首都经济贸易大学
CAPITAL UNIVERSITY OF ECONOMICS AND BUSINESS

专业硕士学位论文

基于 Django 的电商秒杀系统的设计与实现

培养单位：管理工程学院

专业名称：软件工程

作者姓名：陈世宽

指导教师：张军 教授

Design and Implementation of High E-commerce Seckill System Based on Django

Candidate：Chen Shikuan

Supervisor：Prof. Zhang Jun

Capital University of Economics and Business, Beijing, China

独创性声明

本人郑重声明：所呈交的论文是本人在指导教师指导下独立进行研究工作所取得的成果，论文中有关资料和数据是实事求是的。尽我所知，除文中已经加以标注和致谢外，本论文不包含其他人已经发表或撰写的研究成果，也不包含本人或他人为获得首都经济贸易大学或其它教育机构的学位或学历证书而使用过的材料。与我一同工作的同志对研究所做的任何贡献均已在论文中作出了明确的说明。

若有不实之处，本人愿意承担相关法律责任。

学位论文作者签名：陈世宽　　日期：2020 年 6 月 5 日

关于论文使用授权的说明

本人完全同意首都经济贸易大学有权使用本学位论文（包括但不限于其印刷版和电子版），使用方式包括但不限于：保留学位论文，按规定向国家有关部门（机构）送交学位论文，以学术交流为目的赠送和交换学位论文，允许学位论文被查阅、借阅和复印，将学位论文的全部或部分内容编入有关数据库进行检索，采用影印、缩印或其他复制手段保存学位论文。

保密学位论文在解密后的使用授权同上。

学位论文作者签名：陈世宽　　日期：2020 年 6 月 5 日

指导教师签名：　　日期：2020 年 6 月 5 日

摘要

随着互联网的发展，网上购物已经成为最受人们欢迎的购物方式之一，而在线秒杀活动作为一种在线促销方式也越来越受到人们的欢迎。秒杀活动具有一定的时限性，需要在规定的时间内参与商品的抢购。参与秒杀活动的商品通常低于市场价格，数量有限，售完即止，所以秒杀活动极大刺激了消费者的购买欲望。

M 公司是一家传统的电商公司，现在需要上线一套秒杀系统来满足新的需求。M 公司对秒杀系统提出如下要求：第一，该系统必须满足公司和消费者的功能和非功能需求。第二，考虑到参加秒杀活动人数具有不可预期性，系统应该具有良好的可扩展性和可配置性。第三，由于秒杀活动具瞬时高并发特点，所以秒杀系统应支持高并发。针对秒杀活动的特点，该秒杀系统应在满足公司和消费者功能性需求的同时，也应该具满足高并发、高可用和可扩展的非功能性需求。

本文以上述系统为研究对象，根据秒杀系统的业务场景和特点以及 M 公司的实际需求，设计并实现了一套秒杀系统，主要研究内容如下：第一，对商家和消费者进行详细的功能和非功能性需求分析，第二，在对商家的需求分析的基础上，设计并实现了面向公司管理人员的后台管理系统，第三，在对消费者的需求分析的基础上，设计并实现了面向消费者的前台秒杀系统。本系统使用的解决方案主要包含如下内容：使用 Nginx+keepalive 技术实现高可用、高效的流量分发网关；以 Django 框架为基础，实现分布式集群的搭建；在代码流程设计中充分考虑流量控制方案，尽可能将无效流量拦截在上游，减少对缓存和数据库的压力，保证系统高效；使用 redis sentinel 模式，实现具有读写分离、错误故障自动转移的高可用缓存层。

关键词：秒杀系统；高并发；缓存；分布式

Abstract

With the development of the Internet, online shopping has become one of the most popular ways of shopping, and online seckilling as an online promotion is more and more popular.Seckill activity has a certain time limit, need to participate in the goods within the specified time.The product that participates in seckill activity is below market price commonly, quantity is limited, sell out namely, so seckill activity stimulates the desire of consumer to buy greatly.

M company is a traditional e-commerce company, and now it needs to launch a seckilling system to meet the new demand.M company put forward the following requirements for the seckilling system: first, the system must meet the functional and non-functional needs of the company and consumers.Second, considering the unpredictability of the number of participants, the system should have good scalability and configurability.Thirdly, because of the characteristics of instantaneous high concurrency, the second killing system should support high concurrency.According to the characteristics of seckilling activities, the seckilling system should not only meet the functional requirements of companies and consumers, but also meet the non-functional requirements of high concurrency, high availability and scalability.

This article takes the above system as the research object, and designs and implements a set of spike system based on the business scenarios and characteristics of the spike system and the actual needs of company M. The main research content is as follows: First, detailed functions and business Non-functional requirements analysis. Second, based on the analysis of the needs of the business, the design and implementation of a background management system for the company's management personnel. Third, based on the analysis of the needs of consumers, the design and implementation of Consumer-oriented front-end spike system. The solution used in this system mainly includes the following: using Nginx + keepalive technology to achieve a highly available and efficient traffic distribution gateway; based on the Django framework to implement the construction of a distributed cluster; fully considering the flow control scheme in the code flow design, Intercept invalid traffic upstream as much as possible, reduce the pressure on the cache and database, and ensure the efficiency of the system; use the redis sentinel mode to achieve a highly available cache layer with read-write separation and error failover.

Keywords: spike system; high concurrency; cache; distributed

目录

第 1 章　绪论

1.1　项目背景

得益于科技的发展和互联网的普及，电子商务作为一种方便、快捷和不受地域限制的线上的购物方式已经逐渐普及[1]。由于，近年来电子商务公司的用户量和交易规模不断快速增加，取得了突破性的增长，对传统商务造成巨大的冲击。传统的实体店售卖模式已经不适合新时代下的公司发展，传统公司需要依托规模庞大的互联网用户群体实现新一轮的消费升级[2]。

商品秒杀活动作为电子商务主要的促销方式之一，是电子商务一种重要的营销方式[3]。秒杀活动具有时限性，会在指定的时间开始和结束，所以参与的用户必须在指定的时间进行抢购商品。参与秒杀活动的商品必须要具备以下两点属性：一、该商品比较稀有，数量有限。二、商品价格比平时低，对消费者具有很大的吸引力。综上可知，秒杀活动会吸引大量用户在规定的时间段内、在指定的网站页面内抢购一定数量的商品。通常情况下，消费者对商品的请求购买数量一般会远远大于商品库存数量，而且可以进行秒杀的时间段很短。因此在秒杀时，大量用户会在同一时间内集中抢购商品，向服务器发送大量查询和购买请求，这样会造成网站访问流量瞬时成百上千倍甚至上万倍的激增，对服务器和数据库产生极大的压力，而且所有购买请求中只有少数请求购买成功，因此秒杀系统是一个典型的高并发应用[4]。

面对访问量瞬时成千上百甚至上万倍的激增，如何保证网站服务器和数据库能够正常有效的提供服务，在高并发的情况下，如何做到对可能出现的故障实现自动修复。这些问题对系统的建设者提出了新的挑战。对于拥有雄厚资金的大型电商公司来说可以投入大量的人力物力去逐个攻克这些难题。例如，淘宝网为了解决存储问题和提高系统性能自主研发了数据存储系统、缓存系统和负载均衡系统，同时为了加快请求响应速度，部署了几十个 CDN 节点，据不完全统计，仅用于生成淘宝首页的服务器就有上千台[5]。如此庞大的硬件环境需要的成本也是巨大的，如此高的成本很难应用于普通中小企业。

现在M公司为了迎合当下发展趋势以及扩大知名度增加营业额，需要上线一套在线秒杀系统以满足当前用户的使用。目前该公司有一套自己的电商平台，由于现有的电商系统在设计初期没有考虑后期业务扩展，对现有系统改造难度大、不经济需要改造部分比较多，同时秒杀系统具有瞬时高并发特点，造成服务器压力较大，容易出现故障。为了避免秒杀系统出现故障从而对整个系统造成影响，所以需要独立开发部署。综合考虑改造成本远大于重新开发一套独立的系统。由于商家资金有限，不希望在没有产生经济效益前投入太多成本来购买硬件设备。所以要在资金有限的情况下设计一套支持高并发、高可用和可扩展的秒杀系统。

1.2　研究目标

本文在对目前秒杀系统所设计到的技术要点进行详细分析和研究，结合 M 公司的实际需求，设计并实现一套具有高并发、高可用和可扩展的秒杀系统[6]。该系统需要满足以下要求：

（1）满足秒杀业务需求

该系统需要同时满足管理人员对后台管理的需求和面向消费者的前台秒杀需求。管理人员可以策划发布秒杀活动，对秒杀商品和会员进行管理，对于异常账号进行禁用处理等。用户可以查看秒杀活动，参与秒杀活动进行下单、支付和订单的取消等操作。

（2）支持高并发、高可用

秒杀活动是对一些受欢迎的商品以低于平时的价格定时出售活动。从而给系统带来的问题就是在某一段时间内系统的瞬时访问量会非常高，这也就对秒杀系统的并发性和可用性提出了较高的要求[7]。并发性不好的系统会出现反应慢、卡顿等现象，会给用户带来不良的体验。可用性不好的系统会出现卡死、宕机等不能正常提供服务的情况。因此为了满足消费者的秒杀需求，系统应满足以下两点以实现高并发：第一，系统对任何请求的反应时间不应该超过 3 秒。第二，系统应当支持最高每秒 10000 次的并发量。为了满足高可用性系统应该满足以下两点：第一、秒杀系统要在 24 小时运行的情况下，保证秒杀活动结果出错概率小于 1%[8]。第二，系统应该具有一定的自我保护能力，可以应对一些异常流量的攻击。

（3）实现可扩展的分布式服务

在秒杀活动进行时，网站的瞬时访问量会成倍增加，所以秒杀系统一般采用可扩展的分布式系统进行搭建，通过多台服务器组成一个集群来处理所有请求，这样就减轻单台服务器的压力了。但是参与秒杀活动的客户人数是不可预期的，因此整个秒杀系统应该具有随着需要可以灵活配置的能力。在参与秒杀活动人数少，集群服务器利用率低时，可以减少集群机器数量，避免造成资源闲置。同时在现有的集群不能满足不断增长的消费者数量时，整个网站会出现卡顿、反应迟缓和秒杀成功率低等情况，这时可以通过增加机器数量的方式来提高网站的响应速度和提高秒杀成功率。所以系统应该还具有良好的可扩展性，集群对集群数量的管理，可以通过修改尽可能少的代码来实现[9]。

1.3　研究内容

本文主要研究内容如下：

第一，对 M 公司需求进行详细分析，主要包括面向消费者秒杀系统的功能需求和非功能需求分析以及面向公司管理员后台系统的需求分析和非功能需求分析。

第二，在需求分析的基础上，对面向消费者秒杀系统进行设计与实现。首先进行总体设计，把整个系统分为客户端层、网关层、业务层、数据访问层和数据层。对每层进

行详细设计与实现。在网关层使用 Nginx+keepalived 方案，实现高可用的、高效的负载均衡，同时实现对恶意请求的拦截[10]。在业务层使用 Django 框架开发，搭配 supervisor 进程管理工具，实现了对业务层各个服务监控和自动化启动。同时在代码流程设计上实现请求拦截，尽量将无效请求拦截在上游，减轻对下游数据层的压力。数据访问层，使用 ORM 框架实现在数据库和对象之间做一个映射，通过操作 ORM 框架提供的接口就可直接操作数据库，避免和复杂的 SQL 语句打交道[11]。数据层，缓存集群采用 Redis sentinel 方案，实现读写分离和高可用的缓存集群，同时使用 MySql 和 MySQL-Proxy 中间件实现读写分离和高可用的数据库集群[12]。

第三，在需求分析的基础上，对面向管理人员的后台管理系统进行设计与实现。由于后台管理系统需求比较简单，大部分功能采用 Django 自带的后台管理系统插件实现，对于部分复杂业务单独进行开发。

第四，在完成系统开发后对整个系统进行测试。完成测试环境部署后先分别对面向消费者前台秒杀系统和面向管理人员的后台管理系统进行功能测试。完成功能测试后，对整个系统使用压力测试工具进行非功能测试，包括并发性测试、可用性测试，最后对系统测试结果进行总体分析。

1.4 本人工作总结

本人在实习阶段参与了该项目整个开发周期，从项目立项、产品调研、可行性分析、需求分析到系统的整体框架设计以及部分功能开发和测试的整个过程，对项目的现实背景和意义以及实现过程有深刻的了解。

在前期项目规划和调研时期，本人分别站在公司和用户的立场上，同用户和公司相关人员进行广泛深入讨论，了解到各自的需求和目前的痛点。在对双方的需求和痛点进行深入探讨分析的基础上，完成了市场调查报告和需求文档。在系统总体设计阶段，参考大量相关文献，完成了系统的总体架构设计。在参考同类系统的基础上，详细阐述本系统将要遇到的问题以及给出可能的解决方案，在对所有解决方案进行深入剖析后选出一个最优方案作为该问题的最终解决方案。在系统实现和测试阶段，完成了秒杀应用、个人中心应用、后台管理模块和数据库的详细设计和开发，最后对整个系统进行功能和非功能性测试。

1.5 本文的组织结构

第一章 绪论。说明课题研究的背景和意义，详细介绍了秒杀活动的背景和产生的问题，还包括论文的研究内容以及论文的整体结构设计。

第二章 系统相关技术介绍。这章将会对在开发秒杀系统过程中所使用开发技术进行详细分析和研究，对技术解决方案过程进行详细阐述。

第三章 系统需求分析。这章将会根据管理员和消费者的实际情况进行系统需求分析，包括功能性需求和非功能性需求。

第四章 面向消费者的秒杀系统设计与实现。本章在上章需求分析和参考同类系统的基础上，对面向消费者的前台系统进行总体设计，然后在对各层进行详细设计和实现。

第五章 对面向管理人员的后台管理系统进行详细设计与实现以及对数据库进行详细设计。

第六章 系统测试与分析。本章在系统完成后，为了保证系统符合要求，对系统各功能模块设计测试用例进行仿真测试，最后总结测试结果。

第七章 总结与展望。这章将会总结本系统的优势与不足以及结合现在流行的技术和实际需求进一步提出未来可改进的方案。

第 2 章　技术综述

本系统采用 Python 作为主要开发语言，使用 Django Web 框架进行模型管理、应用管理、数据库和缓存配置管理。使用 ORM 框架实现数据持久层的操作，Redis 实现分布式缓存，使用 MySql 数据库构建一套高并发、高可用、可扩展的分布式系统。

2.1　Django 框架

Django 是一个遵循模块化的、开源免费的 web 框架，其内部各个模块之间相互独立的同时也紧密相连，鼓励快速开发和 DRY（Do Not Repeat Yourselt）的原则，兼顾组件可重复使用以及组件可插拔等良好的设计原则[13]。Django 在整个设计规范中都体现了“高内聚，低耦合”模块化的设计思想。在横向上，沿用了 MVC 三层设计模式思想，形成自己的 MVT 设计规范，在纵向上，由多个独立的应用组成，应用之间通过方便灵活的路由系统来互相调用实现消息传递。

MVC 是一种架构设计模式，其核心理念是将软件的用户交互界面和逻辑模型分离并通过控制器实现界面和模型的数据同步更新，这种模式将软件分为三层：模型、视图和控制层，每一层都是相互独立的，同时每一层也会提供接口供上一层调用，对每层代码的修改不会影响到其它层，便于软件维护和代码重用[14]。MVC 的架构组成如图 2.1 所示。

