**谨以此论文献给我的导师和亲人！**

—————————— **武 斌**

**基于** Spring MVC **架构** WEB **应用的系统优化策略研究**

学位论⽂答辩⽇期： 2017 年 05 ⽉ 28 ⽇

指导教师签字：

答辩委员会成员签字：

**独 创 声 明**

本⼈声明所呈交的学位论⽂是本⼈在导师指导下进⾏的研究⼯作及取得的研究成果。据我所知， 除了⽂中特别加以标注和致谢的地⽅外， 论⽂中不包含其他⼈已经发表或撰写过的研究成果， 也不包含未获得 （注： 如没有其他需要特别声明的，本栏可空）或其他教育机构的学位或证书使⽤过的材料。与我⼀同⼯作的同志对本研究所做的任何贡献均已在论⽂中作了明确的说明并表⽰谢意。

学位论⽂作者签名： 签字⽇期： 年 ⽉ ⽇

**学位论文版权使用授权书**

本学位论⽂作者完全了解学校有关保留、使⽤学位论⽂的规定，并同意以下事项：

1. 学校有权保留并向国家有关部门或机构送交论⽂的复印件和磁盘，允许论⽂被查阅和借阅。
2. 学校可以将学位论⽂的全部或部分内容编⼊有关数据库进⾏检索，可以采⽤影印、缩印或扫描等复制⼿段保存、汇编学位论⽂。同时授权清华⼤学“中国学术期刊(光盘版) 电⼦杂志社”⽤于出版和编⼊CNKI《中国知识资源总库》， 授权中国科学技术信息研究所将本学位论⽂收录到《中国学位论⽂全⽂数据库》。

（保密的学位论⽂在解密后适⽤本授权书）

学位论⽂作者签名： 导师签字：

签字⽇期： 年 ⽉ ⽇ 签字⽇期： 年 ⽉ ⽇

**基于** Spring MVC **架构** WEB **应用的系统优化策略研究摘 要**

近⼏年，随着中国经济的不断发展以及中国互联⽹整体⽔平的不断提⾼，虚拟经济在中国的经济发展中扮演着越来越重要的作⽤，WEB 应⽤作为虚拟经济的主要体现⽅式也获得了⾮常⼤的发展。越来越多⾯向不同消费群体的 WEB 应⽤被设计和开发出来为消费者提供各种的服务。

随着 Internet ⽹的不断告诉发展，⼈们对于 WEB 应⽤的需要不再基于可⽤性那么简单，应⽤的⽤户体验越来越被重视[[1](#_bookmark86)]。然⽽，随着⽤户数量的不断增加，

WEB 服务器的信息量和访问量成⼏何数级的增长，⽹络拥塞和服务超载⽇益成为开发⼈员必须⾯对的严峻问题。

因此在 WEB 应⽤的开发过程中，开发⼈员不但需要在代码的书写上要进⾏不断优化，降低代码的错误率和提升代码的执⾏效率外，⽽且要将很⼤的精⼒放到服务器性能优化⽅⾯，通过调整 WEB 服务器的运⾏参数提升服务器的响应效率、优化缓存机制提升数据的存取效率，设计负载均衡⽅案实现应⽤服务器的负载均衡，开发数据库的主从复制和延时复制等策略保证数据的稳定性以及编写监控脚本实现服务的实施监控等措施来提升应⽤的体验和增强系统的安全性。

本⽂的创新点主要有：

* 通过 Jenkins 实现 WEB 应⽤的⾃动构建和部署，提升了 WEB 应⽤的稳定性， 降低代码错误率；
* 基于Couchbase 缓存机制，加快了应⽤的数据读取，降低了系统的响应时间；
* 通过Docker 容器编排技术，⽅便的实现服务集群节点的扩展，增强了系统的安全性，实现了统⼀管理和维护；
* 通过负载均衡，降低了应⽤服务器和数据服务器的压⼒，提升了⽤户的体验；
* 充分利⽤API 实现监控脚本，实现了应⽤的实时监控，很⼤程度上减少了发现问题和故障处理的时间；

**关键词：** Spring MVC **框架应用开发**; WEB **应用性能优化**; **数据库性能优化**; **服务监控及应对方案**;

Research on Web Platform System Optimization Strategy Based on Spring MVC Framework

Abstract

In recent years, with the continuous development of China’s economy and the con- tinuous improvement of the overall level of China’s Internet, virtual economy plays an increasingly important role in China’s economic development, WEB application as a vir- tual economy is also the main presentation of the very Big development. More and more for diﬀerent consumer groups of WEB applications are designed and developed to provide consumers with a variety of services.

With the continuous development of the Internet to tell the people, the need for WEB applications is no longer based on usability so simple, application user experience more and more attention. However, with the increase of the number of users, the amount of information and the number of visits of the WEB server grows geometrically. Network congestion and service overload are increasingly becoming the serious problem that the developers must face.

