系统的不断发展，越来越多的人开始在网上购物，高并发压力不断增大，传统单体应用架构和面向服务的架构已经满足不了实际需求。2011年微服务架构被提出，其定义为：根据业务逻辑划分，采用全自动部署，有独立进程和轻量级处理的单一应用程序，服务间通过http通信。微服务架构的出现使得应对这些问题变得容易，微服务架构在面向服务的架构基础上进一步划分和改进，模块的力度更细，服务的规模更小，可以实现业务的彻底分离。spring团队适应当今社会需求，推出了spring boot和springcloud项目，选择使用这两个项目可以实现一个高性能，可扩展的，易部署的微服务商品中心系统。

研究目标

明确微服务的定义，使用场景，以及微服务架构是否适用于该系统的改造。微服务架构指的是一种特定的软件应用程序设计方式——将大型软件拆分为多个独立可部署服务组合而成的套件方案，这些服务围绕业务功能建立而成，且凭借自动化部署机制实现独立部署。由于在电子商务系统中业务功能繁多，各种模块服务相对独立，天然适合微服务架构，而且，微服务化是未来趋势，因此可以将原有的商品中心这一大且单一的模块拆分成多个服务，实现系统的微服务化工作。通过调研微服务相关技术栈，结合公司技术人员掌握的编程语言情况，选择springcloud框架作为商品中心改造的技术栈，根据业务特点，将原有的商品中心数据库拆分为三个独立的数据库，根据业务特点，划分不同的微服务系统有mall-product-manager,mall-product-price,mall-product-sales,mall-product-back,mall-product-common,mall-merchants,mall-product-search等服务，推进需求分析和编码工作，推动测试以及部署迭代工作，一步步从完成某个模块的工作到整个系统的完整搭建与开发，将业务功能，非业务功能完善，实现以上系统间的正常调用流程，系统监控等，实现商品中心的重构，部署上线平稳运行，达到预期目标。

研究内容

调研微服务架构的相关技术和重要组件，为商品中心系统选择合适的微服务技术方案，服务注册和发现，服务间通信方式以及服务部署是其关键技术。

研究商品中心改造中各个模块的划分方案以及各个模块对应的功能界限与设计实现，本系统将商品中心划分为8个耦合度低的微服务。

研究该系统的部分基础功能实现，如何使用阿里与作为OSS存储实现文件上传，服务间异步通信，与前端接口的交互设计等。

由于部分功能流程较长，耗时较长，拟采用服务间异步通信方式提升客户端响应速率，服务间异步通信使用消息队列中间件实现，可以有效提升系统的可用性。

拟解决的关键问题

由于整个系统采用微服务架构，系统间的调用链关系是较为难以解决的问题，而调用链的确定需要确认好每个系统的功能界限，不能出现功能重复或功能交差的情况，为了防止当系统某系统不可用而其他服务不停访问导致线程阻塞问题，在调用链上添加了Hystrix实现自我保护能力，Hystrix使用命令模式将所有对外部服务（或依赖关系）的调用包装在HystrixCommand或HystrixObservableCommand对象中，并将该对象放在单独的线程中执行，每个依赖都维护着一个线程池或信号量，线程池被耗尽则拒绝请求而不是让请求排队，记录请求成功，失败，超时和线程拒绝的次数，当服务错误百分比超过了阈值，熔断器开关自动打开，一段时间内停止对该服务的所有请求，请求失败，被拒绝，超时或熔断时执行降级逻辑，近实时地监控指标和配置的修改。现将系统中涉及到的服务模块列举如下：

mall-product-back，提供商品最基础的功能，是与数据库，消息队列直接接触的模块，如商品添加，商品审核，产品模板配置等。

mall-product-manager，mall-whitehorse-api面向管理平台的接口，该模块实现了mall-product-price,mall-product-back,mall-product-stock,mall-product-common等模块接口的整合，统一向管理平台提供接口api。