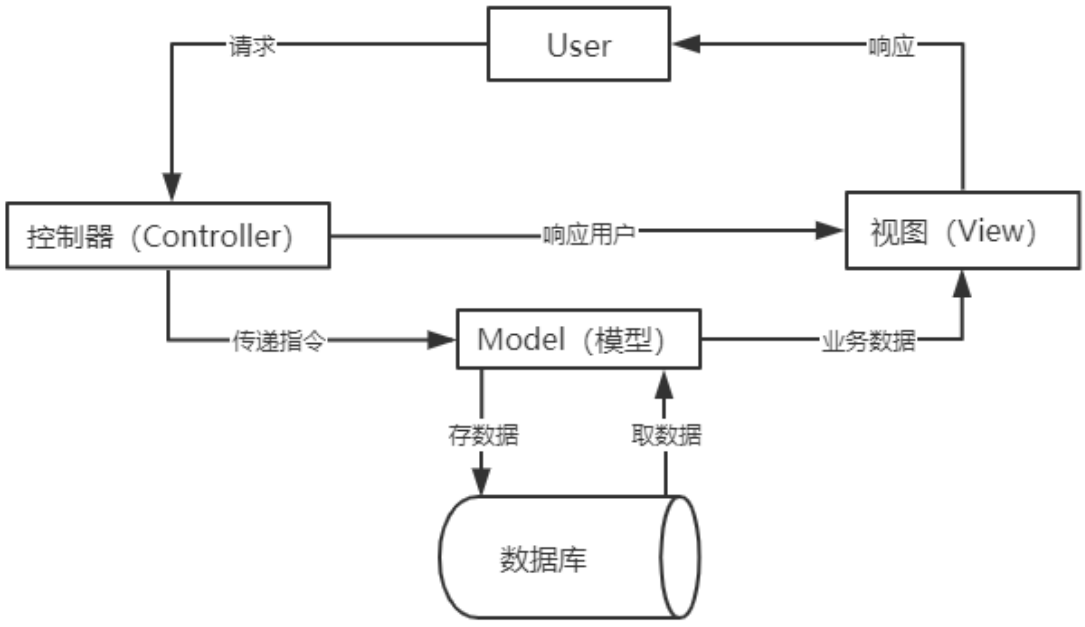

图 2.1　MVC 架构图

在 MVC 架构中，各个核心部件的主要功能如下：

（1）模型：对内负责与数据库交互，读写数据库中的数据，对外提供接口定义如何操作这些数据。

（2）视图：根据来自模型的数据，构建并向用户展示界面。

（3）控制器：连接视图和模型，对来自用户的请求进行处理，并按照需求调用指定的模型进行数据获取，之后调用指定视图对处理结果进行封装并将结果返回给用户。

MVT 采用了与 MVC 相似的架构模式，双方中的 M 的全拼为 Model，其功能相同都是与数据库交互，传输数据。不同之处在与 MVT 中的 V 和 MVC 中的 V 功能不同，前者与 MVC 中的 C 相似，负责按照业务逻辑对数据进行处理，而 MVT 中的 T 和 MVC 中的 V 功能相似，负责生成 html 文档，发送给客户端进行展示。

Django 框架还自带了一些功能优秀的组件，主要有：

（1）ORM 框架：对象关系映射框架，把关系型数据库和模型做一个关系映射，把对数据库的操作进行封装，提供一套功能完善的操作数据库的 API，使用者不再关注于具体的数据库操作，把精力投入到业务逻辑开发上面，大大提高了系统开发效率和可维护性 。

（2）admin 后台管理系统：功能强大的后台管理系统，通过与模型一一对应，可以实现对模型的增删改查以及过滤等操作，最终通过 ORM 对象关系映射框架实现客户端和数据库的数据同步。提高了系统的开发效率，避免在后台维护上花费过多的精力。

（3）灵活的路由系统（URL 映射）：通过正则表达式匹配 URL，将来自客户的请求与对应的视图进行匹配。

2.2　分布式缓存 Redis 技术

在进行秒杀活动时，大量请求涌入服务器，包括但不限于查看商品以及其库存、下单购买商品等请求，这些请求都会对数据库进行读或写的操作，如果不做一些措施，如此频繁的数据库操作会直接压垮数据库，所以需要引用缓存机制，把经常访问的热点数据加入缓存层，对数据的读写请求首先要在缓存中进行操作，由于缓存是基于内存的，运行效率远超传统的基于硬盘的数据库，这样实现把一大部分对数据库的请求转移到对缓存的请求，从而达到减轻数据库压力的目的[15]。本系统采用 Redis 来实现缓存层。

Redis 是用 C 语言开发的一个基于内存的高性能非关系型数据库[16]。相比于传统的关系型数据库它的运行效率更高，响应请求的速度更快，在一般机器上其读写速度可达 100000 次/秒[17]。Redis 如此高效的原因有以下几点：

（1）基于键值对数据存取方式，没有关系型数据库中关系的约束，而且大部分操作都是在内存中完成。

（2）支持的数据结构都是经过专门设计的，数据结构简单，易于操作。

（3）一般情况下 CPU 处理内存数据的速度要远高于网卡接收数据的速度，所以 CPU 完全可以满足 Redis 的需求，限制 Redis 性能的因素大多来自内存和网络带宽。因此，Redis 使用单线程处理所有网络请求，从而避免了线程切换以及锁相关问题所带来的额外开销。

（4）采用异步非阻塞 IO 模型，不在 IO 上浪费额外时间。

同时，为了满足不同的场景需要，Redis 提供了如下几种常用的模式[18]：

（1）单机模式

使用一台机器，部署一个 Redis 节点，如果出现宕机可能会导致所有连该接 Redis 缓存的服务失效，该模式不适合生产环境。

（2）主从模式

一般情况下该模式采用一个 master 节点和多个 slave 节点，该模式主要有两个优点：一是支持数据备份，所有写入 master 节点的数据都会自动备份到所有 slave 节点上。二是实现读写分离，所有对写请求都会在 master 节点上处理，读请求都会分配到 slave 节点上处理，减轻 master 节点压力，提高并发性能。但是该模式在 master 节点宕机不能提供服务的情况下，所有的写请求将不能处理。

（3）Sentinel 模式

为了解决主从模式中，master 节点宕机后整个缓存不能提供写服务的问题，引入 Sentinel 机制，Sentienl 通常翻译为哨兵。Sentinel 模式架构图如图 2.2 所示。

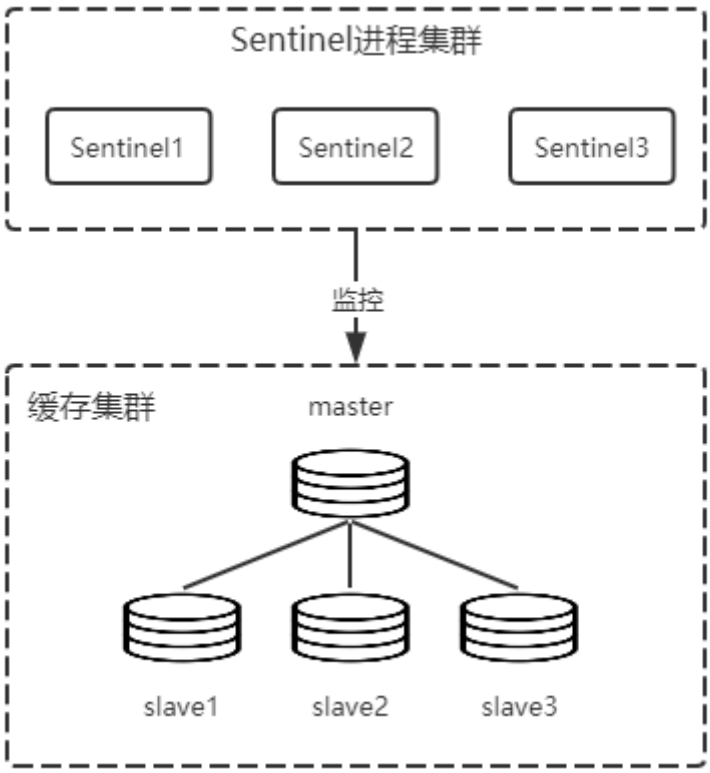

图 2.2　Sentinel 模式架构图

Sentinel 实现了对 Redis 集群中节点状态实时监控，在 master 节点出现故障后，在从 slave 节点中选取一个运行状态正常的 slave 节点代替原 master 节点。由于 Sentinel 服务本身也会出现故障，所以为了避免出现单点故障，需要设置多个 Sentinel 节点实现高可用的 Sentinel 集群。Sentinel 监控机制的原理是：首先 Sentinel 会通过心跳机制来确认 master 是否存活，如果 Sentinel 在规定时间内没有接收到或者接收到一个错误的对 Sentinel 的回应，那么这个 Sentinel 就认为这个 master 出现故障，当认为 master 出故障的 Sentinel 进程数量达到约定的数量后，Sentinel 集群按照规则从 slave 节点中选出一个作为 master 节点，从而实现故障的自动转移。

2.3　负载均衡技术介绍与研究

负载均衡是所有分布式架构设计所必须考虑的技术，该技术主要功能是将所有请求按照设定的算法均衡分配到集群中所有节点上，避免单台节点处理请求过多[19]。负载均衡技术是基于现有的网络，优化整个系统资源的使用，最大化吞吐量的同时最小化响应时间，从而使整个集群达到最优性能。

2.3.1　常用的负载均衡技术分析

（1）DNS 域名解析负载均衡

域名系统的 IP 分发是实现负载均衡的典型算法，其原理是在 DNS 服务器中为同一个域名设置多个 IP 地址，对于所有访问该域名的请求，DNS 服务器会以轮询算法将请求分配到不同 IP 的机器上去，从而实现请求的分流，达到负载均衡的目的[20]。例如很多大型网站都使用了这个技术，包括雅虎、新浪等站点。

DNS 负载均衡技术优点是可以跟根据用户 IP 来智能分配服务器，选择一个离用户最近的服务器为用户提供服务。该技术缺点：一是无法获知集群中服务器的状态，如果某一服务器宕机，所有分配到该服务器的请求就会没有相应。二是没有考虑集群中服务器性能不同的问题。轮询分发请求，会造成性能好的机器资源利用率低，性能差的机器长期处于超负荷状态运行。

（2）数据链路层（二层）负载均衡

在 OSI 协议中，数据链路层位于倒数第二层，所以在数据链路层实现的负载均衡称为二层负载均衡[21]。所有基于网络的设备都有唯一的物理地址即 MAC 地址，在数据链路层传输的数据帧都需要通过 MAC 地址找到对应的设备，从而实现数据的传递，故可以利用以上特点实现负载均衡。数据链路层负载均衡的实现原理主要是通过修改 MAC 地址来实现流量的分发，首先为负载均衡服务器设置一个对外的虚拟 IP，然后把集群中所有设备的 IP 设置为与负载均衡服务器相同的 IP，但是 MAC 地址必须保持唯一。当请求到达负载均衡服务器时， 的 MAC 地址实现请求的分发。在目标机器收到

来自于负载均衡服务器转发的请求后，由于整个集群和负载均衡服务器 IP 地址相同，故目标机器响应的数据包可以直接发送给客户端，不用再由负载均衡服务器转发，从而减轻负载均衡服务器的负担。

数据链路负载均衡技术优点是集群服务器对客户端的响应不用通过负载均衡服务器，解决了负载均衡服务器网络流量的问题。缺点是配置复杂，技术要求高，且不支持丰富的分发策略。

（3）四层负载均衡

四层负载均衡工作在 OSI 模型的倒数第四层传输层，该层有 TCP 和 UDP 两种协议，两种协议都基于 IP 地址和端口实现数据的传输。基于以上特点，四层负载均衡服务器在接收到来自客户端的数据包后，通过修改数据包中 IP 地址和端口号从而把流量分发到对应的服务器上，实现负载均衡。该种负载均衡优点是修改 IP 实现流量分发都在内核中完成，性能有很大的提高。缺点是请求和响应都会经过负载均衡服务器，因此会受到负载均衡服务器本身的性能和网卡带宽等因素限制。

（4）七层负载均衡

七层负载均衡工作在 OSI 模型的应用层，是在四层负载均衡基础上实现的，由于应用层协议比较多，也就意味着其中包含很多其它内容，因此实现负载均衡的手段比较灵活、也更加智能。比如实现负载均衡除了使用 IP 地址和端口外，可以根据 URL 把对图片的请求分配到静态服务器，把对文字的请求分发到文字相关的服务器等，另外还可以设置多重策略，过滤特定数据包等[22]。

2.3.2 负载均衡算法分析

负载均衡算法主要由如下几种[23]：

（1）轮询（Round-Robin）算法

把请求按顺序轮询的分配到集群中的服务器上，此方式均匀对待每一台服务器，不关心服务器的状态。轮询算法优点是逻辑和配置非常简单，且不用关心服务器状态，在服务器性能平均的情况下，速度非常快。缺点是忽视每个服务器的差异，可能造成配置低的服务器超负荷运行，响应迟缓，而配置较高的服务器可能长期处于低利用率运行状态。

（2）加权轮询（Weighted Round-Robin）算法

加权轮询算法解决了轮询算法忽视每个机器差异的问题。首先对每个服务器进行综合评价得到一组权值，权值大小代表服务器处理能力强弱，权值分值代表被分配请求次数的比例，权值越高的服务器被分配请求次数越多。

（3）源地址哈希（Hash）算法

源地址哈希算法根据请求中的源 IP 地址，使用 Hash 算法映射到服务器地址列表，获得集群中某一服务器的 IP 地址，实现负载均衡。该算法优点是对于具有相同源

地址的请求会被固定分配到集群中某一服务器，有效解决了分布式系统 Session 共享问题。缺点是如果某一服务器宕机，那么分配到该服务器的所有请求都会被拒绝，直到服务器重新启动或者服务器地址列表中删除该服务器地址。

2.4　Nginx

Nginx 是一款轻量级的 Web 服务器，可以实现反向代理和负载均衡服务器等[24]。在本系统中主要利用其三个功能，第一作为 web 服务器使用，第二反向代理功能[25]，第三提供软件的负载均衡功能。

所有来自用户的请求，首先由 Nginx 负载均衡服务器根据请求的 URL，把请求分发到业务处理服务或静态服务器上，这样有两个优点，第一可以隐藏真实的服务器资源，对于用户来说只需要知道负载服务器地址即可，不需要知道真实的服务器地址，这样就可以隐藏真实的服务器资源，保证服务器资源的安全性。第二，通过 Nginx 实现负载均衡，通过负载均衡算法将所有请求合理分发到所有的服务器上，避免单一服务器接收请求过多，压力过大导致宕机[26]。

第 3 章　秒杀系统需求分析

秒杀活动是一个典型的高并发场景，所以秒杀系统应该在满足基本的功能需求基础上还要对系统的并发性、可用性、安全性等非功能需求提出更高的要求[27]。秒杀系统需求根据面向人群不同可被分为秒杀系统前台和后台。秒杀系统前台主要面向消费者用户群体，秒杀系统后台主要面向公司管理员。

