电子科技大学中山学院

University of Electronic Science and Technology of China , Zhongshan Institute

毕业设计(论文)

基于微服务的分布式交易系统

教学单位：计算机学院

专业名称：软件工程

学　　号：2018010102171

学生姓名：熊强

指导教师：何怀文(副教授)

指导单位：计算机学院

完成时间：2022 年 5 月 16 日

电子科技大学中山学院教务处制发

独创性声明

本人声明：所呈交的学位论文，是本人在指导老师的指导下独立进行研究工作所取得的研究成果。除文中特别加以标注、所列参考文献和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学中山学院或其它教育机构的学历、学位或课程、培训等非获奖类证书而使用过的成果材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。本人完全意识到本声明的法律后果由本人承担。

作者签名：熊强　　日期：　2022 年 5 月 16 日

论文使用授权声明

本学位论文作者完全了解电子科技大学中山学院有关保留、使用学位论文的规定，本人同意学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学中山学院可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

论文级别：本科论文

学科专业：软件工程

论文题目：基于微服务的分布式交易系统

作者签名：熊强　　指导老师签名：[illegible]

日期：　2022 年 5 月 16 日

基于微服务的分布式交易系统

摘要

互联网的快速发展以及网络速度的大幅提升给人们的生活提供了极大的便利性，人们足不出户就能作为消费者随时随地的浏览自己喜欢的商品，消费者不需要再忍受现实生活中因交易地点路途遥远所导致的时间浪费以及排队等待。消费者想要在不同地方对多种商品进行对比，往往很多时候会因为店员的跟随，感觉不自在并且也会因为店员的推销而无法客观的对多种商品进行对比，同时受限于实体店面积的大小，大量消费者无法在短时间内进行消费。因此消费者迫切需要一个在线交易平台，一个能随时随地搜索自己想要的商品，能对商品的各项属性进行对比，从而快速得知商品之间差异，并且能解决单体式应用交易系统无法短时间处理海量用户消费的交易平台。

哈哈商城采取了分布式微务的结构方法来设计，采用了 Spring Cloud 微务架构把功能较繁杂的应用拆分为若干个微务功能模块，并通过 Mysql 数据库持久化的保存数据分析；Redis 数据库将对热点数据信息提供缓存，提高系统并发量；RabbitMQ 消息队列进行流量控制，应用解耦，让哈哈商城无论是面对短时间内消费订单请求暴增的秒杀活动，还是面对黑客的恶意攻击都能全天候稳定运行。

关键词：分布式；Spring Cloud；高并发；秒杀；

Distributed trading system based on Microservices

Abstract

The rapid development of Internet and network speed boost provides great convenience to people's life, people never leave home can be anywhere at any time as consumers browse the merchandise that oneself like, consumers don't need to bear in real life, remote location for the deal in the waste of time and waiting in line.When consumers want to compare various commodities in different places, they often feel uncomfortable and unable to compare various commodities objectively because of the assistant's promotion. At the same time, limited by the size of the physical store, a large number of consumers cannot make consumption in a short time.Therefore, consumers are in urgent need of an online trading platform, which can search for the goods they want anytime and anywhere, compare the attributes of the goods, so as to quickly learn the differences between the goods, and solve the problem that the single application trading system cannot deal with massive user consumption in a short time.

Ha Ha Mall adopts distributed micro-service architecture to design. Based on SpringCloud micro-service framework, applications with complex functions are divided into multiple micro-service functional modules, and Mysql database is used to store data continuously.Redis database will cache hot data, improve the system concurrency;RabbitMQ message queue traffic control, application decoupling, ha ha mall in a short period of time in the face of a sudden increase in consumer order to kill activities, or malicious attacks in the face of hackers 24/7 stable operation.

Key Words：Distributed；Spring Cloud；High Concurrency；Second Kill System

目录

图目录

表目录

第 1 章 绪论

1.1 课题背景

“双十一”的出现让交易系统的交易量突破单日千亿，进而打破了人们认为电商不可能赶超零售的传统认知[1]。而实际上“双十一”就是秒杀活动最具代表性的体现，它通过优惠的价格吸引大量的消费者，在零点限定时间内进行秒杀消费，既能刺激消费者的购买欲望，又能提升平台形象，增加平台的影响力，同时在短暂的时间内创造巨大的交易额和销售量。因此秒杀活动对在线交易平台的发展和盈利方面具有不可或缺的作用。

秒杀活动通常是以较低的价格吸引大量的顾客，特征是在较短的持续时间内，有大量分别来自五湖四海的用户进行高频率访问。秒杀活动的商品往往数量少，价格优惠力度大并且只能在有限时间内进行秒杀，因此秒杀订单相比于传统订单具有高并发特点的同时也要求商品交易服务在高并发请求中始终保持稳定，避免出现超买超卖等现象。秒杀系统一般采用分布式的架构进行设计和部署以支持高并发、高可用的系统要求，并且后期随着用户数量的增长，系统并发量的要求也会不断地提升，所以秒杀系统也需要具备符合要求的伸缩和扩展能力。

1.2 目的意义

本论文采用分布式微服务架构实现的交易系统是为了解决使用传统单体式架构项目所面临的单体服务器并发量低、功能模块间耦合度高、后期运维迭代难度大、以及项目中所有功能集成部署在单个服务器上容易面临服务器故障导致整个交易系统崩溃等问题[2]。哈哈商城是一个具有高拓展性，高可用与高并发特点的在线交易系统，系统中各功能模块的单体结构以微服务形式存在（包括购物车管理服务、全局权限认证服务、库存服务、秒杀服务等）。系统中各服务之间通过 Nacos 进行服务治理，解决各服务模块之间的调度安全问题，保证交易系统的稳定运行。系统通过不同角色（用户、商家、管理员）进行系统需求分析，确定系统的业务功能，保证用户与商家以及管理者之间各自具有不同的功能权限，构建一个让用户放心，商家安心，管理者省心并且能全天候 24 小时稳定运行的在线交易平台－哈哈商城，当完成业务功能开发之后需要进行系统性的全面测试以保证交易系统的可用性[3]。

1.3 论文主要工作

采取分布式微服务的结构方法设计，将功能复杂单体式的应用拆分为多种微服务，按不同功能业务进行区分以独立研发和部署，服务之间可以相互通讯，这有利于集群化的部署和运维。本论文详细的设计并实现了面向互联网广大用户的在线交易系统，系统以 SpringCloud 作为微服务开发框架，Mysql 作为服务独享的数据库，并在设计过程中充分利用了 Redis 高性能缓存以及 RabbitMQ 异步处理等特点极大地提高了系统在高并发场景下的响应能力，最后对订单秒杀功能模块进行了全方位系统性测试，将系统各功能模块的测试结果和性能检测结果以图表方式直观地展示出系统所具有的性能参数。

本文内容结构如下：

第 1 章 概述课题背景、课题目的意义和论文的主要工作以及内容结构。

第 2 章 介绍交易系统的技术背景，说明微服务架构所涉及的相关技术。

第 3 章 从系统用例，功能模块，数据处理能力，系统属性要求，时间特性要求等角度对交易系统进行系统分析。

第 4 章 阐述交易系统中的总体框架设计，各功能模块的数据库设计以及业务功能的详细设计。

第 5 章 对交易系统中的关键模块的和关键技术通过代码与图表结合的形式进行讲解，测试系统完成后的性能。

第 6 章 论文总结以及对未来的展望。

第 2 章 技术背景

2.1 微服务架构

什么是微服务？微服务系统是目前比较流行且前沿的架构风格，一个分布式系统，可以将系统根据业务分割为不同的业务单元，从而克服了单体式系统性能不够的困难问题[4]。一个大型软件应用系统由许多业务单元所构成，而系统内的业务单元都可独立部署，因此不同业务单元间也是松耦合的。所以本文重点是对该体系重要技术的介绍，并了解微服务架构设计使用的重要技术手段，微服务体系所实现的重要技术分为由 Spring Cloud 与 Spring Cloud Alibaba 所关联的服务组件、Elasticsearch 搜索引擎，以及 RabbitMQ 消息队列等，基于微服务的分布式交易系统框架如下图 2-1 所示。

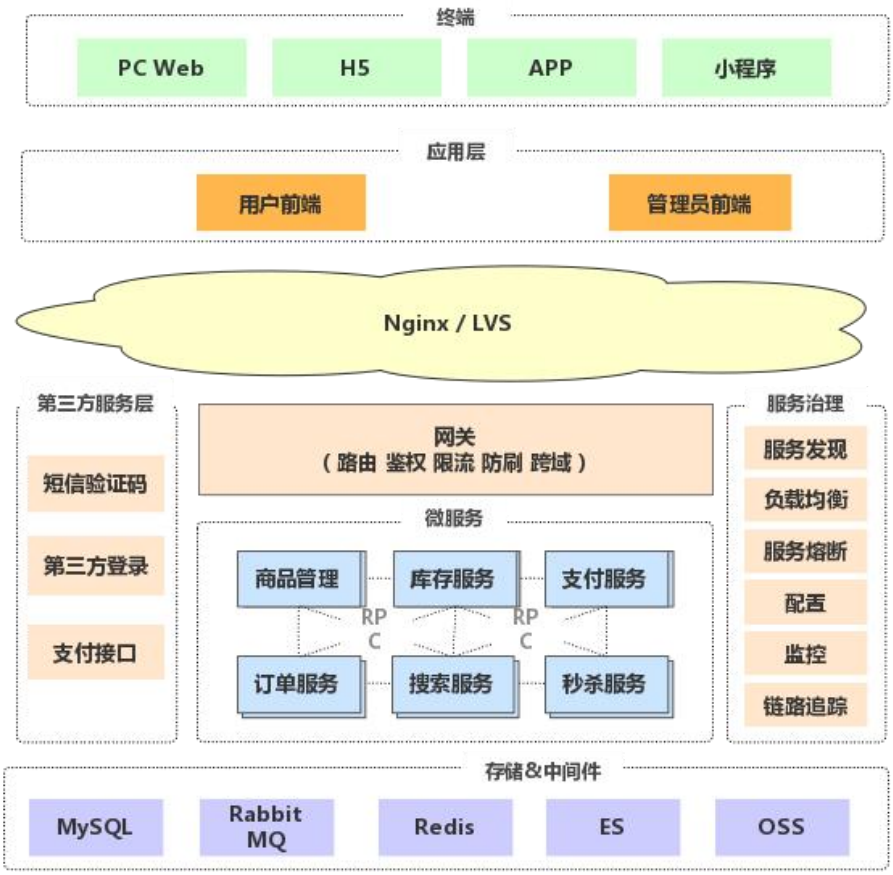

图 2-1 基于微服务的分布式交易系统框架图

2.2 Spring Cloud

Spring Cloud 是一款基于 Spring Boot 实现的微服务框架。它利用 Spring Boot 精简了分布式系统基础架构的研发，Spring Cloud 包含了 Spring Cloud Gateway、Spring Cloud Config 等将近二十个业务模块，这些组件满足了业务控制、业务网关、智能网络路由、负载均衡、熔断器、监控追踪、分布式消息队列、分配管理等行业的方案[5]。Spring Cloud 也被称为构建分布式微服务体系中的全家桶，它并不是某一种技术，而是对各种微业务解决方案以及架构的有序集成。它把市面上成熟的、已经认证的微服务架构集成在一起，并采用了 Spring Boot 的设计思想进行再包装，以屏蔽掉其中繁琐的设置和实施设计原理，最后给开发人员带来了一个简洁易懂、易部署和易于维修的分布式信息系统设计工具箱。

2.3 Spring Cloud Gateway

Spring Cloud Gateway 是 Spring Cloud 团队通过 Spring 5.0、Spring Boot2.0，以及 Project Reactor 等研发的高性能 API 网关组件[6]。Spring Cloud Gateway 旨在提供一种简单而有效的途径来发送 API，并为它们提供横切关注点，例如：安全性，监控/指标和弹性，直接访问服务和通过 API 网关访问服务两种服务访问的区别如下图 2-2 所示。

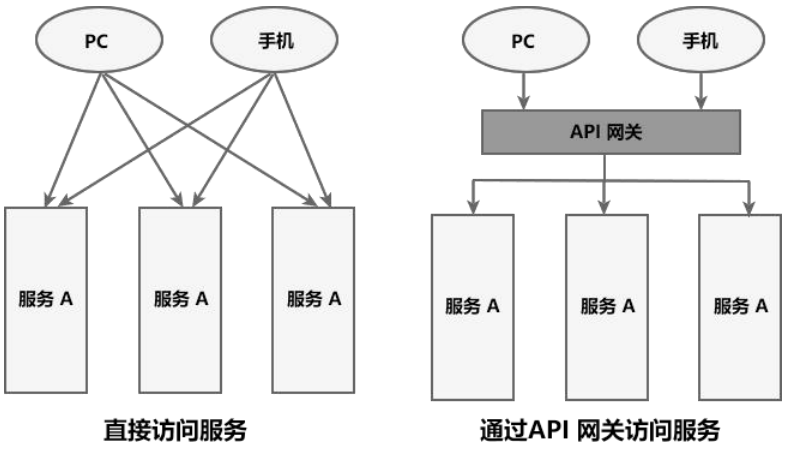

图 2-2 Gateway 服务访问方式对比图

Spring Cloud GateWay 在本项目中最主要的功能是路由转发，而在定义转发规则时主要涉及了以下三个核心概念，如下表 2-1 所示。

表 2-1 Gateway 核心概念表

核心概念	描述
Route（路由）	网关最基本的模块。它由 ID、目标 URI、断言（Predicate）和过滤器（Filter）组成。
Predicate（断言）	路由转发的判断条件，可以通过 Predicate 对 HTTP 请求进行匹配，例如请求方式、请求路径、请求头、参数等，如果请求与断言匹配成功，则将请求转发到相应的服务。
Filter（过滤器）	过滤器，使用它对请求进行拦截和修改，还可以使用它对上文的响应进行再处理。

当客户端向 Spring Cloud Gateway 发出请求时。先将请求交给 Geteway Handler Mapping 处理器确定请求与路由是否匹配，如果匹配则将其发送到 Gateway Web Handler 处理请求[7]。网关设置过滤器去根据用户的请求携带的非法信息，检查请求的路径，判断当前访问的路径是否符合要求的身份，以确定是否一个变更请求在短时间内多次访问，每次请求对应一个检查，完成后将当前请求记录到日志中，以便于往后能根据日志信息排查错误。Spring Cloud Gateway 工作流程如下图 2-3 所示。

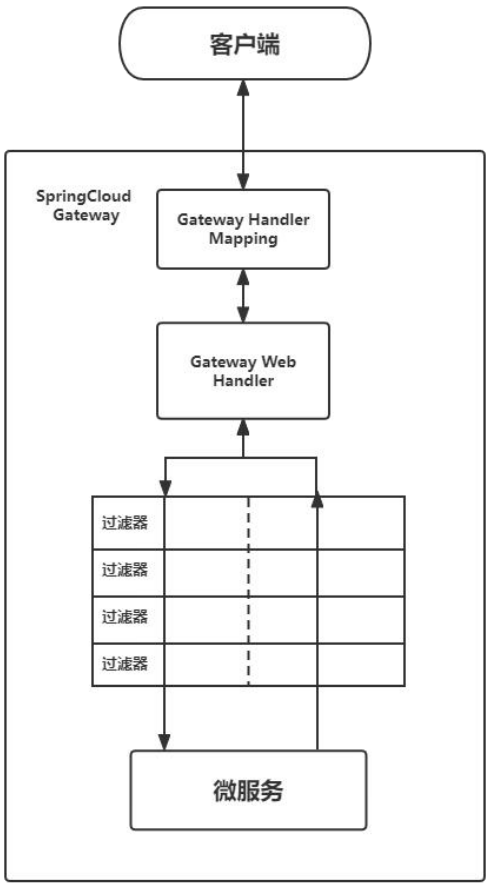

图 2-3 Gateway 工作流程图

2.4 Spring Cloud Alibaba-Nacos

Nacos 英文全程为作为 Dynamic Naming and Configuration Service，是一个由阿里巴巴团队使用 Java 语言开发的开源项目，由于微服务系统的分布式体系结构使得服务独立运行在不同的模块中，因此服务之间的调用关系是通过系统接口来访问的。因此，在查询其他服务的接口时，是否能够发现当前正在运行的服务，是否能够正确获取所需的服务信息就显得尤为重要[8]。Nacos 致力于帮助发现、配置和管理微服务。配置相关信息后即可将当前运行的服务在 Nacos 上进行注册，注册后的服务将保存在列表中。当服务被调用时，它将转到服务列表进行检查。找到当前运行的服务后，即可获得服务运行的接口，实现功能对接。Nacos 让构建、交付和管理微服务平台更加高效且容易。Nacos 作为服务注册中心用于实现分布式微服务之间的服务注册与发现，流程如下图 2-4 所示。