mall-product-price，该模块单独处理价格相关操作，由于网上商城价格经常变动，因此单独抽离一个模块，单独管理，包括普通价格修改和会员价格修改。

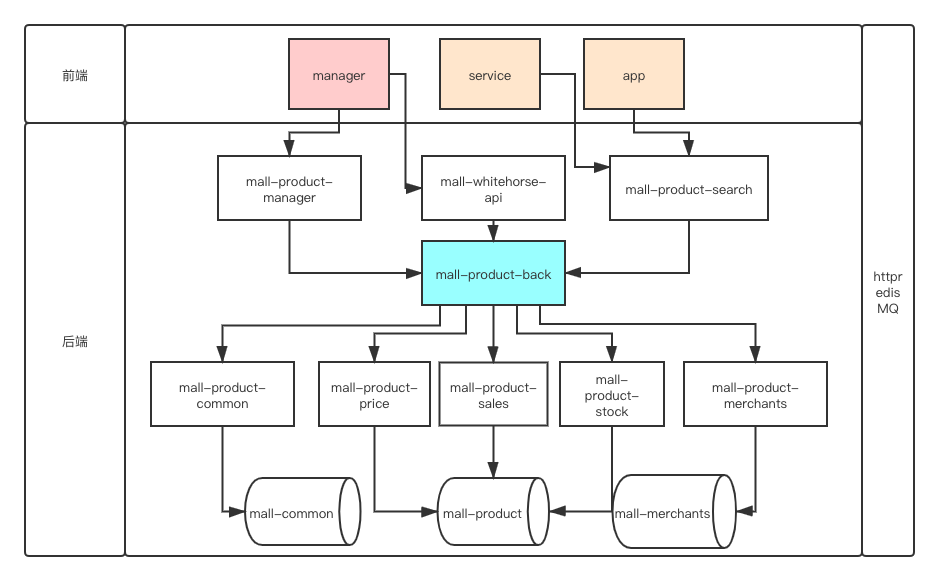
mall-product-stock，与库存相关的操作使用该模块实现，有查询库存，预占库存，增减库存等，因库存也会经常变动，所以单独抽离出一个模块。

mall-product-common，提供通用操作，商品的特点相对固定，如商品会有类目，属性，品牌，税率等，该模块实现以上功能的管理。

mall-merchants，商家处理模块，实现对商家支持的发货区域，运费，资质，商品关联品牌，商家本身的功能。

mall-product-search，对外暴露的接口模块，是对其他项目提供的外部接口，实现对mall-product-back的调用

mall-product-sales，销量处理模块，实现对销量的查询，增减。



研究方法，技术路线

调查法，调查市面上常用的服务架构，在以Java语言为主的开发框架中主要有两个微服务架构，一个是Spring Cloud，另一个是Apache Dubbo。Apache Dubbo是一个分布式服务框架，致力于提供高性能和透明化的RPC远程服务调用方案，以及SOA服务治理。Apache Dubbo最开始是由阿里巴巴设计开发并维护的，但是2015年后阿里巴巴不再维护，后捐献给了Apache基金会，现在由Apache维护，但是错过了近几年的黄金发展期，社区活跃度并不高，在国内逐渐被抛弃。Spring Cloud利用Spring Boot的开发便利巧妙地简化了分布式系统基础设施的开发，如服务发现和注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以使用SpringBoot的开发风格做到一键启动和部署，Spring Cloud提供了Docker及Kubernetes微服务编排支持，支持项目的容器化部署。Spring Cloud是Spring Source的产物，刚出现时在国外比较流行，但最近几年在国内逐渐流行起来，越来越多的企业开始使用该框架，由于基于SpringBoot开发，对Java开发者特别友好，上手极快，受到了开发者的欢迎，社区活跃度极高。

文献研究法，通过查阅与微服务、网上购物系统、容器化等的相关论文，发行在Spring Cloud上有所研究的文献较多，有很强的文献借鉴价值，从宏观上掌握总体架构特点。

技术路线

比较了目前主流的微服务框架dubbo和springcloud，选择springcloud作为实现该商品中心重构的主要技术栈，包括euraka注册中心，ribbon负载均衡，feign客户端，携程Apollo分布式配置中心，中间件选择rocketmq来做系统间异步数据的消息化处理，选择redis做热数据的缓存，选择MySQL做持久数据的保存，系统的持续部署，持续发布采用jenkins+kubernetes实现，形成一整套从开发到部署测试的完整流程解决方案。现将系统中涉及到的组件介绍如下：

