

**本科毕业论文（设计）**

**论文（设计）题目：面向高并发访问优化的内容管理系统**

**学 院： 示范性软件学院**

**专 业： 软件工程**

**班 级： 软工154**

**学 号： 1500170404**

**学生姓名： 杨 毅**

**校内导师： 李 晖**

**校外导师： 廖黎达**

2019 年 6 月 3日

**贵州大学本科毕业论文（设计）**

**诚信责任书**

本人郑重声明：本人所呈交的毕业论文（设计），是在导师的指导下独立进行研究所完成。毕业论文（设计）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。

特此声明。

论文（设计）作者签名：

日 期：

目录

[摘要 V](#_Toc11158849)

[Abstract VI](#_Toc11158850)

[第一章 绪论 1](#_Toc11158851)

[1.1 研究背景与意义 1](#_Toc11158852)

[1.2 研究内容 1](#_Toc11158853)

[1.3 研究所需技术与知识 2](#_Toc11158854)

[1.4 论文结构 4](#_Toc11158855)

[第二章 需求分析 5](#_Toc11158856)

[2.1 角色划分 5](#_Toc11158857)

[2.1.1 游客 5](#_Toc11158858)

[2.1.2 普通用户 6](#_Toc11158859)

[2.1.3 普通管理员 6](#_Toc11158860)

[2.1.4 超级管理员 7](#_Toc11158861)

[2.2 功能性需求分析 8](#_Toc11158862)

[2.2.1 用户功能需求 8](#_Toc11158863)

[2.2.2 系统功能需求 8](#_Toc11158864)

[2.3 非功能性需求分析 10](#_Toc11158865)

[第三章 内容管理系统设计与实现 12](#_Toc11158866)

[3.1 系统设计 12](#_Toc11158867)

[3.1.1 系统整体结构 12](#_Toc11158868)

[3.1.2 系统用例图 13](#_Toc11158869)

[3.1.3 系统工作流程 14](#_Toc11158870)

[3.1.4 各模块数据库设计 16](#_Toc11158871)

[3.2 实现技术 19](#_Toc11158872)

[3.3 系统展示 20](#_Toc11158873)

[3.3.1 门户网站 20](#_Toc11158874)

[3.3.2 系统后台 21](#_Toc11158875)

[第四章 系统架构及并发性能优化 28](#_Toc11158876)

[4.1 初始系统简介 28](#_Toc11158877)

[4.1.1 系统架构 28](#_Toc11158878)

[4.1.2 系统技术实现 30](#_Toc11158879)

[4.1.3 被测接口数据库设计 30](#_Toc11158880)

[4.2 高并发压测原理及方法 33](#_Toc11158881)

[4.2.1 相关工具介绍 33](#_Toc11158882)

[4.2.2 JMeter并发测试理论 34](#_Toc11158883)

[4.2.3 并发测试方法及步骤 34](#_Toc11158884)

[4.2.4 测试期间需要记录的监控数据 36](#_Toc11158885)

[4.3 初始系统性能测试 38](#_Toc11158886)

[4.3.1 测试信息 38](#_Toc11158887)

[4.3.2 测试计划 40](#_Toc11158888)

[4.3.3 测试结果 40](#_Toc11158889)

[4.3.4 性能分析 43](#_Toc11158890)

[4.3.5 测试总结 46](#_Toc11158891)

[4.4 数据库与JVM优化 46](#_Toc11158892)

[4.4.1 优化背景 46](#_Toc11158893)

[4.4.2 优化方案 46](#_Toc11158894)

[4.4.3 优化后的系统变化 47](#_Toc11158895)

[4.4.4 优化后接口性能测试 51](#_Toc11158896)

[4.4.5 优化前后接口性能对比 58](#_Toc11158897)

[4.5 添加缓存及优化 58](#_Toc11158898)

[4.5.1 优化背景 58](#_Toc11158899)

[4.5.2 优化方案 58](#_Toc11158900)

[4.5.3 优化后的系统变化 59](#_Toc11158901)

[4.5.4 优化后接口性能测试 61](#_Toc11158902)

[4.5.5 优化前后接口性能对比 67](#_Toc11158903)

[4.6 容器优化与带宽升级 67](#_Toc11158904)

[4.6.1 优化背景 67](#_Toc11158905)

[4.6.2 优化方案 67](#_Toc11158906)

[4.6.3 优化后的系统变化 68](#_Toc11158907)

[4.6.4 优化后接口压力测试 69](#_Toc11158908)

[4.6.5 优化前后接口性能对比 73](#_Toc11158909)

[4.7 系统集群优化 74](#_Toc11158910)

[4.7.1 优化背景 74](#_Toc11158911)

[4.7.2 优化方案 74](#_Toc11158912)

[4.7.3 优化后的系统变化 78](#_Toc11158913)

[4.7.4 优化后接口压力测试 80](#_Toc11158914)

[4.7.5 优化前后接口性能对比 85](#_Toc11158915)

[4.8 优化总结 86](#_Toc11158916)

[第五章 总结与展望 88](#_Toc11158917)

[5.1 总结 88](#_Toc11158918)

[5.2 展望 89](#_Toc11158919)

[参考文献 91](#_Toc11158920)

[致谢 93](#_Toc11158921)

面向高并发访问优化的内容管理系统

# 摘要

面向高并发访问优化的内容管理系统的核心是“高并发访问优化”，本文主要是体现高并发访问优化的处理过程。近年来由于网络发展飞速，网民数量剧增，使得各大媒体互联网公司发展迅速，获益良多。但大量用户在给这些互联网公司带来收益的同时，也给这些公司的系统带来了并发访问压力，用户量越大系统的访问压力就越大，对系统的并发读取性能要求也就越高，若想保证系统正常运行，就需要在并发环境下对系统进行并发访问优化。

内容管理系统主要用来对各种内容进行发布和管理，系统包含前端信息展示页面和后台管理系统，前端主要用以展示系统所管理的内容信息，并提供用户登录、注册、留言等功能。而后台管理系统则是对前端所展示的信息进行管理，提供系统管理、用户管理、模糊查询、权限动态控制等功能。本文主要对该系统的新闻列表接口进行并发访问测试、分析与优化，以简单MVC三层架构的初始内容管理系统为基础，通过对系统的新闻列表接口进行多次并发访问测试、分析与优化后，使得系统最终能满足高并发访问的性能需求。

关键词：系统，高并发，测试，性能优化

Content management system for high concurrent access optimization

# Abstract

The core of the content management system for high concurrent access optimization is “high concurrent access optimization”. This paper mainly reflects the processing of high concurrent access optimization. In recent years, due to the rapid development of the network and the rapid increase in the number of Internet users, major media Internet companies have developed rapidly and benefited a lot. However, while a large number of users bring benefits to these Internet companies, they also bring concurrent access pressure to these companies' systems. The greater the number of users, the greater the access pressure of the system, and the more concurrent the performance requirements for the system. High, if you want to ensure the normal operation of the system, you need to optimize the concurrent access to the system in a concurrent environment.

The content management system is mainly used to publish and manage various content. The system includes a front-end information display page and a background management system. The front-end is mainly used to display content information managed by the system, and provides functions such as user login, registration, and message. The background management system manages the information displayed by the front end, and provides functions such as system management, user management, fuzzy query, and dynamic control of permissions. This paper mainly conducts concurrent access testing, analysis and optimization of the news list interface of the system. Based on the initial content management system of the simple MVC three-tier architecture, it conducts multiple concurrent access tests, analysis and optimization on the news list interface of the system. So that the system can finally meet the performance requirements of high concurrent access.

Keywords: system, high concurrency, testing, performance optimization

# 第一章 绪论

## 1.1 研究背景与意义

近年来网络发展飞速，网民数量剧增，截至2018年12月，我国网民规模已达8.29亿，2018年全年新增网民5653万[1]，大规模网民的涌入致使各种新闻网站、视频平台、新媒体平台等以内容输出为主体的媒体互联网公司不断冒出，这些公司以发布文章、图片、视频等各种各样的内容来吸引用户，然后从用户与广告中获取收益。不过因为文章、图片、视频等内容又多又杂，为了方便内容的发布与管理，一个能与之对应的内容管理系统就显得十分重要。

虽然大量用户的涌入能为这些媒体互联网公司带来大量收益，但也会给这些公司的服务系统带来巨大的访问压力，用户量越大，系统的并发访问压力就越大，对系统的并发性能要求也就越高，若系统性能无法满足，系统就会出现卡顿或是无法正常响应的现象，从而影响用户体验，致使用户流失，因为系统的特性和功能虽然是重要的，但用户体验对于客户的忠诚度却有着更大的影响[2]。而且当并发量过大时，还可能会导致系统直接崩溃，给公司带来巨大损失。所以，如何在并发环境下提高系统的性能、增强系统的稳定性与可靠性，对于这些互联网公司来说也非常重要。

## 1.2 研究内容

本文的研究目标是开发一个能支撑高并发访问的内容管理系统，主要用来管理新闻信息。系统由系统管理、信息管理和用户管理三大模块组成，不仅能为新闻内容的发布、展示和管理提供便捷，还能保证系统本身的安全性、可靠性和灵活性，最重要的是系统所管理的内容能支持多用户并发访问。

本文首先对内容管理系统的功能性需求和非功能性需求进行分析，然后依据功能需求对系统进行功能设计与实现，最后以非功能性需求中的高性能需求为目标，通过模拟多个用户并发浏览网站新闻的方式，对已实现系统的新闻列表接口进行多轮并发访问测试与优化，直至内容管理系统能满足高性能需求的目标为止。

## 1.3 研究所需技术与知识

（1）相关技术

开发语言：Java8

Servlet容器：Tomcat、Undertow

前端：Html/CSS、JavaScript、JQuery、Bootstrap、Layui、WangEditor等

后端：Spring、SpringBoot、SpringMVC、SpringSecurity、SpringSession、SpringCache、Mybatis、Kaptcha、Lombok、Log4j2、Druid等

前后端交互：Ajax、Json

负载均衡：Nginx

虚拟容器：Docker

缓存：Ehcache、Redis

数据库：Mysql

（2）相关工具

开发工具：IDEA

监控工具：SpringBootAdmin、Druid数据源

测试工具：JMeter

项目结构控制：Maven

项目版本控制：Git/Github

设计与画图：processon、PowerDesigner、Navicat

（3）相关知识

数据库优化相关知识

JVM优化相关知识

JMeter并发测试相关知识

Linux服务部署相关知识

（4）主要技术与工具简介

1. SpringBoot

SpringBoot是一个简化Spring应用开发的开源框架，具有可创建独立的Spring应用程序、直接嵌入式Servlet容器（不需要部署WAR文件）、自动配置、不需要代码生成，也不需要XML配置等特点[3]。

1. SpringSecurity

SpringSecurity是一个功能强大且高度可定制的身份验证和访问控制开源框架，具有全面和可扩展的身份验证和授权支持，可防止攻击，如会话固定，点击劫持，跨站请求伪造等，还可与Servlet API或SpringMVC集成[4]。

1. Mybatis

MyBatis 是一款优秀的持久层开源框架，具有支持定制化 SQL、存储过程以及高级映射，且免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集等特点[5]。

1. Nginx

Nginx [engine x]是一个高性能、开源的HTTP和反向代理服务器，还是一个邮件代理服务器和一个通用的TCP/UDP代理服务器，同时也提供了IMAP/POP3/SMTP服务[6]。

1. Redis

Redis 是一个开源、高性能的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件，并支持多种类型的数据结构[7]。

1. JMeter

Apache JMeter是一个开源的、100%纯Java的应用程序，用于加载测试功能行为和度量性能，可用于测试静态和动态资源、Web动态应用程序的性能，也可以用来模拟服务器、服务器组、网络或对象上的重载，以测试其强度或分析不同负载类型下的总体性能[8]。

## 1.4 论文结构

第一章绪论，主要介绍本论文的研究背景、研究意义和研究内容，并简要介绍了本次研究所需的相关技术与知识。

第二章需求分析，主要介绍内容管理系统的功能性需求分析与非功能性需求分析，重点分析系统各个模块的功能需求，之后再从可靠性、安全性、高性能等方面简要分析系统的非功能性需求，以规定系统应实现的功能与应满足的性能。

第三章内容管理系统的设计与实现，主要介绍系统相关功能模块的设计与实现技术，并对已实现的系统进行截图展示和简要介绍。

第四章系统架构及并发性能优化，此章节本文的重点，主要介绍在对功能已完善的初始系统的新闻列表接口进行并发访问测试与优化，使其满足高并发访问性能需求的整体过程。重点介绍测试过程中，对测试数据的记录与分析，以及对接口性能优化方案的制定、实施和优化前后的性能数据对比等内容。

第五章总结与展望，主要是对本次论文研究的所有工作内容进行总结，介绍本次研究的相关成果，并指明研究过程中还存在的不足之处，最后展望未来的研究内容与研究方向。

# 第二章 需求分析

作为一个面向高并发访问的内容管理系统，需要能对系统内容进行管理并保证系统自身安全，且系统管理的内容能支持多用户并发访问。为了保证系统的安全和方便系统的管理，本章首先对管理系统的角色进行一个大体的划分，确定相关角色的职责与访问权限，然后再去详细分析内容管理系统应当要满足的功能模块，最后再分析面向高并发时，系统应当满足的非功能性需求。

## 2.1 角色划分

为了保证系统的安全、方便系统的管理和提高系统管理的灵活性，本文只对系统做了四个主要角色及其权限的划分：游客、普通用户、普通管理员、超级管理员，其他角色和权限就放给系统的使用者，也就是超级管理员去做决定，如此系统的管理将更为灵活。

### 2.1.1 游客

游客只能浏览网站前端发布的内容信息，可以进行注册和登录操作，不能进行留言，也不能进入后台管理系统，具体权限如表2.1.1所示。

表2.1.1 游客权限

|  |  |
| --- | --- |
| 操作 | 权限 |
| 首页内容访问 | √ |
| 注册 | √ |
| 登录 | √ |
| 发布留言 | × |
| 个人信息管理 | × |
| 进入后台管理系统 | × |
| 系统管理 | × |

表2.1.1（续）

|  |  |
| --- | --- |
| 信息管理 | × |
| 用户管理 | × |

### 2.1.2 普通用户

普通用户即为已登录的用户，可以浏览网站前端发布的内容，可以留言，可以管理自己的个人信息。用户登录后便不会再看到注册与登录的入口，理论上不能再注册与登录，具体权限如表2.1.2所示。

表2.1.2 普通用户权限

|  |  |
| --- | --- |
| 操作 | 权限 |
| 首页内容访问 | √ |
| 注册 | × |
| 登录 | × |
| 发布留言 | √ |
| 个人信息管理 | √ |
| 进入后台管理系统 | × |
| 系统管理 | × |
| 信息管理 | × |
| 用户管理 | × |

### 2.1.3 普通管理员

普通管理员拥有内容管理系统的管理员权限，可以进入后台管理系统，但具体可操作权限需要通过超级管理员进行分配，具体权限如表2.1.3所示。

表2.1.3 普通管理员权限

|  |  |
| --- | --- |
| 操作 | 权限 |
| 首页内容访问 | √ |
| 注册 | × |
| 登录 | × |
| 发布留言 | √ |
| 个人信息管理 | √ |
| 进入后台管理系统 | √ |
| 系统管理 | 根据授权 |
| 信息管理 | 根据授权 |
| 用户管理 | 根据授权 |

### 2.1.4 超级管理员

超级管理员拥有内容管理系统的所有权限，包括添加普通管理员和为普通管理员授权等，具体权限如表2.1.4所示。

表2.1.4 超级管理员权限

|  |  |
| --- | --- |
| 操作 | 权限 |
| 首页内容访问 | √ |
| 注册 | × |
| 登录 | × |
| 发布留言 | √ |
| 个人信息管理 | √ |
| 进入后台管理系统 | √ |
| 系统管理 | √ |
| 信息管理 | √ |
| 用户管理 | √ |

## 2.2 功能性需求分析

功能需求分析即分析系统需要实现哪些功能，本节将根据用户功能需求与系统功能需求对内容管理系统进行具体划分，分析和介绍各需求应当满足的功能模块。

### 2.2.1 用户功能需求

对于用户来说，基本需求便是能够浏览内容管理系统发布在网站前端的内容信息，额外需求是可以注册、登录或发布留言等，所以只需要一个用户模块便可。

（1）用户模块

用以为系统用户提供相关服务，主要包含用户注册、用户登录、个人信息管理、用户留言四个子模块。

1. 用户注册

为游客提供注册功能，注册后的成为正式用户。

1. 用户登录

为用户提供登录功能，登录后的用户可以发表留言等。

1. 用户个人信息管理

为用户提供个人信息管理功能。

1. 用户留言

为用户提供留言功能，但用户的留言需要经过后台审核后才对其他用户可见。

### 2.2.2 系统功能需求

内容管理系统是用来管理内容信息的，所以需要有能对内容信息进行管理的信息管理模块，其次为了方便系统本身的管理，需要有个系统管理模块，最后还需要一个可以对系统和网站用户进行管理的用户管理模块，所以系统功能需求主要分为信息管理、系统管理、用户管理三大模块。

（1）内容信息管理模块

用以对前端页面展示的内容信息进行动态管理，主要包含首页内容信息管理、用户留言管理两个子模块：

1. 首页内容信息管理

即对网站首页所展示的相关内容信息进行管理，可对前端页面展示的内容信息进行增、删、改、查操作。

1. 用户留言管理

即对用户在网站前端的留言信息进行管理，可对用户留言信息进行审查和删除操作，用户发布的留言信息需要经过后台审核才能对其他用户可见。

（2）系统管理模块

用以对后台管理系统进行管理，主要包含系统日志、菜单管理、权限管理、角色管理四个子模块：

1. 系统日志

记录管理员在后台系统的操作日志，如重要信息的增删改等，还可对日志进行查看和删除。

1. 菜单管理

管理后台系统的菜单，可对菜单进行增、删、改，也可隐藏指定的菜单，方便后期系统的维护和扩展。

1. 权限管理

管理后台系统每个菜单的权限，具体菜单由菜单管理中所含的菜单信息提供。菜单权限包含增、删、改、查四种，具体对应的是在该菜单里面进行增、删、改、查操作，是后台系统权限动态控制所需要的。权限管理子模块可以对这些菜单的权限进行增删改查操作。

