目 录

基于 Goroutine 的高并发秒杀系统设计与实现

摘 要：近年来，我国互联网普及率日益增长，对于贴切广大网民生活的网路零售持续稳健发展，成为消费增长的重要动力。随着电商的稳步发展，各大电商平台都不遗余力的太阔新的营销模式来增加消费者的欲望，秒杀活动已随处可见，秒杀活动相比普通的网络零售，秒杀活动中商品价格低廉能够吸引更多的网民进行有限商品的抢购，商家通过电商平台对秒杀商品的大幅推广并定时上架，用户则会在短时间瞬时抢购完部分展示商品。

基于 GoRoutine 的高并发秒杀系统主要面向平台商家和消费者，支持平台商家可以使用秒杀系统后台进行管理，以及消费者可以进行商品的定时秒杀抢购。考虑到秒杀系统的高性能和高可用性，对秒杀系统进行了分布式集群的架构设计。对于高并发场景下，Golang 具有天然的优势，在于单机 Go 服务程序支持开上万个 goroutine 协程，相比多进程，多线程的并发编程模型，能够更好利用单机多核 CPU 的优势。在考虑单机的性能瓶颈后，进一步向集群化来提升系统整体的抗压能力和稳定性，使用到 Nginx 高性能的反向代理服务器来做流量的负载均衡，转发流量至分布式服务集群。

关键词：Goroutine；高并发；秒杀系统；RabbitMQ

Design and Implementation of High Concurrency Spike System Based on Goroutine

Abstract**:**In recent years, China's Internet penetration rate has been increasing day by day, and the continuous and steady development of online retailing, which is relevant to the lives of the majority of netizens, has become an important driving force for consumption growth. With the steady development of e-commerce, major e-commerce platforms have spared no effort to broaden their new marketing models to increase consumers' desires. Spike activities have been seen everywhere. Compared with ordinary online retailing, spike activities are cheaper in commodity prices It can attract more netizens to snap up limited commodities. Merchants promote the spike products through e-commerce platforms and put them on shelves regularly. Users will snap up some of the displayed commodities in a short time.

The GoRoutine-based high-concurrency spike system is mainly aimed at platform merchants and consumers. It supports platform merchants to use the spike system to manage the background, and consumers can carry out regular spike killing of goods. Considering the high performance and high availability of the spike system, a distributed cluster architecture design was carried out for the spike system. For high concurrency scenarios, Golang has a natural advantage in that the stand-alone Go service program supports the opening of tens of thousands of goroutine coroutines. Compared with multi-process and multi-threaded concurrent programming models, it can better utilize the advantages of stand-alone multi-core CPUs.

Key words: Goroutine; High concurrency; Spike system; RabbitMQ

第一章 绪论

1.1 研究背景

近年来，我国互联网普及率日益增长，数字鸿沟不断缩小，截止 2020 年 3 月，我国网民规模达 9.04 亿，较 2018 年底增长 7508 万，互联网普及率达 64.5%，较 2018 年底提升 4.9 个百分点。对于贴切广大网民生活的网络零售则持续稳健发展，成为消费增长的重要动力，移动终端和支付技术的进步助推电商在网民中的渗透率提升，电商体系在中国已发展成熟。随着电商的稳步发展，各大电商平台都不遗余力的开拓新的营销模式来增加消费者的欲望，秒杀活动已随处可见。

秒杀活动相比普通的网络零售，秒杀活动中商品价格低廉能够吸引更多的网民进行有限商品的抢购，商家通过电商平台对秒杀商品的大幅推广并定时上架，用户则会在短时间瞬时抢购完部分展示商品。总的来说，秒杀活动作为电商平台中常用的一种运营手段，通过低价限时来获取用户贪便宜的心理，既解决了消费者的消费需求，平台也通过秒杀活动清理库存，完成了运营的促销 KPI，产品也可拉新获客，每到节日电商平台都会开展“活动”。例如：京东的“618”活动，淘宝“双 11”狂欢节，拼多多的“百亿补贴”，洋码头的“黑五狂欢节”。

然而，相比普通网站购物而言秒杀活动对网站技术却是极大的挑战。秒杀活动是在同一时刻大量请求争抢购买同一商品并完成交易的过程，由于消费者的大量级请求及秒杀商品的限时，限量等特性，使得秒杀活动相比普通的商品购买具有瞬时高并发的特点，因此秒杀系统要满足用户的良好体验以及整体的抗打稳定性，必须保障系统的高性能及高可用。

1.2 研究目标

1.2.1 实现秒杀基本业务

秒杀活动对于商家来说，可以快速获取商品流量，累计店铺销量；卡勠那个低门槛条件吸引买家关注，打造商品特色；通过秒杀带动产品销量提升，并提升该单品的商品搜索排位，进而增加店铺权重。

秒杀系统主要面向平台商家和消费者，即平台商家可以使用秒杀系统后台设置秒杀规则(秒杀开始时间及结束时间)及秒杀商品的详情信息，对秒杀商品进行管理以

及对用户成功下单的订单进行状态修改；消费者则通过秒杀系统进行秒杀商品的浏览和对秒杀活动的详细查阅，在特定的时间消费者可以对秒杀商品进行下单抢购。

1.2.2 分布式集群扩展高并发

秒杀活动是在同一时刻大量请求争抢购买同一商品并完成交易的过程，由于消费者的大量级请求及秒杀商品的限时，限量等特性，使得秒杀活动相比普通的商品购买具有瞬时高并发的特点，因此秒杀系统要满足用户的良好体验以及整体的抗打稳定性。

单机的架构设计无法应对高并发的秒杀活动，因此需要设计出分布式集群的系统架构。

1.3 主要工作

基于 GoRoutine 的高并发秒杀系统主要面向平台商家和消费者，支持平台商家可以使用秒杀系统后台设置秒杀规则(秒杀开始时间及结束时间)及秒杀商品的详情信息，对秒杀商品进行管理以及对用户成功下单的订单进行状态修改；消费者则通过秒杀系统进行秒杀商品的浏览和对秒杀活动的详细查阅，在特定的时间消费者可以对秒杀商品进行下单抢购。

同时考虑系统的高性能和高可用性，对秒杀系统进行了分布式集群的架构设计。对于高并发场景下，Golang 具有天然的优势，在于单机 Go 服务程序支持开上万个 goroutine 协程，相比多进程，多线程的并发编程模型，能够更好利用单机多核 CPU 的优势。在考虑单机的性能瓶颈后，进一步向集群化来提升系统整体的抗压能力和稳定性，使用到 Nginx 高性能的反向代理服务器来做流量的负载均衡，转发流量至分布式服务集群。

主要研究内容及取得的成果有：

1. 对面向商家的秒杀系统后台进行详细分析，主要实现商家的管理模块，包括商品管理模块，订单管理模块，黑名单管理模块，以及登录注册模块。后台实现使用到了 beego 框架，一款高性能的 mvc 框架，其中前端框架使用 bootstrap 和 jquery 展示动态效果的网页，前后端的交互通过 golang 的 tmplate 模板引擎从数据库获取数据后生成 HTML 静态文件。

2. 对面向消费者的商品秒杀系统进行架构设计，分为四层，分别为 Nginx 网关

层用于流量的负载均衡；接入层实现可以分布式扩容的服务层，用于权限拦截和数量控制；业务逻辑层用于异步化订单消息至 RabbitMQ 对列中，另一端用于持久化订单入数据库；存储层通过 MYSQL 事务特性保障数据的严格一致性。

3. 对秒杀系统进行测试环境的搭建后，进行全面的功能测试和性能测试，并对测试结果进行分析。

1.4 组织结构

本文总共有六个章节，组织结构如下：

第一章是绪论，分析了秒杀系统的研究背景以及相比普通网购的技术难度和挑战性，分析了基于 Goroutine 的高并发秒杀系统研究目标，即面向平台商家和消费者的业务体验和高性能，可伸缩的系统目标。

第二章是技术综述，本章介绍了项目设计到的核心技术和算法，包括 GoRoutine 的研究分析,RabbitMQ 消息中间件，Redis 与 MemCache 缓存对比以及一致性 Hash 算法。。

第三章是需求分析和概要设计，本章分析了基于 GoRoutine 的高并发秒杀系统的功能需求和非功能需求，并基于需求分析对系统进行了概要设计包括系统的架构设计与持久化模型设计，为第四章系统的模块化详细设计提供了支持

第四章是详细设计与实现，本章从秒杀系统后台和面向消费者的秒杀系统两个方面进行了详细模块的分析和设计，并提供了模块的相关核心代码。

第五章是系统测试与分析，本章对秒杀系统的环境部署进行了介绍，以及对研究所取得的成果进行验证测试，主要从秒杀系统的单机到集群进行性能测试，以及验证系统的可伸缩性。

第六章是总结和展望，本章总结了整个秒杀系统设计与论文撰写的工作，回顾整个项目的不足并且对未来发展进行了一定的展望。

第二章 相关技术研究

2.1 GoRoutine 的研究与分析

Goroutine 也称为 Go 协程，不同于进程和线程的并发模型，协程是一种用户态的轻量级线程，并且协程的调度完全由用户(Goroutine 的调度由 go 运行时调度器调度)控制。Goroutine 非常轻量，主要体现在上下文切换代价小和内存占用少两个方面，Goroutine 的上下文切换代价极小只需要涉及到三个寄存器的值修改并且每个 Goroutine 的占空间最小 2K，因此 go 应用能开更多的协程。

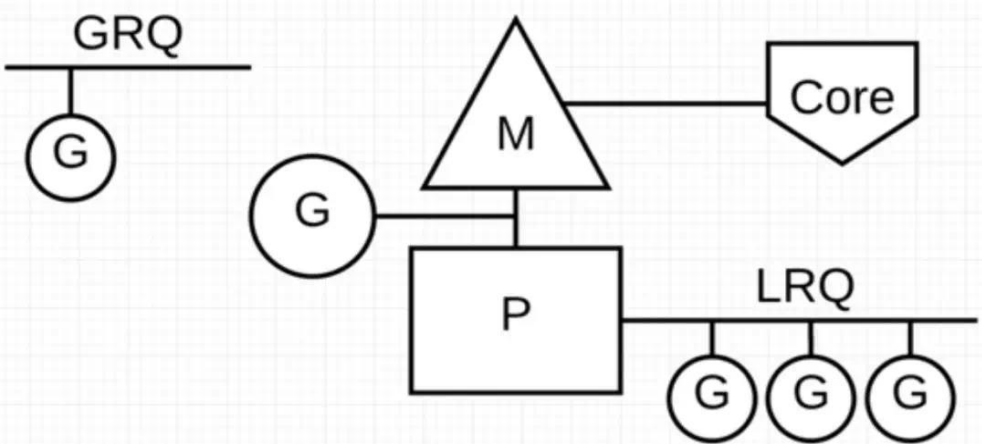

图 2.1 MPG 调度模型

Go 程序通过调度器来调度 Goroutine 在内核线程上执行，并且 Go 的调度器模型也被称为 MPG 模型，如图 2.1。其中 M 表示对内核线程的抽象，代表着真正执行计算的资源；P 表示逻辑处理器，可以认为是 G 需要获取的 CPU 核；G 则对应一个 G 结构体，存储 Goroutine 的运行堆栈，状态以及任务函数。

Go 调度器为了更好的充分利用线程的计算资源，采取了一下几种调度策略:

1. 任务窃取：为调高整体处理效率，当每个 P 之间的 G 任务不均衡时，调度器允许从全局任务队列或者其他 P 的本地任务队列中获取 G 来执行。
2. 当有原子，互斥量或者通道操作调用导致 Goroutine 阻塞，调度器将把当前阻塞的 Goroutine 切换出去，重新调度 LRQ 上的其他 Goroutine;
3. 当有网络请求或 IO 操作导致 Goroutine 阻塞，Go 运行时会提供网络轮询器来处理网络请求和 IO 操作，直到网络轮询器获取已完成的通知便会将 Goroutine 重新放入运行队列中。

4. 当有 Goroutine 中执行一个 sleep 操作导致 M 阻塞时，Go 程序后台会有一个监控线程 sysmon，它会监控那些长时间运行的 G 然后设置可以强占的标识符。

2.2 RabbitMQ 消息中间件

消息中间件关注于数据的发送和接受，利用高效可靠的异步消息传递机制进行平台无关的数据交流，并基于数据通信来进行分布式系统的集成。通过提供消息传递和消息排队模型，它可以在分布式环境下扩展进程间的通信。消息队列则是消息中间件的一种方式，其中常见的消息中间件有 RabbitMQ，RabbitMQ，ActiveMQ，Kafka，ZeroMQ.