Spring Cloud Euraka注册中心

Spring Cloud Euraka是springcloud集合中的一个组件，基于Netflix Euraka实现。主要用于服务注册和服务发现，由多个服务实例组成，这些服务实例可以分为Euraka Server和Euraka Client两种，而Euraka Client又可以分为服务提供者和服务消费者，其负责的功能分别如下：Euraka Server提供服务注册和服务发现机制，服务提供者将自身服务注册到Euraka Server，从而使服务消费方能找到，服务消费者从Euraka Server获取服务注册列表，从而可以消费服务。Euraka是高可用和可伸缩的服务发现组件，突出可用性，因此会牺牲部分一致性，保证能返回数据，服务是可用的。

Spring Cloud Ribbon

Spring Cloud Ribbon是一个基于HTTP和TCP协议的客户端负载均衡工具。基于Netflix Ribbon实现，通过该组件，我们可以轻松的实现将面向服务的REST模板请求自动转换成客户端负载均衡的服务调用。Spring Cloud Ribbon只是一个工具框架，不需要像Euraka，配置中心等单独部署，但是它存在于使用Spring Cloud构建的微服务和基础架构当中，因为每一个微服务间的调用，网关的转发请求，实际上都是通过Ribbon实现的。支持的负载均衡策略有随机，轮询，一致性hash，哈希，加权。

Spring Cloud Open Feign

Spring Cloud Open Feign是一个声明式的Web Service客户端。它的出现使开发Web Service客户端变得很简单。使用Feign只需要创建一个接口加上对应的注解，比如：@FeignClient 注解。Feign有可插拔的注解，包括Feign注解和AX-RS注解。Feign也支持编码器和解码器，Spring Cloud Open Feign对Feign进行增强支持Spring MVC注解，可以像Spring Web一样使用HttpMessageConverters等各种信息转换器。Feign 是一种声明式、模板化的HTTP客户端。在 Spring Cloud 中使用Feign，可以做到使用HTTP请求访问远程服务，就像调用本地方法一样的，开发者完全感知不到这是在调用远程方法，更感知不到在访问HTTP请求，完全屏蔽了内部实现细节。具体特性如下：

可插拔的注解支持，包括 Feign 注解和AX-RS注解。

支持可插拔的HTTP编码器和解码器。

支持Hystrix和它的Fallback。

支持Ribbon的负载均衡。

支持HTTP请求和响应的压缩。

在使用过程中我们只需要依赖注入Bean，然后调用对应的方法传递参数即可。

Apollo分布式配置中心

项目中配置文件比较繁杂，而且不同环境的不同配置修改相对频繁，每次发布都需要对应修改配置，如果配置出现错误，需要重新打包发布，时间成本较高，因此需要做统一的分布式注册中心，能做到自动更新配置文件信息，解决以上问题。Apollo（阿波罗）是携程框架部门研发的分布式配置中心，能够集中化管理应用不同环境、不同集群的配置，配置修改后能够实时推送到应用端，并且具备规范的权限、流程治理等特性，适用于微服务配置管理场景。

Apollo阿波罗的功能特点有以下几点：

1.用户在Apollo修改完配置并发布后，客户端能实时（1秒）接收到最新的配置，并通知到应用程序。

2.统一管理不同环境、不同集群的配置。

3.所有的配置发布都有版本概念，从而可以方便的支持配置的回滚。

4.配置修改实时生效（热发布）。

5.用户在Apollo修改完配置并发布后，客户端能实时（1秒）接收到最新的配置，并通知到应用程序。

6.版本发布管理，所有的配置发布都有版本概念，从而可以方便的支持配置的回滚。

7.灰度发布，支持配置的灰度发布，比如点了发布后，只对部分应用实例生效，等观察一段时间没问题后再推给所有应用实例。

RocketMQ

RocketMQ是一款低延迟、高可靠、可伸缩、易于使用的消息中间件，具有以下特性:

1.支持发布/订阅（Pub/Sub）和点对点（P2P）消息模型。

2.在一个队列中可靠的先进先出（FIFO）和严格的顺序传递

3.支持拉（pull）和推（push）两种消息模式。

4.单一队列百万消息的堆积能力。

5.支持多种消息协议，如 JMS、MQTT 等。

6.分布式高可用的部署架构,满足至少一次消息传递语义。

RocketMQ整体可以分为4个角色，分别如下：

1 NameServer

可以理解为是消息队列的协调者，Broker向它注册路由信息，同时Client向其获取路由信息，如果使用过Zookeeper，就比较容易理解了，但是功能比Zookeeper弱；NameServer本身是没有状态的，并且多个NameServer直接并没有通信，可以横向扩展多台，Broker会和每一台NameServer建立长连接；

