仲恺农业工程学院

毕 业 设 计

基于Web系统的性能优化研究

───以网上购书系统为例

姓 名 黄华冬

院（系） 信息科学与技术学院

专业班级 计算机科学与技术143班

学 号 201420224330

指导教师 赵爱芹 黄景朝（校外）

职 称 讲师 高级工程师

论文答辩日期 2018年5月20日

仲恺农业工程学院教务处制

**Research of Performance Optimization Based**

**on Web**

───**With Online BookShop for Example**

HUANG Hua-dong

College of Information Science and Technology

Zhongkai University of Agriculture and Engineering

Guangzhou, China

**Supervisor: Lecturer Zhao Aiqin**

**Senior Engineer Huang Jingchao**

**（Extramural Tutor）**

**学生承诺书**

本人郑重承诺：所呈交的毕业论文（设计）是本人在导师的指导下独立进行研究所取得的成果。除了文中已用特别标志加以标记的引述内容之外，本设计不含有任何其他个人或集体已经发表或撰写的研究成果。对本设计研究做出过重要贡献的个人或集体，均已在文中以明确的方式标明。若在毕业论文（设计）的各项检查、评比中被发现有抄袭、剽窃或其他的违规行为，本人愿按学校有关规定接受处理，并承担相应的法律责任。

学生签名： 日 期： 年 月 日

摘 要

随着互联网的飞速发展，基于Web的系统用户数不断攀升，存储数据不停累积，导致网站的性能逐步下降，在一定程度上影响了用户的体验。因此，性能优化成为了现代网站开发中亟需解决的问题。

本系统以网上购书系统为例，在实现其基础功能的基础上，对核心功能进行优化。优化思路包括以下四个方面：数据库中创建索引加快查询速度，创建存储过程降低复杂业务的执行时间；系统后台使用Redis缓存热数据，以减少对磁盘数据库的访问；对前后端的交互数据进行GZip压缩，减少网络资源的消耗；系统前端使用Nginx做负载均衡，将高并发的用户请求负载分配到多个后台服务器，减轻单个服务器的负载压力。经过以上四方面的优化，系统能够提供更佳的用户体验。

关键词：性能优化 索引 存储过程 缓存 负载均衡

**Abstract**

With the rapid development of Internet, the amount of users on the Web Based is rising, and the storage data accumulates continuously, which leads to the gradual decline of the performance of the web site, and it affects the user's experience. Therefore, performance optimization has become an unavoidable problem in modern website development.

In this system, with online book-shop for example, and optimizes some of its functions based on the realization of its basic functions. The optimization idea includes the following four aspects: the database creates the index to speed up the query efficiency and creates the procedure to reduce the execution time of the complex business; the backend server uses the Redis caching the hot data to reduce the access to the disk database; GZip compresses the interactive data between the frontend and backend to reduce the consumption of the network resources; The frontend uses Nginx as a load balancing server, assigning high concurrent user request load to multiple backend servers, reducing the load pressure of each single server. Through the above four aspects of optimization, the application can provide a better user experience.

**Keywords:** performance optimization; index; procedure; cache; load balancing

目 录

[1 前言 1](#_Toc515354581)

[1.1 研究目的与意义 1](#_Toc515354582)

[1.2 国内外研究状况 1](#_Toc515354583)

[1.3 研究内容 3](#_Toc515354584)

[1.4 本文所做工作 4](#_Toc515354585)

[2 网上购书系统实现 5](#_Toc515354586)

[2.1 需求分析 5](#_Toc515354587)

[2.1.1 问题陈述 5](#_Toc515354588)

[2.1.2 可行性分析 5](#_Toc515354589)

[2.1.3 功能描述 5](#_Toc515354590)

[2.2 系统设计 6](#_Toc515354591)

[2.2.1 架构设计 6](#_Toc515354592)

[2.2.2 接口设计 6](#_Toc515354593)

[2.2.3 核心类图 9](#_Toc515354594)

[2.2.4 数据库设计 11](#_Toc515354595)

[2.3 系统实现 16](#_Toc515354596)

[2.3.1 项目搭建 16](#_Toc515354597)

[2.3.2 核心功能实现 18](#_Toc515354598)

[3 优化策略 20](#_Toc515354599)

[3.1 数据库 20](#_Toc515354600)

[3.1.1 创建索引 20](#_Toc515354601)

[3.1.2 使用存储过程 21](#_Toc515354602)

[3.2 后台数据缓存 22](#_Toc515354603)

[3.3 静态资源复用与传输内容压缩 23](#_Toc515354604)

[3.3.1 CDN加速 23](#_Toc515354605)

[3.3.2 启用GZip压缩 23](#_Toc515354606)

[3.4 负载均衡 24](#_Toc515354607)

[4 优化实现 24](#_Toc515354608)

[4.1 数据库优化 24](#_Toc515354609)

[4.1.1 图书表创建索引 25](#_Toc515354610)

[4.1.2 编写购买图书存储过程 25](#_Toc515354611)

[4.2 使用Redis缓存 27](#_Toc515354612)

[4.2.1 集成Redis 27](#_Toc515354613)

[4.2.2 代码封装对Redis的操作 27](#_Toc515354614)

[4.2.3 缓存图书信息 28](#_Toc515354615)

[4.3 服务器开启GZip压缩 29](#_Toc515354616)

[4.4 Nginx反向代理后台服务器实现负载均衡 30](#_Toc515354617)

[4.4.1 Nginx配置 30](#_Toc515354618)

[4.4.2 共享Session 31](#_Toc515354619)

[5 系统测试 31](#_Toc515354620)

[5.1 测试目的及任务 31](#_Toc515354621)

[5.2 测试计划 32](#_Toc515354622)

[5.2.1 功能测试 32](#_Toc515354623)

[5.2.2 性能测试 33](#_Toc515354624)

[6 总结与展望 36](#_Toc515354625)

[参考文献 38](#_Toc515354626)

[致谢 39](#_Toc515354627)

仲恺农业工程学院[本科毕业论文(设计)成绩评定表 40](#_Toc515354628)

# 1 前言

## 1.1 研究目的与意义

Web（World Wide Web）即全球广域网，也称为万维网，因其具有传播信息容量大、形态多样、迅速方便、全球覆盖、自由和交互等特点，成为了如今最新以及最流行的传播媒体。今天，Web带给我们的影响是方方面面的，它对我们的工作、生活乃至学习等都提供了极大的便利，促进了社会的发展。然而，随着Web网站的日益普及，使用Web的群体也就变得越来越大，致使Web网站的性能问题变得尤为突出。因此，为了提供快速的用户体验，Web网站的性能优化不可避免地成为了当下人们研究的一个重要课题。

一个优秀的网站，应该做到利用尽可能少的资源，为更大的用户群体提供更快捷的服务。如果一个网站没有很好的利用资源或者响应速度不够迅速，对于开发商和使用客户来说都是一种损失。从用户的角度，对Web站点的满意与否，除了该站点能提供怎样的服务以外，主要以响应速度来衡量，用户并不关心服务的提供细节，他们希望的是在可等待的时间范围内能够使用到提供的服务。如果服务的提供超出了用户的忍耐限度，那么无论该Web站点提供服务再怎么丰富，也不是一个优秀的网站，甚至可以说是一个不及格的Web站点。

影响站点访问速度的因素有很多。客观因素有硬件、软件以及网络等，硬件因素包括服务站点的CPU处理速度、RAM的大小等，软件因素包括使用什么类型的服务器、软件包的大小、应用的实现等，网络因素主要是带宽的限制。主观因素主要是并发问题，例如站点在同一时间访问用户过多，导致服务器响应速度下降，使得用户体验不佳。因此，网站的性能优化应该综合考虑以上各种因素，不断调优，找到一个能够充分发挥站点各方面优势的平衡点，而不是片面优化，否则结果往往会适得其反。

随着Web在日常生活中的应用日益普及，用户对网站的使用越发频繁，要求也随之越来越高。网站是对用户服务的，所以用户对网站的体验与评价体现了网站的价值所在，尤其是在网络营销水平普遍提高的背景下，网站想要在竞争中占据有利位置，一定离不开网站的性能优化。对于新建网站，将网站性能优化融入到网站建设中可以大大提高网站的运营效果以及节省网站优化改造的费用。由此，本课题的研究意义在于对Web网站系统的性能优化，使得资源利用最大化，让互联网用户在资源有限的情况下可以获得更好的用户体验。

## 1.2 国内外研究状况

Web起源于Internet，是建立在Internet上的一种网络服务，最初的目的在于使全球的科学家能够利用Internet交流自己的工作文档。发展到今天，Web早已超过了网络的范畴，它使得足不出户便知天下事，甚至于办天下事，可以说它是信息社会的一个缩影，促进了网络经济雏形的形成。由于Web具有传播信息容量大、形态多样、迅速方便、全球覆盖、自由和交互等特点，已经发展成为新的传播媒体。伴随着浏览器和服务器技术的发展，制作一个动态的、富于创造性的Web站点变得更加容易，Web设计限制的存在是由于技术或浏览器的缺陷所致，Web设计者的工作就是解决这些缺陷。从性质上来说，Web设计师一种多媒体交互设计，即人机交互界面设计。从形态上来说，Web设计与平面设计有一定的相似之处。在Web设计中，经常犯的一个错误是为设计者设计站点，所考虑的是设计者的需求而不是实际用户的需求。如果设计者没有从用户的角度考虑问题，那么所设计出来的Web站点的质量将会大打折扣，所以在Web设计的过程中，需要明确的是以用户的需求进行设计。

数据显示，加载网页的时间过长是Web用户常见的投诉之一，也是最明显的问题。Web开发者的主要问题是：认为自己的平台满足一个目标用户群。但用户的计算机配置与互联网连接速度各不相同，所以开发者应能满足用户的最低要求。对一个网站成功与否的一个重要评估标准是Web的性能。通用指标为：处理时间（指服务器CPU占用率，一般达到70%时，服务就接近饱和）、可用内存数、物理磁盘读写时间等。而Web性能最重要的一个指标是用户所感受到的延迟，即从用户点击一个URL到看到页面显示在屏幕上的额时间间隔。另一个重要的指标是对网络带宽的消耗。一个不合适的方案可能使用过多的网络资源来传送相同的信息，其潜在的影响是引起网络拥塞而导致所有Internet上的应用降级，Web也不能幸免[1]。