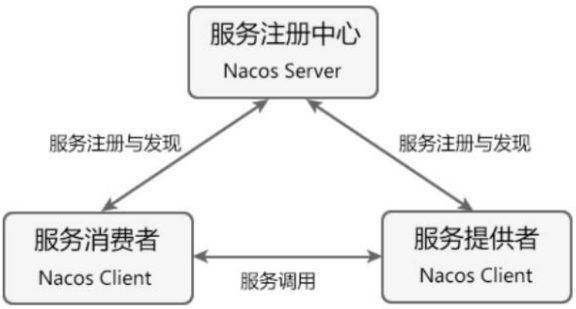

图 2-4 Nacos 服务注册流程图

2.5 Spring Cloud Alibaba-Sentinel

Sentinel 是由阿里巴巴中间件技术集团所研究的开源项目，是一个面对分布式微业务结构的轻量级高可用流量管理组件[9]。从用户访问流量管理、熔断降级、操作系统负载维护等几个层面，保证业务之间的稳定性。流量控制(Flow Control)，基本原理是监测使用访问流量的 QPS 或并发线程数等技术指标，当到达特定阈值范围时对访问流量加以限制，防止网络系统被瞬时的访问流量峰值冲垮，保证应用高可用性。除了流量控制以外，对调用链路中不稳定的资源实行熔断降级(Fusing the drop)也是保证高可用性的关键举措之一。因为调度关系的复杂性，一旦调度链路中的某些信息不平衡，最后就会造成请求产生增加。Sentinel 熔断降级会在调度链路中的一个来源经常出现不平衡状况时(比如调度超时或异常比率上升)，对这些来源的使用加以控制，让请求快速失败，以防止因影响到其他人的资源而造成级联错误。当资源被降解后，在随后的降级时刻窗口内，用户所有对该资源的调用都将自行熔断，从而保护交易平台的稳定性。

2.6 Elasticsearch

Elasticsearch 是一个开源的搜索引擎，可以用于处理文本、地理空间（如坐标）、结构化（如 DB 中的表）、非结构化（如报表、图片）等数据，然后通过简单的 REST API 对其搜索。它的最大特点就在于分布式以及实时速度，可部署到数百甚至上千台服务器上，以便存储处理海量的数据，而且其速度仍然能达到秒级[10]。

它的底层使用的是 Apache Lucene，是一个效率高、能力强劲的搜索引擎库，不过它只是一个库，需要使用 Java 才能集成到应用程序中。因此，Elasticsearch 对其进行了封装，屏蔽了底层的复杂性，对外只提供了简单的 RESTful API。当 Elasticsearch 接收到像 Logstash 这种工具传输过来的数据后便会以文档的形式去分析提取索引，压缩数据，按配置的分片规则将数据均匀存储。在完成这些后，就可以进行可视化查询了，例如使用 Kibana 面板查看。

由于 Elasticsearch 具备了易用性、实时分析、全文搜索、分布部署、高可用等特性，所以除了用来做日志的处理分析外，还可以应用在安全分析、指标分析、性能监控等场景需求。

2.7 RabbitMQ

RabbitMQ 是一种消息代理，一个消息系统的媒介，它能够确保消息系统在信息传递过程中的安全性，消息系统允许软件、应用之间的相互连接并支持扩展，能够使所有的应用设备相互连接起来形成一个规模更大的应用，并且可以让所有应用设备都与数据进行连接[11]。消息管理系统通过把消息的发出与接收各自分开，来进行对应用的异步和解偶，非阻塞处理和推送信息。消息队列的主要功能有如下三种：解耦、异步、削峰。可以极大地降低项目的耦合度，提高项目的并发量。在本次交易系统中中消息队列主要应用于保证系统的高可用性，RabbitMQ 应用流程如下图 2-5 所示。

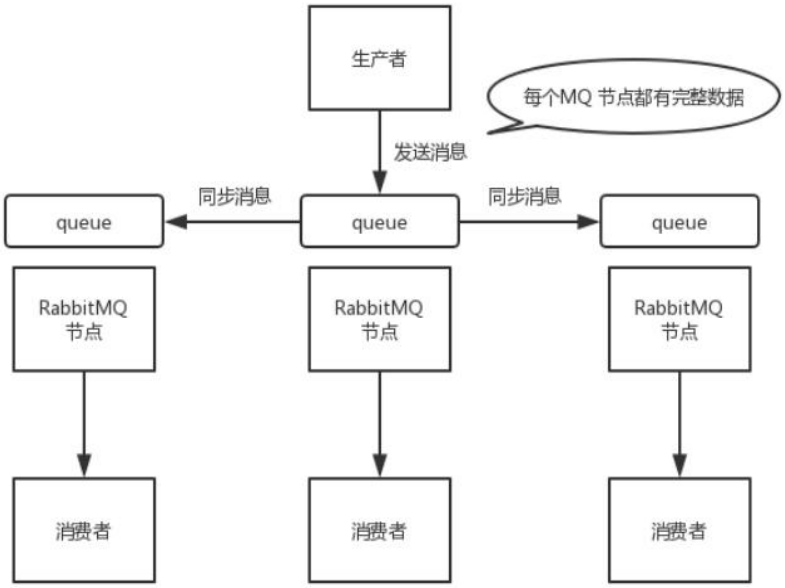

图 2-5 RabbitMQ 应用流程图

以上主要描述了开发过程中将使用到的技术需求，对于 Spring Cloud 和 Spring Cloud Alibaba 的相关组件部分，介绍了网关 Gateway、注册中心 Nacos、限流降级 Sentinel。以上微服务组件，对于系统微服务的架构设计起到了至关重要的作用。同时还介绍了分布式系统中较为关键的技术，搜索引擎 ElasticSearch 和 RabbitMQ 消息队列，通过搜索引擎，能够帮助用户更好对交易信息进行检索，找到自己满意的商品。通过消息队列，保证了消息的异步消费，系统的流量削峰，降低系统耦合性，极大地提升了系统并发能力。

2.8 Redis

Redis 是一款采用键值对格式将数据存储于内存中的非关系型数据库，Redis 中含有多种数据类型，比如 String（字符串），Hash（哈希散列），List（列表），Set（集合）和 Sorted Set（有序集合）等，存储数据以及读取数据的性能极高，通常被广泛应用于系统热点数据的缓存服务[12]。Redis 还可用于提高系统的访问速度和并发能力，Redis 的数据结构特点以及应用场景如下表 2-2 所示。

表 2-2 Redis 数据结构表

类型	简介	特性	场景
String（字符串）	二进制安全	可以包含任何数据，比如 jpg 图片或者序列化的对象，一个键最大能存储 512M	控制数据库表主键 id，为数据库表主键提供生成策略，保障数据库表的主键唯一性，定时任务设置
Hash（哈希散列）	键值对集合，即编程语言中的 Map 类型	适合存储对象，并且可以像数据库中 update 一个属性一样只修改某一项属性值（Memcached 中需要取出整个字符串反序列化成对象，修改完后再序列化成字符串）	存储、读取、修改用户属性
List（列表）	链表（双向链表）	增删快、提供操作区间段元素的 API	1、最新消息排行等功能（比如朋友圈的时间线） 2、消息队列
Set（集合）	哈希表实现，元素不重复	(1) 添加、删除，查找的复杂度都是 O(1) (2) 为集合提供了求交集、并集、差集等操作	1、共同好友 2、利用唯一性，统计访问网站的所有独立 ip 3、好友推荐时，根据 tag 求交集，大于某个阈值就可以推荐
Sorted Set（有序集合）	将 Set 中的元素增加一个权重参数 score，元素按 score 有序排列	数据插入集合时已经进行天然排序	1、排行榜 2、带权重的消息队列

Redis 同时也具有如下特点：

(1) 高性能： 操作完全基于内存，CPU(Central Processing Unit，中央处理器)不会成为瓶颈，无需多线程，采用单线程，因此减少了线程切换的开销，读操作达到 10 万 TPS(Transactions Per Second，每秒处理事务)，写操作达到 7-8 万 TPS。

(2) 数据持久化：存储在内存中的缓存数据可以持久化存储到硬盘中，且支持日志 追加和快照映像，可以在宕机时根据硬盘中存储的操作日志将数据还原至内存中。

(3) 集群部署：可以在多个节点之间进行数据共享，支持主从同步

(Master--Slave)。Master 节点也能够把所有数据都同步至 Slave 节点，同时在 Master 节点出现故障时，选择一个 Slave 节点升级成 Master 节点，保证系统的高效稳定。

(4) 原子性：双向链表的实现基于 Push/Pop 和 Add/Remove 等命令，因此保证了操作结果的原子性，要么成功或者失败，不会存在中间状态。

(5) 顺序性：Redis 为单线程程序，天然保证单机执行的顺序性。

第 3 章 系统分析

3.1 功能需求分析

哈哈商城在线交易系统为对象，从使用者的角度对该系统的需求进行分析，使用者包括用户和管理员。通过角色要在该系统满足的基本功能为出发点，对系统需求进行详细的分析，用户的需求是消费，因此系统的功能要满足用户的消费需求，而对于管理员来说，通过后台的管理能够使用户获得更好的购物体验。因此用户与管理员功能共同结合才能使系统更好为用户服务，通过角色功能用例图更好地了解角色与功能之间的关系。最后通过系统性能测试数据展示分布式系统的性能。系统根据角色的不同，将整体角色分为用户、商家、顾客。各个角色在交易系统中有着各自不同的功能。

3.1.1 系统用例

(1) 由下面图 3-1 用户功能用例图可以看出用户在系统中属于游客状态，可以搜索产品或者查看自身所需要产品，看到商品的详细信息从而做出选择，并且游客能够通过注册会员账号进行会员注册，通过登录系统转为会员状态进行消费，当然注册方法也不仅仅限于通过账号或者密码注册，并且游客还能够使用 App 扫码登录，也可以通过手机短信验证登录。

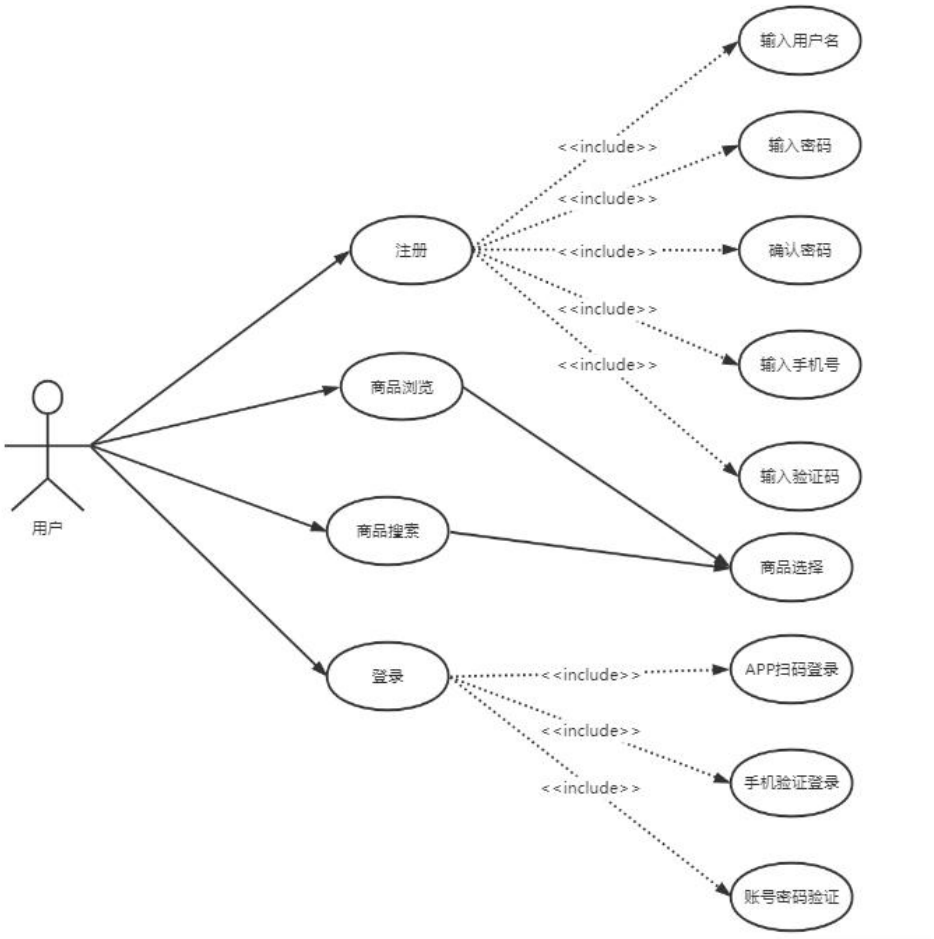

图 3-1 用户用例图

(2) 由下面图 3-2 顾客功能用例图可以看出顾客即已登录的用户，顾客作为系统中的主要消费者，交易系统的功能应该尽可能地满足顾客交易需求。顾客可以通过商品浏览，商品搜索，购物车管理，收货信息管理，订单维护等功能，以保证顾客的消费需求。

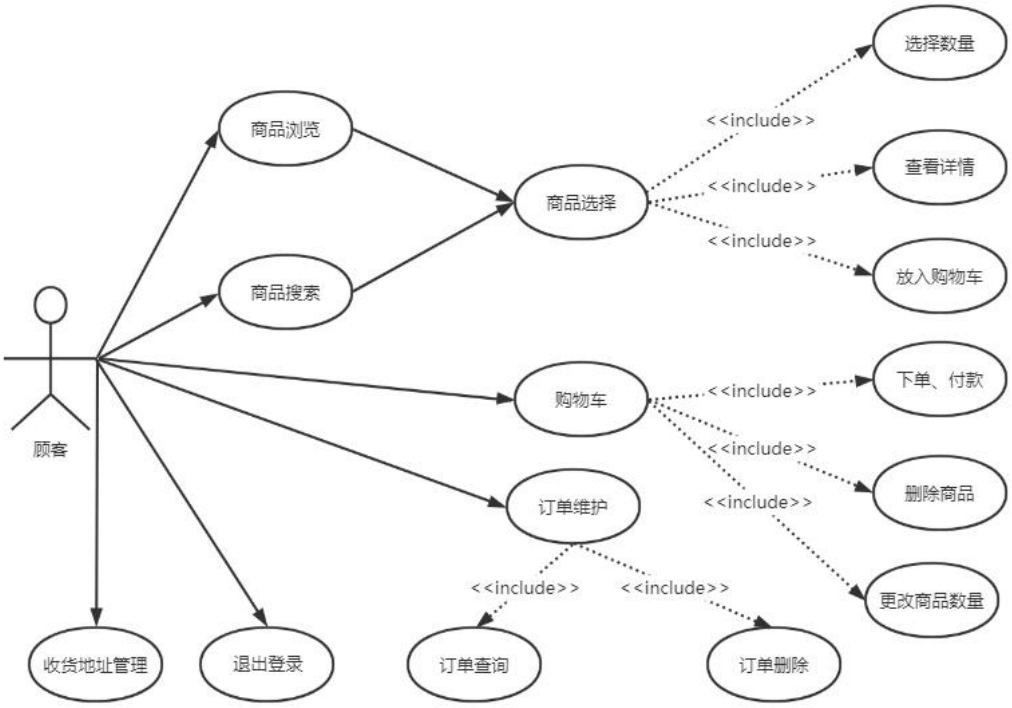

图 3-2 顾客用例图

(3) 由下面图 3-3 商家功能用例图可以看出商家即管理员，能够登录前台网站页面，检验自己的商品是否上架。商家可以登录后台的商家管理页面从而对商品进行上架，修改，下架删除等功能的实现，检索商品查询自己所管理的商品是否已经上架，通过流量图表直观的数据显示最近几天店面的流量情况，交易系统提供留言管理功能可以让商家在特殊情况下，例如疫情时通知顾客注意防护。

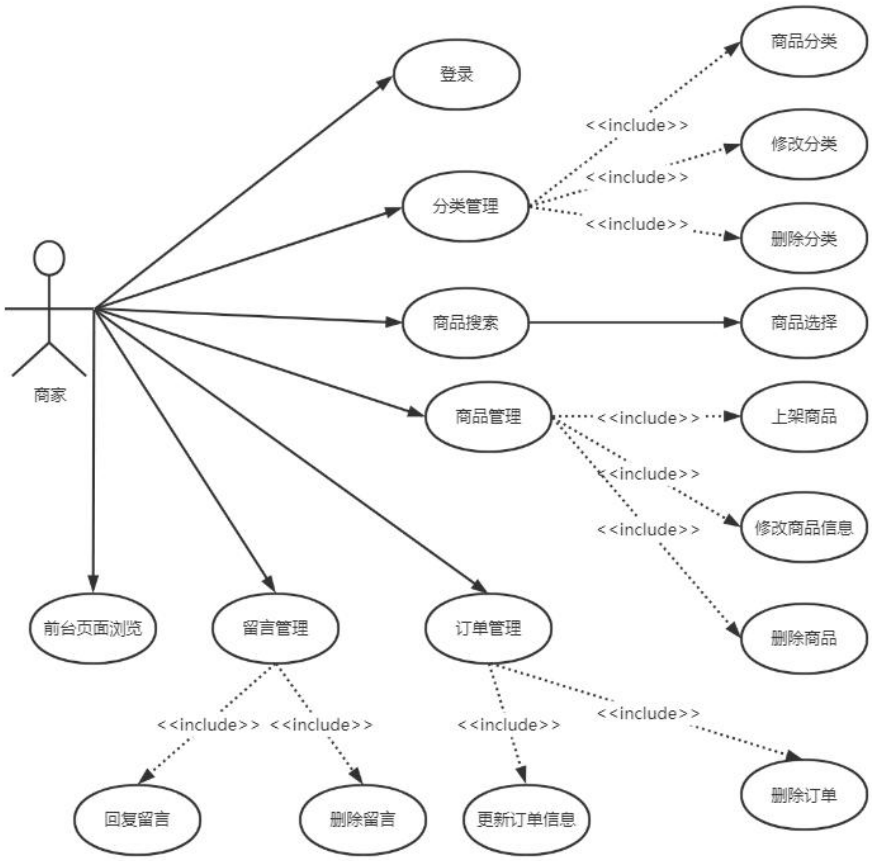

图 3-3 商家用例图

3.1.2 功能模块

哈哈商城功能模块在开发时优先级以及简要描述如下表 3-1 所示。

表 3-1：功能模块表

序号	功能名称	功能需求标识	优先级	简要描述
1	注册配置中心	L1	高	将微服务注册到注册中心，然后集中管理微服务的配置信息。
2	用户登录	L2	高	用户输入正确的用户名和密码即可登录系统。
3	展示商品列表	L3	高	在主界面显示各商品基本信息。
4	查找商品	L4	高	用户可查找商品，并显示热门搜索和相关内容。
5	展示商品详情	L5	中	点击商品可查看详情，显示店铺信息和商品的详细信息。
6	购买商品	L6	高	对商品进行下单和支付。