1. 角色管理

管理整个系统的角色，可对角色进行授权和增删改查操作，授权信息由权限管理中所包含的菜单权限信息提供，是后台系统权限动态控制所需要的。

（3）用户管理模块

用以对系统用户进行管理，主要包含普通用户管理、管理员管理、IP封禁管理三个子模块：

1. 普通用户管理

对普通用户进行管理，可查看用户账号信息，可对指定用户进行账号禁用或是登录IP锁定。

1. 管理员管理

超级管理员对普通管理员用户进行管理，包括增、删、改、查、授权、账号禁用和登录IP锁定。

1. IP封禁管理

可添加或删除指定IP，IP出现在此处的用户，将不能登录系统。

## 2.3 非功能性需求分析

一个良好的，且是面向高并发访问的内容管理系统，不能只满足于系统的功能实现，还需要在高并发环境下保证系统的安全性、可靠性、响应速度等。

（1）安全性需求

确保系统本身和系统数据的安全性，只有已被授权的人员才能访问系统后台，和对系统及系统数据进行操作。

（2）可靠性

确保系统在运行期间是可靠的，即保证系统能正常服务用户。

（3）高性能

即使是100名用户并发访问系统，也要保证系统的平均响应时间在2秒以内，且吞吐量在100以上。

# 第三章 内容管理系统设计与实现

本章主要介绍内容管理系统的功能设计与实现，首先根据功能需求分析的内容，对系统功能模块进行相关设计，然后简要介绍实现的系统技术，最后对构建完成的系统进行截图展现。

## 3.1 系统设计

本节的主要任务是根据功能需求分析的内容，对系统各功能模块进行相关设计，确立各模块功能的实现方向。

### 3.1.1 系统整体结构

内容管理系统分为前台门户网站和后台管理系统两个部分，前台门户网站是用来展示系统所管理的内容信息的，而后台管理系统则用来对系统进行管理，具体如图3.1.1所示。

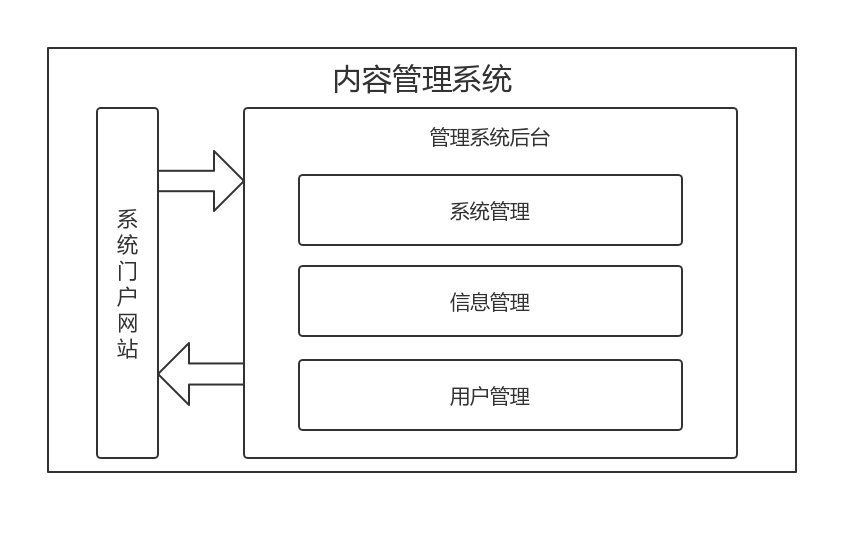


图3.1.1 系统整体结构

### 3.1.2 系统用例图

根据需求分析及系统角色划分，得到内容管理系统的用列图如图3.1.2所示。

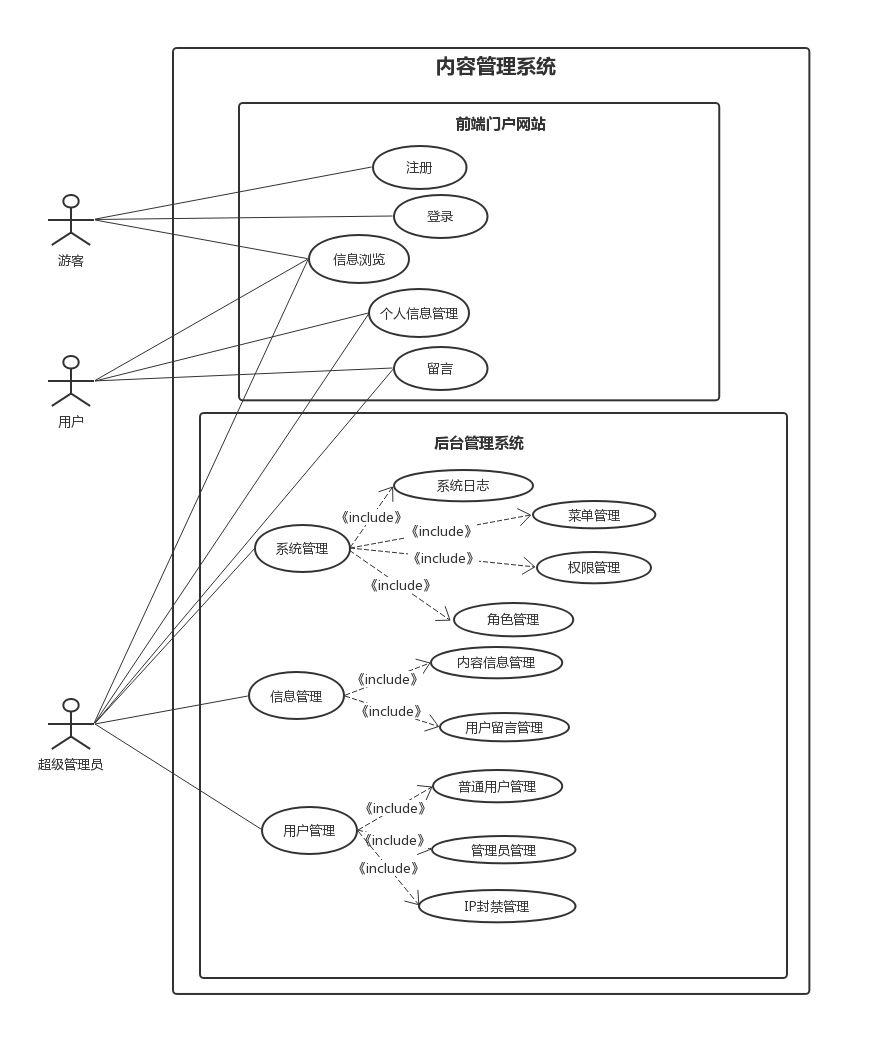


图3.1.2 系统用例图

### 3.1.3 系统工作流程

（1）权限控制流程

权限控制是内容管理系统的核心，只有控制用户在内容管理系统的相关权限，才能保证系统和系统数据的安全。当用户想要进入后台管理系统，或是在后台管理系统进行相关操作时，都需要对其进行授权认证，因需要权限认证的地方较多，此处就以用户请求进入后台管理系统为例，大体展示系统权限认证工作流程，具体如图3.1.3所示。

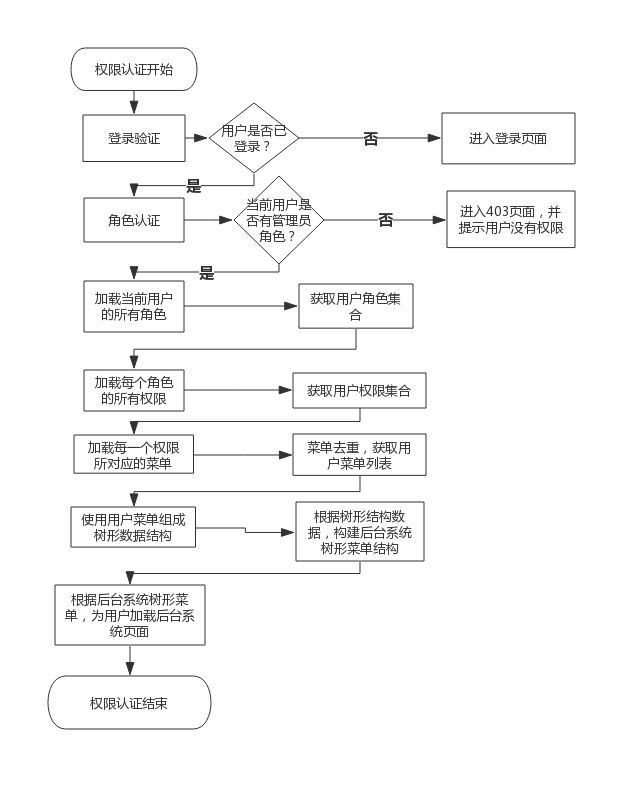


图3.1.3 用户请求进入后台系统权限认证流程

（2）内容信息管理流程

因系统内容管理只是一些简单的增删改查操作，并附加权限控制而已，所有操作流程都相差不大，所以此处就内容信息添加为例，展示系统内容管理的相关工作流程，具体如图3.1.4所示。