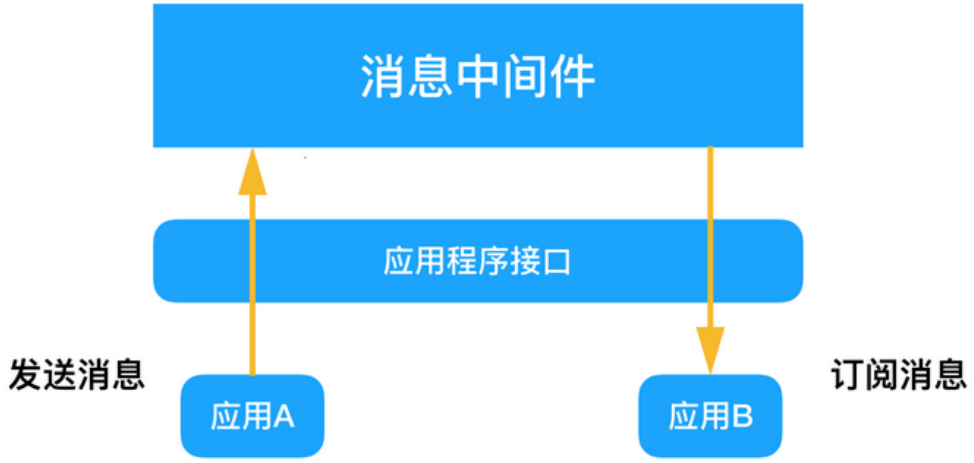

图 2.2 消息中间件通信方式

RabbitMQ 是由高性能，健壮以及可伸缩性出名的 Erlang 写成，一个开源的 Advanced Messsage Queuing Protocol (AMQP)的开源实现。其中 RabbitMQ 由多种工作模式：simple 简单模式适用于生产者和消费者为一对一的关系，一条消息只能被一个消费者；work 工作模式适用于生产者和消费者为一对多的关系，多个消费者可以绑定到一个队列，共同消费队中的消息；订阅广播模式，用于生产者生产一个条消息后匹配规则至不同的消费者。

2.3 Redis 与 MemCache 缓存对比

Memcached 是一个高性能的分布式内存对象缓存系统，用于动态 Web 应用以减轻数据库负载，主要通过在内存中缓存数据和对象来减少读取数据库的次数，，以提高动态 Web 等应用的速度、 提高可扩展性。如图，多个 Memcached 节点之间不互相

通信，其分布式是在客户端实现的，通过内置的算法定制目标数据的节点。

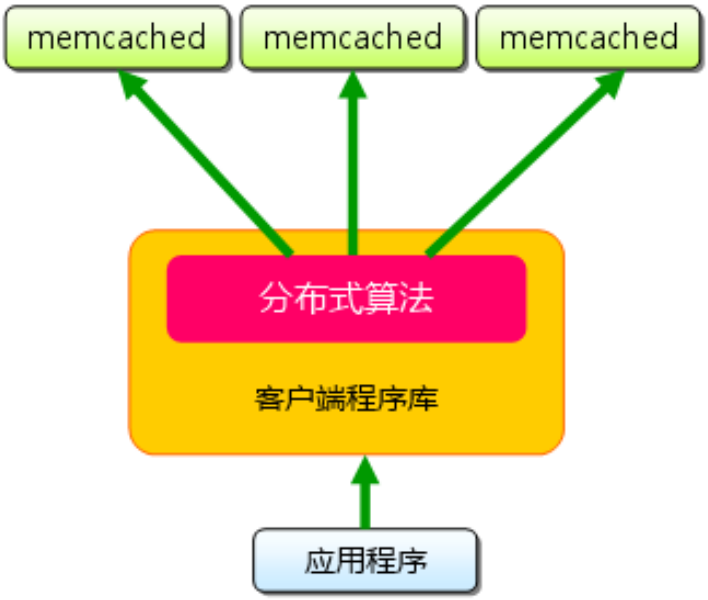

图 2.3 MemCache 分布式通信

Redis 是一个 key-value 存储系统。和 Memcached 类似，它支持存储的 value 类型相对更多，包括 string(字符串)、 list(链表)、set(集合)和 zset(有序集合)。这些数据类型都支持 push/pop、add/remove 及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis 支持各种不同方式的排序。与 memcached 一样，为了保证效率，数据都是缓存在内存中。区别的是 redis 会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，

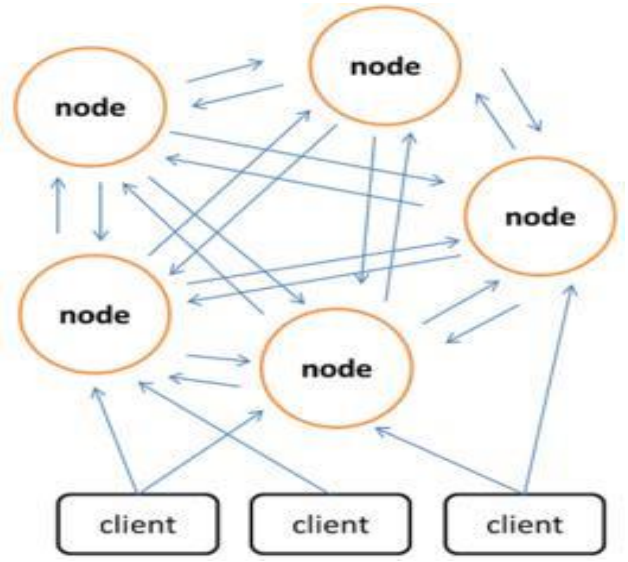

图 2.4 Redis 分布式通信

Redis Cluster 是一个实现了分布式且允许单点故障的 Redis 高级版本，它没有中心节点，各个节点地位一致，具有线性可伸缩的功能。如图给出 Redis Cluster 的分布式存储架构，其中节点与节点之间通过二进制协议进行通信，节点与客户端之间通过 ascii 协议进行通信。在数据的放置策略上，Redis Cluster 将整个 key 的数值域分成 16384 个哈希槽，每个节点上可以存储一个或多个哈希槽，也就是说当前 Redis Cluster 支持的最大节点数就是 16384。

2.4 一致性 Hash 算法

一致性 Hash 算法设计初衷是为了解决因特网中的热点问题，其修正了 CARP 使用的简单哈希算法带来的问题。一致性 Hash 主要是将 key 做 hash 运算，再按照规律取整得到 0-2^32-1 之间的值，然后将其映射到一个节点。规则是首先将服务的 key 按该 hash 算法计算，得到在一致性 hash 环上的位置，在将缓存的 key 用同样的方法计算得到 hash 环上的位置，按顺时针方向，找到第一个大于等于该服务 key 即需要分配的服务器。如图 2.5 所示，当某一节点失效时，如 NODE2，则 NODE2 上的 ke 将分配到 hash 环上相邻的节点，而其他 key 所在位置不变。

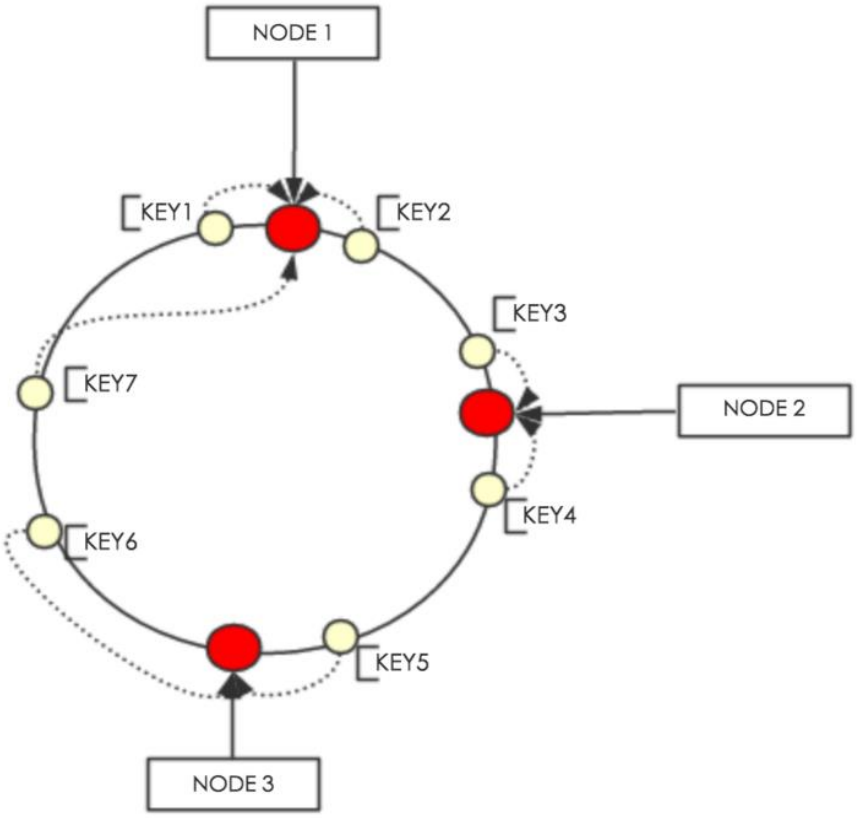

图 2.5 一致性 Hash 算法

同时为了提高均衡性可以添加虚拟节点，让一个是节点对应多个虚拟节点，如图 2.6 所示，

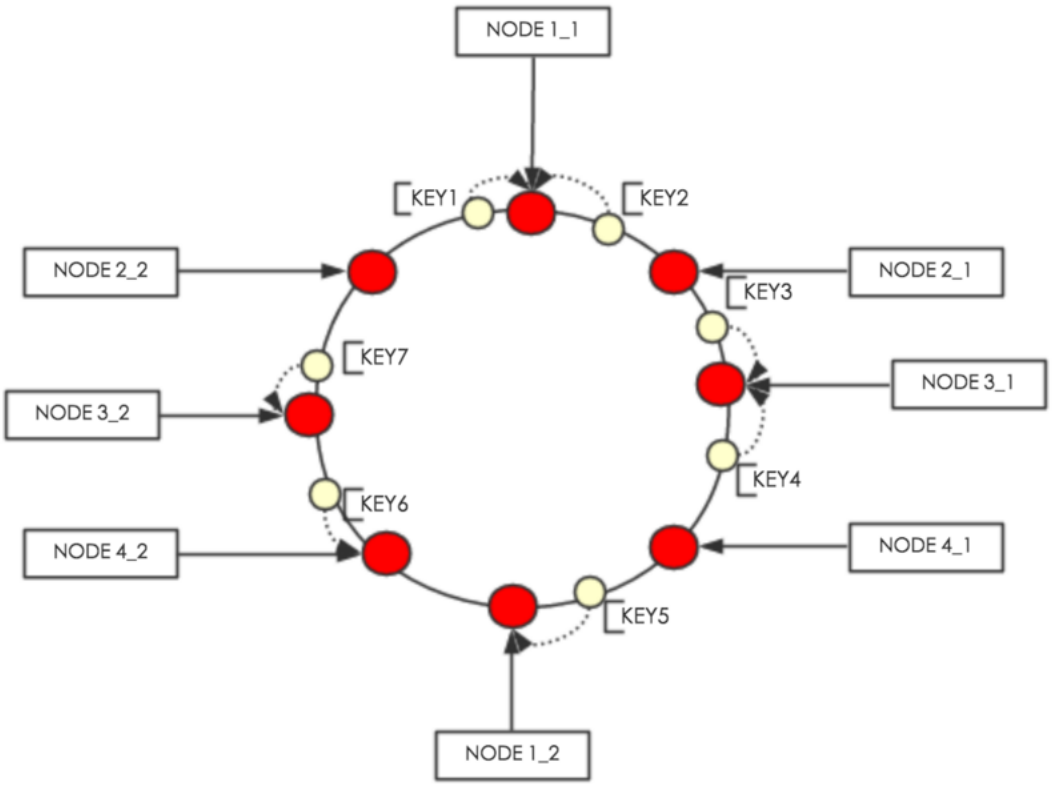

图 2.6 一致性 Hash 算法

第三章 系统需求分析与概要设计

3.1 系统整体概述

秒杀活动作为一种营销手段，就是平台商家通过发布一些超低价格的商品，所有买家在同一时间网上抢购的一种销售方式。用户可以在秒杀页面中进行点击立即抢购，即向平台发送了秒杀的请求，平台服务器做出响应，判断看库存等，判断用户抢购成功后系统扣除库存，修改库存数量，生成用户订单，平台进行发货，物流配送等操作。