与国内行业情况相比，国外行业的Web系统主动权基本上是掌握在高层领导手里。无论是网站本身，还是网站设计公司或者客户，对网站系统的优化都有比较正确的认识。而国内公司对网站优化还不是有很深的认识，而是仅仅停留在实现的层次上，大多数技术人员认为高性能的硬件以及大流量的宽带足以满足对页面的要求。美国是当今信息技术的中心，也是Web的发源地，基于这些方面的原因，美国对于Web的优化设计方面研究起步比较早，而且绝大多数ICP（Internet Content Provider，互联网内容提供商）对于此方面问题也有足够的重视。美国比较重视对Web网站开发的资金投入和总体设计流程，并且对整个网站的架构设计、布局以及日后的优化和维护上也考虑的很细微，针对Web设计中常见的错误，从构思、设计、编码、测试一直到发布和维护全方面着手以达到优化站点性能的目的[1]。因此，欧美的Web整体设计水平较高，Web用户的使用情况也比较好。

## 1.3 研究内容

本文从Web系统的组成结构出发，主要从以下四个方面进行研究：

（1）数据库方面的性能优化

随着用户量的不断攀升以及用户对网站越发频繁的访问，数据库中存储的数据量变得越发庞大，导致的结果是应用程序对数据的操作变得越发缓慢，尤其是查询语句。对于查询操作比较频繁、更新操作较少的系统，通过合理建立的索引能够大幅减少查询的时间。数据库索引就像一本书的目录，可以加快从表中检索行的速度，索引中存储了由数据库的一列或几列生成的键，通过将这些键存储在特定的数据结构中（比如BTree），查询的时候类似一种折半查找，能够使SQL快速有效地查找与键值关联的行。

（2）服务器端的性能有优化

根据著名的2/8法则，任何一组东西中，经常使用的只有其中的一小部分，约占20%，余下的80%尽管是多数，却是很少被应用到的。Web系统也不例外，对于数据库中存储的数据，我们频繁使用的也只是其中的一些部分，对于一些不常变更而且查询比较频繁的数据，如果将它们缓存在内存中，可以大大提高前端对数据的获取效率。不管数据库的性能多好，一个简单的查询也要消耗毫秒级的时间，而且受限于硬盘IO性能，每次都从数据库中查询将会消耗大量的资源，从内存中获取数据远比数据库查询效率要高，更加迅速，因此，合理利用数据缓存能够进一步加快数据获取的速度，缩短浏览器的响应时间，提高系统的用户体验质量。

针对一些特殊的业务，比如商品的秒杀，可以预见到其并发访问量会很高，而且涉及到数据库事务的操作，要求数据的一致性。按照传统的习惯，业务操作基本都是在代码层面实现，面向接口编程，主要便于后期的维护，而对于类似商品秒杀这些特殊业务，因为预见性很强，而且相对固定，可以将业务操作嵌入到数据库的存储过程中，相对于执行SQL语句，存储过程只在创造时进行编译，以后直接调用即可，而SQL语句每执行一次就编译一次，因此使用存储过程可提高数据库执行速度。此外，程序容易出现BUG不稳定，而存储过程存储在数据库中，相对来说更加稳定。

（3）网络方面的优化

服务器端与用户客户端之间的所有数据信息的交互，都是通过网络进行传输的。通过对传输的数据进行压缩后再传输，加快数据传输的同时也能够减少网络带宽的资源消耗，数据量越大，压缩的效果越明显。

（4）部署方面的优化

Web应用出现初期，处理的都是一些企业管理上的业务，用户量比较少，使用一台服务器足够撑起整个应用的正常运行。随着Web的不断发展，服务的业务变得复杂，用户量也逐步提升，受限于硬件，单台服务的应用已经难以支撑整个项目流畅运行，而且容易出现由于单点故障导致整个系统服务不可用的问题。此时，需要对服务进行集群，通过添加冗余单元，将应用部署到多台服务器上，对外提供一个公共接口，根据负载均衡策略，将请求分发到各个服务器，减低单个服务的请求量，从而提升整个系统的并发请求量。

## 1.4 本文所做工作

本论文将先概述Web系统的起源及优化的思路，再以网上购书系统作为案例切入，针对其业务特性做相应的优化方案及实现，接着以系统测试模块对功能实现与优化进行验证，最后，以总结与展望作为本论文的收尾。

（1）前言

对选题背景及应用价值做出大致分析，描述了Web系统性能优化的必要性和可行性，并对系统性能优化思路做了简单描述。

（2）网上购书系统实现

使用传统的构建Web系统的方法实现一个网上购书系统，展示从需求分析到系统实现的一个过程，作为优化的实验样例。

（3）优化策略

针对实现的网上购书系统样例，从多个角度对系统功能进行分析，找出可以优化的方面，制定相应的优化方案。

（4）优化实现

对优化的过程进行展示，包括数据库存储过程的编写、应用程序后端缓存的配置以及应用部署等具体过程。

（5）系统测试

介绍测试时使用的工具，制定测试计划对系统进行测试，分析测试数据以验证优化的作用。

（6）总结与展望

该部分为本论文的收尾，对本文做了大体总结和本系统的优缺点分析，以便更好的认清设计的缺陷和有待完善的地方。

# 2 网上购书系统实现

## 2.1 需求分析

### 2.1.1 问题陈述

随着互联网的发展以及大众理念的逐步开放，网上购物已经成为了大众购物的方式之一。中国互联网络信息中心2006年发布的《2006年中国C2C网上购物调查报告》显示，与传统的线下购物相比，超过一半的用户认为网上购物送货上门比较便捷，接近45%的用户认为网上可以购买到本地没有的商品。显而易见，这些优势是传统购物方式不可比拟的，不断促进了电子商务的快速稳定发展。

### 2.1.2 可行性分析

从卖家的角度看，与传统购物方式比较，优势在于，不受限于地域及营业时间，即消费者不论在任何地方、任何时间都可以对商品进行浏览和购买，只要能够访问互联网；节省了因经营现实场地而需要支付的一些费用，包括管理费、房租等；还可以规避线下经营的一些风险，比如盗窃行为的发生。

从买家的角度分析，电子商务能够使其随时随地浏览全国各地的商品，与传统购物方式相比，这大大节约了买家的时间成本；除此之外，买家能够迅速地比较同一种商品在不同店铺上的一些信息，例如价格、质量等，从而能够购买自己认为性价比高的一间商品，很大程度上满足了消费者以付出尽可能小成本得到尽可能大的效用的一种消费心理。

### 2.1.3 功能描述

基于以上两小点的阐述，使用基于用例的面向对象需求建模的方法做需求分析，相对于传统的结构化需求建模，面向对象建模更易于被用户所理解，因为它是站在用户的角度从系统外部来描述系统的功能，其中用到的用例方法可以作为开发人员与用户之间针对系统需求进行沟通的一个有效手段[2]。

首先确定系统的参与者，有会员、游客、商家和管理员；其次确定系统有哪些用例，即参与者对系统的操作，游客可以查看图书信息、会员可以购买图书、商家可以添加图书、管理员可以审核图书等。系统核心用例图如图1所示。



图1 系统用例图

## 2.2 系统设计

### 2.2.1 架构设计

传统Web系统的组成一般包括以下四层：展示层、控制层、服务层以及数据存储层。其中数据存储层负责与数据库的交互，包括数据的增删改查；服务层作为核心层，从代码层面上实现业务的逻辑；控制层相当于一个中介的角色，将服务层输出的数据交给展示层进行渲染，将展示层输入的数据分发给服务层进行业务逻辑处理；展示层只着重于对数据的接收及渲染。它们之间的关系如图2所示。

### 2.2.2 接口设计

根据2.1.3小节面向对象的用例图展示，系统应该有哪些功能已经一目了然了。接下来，我们应用面向对象的设计方法，结合系统的层次架构对系统功能进行设计，旨在画出用户使用系统核心功能的时序图，从而确定系统的接口。

（1）查看图书时序图

根据查看图书的功能描述，查看图书的时序图如图3所示。

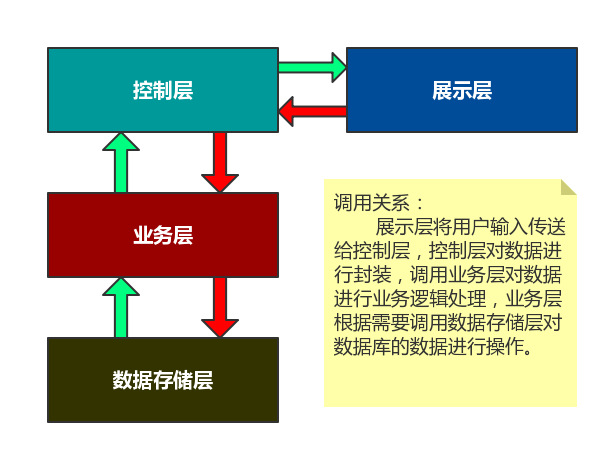


图2 系统用例图



图3 查看图书时序图

说明：会员申请查看图书信息，通过“图书表单”向“图书控制者”发出获得图书列表请求，图书实体类从数据库中取得符合条件的图书信息列表，交付“图书表单”加以显示。会员可以进一步获取图书的详细信息，向“图书控制者”请求图书的详细信息，包括评论信息，这样，图书实体将从数据库获取到图书的基本信息和评论信息，将数据通过“图书表单”进行展示。

由此得知，图书表单的方法有：提交搜索条件、显示图书列表、查看图书详细信息、显示图书详细信息；图书控制者的方法有：请求图书列表、请求图书详细信息；图书的方法有：获得图书列表、获得图书详细信息。

（2）用户订单管理时序图

对其用例图进行面向对象的分析设计，用例订单管理时序图如图4所示。



图4 用户订单管理时序图

说明：会员确定购买商品后，填写订单信息，“订单管理控制者”对来自“订单管理表单”提交的订单信息进行校验，验证通过后，订单实体类就负责向数据库中添加一条新订单信息。会员查看个人订单信息，通过“订单管理表单”向“订单管理控制者”发出获得其信息的请求，订单实体类从数据库中取得该会员的订单信息，交付“订单管理表单”加以显示。然后可以对已经收到商品的订单进行确认收货，“订单管理表单”请求其收货确认，如果会员确认收货，则发送修改的信息给“订单管理控制者”，由其向订单实体类发送确认收货的消息，订单实体类收到消息后，更新这条订单在数据库中的状态。