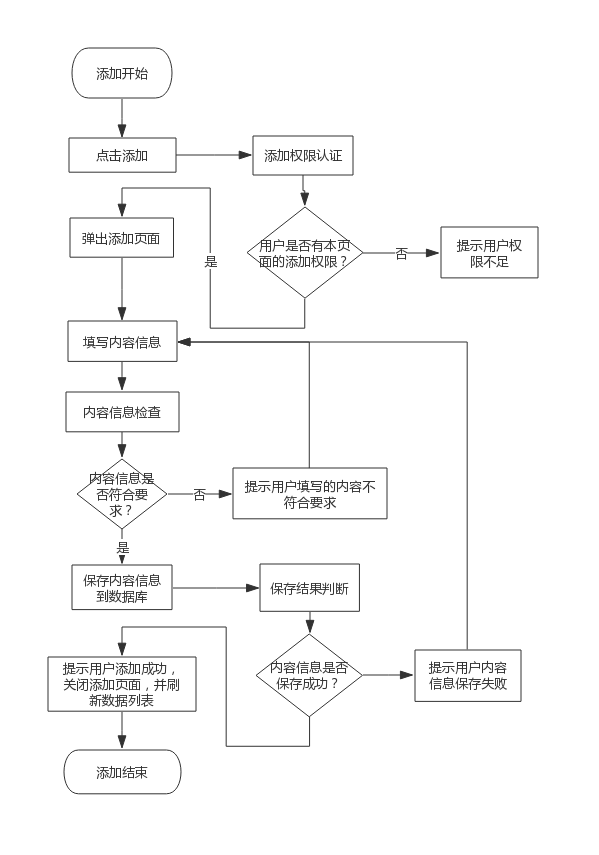


图3.1.4 内容信息添加流程

### 3.1.4 各模块数据库设计

（1）用户模块

1. ER图

用户模块的ER设计图如图3.1.5所示。

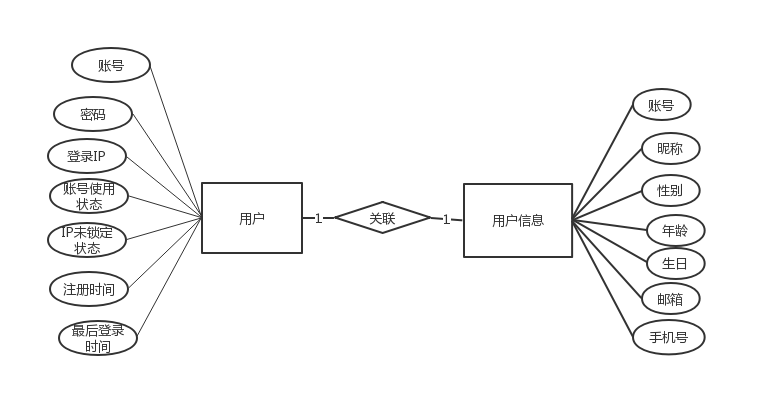


图3.1.5 用户模块ER设计图

1. 数据库关系模型

用户模块的数据库关系模型如图3.1.6所示。

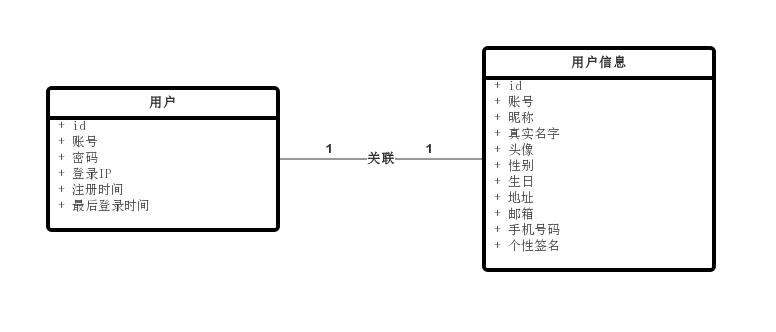


图3.1.6 用户模块关系模型图

（2）系统管理模块

1. ER图

系统管理模块的ER设计图如图3.1.7所示。

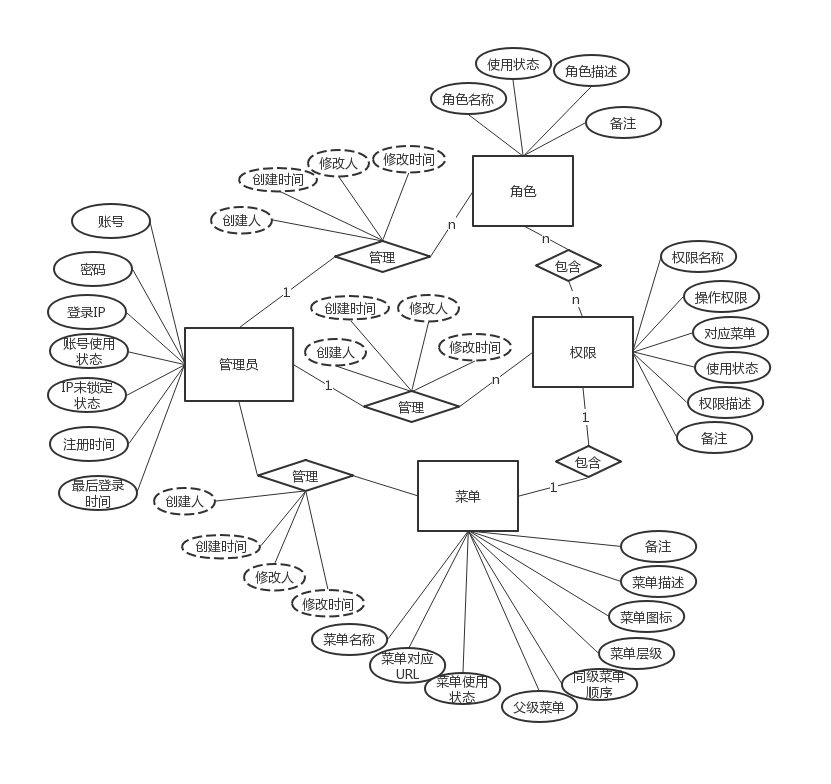


图3.1.7 系统管理模块ER设计图

B、数据库关系模型图

系统管理模块的数据库关系模型如图3.1.8所示。

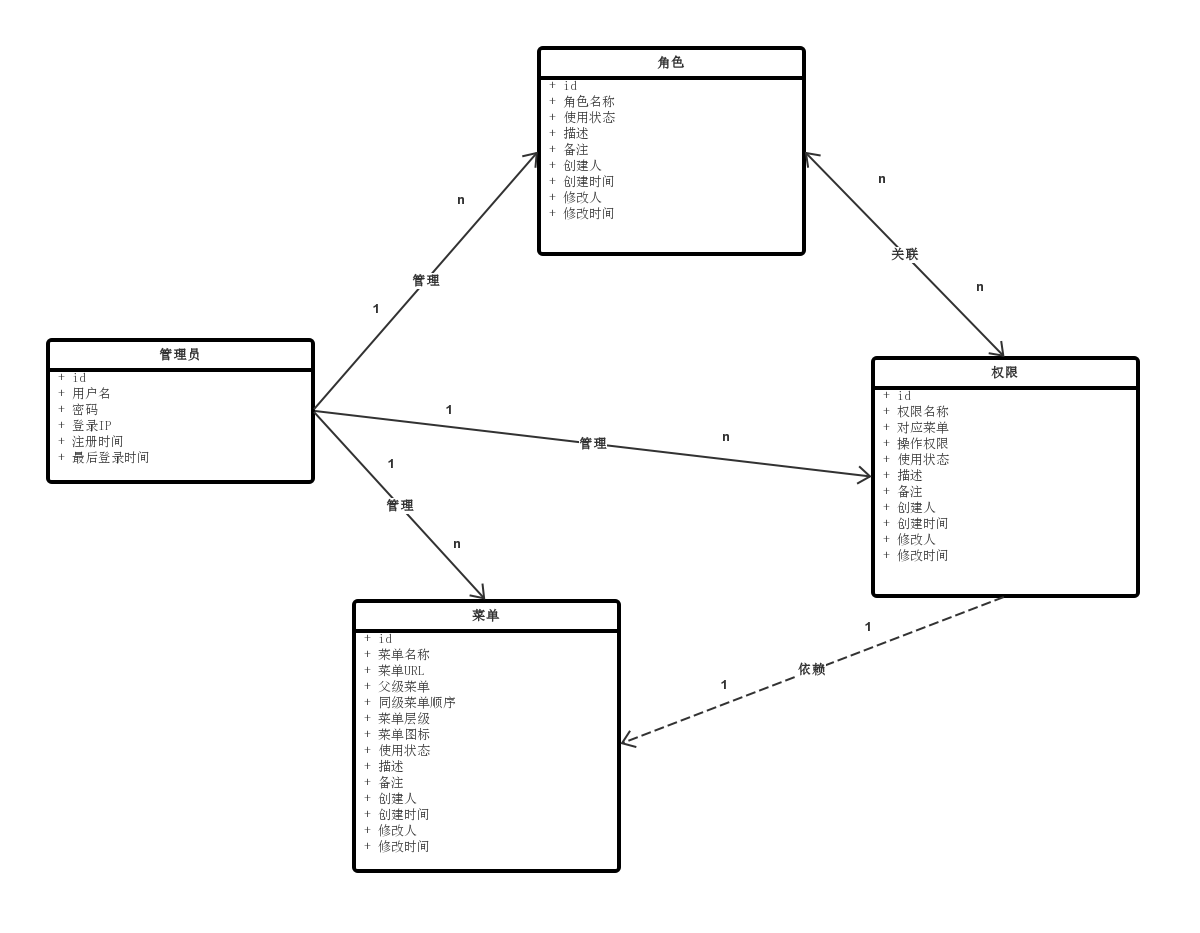


图3.1.8 系统管理模块数据库关系模型图

（3）信息管理模块

1. ER图

信息管理模块的ER设计图如图3.1.9所示。

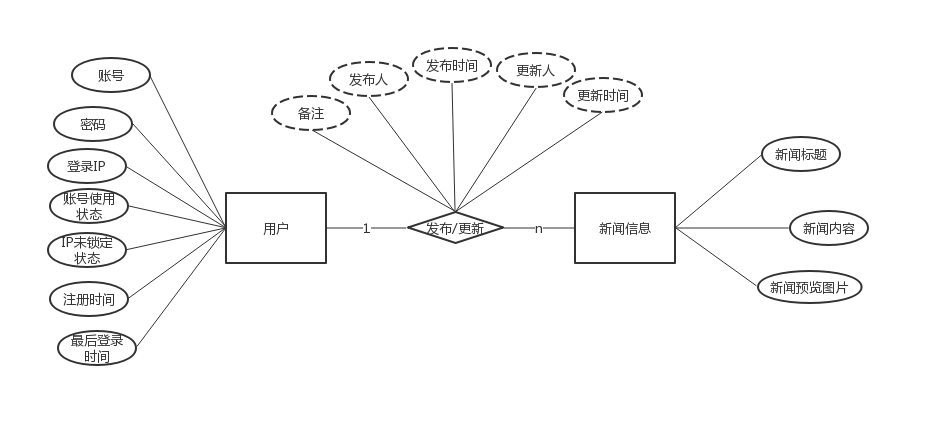


图3.1.9 信息管理模块ER设计图。

1. 数据库关系模型图

信息管理模块的数据库关系模型如图3.1.10所示。

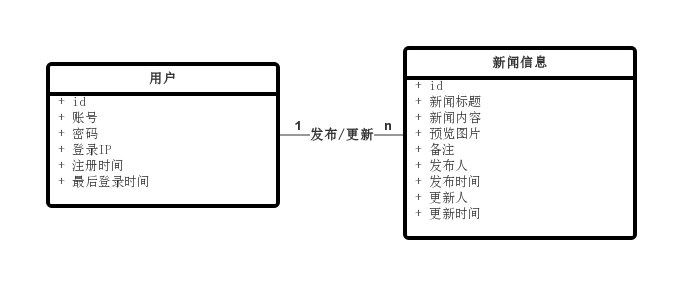


图3.1.10 信息管理模块数据库关系模型图

（4）用户管理模块

与用户模块一致。

## 3.2 实现技术

Servlet容器：Tomcat

权限控制：SpringSecurity

前端：Html/CSS、JavaScript、JQuery、Bootstrap、Layui、WangEditor等。

后端：SpringBoot、SpringMVC、Mybatis、Kaptcha、Lombok等。

前后端交互：Ajax、Json

数据库：Docker-Mysql

项目结构控制：Maven

项目版本控制：Git/Github

开发模式：前后端分离

## 3.3 系统展示

### 3.3.1 门户网站

（1）首页

网站首页即内容管理系统的门户网站，用来给用户展示内容管理系统所管理的相关内容的，用户可在此对相关内容进行浏览，也可进行注册、登录等操作，具体如图3.3.1所示。



图3.3.1 内容管理系统门户网站首页

（2）登录页面

用户登录页面是用来给已注册的用户提供登录入口的，登录后的用户将有权进行更多的操作，如发表留言、维护个人信息等。具体如图3.3.2所示。



图3.3.2 内容管理系统登录页面

### 3.3.2 系统后台

内容管理系统后台是系统的核心，分为系统管理、信息管理、用户管理三大板块，管理员在此可对整个系统的相关内容进行管理，具体如图3.3.3所示。

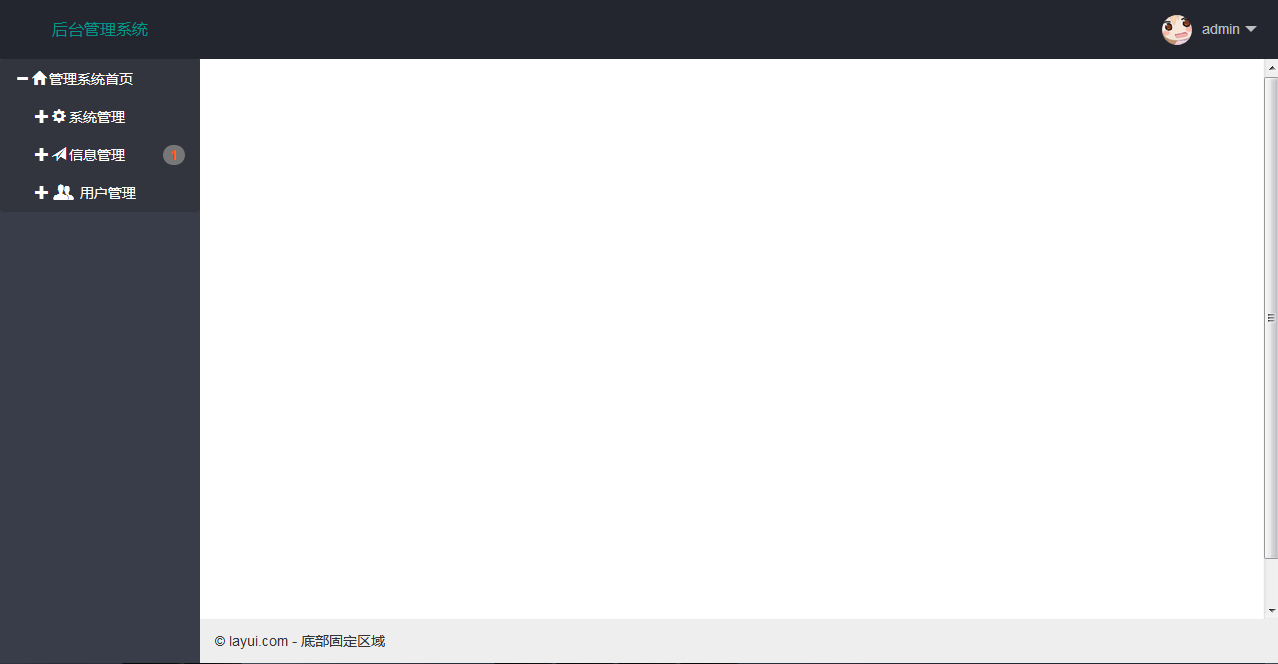


图3.3.3 内容管理系统后台

（1）系统管理

系统管理是用来对系统本身进行管理的，分为系统日志、菜单管理、权限管理、角色管理四个板块，通过这些板块的结合，可对后台系统进行相关操作权限的控制，以保证系统的安全性，对系统来说非常重要。

1. 菜单管理

菜单管理是用来管理后台系统中所有菜单的，结合权限管理与角色管理，使得系统后台可做到对拥有不同权限的用户只展示不同的操作页面，是系统权限控制的重要组成部分，具体如图3.3.4所示。

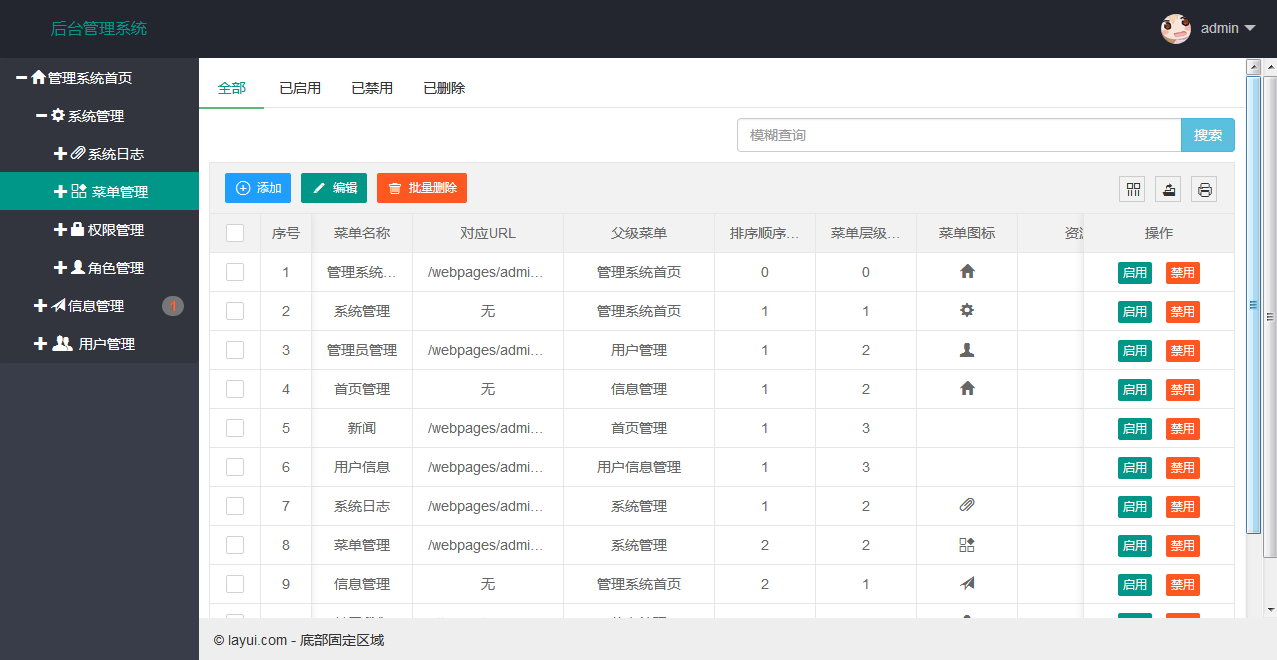


图3.3.4 菜单管理

1. 权限管理

权限管理是用来记录后台系统中的所有菜单的相关权限的，主要是用来为角色授权提供相关菜单的权限选择，是系统权限控制的重要组成部分，具体如图3.3.5所示。

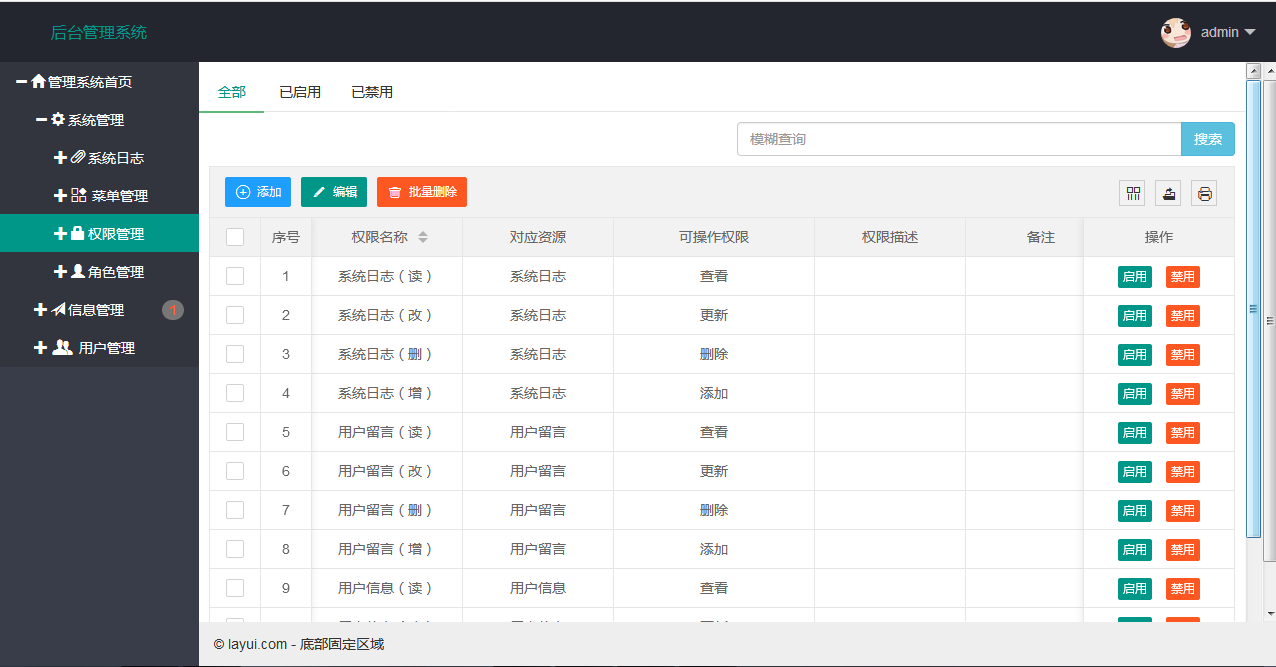


图3.3.5 权限管理

1. 角色管理
   1. 角色管理

角色管理是用来管理系统中的所有角色的，超级管理员可在此处对系统的所有角色进行管理，包括添加角色和为角色授权，是系统权限控制的重要组成部分，具体如图3.3.6所示。

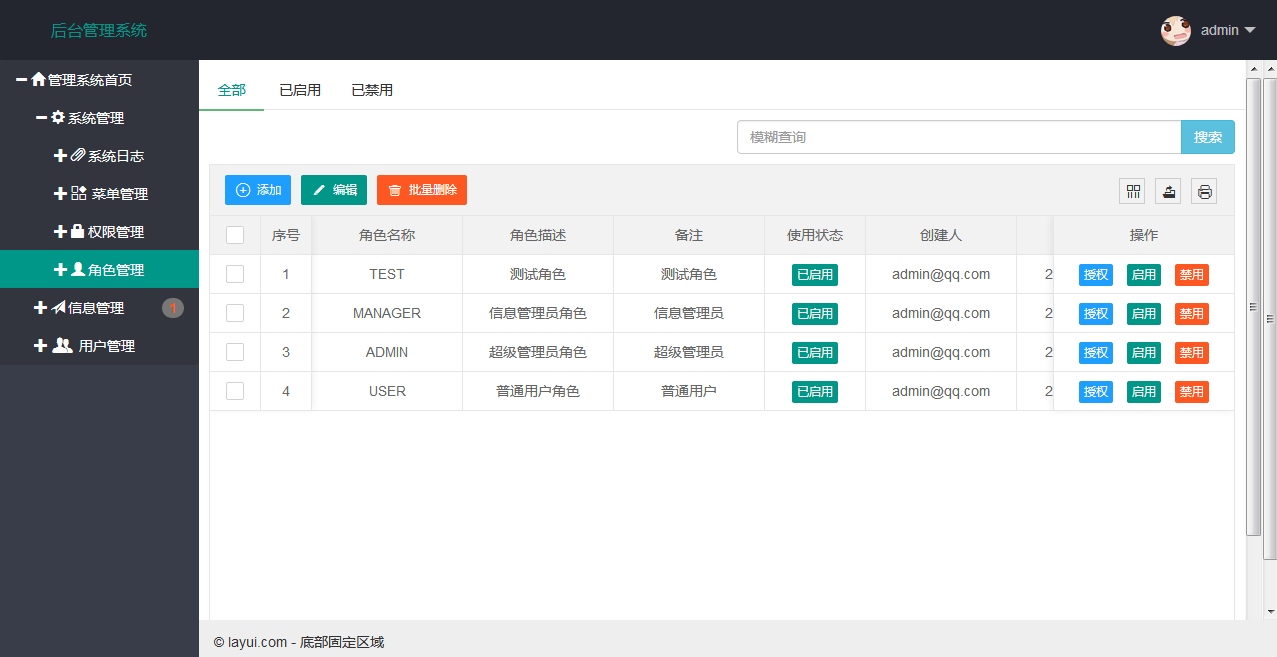


图3.3.6 角色管理

* 1. 角色授权

角色授权即用来给指定角色授予相关后台菜单权限的，只有被授予指定菜单的相关操作权限后，该角色才能在该菜单中进行相关操作，是系统权限控制的重要组成部分，具体如图3.3.7所示。



图3.3.7 角色授权

（2）信息管理

1. 内容管理

内容管理即用来管理网站前端所显示的相关内容的，具体如图3.3.8所示。

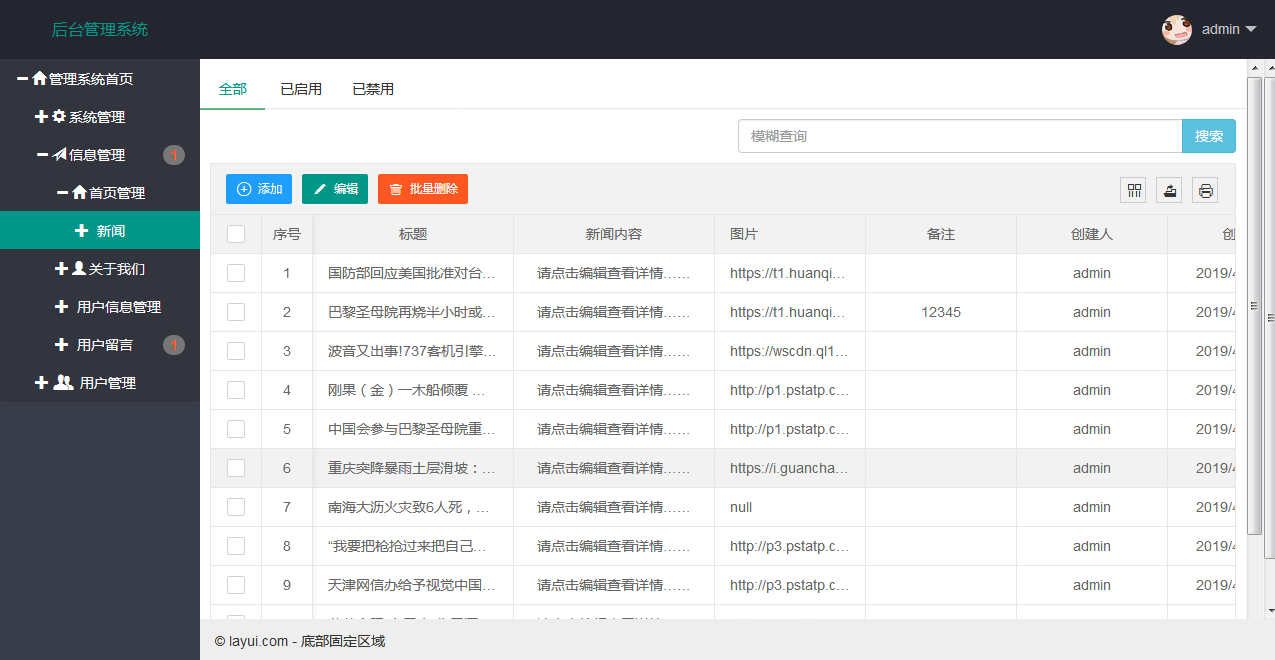


图3.3.8 新闻内容管理

1. 用户留言管理

留言管理即用来管理用户在前端页面的留言信息，用户发布留言后，需要经过后台审核，审核通过后方可对其他用户显示，具体如图3.3.9所示。

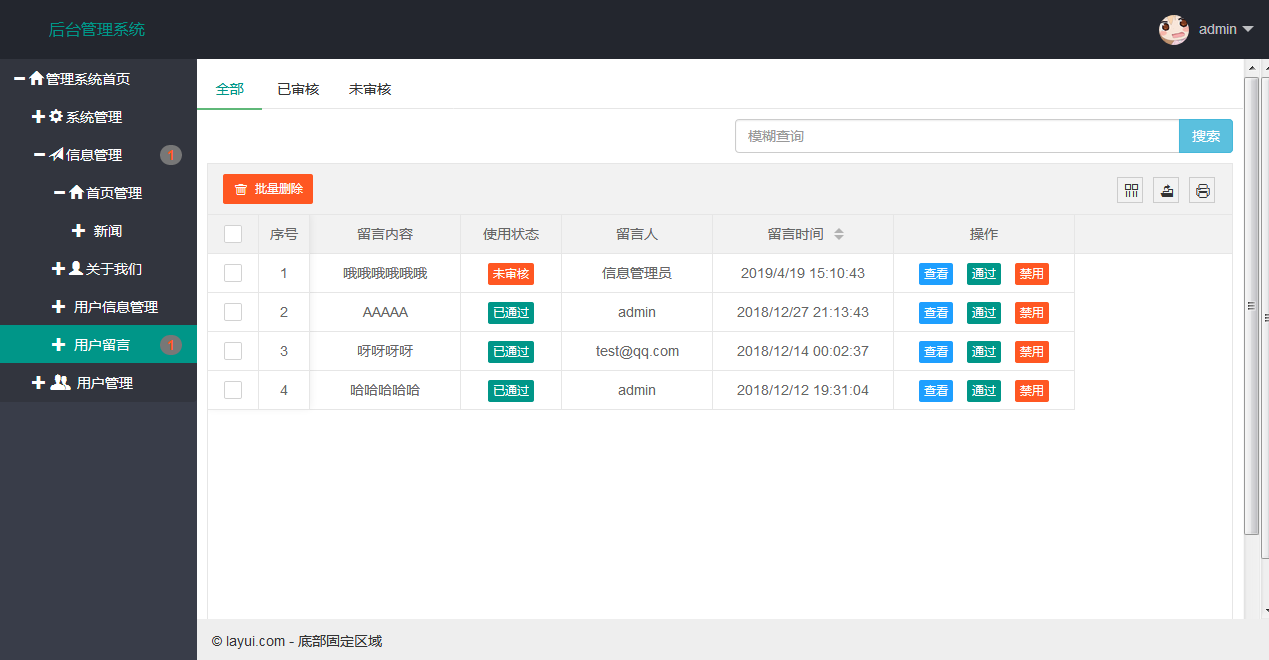


图3.3.9 用户留言管理

（3）用户管理

用户管理包括管理员管理、普通用户管理、IP封禁管理，主要是用来对系统的已注册用户进行相关管理的，包括用户账号的启用/禁用、管理员的添加、删除、授权，和用户登录IP的锁定等。