秒杀活动对于商家来说，可以快速获取商品流量，累计店铺销量；卡勠那个低门槛条件吸引买家关注，打造商品特色；通过秒杀带动产品销量提升，并提升该单品的商品搜索排位，进而增加店铺权重。对于消费者而言，秒杀活动的商品价格低廉，往往一上架就被抢购一空，非常具有吸引力和趣味性。

秒杀系统主要面向平台商家和消费者，即平台商家可以使用秒杀系统后台设置秒杀规则(秒杀开始时间及结束时间)及秒杀商品的详情信息，对秒杀商品进行管理以及对用户成功下单的订单进行状态修改；消费者则通过秒杀系统进行秒杀商品的浏览和对秒杀活动的详细查阅，在特定的时间消费者可以对秒杀商品进行下单抢购。

3.2 系统需求分析

3.2.1 秒杀系统后台功能需求

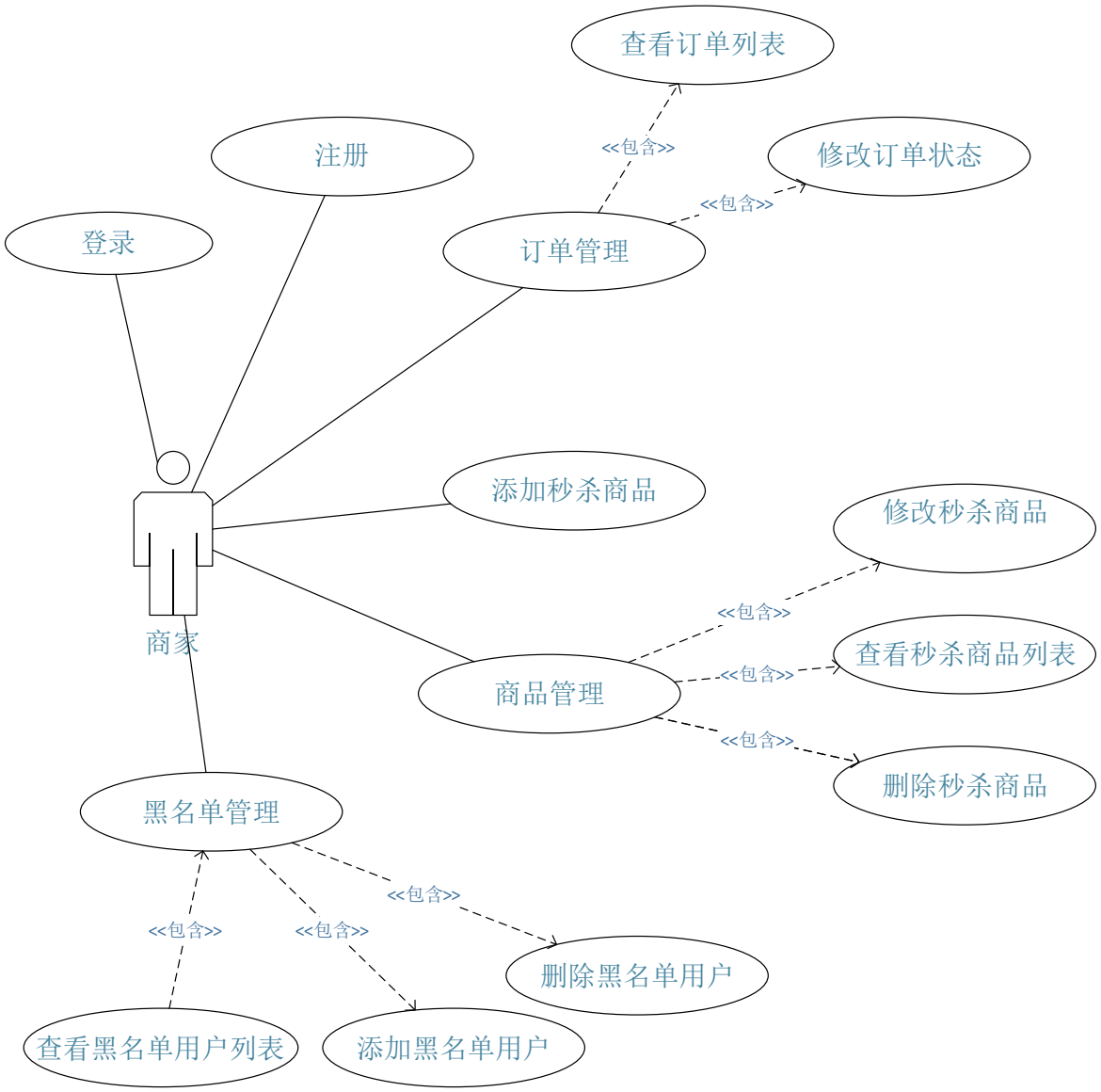

图 3.1 商家用例图

秒杀系统后台主要面向商家，通过对商家的功能需求分析，可知平台商家的用例图如图 3.1 所示：

表 3.1：添加秒杀商品用例描述

ID	UCI
名称	添加秒杀商品
参与者	商家
触发条件	商家点击查看所有商品
前置条件	商家必须已验证登录成功
后置条件	无

优先级	高
正常流程	1. 商家登录完成； 2. 系统左侧界面导航栏展示添加秒杀商品选项； 3. 商家点击新增秒杀商品 4. 系统返回秒杀商品表单 5. 商家填写商品的详细信息以及秒杀活动时间并提交 6 系统返回车功添加的秒杀商品信息
特殊需求	商家删除指定商品时需要再次确认删除

添加秒杀商品主要面向平台商家用于发布和上传秒杀商品的详细信息，商家在登录验证成功后并点击系统左侧界面导航栏添加秒杀商品，系统返回商家待填写的商品信息，包括名称，库存数量，旧价格，新价格，图片，商品描述以及秒杀活动的开始和结束时间。商家上传成功后可以在查看所有秒杀商品列表中查看。查看添加秒杀商品用例描述如表 3.1 所示。

表 3.2：管理秒杀商品用例描述

ID	UC2
名称	管理秒杀商品
参与者	商家
触发条件	商家点击查看所有商品
前置条件	商家必须已验证登录成功
后置条件	无
优先级	高
正常流程	1. 商家登录完成； 2. 系统左侧界面导航栏展示管理秒杀商品选项； 3. 商家点击查看所有秒杀商品 4. 系统分页展示所有已发布的秒杀商品 5. 商家点击修改或删除该撒谎指定商品 6. 系统返回商品详细信息 7. 商家对指定商品进行修改或删除确认
特殊需求	商家删除指定商品时需要再次确认删除

管理秒杀商品主要面向平台商家用于修改和删除已发布的秒杀商品，商家在登录验证成功后并点击系统左侧界面导航栏查看所有秒杀商品，系统分页返回所有已发布秒杀商品列表，用户可以对指定的商品选项进行删除以及修改。当商家选择删

除时，系统会提示删除确认信息。查看管理秒杀商品用例描述如表 3.2 所示。

表 3.3：订单管理用例描述

ID	UC3
名称	订单管理
参与者	商家
触发条件	商家点击查看所有订单
前置条件	商家必须已验证登录成功
后置条件	无
优先级	高
正常流程	1. 商家登录完成； 2. 系统左侧界面导航栏展示管理订单选项； 3. 商家点击查看所有订单 4. 系统分页展示所有已秒杀成功的订单 5. 商家可以修改已付款成功的订单的发货状态 6. 系统返回确认提示
特殊需求	商家只允许修改已付款状态下的订单是否已发货

订单管理主要面向平台商家用于对已成功秒杀后的订单进行查看，并修改订单的状态是否已发货。商家在登录验证成功后点击系统左侧界面导航栏管理所有订单选项，系统分页返回所有已抢购成功的订单，其中订单的信息包括订单 ID，用户名称，商品名称，付款状态，发货状态。商家只能修改已付款状态下的订单是否已发货。查看订单管理用例描述如表 3.3 所示。

表 3.4：黑名单管理用例描述

ID	UC4
名称	黑名单管理
参与者	商家
触发条件	商家点击黑名单管理
前置条件	商家必须已验证登录成功
后置条件	无

优先级	高
正常流程	1. 商家登录完成； 2. 系统左侧界面导航栏展示管理黑名单选项； 3. 商家点击管理黑名单 4. 系统分页展示所有已添加的黑名单用户 5. 商家可以从列表中删除指定用户或查询指定用户名并添加 6. 系统返回已完成界面
特殊需求	无

黑名单管理主要面向平台商家用于对商家禁止的消费者用户的添加，查看和删除。商家在登录验证成功后点击系统左侧界面导航栏管理黑名单选项，系统分页返回所有已被商家禁止的黑名单用户，商家可以继续在页面中添加消费者的用户名，以及删除指定用户。黑名单管理用例描述如表 3.4 所示。

3.2.2 面向消费者功能需求

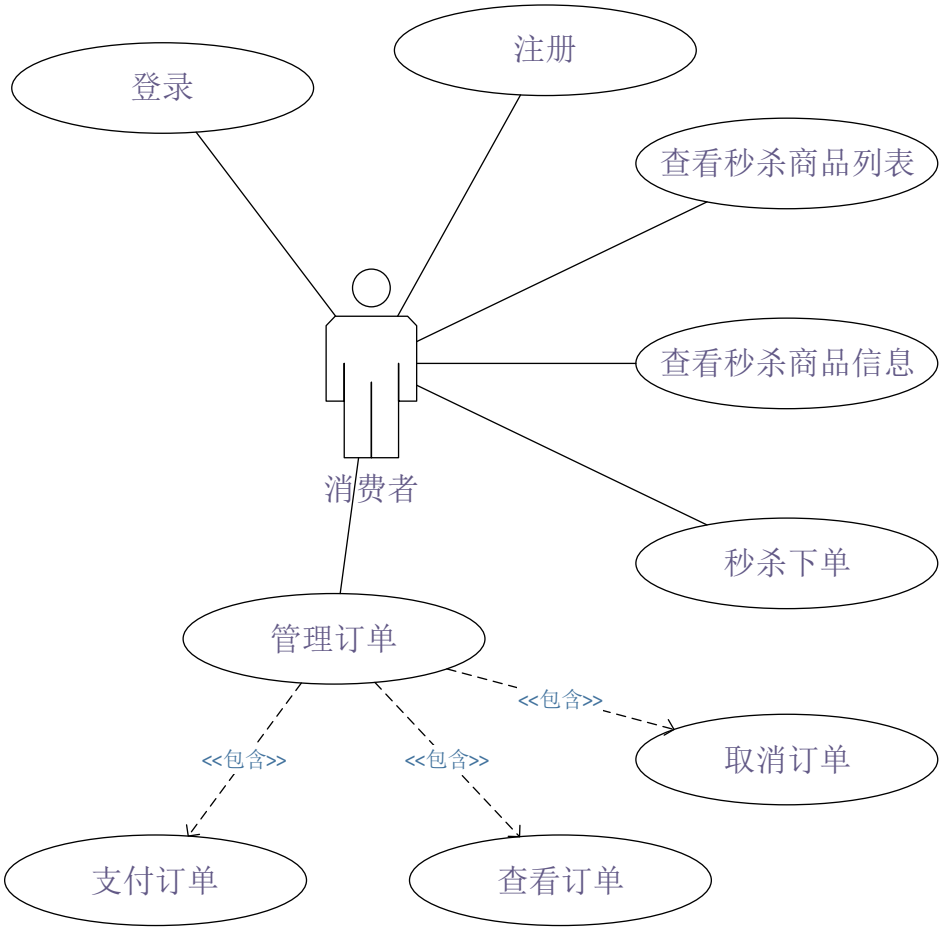

图 3.2 用户用例图

秒杀系统后台主要面向商家，通过对商家的功能需求分析，可知平台商家的用

例图如图 3.1 所示：

查看秒杀商品列表主要面向平台消费者，当消费者进入平台网站后即可获取所有正在秒杀或者秒杀倒计时的秒杀商品。

查看秒杀商品信息主要面向平台消费者，消费者从秒杀商品列表中点击进入获取秒杀商品的详细信息，包括商品的所在店铺，名称，库存数量，图片，描述信息以及是否正在秒杀中或秒杀倒计时。

秒杀下单主要面向平台消费者在秒杀商品约定的抢购时间中，消费者点击一键秒杀按钮进行抢购。如果用户秒成功，则商品库存会减一并会生成订单返回提示消费者付款。如果用户秒杀失败，系统会返回提示用户失败，此时如果其他原因导致失败则需要用户重新刷新进入页面再次点击抢购。