由此得知，订单管理表单的方法有：提交订单信息、查看个人订单列表、查看订单详情、确认收货、显示个人订单信息；订单管理控制者的方法有：接收订单信息、请求个人订单列表、请求订单详情、修改订单状态；订单的方法有：保存订单信息、获得订单列表、获得订单详情、修改订单状态。

### 2.2.3 核心类图

经过2.2.2小节的时序图展示，我们可以很清晰的知道系统应该具备哪些接口，向用户提供怎样的服务，对这些信息进行整合就可以等到系统的类图了。下面仅对图书管理和订单管理两个模块描绘系统的核心类图。

（1）图书管理类图

图5显示了图书管理用例的类/对象及各个类之间的关系，其中各个类增加了属性以及方法。图书类(Book)的作用是为其它类之间的交互提供消息传递和结果返回。获取图书信息时，BookService类会从BookRepository类中获取，获取成功之后，会将图书信息返回给BookController类，由BookController转给页面展示。

（2）订单管理类图

图6显示了订单管理用例的类/对象以及各个类之间的关系，其中各个类增加了属性以及方法。订单类(Order)的作用是为其它类之间的交互提供消息传递和结果返回。添加订单时，首先从购物车中获取商品信息，判断购买的商品数量是否超过库存量，然后组合订单信息，包括用户信息、收货地址信息以及商品信息。保存订单信息的同时，需要修改商品的库存量以及一些销售信息，还要将购买的商品从购物车中移除。



图5 图书管理类图



图6 订单管理类图

### 2.2.4 数据库设计

（1）数据分析

在开始数据库设计之前，需要对系统搭建所需的数据进行分析[3]。根据上一节的用例规约，分析系统存在的实体、属性以及实体间的关系如下：

① 会员信息

会员信息有7个属性（主键id、用户编号、用户名、登录密码、用户头像、电子邮件、注册时间）。会员需要注册个人信息，才能登录系统，执行添加订单、评论图书操作。

② 会员地址信息

地址信息包括5个属性（主键id、用户编号、收货地址、联系人姓名、联系电话）。一个用户可以维护多个地址，为用户添加订单收货地址时提供选择。

③ 店家信息

店家信息区别于会员信息，便于管理员的管理，有以下9个属性（主键id、店家编号、店家账号、登录密码、身份证号、联系电话、店铺编号、店铺名称、注册时间）。店家需要注册个人信息，才能够申请开店铺、管理店铺图书以及店铺订单。

④ 店铺信息

店铺信息有13个属性（主键id、店铺编号、店铺名称、店铺logo、营业执照、销售额、销售量、评论数、评分、店铺简介、创建时间、店家编号，店家账号）。一家店铺由一个店家管理，存储了店铺经营的许可信息，以及经营过程中的一些销售记录汇总。

⑤ 图书信息

图书是系统的核心，有以下21个属性（主键id、图书编号、图书名称、所属店铺编号、所属店铺名称、类别、价格、销售额、销售量、库存量、评论数、评分、作者、出版社、出版时间、上架时间、下架时间、状态、ISBN、简介、封面）。除了一些基本信息，图书还包括了所属店铺信息以及售卖过程中的汇总记录，向消费者购买图书时提供了多方面的参考信息。

⑥ 订单信息

订单信息至少包括15个属性（主键id、订单编号、订单流水号、下单时间、买家编号、买家名称、收货地址、联系人、联系电话、订单总价、订单总商品数、状态、完成时间、店铺编号、店铺名称）。一个用户可以有多条订单，店铺同理，所以订单信息包含了买家信息和店铺信息，便于买家和卖家对订单进行管理。

⑦ 订单商品信息

订单商品保存了订单对应的商品信息，有7个属性（主键id、订单编号、商品编号、商品名称、商品价格、购买数量、商品封面）。一条订单信息可以对应店铺内多件商品信息，这样可以减少订单的条数，也方便商品的统计与管理。

⑧ 图书评论信息

当买家对购买图书评价时，会生成一条图书评论信息，包括以下属性（主键id、用户编号、用户名称、图书编号、图书名称、店铺编号、店铺名称、订单编号、评分、评论内容、评论时间）。只要购买了该图书，才能对图书进行评价，图书评论信息能够向后续买家提供一个重要的参考，一本图书可以有多个评价。

（2）ER图设计

经过上述的数据分析，可以得出系统至少包含8种实体，分别是会员、地址、店家、店铺、图书、图书评论、订单、订单商品，系统的ER图如图7所示。

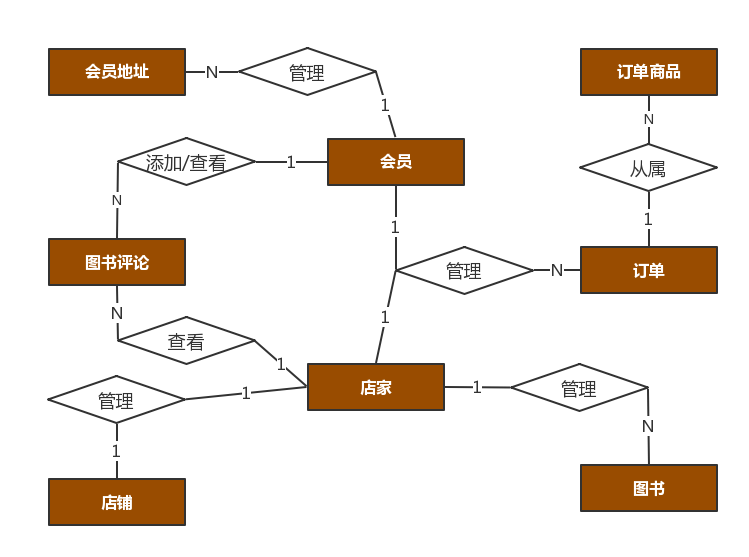


图7 系统ER图

（3）数据库表设计

经过了数据分析以及ER图的设计，系统至少需要8张表对数据进行存储。以下是每张表的设计与实现。

① 用户表（tb\_user）

用户表存储会员基本信息，其结构如表1所示。

表1 用户表（tb\_user）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 注释 | 说明 |
| Id | bigint(20) | 主键id | 系统主键 |
| user\_id | varchar(100) | 用户编号 | 业务主键 |
| username | varchar(100) | 用户名 |  |
| password | varchar(100) | 密码 | 加密存储 |
| email | varchar(100) | 电子邮件 |  |
| user\_avatar | varchar(100) | 用户头像 | 存储图片路径 |
| registry\_time | datetime | 注册时间 |  |

② 地址表（tb\_address）

地址表用于保存用户的收货地址信息，其表结构如表2所示。

表2 地址表（tb\_address）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 注释 | 说明 |
| id | bigint(20) | 主键id | 系统主键 |
| user\_id | varchar(100) | 用户编号 | 地址所属用户 |
| address | varchar(100) | 收货地址 | 比如xx市xx街 |
| link\_man | varchar(100) | 收货联系人 |  |
| link\_phone | varchar(100) | 联系人电话 |  |

③ 店家信息表（tb\_shop\_owner）

店家用于保存卖家的个人信息，其表结构如表3所示。

表3 店家信息表（tb\_shop\_owner）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 注释 | 说明 |
| id | bigint(20) | 主键id | 系统主键 |
| owner\_id | varchar(100) | 店家编号 | 业务主键 |
| owner\_name | varchar(100) | 店家名称 |  |
| owner\_password | varchar(100) | 登录密码 | 加密存储 |
| id\_code | varchar(100) | 店家身份证号 | 保证信息可靠性 |
| phone | varchar(100) | 店家联系方式 |  |
| shop\_id | varchar(100) | 店铺编号 | 拥有的店铺 |
| shop\_name | varchar(100) | 店铺名称 |  |
| registry\_time | datetime | 注册时间 |  |

④ 图书表（tb\_book）

图书表存储者图书的基本信息以及售卖过程中的一些汇总记录信息，其表结构如表4所示。

表4 图书表（tb\_book）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 注释 | 说明 |
| id | bigint(20) | 主键id | 系统主键 |
| book\_id | varchar(100) | 图书编号 | 业务主键 |
| book\_name | varchar(100) | 图书名称 | 用作索引，加快查询 |
| shop\_id | varchar(100) | 店铺编号 | 图书所属店铺 |
| shop\_name | varchar(100) | 店铺名称 |  |
| book\_class | varchar(100) | 图书类别 |  |
| book\_price | double(10,2) | 图书价格 | 单位（元） |
| book\_sales | double(10,2) | 图书销售额 | 单位（元） |
| sale\_volumes | int(11) | 图书销售量 | 单位（本） |
| book\_storage | int(11) | 图书库存量 | 单位（本） |
| book\_comments | int(11) | 图书评论数 | 单位（条） |
| comment\_score | double(10,2) | 图书评分 |  |
| book\_author | varchar(100) | 图书作者 |  |
| book\_publisher | varchar(100) | 图书出版社 |  |
| publish\_time | datetime | 出版时间 |  |
| up\_time | datetime | 图书上架时间 |  |
| down\_time | datetime | 图书下架时间 |  |
| book\_state | varchar(100) | 图书状态 | 上架/下架/库存不足 |
| book\_isbn | varchar(100) | 国标图书编码 |  |
| book\_descript | varchar(255) | 图书简介 |  |
| book\_image | varchar(100) | 图书封面 | 存储图片路径 |

⑤ 订单商品表（tb\_order\_product）

订单商品表用于保存订单包含的商品信息，其表结构如表5所示。

表5 订单商品表（tb\_order\_product）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 注释 | 说明 |
| id | bigint(20) | 主键id | 系统主键 |
| order\_id | varchar(100) | 订单编号 | 商品所属订单 |
| book\_id | varchar(100) | 商品编号 |  |
| book\_name | varchar(100) | 商品名称 |  |
| book\_price | double(10,2) | 商品价格 |  |
| book\_count | int(11) | 商品购买数量 |  |
| book\_image | varchar(100) | 商品图片信息 | 存储图片的路径 |