7	查看订单	L7	中	购买后可以查看个人订单。
8	评价与删除订单	L8	中	可对订单进行删除和评价。
9	检索服务	L9	中	商品的检索 ElasticSearch。
10	意见反馈	L10	中	用户可以通过此功能将意见反馈给该系统负责人。
11	帮助中心	L11	中	常见问题以及账号引导。
12	网站信息展示	L12	中	网站详情信息展示和网站公告信息展示。

3.2 非功能需求分析

3.2.1 数据处理能力

交易系统可支持 1000 个终端数量，支持 100 个用户并行操作，商家定期发布低价商品活动，引入大量消费者在短时间内进行商品秒杀。商品活动引入的瞬时流量往往非常大，考验系统对峰值流量的控制，所以秒杀系统需要进行独立部署，防止秒杀功能服务模块崩溃进而导致整个交易系统宕机。

3.2.2 时间特性要求

(1) 响应时间：一般操作的响应时间应在 1~2 秒内。

(2) 数据转换传送时间：主要是网页渲染的时间，当用户点击各个网页时，整个网页渲染完成的时间控制在 3 秒内。

(3) 检索的处理时间：用户在搜索框进行检索操作，输入内容点击检索后显示商品的时间应控制在 1-2 秒内。

(4) 适应性：作为基于微服务架构的交易系统应当保证系统能够独立运行。系统采用前后端分离的开发方式，便于日后发展出移动端的应用，更具灵活性。

(5) 灵活性：尽可能全面地考虑用户使用感受，考虑到秒杀功能在使用过程中可能产生的问题以及用户的使用需求，当用户需求发生变化时，通过修改接口实现类的方式，快速修改功能业务，提升用户使用感受。在 100 个并发用户的高峰期，交易系统基本功能，处理能力至少达到 10TPS

3.2.3 系统属性要求

基于微服务分布式交易系统，不光需要本身功能完善，同时对交易系统的安全性、可靠性、可用性、响应时间、可扩展性等方面都有较高的要求。

(1) 系统安全性：交易系统是直接与用户资金挂钩的，因此系统需要保证足

够的安全性，防止由系统漏洞带来的经济损失。如今抢购软件盛行，用户的每次访问都是对系统资源的访问，因此为了避免网站的恶意爬取，需要引入网关相关组件进行流量控制，黑白名单等功能，对于涉及到支付交易业务的接口调用，需要进行特殊的权限鉴定。同时采用 MD5（Message-Digest Algorithm 5），信息摘要算法)对密码进行盐值加密，防止用户密码的泄漏，尽可能地将交易系统所面临的风险降到最低。

(2) 系统可靠性：交易系统中最基础的部分是订单支付方面的交互，当系统在服务器运营中，为了保证系统在大数据处理层面上的安全性，在交易系统瞬间涌入大访问量的多并发环境中必须依靠消息队列作为分布式事务方案从而确保分布式数据处理的稳定性，并且提前进行热点数据缓存，以确保客户的迅速反馈，防止因客户端在宕机后迅速重启而造成的数据损失，确保数据的准确性。

(3) 系统可用性：为了保证系统可以全天候不间断地进行业务，所以要最大程度地减少服务器崩溃概率，因此关于服务器的可用性，可以采用设置服务器的熔断策略以及设置主从服务器，在服务器的压力剧增时设置降级页面，从而及时释放服务器资源，以确保核心服务的正常工作等手段，让交易系统能够在 7*24 小时全天候的稳定运行，在任何时刻都能提供良好的交易环境。

(4) 系统平均响应时间：系统的性能主要体现在用户的点击一次发布申请后系统的平均响应时间，对网上交易平台来说，用户浏览一次的平均等待时间在较大程度上会决定用户是否愿意最终消费，所以对系统性能要求也比较高，用户在消费前后都必须进行大量的访问和检查，为改善客户体验，系统必须确保所有用户的申请信息都能在较短的时间内反馈结果，所以通过 Nginx 服务器对用户请求进行动静分离，当服务器面对大量静态请求时，可以直接抛给 Nginx 处理，由 Tomcat 专门处理大量动态请求，从而降低了服务器压力，提高服务器性能，保证系统平均响在 1 秒钟之内，最长响应时间控制在 5 秒钟左右

(5) 系统可扩展性：交易系统平台在开发时需要注重服务和业务的可扩展性，应当对系统模块进行合理的规划，使得系统能更加方便地进行新业务模块的开发，在需要实现新功能时尽可能避免代码的大幅度重构。

第 4 章 系统设计

4.1 总体设计

微服务架构是一个分布式系统，Spring Could 是分布式微服结构的一站式解决方案，提供一套简单易用的编程模型，可以在 Spring Boot 的基础上轻松实现微服务系统的构建，Spring Cloud Alibaba 是 Spring Cloud 的第二代实现，主要由 Nacos、Sentinel、Seata 等组件组成。因此基于微服务的分布式交易系统主要依托于 Spring Could Alibaba 所提供的服务治理、服务网关。智能路由、负载均衡、分布式消息队列、服务配置中心管理等解决方案[13]。

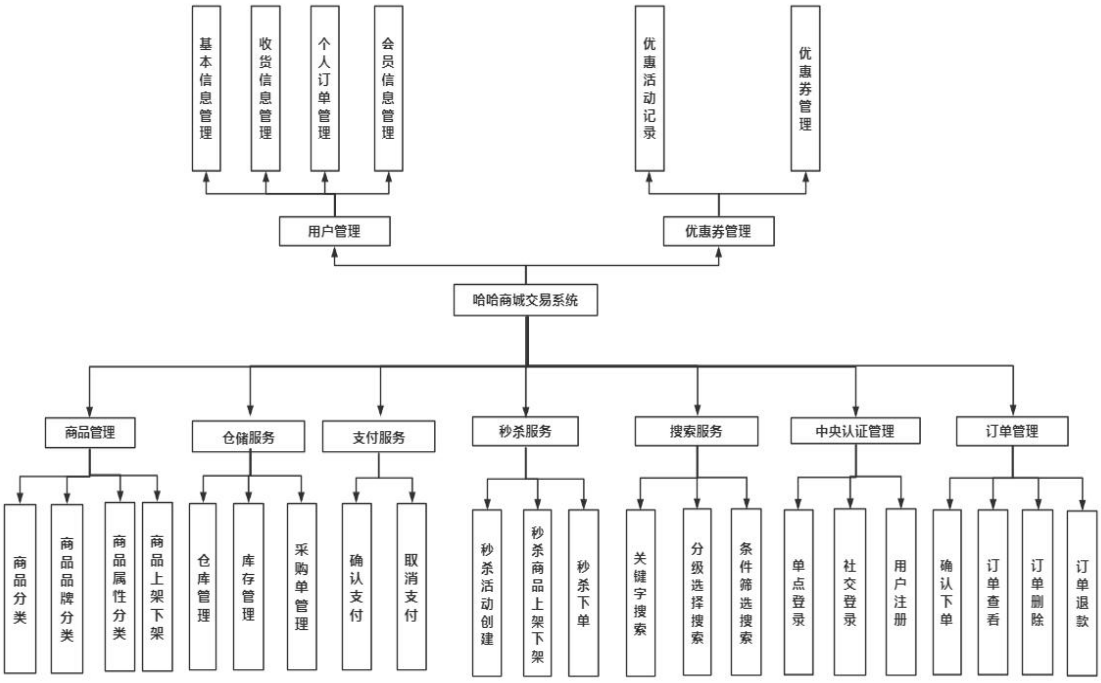

图 4-1 系统功能模块图

根据系统的需求分析，将基于微服务的分布式交易系统分为 9 个微服务模块如图 4-1 所示。有用户管理模块、优惠券管理模块、商品管理模块、仓储服务模块、支付服务模块、秒杀服务模块、搜索服务模块、中央认证管理模块、订单管理模块，订单模块和支付模块以及秒杀模块是交易系统的核心，在模块设计时需要考虑到核心业务的具体实现，以及访问负载能力的大小。其中秒杀模块对性能的要求最为突出，秒杀模块的性能直接影响到秒杀活动能否顺利进行，对秒杀活动中的成交量有重要影响，通常需要考虑到在秒杀活动开始期间，会有大量的用户浏览网页已经点击商品发出请求，因而服务器通常需要在短时间内处理大量的用户请求，并且给予响应，秒杀模块不但需要考虑到高并发带来的多线程处理问

题，而且还需要考虑到秒杀请求失败时库存系统的数据一致性问题，成功则将请求录入系统，失败则进行事务回滚并且撤销相关数据的修改，保证系统的可靠性。哈哈商城在后端代码中的微服务功能模块对应的名称以及组织结构如下。

```
├─── hahamall-auth-server -- 中央认证中心（社交登录、单点登录）
├─── hahamall-common -- 工具类及通用代码、注册中心、公共依赖
├─── hahamall-coupon -- 商品优惠券管理服务
├─── hahamall-gateway -- 统一配置网关
├─── hahamall-order -- 订单管理服务
├─── hahamall-product -- 商品管理服务
├─── hahamall-search -- 商品搜索服务
├─── hahamall-third-party -- 第三方服务（包含中央认证管理）
├─── hahamall-ware -- 商品仓储服务
├─── hahamall-member -- 用户管理服务
├─── hahamall-seckill-- 用户秒杀服务
├─── renren-fast -- 人人开源项目后台管理框架
└─── renren-generator -- 人人开源项目的代码生成器
```

4.2 数据库设计

4.2.1 数据库设计

基于微服务的分布式交易系统中数据存在大量一对一，一对多，多对多的关系，因此本系统的数据库采用了开源免费，并且性能优良的 Mysql 数据库作为系统的数据库管理系统，满足系统数据的持久化保存。由于交易系统采用的是微服务架构本次涉及到的表结构总共 52 张，因此系统数据库设计需要根据功能模块进行如下的划分：管理服务数据库：hahamall_admin、订单服务数据库：hahamall_oms、营销服务数据库：hahamall_sms、仓储服务数据库：hahamall_wms、用户服务数据库：hahamall_ums、商品服务数据库：hahamall_pms，交易系统数据库 ER 图如下图 4-2 所示。

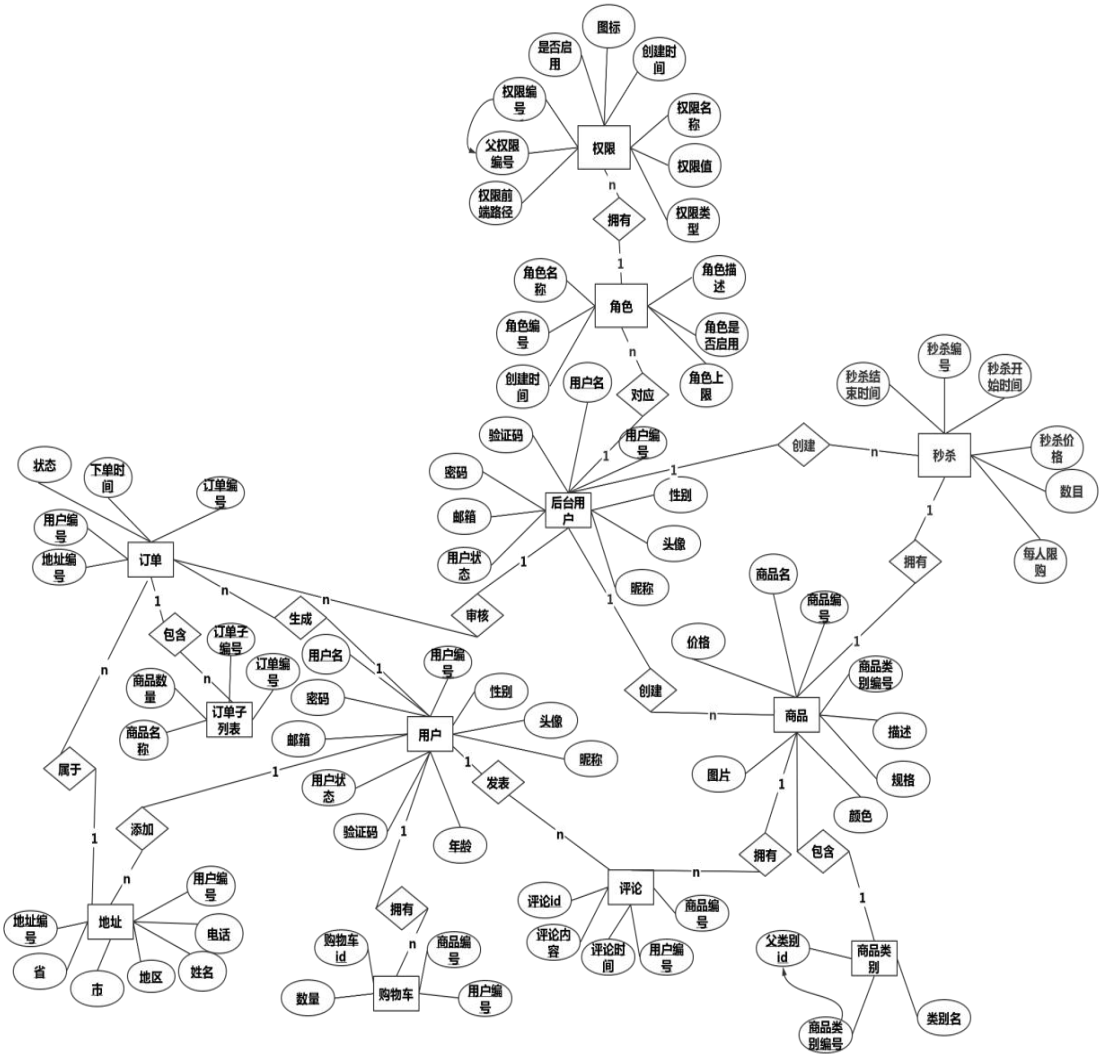

图 4-2 系统数据库 E-R 模型图

4.2.2 数据表设计

由于分布式交易系统涉及的业务功能以及对应的数据库表众多，因此表的结构设计主要由字段名、数据类型、字段长度、小数点数量、字段是否为 null、是否为主键、注释构成。每张表的字段名需要见名知意，数据类型是根据字段含义进行匹配的比如说字段是 id，number 等数字类型就用 int、string，文字类型的就用 varchar、时间类型的字段使用 datetime。通过字段长度的设计来保证字段在表中的最大长度，输入数据的规范；小数点数量确保交易平台支付业务的准确性；字段是否为 null 可灵活应用于处理业务需求，在对一些不能为空的重要字段（密码等）合理规范用户的输入。由于本交易系统采用的是微服务架构，数据库的位置不能保证一直在同一台机器上，无法保证外键约束，因此本系统中所有的表以及字段都不设置外键，从而减少联表操作并提高数据响应速度。