2 Broker

Broker是RocketMQ的核心，提供了消息的接收，存储，拉取等功能，一般都需要保证Broker的高可用，所以会配置Broker Slave，当Master挂掉之后，Consumer然后可以消费Slave；Broker分为Master和Slave，一个Master可以对应多个Slave，Master与Slave的对应关系通过指定相同的BrokerName，不同的BrokerId来定义，BrokerId为0表示Master，非0表示Slave；

3 Producer

消息队列的生产者，需要与NameServer建立连接，从NameServer获取Topic路由信息，并向提供Topic服务的Broker Master建立连接；Producer无状态，看集群部署；

4 Consumer

消息队列的消费者，同样与NameServer建立连接，从NameServer获取Topic路由信息，并向提供Topic服务的Broker Master，Slave建立连接；

在介绍完以上4个角色以后，还需要重点介绍一下上面提到的Topic和Message Queue；字面意思就是主题，用来区分不同类型的消息，发送和接收消息前都需要先创建Topic，针对Topic来发送和接收消息，为了提高性能和吞吐量，引入了Message Queue，一个Topic可以设置一个或多个Message Queue，有点类似kafka的分区(Partition)，这样消息就可以并行往各个Message Queue发送消息，消费者也可以并行的从多个Message Queue读取消息；

Redis

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。 它支持多种类型的数据结构，如 字符串（strings）， 散列（hashes）， 列表（lists）， 集合（sets）， 有序集合（sorted sets） 与范围查询， bitmaps， hyperloglogs 和 地理空间（geospatial） 索引半径查询。

可行性分析

目前国内外包括阿里，京东等企业的对外购物平台都实现了系统的微服务化，虽然各家公司采用的技术栈不尽相同，但是其功能原理是类似的，因此在实现的过程中有大量可以借鉴的经验。Spring Cloud的生态非常好，有诸多的开源平台，社区贡献者提供较为良好的实现方案，Spring Cloud的再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。该项目采用的开发语言为Java，是目前开发基于B/S架构的系统最为流行的语言，有大量的工具包可以使用，避免重复造轮子，提高了开发效率。

本研究是针对原有系统的改造，业务上符合行业上的规定，技术上采用当前流行的微服务架构，符合某公司的发展规划

对原商品中心系统的改造，业务功能改变较少，主要是技术栈的选型，工作量适中

研究者本人从事一线开发多年，其中服务端开发近7年时间，有一定开发经验，近几年也不断学习新技术，提升自己的技术水平，能够胜任该项工作

在公司业务发展遇到瓶颈时立项，公司已有持续集成，持续部署等平台，且有运维人员支持，云平台资源充足

论文的创新之处

本论文的研究内容是针对电商系统中商品中心单个模块进行的微服务重构，粒度相对与整个项目来说较小，使用当前较为流行的springcloud技术栈实现微服务划分，其中在一些耗时的操作上使用了消息机制解决异步通信的问题，部署采用Jenkins+kubernetes，是比较新的部署方式

可预期的成果

完整实现商品中心的微服务改造

项目上线正常运行，稳定对外提供服务

面对系统需求的持续迭代时，能够模块化开发，模块化部署发布

可能遇到的问题

需求模糊问题

与产品经理随时保持沟通，明确需求

资源调度问题

督促运维人员保证云平台资源的供给

部署测试问题

加强系统的功能性建设，稳定性设计，降低bug率

系统调用链问题

明确各个系统的功能边界，以及各个系统的特点，防止出现重复开发，重复引用

论文进度计划

2020年8月，查阅相关资料，完成项目需求的功能分析与技术选型调研

2020年9月，搭建项目，进行系统设计，数据库设计

2020年10月，继续开发功能，查阅相关文献，开始撰写开题报告

2020年11月，项目重构完成，验证上线，完成开题报告

2020年12月，开始构思毕业论文

2021年1月，初步撰写毕业论文

2021年2月，继续撰写毕业论文

2021年3月，形成论文初稿，优化论文

2021年4月，论文修改，完善论文

2021年5月，撰写PPT报告，进行论文答辩