1. 管理员管理

管理员管理是超级管理员用来管理系统中的普通管理员的，可在此对指定的管理员进行授权等操作，是系统权限控制的重要组成部分，具体如图3.3.10所示。



图3.3.10 管理员管理

1. 管理员授权

管理员授权是将指定的角色及其包含的权限授给指定的管理员，让其有权在系统后台进行相关操作，是系统权限控制的重要组成部分，具体如图3.3.11所示。

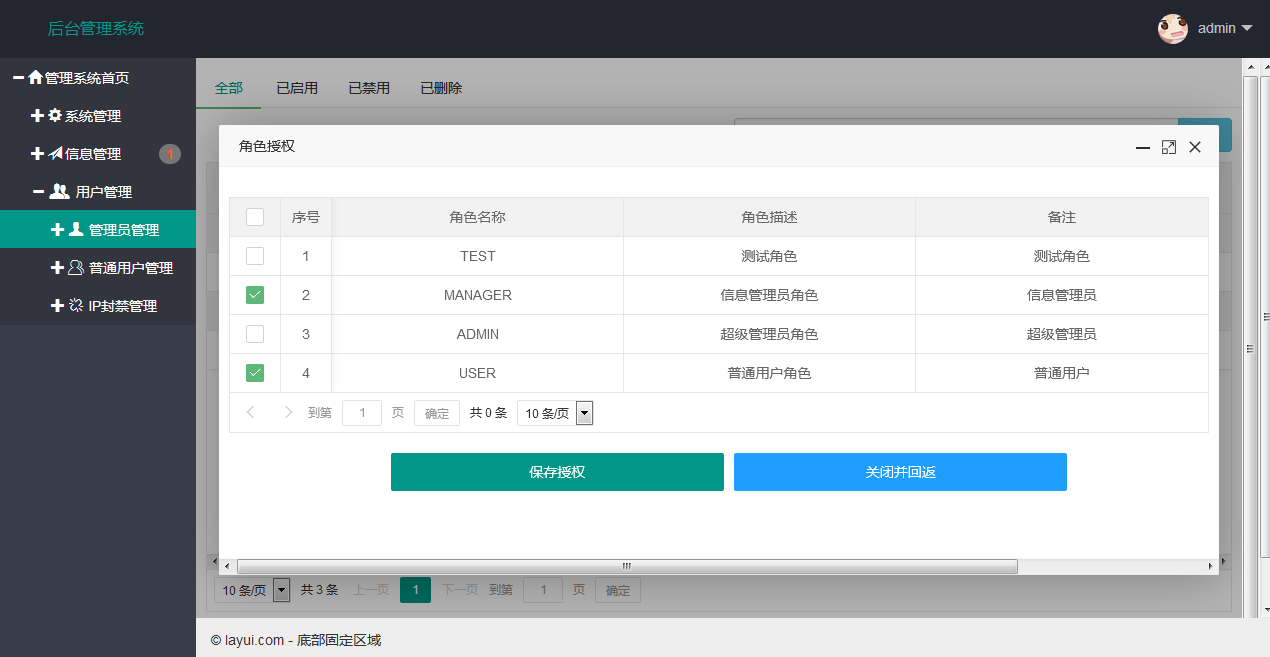


图3.3.11 管理员授权

# 第四章 系统架构及并发性能优化

新闻列表接口是系统前端页面，即系统门户网站中所有数据的来源接口，当多个用户并发访问内容管理系统前端的新闻信息时，等同于并发访问新闻列表接口，所以只要新闻列表接口能满足高并发访问的性能需求，就相当于内容管理系统能满足高并发访问。

系统能否满足“高并发访问”的性能需求，需要在其在构建完成后，对其进行并发访问测试才知晓，若系统无法满足性能要求，则需要对其进行优化。本章是此次论文研究的重点，目的是通过不断的对“新闻列表接口”进行并发访问测试与优化，使其最终能满足高并发访问的性能需求。

本章将详细介绍使用JMeter模拟多用户并发访问“新闻列表接口”的测试与优化过程，记录测试过程中的相关性能数据，并对性能数据进行分析，依据分析结果为被测接口制定出相应的优化方案，最后执行优化方案对被测接口进行优化。优化操作包括代码优化、数据库优化、JVM优化、添加缓存优化、系统集群优化等[9]。

## 4.1 初始系统简介

初始内容管理系统，即根据需求分析完成了所有功能实现的系统，但因没有经过任何测试与优化，所以其性能和可靠性没有任何保证。

本节将大体介绍初始系统中，会影响“新闻列表”接口性能的系统架构、所用技术及数据库设计等，以便与后面的优化章节做对比。

### 4.1.1 系统架构

（1）架构图

初始系统的系统架构如图4.1.1所示。

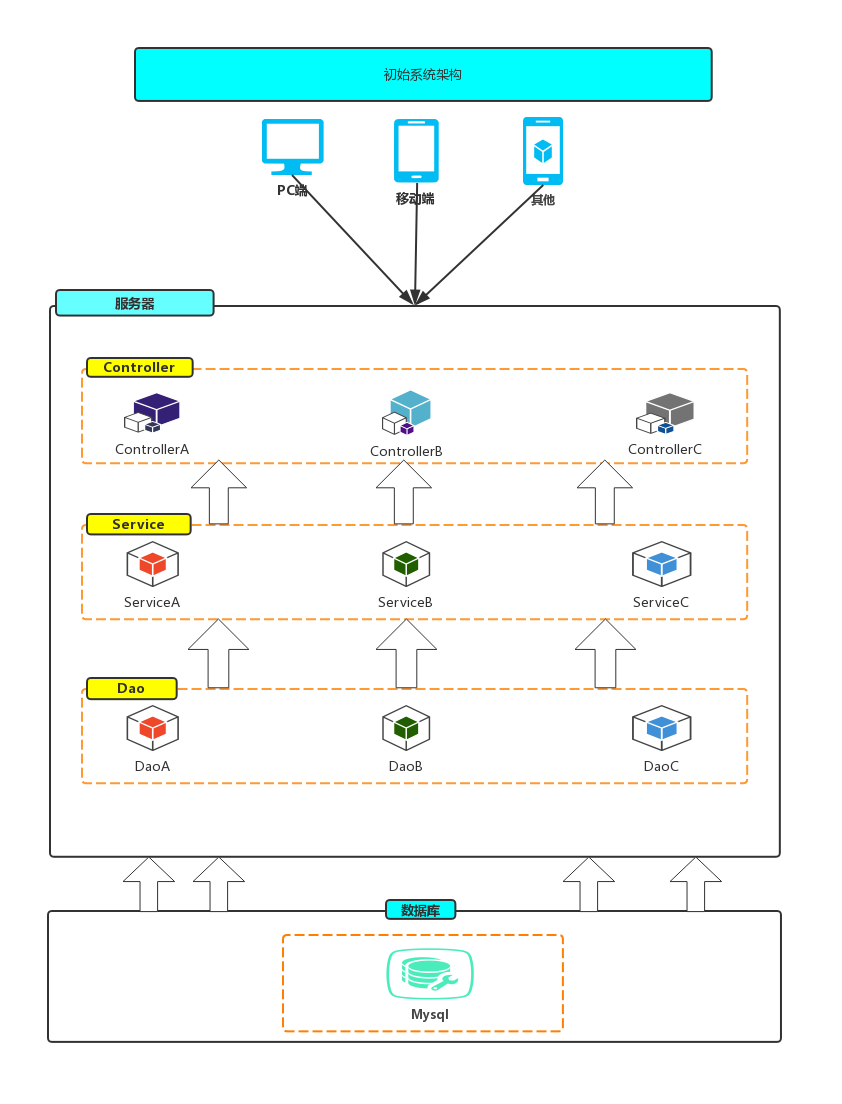


图4.1.1 初始系统架构图

### 4.1.2 系统技术实现

Servlet容器：Tomcat

权限控制：SpringSecurity

前端：Html/CSS、JavaScript、JQuery、Bootstrap、Layui、WangEditor等。

后端：SpringBoot、SpringMVC、Mybatis、Kaptcha、Lombok等。

前后端交互：Ajax、Json

数据库：Docker-Mysql（除了主键，没有其他索引）

项目结构控制：Maven

项目版本控制：Git/Github

开发模式：前后端分离

### 4.1.3 被测接口数据库设计

因被测的是新闻列表接口，所以本章只重点展示与该接口相关的数据库设计，其他设计不再展示。

1. 数据库关系模型

初始系统新闻列表接口涉及的数据库关系模型如图4.1.2所示。

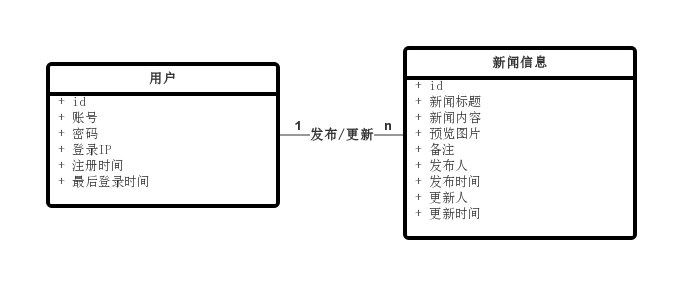


图4.1.2 新闻列表接口数据库关系模型图

1. 数据库表结构及其SQL语句
2. 用户表
   1. 表结构

初始系统用户表的表结构如图4.1.3所示。

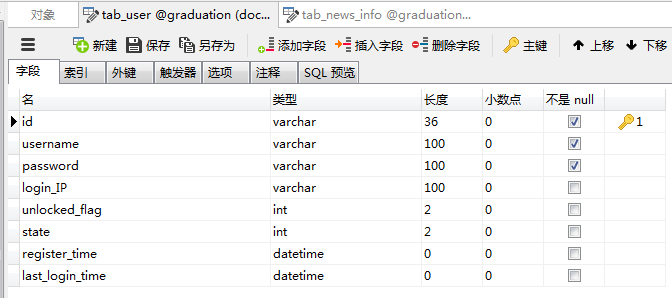


图4.1.3 用户表表结构

* 1. SQL语句

初始系统用户表的SQL语句如图4.1.4所示。



图4.1.4 用户表SQL语句

1. 新闻信息表
   1. 表结构

初始系统新闻信息表的表结构如图4.1.5所示。

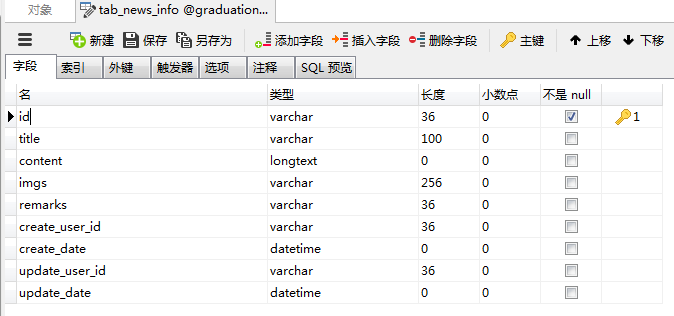


图4.1.5 新闻信息表表结构

* 1. SQL语句

初始系统新闻信息表的SQL语句如图4.1.6所示。



图4.1.6 新闻信息表SQL语句

## 4.2 高并发压测原理及方法

接下来本文中会多次涉及到使用JMeter对系统接口进行并发访问测试，所以本节需要先简要介绍JMeter及使用JMeter测试时的并发测试理论、测试方法和大体测试步骤。

### 4.2.1 相关工具介绍

（1）JMeter简介

JMeter是Apache组织的开放源码项目，它是一款在国外非常流行和受欢迎的开源的功能和性能测试工具，纯Java实现，可以用于测试静态、动态资源的性能，如Java Servlets、CGI Scripts、数据库等[10]，它用于模拟在服务器、网络或者其他对象上附加高负载以测试它们提供服务的受压能力以及分析它们提供的服务在不同负载下的总体性能情况，它有图形化的用户界面，方便测试人员使用，学习曲线低，且有开放的源代码供测试人员研究[11]。

JMeter的测试计划由各种JMeter元件组成，一个完整的测试计划包含一个或多个线程组（ThreadGroup），取样器（Sampler），配置元件（Config Elements），定时器（Timer），逻辑控制器（Logic Controller），前置处理器，后置处理器，断言（Assertion），监听器（Listener）[12]等，具体可去JMeter官网查看这里不再详述。

（2）SpringBootAdmin简介

Spring Boot Admin是一个社区项目，用于管理和监视Spring Boot应用程序，可监控SpringBoot应用的GC次数和Java内存的使用情况等数据[13]。

（3）Druid数据源

Druid是Java语言中最好的数据库连接池，能够提供强大的监控和扩展功能，能监控项目的URI请求状况和SQL执行状况等重要信息[14]。

### 4.2.2 JMeter并发测试理论

开启JMeter线程组进行接口测试时，需要配置线程组的线程数（Number of Threads）、线程组所有线程数的总启动时间（Ramp-Up Period）以及每一个线程的循环次数（Loop Count）[12]，要想在JMerer测试时，达到真正的并发访问效果，需要如下的JMeter并发测试理论：

设线程组的线程数为n，线程组的总启动时间为T，线程组的循环次数为a，被测接口的平均响应时间为t，单个线程的运行时间为R，这n个线程从测试开始时到n个线程全部并发访问被测接口的时间段为[S,R]，则有：

* + 1. S = (T-T/n)，即最后一个线程的启动时间（第x秒）等于总启动时间减去总启动时间除以线程数。
    2. a>S/t，即若想n个线程达到并发访问测试接口的效果，这n个线程所需要的循环次数必须大于最后一个线程的启动时间除以被测接口的平均响应时间。
    3. R=a\*t，即单个线程在测试过程中的总运行时间等于接口平均响应时间乘以线程组的循环次数。

综上可知，n个线程在对接口进行并发测试时，并发时间段为[S,R]，即第S秒到第R秒，其他时间段都不是n个线程对测试接口进行并发访问。

其中，被测接口的平均响应时间（t）需要通过一次JMeter并发测试的聚合报告来获取。本文的获取方式是通过设置一个线程数一定，且循环次数足够多的线程组（5个线程，循环1000次）来对被测接口进行预测式，如此获取的平均响应时间与实际测试时的误差才会足够小。

### 4.2.3 并发测试方法及步骤

（1）测试方法

根据JMeter并发测试理论，本人将对新闻列表接口的并发访问压力测试分为了接口性能测试与接口压力测试两个阶段进行，具体如下：

1. 阶段一，接口性能测试。

本阶段的测试目的是为提升被测接口的响应速度提供优化依据，主要是测试被测接口的响应性能，所以只用一个包含10个线程的线程组对被测接口进行并发访问测试，需记录测试前、测试时、测试后各个监控工具的监控数据和JMeter聚合报告的数据。

1. 阶段二，接口压力测试。

本阶段的测试目的是为提升被测接口的并发承载量提供优化依据，主要是测试被测接口的在不同线程组并发访问下的响应速度和吞吐量，以确定被测接口的能承受的并发访问量，所以需要使用线程数逐渐增多的多个线程组对被测接口进行并发访问测试。

因本阶段测试次数较多，执行本阶段测试的前提是被测接口已经过阶段一的测试，且平均响应时间在500毫秒以内，本阶段才能进行，否则就需要先对被测接口进行响应性能优化。

（2）测试步骤

为了能达到对新闻列表接口的并发访问效果，两个测试阶段的每一次测试都需要按照如下测试步骤进行：

1. 按照JMeter并发测试理论，测试前需要用JMeter设置线程数为5，循环次数为1000的线程组，对待测接口进行HTTP并发访问，根据JMeter聚合报告，得出待测接口的平均响应时间。
2. 根据并发测试理论，计算被测接口并发访问时间段。
3. 测试前，需记录各监控工具的监控数据。
4. 测试时，需记录接口并发访问时间段内的各监控工具的监控数据。
5. 测试后，需记录各监控工具的监控数据及JMeter聚合报告的数据。

### 4.2.4 测试期间需要记录的监控数据

测试期间需要记录的监控数据是根据测试阶段而变化的，不同测试阶段需要记录的数据维度不一样。

（1）阶段一，接口性能测试。

本阶段的测试是为提高被测接口响应速度提高依据，因影响接口响应速度的因素比较多，所以本阶段需要监控和记录的数据也比较多，具体如下：

1. JMeter聚合报告相关数据。
   1. 请求样本数
   2. 平均响应时间
   3. 请求失败率
   4. 吞吐量
2. Druid数据源监控数据。
   1. URI请求数
   2. URI请求时间和
   3. URI最大并发数
   4. Jdbc出错数
   5. SQL执行数
   6. SQL请求时间和
   7. SQL最大并发数
3. SpringBootAdmin监控数据。
4. 测试期间系统Java堆内存与非堆内存的使用图
5. 测试过程中的系统GC日志数据。
6. 测试过程中的GC总次数及总耗时
7. YGC次数及耗时
8. FGC次数及耗时
9. 测试过程中的ECS服务器监控数据。
   1. 最高CPU使用率
   2. 网络带宽出方向最高使用
   3. 网络带宽入方向最高使用

（2）阶段二，接口压力测试。

本阶段的测试目的是为提高被测接口的吞吐量提供优化依据，在本阶段之前，系统性能因经过阶段一的测试，已被优化到了一定程度，所以本阶段只重点监控如下数据：

1. JMeter聚合报告相关数据
   1. 平均响应时间
   2. 请求失败率
   3. 吞吐量
2. Druid数据源监控数据
   1. URI请求数
   2. URI请求时间和
   3. URI最大并发数
3. 测试过程中的ECS服务器监控数据
   1. ECS服务器监控图
   2. 最高CPU使用率
   3. 网络带宽出方向最高使用
   4. 网络带宽入方向最高使用

## 4.3 初始系统性能测试

本节主要介绍使用JMeter模拟多用户并发访问初始系统的新闻列表接口，记录相关测试数据并分析新闻列表接口的性能，目的是看初始系统是否满足高并发访问的性能需求。

### 4.3.1 测试信息

（1）测试环境

对初始系统的新闻列表接口进行性能测试时的测试环境如表4.3.1所示。

表4.3.1 初始系统测试环境信息

|  |  |
| --- | --- |
| 属性 | 值 |
| 服务器 | 阿里云ECS云服务器 |
| 运行系统 | CentOS 7.3 64位 |
| CPU | 1 vCPU |
| 内存 | 2GB(IO优化) |
| 带宽 | 1MBbps |

表4.3.1 （续）

|  |  |
| --- | --- |
| JDK | 1.8.0\_151 |
| JVM | Java HotSpot(TM) 64-Bit Server VM |

（2）被测接口信息

1. 接口信息

被测新闻列表接口的相关信息如表4.3.2所示。

表4.3.2 被测接口信息

|  |  |
| --- | --- |
| 属性 | 值 |
| 接口访问地址 | http://120.78.59.162 |
| 接口访问端口 | 8081 |
| 接口URL | /newsInfo/list |
| 接口参数 | currentPage=1&rows=10&condition=%7B%22searchContent%22%3A%22%22%7D |
| 请求方法 | get |
| 单表数据量 | 1600 |

1. 被测接口涉及的SQL语句

本轮测试涉及到的SQL语句如图4.3.1所示。

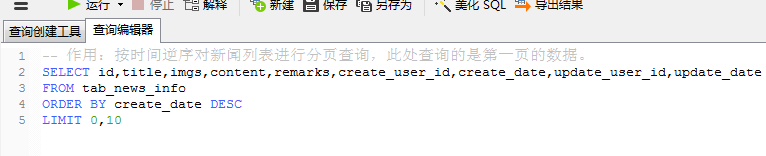


图4.3.1 被测接口SQL语句

（3）测试工具

Apche-JMeter(5.1.1 r1855137)

### 4.3.2 测试计划

根据4.2节中所述的高并发压测原理及方法，预先使用JMeter（5线程，循环1000次）并发访问被测的新闻列表接口，获取到的平均响应时间为234030毫秒，故为其制定第一阶段的测试计划。根据平均响应时间和JMeter并发测试理论，为本轮测试配置JMeter相关参数，具体如表4.3.3所示。

表4.3.3 JMeter测试配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 并发所需最低循环次数（次） | 实际设置循环次数（次） | 并发时间段 | 请求/响应超时限制（秒） |
| 10 | 1 | 1000 | 第9秒~第235秒 | 无 |

### 4.3.3 测试结果

（1）测试时间

本轮测试的时间信息如表4.3.4所示。

表4.3.4 测试时间信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 测试开始时间 | 并发记录时间 | 测试结束时间 | 测试总耗时（秒） |
| 10 | 2019年4月22日13:01:09 | 2019年4月22日13:01:42 | 2019年4月22日13:09:13 | 484 |

（2）JMeter聚合报告相关数据

本轮测试后的JMeter聚合报告数据如表4.3.5所示。

表4.3.5 JMeter聚合报告

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 请求样本数（次） | 平均响应时间（毫秒） | 吞吐量（次/秒） | 请求失败率（%） |
| 10 | 10 | 466280 | 0.02 | 0 |