4.2.3 用户数据库表设计

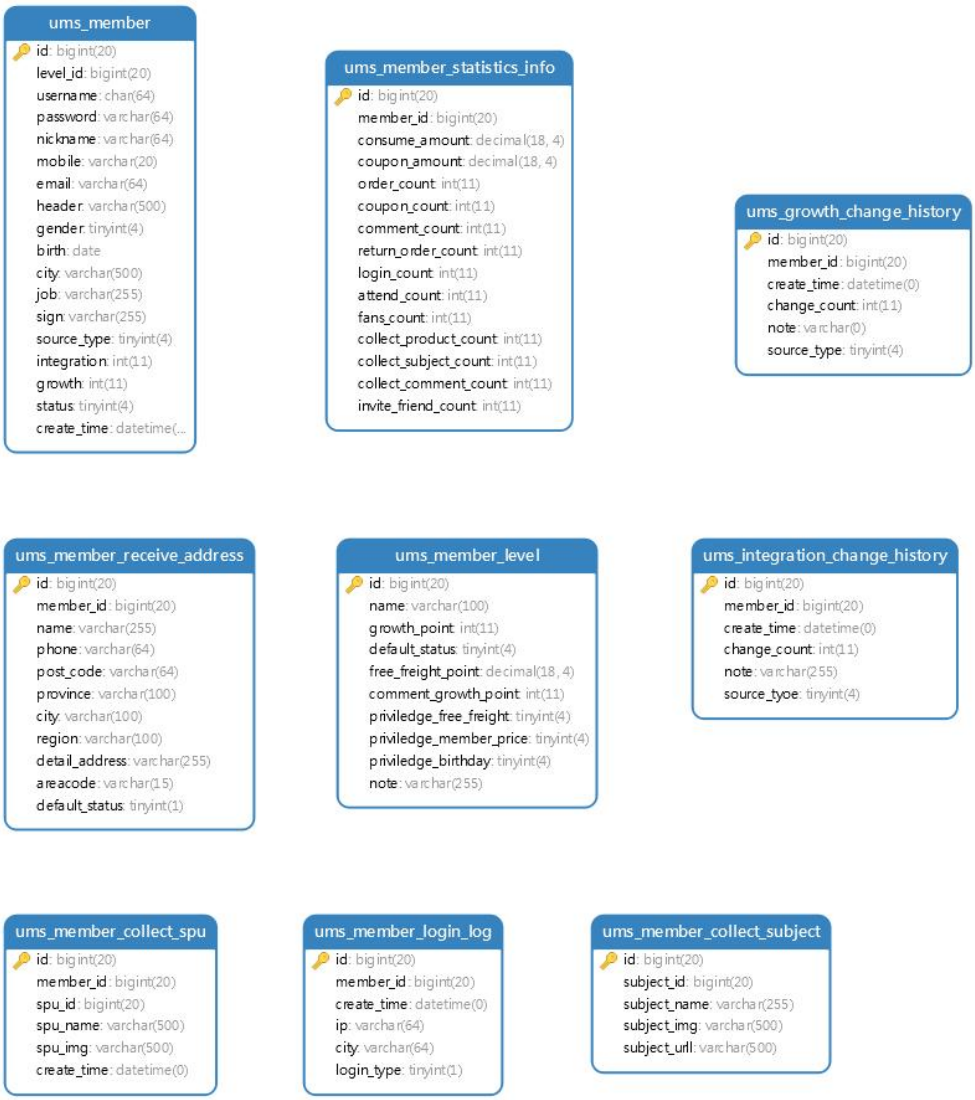

图 4-3 用户数据库 E-R 模型图

用户管理数据库（hahamall_ums）是交易系统中用户管理模块的关键，用户数据库 E-R 模型如图 4-3 所示，会员的基本信息存储在 ums_member 表中，涉及到将用户注册成会员时所需要填写的个人信息保存，以方便后续地址管理业务以及订单付款业务等业务信息的填写。会员表中各字段解释如表 4-1 所示。

表 4-1 会员基础信息表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	level_id	bigint(20)	会员等级 id

3	username	char(64)	用户名
4	password	varchar(64)	密码
5	nickname	varchar(64)	昵称
6	mobile	varchar(20)	手机号码
7	email	varchar(64)	邮箱
8	header	varchar(500)	头像
9	gender	tinyint(4)	性别
10	birth	date	生日
11	city	varchar(500)	所在城市
12	job	varchar(255)	职业
13	sign	varchar(255)	个性签名
14	source_type	tinyint(4)	用户来源
15	integration	int(11)	积分
16	growth	int(11)	成长值
17	status	tinyint(4)	启用状态
18	create_time	datetime	注册时间

会员统计信息存储在 ums_member_statistics_info 表中，通过这些会员的统计信息可以为每个会员进行定制化的商品展示以及优惠券发放，类似于现实社会中的熟客优惠制度，会员统计信息数据表中各字段解释如表 4-2 所示。

表 4-2 会员统计信息表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	member_id	bigint(20)	会员 id
3	consume_amount	decimal(18,4)	累计消费金额
4	coupon_amount	decimal(18,4)	累计优惠金额
5	order_count	int(11)	订单数量
6	coupon_count	int(11)	优惠券数量
7	comment_count	int(11)	评价数
8	return_order_count	int(11)	退货数量
9	login_count	int(11)	登录次数
10	attend_count	int(11)	关注数量
11	fans_count	int(11)	粉丝数量
12	collect_product_count	int(11)	收藏的商品数量
13	collect_subject_count	int(11)	收藏的专题活动数量
14	collect_comment_count	int(11)	收藏的评论数量
15	invite_friend_count	int(11)	邀请的朋友数量

会员成长值变化历史信息存储在 ums_growth_change_history 表中，通过这些字段信息可以清楚地追溯每个会员的消费记录，消费习惯，以及消费时间等数据，存储这些字段数据可以应用于以后的大数据分析场景，从而推荐给用户特定的需

求商品。会员成长值变换历史表中各字段解释如表 4-3 所示。

表 4-3 会员成长值变化表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	member_id	bigint(20)	member_id
3	create_time	datetime	create_time
4	change_count	int(11)	改变的值（正负计数）
5	note	varchar(0)	备注
6	source_type	tinyint(4)	积分来源

会员收货地址信息存储在 ums_member_receive_address 表中，用于会员的收货地址管理业务功能的实现，方便用户地址信息的存储。会员收货地址表中各字段解释如表 4-4 所示。

表 4-4 会员收货地址表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	member_id	bigint(20)	member_id
3	name	varchar(255)	收货人姓名
4	phone	varchar(64)	电话
5	post_code	varchar(64)	邮政编码
6	province	varchar(100)	省份/直辖市
7	city	varchar(100)	城市
8	region	varchar(100)	区
9	detail_address	varchar(255)	详细地址(街道)
10	areacode	varchar(15)	省市区代码
11	default_status	tinyint(1)	是否默认

会员等级信息存储在 ums_member_level 表中，涉及到会员是否享受的免运费，以及对于不同会员等级的商品价格，会员生日特权等功能的实现。会员等级表中各字段解释如表 4-5 所示。

表 4-5 会员等级表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	name	varchar(100)	等级名称
3	growth_point	int(11)	等级需要的成长值
4	default_status	tinyint(4)	是否为默认等级
5	free_freight_point	decimal(18,4)	免运费标准
6	comment_growth_point	int(11)	每次评价获取的成长值
7	priviledge_free_freight	tinyint(4)	是否有免邮特权

8	priviledge_member_price	tinyint(4)	是否有会员价格特权
9	priviledge_birthday	tinyint(4)	是否有生日特权
10	note	varchar(255)	备注

积分变化历史记录信息存储 ums_integration_change_history 表中，该表主要用于整合会员表和成长值积分变化同时记录积分变化的创建时间以及变化原因。会员积分变化记录表中各字段解释如表 4-6 所示。

表 4-6 会员积分变化表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	member_id	bigint(20)	member_id
3	create_time	datetime	create_time
4	change_count	int(11)	变化的值
5	note	varchar(255)	备注
6	source_type	tinyint(4)	来源

会员登录信息存储在 ums_member_login_log 表中，涉及会员的创建时间以及登录所在地还有登录方式的记录，通过该表中登录所在地以及登录 ip 可以进行系统的安全管控。会员登记记录表中各字段解释如表 4-7 所示。

表 4-7 会员登录记录表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	member_id	bigint(20)	member_id
3	create_time	datetime	创建时间
4	ip	varchar(64)	ip
5	city	varchar(64)	city
6	login_type	tinyint(1)	登录类型[1-web，2-app]

会员收藏的专题活动信息存储在 ums_member_collect_subject 表中，涉及会员的收藏活动的名称信息、活动的展览海报、活动的具体报名链接等信息。会员收藏的专题活动表中各字段解释如表 4-8 所示。

表 4-8 会员收藏专题互动表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	subject_id	bigint(20)	subject_id
3	subject_name	varchar(255)	subject_name
4	subject_img	varchar(500)	subject_img

5	subject_url1	varchar(500)	活动 url

4.2.4 订单数据库表设计

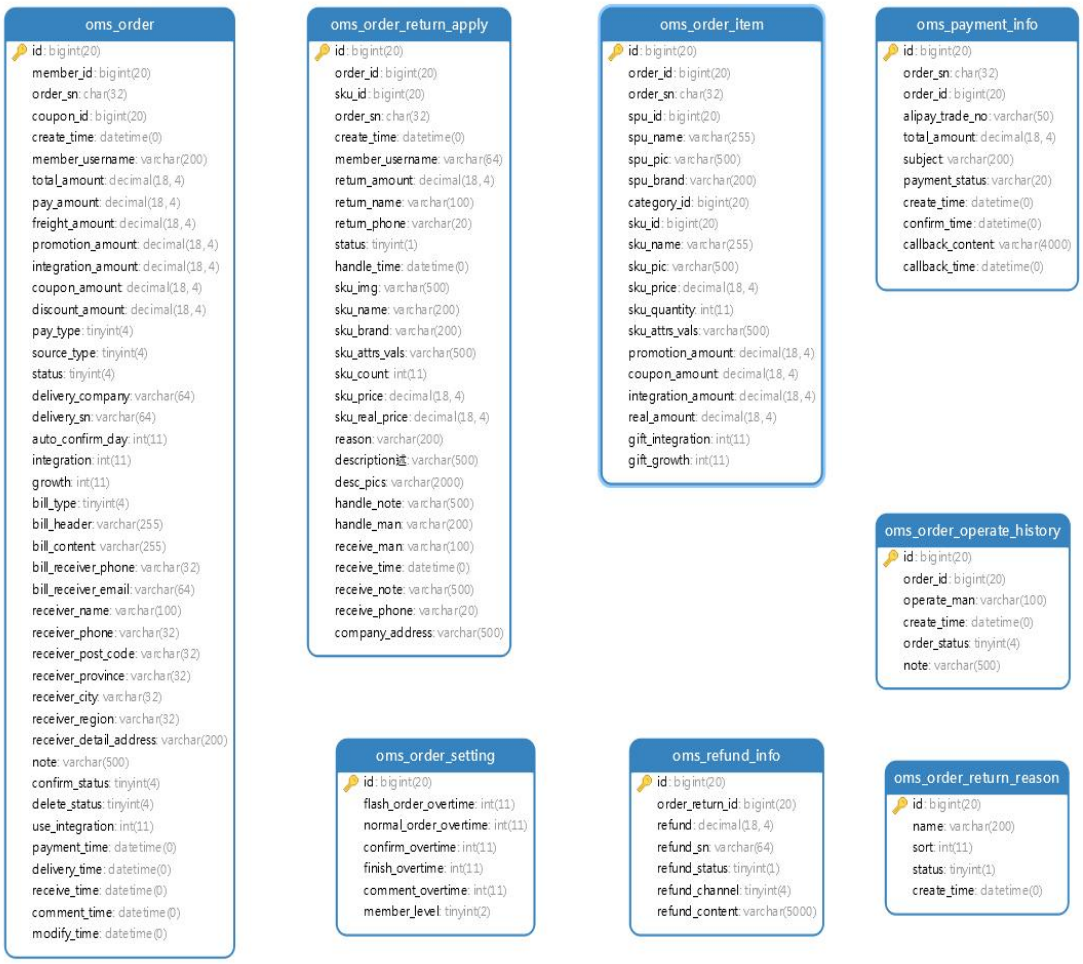

图 4-4 订单数据库 E-R 模型图

订单管理数据库（hahamall_oms）是交易系统中订单管理模块的关键，订单数据库 E-R 模型如图 4-4 所示。订单的支付和管理信息存储在 oms_order 表中，涉及到订单号、使用环境、订单总额、运费金额、促销金额、订单状态（待付款、待发货、已发货、已完成、已关闭、无效订单）、订单来源（PC 订单、app 订单）、支付方式（支付宝、微信、银联、货到付款）等表结构。订单信息各字段解释如表 4-9 所示。

表 4-9 订单统计信息表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	member_id	bigint(20)	member_id
3	order_sn	char(32)	订单号

4	coupon_id	bigint(20)	使用的优惠券
5	create_time	datetime	create_time
6	member_username	varchar(200)	用户名
7	total_amount	decimal(18,4)	订单总额
8	pay_amount	decimal(18,4)	应付总额
9	freight_amount	decimal(18,4)	运费金额
10	promotion_amount	decimal(18,4)	促销优化金额（促销价、满减、阶梯价）
11	integration_amount	decimal(18,4)	积分抵扣金额
12	coupon_amount	decimal(18,4)	优惠券抵扣金额
13	discount_amount	decimal(18,4)	后台调整订单使用的折扣金额
14	pay_type	tinyint(4)	支付方式【1->支付宝；2->微信；3->银联； 4->货到付款；】
15	source_type	tinyint(4)	订单来源[0->PC 订单；1->app 订单]
16	status	tinyint(4)	订单状态【0->待付款；1->待发货；2->已发货；3->已完成；4->已关闭；5->无效订单】
17	delivery_company	varchar(64)	物流公司(配送方式)
18	delivery_sn	varchar(64)	物流单号
19	auto_confirm_day	int(11)	自动确认时间（天）
20	integration	int(11)	可以获得的积分
21	growth	int(11)	可以获得的成长值
22	bill_type	tinyint(4)	发票类型[0->不开发票；1->电子发票；2->纸质发票]
23	bill_header	varchar(255)	发票抬头
24	bill_content	varchar(255)	发票内容
25	bill_receiver_phone	varchar(32)	收票人电话
26	bill_receiver_email	varchar(64)	收票人邮箱
27	receiver_name	varchar(100)	收货人姓名
28	receiver_phone	varchar(32)	收货人电话
29	receiver_post_code	varchar(32)	收货人邮编
30	receiver_province	varchar(32)	省份/直辖市
31	receiver_city	varchar(32)	城市
32	receiver_region	varchar(32)	区
33	receiver_detail_address	varchar(200)	详细地址
34	note	varchar(500)	订单备注
35	confirm_status	tinyint(4)	确认收货状态[0->未确认；1->已确认]

36	delete_status	tinyint(4)	删除状态【0->未删除；1->已删除】
37	use_integration	int(11)	下单时使用的积分
38	payment_time	datetime	支付时间
39	delivery_time	datetime	发货时间
40	receive_time	datetime	确认收货时间
41	comment_time	datetime	评价时间
42	modify_time	datetime	修改时间

订单项信息存储在 oms_order_item 表中，涉及到订单关联的商品信息（spu_id、spu_name、spu_pic）以及商品购买数量、商品促销金额、商品优惠券金额、商品优惠后金额、赠送积分、赠送成长值等表结构。订单信息单项表中各字段解释如表 4-10 所示。

表 4-10 订单商品信息表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	order_id	bigint(20)	order_id
3	order_sn	char(32)	order_sn
4	spu_id	bigint(20)	spu_id
5	spu_name	varchar(255)	spu_name
6	spu_pic	varchar(500)	spu_pic
7	spu_brand	varchar(200)	品牌
8	category_id	bigint(20)	商品分类 id
9	sku_id	bigint(20)	商品 sku 编号
10	sku_name	varchar(255)	商品 sku 名字
11	sku_pic	varchar(500)	商品 sku 图片
12	sku_price	decimal(18,4)	商品 sku 价格
13	sku_quantity	int(11)	商品购买的数量
14	sku_attrs_vals	varchar(500)	商品销售属性组合（JSON）
15	promotion_amount	decimal(18,4)	商品促销金额
16	coupon_amount	decimal(18,4)	优惠券优惠金额
17	integration_amount	decimal(18,4)	积分优惠金额
18	real_amount	decimal(18,4)	商品经过优惠后的金额
19	gift_integration	int(11)	赠送积分
20	gift_growth	int(11)	赠送成长值

订单操作历史记录存储在 oms_order_operate_history 表中，涉及到订单 id、操作人（用户、系统、后台管理员、操作时间、订单状态、备注）等表结构。通过这些表结构可以用于系统的出错处理，当订单出现问题时候能第一时间根据表结构查询后台管理员等相关负责人，以及操作的时间等信息。订单操作历史信息

表中各字段解释如表 4-11 所示。

表 4-11 订单操作历史信息表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	order_id	bigint(20)	订单 id
3	operate_man	varchar(100)	操作人[用户；系统；后台管理员]
4	create_time	datetime	操作时间
5	order_status	tinyint(4)	订单状态【0->待付款；1->待发货；2->已发货；3->已完成；4->已关闭；5->无效订单】
6	note	varchar(500)	备注

支付信息表存储在 oms_order_return_apply 表中，涉及到订单号（对外业务号）、支付宝交易流水号、支付总金额、交易内容、支付状态、等字段结构。主要应用于存储交易系统中支付业务所涉及的相关信息。支付信息表中各字段解释如表 4-12 所示。

表 4-12 支付信息表

序号	列名	数据类型	说明
1	id	bigint(20)	id
2	order_sn	char(32)	订单号（对外业务号）
3	order_id	bigint(20)	订单 id
4	alipay_trade_no	varchar(50)	支付宝交易流水号
5	total_amount	decimal(18,4)	支付总金额
6	subject	varchar(200)	交易内容
7	payment_status	varchar(20)	支付状态
8	create_time	datetime	创建时间
9	confirm_time	datetime	确认时间
10	callback_content	varchar(4000)	回调内容
11	callback_time	datetime	回调时间

4.3 详细设计

哈哈商城的详细设计模块流程如下图 4-5 所示。采用客户端与用户端分离的框架设计将各个服务模块之间通过 RabbitMQ 解耦合，以至于在面对短时间内超高请求并发时，服务器不会因为请求堵塞无法释放资源导致服务器宕机。