管理订单主要面向平台消费者，消费者可以查看所有已抢购成功的订单列表信息，并对订单进行支付，删除。

3.2.3 非功能性需求

消费者会在同一时刻大量请求争抢购买同一商品并完成交易，由于消费者的大量级请求及秒杀商品的限时，限量等特性，使得秒杀活动相比普通的商品购买具有瞬时高并发的特点，因此秒杀系统要满足用户的良好体验以及整体的抗打稳定性，必须保障系统的高性能及高可用。因而秒杀系统需要关注的非功能需求有：

1）性能：

面向消费者的秒杀系统需要保证用户响应时间应小于 200ms，复杂的数据库查询操作不超 500ms，并且能满足多用户高并发

2）可用性：

秒杀系统需要保证 7*24 全天候的稳定性，需要在任意时间点，保证 99%概率系统能够正常运行并不会受瞬时高并发的大流量影响到整体的运作。

3）易用性：

API 设计规范并易读，用户界面简洁好看，用户能够轻松且快速的对指定商品秒杀下单并且付款，具有一定的提示语。

4）伸缩性：

当预估秒杀活动的用户量和并发量比较大时，可以快速对秒杀系统进行横向扩

展即添加多台服务主机节点，多集群。

5）安全性：

为保证秒杀活动的公正公平，防止暴力刷单秒杀软件以及恶意攻击，秒杀系统需要提升整体的安全指标。

3.3 系统概要设计

3.3.1 系统架构设计

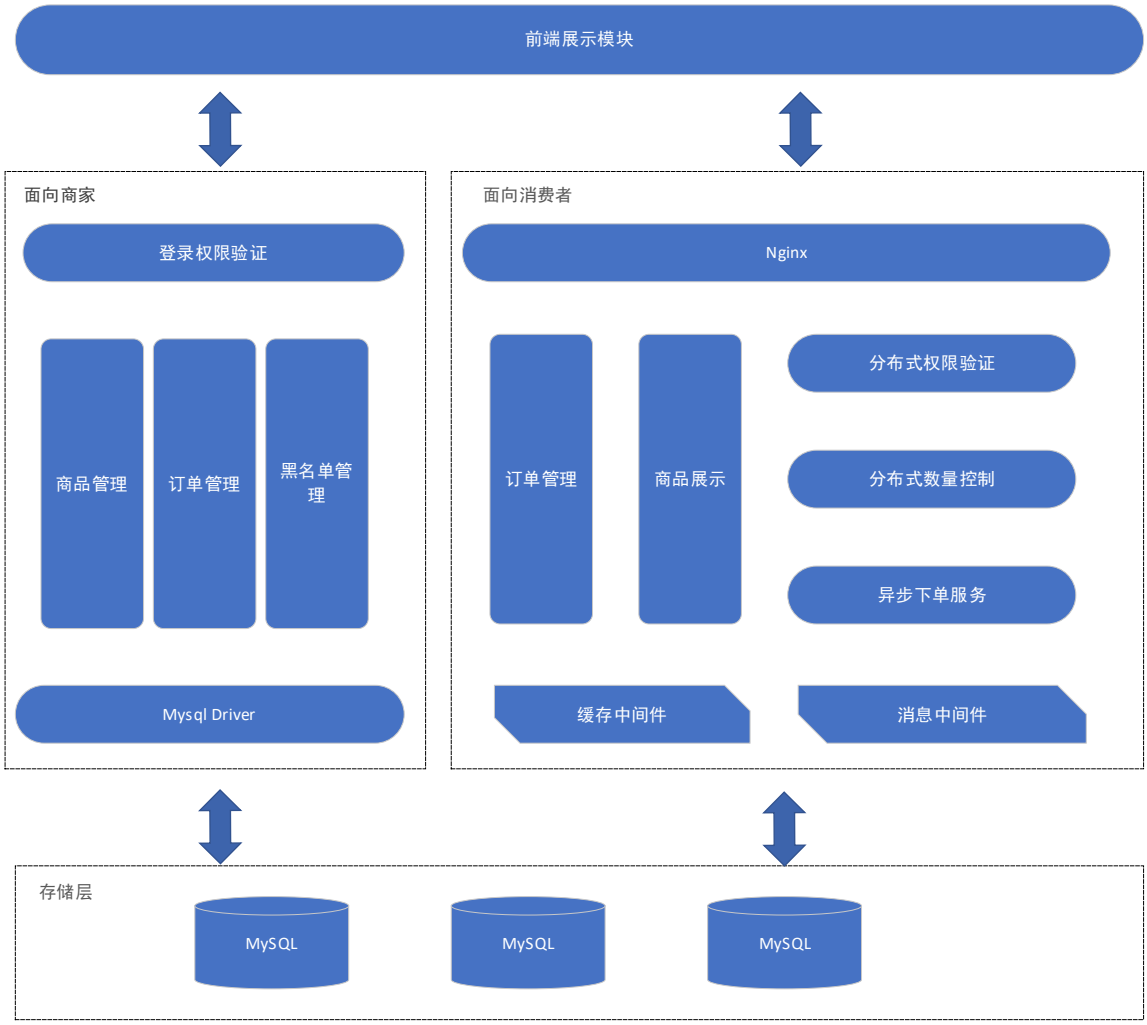

图 3.3 系统架构图

通过分析秒杀系统的功能性需求和非功能性需求，秒杀系统的总体架构主要分为面向商家的秒杀系统后台架构和面向消费者的秒杀系统。进一步对系统的服务划分和功能模块化，我们能够对系统的整体框架以及服务之间的层次结构有清晰的了解，如图 3.3 所示，进一步将需求模型转化为物理架构视图。

前端展示模块中，主要使用到 bootstrap 框架，boostrap 框架是一个用于快速

开发 Web 应用程序和网站的前端框架，其包含了贯穿于整个库的移动设备优先的样式并能够自适应于台式机，平板电脑和手机。前端界面与后端的交互上，在秒杀系统后台主要使用 golang 的 template 库作为生成页面的模板引擎，模板引擎通过将数据和模板组合在一起生成最中的 HTML。而对于面向消费者的前端展示则进行了动静分离，即对静态内容与动态请求分离，主要通过前端的 Ajax 异步调用回去后端的数据，进行 HTTP 协议的通信。

在秒杀系统后台的架构设计中，使用到 beego 框架，beego 是一个典型的 MVC 框架，通过将业务逻辑，数据与界面显示分离的方法来组织代码，从而减少编码时间。为保障不同商家的安全权限和数据隔离，每次请求流量都需要进行登录权限的验证，对非法流量进行拦截过滤。后台服务主要的业务模块包括商品管理，订单管理，黑名单管理，其中商品管理用于平台商家对秒杀商品的上传和管理，订单管理则是商家可以查看消费者已抢购订单的状态是否付款，商家可选择修改状态为已发货，黑名单管理主要是保障商家的权益，用于对部分恶意客户的屏蔽，商家可以选择对黑名单进行管理。

考虑到秒杀系统面向消费者会在秒杀开始的瞬间承受巨大的并发量和请求流量，因此在架构设计上考虑集群分布式服务来支持突发流量。用户在秒杀开始前为保证不会错过秒杀，会不断的刷新页面，为防止这些不必要的请求增加应用服务器和数据库服务器的压力，可以将商品页面内容静态化存储至 CDN。同样，Nginx 作为高性能的 HTTP 服务器和反向代理服务器，可作为秒杀系统的网关层，用于流量的负载均衡以及静态资源的代理缓存服务器。面向消费者的秒杀系统主要提供订单管理，商品展示和抢购秒杀的服务，其中，订单管理用于消费者对订单的删除和支付已改变订单状态持久化至数据库，商品展示主要是对秒杀商品的展示，秒杀商品的信息属于多读少写的数据，因此通过对这写数据缓存至缓存中间件，如 redis 和 memcache 中能够大大提升系统的性能和用户的响应时间。抢购秒杀服务则是秒杀系统的核心服务，为保证每个秒杀用户的公平性和良好的体验以及系统的稳定性，需要分层设计下单秒杀服务，对于传统的互联网业务，属于直通型业务即用户请求 1：1 的洞穿到 db 层，但在秒杀业务中需要从筛选过滤出极少量有限的成功请求，因此设计成漏斗型业务，即用户的请求从客户端到 db 层，层层递减，直到最后到 db 层的请求在

个位数量级。在设计漏斗型的系统架构中，秒杀系统会通过分布式权限验证用于对大部分无效恶意流量的过滤，分布式数量控制则进一步限流并用于对商品库存的数量控制而不发生超卖，异步下单服务则将大量的同步任务异步化进入消息中间件中，最后再逐个的持久化至数据库中。

3.3.2 持久化模型设计

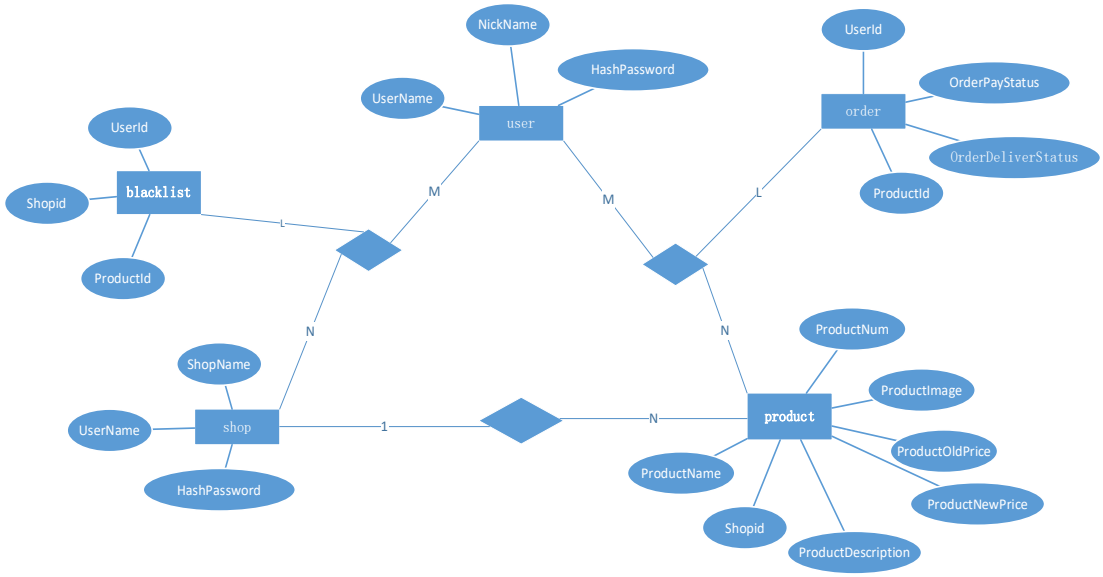

图 3.4 秒杀系统 ER 图

如图 3.5 所示为秒杀系统中的持久化对象模型关系图，主要的业务对象有 user, shop, balcklist, order, product 这几个对象，其中 user 表示用户，属性包括用户昵称，用户名，密码等；shop 表示商家，属性包括店铺名，用户名，密码等；product 表示商品，属性包括商品名，商品图片，商品旧价格，商品新价格，商品描述，商品秒杀开始时间，商品秒杀结束时间等；balcklist 表示黑名单，属性主要对应商家，商品，用户；order 表示订单，属性包括用户，商品，以及订单的付款和发货状态。

3.4 小结

本章首先对秒杀系统整体进行了概述，对秒杀系统的场景，涉众以及功能目的有了初步了解，进一步对系统功能性分析和非功能性分析并加以总结，将需求模型联系技术方案转化为系统架构图，为接下来的系统设计与实现做好充实的准备。

第四章 详细设计与实现

4.1 秒杀系统后台设计与实现

4.1.1 总体功能模块

图 4.1 秒杀系统后台架构图

秒杀系统后台总体功能模块如图 4.1 所示，整体使用使用 MVC 设计思想，前端展示模块使用 template 模板引擎生成 HTML 页面，业务层中 Router 用于对 HTTP 请求的多路复用进行路由分发至不同的 Controller，middleware 模块用于贯穿整个业务的中间组件服务，可以对权限验证和报错的日志记录，业务层的 Controller 主要是对不同的 RESTful 请求做出响应，返回 JSON 数据或 HTML 静态页面，Service 则是对系统整体服务的管理，包括对数据的加解密和清洗等服务，Repository 主要使用 go-sql-driver 驱动 mysql 与数据库存储层进行操作。