（3）Druid数据源监控数据。

本轮测试的Druid数据源监控数据如图4.3.6所示，数据都已经过处理，都是是期间的，即将测试后的数据减去测试前的数据所得。

表4.3.6 Druid监控数据

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 线程组线程数（个） | URI请求数（次） | URI请求时间和（毫秒） | URI最大并发数（个） | Jdbc出错数（条） | SQL执行数（次） | SQL执行时间和（毫秒） | SQL最大并发数（个） |
| 10 | 10 | 4658110 | 10 | 0 | 10 | 4635435 | 10 |

（4）SpringBootAdmin监控数据。

本轮测试的SpringBootAdmin监控数据如图4.3.2所示

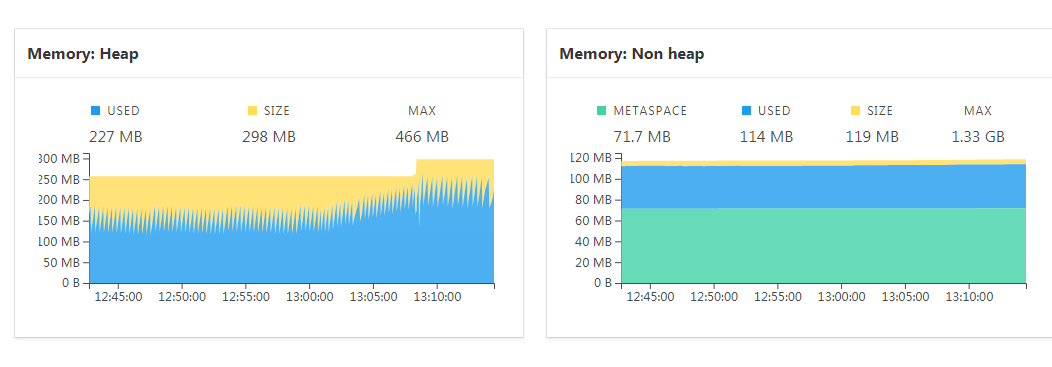


图4.3.2 SpringBootAdmin堆内存使用图

（5）测试过程中的系统GC日志数据。

本轮测试的GC日志数据如表4.3.7所示。

表4.3.7 GC日志数据

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 线程组线程数（个） | YGC次数（次） | YGC耗时（秒） | FGC次数（次） | FGC耗时（秒） | GC总次数（次） | GC总耗时（秒） |
| 10 | 109 | 0.793 | 4 | 0.527 | 113 | 1.32 |

（6）测试过程中的ECS服务器监控数据。

1. ECS服务器监控图

测试期间ECS服务器的监控图如图4.3.3所示。



图4.3.3 ECS服务器监控图

1. ECS服务器监控信息

本轮测试的ECS服务器监控信息如表4.3.8所示。

表4.3.8 ECS服务器监控数据

|  |  |  |  |
| --- | --- | --- | --- |
| 线程组线程数（个） | CPU最高使用率（%） | 网络带宽最高使用-入方向（KB/S） | 网络带宽最高使用-出方向（KB/S） |
| 10 | 20以下 | 1024 | 1024 |

### 4.3.4 性能分析

（1）测试耗时分析

10个线程并发访问被测接口，循环一次耗时484秒，平均每次访问耗时48.4秒，说明被测接口响应速度极慢，系统性能存在极大问题，急需优化。

（2）JMeter聚合报告数据分析

平均响应时间为466280毫秒，吞吐量为0.02次/秒，进一步说明被测接口性能太差，响应时间太长。

（3）Druid监控数据分析

分析Druid监控数据之前，需要将数据进行计算和汇总。

1. 汇总数据

汇总数据如表4.3.9所示。

表4.3.9 Druid监控数据计算汇总

|  |  |
| --- | --- |
| 属性（单位） | 值 |
| URI平均请求耗时（毫秒） | 465811 |
| SQL平均执行耗时（毫秒） | 463543.5 |
| 接口请求SQL执行率（%） | 100 |

1. 表4.3.9的相关计算公式

URI平均请求耗时 = URI请求时间和/URI请求总次数

SQL平均执行耗时 = SQL总执行时间/SQL执行总次数

接口请求SQL执行率 = SQL执行总次数/URI请求总次数\*100%

1. 数据分析

由汇总数据可知：

1. URI平均请求耗时太长，高达465811毫秒，说明被测接口性能太差。
2. SQL执行耗时太长，高达463543.5毫秒，说明查询接口数据的SQL语句存在极大问题，查询效率极低。

使用Mysql Explian命令查看该SQL语句的执行计划，具体信息如图4.3.4所示。

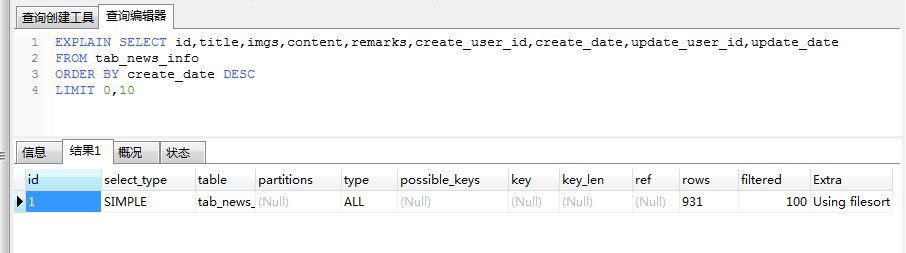


图4.3.4 Mysql Explain查询计划

由SQL执行计划可知，该查询SQL的问题非常严重，没有索引，且是全表扫描，还用了文件做排序。

1. 接口请求SQL执行率为100%，说明系统没有缓存，每次对接口的请求都要查询一次数据库，高并发访问时数据库的压力会很大。

（4）SpringBootAdmin监控数据分析

由监控图可知，测试过程中Java最大堆内存发生了明显扩容，这会影响系统性能。

（5）GC日志数据分析

1. 测试过程中系统发生的GC次数太高，GC频率高达0.233 次/秒，且有4次FGC，对系统性能影响极大。
2. GC频率=测试过程中的GC总次数/测试总耗时

（6）ECS服务器监控数据

A、测试过程中系统CPU使用率不高，在20%以下，说明CPU对测试影响不大。

B、测试过程中网络带宽的出入方向都用到了服务器极限的1MB/S，对系统性能和吞吐量影响很大。

### 4.3.5 测试总结

由4.3.4的测试与分析可知，当前系统无法满足高并发访问的性能需求，具体原因如下：

1. 被测接口响应速度太慢，吞吐量太低。
2. 请求的URI耗时太长。
3. SQL执行耗时太长，查询数据的SQL语句问题很大，查询十分效率低下。
4. 系统没有缓存，每次URI请求都会执行一次低效率的SQL查询，高并发访问时数据库压力会很大。
5. 系统堆内存分配不合理，测试过程中最大Java堆内存发生了扩容，且JVM的GC太频繁，还出现FGC，极大影响了系统性能。
6. 服务器网络带宽太小，限制了系统的吞吐量。

## 4.4 数据库与JVM优化

### 4.4.1 优化背景

由第4.3节的测试结果可知，被测接口的平均响应时间太长，吞吐量太低，说明被测接口性能太差，完全不能满足高并发访问的性能需求，必须对被测接口进行优化。

### 4.4.2 优化方案

（1）问题分析

根据第4.3.5节的测试总结可知，系统整体性能很差，存在问题很多，经过分析，发现主要问题如下：

1. SQL语句问题严重，导致查询效率太低。
2. 系统的JVM堆内存分配不合理，导致GC太频繁，还出现FGC。
3. 系统没有缓存，导致每次请求都要查询数据库，并发量大时数据库压力会很大。
4. 服务器带宽太小，限制了被测接口的吞吐量。

（2）优化方向分析

其中问题①和问题②是系统核心问题，对系统性能的直接影响最大，一个关乎数据库，另一个关乎JVM，所以本轮优化决定先从这两个方向入手。

（2）优化操作

* 1. 数据库优化

要对数据库进行优化需要具备相关的优化知识，内容非常多，常见的优化方向有SQL语句优化、表字段优化、索引优化、分区、分表、分库等，具体操作便不再细说[15] [16]。

* 1. JVM优化

JVM调优是Java中的重点与难点，内容也非常多，常见的优化手段就是使用Java提供的各种命令和观测工具对运行中的JVM进行观测，然后根据JVM的相关信息对JVM参数进行调节，具体内容太多，这里也不再详述[17]。

### 4.4.3 优化后的系统变化

本轮优化是数据库与JVM优化，优化后的系统架构、设计、技术等会有所变化，本节将介绍这些相关的变化。

（1）数据库设计变化

经过优化后，系统的数据库设计发生了变化，具体如下。

* 1. ER设计图

优化后被测接口的ER设计图如图4.4.1所示。

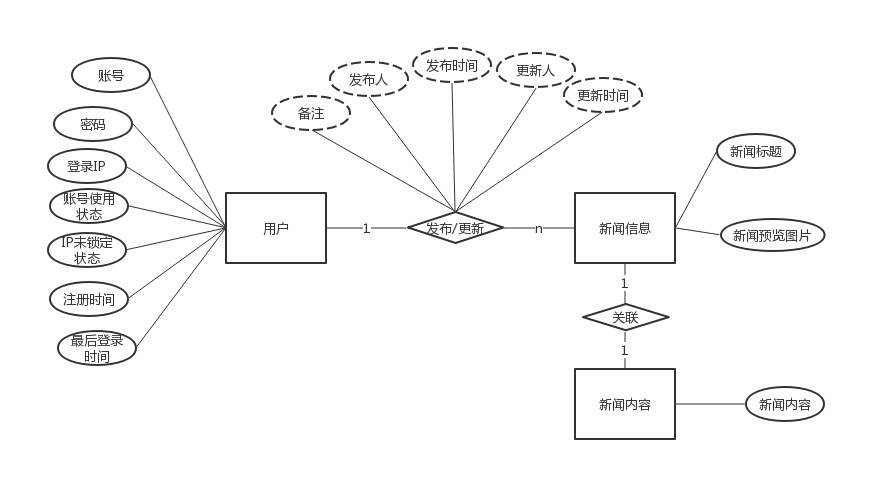


图4.4.1 新闻列表接口ER设计图

* 1. 数据库关系模型

优化后被测接口的数据库关系模型如图4.4.2所示。

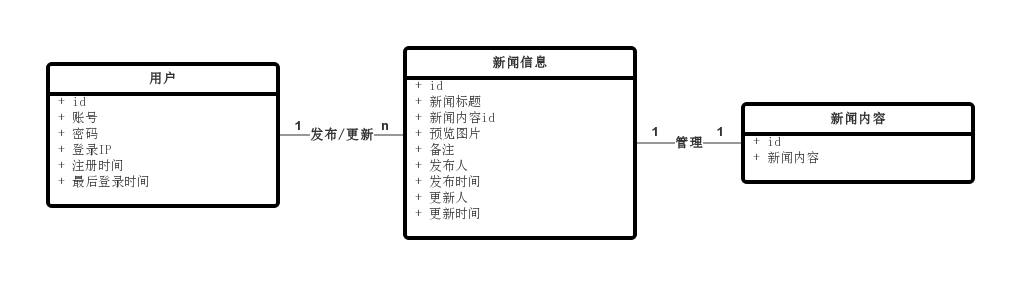


图4.4.2 新闻列表接口数据库关系模型

* 1. 数据库表结构及SQL语句

1. 用户表

无变化。

1. 新闻信息表
   1. 表结构

优化后的新闻信息表结构如图4.4.3所示。

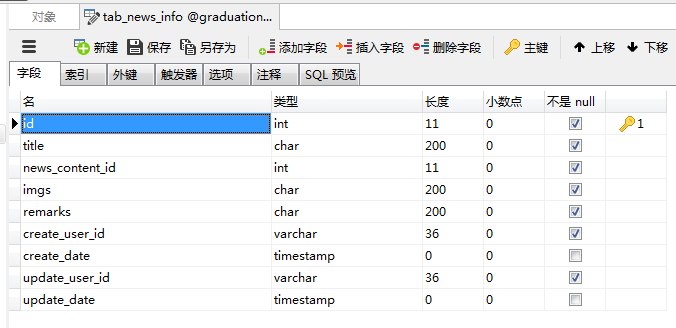


图4.4.3 新闻信息表结构

* 1. SQL语句

优化后的新闻信息表结构如4.4.4所示。



图4.4.4 新闻信息表SQL语句

1. 新闻内容表

新闻内容表是从新闻信息表中分离出来的，目的是为了提高查询效率。

* 1. 表结构

优化有新增的新闻内容表结构如图4.4.5所示。

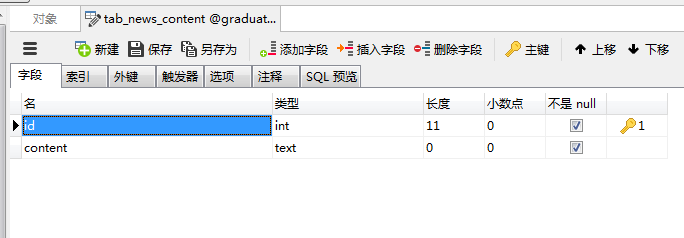


图4.4.5 新闻内容表结构

* 1. SQL语句

优化后新增的新闻内容表SQL语句如图4.4.6所示。

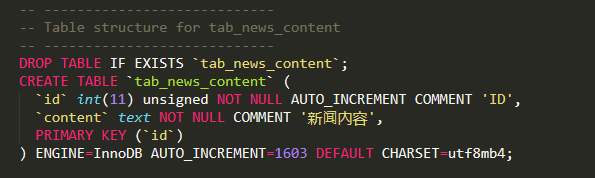


图4.4.6 新闻内容表SQL语句

（2）其他变化

1. 优化后的SQL语句数据查询效果与优化前一样，具体如图4.4.7所示。

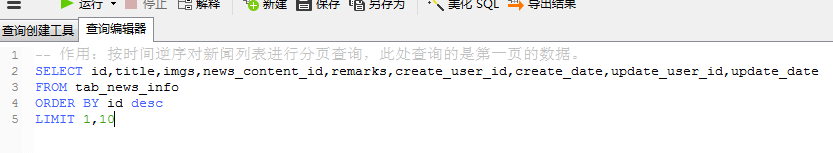


图4.4.7 被测接口SQL语句

1. JVM参数

由于优化了JVM，所以添加了如下的项目启动参数：

-Xms768m -Xmx768m -Xmn256m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss256k -XX:+UseG1GC

### 4.4.4 优化后接口性能测试

（1）测试信息

1. 测试环境

与第4.3.1节保持一致。

1. 被测接口信息
   1. 接口信息访问

端口变成8082，其余与第4.3.1节保持一致。

* 1. 被测接口涉及的SQL语句

如图4.4.7所示。

1. 测试工具

Apche-JMeter(5.1.1 r1855137)

（2）测试计划

根据4.2节中所述的高并发压测原理及方法，预先使用JMeter（5线程，循环1000次）并发访问被测的新闻列表接口，获取到的平均响应时间为348毫秒，因没有经过第一阶段的测试，所以先进行第一阶段的测试。根据平均响应时间和JMeter并发测试理论，为本轮测试配置JMeter相关参数，具体如表4.4.1所示。

表4.4.1 JMeter测试配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 所需最低循环次数（次） | 实际设置循环次数（次） | 并发时间段 | 请求/响应超时限制（秒） |
| 10 | 26 | 1000 | 第9秒~第348秒 | 60 |

（3）测试结果

1. 测试时间

本轮测试的时间信息如表4.4.2所示。

表4.4.2 测试时间信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 测试开始时间 | 并发记录时间 | 测试结束时间 | 测试总耗时（秒） |
| 10 | 2019年4月23日23:07:52 | 2019年4月23日23:08:25 | 2019年4月23日23:20:22 | 750 |

1. JMeter聚合报告相关数据

本轮测试后的JMeter聚合报告数据如表4.4.3所示。

表4.4.3 JMeter聚合报告

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 请求样本数（次） | 平均响应时间（毫秒） | 吞吐量（次/秒） | 请求失败率（%） |
| 10 | 10000 | 723 | 13.3 | 0 |

1. Druid数据源监控数据。

本轮测试的Druid数据源监控数据如图4.4.4所示，数据都已经过处理，都是是期间的，即将测试后的数据减去测试前的数据所得。

表4.4.4 Druid监控数据

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 线程组线程数（个） | URI请求数（次） | URI请求时间和（毫秒） | URI最大并发数（个） | Jdbc出错数（条） | SQL执行数（次） | SQL执行时间和（毫秒） | SQL最大并发数（个） |
| 10 | 10000 | 1596358 | 8 | 0 | 10000 | 1458645 | 8 |

1. SpringBootAdmin监控数据。

本轮测试的SpringBootAdmin监控数据如图4.4.8所示

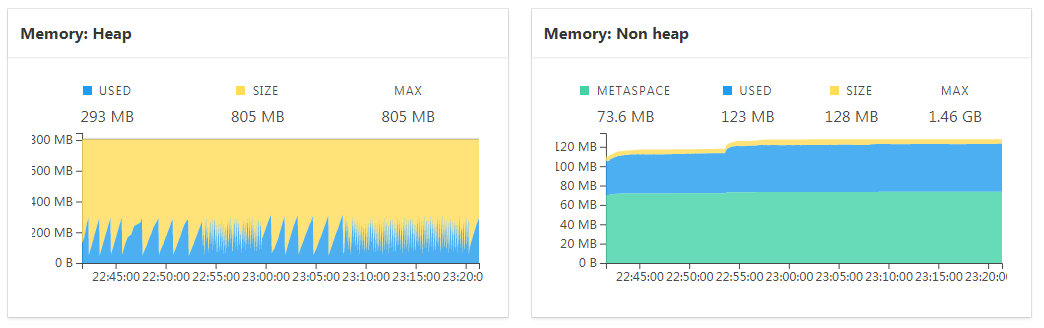


图4.4.8 SpringBootAdmin堆内存信息

1. 测试过程中的系统GC日志数据。

本轮测试的GC日志数据如表4.4.5所示。

表4.4.5 GC日志数据

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 线程组线程数（个） | YGC次数（次） | YGC耗时（秒） | FGC次数（次） | FGC耗时（秒） | GC总次数（次） | GC总耗时（秒） |
| 10 | 60 | 0.518 | 0 | 0 | 60 | 0.518 |

1. 测试过程中的ECS服务器监控数据。
   1. ECS服务器监控图

测试期间ECS服务器的监控图如图4.4.9所示。



图4.4.9 ECS服务监控信息

* 1. ECS服务器监控信息

本轮测试的ECS服务器监控信息如表4.4.6所示。

表4.4.6 ECS服务器监控数据

|  |  |  |  |
| --- | --- | --- | --- |
| 线程组线程数（个） | CPU最高使用率（%） | 网络带宽最高使用-入方向（KB/S） | 网络带宽最高使用-出方向（KB/S） |
| 10 | 50以下 | 700以下 | 1024 |

（4）性能分析

1. 测试耗时分析

10个线程并发访问被测接口，1000次循环耗时750秒，平均每次访问耗时75毫秒，说明被测接口响应速度很快。

1. JMeter聚合报告数据分析

平均响应时间为723毫秒，吞吐量为13.3次/秒，说明被测接口响应速度还行，平均响应时间在1秒以内，但吞吐量不高。

1. Druid监控数据分析

分析Druid监控数据之前，需要将数据进行计算和汇总。

* 1. 汇总数据

汇总数据如表4.4.7所示。

表4.4.7 Druid监控数据计算汇总

|  |  |
| --- | --- |
| 属性（单位） | 值 |
| URI平均请求耗时（毫秒） | 159.64 |
| SQL平均执行耗时（毫秒） | 145.86 |
| 接口请求SQL执行率（%） | 100 |