⑥ 图书评论表（tb\_book\_comment）

图书评论表存储用户对图书的评论信息，其表结构如表6所示。

表6 图书评论表（tb\_book\_comment）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 注释 | 说明 |
| id | bigint(20) | 主键id | 系统主键 |
| user\_id | varchar(100) | 用户编号 | 评论用户 |
| username | varchar(100) | 用户名 |  |
| book\_id | varchar(100) | 图书编号 | 评论的图书 |
| book\_name | varchar(100) | 图书名称 |  |
| shop\_id | varchar(100) | 店铺编号 | 图书所属店铺 |
| shop\_name | varchar(100) | 店铺名称 |  |
| order\_id | varchar(100) | 订单编号 |  |
| comment\_score | int(11) | 评论分数 |  |
| comment\_content | varchar(255) | 评论内容 | 用户对图书的评价 |
| comment\_time | datetime | 评论时间 |  |

⑦ 订单表（tb\_order）

订单表存储用户购买图书时的一些基本信息，其表结构如表7所示。

表7 订单表（tb\_order）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 注释 | 说明 |
| id | bigint(20) | 主键id | 系统主键 |
| order\_id | varchar(100) | 订单编号 | 业务主键 |
| order\_no | varchar(100) | 订单流水号 | 店铺编号+当前时间戳 |
| order\_time | datetime | 下单时间 |  |
| user\_id | varchar(100) | 用户编号 | 订单所属用户 |
| username | varchar(100) | 用户名 |  |
| order\_address | varchar(100) | 订单收货地址 | e.g. xx市xx街xx号 |
| link\_man | varchar(100) | 收货人 |  |
| link\_phone | varchar(100) | 收货人联系电话 |  |
| order\_price | double(10,2) | 订单商品总价格 | 订单包含商品价格之和 |
| order\_amount | int(11) | 订单商品总数 | 订单包含的商品总数 |
| order\_state | varchar(100) | 订单状态 | e.g. 待处理、已完成etc |
| complete\_time | datetime | 订单完成时间 | 买家确认收货的时间 |
| shop\_id | varchar(100) | 店铺编号 | 订单所属店铺 |
| shop\_name | varchar(100) | 店铺名称 |  |

⑧ 店铺表（tb\_shop）

店铺表存储店铺的基本信息以及营业期间的售卖记录汇总，其表结构如表8所示。

表8 店铺表（tb\_shop）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 注释 | 说明 |
| id | bigint(20) | 主键id | 系统主键 |
| shop\_id | varchar(100) | 店铺编号 | 业务主键 |
| shop\_name | varchar(100) | 店铺名称 |  |
| shop\_license | varchar(100) | 店铺营业执照 | 图片路径 |
| shop\_sales | double(10,2) | 销售额 | 单位（元） |
| shop\_sales\_volumes | int(11) | 销售量 | 单位（本） |
| shop\_score | double(10,2) | 店铺评分 |  |
| shop\_descript | varchar(255) | 店铺简介 |  |
| shop\_owner\_id | varchar(100) | 店家编号 | 店铺归属人 |
| shop\_owner\_name | varchar(100) | 店家名称 |  |

## 2.3 系统实现

### 2.3.1 项目搭建

项目后台采用SpringBoot+Mybatis的框架组合，以Eclipse作为开发工具，项目层级如图8所示，其中domain包下存储的是与数据库表相对应的实体类，比如图书类(Book)、订单类(Order)； repository包则封装了MyBatis对MySQL数据库的CRUD操作，其中的SuperRepository<T>类是一个通过的模板，根据调用者传入的不同泛型T相应的数据库表执行操作；service包是本项目的核心，里面包含了系统业务实现的所有细节；web包中，子包api向前端提供访问接口，interceptor用做权限判断，exception则是对前端的一些错误访问做异常处理。

项目前端则选择Vue.js开发框架，SublimeText作为开发工具，图9展示了前端项目层级结构。其中api目录下存放了用于访问后台数据的接口；asset、style这两个目录存放的是全局的静态资源，比如图片；pages和router这两个目录，前者存放的是页面，后者存放的是页面之间的跳转关系。

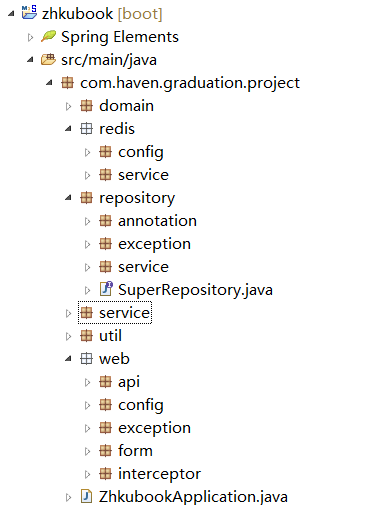


图8 系统后台包结构

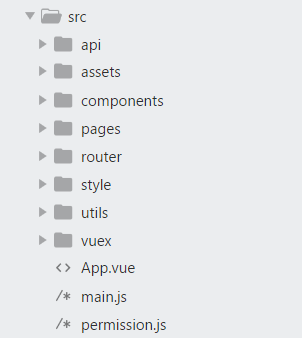


图9 项目前端包结构

### 2.3.2 核心功能实现

本环节将对系统部分的核心功能进行实现，分别是图书查询、添加订单的实现过程、以及实现截图。对于部分关键功能的代码流程，进行截图展示，从而更进一步地熟悉功能的实现原理。

（1）图书查询

以下内容展示的是查询图书前端JS代码，首先获取用户的搜索关键字，然后根据关键字向后台发起查询，查询成功将数据渲染到页面，否则提示错误信息。

|  |
| --- |
| // 查询图书  getSearch() {  // 1. 获取用户输入的查询关键字  if(this.$route.query) {  let query = this.$route.query;  this.searchForm = {  bookname: query.bookname,  bookclass: query.bookclass,  shopname: query.shopname  };  this.keyword = query.bookname  let params = "?bookname="+ query.bookname;  params += "&bookclass="+ query.bookclass;  params += "&shopname="+ query.shopname;  // 2. 向后台发起请求  getBookList(params).then(res => {  // 请求成功，渲染数据  this.page = res.data;  this.result = this.page.data;  this.total = this.page.totalCount;  this.pagesize = this.page.pageSize;  }).catch(err => {  // 请求失败，提示错误信息  if(err.response.data && err.response.data.code === 404) {  this.tip = err.response.data.msg;  // this.$message.warning(err.response.data.msg);  }  this.total = 0;  this.result = null;  });  }  } |

后台服务器接收到前端数据请求，根据查询条件生成对应的SQL语句对数据库执行查询，将返回结果通过JSON的格式返回给前端。核心代码如下所示：

|  |
| --- |
| @Override  public Page<Book> books(String bookName, String bookClass, String shopName) {  WhereCondition condition = new DefaultWhereCondition();  condition.like("bookName", bookName);  condition.eq("shopName", shopName);  condition.eq("bookClass", bookClass);  condition.gt("bookStorage", 0);// 库存量大于0  condition.eq("bookState", "1");// 状态为是上架的图书  condition.orderByDesc("bookSales");  condition.orderByDesc("id");  Page<Book> books = bookService.findByPage(Book.class, condition, 10L);  return books;  } |

（2）添加订单

以下内容展示的是用户直接购买图书的前端JS代码。首先获取用户购买的图书信息，接收用户输入的收货地址，然后调用后台接口执行添加订单操作，添加成功给出成功提示，否则给出错误提示及原因。

|  |
| --- |
| // 添加订单前端JS代码  buyBook() {  // 1. 参数准备  let params = {  bookId: this.temporder.bookId,  addressForm: {  address: this.temporder.address,  linkMan: this.temporder.linkMan,  linkPhone: this.temporder.linkPhone  }  }  // 2. 调用后台接口  buyAtOnce(params).then(res => {  // 购买成功提示  this.$message.success('购买成功');  this.dialogVisible2 = false;  }).catch(err => {  let response = err.response;  // 购买失败提示  if(response.data && response.data.code === 400) {  this.$message.error(response.data.msg);  }  });  } |

添加订单是一个事务操作，主要包括订单的添加和库存量的修改，通过编码方式实现事务的操作，核心代码如下所示：

|  |
| --- |
| // 添加订单后台Java代码  @Transactional  @Override  public Order buyAtOnce(String bookId, User user, AddressForm addressForm) {  // 1. 根据图书编号获取图书信息  WhereCondition condition = new DefaultWhereCondition();  condition.eq("bookId", bookId);  Book book = bookService.findByList(Book.class, condition).get(0);  if (book.getBookStorage() < 1) {  throw new IllegalArgumentException(book.getBookName() + "库存量不足");  }  // 2. 添加订单  Order order = new Order(book, addressForm, user);  orderRepository.insertOrder(order);  // 3. 生成订单商品  OrderProduct product = new OrderProduct(book, order.getOrderId());  List<OrderProduct> productList = new ArrayList<>(1);  productList.add(product);  orderRepository.insertOrderProduct(productList);  // 4. 修改店铺销售额、销售量  orderRepository.updateShop(book.getShopId(), book.getBookPrice(), 1);  // 5. 修改图书销售额、销售量及库存量  orderRepository.updateBook(bookId, book.getBookPrice(), 1);  return order;  } |

# 3 优化策略

## 3.1 数据库

以下以MySQL数据库作为讨论，对网上购书系统的常用功能指定优化策略。

### 3.1.1 创建索引

（1）数据库索引的原理及适用场景

创建数据库索引能够加快数据检索的效率，很大程度上提高系统的性能，通过使用索引，能够在查询过程中使用数据库引擎的优化隐藏器，达到加快查询的目的。想知道索引为什么能够加快查询，首先要从MySQL数据库的数据存储方式说起，数据在磁盘上通常是以块的形式存储，访问数据时，数据库为了确保数据操作的原子性，会一起访问所有的数据块，而数据块之间通过类型链表的数据结构联系起来，也就是说一个数据块中除了存储数据之外，还包含了下一个数据块的内存地址。一般情况下，MySQL数据库中对于非键字段的查询采用的是线性检索的方式，最坏情况下查询一条数据需要遍历整张表，时间复杂度为O(n)[4]。如果对数据表中的某个字段创建了索引，数据库会根据该字段的值生成一个摘要，这一整列数据就对应一个摘要集合，然后数据库会将该集合按照类似二叉树的数据结构存储，当再次执行查询时，是通过二分查找的方式，查询时间复杂度降到了O(log2n)，性能有了实质性的提升。