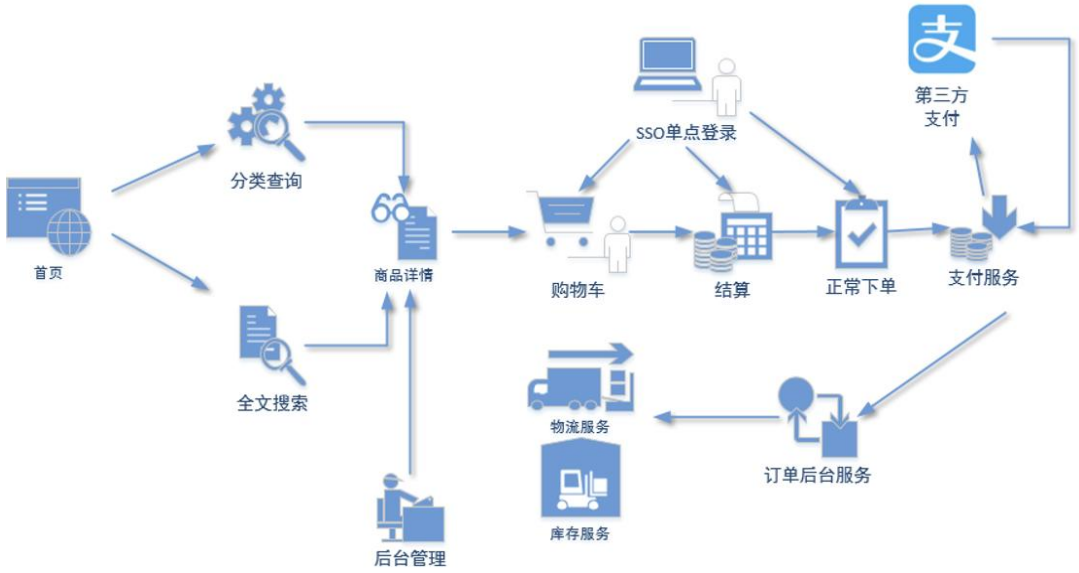

图 4-5 功能模块设计流程图

4.3.1 登录与注册

登录与注册模块主要是通过用户向发起请求，WEB 服务器检查 cookie 是否有 token，如 cookie 中不存在 token 则重定向到认证中心提示用户去注册登录，用户填写完登录信息并确定时通过认证中心核对用户信息，提示用户登录成功，下次用户发起登录请求时，请求中携带 token 则写入 cookie 中，方便下次用户登录时能以此作为认证信息进入首页。

实现思路：通过用户填写的用户名和密码进行与后台 MySQL 数据库进行核对，如果核对成功则将用户的信息存储到缓存数据库 Redis 中，Redis 中有改用户则是为登录状态，用 userId+当前用户登录 ip 地址+密钥的结构来生成 token，然后服务器重定向到用户请求初始地址，同时把 token 作为参数加入到请求中。

4.3.2 商品检索

商品检索模块在本次项目中主要应用于两个业务场景，一个是首页的分类，一个是搜索栏，由于哈哈商城是通过 Elasticsearch 开源搜索引擎实现的，因此需要思考那些字段需要分词？那些字段需要过滤？那些字段需要通过搜索显示出来？解决方案如下表 4-13 所示。

表 4-13 ES 分词字段表

分词的字段	sku 名称、sku 描述	分词、定义分词器
用于过滤的字段	平台属性、三级分类、价格	要索引
其他显示的字段	skuId 图片路径	不索引

SPU 标准化产品单元，是商品信息聚合的最小单元，是一个由不同销售属性组合而成的产品集合；SKU 库存量单位是库存数量统计的基本单元，也就是具体的产品，每个产品都对应一个 SKU，一组 SKU 产品被称为 SPU。后台实现查询数据的方法就是首先对想要查询的物品（SKU）进行分析，通过关键字、分类、属性值、分页等查询条件，然后通过编写 DSL 语句，再制作传入参数的类（这里由于底层数据使用流的形式因此类需要实现序列化接口），用 Java 语句字符串拼接等方式构造查询 DSL 语句，最终使用 JestClient 处理返回值。

4.3.3 购物车

购物车模块在哈哈商城中的应用体现在通过商品列表的添加按钮将商品加入到购物车，以及在商品详情页中添加商品到购物车。用户选择心仪的商品加入购物车时，后台代码实现的流程：首先检查用户的购物车中是否已经有商品，如果购物车中已经存在选择的商品，直接把对应的商品数量累加，同时更新 Redis 缓存，如果购物车中没有，则将商品的加入到购物车中，同时在 Redis 数据库中增加商品缓存信息。注意点：Redis 中取出来的数据要进行反序列化；并且由于 Redis 中的 hash 结构是无序的，因此使用时间戳进行排序；并且在将商品加入缓存的时候一定要设置失效时间防止出现缓存导致数据不一致的问题。

由于商品加入购物车时，用户可能存在登录和未登录两种状态，如果用户登录了，将登录前用户的 cookie 中保存的购物车信息与当前购物车信息合并（同时将 cookie 中的信息情况）到数据库中并进行刷新缓存如图 4-6 所示。

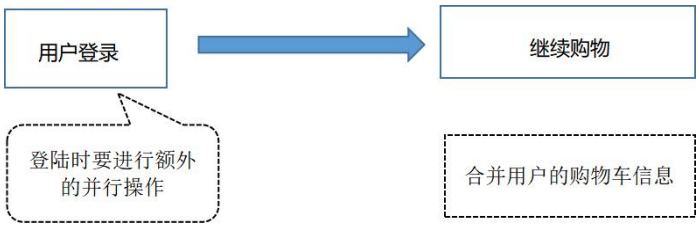

图 4-6 购物车登录合并图

4.3.4 订单系统

订单业务在整个系统中属于比较核心的业务，整个订单模块是由结算、下单、转跳支付服务、对接库存服务四个部分组成。结算页面包含用户地址列表、购物车中选择的商品列表、点击结算后是否更新数据库，因此在订单系统中引入 CartService，并增加购物车列表方法 getCartchekedList 传入用户 id，优先从缓存中取值，Redis 中存储的格式设置为“user:”+userId+“:cartChecked”，然后将其

反序列化成 CartInfo 对象并加入到列表后点击提交进行结算，创建订单表。提交订单时需要进行验价和验库存避免超买超卖问题，并且保存订单相关数据，保存以后将购物车中的商品删除，删除后重定向到支付页面。验价主要是防止用户不规范输入以及恶意修改订单价格，验库存一般通过 restful 接口查询商品是否有库存，一般的库存不是由交易系统本来管理，一般是另外一套商家的库存管理系统，因此这里采用调用第三方接口的方式调用库存系统。查询库存接口如表 4-14 所示。

表 4-14 请求库存接口表

接口	/hasStock
请求参数	skuId:商品 skuId num: 商品数量
请求方式	get
例	/hasStock?skuId=6666&num=6

4.3.5 秒杀功能

由于秒杀功能通常是面对短时间大量请求激增的业务场景，针对这一个特点哈哈商城做了限流（接口）+异步(线程池)+缓存（Redis）+独立部署等措施防止资源耗尽宕机，这个时候用户的请求通过 MySQL 数据库读数据以及写数据的方式不再适用，哈哈商城通过提前将数据序列化成 JSON 字符串后存入到 Redis 将数据提前进行缓存，商品所有的数据直接从缓存流中拿，直接在缓存中扣库存，并且为了防止超买超卖问题。基于 redis 秒杀方案如图 4-7 所示。

图 4-7 redis 秒杀方案图

通过设置分布式信号量作为库存扣减的凭证，用户请求得到信号量则进行信号量扣减，信号量为 0 时则直接返回没有库存，这样就能让服务器在面对超高并发请求时进行快速响应，并且通过 nginx 做好动静分离，保证秒杀功能和商品检索详情页的动态请求才发送给服务器，页面等请求直接通过 nginx 进行返回，使用 CDN 网络分担集群压力，同时通过在秒杀时候设置人工输入验证码的方式进行流量错峰，并且使用了 Sentinel 组件进行限流，如果流量超出限定则将服务进行熔断，使得服务器能保持正常运行，并且通过消息队列将秒杀请求保存到集群队列使用订阅－监听的方式让前台请求快速响应，后台服务根据接收到的消息稳定运行创建订单。缓存秒杀模型如图 4-8 所示。

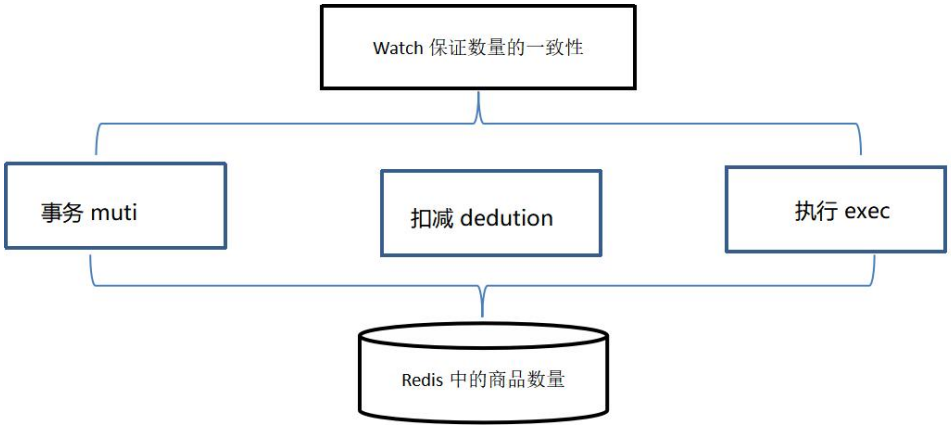

图 4-8 redis 缓存秒杀模型图

第 5 章 系统实现与测试

5.1 系统实现

5.1.1 登录与注册

传统服务器中为了保证登录信息在 web 服务之间传递往往采取 session 共享方式，但是由于微服务架构中各服务模块数据独立管理，不适合统一更新 session 数据，因此本项目采用了单点登录技术实现登录业务如图 5-1 所示。

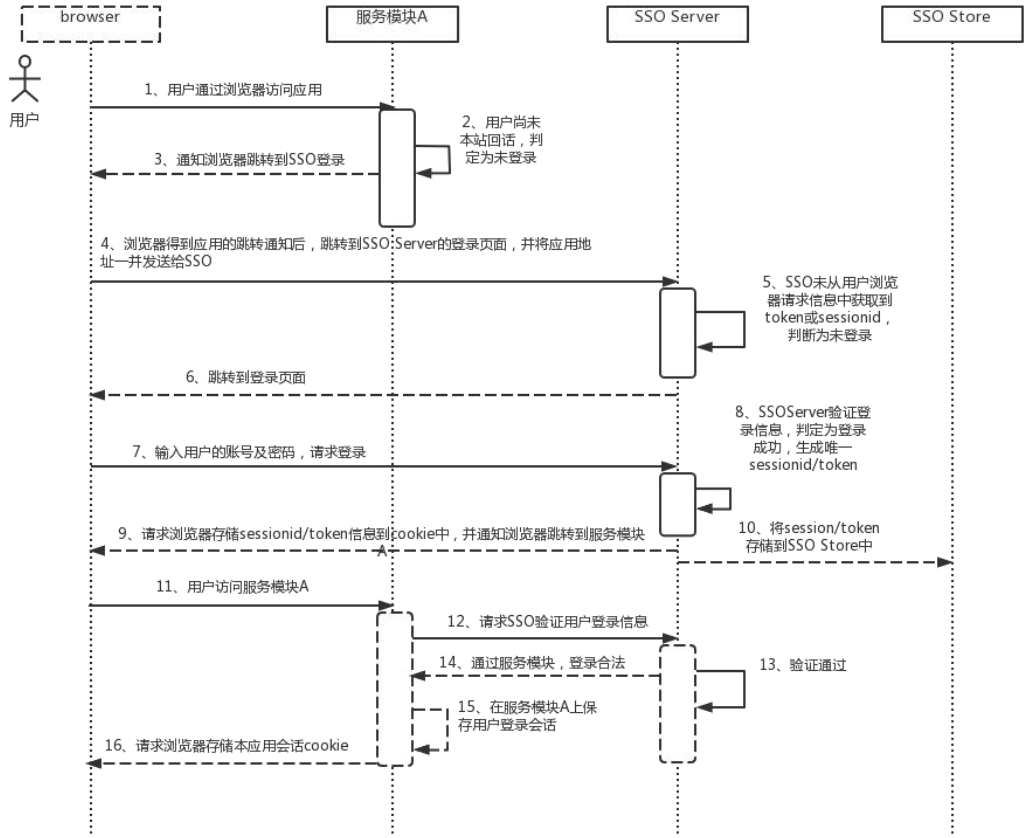

图 5-1 单点登录 UML 图

注册功能与登录功能的整合实现流程如下图 5-2 所示。首先根据用户请求的 cookie 中是否携带 token 判断用户是否在认证中心，若在认证中心则通过程序照常执行否则跳转到认证模块接口填写登录信息并验证，若登录校验成功则记录登录时间的相关数据到用户表中并生成 token 缓存用户信息。若用户点击注册则转跳到注册页面填写注册信息并校验。若用户忘记密码，则提供输入手机号码，发送短信验证码的方式让用户修改密码，与此同时更新数据库中用户数据。

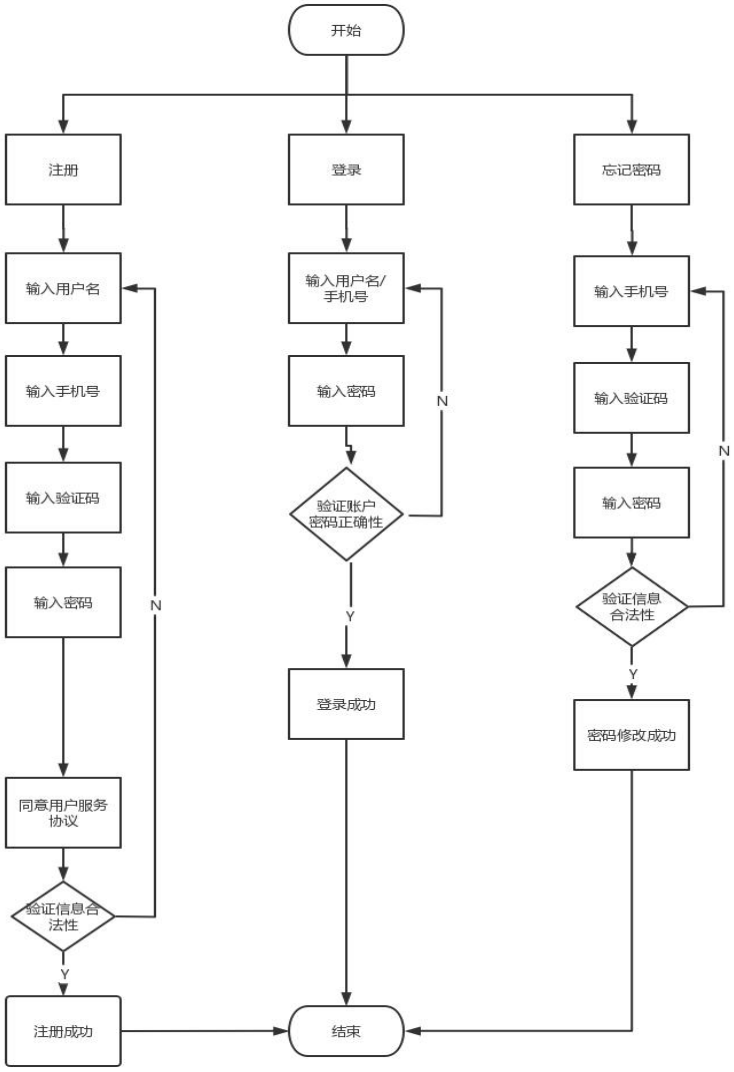

图 5-2 注册与登录流程图

注册功能涉及表单提交，因此采用 POST 的请求方式来设计/register 接口通过 JSR303 规范用户注册时填写的数据格式，通过 Map 将注册失败的错误属性与错误信息一一封装进行缓存。用户开始注册时将填写的数据以及验证码封装成 UserRegisterVo 类对象，然后将对象传入远程服务 hahamall-member 的 register 方法进行远程调用，如果成功就重定向到登录页面，失败则记录信息并重定向到注册页面。

```
@PostMapping("/register")
        public String register(@Valid UserRegisterVo vo,BindingResult result,
          RedirectAttributes redirectAttributes){
        if(result.hasErrors()){
            // 将错误属性与错误信息一一封装
            Map<String, String> errors = result.getFieldErrors()
              .stream().collect(Collectors.toMap
              (FieldError::getField, fieldError -> fieldError.getDefaultMessage()));
            // addFlashAttribute 这个数据只取一次
```

```
            redirectAttributes.addFlashAttribute("errors", errors);
            return "redirect:http://auth.hahamall.com/reg.html";
          }
        // 开始注册  调用远程服务
        // 1.校验验证码
            String code = vo.getCode();
            String redis_code =
             stringRedisTemplate.opsForValue().get(AuthServerConstant.
             SMS_CODE_CACHE_PREFIX + vo.getPhone());
             if(!StringUtils.isEmpty(redis_code)){
             // 验证码通过
             if(code.equals(redis_code.split("_")[0])){
             stringRedisTemplate.delete(AuthServerConstant.
             SMS_CODE_CACHE_PREFIX + vo.getPhone());
             // 调用远程服务进行注册
             R r = memberFeignService.register(vo);
             if(r.getCode() == 0){
                 // 成功
                 return "redirect:http://auth.hahamall.com/login.html";
             }else{
           Map<String, String> errors = new HashMap<>();
           errors.put("msg",r.getData("msg",newTypeReference<String>(){}));
           redirectAttributes.addFlashAttribute("errors",errors);
           return "redirect:http://auth.hahamall.com/reg.html";}
             }else{
           Map<String, String> errors = new HashMap<>();
           errors.put("code", "验证码错误");
           redirectAttributes.addFlashAttribute("errors", errors);
           return "redirect:http://auth.hahamall.com/reg.html"; }
         }else{
           Map<String, String> errors = new HashMap<>();
           errors.put("code", "验证码错误");
           redirectAttributes.addFlashAttribute("errors", errors);
            return "redirect:http://auth.hahamall.com/reg.html";
         }
       }
```