3.1　秒杀系统前台功能性需求

秒杀系统前台主要使用人员为消费者，消费者用例图如图 3.1 所示。

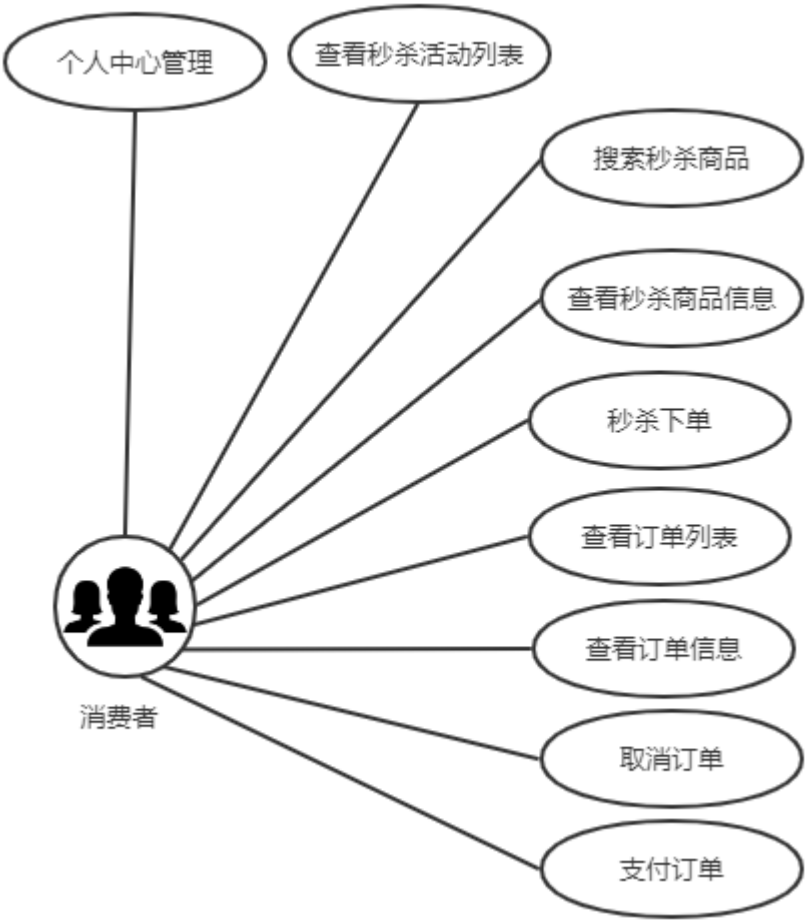

图 3.1　消费者用例图

（1）个人中心管理：个人中心管理主要包含两部分内容，用户的注册登录和对个人信息的管理。注册登录是系统的基础功能，注册需要输入邮箱、手机号码、密码等个人信息。完成注册后需要激活账号，才可以输入邮箱密码进行登录。个人信息管理主要是针对个人信息进行操作，包括头像修改，密码修改，地址修改等相关操作。

（2）查看秒杀活动列表：用户即可以查看不同时间段内的秒杀活动列表，也可以查

看包含不同类别商品的秒杀活动列表。

（3）搜索秒杀商品：用户可输入商品名称或关键字实现精确或模糊查询秒杀商品。

（4）查看秒杀商品信息：秒杀系统需要有商品详情页面用来展示商品的完整信息，包括商品的标题、图片、库存、价格、数量等各种参数。

（5）秒杀下单：系统要保证对于非黑名单用户,可以在秒杀活动进行的时间内进行下单操作。如果购买商品的数量满足商品允许可购买量并且商品库存数量大于等于购买量，则进行下单、生成订单。反之则下单失败并返回结果。

（6）查看订单列表：用户在完成商品秒杀，得到系统秒杀成功通知后，可以在订单列表中查看订单。

（7）查看订单信息：系统支持用户查看个人订单信息。订单信息包括订单的当前状态（未支付、已付款、已取消）、下单时间、商品编号等。

（8）取消订单：系统需要有取消订单功能，当用户取消订单后，相应商品的库存需要增加订单中购买商品的数量，同时修改订单状态为已取消。

（9）支付订单：当用户下单成功后，订单是未支付状态，用户需要对订单进行支付，完成支付后，订单状态信息需要修改为已支付。

3.2　秒杀系统前台非功能性需求

秒杀系统前台主要针对消费者，考虑到秒杀活动时间短，参与的消费者多以及参与商品价值高、价格低和对消费者具有很大吸引的特点，所以秒杀系统主要关注的非功能需求有：

（1）安全性：参与秒杀活动的商品必然是十分优惠且具有很大的商用价值，对消费者有巨大的吸引力，这也就会导致一些不法分子利用抢购软件进行恶意抢购行为的发生。这些恶意抢购行为会在短时间内向系统发送大量请求，如果不采取相应限制措施，这些请求会对系统造成极大压力，以至会宕机，所以秒杀系统必须要采取一些措施来限制这些恶意行为，同时还需要尽量防范抢购软件造成的 DDos 攻击[28]。

（2）高并发：秒杀系统是一个典型的高并发系统，在秒杀活动中，系统会处理大量请求，所以系统并发性能是必须要重点考虑的一个指标。系统应该在提供给消费者良好的购物体验的同时保证系统可以正常稳定运行。系统要保证用户的所有请求操作耗时不应超过 3 秒，系统必须在 3 秒内对请求作出反应，缓存读取操作不应超过 10 毫秒。

（3）易用性：系统设计尽量符合消费者使用习惯，界面设计简单易懂易操作；在关键操作或者不易理解地方要给出相应提示；对账号密码以及个人信息等要进行二次认证，以免重要信息错误；对复杂操作要提供相应的操作解释文档。

（4）高可靠性：系统应该从两方面来保证系统的高可靠性。首先在系统设计时要考虑到可能出现问题的情景，针对可能出现的问题做出相应的预防措施。另一方面在系统

出现问题后，要做到系统具有一定的自动修复问题的能力，例如：服务的重启，数据的恢复等[29]。

3.3 秒杀系统后台功能需求

秒杀系统后台主要针对管理员，用例图如图 3.2 所示。

图 3.2 管理员用例图

（1）用户管理：管理员可以增加新的用户、删除指定的用户、修改用户的信息以及查找用户。同时，管理员还可以实现黑名单用户的增加和删除。

（2）商品管理：管理员可以对商品进行增加、删除、修改和查找的操作。

（3）秒杀活动管理：管理员可以对秒杀活动进行增加、删除、修改和查找操作。

（4）订单管理：管理员可以对所有订单进行增加、删除、修改和查找操作。

（5）秒杀监控：管理员可以在秒杀活动进行时监控所有秒杀活动，发现异常活动可随时进行处理。

3.4 秒杀系统后台非功能性需求

秒杀系统后台主要是管理员使用，故面向的群体数量少、访问量不高，因而后台系统考虑的非功能需求有：

（1）安全性：管理员可以直接在后台进行操作，具有最高权限，所以后台的安全性是重中之重，所以要和普通用户作好权限隔离。同时要设计一套安全可靠的登录系统，防止注入攻击、暴力破解等非法登录手段。

（2）易用性：系统后台要保证界面对用户友好，操作简单，复杂部分需要有操作文档。

第 4 章　秒杀系统前台的设计与实现

本章在需求分析基础上进一步对系统前台进行设计与实现。包括系统总体架构设计、各层内的架构设计以及最后对业务层秒杀应用和个人中心应用进行详细的设计与实现。

4.1　系统架构设计

根据需求分析，对系统前台进行系统架构设计。

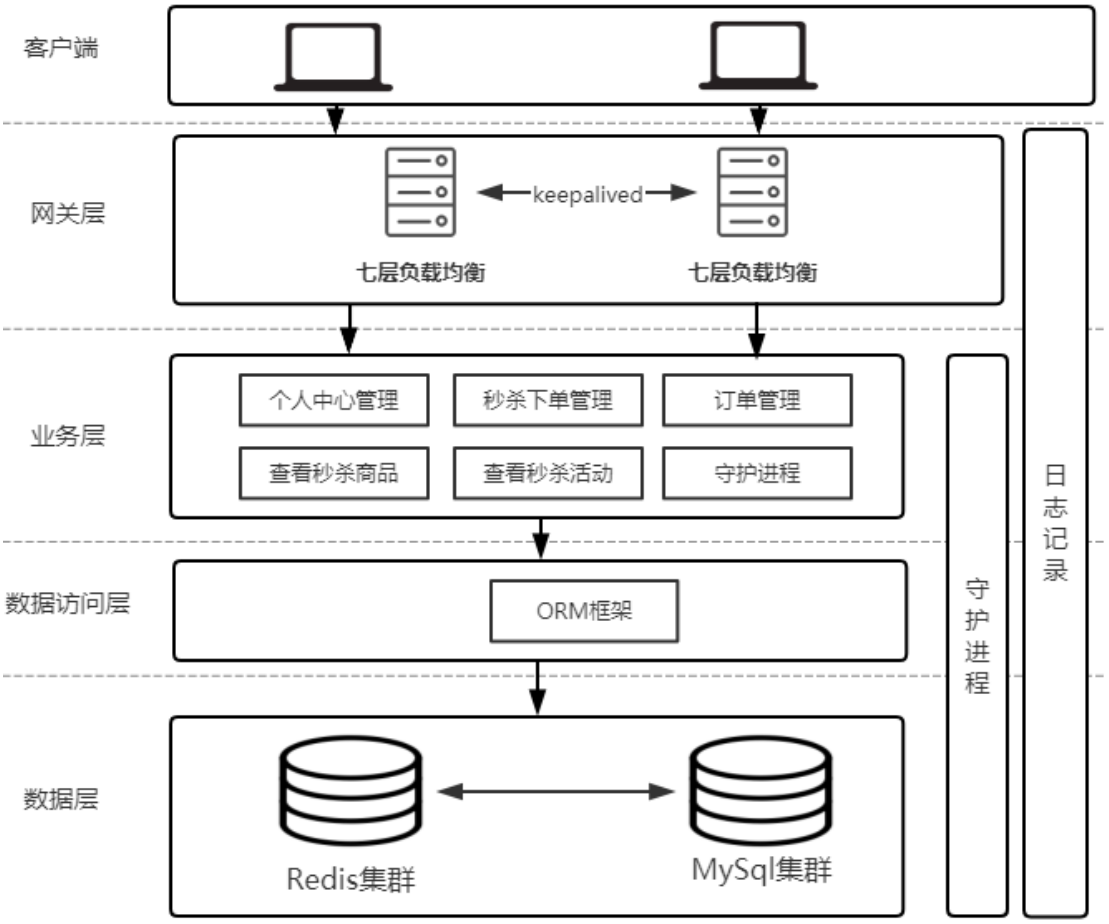

图 4.1　系统前台架构图

如图 4.1 所示是系统前台架构图，首先用户在客户端层通过浏览器向系统发送请求，请求首先经过网关层，网关层通过负载均衡技术把流量的合理均衡的分发到业务服务器，以保证整个系统的低延迟和高吞吐量。请求在业务层通过数据访问层实现和数据库的交互，完成对数据的逻辑处理。

4.1.1　网关层架构设计

网关层主要是负载均衡服务器管理请求分发，是整个系统的门户，网关层设计的好坏对整个系统有重大影响，故设计过程中应该遵循以下原则：

（1）安全性原则：安全性对于整个系统来说是必须首先考虑的，如果系统不能保证安全性，那么整个系统是没有意义的。在设计过程中无论是算法还是技术产品的选择都要首先考虑到安全性。为了保证负载均衡服务器的安全性必须保证两点：一是要保证负载均衡器自身的高可用性，在出现故障时有一套完整可靠的故障处理流程，具有一定的故障修复能力，保证整个服务器可以长时间稳定正常运行。二是要保证自身有一定抵御外界攻击的能力，可以对异常流量进行限制等。

（2）可扩展性原则：参与秒杀活动的人数具有不可预测性，在参与人数过少时，集群服务器利用率低，应减少硬件资源以作他用，避免资源的浪费。在消费者数量过多时会，集群服务器会长期处于高负荷运行状态，容易造成服务器反应过慢，在现在有的服务器集群不能满足现有请求压力时，在网关层可以通过配置来增加节点数量来解决压力过大问题。综上，网关层的设计应具有良好的扩展性，在对现有系统以最小的改动基础上实现集群节点的增加或减少。

（3）高效性和稳定性：负载均衡服务器承担着系统请求分发的作用，是整个网站的门户，所以为了保证负载均衡服务器的在性能上必须要高效，以快速实现流量的分发，以满足整个系统性能的要求。同时，要保证服务器在请求量激增时仍保持稳定高效的正确运行。

综上网关层的整体架构进行设计，架构图如图 4.2 所示。

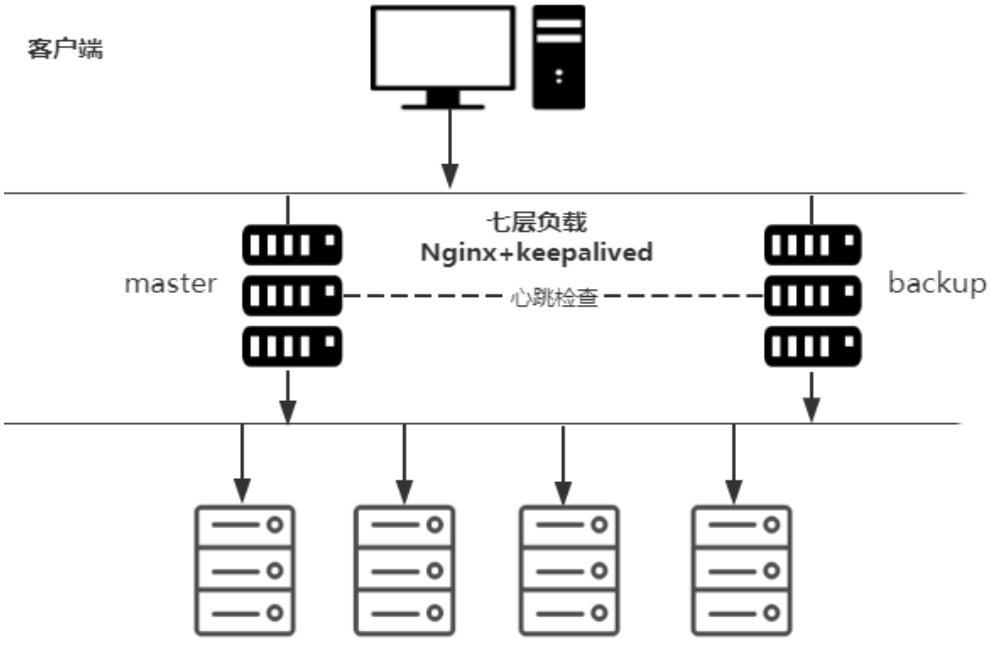

图 4.2　网关架构图

根据以往经验，预测系统最高并发量不会超过 10000 次/秒，同时 Nginx 最大并发量可以高达 50000 次/秒[30]，所以 Nginx 完全可以承担器负载均衡的作用。虽然有很多种负载均衡技术，但是为了减轻业务服务器的压力，实现动态服务器和静态服务器分离，所以负载均衡服务器需要根据请求的 URL，来决定把请求分发到业务服务器还是静态服务器，所以本系统使用七层负载均衡技术。