数据库索引是通过以空间换时间的方式加快数据的检索，数据库系统需要额外的内存空间来维护它，因此数据量越大，索引的效果越明显。数据索引适用于作为查询条件、而且极少修改的一些字段，因为索引字段发生修改会导致整张表的索引摘要关系的重新计算，不利于索引的维护。由此，对于经常出现的Where字句和Order字句的且不经常修改的字段比较适合创建索引。

（2）图书名称和类别字段作为索引

根据以上对索引的分析，结合网上购书系统的图书查询业务，非常适合对图书名称和图书类别建立索引，一来图书随着系统的不断使用，图书表中的图书信息会不断的增加，在不创建索引的情况下，图书查询的效率只会越来越低，响应时间的延长严重影响用户体验，二来根据业务特性，图书的名称和类别极少情况会发生改变，总的来说对图书查询条件建立索引很大程度上提升系统的性能。

### 3.1.2 使用存储过程

（1）存储过程的优点及适用场景

存储过程是存储在数据库服务器上的一组预编译的SQL语句集合，类似与操作系统的脚本文件。由于存储过程是存储在数据库端的，数据库在第一次调用时对其进行语法解析、语句优化，然后生成执行计划编译到数据库中，以后只需要调用该执行计划就可以了。相对于传统的SQL语句，每次调用都需要进行编译和优化，因此，存储过程执行的速度会更快。对于复杂的业务操作来说，存储过程的优势更加明显，在应用程序中执行一组事务操作的SQL语句时，由于网络延迟等原因，占用数据库连接的时间更长，这意味着对数据加锁的时间也更长，不利于高并发的访问，而且批处理的SQL语句集合需要占用更多的网络带宽资源进行传输。因此，对复杂业务使用存储过程不仅能够加快SQL语句执行的速度，还可以在一定程度上减少网络资源的消耗。对于处理逻辑经常发生变化的业务则不应该使用存储过程，这样给数据库的维护带来很大的难度。

（2）对购买图书使用存储过程

购买图书的核心业务流程是添加订单、修改图书库存量以及修改店铺的销售量，如果在应用程序代码上实现的话，首先需要开启事务，然后至少需要向数据库发送三条SQL语句，接下来就是等待数据库返回结果之后再释放事务，等待的这段时间持有对数据的锁，不能执行任何操作，也导致其它的线程也不能对被锁的数据进行访问，非常的耗费资源。因此，将购买图书的业务编写成一个存储过程，用户执行购买图书操作时，直接调用存储过程，减少资源消耗的同时，也缩短了系统的响应的时间，提升用户体验。

## 3.2 后台数据缓存

缓存的方式多种多样，包括浏览器缓存、内存缓存和CPU缓存等，本小节讨论的是应用程序后端的缓存，即通过将系统常用的一些数据对象缓存在内存中，在用户获取数据的时候能够直接从内存中获取，从而避免了从数据库中加载数据，达到减少系统资源消耗，提高系统性能的目的。

（1）使用缓存的优点及适用场景

传统的Web应用中，数据大都存储在关系型数据库，每次需要获取数据时都会对数据库进行查询操作。如果将查询结果存储在内存中，当用户请求过来，应用程序先从内存中获取，只有不命中的情况下才会访问数据库，这样就能够避免数据库连接的反复获取与释放以及减少磁盘的I/O操作，节省数据库资源的同时也能够加快系统的响应的时间。另一方面，假如数据库与后台服务器部署在不同区域的机器上，对于传统的查询方式来说，存在网络延迟问题，即应用程序需要等待数据库通过网络传输返回的执行结果，如果缓存了查询结果，能够在一定程度上减缓网络延迟，也节省了网络带宽资源。当然，数据缓存有其局限性，经常变更的数据不宜使用缓存，由于数据的一致性问题，每次数据更新都需要刷新缓存，缓存数据的频繁更新会使得内存抖动较大，失去了缓存的意义所在。其次，不常使用的数据也没必要使用缓存，内存使用资源是昂贵的，将不常用的数据驻留的内存中是一种浪费的行为，与数据缓存空间换时间的思想相违背。因此，如果数据的使用热度高，而且不常变更，对其进行缓存能够很好地提供系统的性能。

（2）缓存图书查询结果

结合网上购书系统的业务，当用户进入系统首页时，系统会从数据库中获取新上架和最近热销的图书信息展示给用户。由于Web系统中首页的访问量远超于其他页面，所以首页的数据访问热度无疑是最高的，对首页数据的缓存，即对新上架和热销图书信息进行缓存是优先的选择。从业务角度分析，这两类数据并不要求强实时性，只要能够保证一段时间内数据值在允许误差范围之内就可以了，这时可以对缓存数据设置过期时间，使其能够在设置的单位时间内刷新一次。由此，对新上架/热销图书信息进行缓存，减少了图书信息的重复查询，缩短了页面获取数据的时间，从而提高了用户使用体验，达到优化系统性能的目的。

## 3.3 静态资源复用与传输内容压缩

从用户使用的角度来说，用户只需要在浏览器地址栏输入请求地址，等待片刻就能够看到服务器返回的页面内容，看似简单的一个操作，里面却蕴含了很多内容。首先Web浏览器对输入地址进行域名解析得到服务器的IP地址，浏览器与服务器之间的连接需要进行路由寻址验证是否能够联通，接下来就是建立TCP连接并由浏览器发起HTTP请求，服务器接收到浏览器的请求之后，将响应内容通过TCP连接返回给浏览器端。

### 3.3.1 CDN加速

CDN（Content Delivery Network）的全称是内容分发网络，是构建在网络之上的，主要提供的功能是内容的缓存和分发，是在客户浏览器端和服务器之间增加缓存层，当用户获取资源时会根据域名解析引导客户端从缓存中获取，减少对源服务器的请求连接，缩短用户响应时间。

网上购书系统前端服务器响应内容的通常是一个HTML页面，而且每个页面中基本都使用了JQuery和Vue.js 的类库，便于编写JS代码实现与用户的交互。这些静态资源都是从本地服务器引入，也就是说客户端每发起一次页面请求，都会顺带的对额外的资源发起请求，无疑增加了服务器的压力，如果将这些常用的资源通过CDN的方式引入，那么应对浏览器的请求，前端服务器只需要响应页面内容，对于页面中需要用到的一些常用库，比如JQuery等，会从离用户位置最近的CDN节点服务器中获取，这样就可以节省网络带宽，加快用户响应时间，提高用户体验。

### 3.3.2 启用GZip压缩

另一方面，用户等待页面响应的时间长短取决于页面的大小，页面越大，占用的带宽资源就越多，响应时间就越长。如果能够在数据进行传输之前对其进行压缩，浏览器接收到之后再对其进行解压缩显示，那么数据传输效率将会得到很大的提升。目前主流的浏览器对解压缩提供了很好的支持，最流行的做法是使用Gzip的Deflate算法对传输数据进行压缩。数据显示，当使用Gzip对一个纯文本文件进行压缩时，压缩比能够达到70%以上，也就是说一个400Kb的HTML页面经过Gzip压缩之后大约减少到100Kb，这取决于文件的内容[5]。数据压缩时将文件中的重复字符使用占位符替换，解压显示时再使用原来的字符将占位符替换掉，结合HTML文件的特性，里面存储着很多重复的标签，所以对其使用Gzip压缩能够明显减少了文件的大小，加快响应页面的传输效率。

## 3.4 负载均衡

系统负载均衡的方式分为软负载均衡和硬负载均衡，硬负载均衡主要通过在服务器和外部网络之间安装负载均衡设备，比如常用的F5负载均衡器，软负载均衡则是在软件层面上实现负载均衡，通过添加中间件（如Nginx）统一接收客户端请求，根据不同的负载策略将请求分发到不同的服务器中进行处理。由于软负载方式成本低廉、配置简单等优点，以下仅从软负载均衡的角度考虑优化策略。

目前常用的Web服务器，以Tomcat8为例，默认情况下最大并发数为200，也就是说同一时间最多只能创建200个线程处理用户的请求，这时候当第201个请求过来时，只能够等待前面请求处理完成有多余线程空闲出来时，才真正响应该请求，执行相应的业务逻辑。随着用户量的不断上涨，受限于硬件原因，单服务器应用已经不足以支撑起成千上万的并发请求操作了，因此，将多台服务器绑定成一个整体，向前端提供一致访问接口的负载均衡机制应运而生。通过服务器的冗余部署，将对服务大流量的集中式请求转化为小流量的分布式请求，减轻单台服务器负载压力的同时，也避免单应用服务中服务器宕机导致整个系统不可用的情况发生，系统健壮性得到了提升。

对于网上购书系统而言，当用户量增长到一定程度之后，可以通过分布式部署的方式以适应用户高并发的请求，由于代码实现中使用Session对用户登录信息数据进行存储，为了避免在A服务器的Session存了值，在B服务器上获取不到的情况发生，需要解决后台服务器之间Session共享的问题。

# 4 优化实现

本节将针对上一节指定的优化策略，采用相关的技术对相应的业务进行优化。

## 4.1 数据库优化

数据库方面的优化主要包括以下两个方面：

（1）为数据库图书表的图书名称字段和图书类别字段创建索引；

（2）将购买图书的业务代码使用数据库存储过程代替。

### 4.1.1 图书表创建索引

以下内容展示的是为图书表的图书名字段和图书类别字段创建索引的SQL语句，使用的是普通索引类型，索引方法则是BTREE，会将索引值经过排序之后组合成类似二叉树的数据结构，加快数据检索的速度。

|  |
| --- |
| # 图书表创建索引  ALTER TABLE `db\_graduation\_project`.`tb\_book`  ADD INDEX `idx\_book\_name`(`book\_name`)  USING BTREE COMMENT ‘以图书名称作为索引’,  ADD INDEX `idx\_book\_class` (`book\_class`)  USING BTREE COMMENT ‘以图书类别作为索引’; |