* 1. 表4.4.7的相关计算公式

与表4.3.9的计算公式一致。

* 1. 数据分析

由汇总数据可知：

* + - 1. URI平均请求耗时为159.64毫秒，速度还不错。
      2. SQL平均执行耗时为145.86毫秒，查询速度较快，说明SQL语句查询效率较高。

使用Mysql Explian命令查看该SQL语句的执行计划，具体信息如图4.4.10所示。

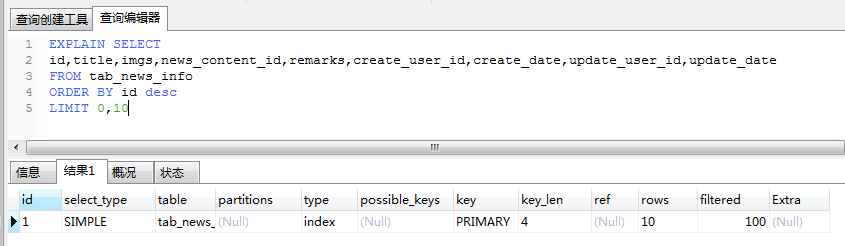


图4.4.10 Mysql Explain查询计划

由SQL执行计划可知，该sql语句查询的是索引树，用到了主键索引，排序也是根据索引排的，所以查询效率较高。

* + - 1. 接口请求SQL执行率为100%，说明系统没有缓存，每次对接口的请求都要查询一次数据库，数据库压力很大。

1. SpringBootAdmin监控数据分析

由监控图可知，测试时间段内Java最大堆内存与最大非堆内存保存不变，没有发生扩容，对测试结果影响不大。

1. GC日志数据分析
   1. 测试过程中GC次数较低，GC频率为0.08次/秒，且没有出现FGC，说明JVM优化得不错，系统性能良好。
   2. GC频率=测试过程中的GC总次数/测试总耗时
2. ECS服务器监控数据
   1. 测试过程中系统CPU使用率不是很高，在50%以下，说明CPU对测试影响不大。
   2. 测试过程中网络带宽入方向的使用虽然高，但还没满，不过出方向却用到了服务器极限的1MB/S，对系统吞吐量影响较大。

（5）测试总结

* + - 1. 被测接口响应速度还不错，就是吞吐量还是很低。
      2. 请求的URI响应速度很快。
      3. SQL执行速度较快，SQL语句的查询效率较高。
      4. 系统没有缓存，每次URI请求都会执行一次SQL。
      5. JVM运行良好。
      6. 系统网络带宽太小，限制了系统的吞吐量。

### 4.4.5 优化前后接口性能对比

优化前后的性能数据如表4.4.8所示。

表4.4.8 优化前后性能对比

|  |  |  |  |
| --- | --- | --- | --- |
| 属性（单位）/阶段 | 优化前 | 优化后 | 提升（倍） |
| JMeter平均响应耗时（毫秒） | 466280 | 723 | 643.92 |
| 吞吐量（次/秒） | 0.02 | 13.3 | 664.00 |
| URI平均响应时间（毫秒） | 465811 | 159.64 | 2916.88 |
| SQL平均执行时间（毫秒） | 463543.5 | 145.86 | 3177.00 |
| SQL执行率（%） | 100 | 100 | 0.00 |
| GC频率（次/秒） | 0.233 | 0.08 | 1.91 |

由表4.4.8 的数据可知，经过数据库与JVM优化后的被测接口，除了SQL执行率外，其他性能都得到了大幅提升。

## 4.5 添加缓存及优化

### 4.5.1 优化背景

由第4.4.4节的接口性能测试可知，经过数据库与JVM优化后，被测接口的性能得到了很大提升，但还存在系统没有缓存、被测接口吞吐量太低、服务器带宽太小等问题，尚未满足吞吐量在100以上的高性能需求，所以还需继续对被测接口进行优化。

### 4.5.2 优化方案

（1）问题分析

根据第4.4.4节的测试总结可知，被测接口优化后还存在几个较为突出的问题，经过分析，发现具体问题如下：

* 1. 系统没有缓存，每条请求都要查询一次数据库，并发访问量大时数据库的压力会很大。
  2. 吞吐量很低。
  3. 服务器带宽太小。

（2）优化方向分析

其中问题①关乎系统本身，需要为系统添加一层缓存，以减轻并发访问量较大时的数据库压力。而问题2和问题3都关乎被测接口的吞吐量，且问题明确指向服务器的带宽，会涉及到服务器自身的硬件性能优化。因系统本身的问题优先于服务器上的问题，所以本轮的优化方向是为系统添加缓存。

（3）优化操作

目前市面上常见的专业缓存工具有Memcache、Redis和Ehcache，三种缓存各有优势，可根据需要具体选择。本轮优化选用的是Ehcache，原因Ehcache是一个专业的Java项目本地缓存，有够快、够简单、够轻量的特点[18]，加上当前的系统只是一个单体MVC项目，没有涉及到复杂的分布式问题，所以选用Ehcache作为缓存性能较高。

添加缓存的具体操作就是利用Mybatis预留的缓存接口，在项目中添加Mybatis与Ehcache的整合组件，让将Ehcahe整合到Mybatis上即可[19]。

### 4.5.3 优化后的系统变化

经过添加缓存后，系统整体会发生响应变化，本节将介绍这些变化。

（1）系统架构变化

优化后的系统架构如图4.5.1所示。

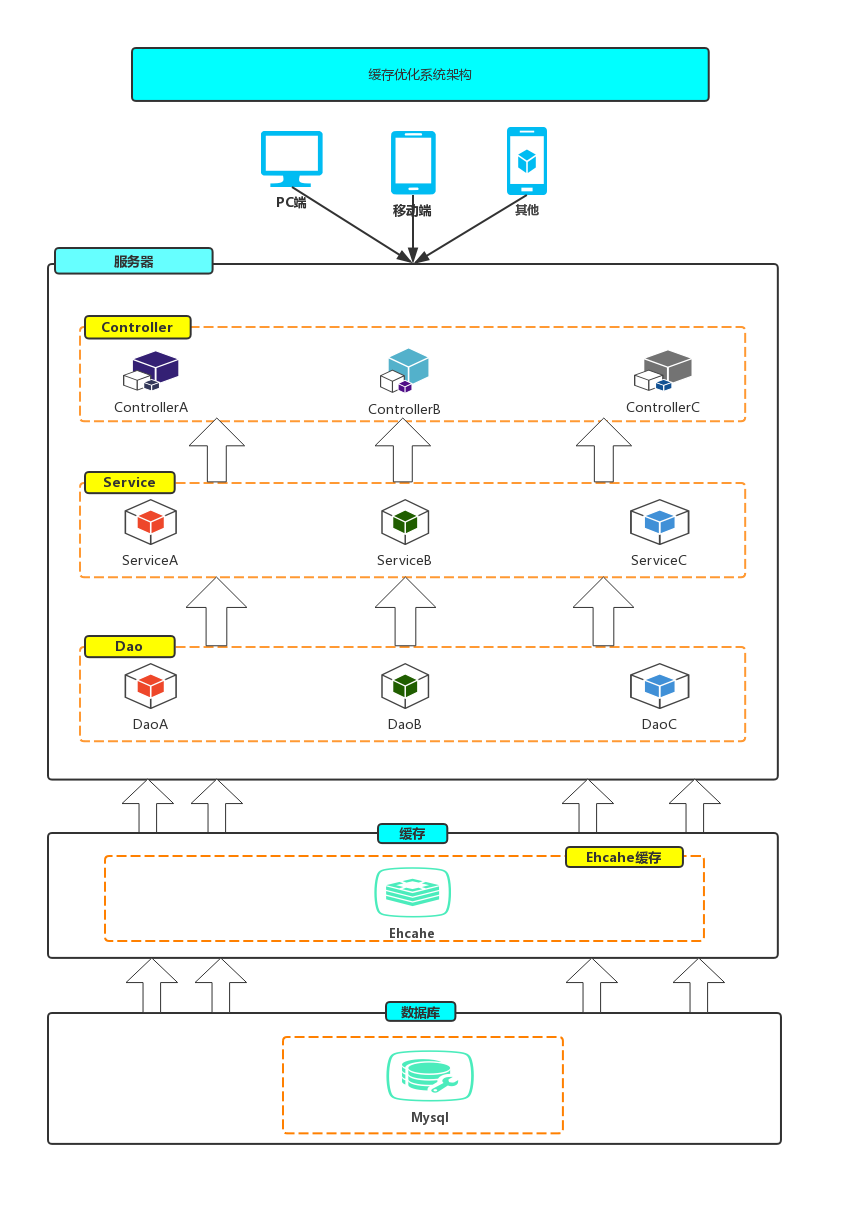


图4.5.1 添加缓存后的系统架构

（2）技术变化

引入Ehcache作为本地缓存。

（3）其他变化

无变化。

### 4.5.4 优化后接口性能测试

（1）测试信息

所有测试信息均与第4.4.4节保持一致。

（2）测试计划

根据4.2节中所述的高并发压测原理及方法，预先使用JMeter（5线程，循环1000次）并发访问被测的新闻列表接口，获取到的平均响应时间为166毫秒，因没有经过第一阶段的测试，所以先进行第一阶段的测试。根据平均响应时间和JMeter并发测试理论，为本轮测试配置JMeter相关参数，具体如表4.5.1所示。

表4.5.1 JMeter测试配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 所需最低循环次数（次） | 实际设置循环次数（次） | 并发时间段 | 请求/响应超时限制（秒） |
| 10 | 55 | 1000 | 第9秒~第166秒 | 60 |

（3）测试结果

1. 测试时间

本轮测试的时间信息如表4.5.2所示。

表4.5.2 测试时间信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 测试开始时间 | 并发记录时间 | 测试结束时间 | 测试总耗时（秒） |
| 10 | 2019年4月26日16:35:21 | 2019年4月26日16:37:14 | 2019年4月26日16:42:07 | 406 |

1. JMeter聚合报告相关数据

本轮测试后的JMeter聚合报告数据如表4.5.3所示。

表4.5.3 JMeter聚合报告

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 请求样本数（次） | 平均响应时间（毫秒） | 吞吐量（次/秒） | 请求失败率（%） |
| 10 | 10000 | 367 | 24.6 | 0 |

1. Druid数据源监控数据。

本轮测试的Druid数据源监控数据如图4.5.4所示，数据都已经过处理，都是是期间的，即将测试后的数据减去测试前的数据所得。

表4.5.4 Druid监控数据

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 线程组线程数（个） | URI请求数（次） | URI请求时间和（毫秒） | URI最大并发数（个） | Jdbc出错数（条） | SQL执行数（次） | SQL执行时间和（毫秒） | SQL最大并发数（个） |
| 10 | 10000 | 115169 | 8 | 0 | 24 | 3093 | 2 |

1. SpringBootAdmin监控数据。

本轮测试的SpringBootAdmin监控数据如图4.5.2所示

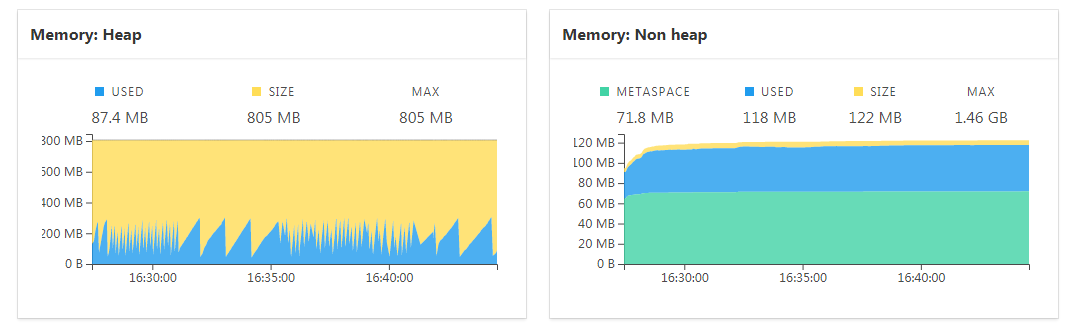


图4.5.2 SpringBootAdmin堆内存信息

1. 测试过程中的系统GC日志数据。

本轮测试的GC日志数据如表4.5.5所示。

表4.5.5 GC日志数据

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 线程组线程数（个） | YGC次数（次） | YGC耗时（秒） | FGC次数（次） | FGC耗时（秒） | GC总次数（次） | GC总耗时（秒） |
| 10 | 41 | 0.334 | 0 | 0 | 41 | 0.334 |

1. 测试过程中的ECS服务器监控数据。
   1. ECS服务器监控图

测试期间ECS服务器的监控图如图4.5.3所示。



图4.5.3 ECS服务监控信息

* 1. ECS服务器监控信息

本轮测试的ECS服务器监控信息如表4.5.6所示。

表4.5.6 ECS服务器监控数据

|  |  |  |  |
| --- | --- | --- | --- |
| 线程组线程数（个） | CPU最高使用率（%） | 网络带宽最高使用-入方向（KB/S） | 网络带宽最高使用-出方向（KB/S） |
| 10 | 60以下 | 300以下 | 1024 |

（4）性能分析

1. 测试耗时分析

10个线程并发访问被测接口，1000次循环耗时406秒，平均每次访问耗时40.6毫秒，说明被测接口响应速度很快。

1. JMeter聚合报告数据分析

平均响应时间为367毫秒，吞吐量为24.6次/秒，说明被测接口响应速度比较快，平均响应时间在0.5秒以内，但吞吐量只有24.6，还是不高。

1. Druid监控数据分析

分析Druid监控数据之前，需要将数据进行计算和汇总。

* 1. 汇总数据

汇总数据如表4.5.7所示。

表4.5.7 Druid监控数据计算汇总

|  |  |
| --- | --- |
| 属性（单位） | 值 |
| URI平均请求耗时（毫秒） | 11.52 |
| SQL平均执行耗时（毫秒） | 128.86 |
| 接口请求SQL执行率（%） | 0.24 |

* 1. 表4.5.7的相关计算公式

与表4.3.9的计算公式一致。

* 1. 数据分析

由汇总数据可知：

* + 1. URI平均请求耗时为11.52毫秒，响应速度很快，说明被测接口的性能不错。
    2. SQL平均执行耗时为128.86毫秒，查询速度挺快，说明SQL语句查询效率很高，因为SQL执行计划与第3.3.4节一致。
    3. 接口请求SQL执行率为0.24%，说明系统有缓存，且缓存为数据库挡住了99.76%的查询请求，效果非常不错。

1. SpringBootAdmin监控数据分析

由监控图可知，测试时间段内Java最大堆内存与最大非堆内存保存不变，没有发生扩容，对测试结果影响不大。

1. GC日志数据分析
   1. 测试过程中GC次数较低，GC频率为0.10次/秒，且没有出现FGC，说明JVM优化得不错，系统性能良好。
   2. GC频率 = 测试过程中的GC总次数÷测试总耗时
2. ECS服务器监控数据
   1. 测试过程中系统CPU使用率不是很高，在60%以下，说明CPU对测试影响不大。
   2. 测试过程中网络带宽入方向的使用虽然高，但还没满，不过出方向却用到了服务器极限的1MB/S，对系统吞吐量影响较大。

（5）测试总结

* + 1. 被测接口响应速度挺快，不过吞吐量还是很低。
    2. 请求的URI响应飞快。
    3. SQL执行速度较快，SQL语句的查询效率较高。
    4. SQL执行率只有0.24%，系统缓存为数据库挡住了99.76%的查询请求。
    5. JVM运行良好。
    6. 系统网络带宽太小，限制了系统的吞吐量。

### 4.5.5 优化前后接口性能对比

优化前后的性能数据如表4.5.8所示。

表4.5.8 优化前后性能对比

|  |  |  |  |
| --- | --- | --- | --- |
| 属性（单位）/阶段 | 优化前 | 优化后 | 提升（倍） |
| JMeter平均响应耗时（毫秒） | 723 | 367 | 0.97 |
| 吞吐量（次/秒） | 13.3 | 24.6 | 0.85 |
| URI平均响应时间（毫秒） | 159.64 | 11.52 | 12.86 |
| SQL平均执行时间（毫秒） | 145.86 | 128.86 | 0.13 |
| SQL执行率（%） | 100 | 0.24 | 415.67 |
| GC频率（次/秒） | 0.08 | 0.10 | -0.25 |

由表4.5.8的数据可知，经过缓存优化后的被测接口，除了GC频率稍微增加，其他性能都有所提升，特别是数据库压力大幅下降。

## 4.6 容器优化与带宽升级

### 4.6.1 优化背景

由第4.5.4节的接口性能测试可知，给系统添加了缓存后，被测接口的性能得到了进一步提升，但被测接口的吞吐量还是很低，无法满足吞吐量在100以上的高性能需求，且服务器带宽太小的问题还没有得到解决，所以还需要继续对被测接口进行优化。

### 4.6.2 优化方案

（1）问题分析

根据第4.5.4节的测试总结可知，被测接口优化后还存在几个较为突出的问题，经过分析，发现具体问题如下：

1. 吞吐量很低。
2. 服务器带宽太小。

（3）优化方向分析

两个问题都关乎被测接口的吞吐量，而吞吐量较低的原因有许多，常见的有系统内部问题、Servlet容器问题和服务器本身的性能问题。经过前两轮优化后，系统的内部问题已基本解决，剩下的便是Servlet容器和服务器的问题，所以本轮优化就从这两个方向入手。

（4）优化操作

1. Servlet容器优化

SpringBoot中嵌入的Servlet容器有Tomcat、Jetty和Undertow，每种Servlet容器都各有所长，如Jetty更擅长处理长连接，Undertow更擅长处理高并发，而Tomcat是SpringBoot默认配置的，性能介于两者中间，相对来说更加稳定[20]。本文研究重点是面向高并发访问优化，而本轮需要优化的问题是被测接口吞吐量太低，所以Undertow容器更符合需求。

1. 服务器带宽升级

因为服务器带宽太小，限制住了被测接口的吞吐量，所以需要做的便是提升服务器的带宽，而系统是部署在阿里云ECS云服务器上的，带宽、CPU等配置均可进行升级，所以要提升服务器的带宽，只需付费进行带宽升级即可。

### 4.6.3 优化后的系统变化

本轮优化是容器优化与带宽升级，默认的Tomcat容器被换成了Undertow容器，而服务器的带宽也从1MB/S被提升到了15MB/S。

（1）技术变化

Tomcat容器被换成了Undertow容器。

（2）其他变化

服务器带宽从1MB/S提升到了15MB/S。

### 4.6.4 优化后接口压力测试

（1）测试信息

除了服务器带宽从1MB/S提升到了15MB/S外，其他测试信息均与第4.5.4节保持一致。

（2）测试计划

根据4.2节中所述的高并发压测原理及方法，预先使用JMeter（5线程，循环1000次）并发访问被测的新闻列表接口，获取到的平均响应时间为123毫秒，由第4.5.4的性能测试可知，被测接口的平均响应时间已降至367毫秒，符合阶段二的测试要求，故本轮举行阶段二的测试。根据平均响应时间和JMeter并发测试理论，为本轮测试配置JMeter相关参数，具体如表4.6.1所示。

表4.6.1 JMeter测试配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数 （个） | 所需最低循环次数（次） | 实际设置循环次数（次） | 并发时间段 | 请求/响应超时限制（秒） |
| 20 | 155 | 1000 | 第19秒~第123秒 | 60 |
| 30 | 236 | 1000 | 第29秒~第123秒 | 60 |

表4.6.1 （续）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 40 | 318 | 1000 | 第39秒~第123秒 | 60 |
| 50 | 399 | 1000 | 第49秒~第123秒 | 60 |
| 100 | 805 | 1000 | 第99秒~第123秒 | 60 |