本系统选择加权轮询算法作为负载均衡器分发请求的算法，该算法以权值高低作为分配几率，这样不能保证来自同一个客户端的请求分配到集群中同一个节点，这样就会造成 Session 同步问题。当某一客户端第一次访问负载均衡服务器时，其请求被分发到集群中某一节点上，然后在该节点上创建 Session，当该客户端再次访问负载均衡服务器时，其请求可能被分发到集群中其它节点上，由于此时该节点没有对应于该客户端的 Session 信息，因此会认为该客户端是第一次请求，可能会拒绝提供服务。这种 Session 同步问题在集群服务器中是一个关键的问题。为了解决该问题，需要把整个集群中所有节点设计为无状态的，即任一节点不保存上下文信息，集群内服务器状态一致，任意一台节点处理请求得到的结果是相同的。本系统采用 Session 数据集中存储方案，该方案不把 Session 存放到单个节点上，而是把所有 Session 数据集中存放到缓存中统一管理，所有服务器都应该从服务器中获取 Session。这样不论那一台那一个节点，修改哪一个 Session 都会在缓存中更新，这样就保证了 Session 同步[31]。

一台 Nginx 负载均衡器容易产生单点故障，造成整个系统的不能正常运行的后果。为了保证整个 web 集群高可用性，防止单点故障造负载均衡服务器不能工作，系统使用 keepalived 技术来解决这个问题。首先准备两台负载均衡服务器，一台为 master 服务器，正常情况下该服务器对外提供一个 VIP（虚拟 IP），提供负载均衡服务，另外一台为 backup 服务器，正常情况下备份服务器处于空闲状态，当主服务器出现故障时，keepalived 会通过心跳检测发现主服务器出现故障，把 backup 服务器的 IP 设置为 master 的 VIP，从而在极短时间内实现备份服务器代替主服务器提供负载均衡服务。

4.1.2 业务层架构设计

根据需求分析和系统整体设计，现在把整个系统分为四个独立的应用，分别为：个人中心应用、订单服务应用、支付服务应用和秒杀服务应用。所有应用通过 Django 的路由系统来互相调用，业务层的调用关系示意图如图 4.3 所示。

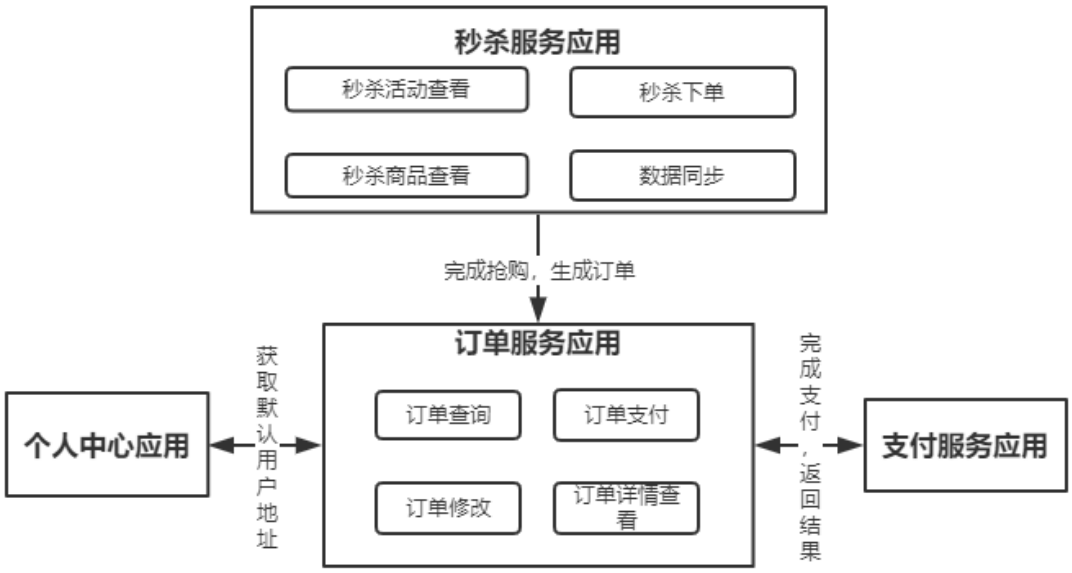

图 4.3　业务层架构图

（1）秒杀服务应用：秒杀服务应用是整个系统的核心，也是本文研究的重点。整个秒杀应用可以分为以下几个模块。

a)秒杀页面行为控制模块

秒杀活动具有时效性，所以在活动开始前后页面应该有不同的行为。在秒杀开始前，页面会有倒计时提示，同时隐藏购买按钮和验证码。这里需要考虑两点，一是客户端和服务器时钟不一致，二是可能内部人员提前知道秒杀链接，提前进行抢购。针对第一点，系统以后台服务器时钟为准。对于第二点，需要通过后台时间检查，防止提前购买。在秒杀活动开始后，显示购买按钮和购买验证码同时隐藏倒计时，同时应不断更新商品剩余库存。秒杀活动结束后，购买按钮和购买验证码再次隐藏，同时在界面显示活动结束。

b)秒杀信息查看模块

秒杀信息的获取可分为秒杀活动信息和秒杀商品信息的获取，为了减轻数据库的压力，信息的获取不可以直接对数据库进行操作，所以在获取秒杀信息时，首先在缓存中查找，如果在缓存中没有结果，则在数据库中查找，然后把在数据库中得到的信息存入缓存中，并返回结果。

c)流量控制模块

流量控制在本系统是尤为重要一环，流量控制可以大大减轻服务器的压力。流量控制原则是尽量把无效流量拦截在上游，拦截处理操作越前置，无效流量对整个系统的影响越小。通过层层递减无效请求，从而减少到达服务器的请求数量，减少服务器的压力。

在客户端层，由于用户习惯于在秒杀开始前几秒不断点击购买按钮以免抢购商品失败，此时如果不做限制，大量无效请求涌入服务器会直接导致服务器压力激增以致宕机，所以系统应该在秒杀活动开始隐藏秒杀按钮。在秒杀活动开始后应该设置验证码，来拉

长用户下单过程，同时避免抢购软件恶意抢购。还要设置库存前置检查，如果库存为 0 则禁止下单，显示活动已结束。在网关层，系统可以利用负载均衡技术分流，同时对相同 IP 做出请求频率限制，防止抢购软件恶意抢购。在后端应用层，系统要设置一系列逻辑判断来终止无效请求继续下发。这些逻辑包含：该用户是否已参与过此次活动、验证码是否正确、用户是否在黑名单中、请求是否在活动规定的时间内。最后在缓存中为每个商品设置一个队列，队列长度代表当前正在购买该商品的请求数，同时需要设置一个阈值，购买该商品的人数不能超过这个阈值，考虑到下单失败的场景，可把阈值设置比库存大，如果购买该商品的人数超过这个阈值，直接结束并返回下单失败。

d)秒杀下单模块

秒杀下单模块主要是处理用户的请求，此模块逻辑比较简单，就是更新库存生成订单，但是会产生数据同步方面的问题。在减库存时，首先应该更新缓存中的库存，由于抢购数量可能超过 1，所以此时更新库存不是原子性操作，在并发的情况下可能造成超卖情况的发生。超卖产生原因如图 4.4 所示。

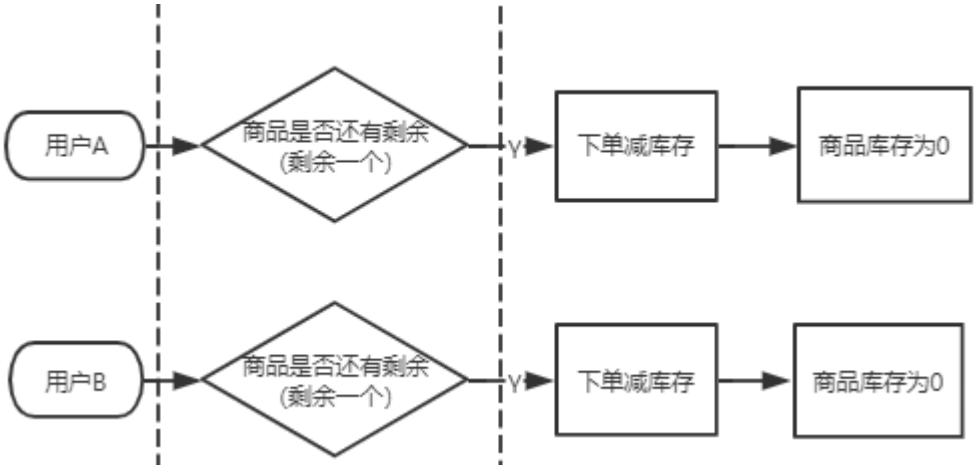

图 4.4　超卖示意图

在上面的场景中，某商品库存为一，这时候系统发来两个并发请求，两个请求同时进行商品是否还有剩余判断，都得到结果为还有一个剩余，都进行下单减库存操作，最后商品库存为 0 或者一个负数，然而却卖出了两个商品，这就造成售出商品数超出库存数的超卖情况发生，解决超卖问题的方法很多，悲观锁和乐观锁都是其解决方案。

悲观锁，在修改数据的时候，被修改数据采用锁定状态，排斥外部请求的修改。其它线程遇到加锁状态的数据，就必须等待。

乐观锁，每次去读写数据的时候都认为其它线程不会修改其监督的数据，所以不会上锁，但是在更新的时候会判断一下在此期间这个数据有没有被其它线程修改过，如果没有则提交更新，反之则更新失败。可以使用版本号机制和 CAS 算法实现。

很显然乐观锁不是真正的加锁，避免了加锁带来的额外开销，效率非常高。悲观锁

相比于乐观锁效率比较低，所以本系统使用乐观锁机制，但是乐观锁可能由于其它线程更新库存导致更新失败，所以对于每个请求的修库存操作有十次尝试机会，十次更新数据都失败则返回失败。在缓存中更新完库存后还需要在数据库中更新库存，同时还需要生成订单。为了防止库存更新成功，生成订单失败造成商品库存减少量和商品出售数量不一致情况出现，要把订单的生成和商品库存更新必须设为一个事务，确保商品库存减少量和商品售出数量相。在生成订单后调用支付应用实现订单支付。

（2）订单服务应用

该应用主要负责用户对订单的相关操作包括：订单列表查看、订单取消、订单支付功能。由于一些用户存在恶意下单的情况，目前还没有很好的方法解决这个问题。为了给所有用户公平良好的体验，订单的取消和修改需要和商家管理人员进行沟通，最后由商家进行订单的修改和取消。订单支付主要是通过支付宝和微信提供的接口完成支付功能。

（3）用户中心应用

用户中心包含三个模块：注册模块、登录模块和个人信息管理模块。

a）注册模块

注册账户必须使用邮箱或者手机号注册。为了减轻服务器压力，账号和密码的格式检查都是在客户端通过 JavaScript 代码自行检查。在服务器端密码需要加密以及向新用户邮箱需要发送激活链接。

b）登录模块

所有用户在参加秒杀活动时需要登录，在登录时除了需要输入账号密码外还需要输入验证码，客户端对账号格式检查通过后会把带有账号密码以及验证码的信息封装成 Json 格式发送到服务器，服务器接收到发送来的账号信息后会检查账号密码和验证码正确性以及该账号是否已激活，如果有错误返回相应的错误信息。

c）个人信息管理模块

该模块主要是对自己的个人信息进行管理，主要是对姓名、性别、邮箱、生日、头像、手机号码和地址等个人信息进行删除和更新操作。

4.1.3　数据访问层架构设计

数据访问层是用来做数据持久化的。在系统开发过程中，很多地方避免不了使用 SQL 语言和数据库进行读写操作，由于 SQL 语言本身比较繁杂维护困难，同时如果使用不当会造成读写效率低下等原因，所以现在一般都会把对数据库的访问，单独作为一层进行精心设计。数据访问层为系统开发人员提供统一的数据库操作类或接口，简化系统开发人员对数据库的操作，避免与复杂的 SQL 打交道，专注于业务逻辑的开发，提高开发效率和系统性能。本系统数据访问层架构图如图 4.5 所示。

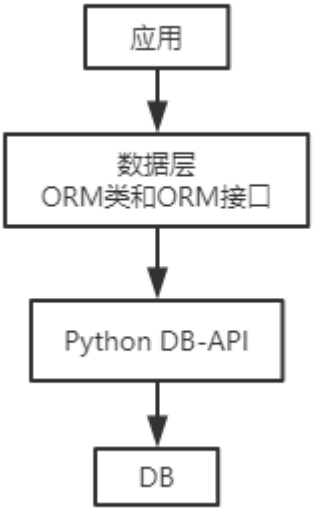

图 4.5　数据访问层架构图

本系统使用 Django 中的 ORM 框架实现关系型数据库和对象模型之间的映射[32]。首先 ORM 框架对外为应用程序提供统一的访问数据库的接口，对内通过调用 Python 自带的数据库 API 实现对数据库的读写。

Django 通过自定义 python 类的形式来定义具体的模型，数据库中的表都和定义的模型一一对应，数据表中的每行数据都代表模型类的一个对象，类中定义的属性以及和其它模型之间的关系分别对应表中的列和其它模型对应的表之间的关系。这样在操作数据库时就不用通过复杂的 SQL 语句和数据库打交道了，只需要通过调用 ORM 提供的接口，像操作对象一样来操作数据库[33]。

4.1.4　数据层架构设计

数据层主要包含 Redis 缓存集群和 MySql 数据库集群两部分。

（1）Redis 缓存集群设计

缓存在本系统中是必不可少的，缓存的作用一是减轻数据库的压力，二是提高系统的响应速度。在本系统中有下面三种典型场景需要频繁与数据库进行交互：1）大量用户不断查看浏览商品信息。2）在抢购页面中需要不断实时显示商品当前库存。3）在抢购过程中产生的订单需要不断更新到数据库中。在上面所有情景中如果不采取任何措施，直接对数据库进行操作会对数据库产生极大的压力，很有可能造成数据库无法提供服务，最终导致整个系统宕机。

为了满足秒杀活动产生的大量对数据库的请求以及整个集群的需求，本系统需要设计一个在分布式环境下运行的缓存用以满足系统的需要，当请求数据库读写数据时，首先需要实现对缓存的读写，如果要操作的数据不存在于缓存中，才对数据库精心操作。这样就减轻了数据的压力。

缓存集群采用 Redis sentinel 模式实现，一个 master 节点三个 slave 节点在配合三个 sentinel 实现高可用方案搭建。该模式有如下特点：所有写数据操作都在主节点上进行，所有读数据操作都在从节点上进行，实现读写分离，减轻缓存服务的压力。在 master 节点出现故障后，sentinel 监视系统会在规定时间内自动在 slave 节点中选出一个节点升级为主节点，实现故障自动修复。

（2）MySql 集群设计

为了减轻数据库压力和数据的备份，本系统使用 Mysql 数据库，选择主从高可用架构，一个主节点和一个从节点还有一个安装 MySQL-Proxy 中间件的节点，主节点负责数据更新操作从节点负责读操作从而实现读写分离，MySQL-Proxy 中间件实现数据库集群在主节点出现故障时，从节点承担起主节点的任务。