### 4.1.2 编写购买图书存储过程

以下内容展示的是网上购书系统直接购买图书的存储过程的核心实现，首先开启事务，根据图书编号获取图书信息，判断库存量是否足够，如果不够直接返回0，表示库存量不足，反之执行购买操作，包括添加订单、修改库存等，最后提交事务，返回1，表示购买图书成功。

|  |
| --- |
| BEGIN  -- 声明变量  DECLARE var\_book\_name, var\_shop\_id, var\_shop\_name, var\_book\_image VARCHAR(100);  DECLARE var\_book\_storage INT(11);  DECLARE var\_book\_price DOUBLE(10, 2);  DECLARE t\_error INT(11) DEFAULT 0;  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET t\_error=1; -- 发生异常时设值为1，用于判断提交/回滚事务  -- 获取图书信息  SELECT book\_name, shop\_id, shop\_name, book\_price, book\_storage, book\_image into var\_book\_name, var\_shop\_id, var\_shop\_name, var\_book\_price, var\_book\_storage, var\_book\_image FROM tb\_book WHERE book\_id = var\_book\_id;  -- 判断库存量是否足够，如果足够执行下单操作，返回1，否则什么也不做，返回0  IF var\_book\_storage > 0 THEN -- 库存足够  START TRANSACTION;-- 开启事务  -- 执行下单操作，首先添加订单，其次添加订单商品，然后修改图书库存量、销售额及销售量，最后修改店铺销售额及销售量  -- 添加订单  INSERT INTO  tb\_order(order\_id, order\_no, order\_time, user\_id, username, order\_address, link\_man, link\_phone, order\_price, order\_amount, order\_state, shop\_id, shop\_name)  values(var\_order\_id, var\_order\_no, CURRENT\_TIMESTAMP(), var\_user\_id, var\_username, var\_order\_address, var\_link\_man, var\_link\_phone, var\_book\_price, 1, '0', var\_shop\_id, var\_shop\_name);  -- 添加订单商品  INSERT INTO tb\_order\_product(order\_id, book\_id, book\_name, book\_price, book\_count, book\_image)  VALUES(var\_order\_id, var\_book\_id, var\_book\_name, var\_book\_price, 1, var\_book\_image);  -- 修改图书库存量、销售额及销售量  UPDATE tb\_book  SET book\_storage = book\_storage - 1, book\_sales = book\_sales + var\_book\_price, sale\_volumes = sale\_volumes + 1  WHERE book\_id = var\_book\_id;  -- 修改店铺销售额及销售量  UPDATE tb\_shop  SET shop\_sales = shop\_sales + var\_book\_price, shop\_sale\_volumes = shop\_sale\_volumes + 1  WHERE shop\_id = var\_shop\_id;  IF t\_error = 1 THEN  ROLLBACK;-- 回滚事务  SET ret\_val = 0;  ELSE  COMMIT;-- 提交事务  SET ret\_val = 1;  END IF;  ELSE -- 库存不足  SET ret\_val = 0;  END IF;  END |

以下内容展示的是在应用程序中调用购买图书存储过程的代码实现，主要包括调用前参数的准备以及调用完成后根据返回结果的不同给出不一样的提示。

|  |
| --- |
| // 调用购买图书存储过程Java代码  @Override  public Integer buyAtOnceProc(String bookId, User user, AddressForm addressForm) {  Map<String, Object> params = new HashMap<>();  params.put("userId", user.getUserId());  params.put("username", user.getUsername());  params.put("bookId", bookId);  params.put("orderId", UUID.randomUUID().toString());  params.put("orderNo", DateUtil.getTimeNo());  params.put("orderAddress", addressForm.getAddress());  params.put("linkMan", addressForm.getLinkMan());  params.put("linkPhone", addressForm.getLinkPhone());  Integer retValue = -1;  params.put("retValue", retValue);  orderRepository.buyAtOnceProc(params);  return (Integer) params.get("retValue");  } |

## 4.2 使用Redis缓存

以下针对图书查询结果使用Redis进行缓存，主要包含三部分的工作：

（1）集成Redis，使后台服务器能够连接Redis服务器；

（2）编写代码实现对Redis的访问，主要是获取连接，设置缓存以及获取缓存；

（3）缓存图书信息。

### 4.2.1 集成Redis

由于系统后台使用的SpringBoot框架对Redis进行很好的集成，只需要在项目的POM文件中加入相应的依赖，在配置文件中添加以下配置信息之后[6]，便可以在项目中对Redis服务器进行操作了。

|  |
| --- |
| spring:  redis:  database: 0 #数据库的唯一标识  host: 127.0.0.1 #Redis服务器的IP地址或者域名  port: 6379 #Redis服务器的端口号  password: #连接Redis服务器的密码，默认为空  pool: #Redis连接池的配置  max-active: 512 #连接池最大连接数  max-idle: 64 #空闲时最大连接数  min-idle: 8 #空闲时最小连接数  max-wait: 60000 #最大阻塞时间，为1分钟  timeout: 15000 #获取连接超时时间，为15秒钟 |

### 4.2.2 代码封装对Redis的操作

完成SpringBoot与Redis的集成之后，为了便于后期的管理与维护，需要对Redis服务器的操作进行封装，以下内容展示了封装Redis操作的核心代码，包括向Redis数据库存储数据以及从Redis数据库获取数据。

|  |
| --- |
| // 向Redis数据库存值  @Override  public Boolean set(String key, Object value, Long expireTime) {  if(this.isNullEmpty(key)) {  // 判断key值是否为空  logger.debug("写入缓存失败，key不能为空！");  return false;  }  Boolean success;  try {  // 将键值对存储到Redis服务器中  opsForValue.set(key, value);  redisTemplate.expire(key, expireTime.longValue(), TimeUnit.SECONDS);  logger.debug("写入缓存成功 ========》 {key: "+ key +"}");  success = true;  } catch(Exception e) {  logger.error("写入缓存失败 ========》 {key: "+ key +", errMsg: "+ e.getMessage() +"}", e);  success = false;  }  return success;  }  // 从Redis数据库中取值  @Override  public Object get(String key) {  Object result = null;  try {  result = opsForValue.get(key);  logger.debug("读取缓存成功 ========》 {key: "+ key +"}");  } catch(Exception e) {  logger.error("读取缓存失败 ========》 {key: "+ key +", errMsg: "+ e.getMessage() +"}", e);  }  return result;  } |

### 4.2.3 缓存图书信息

这里以缓存最新上架、最近热销的图书为例，展示优化后的业务逻辑，当用户进入系统首页时，会向应用程序后台发出获取新上架/热销图书信息的请求，后台接收到请求后，根据相应的键值（如“Hot”）从Redis缓存中获取对应的值，如果不为空，直接返回给前端，免去了对MySQL数据库的操作，核心代码如下所示：

|  |
| --- |
| // 从Redis中获取销量前十的图书信息  @SuppressWarnings("unchecked")  @Override  public List<Book> findBookHotSale() {  String key = KeyGeneratorUtil.keyGenerate(Book.class, "HotSale");  List<Book> books = (List<Book>) redisService.get(key);  if(books == null) {  books = bookService.hotSale(null);  if(books != null)  redisService.set(key, books, 1800L);  }  return books;  }  // 从Redis中获取新上架的十本图书信息  @SuppressWarnings("unchecked")  @Override  public List<Book> findBookNew() {  String key = KeyGeneratorUtil.keyGenerate(Book.class, "New");  List<Book> books = (List<Book>) redisService.get(key);  if(books == null) {  books = bookService.bookNew(null);  if(books != null)  redisService.set(key, books, 1800L);  }  return books;  } |

## 4.3 服务器开启GZip压缩

目前主流的服务器和浏览器都对压缩提供了很好的支持，对于网上购书系统使用的SpringBoot框架而言，因为其内嵌了Tomcat服务器，所以开启GZip压缩非常的方便，只需要在配置文件添加如下配置信息[7]：

|  |
| --- |
| server:  compression:  enabled: true #启用GZip压缩功能  min-response-size: 2048 #响应体超过2Kb才执行压缩  mime-types: #配置允许压缩的数据类型  - text/html  - text/plain  - application/json  - application/javascription  - text/css |

## 4.4 Nginx反向代理后台服务器实现负载均衡

Nginx由于其配置灵活、占用内存小以及基于非阻塞和IO复用的事件驱动模型等特性，成为了目前最流行的代理服务器[8]，经常使用的场景是作为反向代理服务器，将前端请求通过负载均衡策略分发给后台服务器进行处理，然后将结果返回。以下结合网上购书系统，配置Nginx服务器，使其能够代理后台服务器，同时解决多台服务器之间Session共享的问题。

### 4.4.1 Nginx配置

以下内容展示的是Nginx服务器的核心配置，首先配置的是处理前端请求的进程数，默认与当前计算机的CPU核心相等，相对效果会更好；接下来时就是配置Nginx的事件模块，该模块的配置决定着系统并发处理能力的上限，其中效果最好的是Linux内核2.6以上的版本支持的epoll事件驱动模型，通过多路复用的方式，加快了请求的分发效率；使用upstream关键字配置的需要代理的后台服务器信息，包括被代理主机名及端口号；最后配置的是代理服务器的访问主机名和端口号，并通过正则表达式的方式匹配URL以减少路径的配置[9]。根据以下所示的配置信息后，当接收到一个前端请求<http://localhost:80/books>时，Nginx服务器根据负载均衡策略，将该请求分发给目前相对空闲的后台服务器，比如后台服务器1，那么真正请求的URL就转化为<http://127.0.0.1:8080/books>。

|  |
| --- |
| #配置允许Nginx进程生成的worker process数，一般与CPU核数相等  worker\_processes auto;  events {  #时间驱动模型，有select、poll、epoll等，epoll只支持linux2.6以上系统，效率最好  use epoll  #最大连接数，直接决定并发处理能力  worker\_connections 2048;  }  #配置上游服务器  upstream backend {  server 127.0.0.1:8080; #后台服务器1  server 127.0.0.1:9090; #后台服务器2  server 127.0.0.1:9999; #后台服务器3  }  server {  #监听端口  listen 80;  server\_name localhost;  #通过客户端请求头信息  proxy\_pass\_request\_headers on;  #保留客户端的真实信息  proxy\_set\_header Host $host;  proxy\_set\_header X-Real\_IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  #代理图书访问接口  location ~ /books/ {  proxy\_pass http://backend;  }  location ~ /book/ {  proxy\_pass http://backend;  }  location ~ /shop/ {  proxy\_pass http://backend;  } |