4.1.2 登录注册模块详细设计与实现

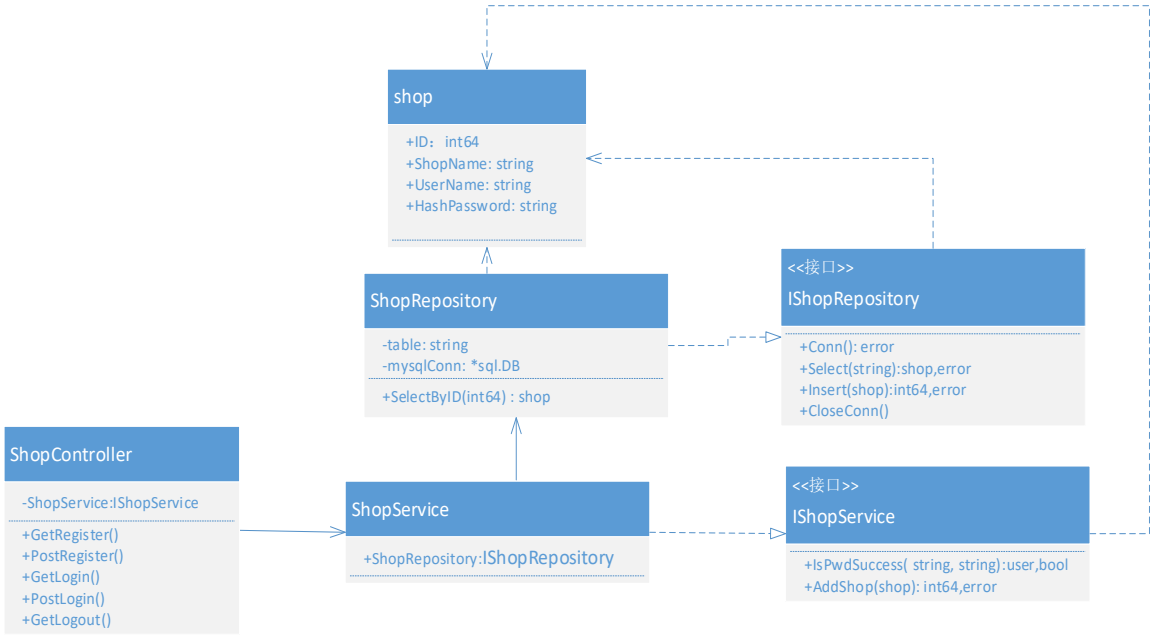

图 4.2 商家登录注册模块类图

商家登录注册模块类图如图 4.2，方法函数名遵循 RESTful 的命名规则，主要以 GET,POST 作为前缀，其中 ShopController 主要响应商家相关的请求，包括 GetRegister(),GetLogin() 分别用于返回商家的登录和注册静态页面，PostRegister()中调用了 ShopService 的 AddShop()，用于商家账户的添加，PostLogin()中调用了 ShopService 的 IsPwsSuccess()，用于对商家账号和密码的确认。

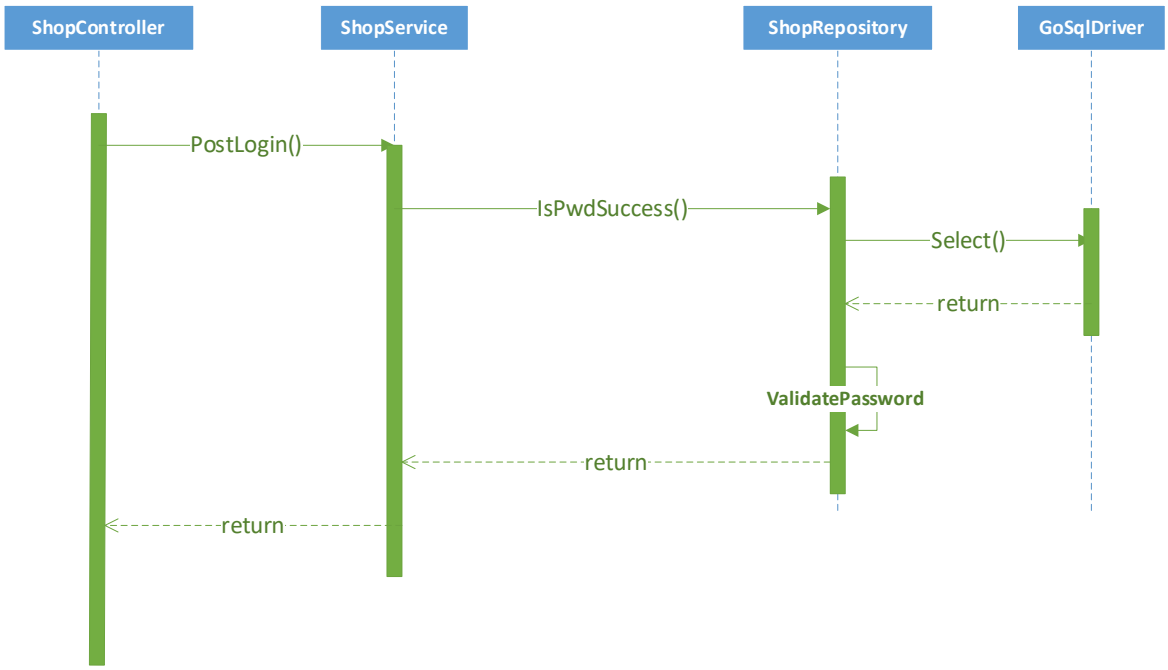

图 4.3 商家登录注册模块顺序图

4.1.3 商品管理模块详细设计与实现

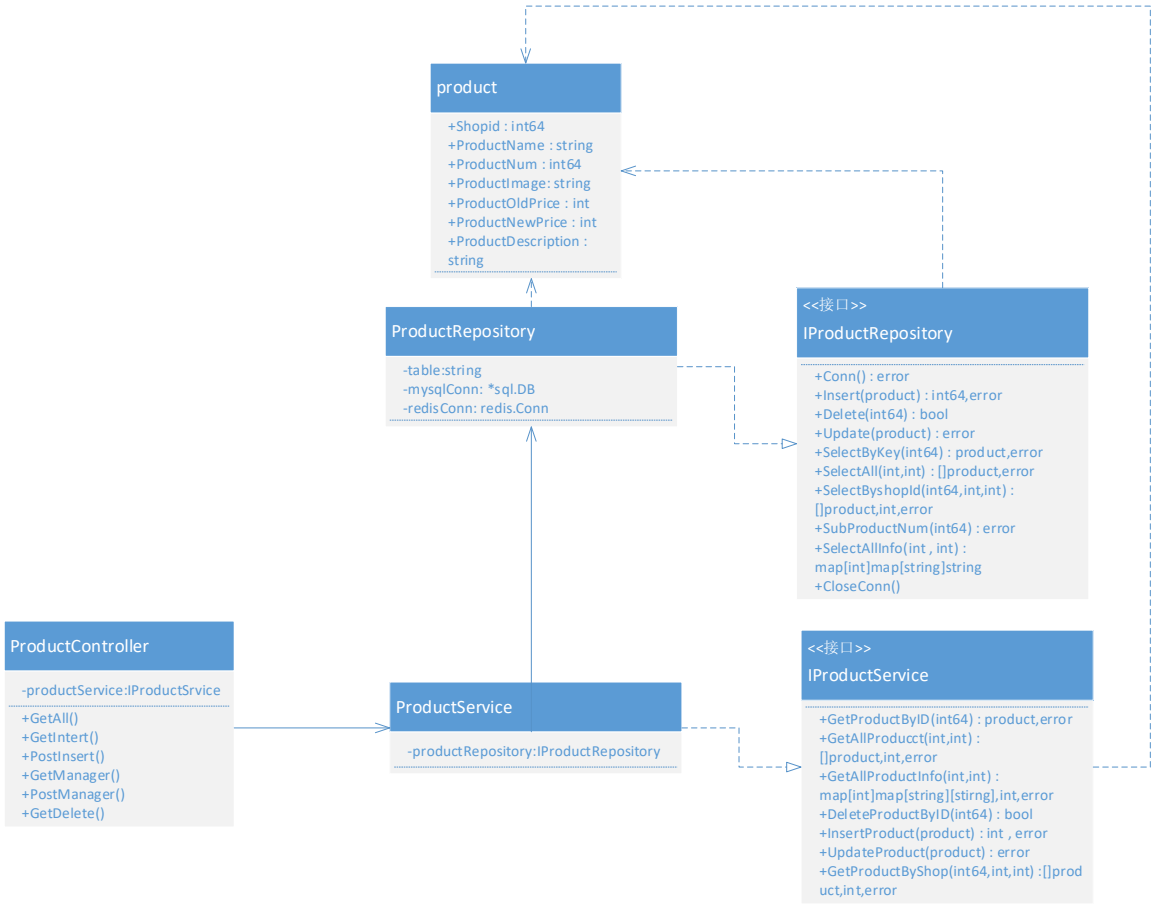

图 4.4 商品管理模块核心类图

商品管理模块类图如图 4.4，方法函数名遵循 RESTful 的命名规则，主要以 GET,POST 作为前缀，其中 ProductContorllers 主要用于响应商品管理相关的请求，包括 GetAll()调用 ProductService 的 GetProductByShop()获取商家已上架的所有商品；GetInsert(),GetManager()分别获取商家的新增和管理界面；PostManager(),GetDelete()调用 ProductService 的 UPdateProduct(),DeleteProduct()方法进行商品详细信息的修改。

IProductService 接口定义了 GetProductByID(),GetAllProduct(),GetProductInfo(),DeleteProductByID(),InsertProduct(),UpdateProduct(),GetProductByshop() 分别表示通过商品 ID 获取商品，指定页面获取所有的商品，指定页面获取所有商品的详情信息包括商品的所属商家，删除指定商品，更新指定商品，获取指定商家的所有商品。