4.2　业务层详细设计与实现

本节将会对业务层中的两个应用进行详细设计，同时对主要功能和复杂的代码片段进行详细阐述。

4.2.1　秒杀应用详细设计与实现

（1）秒杀页面行为控制模块详细设计与实现

该模块主要是根据服务器当前时间和秒杀活动开始时间进行比较，通过服务器后端和客户端的配合，实现在不同时间段内为用户展现出秒杀活动开始前、秒杀活动进行中和秒杀活动已结束三种不同的界面。在秒杀活动开始前，界面中需要显示倒计时，此时库存不刷新。当秒杀活动开始后，需要隐藏倒计时功能，同时显示秒杀前验证码和立即秒杀按钮，库存也开始以固定频率进行刷新，同时会对商品库存进行判断，库存为 0 或者到达活动结束时间则结束活动。在活动结束后，界面中需隐藏秒杀验证码和立即秒杀按钮，同时显示活动已结束界面。故该模块主要有三个功能：行为控制功能、实时库存功能和倒计时功能。下面将对三个功能进行详细阐述：

a）行为控制功能

行为控制功能主要是通过 view、template 和前端配合实现的。下面是行为控制功能的核心代码。

view 中代码：

```
def productdetail(request):
    isbegin = False# 初始化标识是否开始秒杀，是否过期失效均为 False
    isexpire = False#活动是否过期标识符，False 为没有过期
    hasstockqty = True#当前商品是否还有库存
    product_id = request.GET.get('product_id')
    stock_key = 'stock_{}'.format(product_id)
    stock_qty = getcache(stock_key)
    if int(stock_qty) > 0:
        hasstockqty = True
    else:
        hasstockqty = False
    diffstarttime = nowtime - startdatetime
    expiretime = nowtime.hour - enddatetime.hour
    if diffstarttime.days >= 0 and expiretime <= EXPIREMAXTIME:
        isbegin = True  # 开始
    elif diffstarttime.days < 0:
        isbegin = False  # 还未开始
    elif expiretime > EXPIREMAXTIME:
        isexpire = True  # 过期
    return render(request, 'seckill/productdetail.html', locals())
```

view 会把运行结果传递给 template 以生成 html，template 代码：

```
{% if  isexpire %}
        #显示秒杀结束界面
{% else %}
     {% if  isbegin %}
         {% if  hasstockqty %}
             {% if  request.user.is_authenticated %}
                 #显示正在进行秒杀界面
             {% endif %}
              </div>
         {% else%}
               #显示秒杀结束界面
         {% endif %}
     {% else%}
          #显示秒杀开始前界面
     {% endif %}
{% endif %}
```

首先在 view 中的 productdetail()方法中设置三个变量 isbegin、isexpire、hasstockqty，三个变量分别代表的意义是秒杀活动是否已开始，秒杀活动是否已过期和

秒杀商品是否还有剩余。然后在缓存中获得活动和商品信息，如果库存为 0 设置 hasstockqty 的值为 False，否则为 True。同理根据活动开始时间和服务器当前时间计算出当前秒杀活动的状态。如果未开始设置 isbegin 为 False，否则为 True。如果活动以结束则设置 isexpire 为 True，否则为 False。之后 view 会调用 template 生成 html 文档。在 template 中使用 template 语言，根据 view 传进来的 hasstockqty、isbegin 和 isexpire 三个标识变量选择不同的秒杀界面对应的 html 元素进行组装，生成 html 文档，然后返回给用户浏览器进行展示。

b）实时库存功能

该主要功能是实时显示当前库存，对于消费者来说，商品库存主要是作为判断商品有无的参考依据，反而对于其精确度没有要求，故采用 Ajax 轮询的方式向后端发送请求商品库存请求从而实现按一定频率异步更新库存，实现主要代码如下：

客户端代码：

```
var intervalstock = window.setInterval(function fnGetstock(){
    var product_id  = $('#product_id').val();
    $.ajax({
              url:"/getstock/",
              data: {
                   'csrfmiddlewaretoken': $.cookie('csrftoken'),
                   'product_id':product_id,
              },
              success: function(data){
                 //alert(data.msg);
                  if (data.hasstockqty){
                      $('#stockqty').text(data.stock_qty);
                  }else{//没库存，提示秒杀结束
                     $('#stockqty').text(0);
                     $('#J_IptAmount').attr("value",'0');
                     $('#J_SecKill').css('display','none');
                     $('.end-seckill').css('display','block');
                     window.clearInterval(intervalstock);
                  }
              },
        …
        },
 });
}, 2000);
```

后端代码:

```
def getstock(request):
    #product_id = request.REQUEST.get('product_id')
    product_id = request.POST.get('product_id')
    stock_key = 'stock_{}'.format(product_id)
    stock_qty = getcache(stock_key)
    if stock_qty:
        if int(stock_qty) > 0:
            hasstockqty = True
        else:
            hasstockqty = False
    else:
        stock_qty=0
        hasstockqty = False
    return JsonResponse({"hasstockqty": hasstockqty, "stock_qty": int(stock_qty)})
```

首先客户端利用 Ajax 技术每隔 2000ms 向后端发送一条包含商品 ID 的 Json 格式数据，后端收到 Json 格式数据后，获得其中包含的商品 ID 信息，在缓存中查找出该商品 ID 所对应的商品库存并返回给客户端。客户端收到库存后局部更新库存，如果库存为 0，则显示库存为 0，并隐藏购买按钮和购买验证码，同时显示秒杀结束提示，最后要禁止 fnGetstock()函数继续向后台发送获取库存请求。

c）倒计时功能

为了保持所有客户端时间一致以及安全性，该功能所需时间应该从服务器上获得，该功能的实现主要是通过 Ajax 把服务器当前时间传给客户端，在客户端利用 JavaScript 语言实现倒计时功能，主要代码下:

```
$.fn.extend({
countdown:function(){
        this.each(function() {
        var dateStr = $(this).attr("end-date");
        var endDate = new Date(dateStr.replace(/-/g,"/"));//取得时间的总毫秒数
        var now = getseverdatetime();
        var tms = endDate - now;//得到时间差
        if(tms<0){$.countdown.stop();return false;}
        $.countdown.timers.push({tms:tms,content:$(this)});
        $.countdown.start();//启动倒计时插件
        });
    }
});
//倒计时的插件
$.countdown={
```

```
    timers:[],
    status:'init',
    takeCount:function(){
    if(this.status != 'start')return;
    setTimeout("$.countdown.takeCount()", 1000 );//每秒
    var timers = this.timers;
    for (var i = 0, j = timers.length; i < j; i++) {
        timers[i].tms -= 1000;//计数减一
        ...
        var newTimeText = days+"天"+hours+"小时"+minutes+"分"+seconds+"秒";
        timers[i].content.text(newTimeText);
        if (days==0  && hours ==0 &&  minutes == 0 && seconds == 0 ){
            $('#J_SecKill').css('display','block');
            $('#J_SecKillready').css('display','none');
        }
    }
    },
    start:function(){...};//启动倒计时
    stop:function(){...};//停止倒计时
};
function  getseverdatetime() {...} //获得服务器时间
```

首先在 each()函数中调用 getseverdatetime()函数，该函数利用 Ajax 技术获得服务器时间，然后计算出距离该活动开始的时间差 tms，把时间差值传递给 countdown 并运行。在 countdown 中每隔一秒减去时间差 1000ms（1s）并计算出距离开始时间时间差，把距离开始时间差按天，小时，分，秒的格式拼成一个字符串进行显示。如果秒杀开始，则把倒计时模块隐藏并显示活动开始模块。

如图 4.6、4.7、4.8 分别为秒杀活动开始前界面、秒杀活动进行中后界面、秒杀活动结束后界面。

图 4.6　秒杀活动开始前界面

图 4.7　秒杀活动进行中界面

图 4.8　秒杀活动结束后界面

（2）秒杀信息查看模块详细设计与实现

秒杀信息的获取可分为秒杀活动信息和秒杀商品信息的获取，这两种获取信息逻辑都是相同的，即在获取信息时，首先利用 getcache()方法在缓存中查找，如果在缓存中没有结果，则在数据库中查找，然后把在利用 setcache()方法把数据库中得到的信息存入缓存中并返回结果。同时为了节省内存空间，需要把不是热点的数据清理出去，所以要在缓存中设置商品信息过期时间为 10 分钟，在如果在十分钟内没有访问该条数据，则需要把该条数据清除。下面是获取商品信息的主要代码：

```
1.   ...
2.  product_id = request.GET.get('product_id')
3.  product_key = 'product_{}'.format(product_id)
4.  product_detail = getcache(product_key)//获取商品信息
5.  if product_detail:
6.      product_detail =  pickle.loads(product_detail)  #取值并反序列化
7.  elif product_detail is None://在数据库中获取商品信息
8.      product_detail = SaleProducts.objects.filter(id=product_id)
9.  setcache(product_key,60*10,pickle.dumps(product_detail)) #设置十分钟 60*10
10. ...
```

（3）流量控制模块详细设计与实现

流量控制流程如图 4.9 所示。

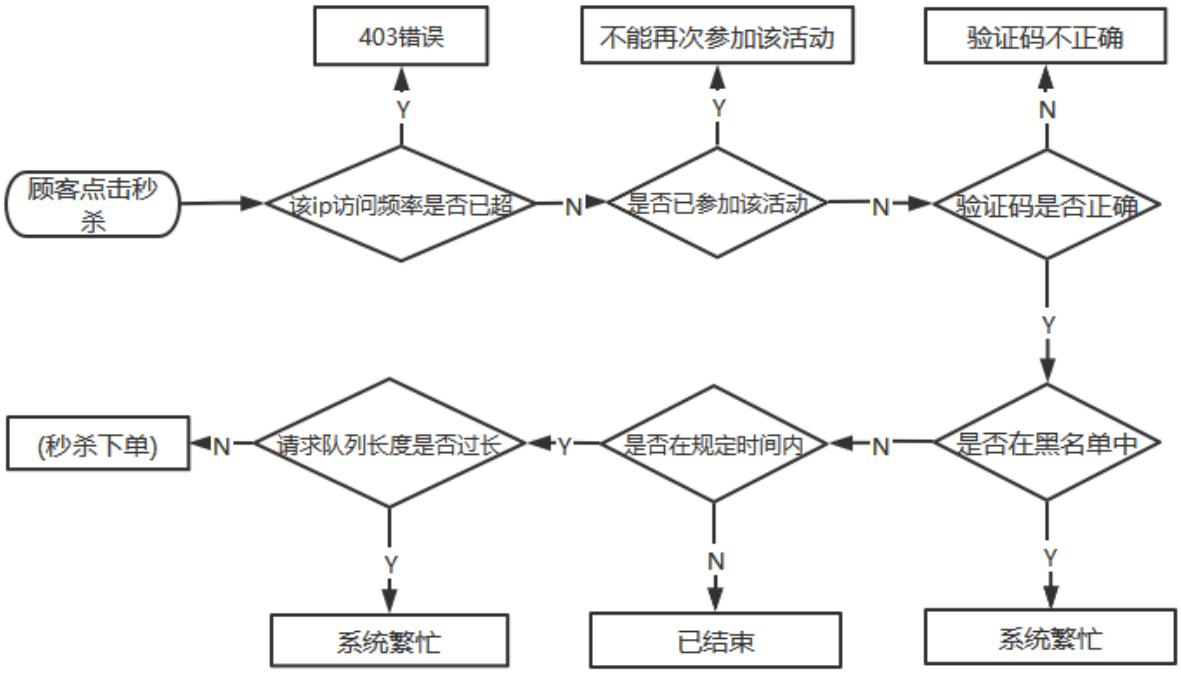

图 4.9　流量控制代码流程

在网关层通过设置 Nginx 配置文件，对来自同一 IP 的请求每分钟只能访问 60

次。在后端在通过代码进行一系列的条件判断实现对流量层层筛选，最后控制在服务器可接受的范围内。首先会通过下单记录检查用户是否已经参加过该活动，参加过则返回不能在参加提示，否则进行验证码检查操作，验证码错误则返回验证码错误提示，正确则进行黑名单检查操作。若用户在黑名单中，则返回系统繁忙提示，正确就在缓存中取商品信息，如果缓存中没有商品信息，就在数据库中取出商品信息加入到缓存中。接下来继续检查提交时间是否在活动规定的时间内，如果不在规定时间内，返回时间错误提示，在规定时间内，则进行防刷单检查，即一个用户一分钟内不能下单超过五次。防刷单是为了防止同一个用户频繁进行下单操作，主要通过在缓存在增加一个（session_key：次数）且设置过期时间为一分钟的键值对，每次访问都会使 session_key 对应的次数加一并重新更新过期时间为一分钟，如果访问频率超过规定频率则返回系统繁忙。通过防刷单检查之后，将商品信息和用户加入缓存请求队列中，队列数据结为 redis 中基本数据类型-队列，其中 key 为 queue_product_key，由产品 id 构成，value 为所有对该商品的请求队列。由于商品库存数量一定，过多的请求没有意义，且会造成服务器额外的压力。同时并不是每一次的请求都会下单成功，所以请求队列的最大长度要设置大于商品库存数同时不能超多库存太多，本系统设置请求队列最大长度为库存数量的 2 倍，如果请求队列长度超过库存量 2 倍，则直接返回请求忙请重试。下面是后台流量控制的主要代码：

```
1.   if isjoin:
2.      msg = u'已参与！不能再参加'
3.      ...
4.  else:
5.      if answer == request.session['answer']:# 验证码检查
6.          check_blackuser = BlackUser.objects.filter(user=request.user).exists()
7.          if check_blackuser:# 检查黑名单
8.              msg = u'系统忙！请重试'
9.              ...
10.         else:
11.             ...
12.             # 检查提交时间是否在有效时间内
13.             startdatetime = product_detail[0].startdatetime
14.             enddatetime = product_detail[0].enddatetime
15.             diffstarttime = posttime - startdatetime
16.             diffendtime = enddatetime - posttime
17.             expiretime = posttime.hour - enddatetime.hour
18.             if diffstarttime.days >= 0 and expiretime < EXPIREMAXTIME:
19.                 session_key = request.session.session_key
20.                 ua_key = 'user_{}'.format(session_key)
21.                 # 防刷，一分钟内不能超过 5 次请求
22.                 if check_request_limit(ua_key, REQUEST_LIMITMAX):
```

```
                    ...
                    stock_qty = int(getcache(key_stock))# 取出当时缓存库存
                    total_quesize = queuesize(queue_procut_key)# 取队列大小
                    if total_quesize > 2 * stock_qty:# 检查队列大小，不符则结束
                        msg = u'系统忙！请重试'
                        ...
                    else:
                        ...
                else:
                    msg = u'请求次数超过{}次'.format(REQUEST_LIMITMAX)
                    ...
            else:
                msg = u'时间不正确'
                ...
    else:
        msg = u'验证码不正确'
        ...
```

（4）秒杀模块设计与实现

秒杀模块的业务逻辑十分简单，总的来说就是用户发起秒杀请求，系统响应请求生成订单，然后在用户支付成功后返回秒杀成功界面。然而，考虑到考虑到系统的并发性和安全性需要采取一些措施来保证系统的正常稳定运行，该模块的时序图如图 4.10 所示。