用户注册界面效果如下图 5-3 所示。

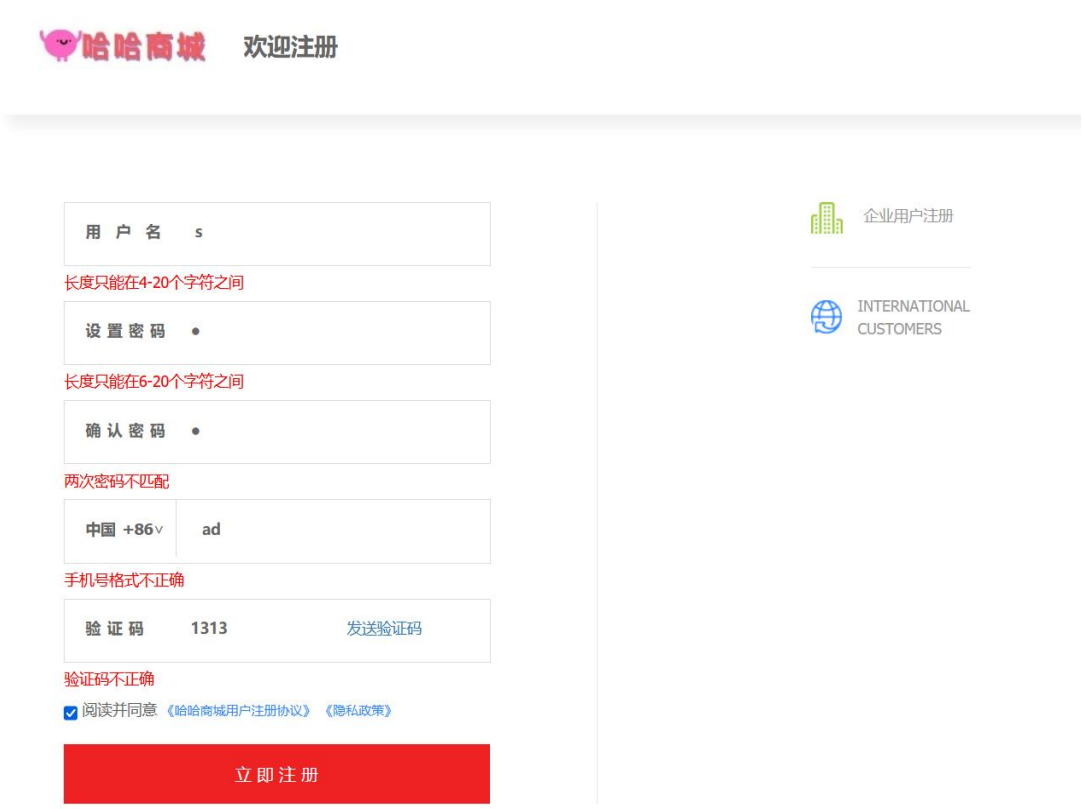

图 5-3 用户注册页面图

登录页面分为普通用户登录和管理员登录，在登录成功后，用户消息通过 SpringSession 保存在 Redis 数据库中，并且向浏览器添加包含用户消息的 Cookie，范围为主域名，用于以后在不同模块间跳转时都可以通过携带的 Cookie 用户信息去 redis 数据库中查到当前用户的 Cookie 进行认证。认证成功则跳转到业务对应页面，认证失败则重定向到用户登录页。用户登录界面效果如图 5-4 所示。管理员登录界面效果如图 5-5 所示。

图 5-4 用户登录页面图

图 5-5 管理员登录页面图

5.1.2 商品检索

商品检索功能在交易系统中使用频繁，并发量大，并且由于一个交易系统中的商品量往往非常多，因此使用传统的关系型数据库对索引遍历检索商品的方式已无法满足使用需求，因此引入 ElasticSearch 实现交易系统的商品检索业务。

（1）在检索界面，用户可以根据当前商品相关的销售属性、品牌来构造筛选条件，根据该条件下包含的属性类型来动态获取筛选结果。检索流程如下图 5-6 所示。

（2）首先通过对所需要查询的商品（数据源）结合查询语句进行分析(analysis)。将分析后得到关键字、分类、属性值、分页等构造查询条件进行查询解析(queryParser)。

（3）然后通过编写 DSL 语句进行搜索(search)，然后对搜索到的结果进行持久化存储（store）到索引(index)中。

（4）在 Java 层面中使用 JestClient 处理 DSL 语句并得到返回值。将返回值传入到 Elasticsearch 将相似的结果(similarty)进行聚合排序构造检索结果。

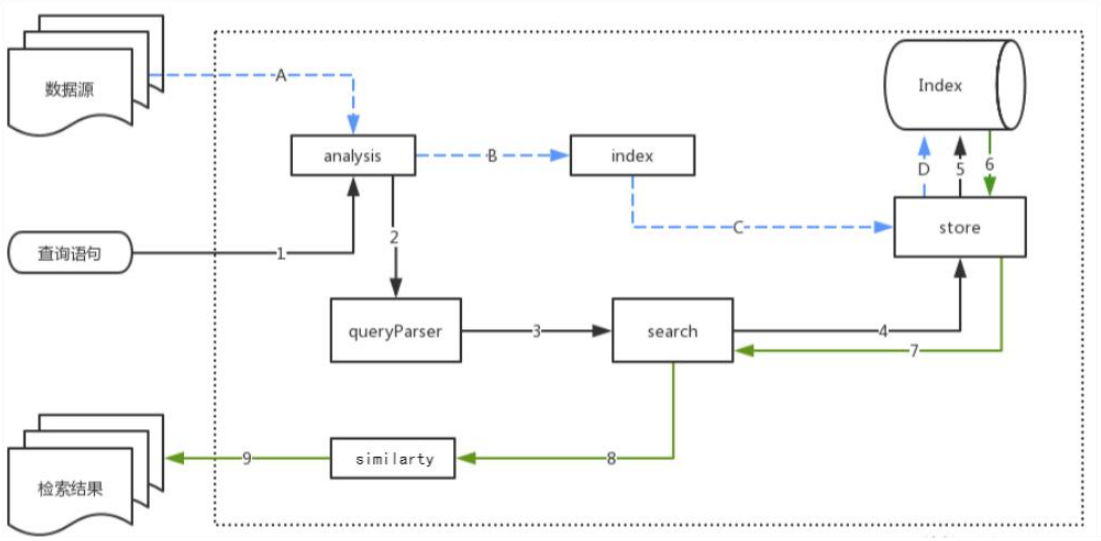

图 5-6 ElasticSearch 检索流程图

ElasticSearch 主要实现在 hahamall-search 服务模块中通过 buildSearchRequest 方法构建 DSL 语句查询数据，然后通过按照品牌 id，属性等条件模糊查询从而过滤数据构建查询语句 try-catch 语句块中引入 ElasticSearch 模块提供的客户端 restHighLeverlClient，通过查询语句获取数据 response，最后将 response 传入到 buildSearchResult 方法中构建 SearchResult 实体类保存 ElasticSearch 查询出的数据。

```
public SearchResult search(SearchParam Param) {
    SearchResult result = null;
    // 1.准备检索请求
    SearchRequest searchRequest = buildSearchRequest(Param);
    try {
        // 2.执行检索请求
        SearchResponse response = restHighLevelClient.
            search(searchRequest, MallElasticSearchConfig.COMMON_OPTIONS);
        // 3.分析响应数据
        result = buildSearchResult(response, Param);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}
```

页面实现效果如图 5-7 所示。

图 5-7 ElasticSearch 检索实现图

ElasticSearch 默认的分词器对英文比较好，但是对中文支持不太友好，在搜索业务中，需要考虑自定义词库的扩展，所以哈哈商城选择使用 ik 中文分词器使得 ElasticSearch 可以对中文词语进行分析搜索。ElasticSearch 分词优化的实现前不支持中文词的分析，汉语句子会被拆分为一个一个的字，无法进行有效分析，优化前效果如图 5-8 所示。

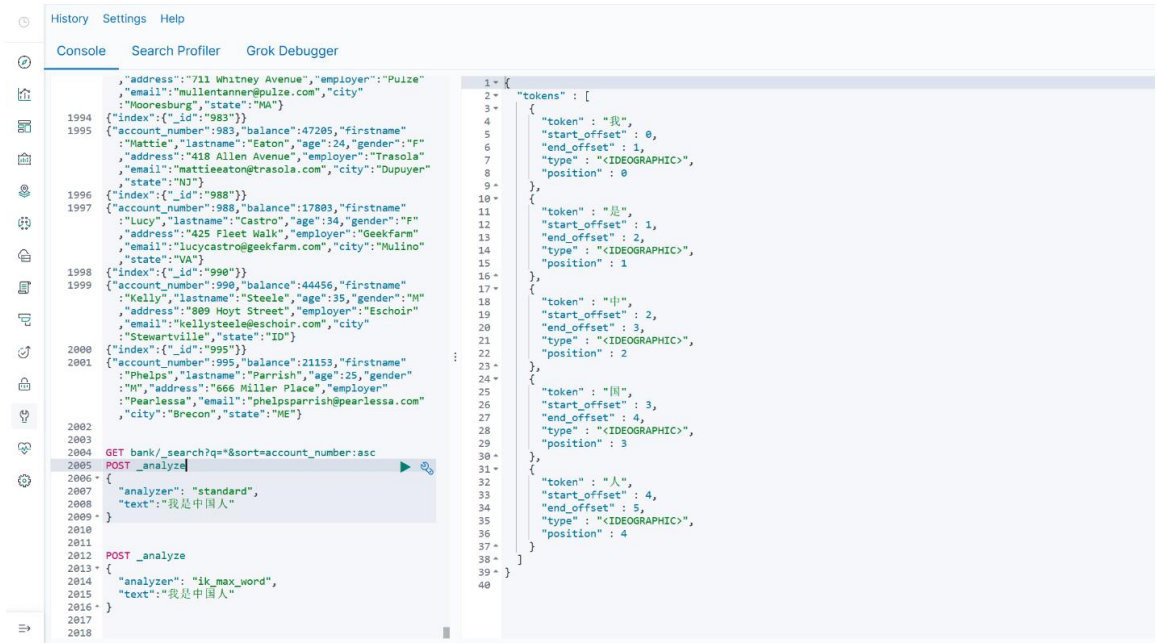

图 5-8 ElasticSearch 分词优化前显示图

优化后通过自定义词库，汉语句子可以被拆分为词语，短句的形式，对句子进行有效分析，优化后效果如图 5-9 所示。

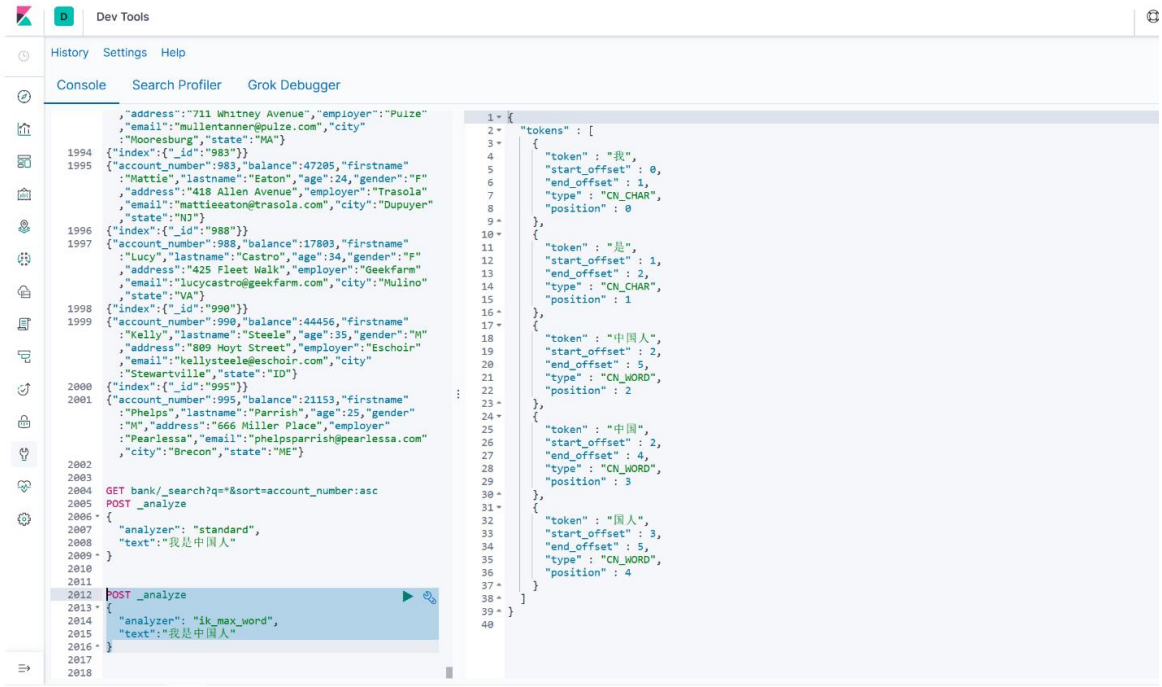

图 5-9 ElasticSearch 分词优化后效果图

5.1.3 购物车

购物车分为未登录状态和登陆状态，在未登陆时，购物车的数据会在 redis 中保存 30 天，在登录后，购物车数据会一直保存，并且会将购物车数据和之前未登录时加入到购物车的数据进行合并。主要使用 Cookie 来保存购物车数据，在第一次执行后服务器会返回一个保存当前临时用户购物车 ID 的 Cookie，在登陆后会通过携带的Cookie，查询对应的购物车数据并与登陆用户ID对应的 Redis 数据进行合并，再删除临时 Cookie 的数据。同时也实现购物车项的增加删除。购物车实现流程如图 5-10 所示。

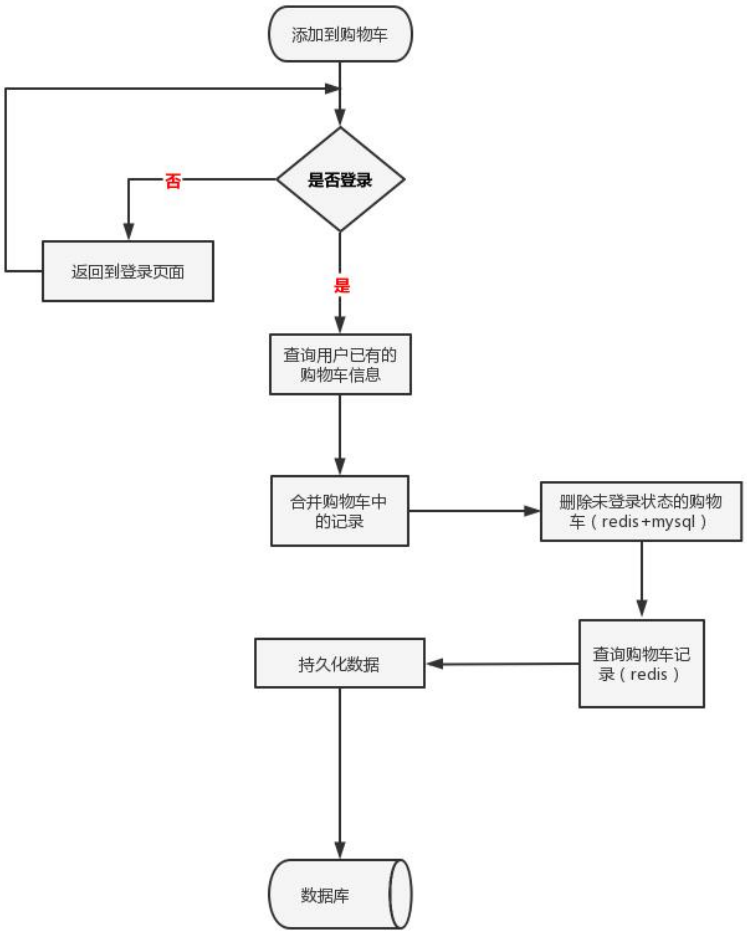

图 5-10 购物车实现流程图

添加商品到购物功能主要通过 addToCart 方法先查询购物车中是否已经有 skuId 对应的商品，已经有就增加该商品数目，更新 redis 中的商品数量缓存，如果没有商品就调用远程服商品服务 productFeignService 获取商品信息，通过 CompletableFuture 提供的异步编排的方式将商品加入到购物车并将商品信息加入到 redis 中进行缓存。

```
public CartItem addToCart(Long skuId, Integer num)
    throws ExecutionException, InterruptedException {
    BoundHashOperations<String, Object, Object> cartOps = getCartOps();
    String res = (String) cartOps.get(skuId.toString());
    if(StringUtils.isEmpty(res)){
    CartItem cartItem = new CartItem();
    // 异步编排
    CompletableFuture<Void> getSkuInfo = CompletableFuture.runAsync(() -> {
        // 1. 远程查询当前要添加的商品的信息
        R skuInfo = productFeignService.SkuInfo(skuId);
        SkuInfoVo sku = skuInfo.getData("skuInfo", new TypeReference<SkuInfoVo>()
          {});
```

```
        // 2. 添加新商品到购物车
        cartItem.setCount(num);
        cartItem.setCheck(true);
        cartItem.setImage(sku.getSkuDefaultImg());
        cartItem.setPrice(sku.getPrice());
        cartItem.setTitle(sku.getSkuTitle());
        cartItem.setSkuId(skuId); }, executor);
    // 3. 远程查询 sku 组合信息
    CompletableFuture<Void> getSkuSaleAttrValues =
      CompletableFuture.runAsync(() -> {
    List<String> values = productFeignService.getSkuSaleAttrValues(skuId);
    cartItem.setSkuAttr(values);
    }, executor);
    CompletableFuture.allOf(getSkuInfo, getSkuSaleAttrValues).get();
    cartOps.put(skuId.toString(), JSON.toJSONString(cartItem));
    return cartItem;
    }else{
    CartItem cartItem = JSON.parseObject(res, CartItem.class);
    cartItem.setCount(cartItem.getCount() + num);
    cartOps.put(skuId.toString(), JSON.toJSONString(cartItem));
    return cartItem;
        }
    }
```