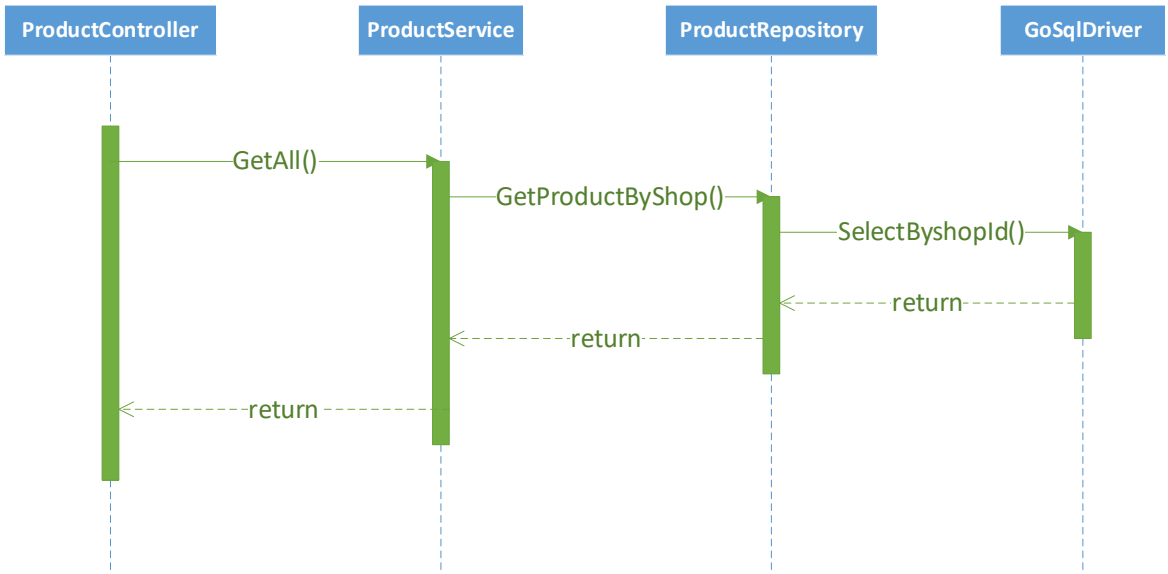

图 4.5 商品管理模块顺序图

4.1.4 订单管理模块详细设计与实现

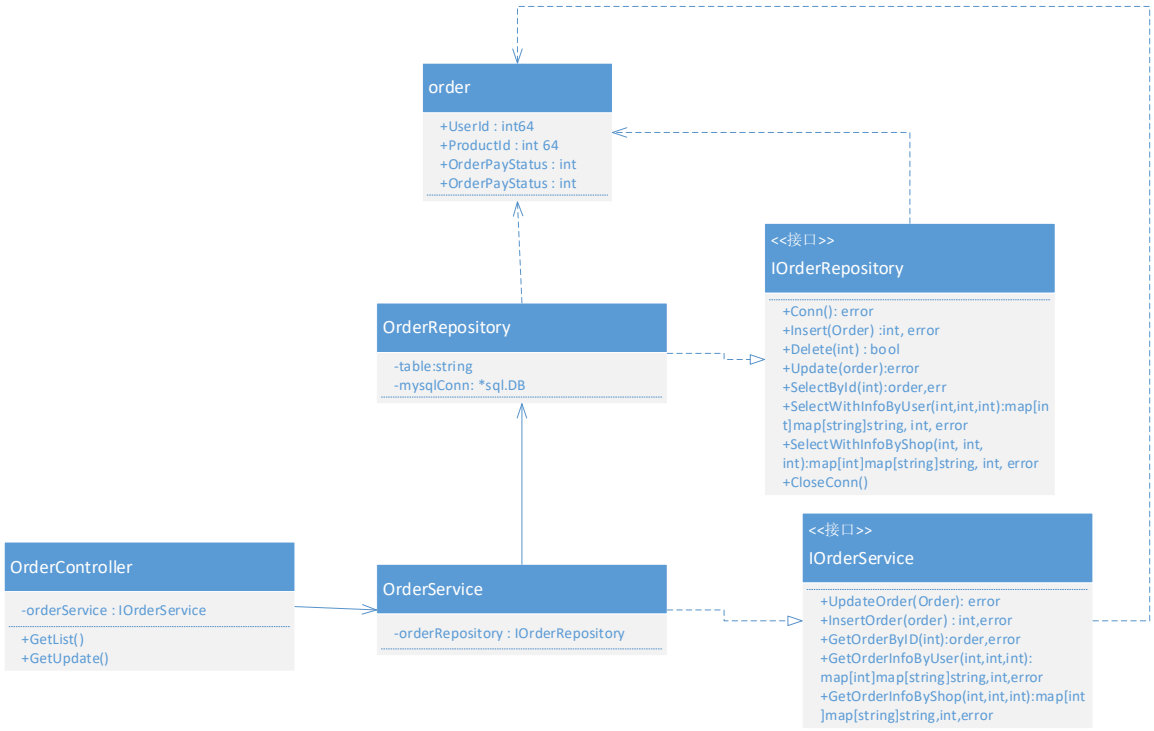

图 4.6 订单管理模块核心类图

订单管理模块类图如图 4.6，方法函数名遵循 RESTful 的命名规则，主要以 GET,POST 作为前缀，其中 OrderController 中 GetList()和 GetUPdate()分别调用 OrderService 的 GetOrderInfoByShop()和 GetOrderByID(),表示获得商家的所有订单以及更新某个订单的状态。

IOrderService 接口定义了 UpdateOrder(),InsertOrder(),GetOrderByID(),GetOrderInfoByUser(),GetOrderInfoByShop(),InsertOrderByMessage()分别表示更新指定订单，添加新的订单，获取指定 ID 的订单，获取指定用户的所有订单，获取指定商家的所有订单，通过消息添加订单。

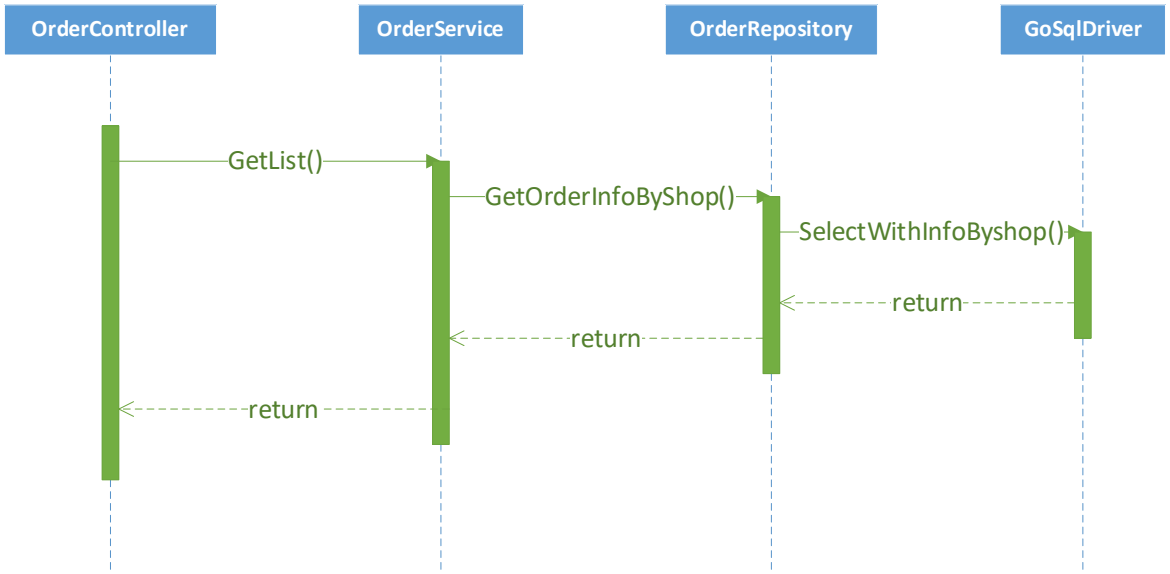

图 4.7 订单管理模块顺序图

4.1.5 黑名单管理模块详细设计与实现

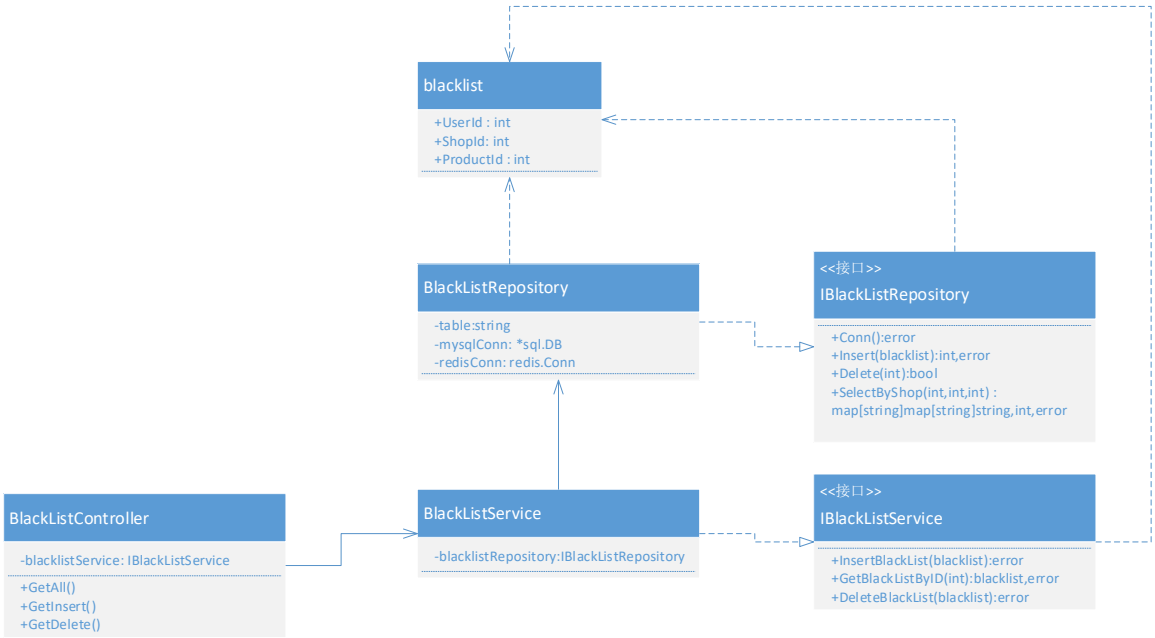

图 4.8 黑名单管理模块顺序图

黑名单管理模块类图如图 4.8，方法函数名遵循 RESTful 的命名规则，主要以 GET,POST 作为前缀,其中 BlackListContorllers 主要用于响应商品管理相关的请求，包括 GetAll()调用 BlackListService 的 GetBlackListByShop()获取商家已加入黑名单的所有用户;GetInsert(),GetDelete()分别获取黑名单的新增和管理界面；BlackListManager()调用 BlackLisService 的 DeleteBlackList()方法进行对指定黑

名单用户的删除

IBlackLisService 接口定义了 InsertBlackList(),GetBlackListByID(),DeleteBlackList（）分别表示添加黑名单用户，获取商家的所有黑名单用户，删除黑名单用户。