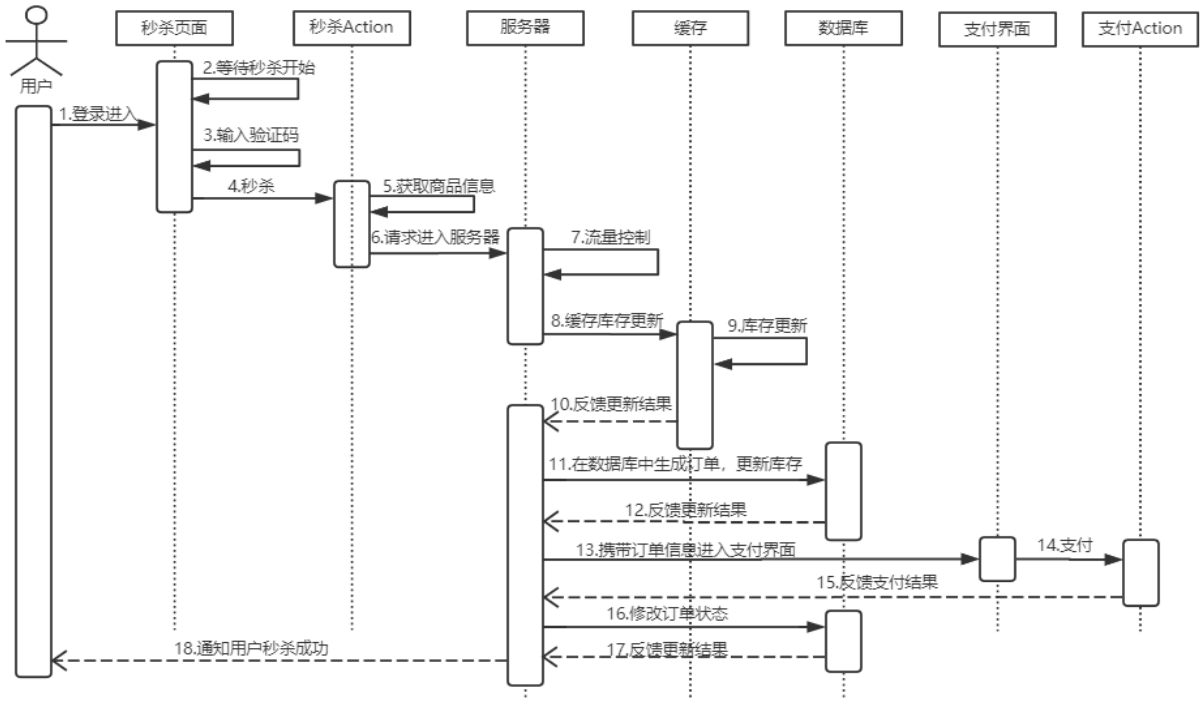

图 4.10　秒杀时序图

由时序图可知，用户登录进入秒杀页面后需要等待秒杀活动开始。秒杀活动开始后，为了延长购买操作时间，避免出现瞬时请求高峰，需要输入验证码然后才可以进行购买操作。在输入验证码和点击立即秒杀按钮后，会向服务器发送包含着商品相关信息的秒杀请求。服务器在接受到购买请求后，首先会调用流量控制模块对请求进行筛选，把无效请求尽量拦截在系统的上游，防止过多无效不符合条件的请求对缓存和数据库产生额外的压力。请求在经过流量控制模块后，服务器会根据包含在请求中的商品信息对存储在缓存中响应的商品库存进行修改，为了防止超卖现象的发生，使用乐观锁对库存进行更新。乐观锁在多线程情况下很有可能由于其它线程对同一库存的修改造成本次减库存操作的失败，所以减库存操作被设置为有十次尝试机会，十次尝试中有一次减库存成功则本次减库存操作成功，如果十次都没有成功返回失败。在完成缓存更新后需要在数据库中对相应的库存更新同时生成订单，为了保持数据的一致性需要把减库存和订单生成设为一个事务。订单生成后订单的状态为未支付状态，然后系统会把包含该订单 ID 的 Json 格式数据传递给前端，前端收到 Json 数据进行解析得到订单 ID，然后重定向到支付界面，用户对订单进行支付后，会反馈支付结果到服务器，服务器得到结果后，会更新数据库中对应的订单状态为已支付。最后返回给用户支付成功消息。

缓存中更新商品库存代码：

```
def update_stock(key,productid, userid,qty,price,CORQ):
    qty=int(qty)
    with master.pipeline() as pipe:
        i=0
        cart_key = 'cart:{}'.format(userid)
        while i<10:#10 次机会
            try:
                pipe.watch(key)
                count = int(pipe.get(key))# 取库存
                if CORQ==1:#生成订单，库存减少
                    if count >= qty:  # 有库存
                        remainqty=count-qty
                        pipe.set(key, remainqty) # 更新库存
                        return True
                    else:
                        return False
                else: #取消订单，库存增加
                    remainqty=count+qty
                    pipe.set(key, remainqty) # 更新库存
                    return True
            except Exception:
                continue
            i+=1
```

```
24.
```

在减库存时，会使用 update_stock()方法更新库存。首先要 pipe.watch(key)，即监视你想要修改库存所对应的 key，此时其它线程也可以对被监视 key 对应的 value 进行读和写，但是修改过程中必须保证本线程监视的数据没有被其它线程修改过，否则本次修改就失败了。CORQ 是一个标志位，其值为 1 代表生成订单，需要减库存，为 0 代表数据库订单生成失败，需要增加商品库存。

更新库存同时生成订单，主要代码如下所示：

```
1.  if update_stock(key_stock, product_id, request.user.id, quantity, price):
2.      with transaction.atomic():# 设置事务，产生订单时必须保持事务一致
3.          amount = Decimal(int(quantity) * price)
4.          profile = Profile.objects.get(user_id=request.user.id)
5.          product = SaleProducts.objects.get(id=product_id)
6.          #生成订单
7.          order = Order.objects.create(...)
8.          OrderItem.objects.create(...)
9.          stock_qty = getcache(key_stock)
10.         product.stock_total = stock_qty# 更改库存
11.         product.save()
```

4.2.2　个人中心应用详细设计与实现

个人中心应用主要包含注册模块、登录模块和个人信息管理模块。对任何系统来说个人中心模块是一切其它功能模块正常运行的前提，下面本文将会在需求分析中用户用例图的基础上，并利用时序图来对个人中心的三个模块进行详细设计与实现。

在基于对用户需求分析的基础上，得到如图 4.11 所示的登录注册模块时序图。

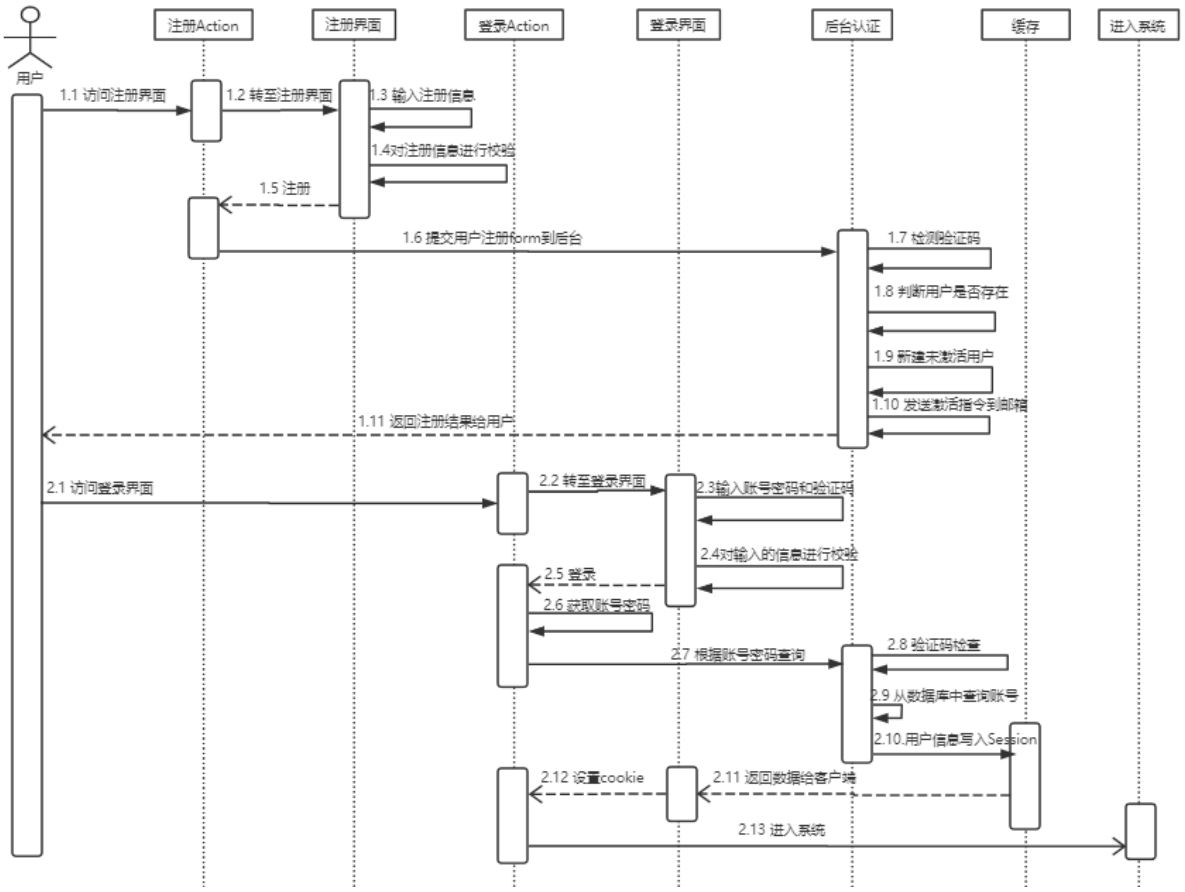

图 4.11　登录注册时序图

（1）注册模块的详细设计与实现

由时序图可知，用户通过注册操作进入注册页面，在注册页面输入个信息，个人信息包括用户名、邮箱、密码以及验证码，密码在输入注册信息以及验证码后，会在客户端对邮箱格式和密码格式进行格式校验，密码必须由 8-16 位字符串组成，包含数字和字母。完成校验后点击注册按钮，会把包含注册信息的请求提交到服务器端。服务器接收到包含有注册信息的请求后，首先会对验证码进行检查，验证码不正确则返回验证码错误结果给客户端展现给用户。验证码正确则在需要检查用户名是否已存在，邮箱是否已注册，如果用户名或邮箱已存在数据库中，返回已存在结果给客户端。经过以上检查都通过后，在数据库中创建一个新用户，其状态为未激活，此时用户还不能登录。然后使用 celery 定时发送一封包含经过加密的用户信息的激活链接到注册邮箱账号，加密方法使用 Serializer ()。用户在规定的时间访问此链接，同时会把经过加密的用户信息传入服务器，服务器经过解密得到用户信息，激活此用户。此时用户可以正常登录。

注册模块注主要代码：

```
def ajax_register(request):
    redirect_url = request.META.get)
    try:
        if request.method == 'POST':
            json_data = json.loads(request.body)
            email = json_data["email"]
            username = json_data["username"]
            password = json_data["password"]
            checkcode = json_data["checkcode"]
            if checkcode.upper() != request.session['valicode'].upper():
                result = 'failed'
                msg = u"注册码不正确"
            else:
            try:
                user = User.objects.get(username=userName,is_active=1)
            except Exception as e:
                user = None
            if user != None:
                result = 'failed'
                msg = u"用户名已存在"
            try:
                user = User.objects.get(email=email,is_active=1)
            except Exception as e:
                user = None
            if user != None
                result = 'failed'
                msg = u"该邮箱已注册"
            user = User.objects.create_user(userName, emailAddress, passWord)
            user.is_active = 0
            user.save()
            serializer = Serializer(settings.SECRET_KEY, 3600)
            info = {'confirm': user.id}
            token = serializer.dumps(info)
            token = token.decode()
            send_register_active_email.delay(emailAddress, userName, token)
            result = 'success'
            status = 200
            msg = u"注册成功"
    except Exception as e:
        print(e)
    return JsonResponse(…)
```

注册界面如图 4.12 所示。

图 4.12　注册界面

（2）登录模块的详细设计与实现

由时序图可知，用户在进行登录操作时首先要进入登录界面，然后输入账号密码以及验证码，账号密码格式都会在客户端通过 JavaScript 代码检查以减轻后台服务器的压力，完成格式检查后通过 Ajax 技术把账号密码和验证码等信息以 Json 格式传到后台，在后台对验证码进行检查，验证码有误则返回客户端验证码错误。否则使用 authenticate() 方法在数据库中查找对应的用户信息，返回 user 对象则用户存在，返回 null 代表用户不存在。最后使用 login()把用户信息写入 Session 中，同时向客户端发送 Json 格式数据更新登录状态。

登录后台代码：

```
def ajax_login(request):
    if request.method == 'POST':
        request_data = request.body.decode('utf-8')
        json_data = json.loads(request_data)
        username = json_data["username"]
        password = json_data["password"]
        checkcode = json_data["checkcode"]
        if checkcode.upper() != request.session['valicode'].upper():
            result = 'failed'
            msg = u"验证码不正确"
            status = 401
        else:
            user = authenticate(username=username, password=password)
```

```
            if user is not None:
if not user.is_active:
                      result = 'failed'
                    msg = u"用户未激活"
                    status = 401
                else:
                    login(request, user)
                    result='sucess'
                    status = 200
                    msg = u'sucess'
                    redirect_url = redirect_url
            else:
                result = 'failed'
                msg = u"密码或者用户名不正确"
    status = 401
    return JsonResponse(…)
```

用户登录界面如图 4.13 所示。

图 4.13　登录界面

（3）个人信息管理模块的实现

该模块主要是展示个人信息并对个人信息进行编辑，个人信息包括：姓名、邮箱、生日、头像、性别、地址和手机。

在用户登录进入个人管理中心后，会显示当前用户的个人信息，用户可以对自己的个人信息进行编辑，编辑完成后客户端会使用 JavaScript 语言自动实现格式检查，通过格式检查后，会把包含用的 form 提交给服务器，服务器获得用户个人信息后再数据库中更新。个人信息编辑界面如图 4. 14 所示。

图 4.14　个人信息界面

第 5 章　电商秒杀系统后台的设计与实现

本章将对后台管理模块和数据库进行设计与实现，由于后台管理模块需求并不复杂，主要是对用户、商品、秒杀活动等实体进行管理，所以采用 Django 自带的 admin 模块实现后台管理。

5.1　后台管理模块设计与实现

后台管理模块是针对管理员设计的，从需求分析的用例图来看管理模块主要功能是对用户、商品和秒杀活动进行管理，管理的操作主要是对几个实体的增删改查，业务逻辑并不复杂。Django 为开发者提供了一套具有强大功能的后台管理工具 admin，可以通过配置实现 admin 的汉化。模型可以通过注册快速在 admin 中自动创建增删改查界面，而且模型与数据库利用 ORM 实现模型与数据表的映射关系，从而 admin 对模型的操作最终会更新到数据库对应的表中。基于以上特点，admin 完全满足本系统的要求，故本系统采用 admin 实现后台管理模块的开发。下面是管理模块中涉及操作的时序图如图 5.1 所示。