### 4.4.2 共享Session

默认情况下，Session都是由Web服务器自身管理的，想要实现多台服务器共享Session，一个解决方案是将Session交由第三方应用存储管理，并要求各服务器能够方便地第三方应用进行交互。由于网上购书系统后台使用了SpringBoot作为开发框架，而且使用了Redis作为数据缓存，那么通过整合Spring旗下的Spring-Session模块，添加以下的配置信息即可实现多台后端服务器共享Session的问题。其中的实现原理是将Session交由Redis管理，当服务器需要使用Session时，不再是从Tomcat容器中获取，而是直接从Redis内存数据库中获取，保证了同一用户仅对应一个Session示例。

|  |
| --- |
| # spring-session config  spring.session.store-type=redis #将Session托管给Redis管理  spring.session.redis.flush-mode=on-save #执行保存操作时刷新Session |

# 5 系统测试

## 5.1 测试目的及任务

系统实现完成之后，为了保证其正确性及健壮性，需要在部署到生产环境前对其进行程序测试。测试的目的在于发现程序的错误，通过在计算机上执行程序，暴露程序中的错误[10]。测试的任务除了保证系统功能的正确实现，还包括容错性的判断以及健壮性的检验。

## 5.2 测试计划

在执行测试之前，一个明确的测试计划不仅能够提升程序测试的速度，也避免了测试过程中一些人为因素的干扰，使得测试出来的数据更具有代表性，这样，才能更好的体现测试数据的分析结果的价值所在。本小节主要对系统的核心模块从功能和性能两个方面进行测试，功能上通过有效等价类的方法设计测试用例，性能上展示不同数量的用户并发访问同一个接口时的响应时间的差距。

### 5.2.1 功能测试

本小节主要测试系统核心的功能是否能够正常运行。

（1）测试用例

表9至表11是基于等价类划分法和边界值分析设计的测试用例，包括用户登录、查询图书和购买图书。等价类涵盖了所有可能的输入情况，能够测试出功能的可靠性，边界值分析作为等价类划分的补充，通过对等价类的边界赋值测试，可以测试出系统更多的错误。

表9 用户登录测试用例

|  |  |  |
| --- | --- | --- |
| 用例编号 | 输入数据 | 预期结果 |
| 1 | 输入正确用户名和密码 | 登录成功 |
| 2 | 什么都不输 | 提示：请输入登录信息 |
| 3 | 只输入用户名 | 提示：请输入密码 |
| 4 | 只输入密码 | 提示：请输入用户名 |
| 5 | 输入错误信息 | 提示：用户名和密码不一致 |

表10 查询图书测试用例

|  |  |  |
| --- | --- | --- |
| 用例编号 | 输入数据 | 预期结果 |
| 1 | 什么都不输 | 显示所有图书 |
| 2 | 输入书名“java” | 只显示书名包含“java”的图书 |
| 3 | 选择图书类别为“计算机” | 只显示类别为“计算机”的图书 |
| 4 | 输入书名“java”、选择类别“计算机” | 只显示书名包含“java”、类别为“计算机”的图书 |

表11 购买图书测试用例（此处假设库存量为5）

|  |  |  |
| --- | --- | --- |
| 用例编号 | 输入数据 | 预期结果 |
| 1 | 4 | 购买成功，生成订单 |
| 2 | -1 | 提示：购买数量不能为负数 |
| 3 | 6 | 提示：库存量不足 |
| 4 | 4.5 | 提示：请输入整数 |

（2）测试结果及分析

应用上述测试用例对系统进行测试，实际测试结果与预期结果基本一致，这说明了系统的功能基本能够正常执行，操作方便，并且具备一定的容错能力。

### 5.2.2 性能测试

（1）测试用例

表12至15描述了对系统核心功能进行压力测试的测试用例，其中输入数据表示同一时间并发访问的用户数，输出的是90%的用户响应时间，单位为毫秒。

表12 优化前获取图书信息测试用例

|  |  |  |
| --- | --- | --- |
| 用例编号 | 输入数据 | 输出 |
| 1 | 100 | 1528 |
| 2 | 200 | 2007 |
| 3 | 300 | 2142 |
| 4 | 400 | 4214 |
| 5 | 500 | 5837 |

表13 优化后获取图书信息测试用例

|  |  |  |
| --- | --- | --- |
| 用例编号 | 输入数据 | 输出 |
| 1 | 100 | 228 |
| 2 | 200 | 814 |
| 3 | 300 | 1632 |
| 4 | 400 | 2055 |
| 5 | 500 | 3225 |

表14 优化前购买图书测试用例

|  |  |  |
| --- | --- | --- |
| 用例编号 | 输入数据 | 输出 |
| 1 | 100 | 234 |
| 2 | 200 | 934 |
| 3 | 300 | 2114 |
| 4 | 400 | 2803 |
| 5 | 500 | 3706 |

表15 优化后购买图书测试用例

|  |  |  |
| --- | --- | --- |
| 用例编号 | 输入数据 | 输出 |
| 1 | 100 | 96 |
| 2 | 200 | 358 |
| 3 | 300 | 864 |
| 4 | 400 | 1427 |
| 5 | 500 | 1969 |

（2）测试结果及分析

根据测试用例，使用JMeter工具对系统进行压力测试，记录系统优化前后的响应时间对比。图10与图11显示了优化前后不同数量的用户同时获取图书信息时90%的用户响应时间，从图中可以得出随着用户数的增加，系统对用户的响应时间也在逐步增大，通过两幅图展示的数据对比，优化前后对于同一用户数量的访问，优化前的响应时间是优化后的两倍，即优化后用户获取图书信息的等待时间缩短了一半，这很大程度上提高了用户浏览图书的体验。

图12和13显示了优化前后用户购买图书时响应时间的对比，根据两幅图的数据对比，可以明显看出响应时间随着用户数的增加而不断延长，优化前500个用户同时购买图书时，响应时间为3706毫秒，而优化后这一数据降到了1969毫秒，比优化前缩短了差不多一半，这充分说明了优化后系统的性能有了大幅提高，提供的服务更快，用户体验更佳。