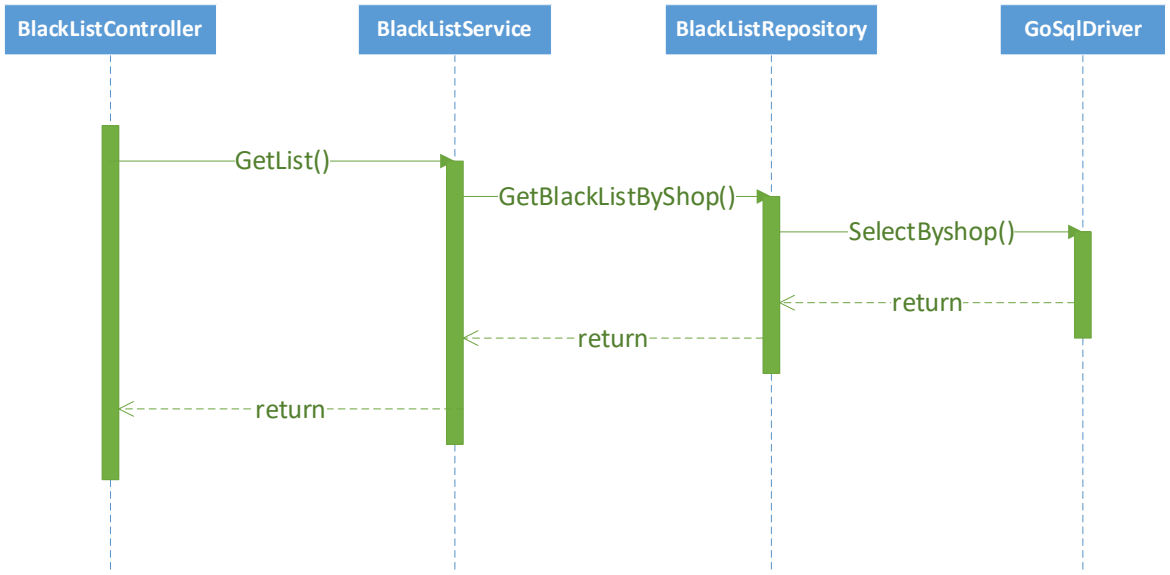

图 4.9 黑名单管理模块顺序图

4.2 面向消费者秒杀系统设计与实现

4.2.1 总体功能模块

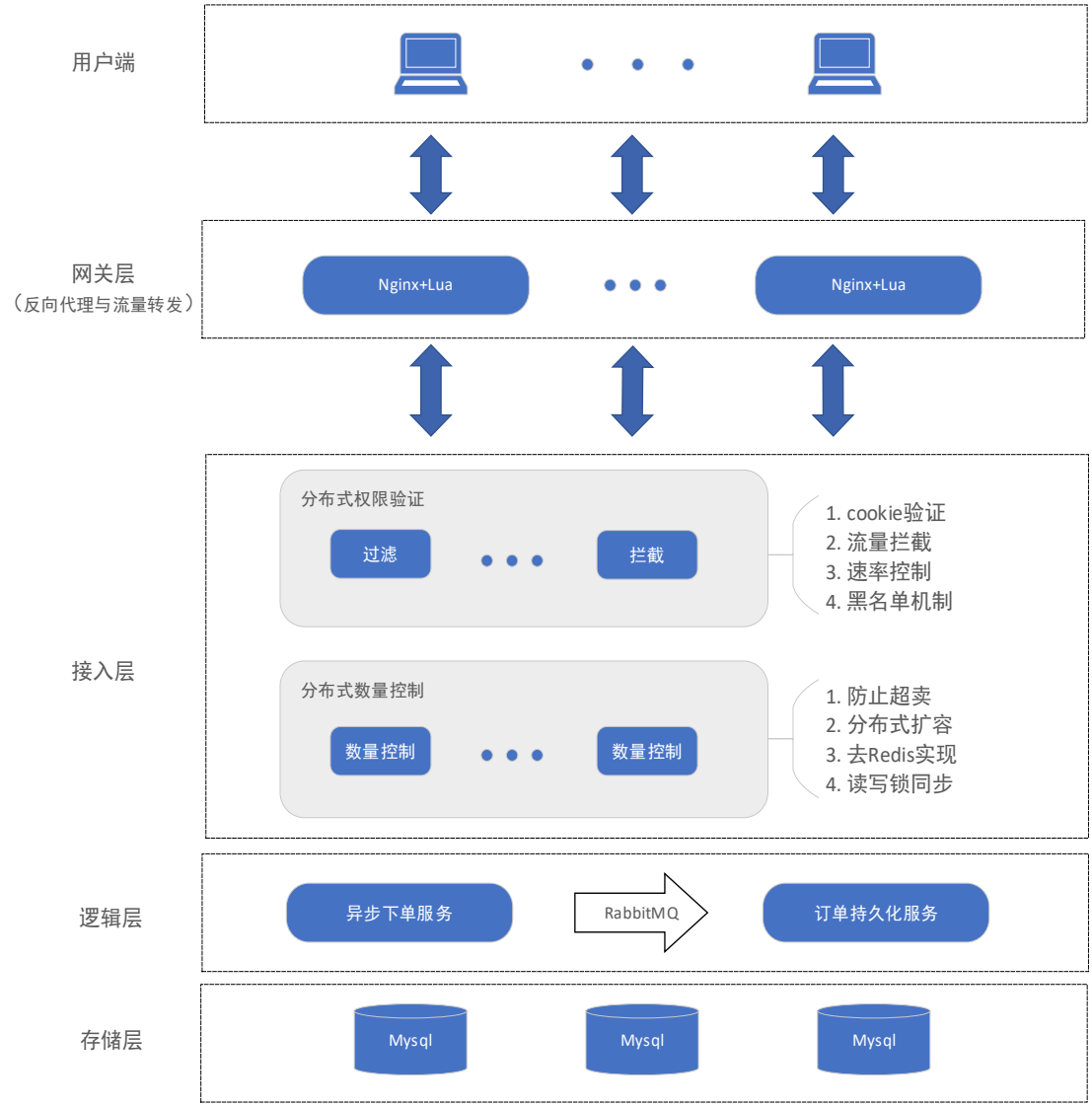

图 4.10 面向消费者的秒杀系统架构设计

结合对面向消费者的秒杀系统功能需求分析以及对系统有高并发，高可用，伸缩性和安全性等非功能需求，进一步设计出面向消费者的系统架构图，如图 4.10，网关层设置 Nginx 代理服务器进行反向代理和流量转发，使用 Nginx 进行负载均衡使用随机分配算法转发流量至接入层。接入层主要用于对大流量的过滤拦截，其中权限验证可用于 cookie 验证，对恶意流量的拦截以及实现黑名单机制；数量控制主要提供数量控制的保护，防止商品库存的超卖问题。为实现接入层的可伸缩性及分布式扩容，通过一致性 hash 算法来确保不同的商品的库存扣减能够到达固定的主机，并且能够通过主机间的代理来让动态伸缩扩容。最后真正减下库存的请求会进入业务逻辑层，将大量的同步请求异步化，即生产订单消息至 RabbitMQ 消息队列，在消息队列的消费端会有订单持久化服务对成功的订单持久化至 MySQL 数据库中。

4.2.2 权限验证模块

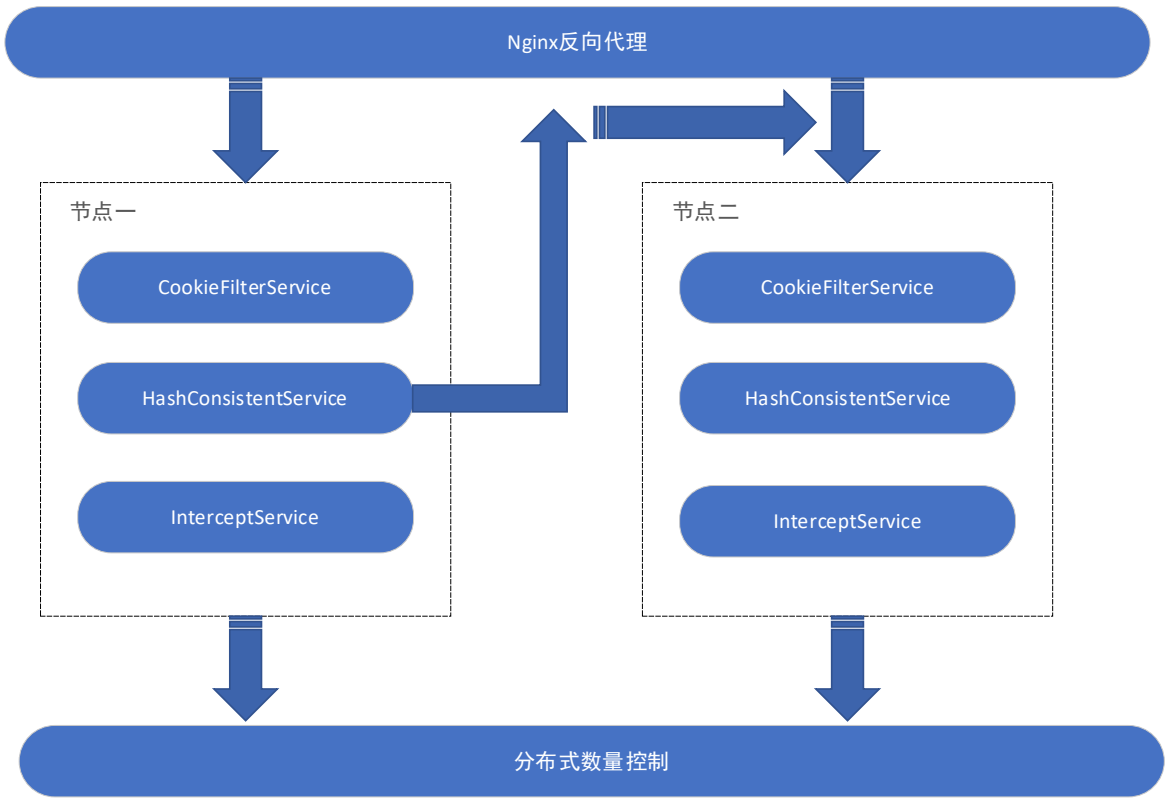

图 4.11 权限验证模块设计

权限验证模块主要提供三个服务，即 CookieFilterService，HashConsisitentService 和 InterceptService,分别是 Cookie 验证服务，一致性 Hash 节点服务，自定义拦截服务。

CookieFilterService:Cookie 验证使用 cookie 的双重加解密来代替 session，降低了分布式 session 的可能带来的不一致性以及消耗存储空间的复杂度，因此只要对 Cookie 进行私钥的加密并携带身份凭证，当用户请求秒杀抢购的时候需要带上加密数据和身份凭证，当 cookie 验证失败提前返回用户失败的结果响应。

HashConsistentService:实现一致性 hash 算法来对每个集群节点 ip 和用户 uid 的映射，因此每次用户的请求平均分配到指定节点并且每次会请求同一节点。并且一致性 Hash 算法更易于分布式节点的扩容,添加或删除节点对整个系统的影响偏低。HashConsistentService 主要用来判断当前请求的用户 uid 是否对于本机节点，若不对应则作为代理转发至正确的节点。

InterceptService：主要用于自定义系统的拦截规则，如黑名单用户，限购一

件。当通过了所有的拦截规则，用户请求则会进入分布式数量控制模块，进行异步下单的竞争。

其中实现一个过滤中间件核心组件的代码实现，如下：

```
func (f *Filter) Handle(webHandle WebHandle) func(rw http.ResponseWriter, r *http.Request) {
    return func(rw http.ResponseWriter, r *http.Request) {
        for path, handle := range f.filterMap {
            if strings.Contains(r.RequestURI, path) {
                //执行拦截业务逻辑
                err := handle(rw, r)
                if err != nil {
                    rw.Write([]byte(err.Error()))
                    return
                }
                //跳出循环
                break
            }
        }
        //执行正常注册的函数
        webHandle(rw, r)
    }
}
```

4.2.3 数量控制模块

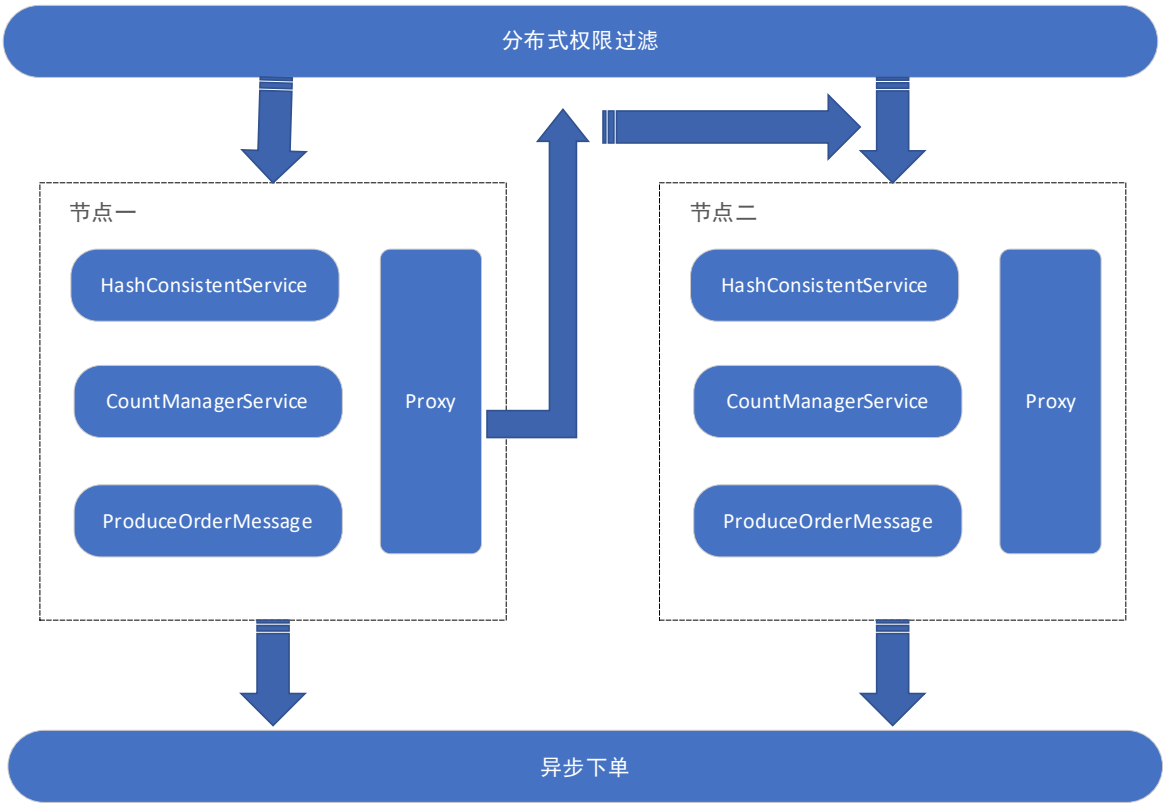

图 4.12 数量控制模块设计

为防止秒杀商品数量库存发生超卖，应当严格保证数据的一致性。秒杀系统对于秒杀商品数量控制模块设计如图 4.12，其中主要提供 4 个服务:HashConsistentService,CountManagerService,ProduceOrderMessage,Proxy 即一致性 Hash 节点服务，数量控制服务，生产订单消息服务以及代理节点转发功能。

HashConsistentService：现一致性 hash 算法来对每个集群节点 ip 和用户 uid 的映射，因此每次用户的请求平均分配到指定节点并且每次会请求同一节点。并且一致性 Hash 算法更易于分布式节点的扩容,添加或删除节点对整个系统的影响偏低。HashConsistentService 主要用来判断当前请求的用户 uid 是否对于本机节点，若不对应则作为代理转发至正确的节点。

CountManagerService: 数量控制服务，为保证并发条件下的数据一致性需要采取同步手段，这里使用 golang 提供的读写锁机制，当某个商品的数量库存发生变化时会进行写锁即排斥其他的线程，直到该线程释放了写锁。该服务对外暴露出两个方法，即 SetCount(),getOne()分别用于设置带秒杀商品的库存数和下单秒杀服务。数量控制服务采取本地缓存机制，即库存的数据都存储在服务的内存中，并且通过同步锁方式严格控制了数据不会发生超卖。

ProduceOrderMessage: 生产订单消息并与 RabbitMQ 主机通信，此后订单消息会在 RabbitMQ 的消息队列中存在，完成后无需等待是否持久化订单直接返回秒杀成功的结果，起到了将大量的并发同步竞争异步化，极大的提高了系统并发量。