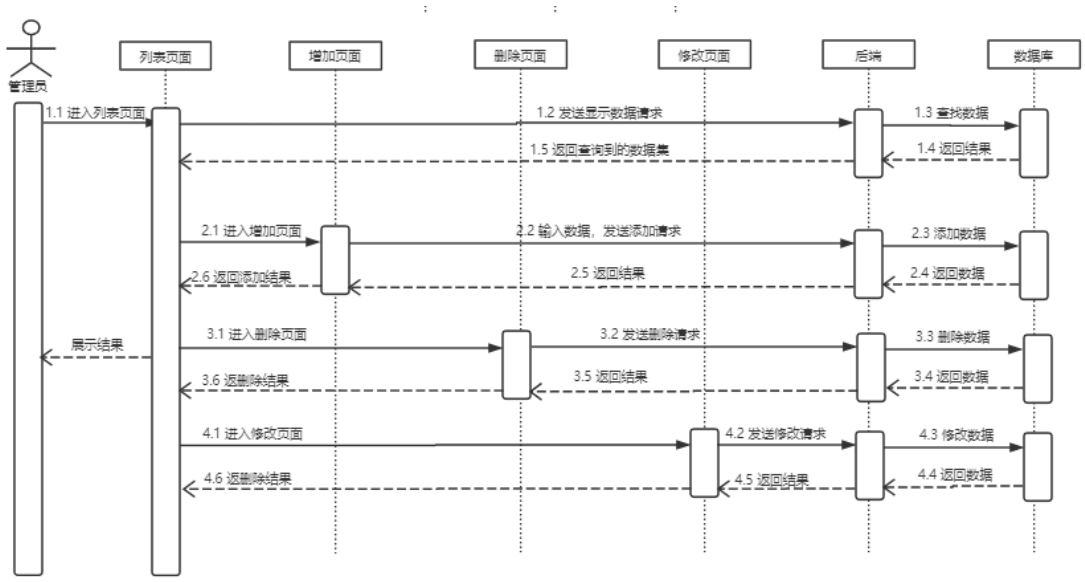

图 5.1　后台管理时序图

可以看到参与时序交互的角色是管理员，其通过入口，即列表页面、增加页面、删除页面和修改页面，实现与后端和数据库的交互以及对模型的管理。总体来看 admin 为管理员提供了一套功能齐全的后台管理系统，节约了开发周期和成本。

5.2　数据库设计

数据库是整个系统的基石，系统内各个模块的正常运行都需要与数据库打交道。数

据库设计的好坏对整个系统有最为直接和重大的影响，高效稳定的数据库是一个高效稳定的系统不可缺少的一部分。因此，如何实现合理高效的数据读取是数据库建设的关键。

5.2.1 数据库概念设计

实体-联系图即 E-R 图用来对现实世界中实体和关系描述的概念模型，是数据库设计中经常使用的方法。在基于前面需求分析的基础上，设计出与现实中实体对应的概念模型和属性，最终通过 E-R 图进行展示，整个系统 E-R 图如图 5.2 所示。

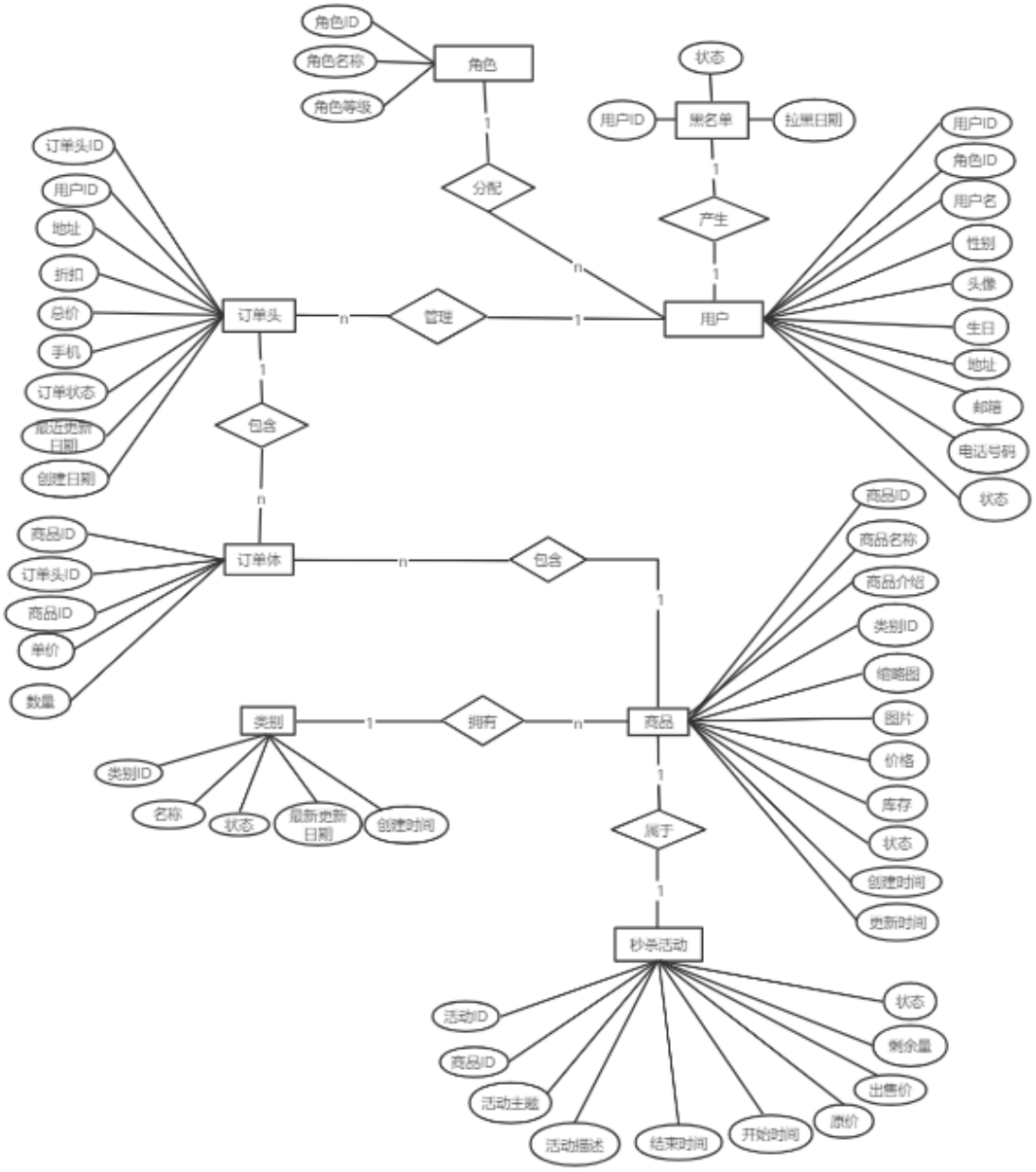

图 5.2 秒杀系统 E-R 图

5.2.2　数据库表设计

由图 5.2 可知，本系统主要包含八个实体对象，下面将把八个实体转换成对应的数据表，并对表以及表内属性进行解释。

如表 5.1 所示是用户表，用户中主要包含十个属性，id 为主键自增，roleId 为角色 id，作为外键与角色表关联，userName 代表用户名称， dateOfBirth 代表出生日期，photo 代表用户肖像存储位置的路径，gender 代表性别，address 代表用户地址，email 代表用户电子邮箱，moible 代表用户联系方式，status 代表该用户是否激活，默认为 False 没有激活。

表 5.1　用户表

字段名	类型	可为空	默认值	注释
id	Int(11)	N		主键，自增
roleId	Int(11)	N		外键，角色 id
userName	Varchar(20)	N		姓名
dateOfBirth	Date	Y		出生日期
photo	Varchar(100)	Y		肖像存储路径
gender	Nchar(2)	Y		性别
address	Varchar(200)	N		地址
email	Varchar(100)	N		邮箱
mobile	Varchar(20)	N		电话
status	Boolean	N	False	状态

如表 5.2 所示，是黑名单表，用于记录恶意攻击本系统或者有不良行为记录的会员信息。黑名单实体主要包含三个属性，userId 作为一个外键与用户表是一对一的关系，status 表示该条黑名单记录是否有效，默认为 True 有效，False 代表无效。lockdate 代表加入黑名单的日期，该属性自动写入，取加入黑名单时候的时间。

表 5.2　黑名单表

字段名	类型	可为空	默认值	注释
id	Int(11)	N		主键，自增
userId	Int(11)	N		外键与用户表关联
status	Boolean	N	True	状态
lockdate	Timestamp	N		加入黑名单时间

如表 5.3 所示为角色表，roleName 代表角色名称，roleLevel 代表角色等级，status 代

表状态即角色是否可用。

表 5.3　角色表

字段名	类型	可为空	默认值	注释
id	Int(11)	N		主键，自增
roleName	Int(11)	N		外键与用户表关联
rolelevel	Int(11)	N		角色等级
status	Boolean	N	True	状态

如表 5.4 所示，订单头表中共包含 9 个属性，userId 作为一个外键与用户表关联。email、address、mobile 代表消费者的邮箱、地址、电话。created 代表订单创建日期，取订单生成时候的时间自动填入。updated 代表订单最后更新日期，取最后修改订单时候的日期自动填入。status 属性作为一个标记，代表订单的各种状态，可取值范围分别为：‘0’代表‘取消’，‘1’代表‘待支付’，‘2’代表‘已支付’，‘3’代表‘退款’。totalPrice 代表该订单总金额。

表 5.4　订单头表

字段名	类型	可为空	默认值	注释
id	Int(11)	N		主键，自增
userId	Int(11)	N		外键，与会员表关联
email	Varchar(100)	N		订单创建者邮箱
address	Varchar(200)	N		订单创建者地址
mobile	Varchar(20)	N		电话号码
created	Timestamp	N		订单创建时间
updated	Timestamp	N		订单最新更新时间
status	Nchar(2)	N	2	订单状态
totalPrice	Decimal(13,2)	N		总价

如表 5.5 所示是订单体表，其中 orderId 作为一个外键与订单头表关联，是一对多的关系，即一个订单里面可以包含多条商品购买记录。productId 也作为一个外键与商品表对应。price 代表商品在秒杀活动中的单价。quantity 代表购买商品的数量。

表 5.5　订单体表

字段名	类型	可为空	默认值	注释
id	Int(11)	N		主键。自增
orderId	Int(11)	N		外键，和订单表关联
productId	Int(11)	N		外键，和商品表关联
price	Decimal(13,2)	N		单价
quantity	Int(11)	N		数量

如表 5.6 所示是类别表，类别共包含 5 个属性，categoryName 代表类别名称，created 代表类别创建的时间，取创建时候的时间自动填入。updated 代表类别更新的最后时间，取最近更新时间自动填入，status 代表类别的状态，可取值范围为：‘1’代表‘正常’，‘0’代表‘冻结’。

表 5.6　类别表

字段名	类型	可为空	默认值	注释
id	Int（11）	N		主键，自增
categoryName	Varchar(20)	N		类别名称
created	Timestamp	N		类别创建时间
updated	Timestamp	N		类别最新更新时间
status	Nchar(2)	N		代表类别的状态

如表 5.7 所示是商品表，该实体拥有 11 个属性。productName 代表商品名称，description 代表描述商品的内容。categoryId 作为一个外键与类别表相关联，代表商品类别。image 代表商品缩略图所在位置路径，Largeimage 代表商品大图所在位置路径，在详情页面显示商品大图，price 代表商品单价，stock 代表商品目前库存。status 代表该商品是否有效，可选值范围：True 代表有效，False 代表无效。created 代表类别创建的时间，取创建时候的时间自动填入。updated 代表类别更新的最后时间，取最近更新时间自动填入。

表 5.7　商品表

字段名	类型	可为空	默认值	注释
id	Int(11)	N		主键自增
productName	Varchar(50)	N		商品名称
description	TINYTEXT	Y		秒杀内容介绍
categoryId	Int(11)	N		外键，与类别关联
image	Varchar(100)	Y		小图路径
largeImage	Varchar(100)	Y		大图路径
price	Decimal(13,2)	N		原价
stock	Int(11)	N		库存
status	Boolean	N	True	状态
created	Timestamp	N		商品入库时间
updated	Timestamp	N		商品最新更新时间

如表 5.8 所示为秒杀活动表，该表一共有 10 个属性。title 代表秒杀商品在秒杀活动中的标题。productId 作为一个外键与商品表关联，两表是一一对应的关系，title 秒杀活动的主题，description 为秒杀活动的详细介绍，marketprice 代表市场价。price 代表秒杀价，resStock 代表商品库存剩余量。Status 代表该秒杀商品的当前状态，可取值范围为：True 代表‘正常’，False 代表‘冻结’。

表 5.8　秒杀活动表

字段名	类型	可为空	默认值	注释
id	Int(11)	N		主键，自增
productId	Int(11)	N		外键，和商品表关联
title	Varchar(200)	N		秒杀活动标题
description	TINYTEXT			秒杀活动介绍
marketprice	Decimal(13,2)	N		原价
price	Decimal(13,2)	N		活动价
startDatetime	Date	N		秒杀活动开始时间
endDatetime	Date	N		秒杀活动结束时间
resStock	Decimal(13,2)	N		商品剩余库存
Status	Boolean	N	True	商品状态

第 6 章　系统测试与分析

在系统开发完成后，系统测试是必不可少的环节，同时也是整个项目开发过程中最重要环节之一。系统测试的目的是确保系统与设计一致，各个模块之间的逻辑以及模块内部的逻辑处理正确，确保系统可正常的运行。本章分别对系统后台和前台进行功能性测试，对面向消费者的前台进行非功能性测试。

6.1　测试环境部署

本系统的所有硬件设备都在同一网段内，整个测试环境都处于同一局域网内。使用的机器规格参数如表 6.1 所示，系统整体部署示意图如图 6.1 所示。

表 6.1　系统测试环境

机器	处理器	内存	操作系统
s1	Intel i7-6700 CPU 4GHz	32G	Ubuntu 16.04
s2,s3	Intel i5-9400F CPU 4.1GHz	8G	Ubuntu 16.04
s4	Intel i5-9400F CPU 4.1GHz	8G	Ubuntu 16.04
s5,s6,s7	Intel i5-9400F CPU 4.1GHz	8G	Ubuntu 16.04
s8,s9	Intel i7-6700 CPU 4GHz	32G	Ubuntu 16.04
s10	Intel i7-6700 CPU 4GHz	32G	Ubuntu 16.04

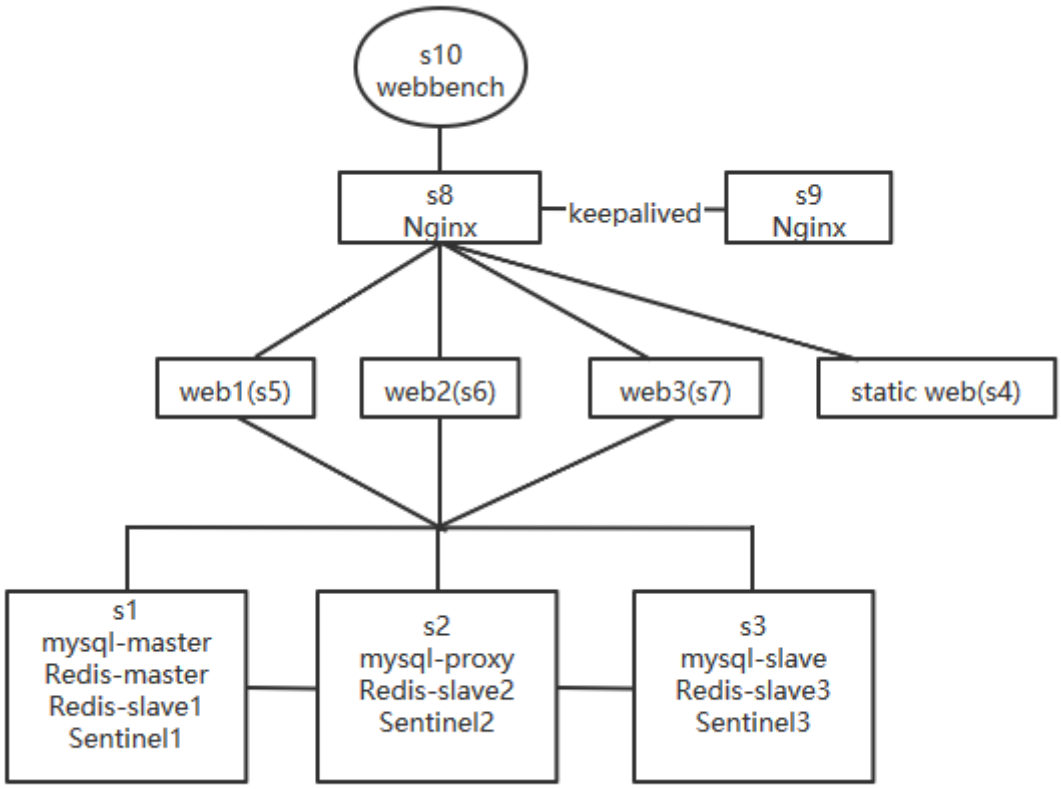

图 6.1　系统整体部署示意图

（1）数据库和缓存的部署

数据库集群和缓存集群都部署在 s1,s2,s3 机器上。数据库集群主要由一个 master 节点一个 slave 节点和一个部署 mysql-proxy 插件的节点，从而实现支持主从复制和读写分离的高并发、高可用集群。缓存集群由一个 master 节点，三个 slave 节点和三个 sentinel 节点组成，由于 s1 配置较高，所以 master 部署在 s1 机器上, 三个 slave 和三个 sentinel 分别同时部署在 s1,s2,s3 机器上。