（3）测试结果

1. 测试时间

本轮测试的时间信息如表4.6.2所示。

表4.6.2 测试时间信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 测试开始时间 | 并发记录时间 | 测试结束时间 | 测试总耗时（秒） |
| 20 | 2019年4月27日15:19:39 | 2019年4月27日15:20:29 | 2019年4月27日15:25:22 | 343 |
| 30 | 2019年4月27日15:55:21 | 2019年4月27日15:56:22 | 2019年4月27日16:03:23 | 482 |
| 40 | 2019年4月27日16:32:23 | 2019年4月27日16:33:25 | 2019年4月27日16:43:08 | 645 |
| 50 | 2019年4月27日17:01:13 | 2019年4月27日17:02:52 | 2019年4月27日17:14:18 | 785 |
| 100 | 2019年4月27日17:23:09 | 2019年4月27日17:24:49 | 2019年4月27日17:48:07 | 1498 |

1. JMeter聚合报告相关数据

本轮测试后的JMeter聚合报告数据如表4.6.3所示。

表4.6.3 JMeter聚合报告

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 请求样本数（次） | 平均响应时间（毫秒） | 吞吐量（次/秒） | 请求失败率（%） |
| 20 | 20000 | 324 | 58.4 | 0 |
| 30 | 30000 | 456 | 62.1 | 0 |
| 40 | 40000 | 590 | 63.9 | 0 |
| 50 | 50000 | 741 | 63.7 | 0 |
| 100 | 100000 | 1409 | 62.7 | 0 |

1. Druid数据源监控数据。

本轮测试的Druid数据源监控数据如图4.6.4所示，数据都已经过处理，都是是期间的，即将测试后的数据减去测试前的数据所得。

表4.6.4 Druid监控数据

|  |  |  |  |
| --- | --- | --- | --- |
| 线程组线程数（个） | URI请求数（次） | URI请求时间和（毫秒） | URI最大并发数（个） |
| 20 | 20000 | 2179437 | 16 |
| 30 | 30000 | 2339148 | 16 |
| 40 | 40000 | 2891824 | 16 |
| 50 | 50000 | 3599199 | 16 |
| 100 | 100000 | 6920121 | 16 |

1. SpringBootAdmin监控数据。

接口压力测试是不同线程组的多次测试，每个线程组都有自己的ECS监控图，图片较多，就不再一一展示，每个线程组的ECS监控数据如表4.6.5所示。

表4.6.5 ECS服务器监控数据

|  |  |  |  |
| --- | --- | --- | --- |
| 线程组线程数（个） | CPU最高使用率（%） | 网络带宽最高使用-入方向（KB/S） | 网络带宽最高使用-出方向（KB/S） |
| 20 | 100 | 500以下 | 4096以下 |
| 30 | 同上 | 同上 | 同上 |
| 40 | 同上 | 同上 | 同上 |
| 50 | 同上 | 同上 | 同上 |
| 100 | 同上 | 同上 | 同上 |

（4）性能分析

1. 测试耗时分析

测试耗时随着线程组线程数的增多而增多，但最高耗时的100线程，平均每次访问耗时也才14.98毫秒，说明被测接口响应速度很快。

1. JMeter聚合报告数据分析

平均响应时间随线程组线程数的增多而增大，而吞吐量却在线程组线程数为40时达到最高，最高吞吐量为63.9次/秒，线程数再增加吞吐量会下降，说明被测接口存在并发访问性能瓶颈，可承受的并发量不多。

1. Druid监控数据分析

URI请求耗时随线程组线程数的增大而增大，但URI最大并发数却一直是16，说明被测接口存在并发性能瓶颈。

1. ECS服务器监控数据
   1. 当线程组线程数为20时，CPU的使用率就已达到极限的100%。线程数再增加时，因CPU无法再提供更多的处理能力，从而导致被测接口的平均响应时间越来越长，说明CPU限制了系统的并发能力。
   2. 测试过程中，服务器的网络带宽使用都不高，入方向在500KB/S以下，出方向在4MB/S以下，而服务器出、入方向的最高带宽都为15MB/S，所以服务器的网络带宽对测试的影响不大。

（5）测试总结

1. 被测接口响应速度很快，但随着访问线程数的增多，响应速度会降低。
2. URI请求耗时随线程组线程数的增大而增大，但服务器的URI最大并发量只能到16，不再增加。
3. 各线程组在测试过程中，CPU的使用率都到达了100%，CPU资源已被用尽。

### 4.6.5 优化前后接口性能对比

本轮测试是接口并发访问压力测试，注重的是被测接口的平均响应时间和吞吐量，当平均响应时间在可承受范围内时，吞吐量越高越好。

本轮测试是阶段二的压力测试，理论上是不能与阶段一的测试做对比的，但为了突出容器优化与带宽升级后对被测接口的性能提升，此处便选取线程数为20的线程组的测试数据，来与优化前线程组线程数为10的测试数据做对比，相关数据如表4.6.6所示。

表4.6.6 优化前后性能对比

|  |  |  |  |
| --- | --- | --- | --- |
| 属性（单位）/阶段 | 优化前 | 优化后 | 提升（倍） |
| JMeter平均响应耗时（毫秒） | 367 | 324 | 0.133 |
| 吞吐量（次/秒） | 24.6 | 58.4 | 1.37 |

由表4.6.6的数据可知，经过带宽升级优化后，被测接口的响应速度虽然提升不明显，但其吞吐量却得到了很大提升，进而提高了被测接口的整体性能。

## 4.7 系统集群优化

### 4.7.1 优化背景

由第4.6.4节的接口性能测试可知，在进行容器优化与带宽升级后，被测接口的吞吐量虽然有了较大提升，但因服务器CPU的影响，吞吐量还是没有达到100以上的高性能需求，所以还需要继续对被测接口进行优化。

### 4.7.2 优化方案

（1）问题分析

由第4.6.4节的性能分析可知，当前被测接口的性能瓶颈在服务器的CPU上，要想继续提高被测接口的并发访问性能，就需要对服务器的CPU进行升级。

（2）优化方向分析

针对CPU的升级，此处有两个解决方案：

* + 1. 方案一：提高服务器的资源配置。如将服务器的配置从1CPU+2GB内存提升到2个CPU+4GB内存或者4个CPU+8GB内存。
    2. 方案二：使用多台服务器进行集群。既然一台服务器的资源被用尽，无法解决问题，那就提升到3台、5台等等，让多台服务器一起工作。

简单来说，方案一无疑是最省事的，只需要将服务器的配置升级即可，而方案二还要涉及到系统集群的各种问题。不过在实际应用中，方案一也存在很多问题，并不是想象的那么简单，主要问题如下：

1. 高配置服务器价格昂贵。配置越高的服务器往往价格越贵，当服务器配置到达一定程度时，想要继续提升性能，那价格将呈指数级增长。
2. 受硬件极限的制约。排除价格原因，单服务器的性能始终是受硬件制约的，服务器的性能不可能想提升就提升，当服务器的配置已经达到了目前硬件所能支持的极限时，就无法再使用方案一解决了，最终还是要走集群的路线。
3. 单台服务器可靠性太差。过度依赖单台服务器，会导致系统的可靠性很差，一旦服务器因某种不可抗力因素而宕机，比如服务器所在地出现地震、海啸或是大规模停电等，那整个系统将直接瘫痪，无法再对用户提供服务。

综上所述，本轮优化不再是简单的对服务器CPU进行升级，而是用处于不同物理区域的多台服务器来进行系统集群。

（3）优化操作

1. 集群介绍

集群是将多个相互独立的服务器通过高速网络互联，构成成一个服务器组，并以单一系统的模式加以管理，当一个客户与集群相互作用时，集群就像是一个独立的服务器一样为客户提供服务，集群目的是让多台服务器共同为系统工作，以较低的成本来提高系统的性能、可靠性和可扩展性，集群的核心是其任务调度技术[21]。

1. 负载均衡

为了能让集群的多个服务器共同为系统工作，需要一个统一的任务调度器，以将请求任务分发到不同的服务器中进行处理。本轮优化选用的是既可作为Web服务器，也可作为负载均衡服务器，还有使用简单、性能高效等特性的Nginx[22]。

1. 集群时需要考虑的问题
   1. Session共享问题。因系统存在权限控制功能，可能会出现同一用户的请求被分配到不同服务器处理时，需要在多个服务器都进行登录授权的问题。
   2. 缓存数据一致性问题。因系统使用的是Ehcache作为本地缓存，所以集群时可能会出现一台服务器有缓存数据，而另一台服务器却没有，这就会导致同一用户多次请求同一URI，却得到不同响应结果的缓存数据不一致问题。
2. 优化操作说明
   1. 添加两台配置与当前服务器配置相同，但所处物理区域不同的服务器，具体信息如表4.7.1所示。

表4.7.1 集群服务器信息

|  |  |  |  |
| --- | --- | --- | --- |
| 属性/服务器 | 服务器1 | 服务器2 | 服务器3 |
| 服务器名称 | 阿里云ECS服务器 | 阿里云ECS服务器 | 阿里云ECS服务器 |
| IP地址 | 120.78.59.162 | 47.101.189.171 | 47.101.214.48 |
| 所处物理区域 | 华南1（深圳） | 华东2（上海） | 华东2（上海） |
| 运行系统 | CentOS 7.3 64位 | CentOS 7.3 64位 | CentOS 7.3 64位 |
| CPU | 1 vCPU | 1 vCPU | 1 vCPU |
| 运行内存 | 2GB(IO优化) | 2GB(IO优化) | 2GB(IO优化) |
| 网络带宽 | 15 MBbps | 15 MBbps | 15 MBbps |
| JDK | 1.8.0\_151 | 1.8.0\_151 | 1.8.0\_151 |
| JVM | Java HotSpot(TM) 64-Bit Server VM | Java HotSpot(TM) 64-Bit Server VM | Java HotSpot(TM) 64-Bit Server VM |

* 1. 更改系统的架构、设计、技术和业务代码，使其符合系统集群的需求。如引入SpringSession来处理系统集群时的Session共享问题，用SpringCahce+Redis的分布式缓存方案替换Mybatis+Ehcahe的本地缓存方案，以此解决缓存数据不一致的问题。为了保证Redis的高可用，可通过Redis哨兵集群来增强Redis缓存节点的可靠性[23]。
  2. 使用Nginx服务器作为系统集群的负载均衡解决方案。

1. 集群后各服务器节点信息如表4.7.2所示。

表4.7.2 集群服务器信息

|  |  |  |  |
| --- | --- | --- | --- |
| 服务端口/服务器 | 服务器1：120.78.59.162 | 服务器2：47.101.189.171 | 服务器3：47.101.214.48 |
| BootAdmin监控端口 | 9999 | \ | \ |
| Nginx端口 | 8080 | \ | \ |
| Docker-Mysql端口 | 3308->3306 | \ | \ |
| Redis节点端口 | 6379（master） | 6379 | 6379 |
| Redis哨兵端口 | 26379 | 26379 | 26379 |
| 系统应用端口 | 8888 | 8888 | 8888 |

1. Redis哨兵集群信息

redis哨兵集群信息如图4.7.1所示。

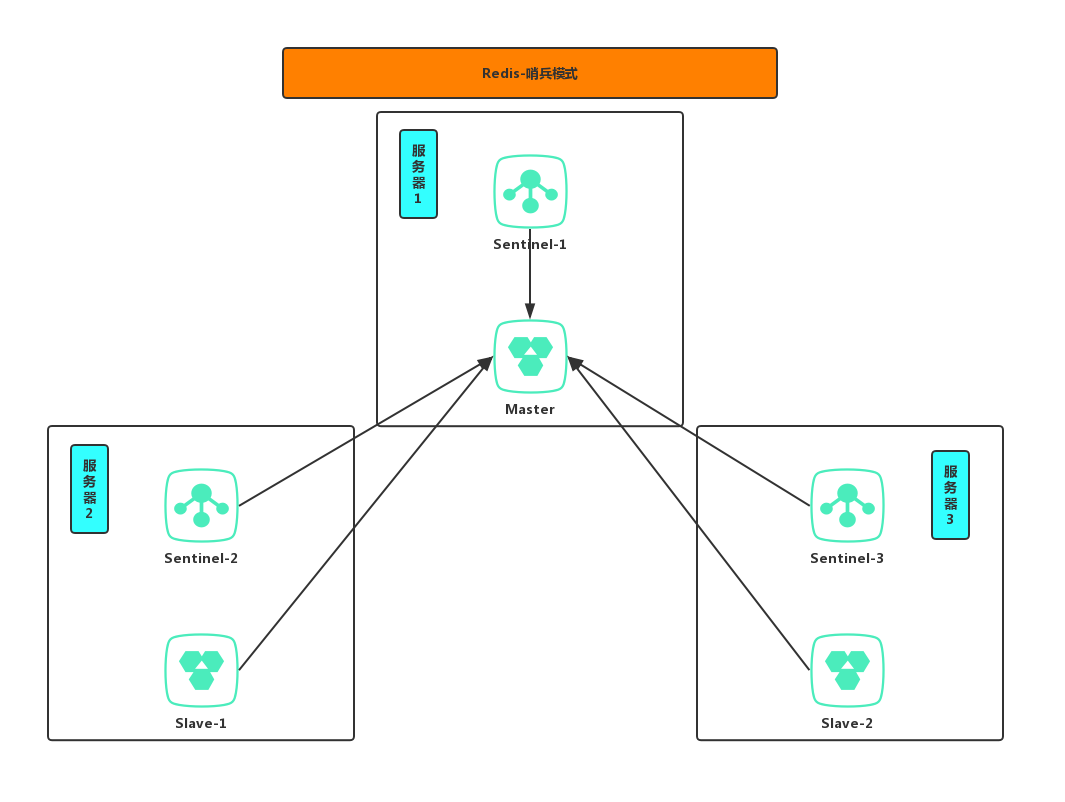


图4.7.1 Redis哨兵集群

1. Nginx负载均衡配置信息

由表4.7.2可知，服务器1又要跑监控，又要跑数据库，还要跑Nginx，压力比较大，所以本人对 Nginx的负载均衡权重配置如下，希望测试期间由服务器2和服务器3来承受大部分请求，以此来减轻服务1的测试压力，具体如图4.7.2所示。

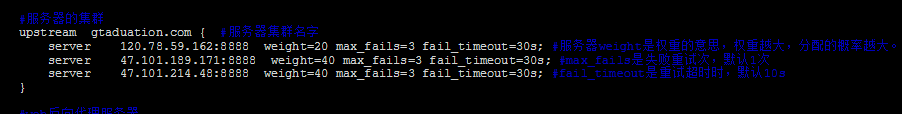


图4.7.2 Nginx负载均衡权重配置

### 4.7.3 优化后的系统变化

（1）系统架构变化

集群后的系统架构如图4.7.3所示。

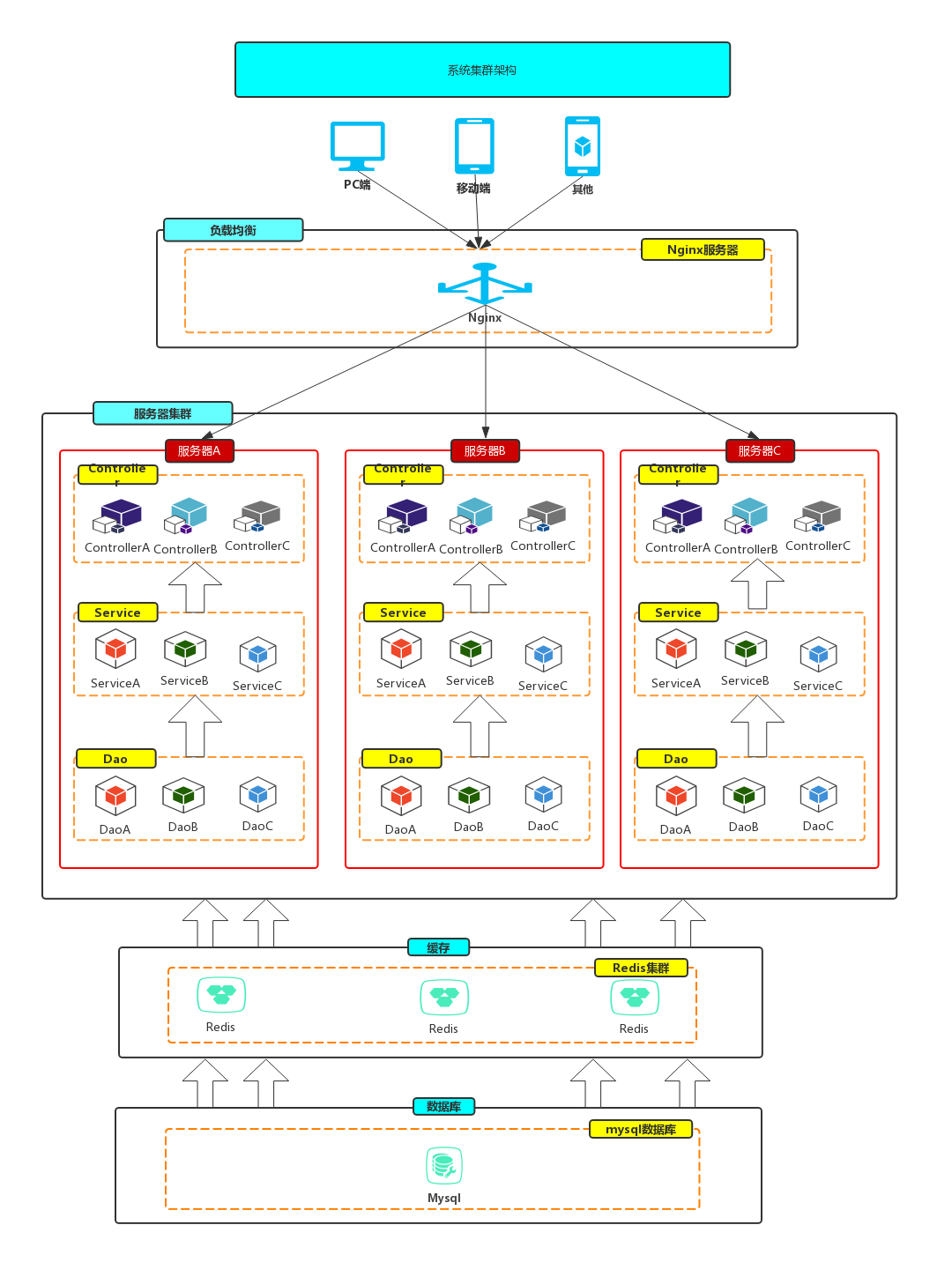


图4.7.3 系统集群架构

（2）技术变化

系统集群后技术变化有点多，这里就将系统集群后的相关技术全部展示出来更加直观。

负载均衡：Nginx

Servlet容器：Undertow

权限控制：SpringSecurity

Session管理：SpringSession

缓存管理：SpringCahce、Redis-Lettuce

数据库：Docker-MySql

前端：Html/CSS、JavaScript、JQuery、Bootstrap、Layui、WangEditor等。

后端：SpringBoot、SpringMVC、Mybatis、Kaptcha、Lombok等。

前后端交互：Ajax、Json

缓存：Redis哨兵模式

监控：SpringBootAdmin、Druid数据源、阿里云ECS监控

（3）其他变化

无。

### 4.7.4 优化后接口压力测试

（1）测试信息

本轮测试是集群并发压力测试，测试入口是Nginx，即120.78.59.162:8080，所以除了测试端口变为8080外，其他测试信息均与第4.6.4节保持一致。

（2）测试计划

根据4.2节中所述的高并发压测原理及方法，预先使用JMeter（5线程，循环1000次）并发访问被测的新闻列表接口，获取到的平均响应时间为232毫秒。因本轮测试是接口压力测试的后续，所以进行阶段二的测试。根据平均响应时间和JMeter并发测试理论，为本轮测试配置JMeter相关参数，具体如表4.7.3所示。