其中核心的代码实现，如下：

```
func Proxy(host string, request *http.Request, url string) bool {
    uidPre, err := request.Cookie("uid")
    uidSign, err := request.Cookie("sign")
    client := &http.Client{}
    req, err := http.NewRequest("GET", url, nil)
    cookieUid := &http.Cookie{Name: "uid", Value: uidPre.Value, Path: "/"}
    cookieSign := &http.Cookie{Name: "sign", Value: uidSign.Value, Path:
"/"}
    req.AddCookie(cookieUid)
    req.AddCookie(cookieSign)
    response, err := client.Do(req)
    body, err := ioutil.ReadAll(response.Body)
    if response.StatusCode == 200 {
        if string(body) == "true" {
            return true
        } else {
            return false
        }
    }
    return false
}
```

4.2.4 异步下单模块

RabbitMQ 作为目前应用相当广泛的消息中间件，在企业级应用，微服务应用中给充当着重要的角色，特别是在一些典型的应用场景以及业务模块中具有重要的作用，比如业务模块解耦，异步通信，高并发限流，超时延迟处理等均有广泛的应用。

在设计异步下单模块时考虑到使用 RabbitMQ 作为消息中间件，主要用于将大量

的并发流量异步化，当最终抢购成功的请求到达异步下单模块时会生产一个带有订单信息的消息。在消息队列的另一端会一直阻塞等待从异步队列中取出订单消息，golang 中通过获取一个 channel 管道能够每取出一个订单消息变开启一个事务，执行库存的减一和订单生成的原子性操作，如果失败则回滚。

其中核心的代码实现，如下：

```
go func() {
        for d := range msgs {
            message := &Message{}
            err := json.Unmarshal([]byte(d.Body), message)
            tx, e := db.Begin()
            if e != nil {
                updateSQL := "UPDATE `product` SET productNum = productNum -1
WHERE ID = ?"
                _, e2 := tx.Exec(updateSQL, message.ProductID)
                insertSQL := "INSERT `order` set
userID=?,productID=?,orderPayStatus=?,orderDeliverStatus=?"
                _, e1 := tx.Exec(insertSQL, message.UserID,
message.ProductID, datamodels.PayWait, datamodels.DeliverWait)
                if e1 != nil || e2 != nil {
                    tx.Rollback()
                } else {
                    tx.Commit()
                }
            }
            d.Ack(false)
        }
    }()
```

4.3 小结

本章主要分别对面向平台商家的秒杀系统和面向消费者的秒杀系统做了详细设计与实现。面向商家的秒杀系统后台主要实现商家的管理模块，包括商品管理模块，订单管理模块，黑名单管理模块，以及登录注册模块。后台实现使用到了 beego 框架，一款高性能的 mvc 框架，其中前端框架使用 bootstrap 和 jquery 展示动态效果的网页，前后端的交互通过 golang 的 tmplate 模板引擎从数据库获取数据后生成 HTML 静态文件。面向消费者的秒杀系统分为四层，分别为 Nginx 网关层用于流量的负载均衡；接入层实现可以分布式扩容的服务层，用于权限拦截和数量控制；业务逻辑层用于异步化订单消息至 RabbitMQ 对列中，另一端用于持久化订单入数据库；存储层通过 MYSQL 事务特性保障数据的严格一致性。

第五章 系统测试与分析

5.1 测试环境部署

针对秒杀系统的测试主要从两个方面进行测试，即秒杀系统的功能性测试，对秒杀系统后台与面向消费者的秒杀系统的功能需求测试；秒杀系统的非功能性测试，主要对秒杀系统的单机性能测试，以及验证集群高并发测试和系统可横向扩展测试。

5.2 秒杀系统测试

5.2.1 秒杀系统功能性测试

表 5.2 秒杀系统后台测试用例

测试名称	测试用例描述	结果描述
OrderList_Test	查看订单列表	正常
OrderUpdate_Test	修改指定订单状态	正常
ProductAdd_Test	添加秒杀商品	正常
ProductUpdate_Test	修改秒杀商品	正常
ProductList_Test	查看秒杀商品列表	正常
ProductRemove_Test	删除指定秒杀商品	正常
BlacklistAdd_Test	添加黑名单用户	正常
BlacklistRemove_Test	删除黑名单用户	正常
BlacklistAll_Test	查看黑名单用户	正常

对照秒杀系统后台的功能需求分析用例，以上对秒杀系统后台做出了以上 9 个测试，其结果均为正常运行。

表 5.3 面向消费者测试用例

测试名称	测试用例描述	结果描述
ProductListAll_Test	获取所有的秒杀商品	正常
ProductItem_Test	获取指定商品详细信息	正常

OrderListAll_Test	获取所有的订单信息	正常
OrderRemove_Test	删除指定的订单	正常
OrderPay_Test	付款指定的订单（修改状态）	正常
SeckillNormal_Test	正常状态下的秒杀	正常
SeckillLost_Test	库存不足状态下的秒杀	返回秒杀失败，库存不足
SeckillBlacklist_Test	被记入黑名单下的秒杀	返回秒杀失败，黑名单用户

对照秒杀系统后台的功能需求分析用例，以上对面向消费者的秒杀系统做出了8个测试用例，其中对正常状态下的秒杀，库存不足状态下的秒杀，被进入黑名单下的秒杀做出了测试，结果分别为正常，秒杀失败，秒杀失败。

5.2.2 秒杀系统分布式集群测试

因实验环境的限制，为对本项目进行分布式集群测试能够更加直观体现整体系统的并发能力，对秒杀系统的限制过滤级别设置为最低，以及为避免硬件带来的误差，为每个服务主机配置相同的配置，即 CPU : Intel Core i5-400 CPU @ 2.80GHz，硬盘： 50GB ,内存 ：2GB，同一局域网网络环境。

通过使用 Apache JMeter 压测工具进行了四组测试，统计得到系统并发数，服务器节点 CPU 平均利用率，平均响应时间三类数据。

表 3.4 测试一数据

并发数	单台服务器	平均响应时间
1W	98%	8ms

测试一：仅使用一台服务器，MySQL 与 Go 应用同时向外提供服务，测试结果如表 5.4.

表 3.5 测试二数据

并发数	Nginx服务器	Web服务器	数据库服务器	缓存服务器	平均响应时间
4W	50%	60%	50%	20%	7ms

测试二：使用四台服务器，一台 Nginx 主机 两台服务主机， 一台 Mysql 服务主机 一台缓存主机。

表 3.6 测试三数据

并发数	消息中间件服务器	Nginx服务器	Web服务器	数据库服务器	缓存服务器	平均响应时间
10W	20%	50%	70%	60%	40%	6ms

测试三：使用九台服务器，2 台 Nginx 主机，四台服务主机，一台 Mysql 服务主机， 一台缓存主机， 一台消息中间件主机。

通过以上三组测试的结果可以得出，不断地增加服务节点的数量和并发数，可以看出系统并发的承载能力不断提高，系统能够横向扩展，分布式扩容来提升整体的并发能力。

5.3 小结

本章主要对秒杀系统进行功能性测试与非功能性测试，功能性测试对秒杀系统后台与面向消费者的秒杀系统的功能性需求用例进行逐个测试，结果显示正常。对于非功能性需求主要是对系统性能以及分布式集群扩展的测试，测试结果显示正常。

第六章 总结与展望

6.1 总结

秒杀活动相比普通的网络零售，秒杀活动中商品价格低廉能够吸引更多的网民进行有限商品的抢购，商家通过电商平台对秒杀商品的大幅推广并定时上架，用户则会在短时间瞬时抢购完部分展示商品。

基于 Goroutine 的高并发秒杀系统主要研究两个方向，其一是满足秒杀活动的基本业务需求，包括面向商家的秒杀系统后台以及面向消费者的秒杀系统。其二是研究分布式集群扩展高并发，对于单机的并发无法满足秒杀活动的瞬时高并发的特点，因此通过分布式集群来提高系统整体的性能以及可用性。

面向商家的秒杀系统后台主要实现商家的管理模块，包括商品管理模块，订单管理模块，黑名单管理模块，以及登录注册模块。后台实现使用到了 beego 框架，一款高性能的 mvc 框架，其中前端框架使用 bootstrap 和 jquery 展示动态效果的网页，前后端的交互通过 golang 的 tmplate 模板引擎从数据库获取数据后生成 HTML 静态文件。面向消费者的秒杀系统分为四层，分别为 Nginx 网关层用于流量的负载均衡；接入层实现可以分布式扩容的服务层，用于权限拦截和数量控制；业务逻辑层用于异步化订单消息至 RabbitMQ 对列中，另一端用于持久化订单入数据库；存储层通过 MYSQL 事务特性保障数据的严格一致性。

6.2 展望

秒杀系统的普遍性依然在，挑战性也会越来越高，本项目在设计的初衷主要对高并发进行初探，从中学到了关于 Golang 天然支持并发的高效编程语言，也学到高性能的中间件以及分布式集群的架构设计。项目依然存在不足的地方，对于分布式也有很大的可学习的领域，如将单体应用走向服务化，微服务架构，主机间的远程服务调用，服务监控与治理等。

参考文献

[1] 刘书健. 基于协程的高并发的分析与研究[D].昆明理工大学,2016.

[2] 刘磊.一种高并发电商秒杀系统的设计与实现[J].现代计算机(专业版),2019(02):95-100.

[3] 王宽. 基于 SOA 的高并发 B2B2C 电商平台设计与实现[D].兰州大学,2019.

[4] 朱丽叶.面向电商平台的秒杀系统设计与实现[D],上海交通大学,2018.

[5] 刘昆鑫,卜庆凯.基于 SSM 框架的 WEB 系统秒杀优化设计[J].青岛大学学报(工程技术版),2017,32(04):114-119.

[6] L., F. Design and Implementation of Super Mall System Based on SOA Distributed Architecture. in 2019 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS). 2019.

[7] J., Y., et al. Message-oriented Middleware: A Review. in 2019 5th International Conference on Big Data Computing and Communications (BIGCOM). 2019.

[8] N., D. and L. J. An Empirical Study of Messaging Passing Concurrency in Go Projects. in 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). 2019.

[9] 刘书健,基于协程的高并发的分析与研究[D],昆明理工大学,2016.

[10] 陈猛.基于 Java 的购物网站设计与开发[J].农家参谋,2020(08):200.

[11] 张宝祥,何利力.高并发集群系统下的负载均衡技术研究[J].工业控制计算机,2017,30(10):76-77+138.

[12] 王瑛.基于 Java 应用的高并发高可用集群服务器的设计与实现[J].电子技术与软件工程,2019(20):139-140.

[13] 米沃奇.探秘电商秒杀抢购背后的 IT 架构[J].电脑知识与技术(经验技巧),2016(12):107-109.

[14] 张会敏.电商网站秒杀系统的研究与对策[J].科技与创新,2017(22):75-76.

[15] 陈文楷. 基于 docker 容器的高并发 web 系统架构设计与实现[D].北京邮电大学,2019.

[16] 涂腾飞. Go 语言中的并发问题研究[D].北京邮电大学,2019.

[17] 陈伟颖,李艳平,翟玥.基于 goroutine 的 web 并发编程的研究与应用[J].电脑知识与技术,2015,11(33):52-54.

[18] 郭志伟. 面向高并发的分布式购物平台设计与实现[D].浙江工商大学,2019.

[19] 杨安.基于 Web 集群的反向代理负载均衡研究与实践[J].信息通信,2019(11):13+31.

[20] 曹思远. 基于 Node.js 高性能高并发网络应用构架的研究和实现[D].杭州电子科技大学,2018.

谢 辞

时光荏苒，岁月如梭，时间过的悄无声息，大学期间的求学生涯就要结束了，在这四年的时间里，无论是专业知识还是人际交往，自己都得到了成长。虽然有很多不舍，但是很清楚人生的路还有很长，求学的道路没有止境，大学只是一阶段，在这里我遇到了很多负责任的老师，优秀的小伙伴帮助了我很多，对此表示由衷的感谢。感谢经纬度团队，在团队两年时间从一个渴望知识的小白到带领团队的队长，期间学到的东西是整个大学最宝贵的东西，感谢团队的师兄师姐对我们新生的关照，感谢团队的老师给我们的指导和教育，感谢团队的伙伴一起学习与奋斗。其次要感谢的是毕设的指导老师，李萌老师，感谢老师能细心陪伴我们度过最后的大学生涯，耐心给我们教导，指导完成毕设项目。李萌老师亦师亦友，对生活充满乐观，为人幽默，知识渊博，勇于创新，细心严谨的做事风格深深感染到了我，让我在今后的学习中披荆斩棘，乐观前行。最后要感谢南华大学计算机学院，给予我们浓厚的学习环境和教育`