购物车页面效果如图 5-11 所示。

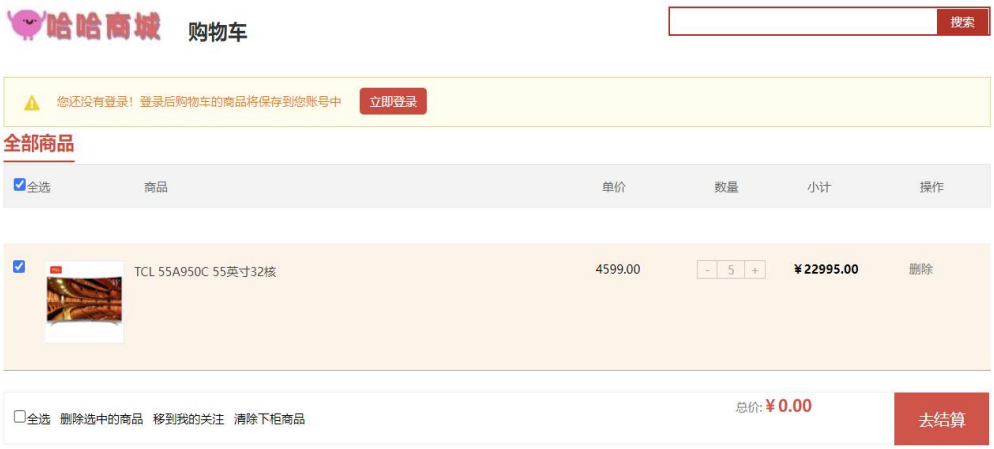

图 5-11 购物车页面效果图

5.1.4 订单系统

订单系统在整个交易平台中处于核心位置，也是比较重要的一块业务，整个模块由结算，下单，对接支付服务，对接库存管理系统四个部分组成。订单系统

中使用了 RabbitMQ 的延时队列+本地事务 来解决分布式事务。同时使用手动确认来保证消息可靠性。

由于下单过程会涉及到很多模块，并且其中数据的准确性也是要求最高的，任意一个数据出错都会对整块数据造成影响。所以应该使用事务对整个过程进行控制，但是因为数据的变动涉及到多个模块，传统的单体事务并不能满足这个场景，所以就引入了分布式事务，但是由于强一致性的分布式事务解决方案会使得系统的性能下降，所以本项目使用的是弱一致性+最终一致性。在调用远程方法锁库存时为了防止发生异常添加了事务，并且发送解库存消息给延时队列，保证在之后主业务发生异常之前锁的库存也可以正常解锁，实现数据最终一致性。订单系统业务流程如图 5-12 所示。

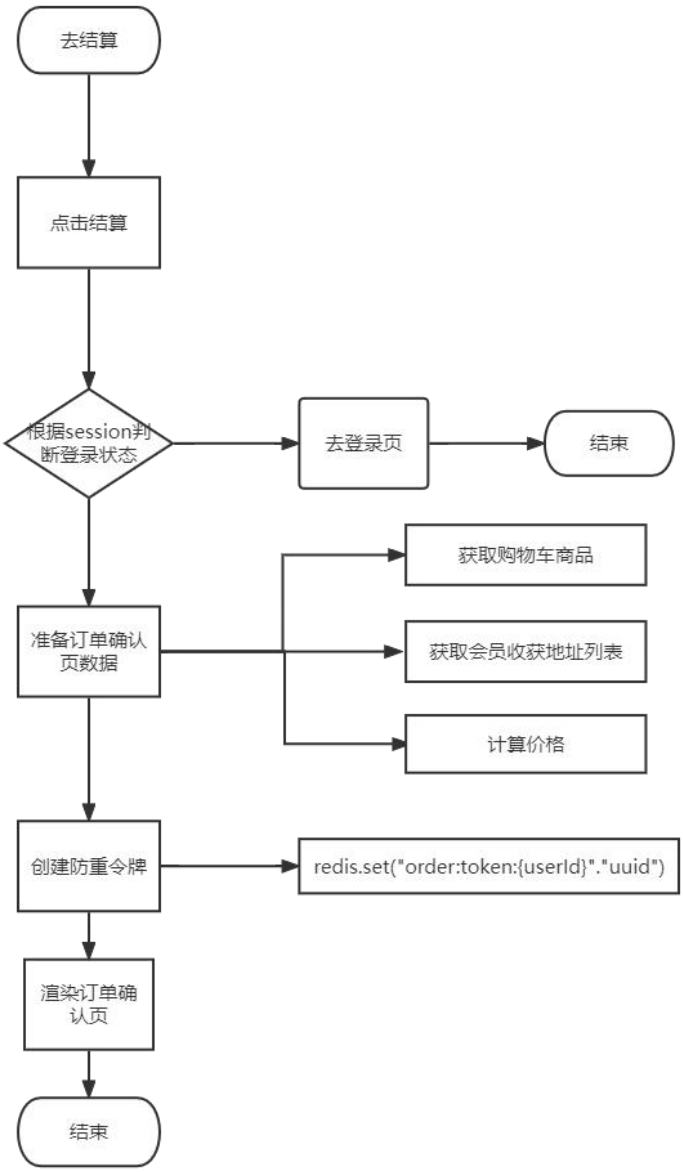

图 5-12 订单系统流程图

订单生成的具体实现就是在订单模块的事务方法中进行订单、订单项数据的添加，并调用远程接口进行锁库存操作。然后根据这次远程调用的结果选择订单的状态，消息队列设计流程如图 5-13 所示。

(1) 发送一个释放库存、修改订单状态的消息给延时队列，并且调用远程的锁库存操作也会发送一个释放库存的消息给延时队列。一段时间内用户未支付订单库存会自动解锁。

(2) 如果调用的远程锁库操作正常执行，但是本地事务方法发生了异常，那么远程未回滚库存也会在一段时间后也会自动解锁。

(3) 使用 Redis 存储订单令牌，以此实现支付订单接口的幂等性。通过 lua 脚本来验证和删除令牌（在进入结算页面时会在 redis 存入一个数据作为令牌，随后生成订单时则会通过脚本来检查这个数据是否存在，如果存在就将其删除并返回 1，否则返回 0，因为需要查询和删除的通过是原子性所以使用脚本来实现），成功后执行业务代码。

(4) 使用支付宝沙箱环境模拟支付过程。账号：xiongqiang@sandbox.com。密码和支付密码都是 2018010102171。在支付成功后修改订单状态并保存支付流水记录。

(5) 为了防止在支付界面一直等待直到后台队列将订单取消再完成支付，导致数据错乱，在消息取消订单时同时会关闭支付界面

(6) 为了防止由于消息积压、网络波动等原因导致解库存的消息先于取消订单的消息被接收执行，解库存失败（解库存会先判断订单状态是否是取消状态再进行解库存）所以在取消订单中也会发送一条解库存的消息。并且为了保证解库存的幂等性，在库存表中添加了库存工作单记录用于标记当前订单商品的库存是否已经解锁了。

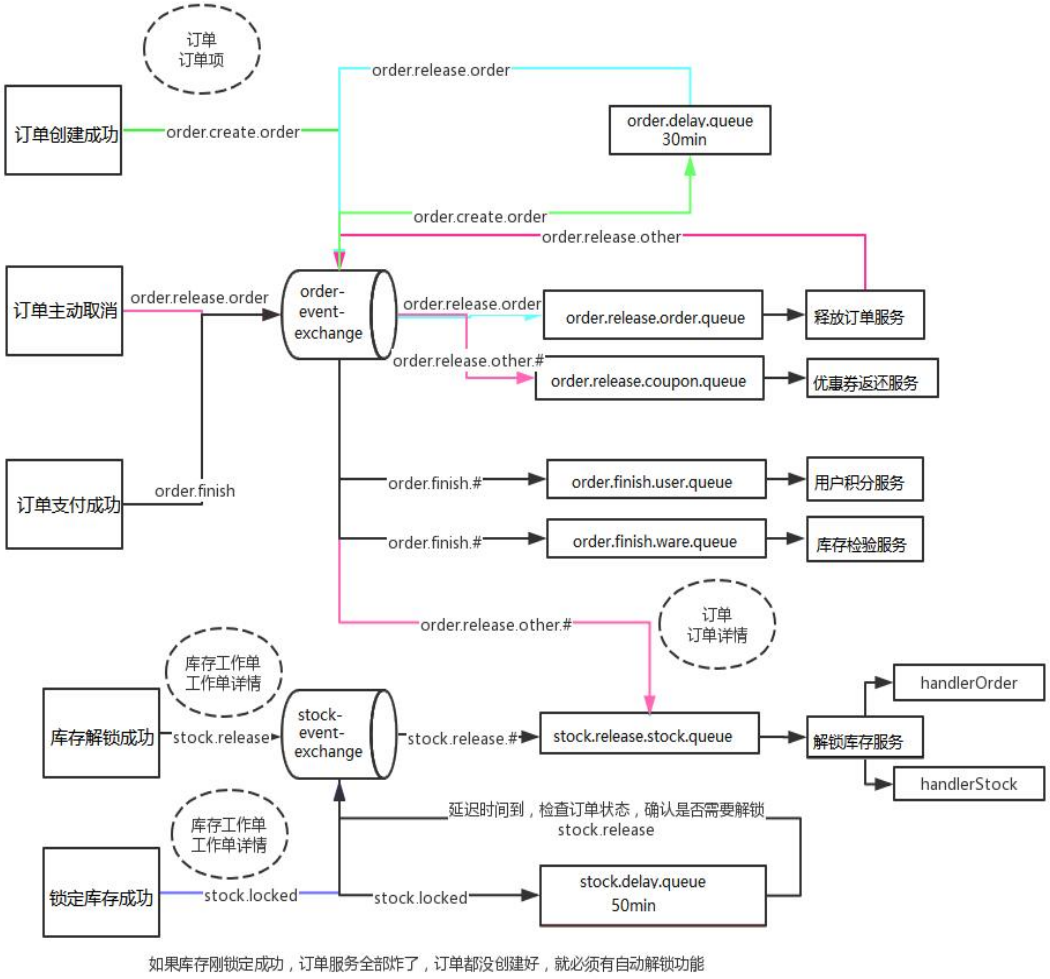

图 5-13 订单系统中 RabbitMQ 部署流程图

下单功能通过订单服务模块 orderService 中的 submitOrder 方法提交订单信息，然后根据响应结果 responseVo 中的 code 属性判断下单状态，设置对应的提示信息。

```
@PostMapping("/submitOrder")
public String submitOrder(OrderSubmitVo submitVo, Model model,
  RedirectAttributes redirectAttributes){
    try {
        SubmitOrderResponseVo responseVo = orderService.submitOrder(submitVo);
        // 下单失败回到订单重新确认订单信息
        if(responseVo.getCode() == 0){
            // 下单成功取支付选项
            model.addAttribute("submitOrderResp", responseVo);
            return "pay";
        }else{
            String msg = "下单失败";
            switch (responseVo.getCode()){
                case 1: msg += "订单信息过期,请刷新在提交";break;
```

```
                    case 2: msg += "订单商品价格发送变化,请确认后再次提交";break;
                    case 3: msg += "商品库存不足";break;
                }
                redirectAttributes.addFlashAttribute("msg", msg);
                return "redirect:http://order.hahamall.com/toTrade";
            }
        } catch (Exception e) {
            if (e instanceof NotStockException){
                String message = e.getMessage();
                redirectAttributes.addFlashAttribute("msg", message);
            }
            return "redirect:http://order.hahamall.com/toTrade";
        }
    }
```

购物车模块中的提交订单业务如下，通过 orderService.checkTradeNo 方法进行时间校验，如果结算页面失效则需要重新结算，否则进行验证价格，如果价格发生改变，就转跳到购物车页面重新结算。

```
@RequestMapping(value = "submitOrder" ,method = RequestMethod.POST)
@LoginRequire(debugUserId = "8")
public String submitOrder(OrderInfo orderInfo , HttpServletRequest request){
    String userId = (String) request.getAttribute("userId");
    String tradeNo= request.getParameter("tradeNo");
    Boolean hasTradeNo=orderService.checkTradeNo(userId,tradeNo);
    if(!hasTradeNo){
        request.setAttribute("errMsg","结算页面过期或已失效，请重新结算。");
        return "tradeFail"; }
    List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();
    for (OrderDetail orderDetail : orderDetailList) {
        SkuInfo skuInfo = manageService.getSkuInfo(orderDetail.getSkuId());
        //验价
        if(!orderDetail.getOrderPrice().equals(skuInfo.getPrice())){
            cartService.loadCartCache(userId);
            request.setAttribute("errMsg","商品["+skuInfo.getSkuName()+"]价格已发生变
            化，请在购物车页面重新进行结算");
            return "tradeFail";
        }
        orderDetail.setImgUrl(skuInfo.getSkuDefaultImg());
        orderDetail.setSkuName(skuInfo.getSkuName());
    }
    orderInfo.setUserId(userId);
    orderInfo.sumTotalAmount();
```

```
        orderService.saveOrder(orderInfo);
        List<String> skuIdList =new ArrayList<>();
        for (OrderDetail orderDetail : orderDetailList) {
            skuIdList.add(orderDetail.getSkuId());
        }
        cartService.delCartCheckedList(userId,skuIdList);
        orderService.delTradeNo(userId);
        return "redirect://payment.hahamall.com/index";
    }
```

订单系统页面实现效果如图 5-14 所示。

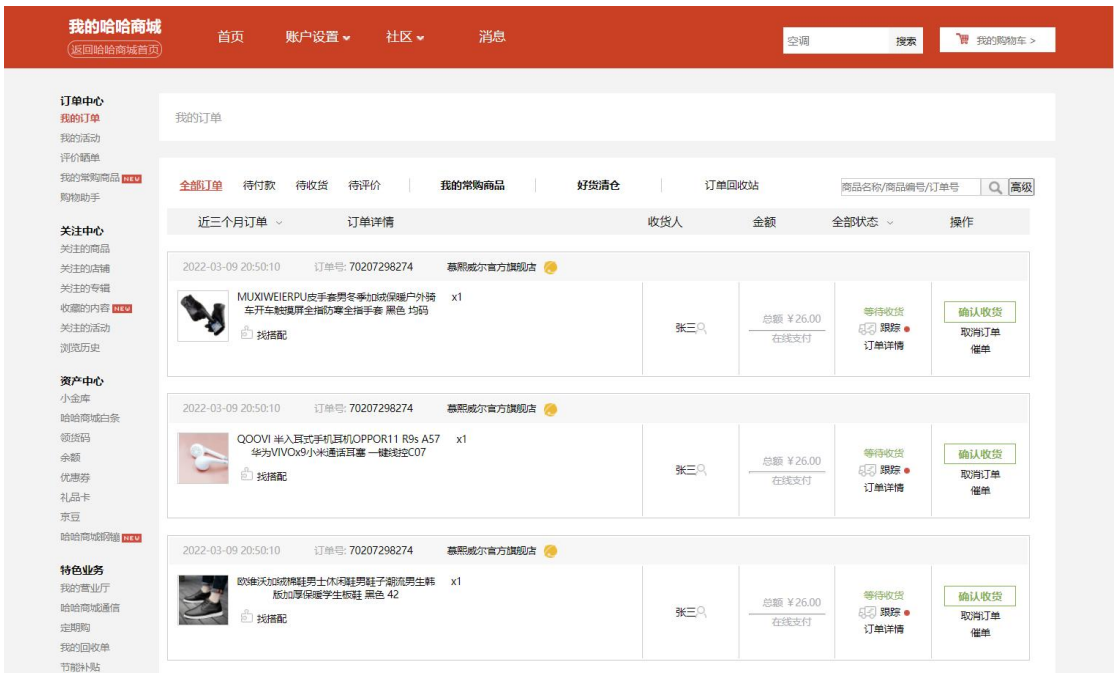

图 5-14 订单系统页面效果图

5.1.5 秒杀功能

整个交易平台中最能体现高并发性能的微服务模块无疑是秒杀功能，在交易系统中使用定时任务+异步任务来实现秒杀功能。在特定时间触发定时异步任务，将接下来的所有秒杀场次以及秒杀商品的相关信息添加到 Redis 中（防止高并发的请求访问数据库），秒杀方法使用 Sentinel 进行限流控制，秒杀商品会在首页展示，并且在商品详情页中提示。秒杀功能流程如图 5-15 所示。