表4.7.3 JMeter测试配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 所需最低循环次数（次） | 实际设置循环次数（次） | 并发时间段 | 请求/响应超时限制（秒） |
| 50 | 212 | 1000 | 第49秒~第232秒 | 60 |
| 100 | 427 | 1000 | 第99秒~第232秒 | 60 |
| 150 | 643 | 1000 | 第149秒~第232秒 | 60 |
| 200 | 858 | 1000 | 第199秒~第232秒 | 60 |
| 300 | 1289 | 2000 | 第299秒~第464秒 | 60 |

（3）测试结果

1. 测试时间

本轮测试的时间信息如表4.7.4所示。

表4.7.4 测试时间信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 测试开始时间 | 并发记录时间 | 测试结束时间 | 测试总耗时（秒） |
| 50 | 2019年5月5日13:18:41 | 2019年5月5日13:19:45 | 2019年5月5日13:25:02 | 381 |
| 100 | 2019年5月5日14:11:02 | 2019年5月5日14:13:12 | 2019年5月5日14:22:27 | 685 |

表4.7.4 （续）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 150 | 2019年5月5日14:33:25 | 2019年5月5日14:36:14 | 2019年5月5日14:49:52 | 987 |
| 200 | 2019年5月5日15:02:17 | 2019年5月5日15:05:40 | 2019年5月5日15:24:41 | 1344 |
| 300 | 2019年5月5日15:45:37 | 2019年5月5日15:50:43 | 2019年5月5日16:49:57 | 3860 |

1. JMeter聚合报告相关数据

本轮测试后的JMeter聚合报告数据如表4.7.5所示。

表4.7.5 JMeter聚合报告

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 请求样本数（次） | 平均响应时间（毫秒） | 吞吐量（次/秒） | 请求失败率（%） |
| 50 | 50000 | 328 | 131.5 | 0 |
| 100 | 100000 | 590 | 146.1 | 0 |
| 150 | 150000 | 844 | 152.0 | 0 |
| 200 | 200000 | 1142 | 148.7 | 0 |
| 300 | 600000 | 1769 | 145.1 | 0 |

1. Druid数据源监控数据。

本轮测试的Druid数据源监控数据如图4.7.6所示，数据都已经过处理，都是是期间的，即将测试后的数据减去测试前的数据所得。

表4.7.6 Druid监控数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 服务器 | URI请求数（次） | URI请求时间和（毫秒） | URI最大并发数（个） |
| 50 | 服务器1 | 10000 | 105257 | 5 |
| 50 | 服务器2 | 20000 | 4560471 | 16 |
| 50 | 服务器3 | 20000 | 5338819 | 16 |
| 100 | 服务器1 | 20000 | 187584 | 4 |
| 100 | 服务器2 | 39999 | 9632677 | 16 |
| 100 | 服务器3 | 40001 | 10290647 | 16 |
| 150 | 服务器1 | 30000 | 245926 | 5 |
| 150 | 服务器2 | 60000 | 13358937 | 16 |
| 150 | 服务器3 | 60000 | 15132666 | 16 |
| 200 | 服务器1 | 40000 | 335193 | 4 |
| 200 | 服务器2 | 80002 | 18081878 | 16 |
| 200 | 服务器3 | 79998 | 20746349 | 16 |
| 300 | 服务器1 | 120000 | 1058805 | 6 |
| 300 | 服务器2 | 239998 | 60548308 | 16 |
| 300 | 服务器3 | 240002 | 53628115 | 16 |

1. SpringBootAdmin监控数据。

接口压力测试是不同线程组的多次测试，每个线程组都有自己的ECS监控图，图片较多，就不再一一展示，每个线程组的ECS监控数据如表4.7.7所示。

表4.7.7 ECS服务器监控数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程组线程数（个） | 服务器 | CPU最高使用率（%） | 网络带宽最高使用-入方向（MB/S） | 网络带宽最高使用-出方向（MB/S） |
| 50 | 服务器1 | 50 以下 | 10以下 | 接近15 |
| 50 | 服务器2 | 同上 | 5以下 | 5以下 |
| 50 | 服务器3 | 同上 | 5以下 | 5以下 |
| 100 | 服务器1 | 同上 | 10以下 | 接近15 |
| 100 | 服务器2 | 同上 | 5以下 | 5以下 |
| 100 | 服务器3 | 同上 | 5以下 | 5以下 |
| 150 | 服务器1 | 同上 | 10以下 | 接近15 |
| 150 | 服务器2 | 同上 | 5以下 | 5以下 |
| 150 | 服务器3 | 同上 | 5以下 | 5以下 |
| 200 | 服务器1 | 同上 | 10以下 | 接近15 |
| 200 | 服务器2 | 同上 | 5以下 | 5以下 |
| 200 | 服务器3 | 同上 | 5以下 | 5以下 |
| 300 | 服务器1 | 同上 | 10以下 | 接近15 |
| 300 | 服务器2 | 同上 | 5以下 | 5以下 |
| 300 | 服务器3 | 同上 | 5以下 | 5以下 |

（4）性能分析

1. 测试耗时分析

测试耗时随着线程组线程数的增多而增多，但最高耗时的300线程，平均每次访问耗时也才6.43毫秒，速度非常快。

1. JMeter聚合报告数据分析

平均响应时间随线程组线程数的增多而增大，而吞吐量在线程组线程数为150时达到最高，最高吞吐量为152次/秒，线程数再增加吞吐量会下降，当线程组线程数达到300时，被测接口的平均响应时间超过1秒，说明被测接口存在并发访问性能瓶颈，影响系统并发承载量。

1. Druid监控数据分析

URI请求耗时随线程组线程数的增大而增大，但在各个服务器中，URI的最大并发数却一直是16，无法再提升，说明被测接口还存在并发访问性能瓶颈。

1. ECS服务器监控数据
   1. 测试过程中，各服务器的CPU使用率都在50%以下，对测试影响不大。
   2. 测试过程中，服务器2与服务器3的网络带宽使用都不高，但因Nginx部署在服务器1上，所有请求的进出都要经过服务器1，所以服务器1的网络带宽使用较高，入方向还好，但出方向已接近极限的15MB/S，可能会成为性能瓶颈。

（5）测试总结

* + 1. 被测接口响应速度很快，但随着访问线程数的增多，响应速度会降低。
    2. URI请求耗时随线程组线程数的增大而增大，但各个服务器的URI最大并发量只能到16，不再增加。
    3. 各线程组在测试过程中，各服务器的CPU使用没有问题，但服务器1的网络带宽使用较高，接近了极限。

### 4.7.5 优化前后接口性能对比

本轮是系统集群优化，目的是提高系统的性能、吞吐量和稳定性，所以此处的性能对比便只对比被测接口优化前后的最高吞吐量，及达到最高吞吐量时被测接口的平均响应时间，具体数据如表4.7.8所示。

表4.7.8优化前后性能对比

|  |  |  |  |
| --- | --- | --- | --- |
| 属性（单位）/阶段 | 优化前 | 优化后 | 提升（倍） |
| 最高吞吐量（次/秒） | 63.9 | 152.0 | 1.38 |
| 最高吞吐量对应的平均响应耗时（毫秒） | 590 | 844 | -0.43 |

结合表4.7.8的数据和集群测试过程可知，系统集群后，请求与响应都需要经过Nginx进行分发，涉及到网络传输问题，所以集群时的最高吞吐量对应的平均响应时间降低了，但还在可接受范围之内，且集群对被测接口吞吐量的提升十分明显，集群后的服务器跨区域分布，使得系统的可靠性大大增强。

## 4.8 优化总结

（1）总结背景

由4.7.4的压力测试可知，新闻列表接口的吞吐量已达到了152，且当线程数为300时，接口的平均响应时间还在2秒以内，因服务器跨区域分布的原因，系统的可靠性也得到了大幅加强，整体已满足了需求分析中的所有要求，故不再继续进行优化。

（2）初始系统与集群系统性能对比

经过对被测接口的一次次测试与优化，使得优化后的被测接口的整体性能得到了非常大的提升，通过对优化前后的JMeter测试数据进行对比，得到具体形态提升数据如表4.8.1所示。

表4.8.1 优化前后性能对比

|  |  |  |  |
| --- | --- | --- | --- |
| 属性（单位）/阶段 | 优化前 | 优化后 | 提升（倍） |
| JMeter平均响应耗时（毫秒） | 466280 | 844 | 551.46 |
| 吞吐量（次/秒） | 0.02 | 152.0 | 7599.00 |

（3）优化总结

结合表4.8.1的数据与各个优化阶段的测试和性能分析可知，优化后被测接口的响应速度、吞吐量和可靠性都得到了大幅提升，其中响应速度提升了551.46倍、吞吐量提高了7599倍！使得被测接口在面对300线程的并发访问时，平均响应时间还能保持在2秒以内，说明对系统的优化非常成功。

# 第五章 总结与展望

## 5.1 总结

论文通过对内容管理系统各个模块进行分析、设计、实现和部署后。通过测试发现，只是满足功能需求的系统还存在着各种各样的非功能性问题，导致系统整体性能很差，无法满足需求，必须对系统进行优化。由此可知，一个系统应用在被开发出来后，即便是功能齐全，也不一定是可用的，特别是当系统应用面向的用户较多，并发量较大时，就必须对系统进行相关测试与优化，以保证系统性能。

本论文以内容管理系统中的新闻列表接口作为示列，通过对接口的一次次并发访问测试，不仅找出了JVM、数据库结构、SQL语句等，严重影响着接口性能的系统内部因素，还找出了服务器网络带宽、服务器CPU等，严重制约着接口吞吐量的外部因素，接着以这些因素为核心，结合各个因素所需的优化知识，为每一个影响接口性能的因素制定对应的优化方案，通过对接口的不断优化，使得优化后的接口的整体性能得到了很大提升，即便面对300线程的并发访问，接口的响应速度还能维持在2秒以内，满足了基本的高并发访问性能需求。

虽然优化后的被测接口的性能提升很大，但服务器那1CPU+2GB内存的配置终究还是太差了，即便是3台服务器集群，最高吞吐量也才有152，当并发访问量在300以上时，被测接口的响应时间也许就不能接受了。

被服务器硬件配置所限制，使得优化后的系统的整体性能没能得到更好的体现，这终究是一大遗憾，但高配置服务器的价格对于还是学生的本人来说，终究还是太贵了，实在无力支撑，所以只能抱着遗憾继续前行，毕竟优化之路道阻且长，以后接触的机会还有很多，本文所展现的只是系统优化相关知识的冰山一角罢了。

## 5.2 展望

通过对被测接口进行优化后，系统的整体架构从简单的MVC三层架构演变到了拥有负载均衡与缓存的集群架构，使得系统的整体性能得到了很大提升。虽然这种集群架构也常见于互联网公司的实际生成环境中，但对于现阶段用户量越来越大、业务越来越复杂，且变化越来越快的互联网行业来说，集群架构存在着诸多不便，如系统太过臃肿与庞大、灵活性不高、不易于扩展与维护等。

相比于将系统的所有功能都集成在一起应用，然后再进行集群的集群架构，现今的大型互联网公司更倾向于将系统中的各个功能模块分离出去，形成一个个微小的服务，之后每个服务再根据自身的需求进行集群的微服务架构。

微服务架构是一种架构模式或者说是一种架构风格，它提倡将单一的应用划分为一组小的服务，每个服务运行在自己独立的进程中，服务与服务之间通过轻量级的通信机制（通常是基于HTTP的 RESTful API）相互通信、相互协调、相互配合，为用户提供最终的价值[24]。服务可以由不同的语言开发，可以单独启动或销毁，甚至可以有自己独立的数据库[24]。微服务架构还有着高内聚、低耦合、开发简单快速、非常灵活、易于扩展、易维护等特点[24]，比传统的集群架构更适用于当前高速发展的互联网行业，所以当系统还需继续优化时，可将系统架构朝着微服务方向去发展，微服务的大体架构如图5.2.1所示。

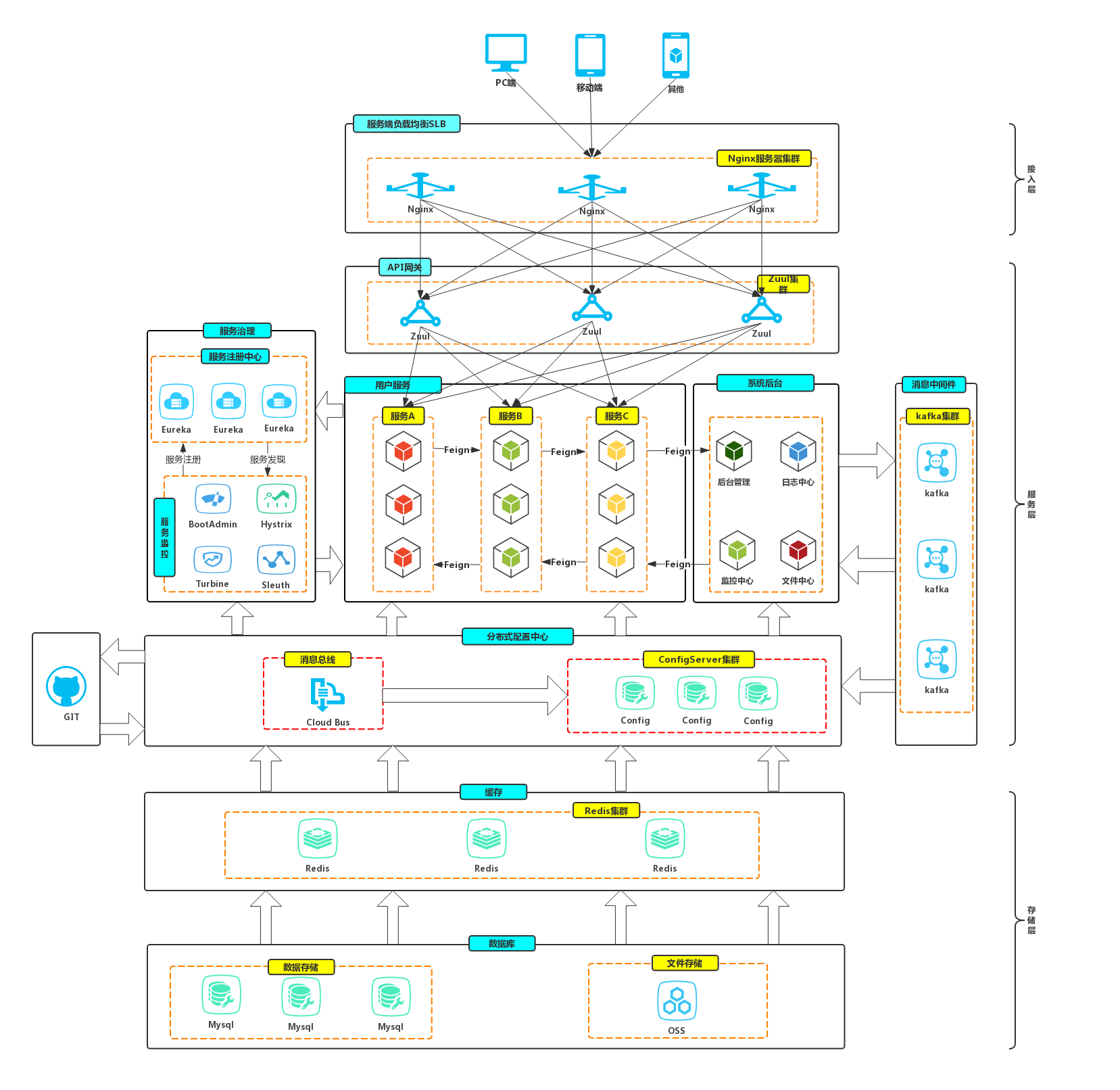


图5.2.1 微服务架构

# 参考文献

[1] 范梓萌. 中国网民规模达8.29亿[N]. 人民日报海外版，2019-05-07.

[2] Jesse James Garrett. 用户体验的要素[M]. 第二版，北京：机械工业出版社，2008. 2～17.

[3] SpringBoot官方文档[DB/OL]，https://spring.io/projects/spring-boot/.

[4] SpringSecurity官方文档[DB/OL]，https://spring.io/projects/spring-security.

[5] Mybatis官方文档[DB/OL]，http://www.mybatis.org/mybatis-3/zh/index.html.

[6] Nginx官方文档[DB/OL]，http://nginx.org/en/.

[7] Redis官方文档[DB/OL]，http://www.redis.cn/.

[8] JMeter官方文档[DB/OL]，https://jmeter.apache.org/.

[9] 美团技术博客. 常见性能优化策略的总结[DB/OL]，https://tech.meituan.com/2016/12/02/performance-tunning.html.

[10] JMeter User's Manual[DB/OL]， http://jakarta.apache.org/jmeter/usermanual/index.html.

[11] 江新. 基于JMeter的MSWeb应用系统的性能测试研究[D]. 南京：南京航空航天大学，2011

[12] JMeter Elements of a Test Plan[DB/OL]，https://jmeter.apache.org/usermanual/test\_plan.html.

[13] SpringBootAdmin官方文档[DB/OL]，http://codecentric.github.io/spring-boot-admin/2.0.3/.

[14] Druid官方文档[DB/OL]，https://github.com/alibaba/druid/wiki/%E5%B8%B8%E8%A7%81%E9%97%AE%E9%A2%98.

[15] Baron Schwartz，Peter Zaitsev，Vadim Tkachenko，等. 高性能MySQL[M]. 第3版，北京：电子工业出版社，2013.

[16] apio Lahdenmaki，Michael Leach，曹怡倩，等. 数据库索引设计与优化[M]. 北京：电子工业出版社，2015.

[17] 周志明. 深入理解Java虚拟机[M]. 第2版，北京：机械工业出版社，2013.

[18] Ehcahe官方文档[DB/OL]. http://www.ehcache.org/documentation/.

[19] mybatis-ehcache官方文档[DB/OL]. http://www.mybatis.org/ehcache-cache/.

[20] Andy Beck. Tomcat vs Jetty vs Undertow: Comparison of Spring Boot Embedded Servlet Containers[DB/OL]，https://examples.javacodegeeks.com/enterprise-java/spring/tomcat-vs-jetty-vs-undertow-comparison-of-spring-boot-embedded-servlet-containers/.

[21] 卢军. 高性能网络存储技术研究[D]. 成都：电子科技大学，2002.

[22] 陶辉. 深入理解Nginx[M]. 第2版， 北京：机械工业出版社，2016.

[23] Josiah L. Carlson，黄健宏. Redis实战[M]. 北京：人民邮电出版社，2015.

[24] Martin Fowler. Microservices[DB/OL]. https://martinfowler.com/articles/microservices.html.

# 致谢

时光飞逝，转眼间四年的大学生活便已到了落幕的季节，回首四年之间，有辛酸亦有收获，艰苦之中学到了很多的技术与知识，为人生的下一段征程做好了铺垫，使得前路更加光明。

在校园生活的最后之际，能做此研究乃是本人的一大幸事，本次研究不仅总结了自己的过往所学，还使自己闯入了许多以往不曾考虑的知识盲区，在李晖老师的辛勤指导之下，以需求为驱动，以优化为目标，一路上磕磕碰碰，解决掉了许多难题，填补了自己技术路线上的诸多空白之处，收获非常之大。

首先，本人要感谢一路陪伴，对本人又是指引又是指导的李晖老师。老师为人亲和、学识渊博，为本次论文研究提供了非常大的帮助，感谢他在本人论文研究期间所给予的实验资料和技术指导，感谢他的一路陪伴与付出，这才让本人的论文研究得以完结。

其次，本人要感谢母校贵州大学。拥有110多年历史的贵大人文气息浓厚，学习氛围良好，让本人在校期间学到了很多技术与知识，为本次论文研究提供了很多技术积累。

再来，本人要感谢四年以来，共同艰苦学习的老师与同学们，因为有了他们的陪伴，本人的求学之路才充满了活力与动力。

最后，本人还需要感谢“开源”思想以及互联网上那众多心怀“开源”观念的程序员们，因为“开源”，本人才能学到那么多的知识与技术，因为“开源”，本人也才能见识到如此广阔的技术世界！

杨 毅

2019年6月3日