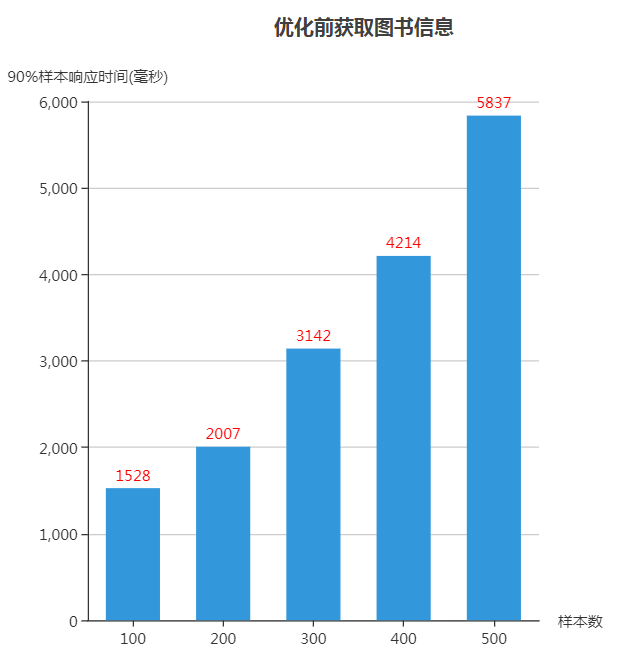


图10 优化前获取图书信息测试数据展示

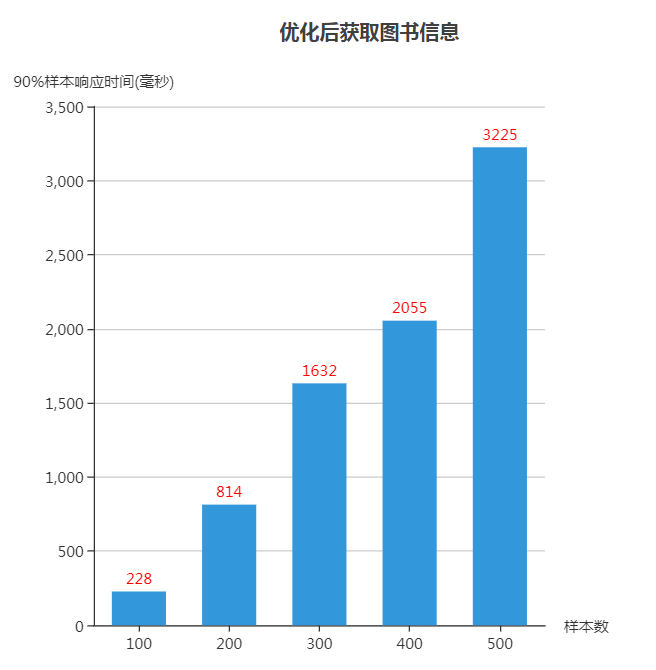


图11 优化后获取图书信息测试数据展示

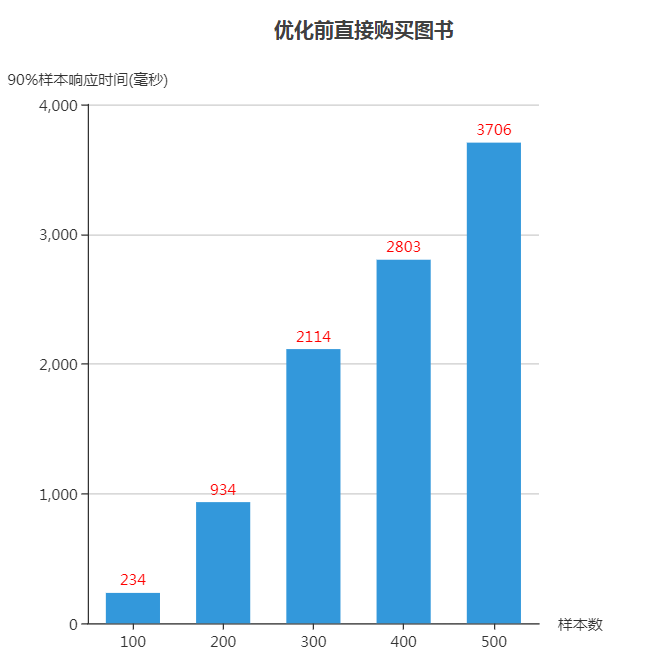


图12 优化前购买图书

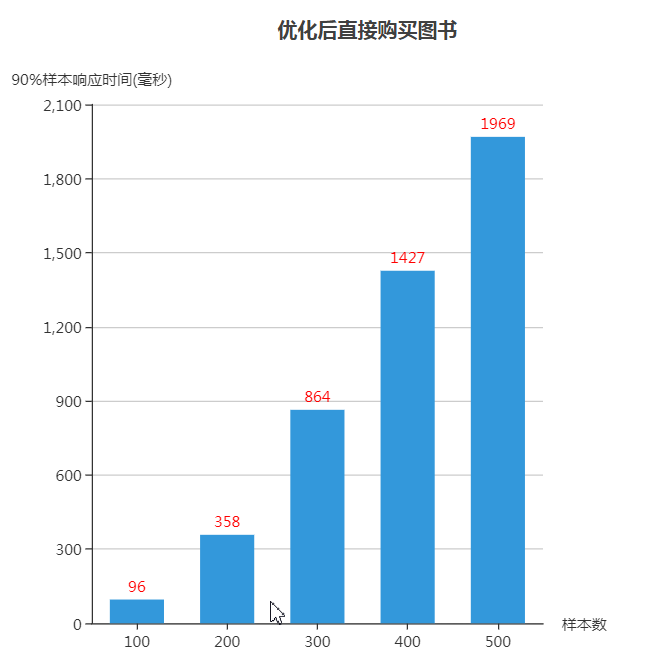


图13 优化后购买图书

# 6 总结与展望

本次毕业设计题目是基于Web系统的性能优化研究，以网上购书系统为例，从数据库的角度，对购买图书功能进行优化，通过定义存储过程，减少了SQL语句执行过程中的语句解析和优化的时间；从应用程序的角度，对图书查询进行优化，通过引入Redis服务器对查询结果进行缓存，减少对数据库的访问，缩短了后续查询的响应时间；从管理的角度，将项目的前后端分离，后台只关注数据的提供，而前端则致力于数据的渲染，分工明确，便于系统的管理；从高并发的角度，将后台应用部署到多台服务器上，前端使用Nginx作为反向代理服务器，利用软负载算法将前端数据请求分发到后台服务器，减轻单个后台服务器的负载压力。为了提高数据缓存的健壮性及容量，对Redis服务器进行了集群，并配置主从节点，主节点宕机后从节点自动升级为主节点。

本次设计中，使我对WEB应用的整个开发流程有了更清晰的认知，从前端发起请求到得到响应结果，途中需要经历很多环节，只有梳理好环节之间的联系以及各个环节的功能以及实现原理，才能更好地针对某些环节进行优化，通过不断累积，能够大大缩短整个流程的响应时间，从而给用户更好的体验。开发过程中的每个阶段都需要认真对待，不能只关注代码而忽略了需求分析的重要性，否则会给系统后期的维护带来很大的难度。

系统优化涉及很多方面，如何将各方面整合成一个更优的整体是很有难度的，只有结合业务特性不断调整，才有可能达到好的效果，稍有不慎，可能导致适得其反的后果。本次的优化研究是比较粗糙的，只是阐述了系统能够优化的方面以及对其中的某些方面进行针对性的优化，真正生产环境中需要结合业务数据，从整个系统的角度思量、分析，包括关系数据库的I/O使用率、后台服务器的CPU占用率以及内存的使用率、网络带宽的消耗等。希望以后针对不同的项目的不同方面，可以更加深入的理解其中业务，针对业务特性使用对应的技术进行优化，使得优化的效果更大化。

参 考 文 献

[1] 谭建平. Web网站系统性能优化研究及其应用 [D]. 重庆：重庆大学软件工程学院，2007.10.

[2] 史济民，顾春华，郑红. 软件工程——原理、方法与应用（第3版） [M]. 北京：高等教育出版社，2009.03.

[3] 石玉强,闫大顺,赵爱芹等. 数据库原理及应用 [M]. 北京: 中国水利水电出版社, 2009.08.

[4] 唐汉明，翟振兴，关宝军等. 深入浅出MySQL：数据库开发、优化与管理维护（第2版） [M]. 北京：人民邮电出版社，2014.01.

[5] 沉默王二. gzip压缩tomcat服务器响应包，大幅提升web性能 [EB/OL]. https://blog.csdn.net/qing\_gee/article/details/51538598, 2016.05.10.

[6] Josiah L. Carlson. Redis in Action [M]. 北京：人民邮电出版社，2015.11.

[7] Craig Walls. Spring Boot in Action [M]. 北京：人民邮电出版社，2016.09.

[8] 苗泽. Nginx高性能Web服务器详解 [M]. 北京：电子工业出版社，2013.10.

[9] 陶辉. 深入理解Nginx：模块开发与架构解析（第2版） [M]. 北京：机械工业出版社，2016.02.

[10] 王顺，潘娅，盛安平等. 软件测试方法与技术实践指南Java EE篇（第3版） [M]. 北京：清华大学出版社，2014.06.

致 谢

大四即逝，是结束也是开始，结束是我的校园青春生活，开始是我的社会工作生涯。这四年，有苦有辣，有酸有甜，苦是军训踢正步一次不行再来一次，辣是因修路而停水停电依然不忘图书馆学习，酸是毕业实习舍友各奔东西无聊一人开黑，甜是有你，有空帮我拿下快递。在这即将离去的日子里，我要感谢我的舍友给了我四年美好时光，要感谢班级组织了多次难忘的团日活动，更要感谢老师和学校教给我许多受用的知识与做人的道理。本次设计，凝结了我四年来所学的知识，其中少不了同学的建议与指点，更少不了导师的安排与指导，可以说，没有你们本设计将无法顺利完成。你们的帮助，使我能以此文，为我大学生活画上一个完美的句号，而我，将以此文为旗，定于人生时间轴，左为难忘校园青春日常，右是憧憬社会阅历生活，以此文作为我迈向社会的序章。最后，由衷感谢这大学四年对我有授业之恩的老师与和我有共同求索之志的同学。

**仲恺农业工程学院**

# 本科毕业论文(设计)成绩评定表

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓 名 | | 黄华冬 | | | | 学 号 | | | 201420224330 | | | |
| 学 院 | | 信息科学与技术学院 | | | | 专业、班级 | | | 计算机科学与技术143班 | | | |
| 毕业论文(设计)题目 | | | 基于Web系统的性能优化研究  ─── 以网上购书系统为例 | | | | | | | | | |
| 指导教师姓名、职称 | | | 赵爱芹、讲师 | | | | | | | | | |
| 指导教师评语及评分：  该学生毕业设计选择的题目为《基于Web系统的性能优化研究》，并以网上购书系统为例进行了编程设计进而进行性能优化。系统的优化包括在数据库中创建索引以及存储过程、后台使用Redis进行缓存、前台使用负载均衡策略等几个方面，经测试，可以大大提高系统的性能，实用性较强。不足之处是测试时由于受硬件的限制，无法完全展现优化的效果。  在毕业设计过程中，该同学按照学校要求，在指导老师的指导下完成整个毕业设计工作和论文的撰写。设计的系统性能良好，优化效果明显。论文格式规范，符合规划化要求。在整个毕业设计过程中，态度端正，跟老师积极交互，时间安排合理，能按时完成各阶段任务。整个设计工作量充足，满足本科生毕业设计的要求。  给予成绩为：95分。  签 名：  年 月 日 | | | | | | | | | | | | |
| 评阅老师评语及评分：  该同学的毕业设计论文选题符合专业培养目标，实用性较强，论文题目与论文的内容完全相符；其设计实现了系统的应有的功能，软件在优化之后可以正常运行；论文从基本的系统实现到优化实现，层次清晰，结构严谨，逻辑性强，语言表达流畅，完全符合规范化要求。  不足之处是性能测试不能完全展现出来。  评分：94分  签 名:  年 月 日 | | | | | | | | | | | | |
| 答辩记录：  1、哪种优化策略最为有效？  答：基于Nginx的负载均衡优化。Nginx作为反向代理服务器，接收前端用户请求，根据负载均衡策略将请求分发到不同的后端服务器，由之前的一台服务器提供服务到多台服务器提供服务，因此，优化策略最为明显的是基于Nginx的负载均衡。  2、使用数据库索引的禁忌是什么？  答：由于索引需要占据内存空间，数据量较大的列不适宜做索引，经常发生变动的数据列也不适于建索引，不利于索引的维护。  3、压力测试下书的数量是多少？  答：由于时间紧凑，没有做数据库索引的测试，因为只有数据量达到上万级别的时候才能展示索引的优势，因为时间原因，没能录入这么多的数据。  4、优化方法是什么，何为优化前后？  答：分析Web请求流程，结合实际的业务，找出可优化的点，比如缓存系统的热数据；通过加机器提高系统的并发能力。优化前的系统架构包括展示层、控制层、服务层和数据存储层，优化后在系统原有架构上添加缓存层缓存热数据，增加负载均衡层加大系统整体的并发能力。  答辩秘书：  年 月 日 | | | | | | | | | | | | |
| 答辩小组评价意见及评分：  该生论文选题新颖，符合专业毕业要求，且有自己独立的见解与研究，实用性较强。论文结构严谨，逻辑性强，层次清晰，格式符合规范化要求。答辩过程中，语言表达能力较好，能简明扼要的阐述论文的主要内容。对教师提出的问题，思路清晰，能准确流利的回答问题。  经答辩小组集体评议，该生论文答辩得分：93分。  答辩组长：  年 月 日 | | | | | | | | | | | | |
| 论文(设计)成绩 | 指导老师（40%） | | | | 评阅老师（20%） | | | 答辩小组（40%） | | | 总评分 | 等级 |
| 原始分 | | | 折算分 | 原始分 | | 折算分 | 原始分 | | 折算分 |
| 95 | | | 38 | 94 | | 18.8 | 93 | | 37.2 | 94 | 优秀 |
| 学院答辩领导小组(委员会)审核意见 | 签章：  年 月 日 | | | | | | | | | | | |

注：1、论文(设计)成绩一栏中，折算分分别由指导老师（40%）、评阅老师（20%）、答辩小组（40%）给出的原始分乘以各自百分比例所得。总评分由折算分相加所得。

2、“等级”：90分以上为“优秀”、80―89分为“良好”、70―79分为“中等”、60―69分为“及格”、59分以下为“不及格”。