秒杀功能中高并发的体现：从前端秒杀请求发送地开始到快速创建秒杀订单给前端返回页面这一过程中没有操作任何一次数据库，没有做过任何一次远程调用，因此这是一个快速响应的过程，只需要校验好合法性就行（所有数据都在缓

存中），获取信号量的请求不直接调用订单服务而是通过 RabbitMQ 发送消息到后台订单服务慢慢进行消费（流量削峰）。

秒杀流程如下：

(1) 用户在前端页面点击立即抢购后发送 ajax 请求（/seckill?key=随机码&skuId=6&num=6）。

(2) 通过 Spring-Session 检验用户请求是否携带 sessionId 并且与 repository 中的 sessionId 进行对比，没有登录就重定向到登录页面重新登录，同时存储 sessionId。

(3) 校验合法性（确定秒杀时间是否一致，请求时间是否在秒杀场次时间内，校验商品 id 与随机码是否与发布商品时生成的 redis 中的随机码一一对应）。

(4) 尝试向 redis 保存 "respVo.getId()"+"_"+sesssionId+ "_"+skuId" 为 key 的数据，如果 setnx 成功设置过期时间，否则说明当前是重复操作，响应失败。

(5) 调用信号量 Semaphore 作为获取库存，创建订单的凭证。

(6) 快速创建秒杀订单（用户名、订单号、商品）后直接转跳收货信息确认页的同时发送 MQ 消息到订单服务队列中，避免耗费大量时间同步等待响应结果，快速响应用户请求结果，交易系统的订单服务稳定消费队列中的 MQ 消息。

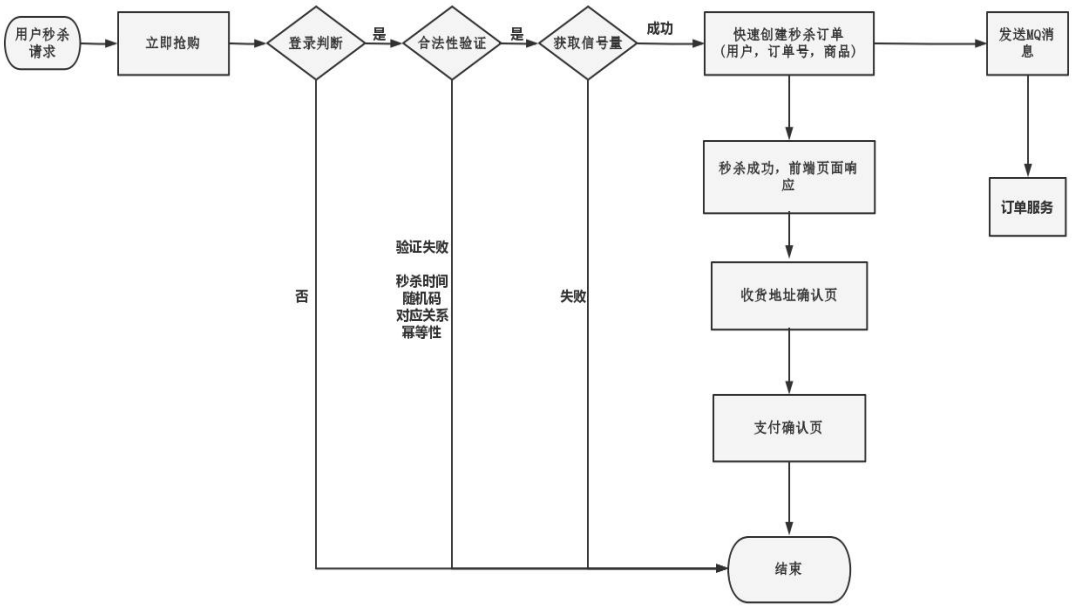

图 5-15 秒杀功能流程图

秒杀功能中 RabbitMQ 部署流程如图 5-16 所示。

图 5-16 秒杀功能中 RabbitMQ 部署流程图

秒杀功能的实现主要是先通过 LoginUserInterceptor 获取用户信息，再根据用户所需要秒杀的商品 skuId 查询 redis 中是否已经缓存，通过时间以及 getSeckillLimit 方法对业务中顾客所规定的购买数量进行合法性校验，如果校验通过则使用 Redisson sempaphore 信号量进行扣减，防止超买超卖情况发送，构建订单信息后通过RabbitTemplate提供的convertAndSend方法发送构建好的订单到消息队列中进行异步扣减。

```
public String kill(String killId, String key, Integer num) {
        MemberRsepVo memberRsepVo = LoginUserInterceptor.threadLocal.get();
        // 1.获取当前秒杀商品的详细信息
        BoundHashOperations<String,String,String>hashOps=
        stringRedisTemplate.boundHashOps(SKUKILL_CACHE_PREFIX);
        String json = hashOps.get(killId);
        if(StringUtils.isEmpty(json)){
            return null;}else{
          SeckillSkuRedisTo redisTo = JSON.parseObject(json, SeckillSkuRedisTo.class);
            // 校验合法性
        long time = new Date().getTime();
        if(time >= redisTo.getStartTime() && time <= redisTo.getEndTime()){
            // 1.校验随机码跟商品 id 是否匹配
        String randomCode = redisTo.getRandomCode();
        String skuId = redisTo.getPromotionSessionId() + "-" + redisTo.getSkuId()
          if(randomCode.equals(key) && killId.equals(skuId)){
            // 2.说明数据合法
        BigDecimal limit = redisTo.getSeckillLimit();
        if(num <= limit.intValue()){
            // 3.验证这个人是否已经购买过了
        String redisKey = memberRsepVo.getId() + "-" + skuId;
            // 让数据自动过期
        long ttl = redisTo.getEndTime() - redisTo.getStartTime();
        Boolean aBoolean= stringRedisTemplate.opsForValue()
           .setIfAbsent(redisKey, num.toString(), ttl<0?0:ttl,TimeUnit.MILLISECONDS);
        if(aBoolean){
            // 占位成功 说明从来没买过
        RSemaphore semaphore= redissonClient
        .getSemaphore(SKUSTOCK_SEMAPHONE + randomCode);
        boolean acquire = semaphore.tryAcquire(num);
        if(acquire){
            // 秒杀成功 快速下单 发送 MQ
        String orderSn=IdWorker.getTimeId() + UUID.randomUUID().
        toString().replace("-","").substring(7,8);
        SecKillOrderTo orderTo = new SecKillOrderTo();
        orderTo.setOrderSn(orderSn);
        orderTo.setMemberId(memberRsepVo.getId());
```

```
            orderTo.setNum(num);
            orderTo.setSkuId(redisTo.getSkuId());
            orderTo.setPromotionSessionId(redisTo.getPromotionSessionId());
            rabbitTemplate.convertAndSend("order-event-exchange","order.seckill.order",
            orderTo);
                return orderSn;
                            }
                        }else {
                        return null;
                    }
                }else{
                    return null;
                }
            }
            return null;
        }
```

页面实现效果如图 5-17 所示。

图 5-17 秒杀功能页面图

5.2 系统测试

5.2.1 系统测试环境

由于服务器的硬件性能，网络传输速率也会在一定程度上影响服务器上程序的运行速度，为保证数据真实可靠，测试的部署环境如下表 5-1 所示。

表 5-1 测试环境展示表

环境	名称	详情
硬件	设备型号	HUAWEI matebook14(2020)
硬件	CPU	AMD Ryzen 5 4600H
硬件	内存	32G
操作系统	Windows 10 家庭中文版	19044.1645
网络	电信宽带	1000M
运行环境	Java	jdk_1.8.0_271

5.2.2 系统性能测试

系统性能测试是为了测试系统秒杀功能的性能指标是否能够满足系统的需求，因此使用 JMeter 压力测试工具，通过控制变量的方式压测不同环境下秒杀接口的性能指标，压力测试内容以及数据如下表 5-2 所示。

表 5-2 系统性能测试表

压测内容	压测线程数	吞吐量/s	90%响应时间	99%响应时间
Nginx（浪费 CPU）	50	2130	9	1105
Gateway（浪费 CPU）	50	9100	8	21
简单服务（返回字符串）	50	9450	7	50
首页一级菜单渲染	50	345	278	500
首页菜单渲染(开缓存)	50	460	125	300
首页菜单渲染(开缓存、优化数据库、关日志)	50	490	100	280
三级分类数据获取	50	5	13250	13200
三级分类(优化业务)	50	17	4000	5902
首页全量数据获取	50	3.1	23983	27211
首页全量数据获取(动静分类)	50	5.1	14913	15899
Gateway+简单服务	50	2950	29	70

压测内容	压测线程数	吞吐量/s	90%响应时间	99%响应时间
全链路(Nginx+GateWay+简单服务)	50	660	82	543

由上表数据可得出中间件越多，性能的损失也就越大，因此本系统采用单一部署秒杀服务的方式通过用户单独请求进行压力测试，压测结果如下表 5-3 所示。

表 5-3 秒杀功能测试表

Label	样本	平均值	最小值	最大值	标准偏差	异常	吞吐量
HTTP 请求	1000	754	532	985	63.28	0.00%	1384.4/sec

通过压力测试结果可以得知当服务器接收的 Http 请求量在 1000 时，接口的平均响应时间控制在 0.7s 左右。并且标准偏差保证在 0.1s 以内，可测量出该交易系统的接口压力性能良好，并且稳定。

5.2.3 系统功能测试

系统的功能测试的目的是保证系统部署在服务器上长时间运行不能出现问题，功能测试需要将系统实现的功能全部测试以保证系统的稳定性，用户的测试包括用户注册、用户登录、管理员登录、购物车检索、商品添加、订单查询、订单完成等功能。测试表如下表 5-4 所示。

表 5-4 系统功能测试表

用例编号	测试步骤	测试预期	测试结果	结论
1	1.注册用户 2.登录用户 3.登录管理员	1.注册后页面显示注册成功 2.登录后页面显示登录成功	1.注册后页面显示注册成功 2.登录后页面显示登录成功	通过
2	1.添加商品到购物车 2.点击商品数量旁的”+”“-” 3.点击商品项中的删除	1.购物车中能看到添加的商品 2.商品商量随着点击进行增减 3.商品从购物车中删除	1.购物车中能看到添加的商品 2.商品商量随着点击进行增减 3.商品从购物车中删除	通过
3	1.搜索小米手机	1.商品列表显示小米手	1.商品列表显示小米手	通

用例编号	测试步骤	测试预期	测试结果	结论
	2. 价格区间输入 2000-3000	机 2. 显示价格区间为 2000-3000 的小米手机	机 2. 显示价格区间为 2000-3000 的小米手机	过
4	1.点击确认购买 2.登录支付账号 3.付款完成	1.购买后转跳到登录页 2.输账号后进入支付页 3.付款后显示付款成功	1.购买后转跳到登录页 2.输账号后进入支付页 3.付款后显示付款成功	通过
5	1.点击订单查询 2.输入订单编号 3.点击搜索	1.展示订单详情页面 2.查询出需要搜索的订单	1.展示订单详情页面 2.查询出需要搜索的订单	通过
6	1.点击秒杀活动商品 2.点击参与秒杀 3.响应秒杀成功或者失败页面	1.进入秒杀商品详情页 2.创建秒杀订单 3.显示秒杀成功页面	1.进入秒杀商品详情页 2.创建秒杀订单 3.显示秒杀成功页面	通过

第 6 章 总结和展望

6.1 本文总结

哈哈商城是采用基于微服务的分布式交易系统，类似于京东商城。之所以选择微服务架构，是由于之前用单体服务做安卓项目的时候在上线过程中由于一台服务器进行过多服务的交互，导致服务器刚上线没多久就崩了，后面在查找问题后才发现是因为有段时间访问量过大导致系统崩溃，于是本项目使用了阿里云服务器做负载均衡，优化后服务器没有崩溃了，可随着功能的不断完善，渐渐发现一个新的服务加入到自己项目中日益复杂，加上应用边界模糊，数据库结构被多个应用依赖，很难进行重构，每次可能只改动一个小功能，也会牵扯到大量逻辑改动，导致开发进程严重受损。于是本次项目对业务进行了水平拆分，对一些复杂服务进行了抽象处理，最终将商城系统抽象成 10 个独立的服务。涉及数据表 52 张，同时上传到码云 gitee 进行代码管理。历时 5 个月不断地学习同时查资料，最终实现了各大电商商城的主要功能。

技术难点：由于自己之前一直准备实习，因此做的是一款面试经常问到的需要兼容高并发高性能的分布式微服务商城，而哈哈商城难点之一在于高并发，可以想象在商品秒杀时候一般都会有 10w+用户同时访问同一个商品页面去抢购手机，如果系统没有经过限流或者熔断处理，那么系统瞬间就会崩掉就好像被 DDos 攻击一样。难点之二在于超卖。由于不同线程读取到的当前库存数据可能下个毫秒就被其他线程修改了也就如数据库的幻读一样，没有一定的锁库存机制那么库存数据必然出错，因此又去看了一下同步锁和分布式锁的知识点。难点之三在于性能，如何进行性能优化，如何保证在高并发请求下，让用户能有更好地体验，如何调参调内存让用户能有更好地体验，这些都是需要考虑的。因此项目引进了 redis 作为数据缓存，nginx 为服务器端做负载均衡。

6.2 未来展望

在本次项目中无数次面对由于版本问题出错而折腾半天还没有一点进展，同时搭建环境时也折腾了很久，由于各种版本依赖的坑，查询的很多资料有些过于老旧，导致 maven 引入依赖库时没有将之前依赖的组件加入新的环境中。

在完成论文的过程中我也清晰地认识到了自己还有很多技术上面的不足，以及对整体性的把握能力不够，设计能力不够等问题，需要加强对业务的分析能力，进一步抽象化，考虑好业务逻辑。总体上这次论文课题的完成让我更加清楚日后面对项目难点时应该如何思考，如何解决困难，也为日后的工作提前准备。

参考文献

［1］ 李林娟.中国网络购物现状与发展趋向分析[J].中国管理信息化,2016(20):135-136.

［2］ 刘颖.基于分布式系统的微服务架构演进[J].通讯世界,2018(07):97-98.

［3］ Chen L.Continuous delivery:Huge benefits,but challenges[J].IEEE software,2015,32(02):50-54.

［4］ 魏茂之.SOA 技术架构在企业中的应用[J].计算机光盘软件与应用,2014,17(09):130-132.

［5］ SpringCloud[EB/OL].https://spring.io/projects/spring-cloud,2020.

［6］ Gateway[EB/OL].https://cloud.spring.io/projects/gateway,2020.

［7］ 廖俊杰,陶志勇.网关的设计及应用[J].自动化技术与应用,2019(08):85-88.

［8］ Nacos[EB/OL].https://www.github.com/alibaba/nacos,2018.

［9］ Sentinel[EB/OL].https://www.github.com/alibaba/Sentinel,2018.

［10］Elasticsearch[EB/OL].https://www.elastic.co/products/elasticsearch,2018.

［11］RabbitMQ[EB/OL].https://www.rabbitmq.com/,2020.

［12］邱祝文.基于 redis 的分布式缓存系统架构研究[J].网络安全技术与应用，2014(10):52-54.

［13］陈康贤.大型分布式网站架构设计与实践[M].电子工业出版社,2014:2.

致谢

大学生活在不知不觉中也就要告一段落了。大学四年一转眼就过去了，从刚开始步入电子科技大学中山学院的懵懵懂懂到如今的不舍，非常感谢这几年来母校给予自己的成长，让自己有能力独立进入社会，不论是实验室的科研还是日常生活的学习都让我受益良多，在本次论文项目的调研，选题，撰写，成型的过程中，我也学习到了一个项目实现的整体流程，咨询了很多人，询问了很多意见，由衷地感谢老师和同学的耐心解答。

特别感谢我的指导老师，何怀文副教授在我大学阶段给予我的指导，无论是生活上还是学习的道路上以及职业的选择方面的疑惑，老师都为我耐心解答，在我的大学生涯中，加入何怀文老师带领的实验室无疑是我人生的一个转折点，老师在实验室为其他同学讲解的耐心，以及平时认真的工作态度无时无刻不在激励着我勇往直前学习更多的知识。

同时感谢我的家人，如果不是他们努力工作赚钱为我支付学费，恐怕我也没有时间去学习知识充实自己。感谢他们在我生活中对我的教导，让我更加有勇气去面对困难，去思考问题，去解决问题，使我在即将进入社会的时候能有勇气面对以后的困难。

最后，感谢电子科技大学中山学院，感谢计算机学院，感谢身边的同学以及朋友，感谢你们让我不断成长，我的大学生涯基本上画上圆满句号，但我会踏踏实实走好以后的人生道路，不断学习，承担社会责任，并且回报社会。