（2）网关和应用的部署

由于 s8 和 s9 配置较高，所以部署 Nginx+keepalive 用来搭建负载均衡转发集群。s4 作为静态服务器。在 s5,s6,s7 上分别部署 Django 应用+Gunicorn，同时在这三个机器上分别部署 supervisor 完成对各个进程监控与管理。最后在 s10 上部署 webbench 进行压力测试，因为压力测试并发量在 1W 左右因此应该选用性能较好的机器。

6.2　秒杀系统功能测试

6.2.1　秒杀系统后台功能测试及结果

对秒杀系统后台功能测试用例和测试结果如表 6.2 所示。

表 6.2　后台管理系统测试结果

编号	测试目标	测试操作	测试结果
A1	管理员能否添加新用户	添加新用户	可正常添加新用户
A2	管理员能否删除用户	删除指定用户	可正常删除用户
A3	管理员能否编辑用户信息	编辑用户信息	可编辑用户信息
A4	管理员能否查看用户信息	查看用户信息	可查看用户信息
A5	管理员能否增加新商品类别	增加新类别	成功增加商品类别
A6	管理员能否删除商品类别	删除商品类别	成功删除商品类别
A7	管理员能否编辑商品类别信息	编辑商品类别信息	成功编辑商品类别信息
A8	管理员能否查看商品类别信息	查看商品类别信息	可查看商品类别信息
A9	管理员能否增加新商品	增加新商品	成功增加新商品
A10	管理员能否删除指定商品	删除指定商品	成功删除指定商品
A11	管理员能否编辑商品信息	编辑商品信息	可编辑商品信息
A12	管理员能否查看商品列表	查看商品列表	可查看商品列表
A13	管理员能否增加新秒杀活动	增加秒杀活动	成功添加秒杀活动
A14	管理员能否删除指定秒杀活动	删除秒杀活动	成功删除秒杀活动
A15	管理员能否编辑秒杀活动	编辑秒杀活动	可编辑秒杀活动
A16	管理员能否查看秒杀活动列表	查看秒杀活动列表	可查看秒杀活动列表
A17	管理员能否编辑订单	编辑订单	可编辑订单
A18	管理员能否查看订单列表	查看订单列表	可查看订单列表
A19	管理员能否添加黑名单用户	添加黑名单用户	成功添加黑名单用户
A20	管理员能否删除黑名单用户	删除黑名单用户	成功删除黑名单用户
A21	管理员能否登录	管理员登录	成功登陆
A22	管理员能否退出账号	退出账号	成功推出登陆状态

6.2.2　秒杀系统前台功能测试及结果

对前台秒杀系统测试用例和测试结果如表 6.3 所示。

表 6.3　前台秒杀系统测试结果

编号	测试目的	测试操作	测试结果
B1	用户能否正常注册账号	注册账号	成功注册账号
B2	用户能否登录	登录	成功登陆
B3	用户能否查看活动列表	查看活动列表	可查看活动列表
B4	用户能否查看秒杀商品详细信息	查看商品详细信息	可查看商品详细信息
B5	用户能否查看秒杀活动详细信息	查看活动详细信息	可查看活动详细信息
B6	用户能否下单秒杀商品	进行商品抢购	成功抢购商品
B7	黑名单用户能否参与商品秒杀	黑名单用户抢购商	禁止黑名单用户抢购商品
B8	同一账号能否快速多次参与秒杀活动	多次快速参与商品秒杀	只有第一次抢购商品成功
B9	用户能否编辑个人信息	编辑个人信息	编辑个人信息成功
B10	用户能否查看个人订单	查看个人订单	可查看个人订单
B11	用户能否取消订单	取消订单	取消订单成功

6.3　秒杀系统非功能测试

在系统上线前，需要对整个系统进行仿真压力测试，这是必不可少的一个环节。通过压力测试，可以知道在模拟的仿真环境下系统的各个性能指标，测试方案的好坏直接影响系统能够顺利上线以及用户的体验。

本系统采用 Webbench 压力测试工具进行仿真压力测试。Webench 是一款方便、简洁、功能强大的测压工具，通过设置参数可以模拟 30000 的并发请求数，同时也可以通过配置参数，实现对测试时间、是否等待服务器回复等功能的控制，对于并发量不太高的中小型网站有着极佳的体验。

6.3.1　并发性能测试

在本次测试中，主要检查系统在仿真环境下的并发性能。一共测试四次，每次测试时间为 60 秒，而模拟客户端数量（并发数）不同，通过请求负载均衡的端口，来测试整个系统的并发性能。测试结果如表 6.4 所示。

表 6.4　并发性能测试结果

	并发数	100	1000	5000	10000
系统指标	100KB/sec	143	114	123	131
	CPU 使用率	30.2%	42.9%	62.4%	81.4%
	平均请求响应时间	31(ms)	33(ms)	506(ms)	1824(ms)
	秒杀成功率	100%	100%	100%	100%

从测试结果来看，随着并发量的增加，平均请求时间也在上升，在并发量达到 10000 时，整个测试过程平均请求响应时间为 1824ms，秒杀成功率为 100%，系统并发性能达到预期要求。

6.3.2　可用性测试

本次测试中，主要检查在某些意外情况下造成某服务出现故障，能否自动进行故障转移，避免出现单点故障。测试主要对三种情况进行测试，一是负载均衡层主服务节点出现故障情况下，热备节点能够自动承担起负载均衡的作用。二是在缓存主节点出现故障时，从节点能否作为主节点继续提供服务。三是在业务服务器部分出现故障时还能否正常提供服务。下面进行三次实验，分别对上面三种情况进行测试，三次测试中并发数量为 10000，持续 60 秒，在测试过程中停止不同的服务以检测不同故障故障对系统的影响，测试结果如表 6.5 所示。

表 6.5　可用性测试结果

	出现故障情况	负载均衡主节点故障	缓存主节点故障	停止一台业务服务器
系统指标	100KB/sec	153	142KB/sec	106KB/sec
	CPU 使用率	81.6%	72.9%	91.4%
	平均请求响应时间	1824(ms)	2632(ms)	2844(ms)
	秒杀成功率	100%	98.2%	99.4%

由实验结果可知，三种出故障情况下对整个系统影响有所不同。第一种情况，在负载均衡主节点宕机情况下基本对整个系统没有什么影响，可见在主节点出故障情况下，热备从节点实现了无缝转移，没有对系统产生明显影响。第二种情况，在缓存主节点出现故障后，系统平均响应时间增加，有秒杀失败情况发生，这是因为缓存从节点切换为主节点需要一定时间，在这段时间缓存主节点不提供服务，会造成一些下单请求超时而失败。第三种情况，在系统运行过程中停止一台业务服务器，请求响应时间有所增加，秒杀成功率为 99.4%，这是由于缺少一台业务服务器，系统处理请求变慢，请求响应时间延长，极少部分请求由于请求超时而出现错误。整体来说，系统可用性较高，不会因

为局部故障而对整个系统造成不可接受的影响。

总结与展望

总结

本文在基于M公司实际需求的基础上,对秒杀系统关键技术进行详细分析与研究，设计并实现了一套完整的电商秒杀系统。秒杀活动是一种高并发的活动，所以秒杀系统主要的设计思路主要有两点：一、通过层层削峰和控制流量，尽可能的拦截无效流量，把流量控制在一个可接受的范围。二、增加系统的处理速度，同时对可能出现的问题提前准备解决方案，保证系统的高并发、高可用。

系统整体采用Django Web框架和Python语言进行开发，整个系统采用模块化开发方法，横向上采用MTV三层架构模式，纵向上分为多个独立的APP，保证了项目开发效率和后期的维护成本；在网关层使用七层负载技术，实现灵活的分发策略，保证请求的高效分发，同时采用Keepalived的方案，保证网关层的高可用性；为了提高系统的响应速度，减轻数据库的压力，用Redis作为缓存存储热点数据，以减轻数据库的压力和系统的响应速度。同时为了保证缓存的高可用性，采用Redis Sentinel集群模式，实现缓存的读写分离和备份，提高缓存集群的并发性和高可用性；数据库采用主从架构的模式，保证数据库的并发性高可用性。通过以上方案，秒杀系统基本满足客户的需求。

展望

由于时间和技术的限制，秒杀系统还可以进一步完善，具体研究方向包含：

(1)在系统部署阶段，由于系统环境部署比较麻烦，导致浪费很多时间和资源，接下来考虑使用以docker等虚拟化技术为核心,简化部署流程，构建一套容易维护的集群环境。

(2)对于秒杀系统的网关层，随着公司的发展，网站使用人数的增长，网关层要采用性能更加高效的LVS/DR模型实现负载均衡，缓存集群可以使用Redis客户端分片技术配合Redis Sentinel方案实现缓存层更加高效和高可用的目的。

参考文献

[1] 蒋会锋. 新媒体环境下电子商务营销策略分析[J].中外企业家,2020(09):97.

[2] 秦志伟. 我国电子商务发展存在的问题及其对策[J].中外企业家,2020(08):101.

[3] 云谦. 网购促销层出不穷 秒杀抢购骗局防不胜防[J].中国防伪报道,2015(03):92-94.

[4] 张小红. 电子商务营销策略之秒杀[J].机械管理开发,2011(02):150-151.

[5] 于晓林. 面向中小型 B2C 的电商平台的限时促销和秒杀功能的设计与实现[D].成都:西南交通大学,2018.

[6] 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.

[7] 王小戏,吴刚,王灏. 高并发高可用零售 O2O 交易系统的架构设计与业务实现[J].计算机与现代化,2016(04):100-108.

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

[9] 王昆. 高可用分布式任务调度与执行系统设计与实现[D].西安:西安电子科技大学,2019.

[10] Hollenback P. Improving network reliability with Keepalived[J]. 2008.

[11] 蹇常林. ORM 在 Django 操作数据库中的应用[J].技术与市场,2020,27(01):56-57

[12] 袁晓东. MySQL Cluster 集群数据库误操作恢复方法研究[J].信息技术与信息化,2019(12):212-215.

[13] 白昌盛. 基于 Django 的 Python Web 开发[J].信息与电脑(理论版),2019,31(24):37-40.

[14] 卢慧雅,王磊. 基于 MVC 设计思想的 Java 实验案例优化[J].计算机教育,2020(03):56-58.

[15] 李军.高并发 Web 系统的设计与优化[D].北京:北京交通大学,2009.

[16] Carlson J L. Redis in Action[M]. Manning Publications Co., 2013.

[17] 周晓场. 基于 Redis 的分布式 Key-Value 系统的优化研究[D].广州:华南理工大学,2018.

[18] 柳皓亮,王丽,周阳辰. Redis 集群性能测试分析[J].微型机与应用,2016,35(10):70-71+78.

[19] 薛军等. 负载均衡技术的发展[J].小型微型计算机系统. 2003

[20] 常潘,沈富可. 使用域名负载均衡技术实现校园网对外服务器的高速访问[J].计算机应用,2007(07):1585-1586+1590.

[21] 齐美彬、李莉. 基于应用层负载均衡策略的分析研究[J].计算机工程与应用.2007 43(32)

[22] 郭成城、陈亮. 一个基于 TCP 迁移机制的第七层负载均衡系统[J].计算机应用研究.2005(5):116 ～ 118.

[23] 李杰. 基于 LVS 集群的动态负载均衡算法研究[D].南昌:南昌航空大学,2018.

[24] 戴伟. 基于 Nginx 高性能 Web 服务器的理论研究与性能改进[D].南京:南京邮电大学,2019.

[25] 邹振麟,赵杉杉,马敏玲,左欣. 基于 Nginx 反向代理的主机管理系统设计与实现[J].电脑编程技巧与维护,2020(02):74-76.

[26] 高原. 基于 Nginx 的 web 服务器负载均衡策略研究[D].海南:海南大学,2019.

[27] 徐士川. 电子商城系统中订单模块与秒杀模块的设计与实现[D].南京:南京大学,2018.

[28] 李文坚. 大数据背景下 DDoS 常见攻击及其治理和缓解[J].网络安全技术与应用,2020(03):21-23.

[29] 彭海平. 电子商务平台的性能优化和高可靠性研究与实现[D].上海:上海交通大学,2007.

[30] 陈大才. 基于 Nginx 的高并发访问服务器的研究与应用[D].中国科学院大学(中国科学院沈阳计算技术研究所),2018.

[31] 陈海洲. 分布式跨域单点登录模型的研究与应用[D].东南大学,2015.

[32] Ganesan C . Introduction to Django[J]. Journal of Child Neurology, 2008, 25(11):1444-9.
[33] 严成武. 支持分库分表和读写分离的 ORM 框架的设计与实现[D]. 哈尔滨工业大学, 2016.

致谢

岁月如梭，转眼间，两年的研究生求学生活即将结束，站在毕业的门槛上，回首往昔，奋斗和辛劳成为丝丝的记忆，甜美与欢笑也都尘埃落定。首都经济贸易大学以其优良的学习风气、严谨的科研氛围教我求学，以其博大包容的情怀胸襟、浪漫充实的校园生活育我成人。值此毕业论文完成之际，我谨向所有关心、爱护、帮助我的人们表示最诚挚的感谢与最美好的祝愿。

首先，感谢我的导师张军教授。感谢张老师对我学习、生活无微不至的关怀与照顾，每每与张老师聊天，都觉得自己的思想境界得到了更高层次的升华！永远记得张老师对我们“高山仰止、景行行止”的教诲。相信在未来的工作、生活和学习中，会经常记起来张老师对我们的教诲，可以说，张老师与我们交谈的内容是我们一生的精神财富。

感谢我的家人，特别是撰写论文这段繁忙而充实的日子里，他们给了我许多帮助，没有他们的支持和鼓励，我的工作和学习是不可能完成的。

最后，要衷心感谢参加答辩审阅论文的各位教授、专家。