Therefore, in the WEB application development process, developers need to not only write code on the continuous optimization to reduce the code error rate and improve the eﬃciency of the code, but also to a lot of energy into the server performance optimization, By adjusting the operating parameters of the WEB server to improve the response eﬃ- ciency of the server and optimize the cache mechanism to improve the eﬃciency of data access, the design load balancing solution to achieve the application server load balancing, database replication master and slave replication and delay replication strategy to ensure data stability As well as the preparation of monitoring scripts to achieve the implementa- tion of service monitoring and other measures to enhance the application experience and enhance the security of the system.

The innovation of this paper：

* Through Jenkins to realize the automatic construction and deployment of WEB applications to enhance the stability and reduce code error rate；
* Through Couchbase cache mechanism to speed up the data read of application, reducing the system response time；
* Through Docker container scheduling technology to facilitate the expansion of service cluster nodes to enhance the security of the system to achieve a unified

management and maintenance；

* Reduces stress on application servers and data servers through load balancing and improves user experience；
* Make full use of API to achieve the monitoring script to achieve the application of real-time monitoring, to a large extent reduce the discovery of problems and troubleshooting time；

### Keywords: WEB Aplication Development Based on Spring MVC; WEB Application Performance Optimization; Database Performance Optimization; Service Monitor- ing and Response Strategies

基于 Spring MVC 架构的 WEB 应⽤的系统优化策略研究

# 目 录

[1 绪论](#_bookmark0) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [1](#_bookmark0)

[1.1 论⽂研究背景及意义](#_bookmark1)................................................................. [1](#_bookmark1)

[1.2 国内外研究现状](#_bookmark2) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [1](#_bookmark2)

[1.3 论⽂主要研究内容](#_bookmark3).................................................................... [2](#_bookmark3)

[1.3.1 论⽂主要⼯作内容](#_bookmark4) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [2](#_bookmark4)

[1.3.2 论⽂⽬标](#_bookmark5) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [3](#_bookmark5)

[1.4 论⽂组织结构](#_bookmark6) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [3](#_bookmark6)

1. [WEB 应⽤开发及持续集成](#_bookmark7) [5](#_bookmark7)
   1. [Spring MVC 开发框架](#_bookmark8) [5](#_bookmark8)
      1. [Spring MVC 框架处理流程](#_bookmark9) [5](#_bookmark9)
      2. [Spring MVC 体系的三层架构](#_bookmark11) [6](#_bookmark11)

[2.2 应⽤开发⼯具](#_bookmark12) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [6](#_bookmark12)

* + 1. [IntelliJ IDEA](#_bookmark13) [7](#_bookmark13)
    2. [Maven](#_bookmark14) [7](#_bookmark14)
    3. [Tomcat](#_bookmark15) [7](#_bookmark15)
    4. [MySQL](#_bookmark16) [7](#_bookmark16)
    5. [Couchbase](#_bookmark17) [8](#_bookmark17)

[2.3 应⽤开发流程](#_bookmark18) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [8](#_bookmark18)

[2.3.1 前端开发](#_bookmark19) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [8](#_bookmark19)

[2.3.2 后端开发](#_bookmark20) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [9](#_bookmark20)

[2.3.3 数据库搭建](#_bookmark21) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [9](#_bookmark21)

[2.4 基于 Jenkins 的持续集成⽅案开发](#_bookmark22) [9](#_bookmark22)

[2.4.1 Jenkins 软件安装配置](#_bookmark23) [9](#_bookmark23)

[2.4.2 ⾃动构建及检查⽅案设计](#_bookmark27)........................................................ [11](#_bookmark27)

[2.4.3 ⾃动部署⽅案设计](#_bookmark29) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [13](#_bookmark29)

[2.5 本章总结](#_bookmark31).............................................................................. [14](#_bookmark31)

[3 应⽤性能优化](#_bookmark32) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [15](#_bookmark32)

* 1. [Couchbase 缓存优化](#_bookmark33) [15](#_bookmark33)
     1. [Couchbase 安装](#_bookmark36) [16](#_bookmark36)
     2. [Couchbase 集群配置](#_bookmark37) [17](#_bookmark37)

iv

* + 1. [Couchbase 缓存对系统性能影响](#_bookmark40) [20](#_bookmark40)
  1. [Tomcat ⾼并发 APR 优化](#_bookmark41) [22](#_bookmark41)
     1. [开启 APR 模式](#_bookmark45) [23](#_bookmark45)
     2. [APR 模式对于系统性能的影响](#_bookmark46) [25](#_bookmark46)
  2. [Docker 分布式优化](#_bookmark48) [25](#_bookmark48)
     1. [使⽤ Docker Compose 管理 Docker 容器](#_bookmark51) [28](#_bookmark51)

[3.3.2 应⽤容器化现状](#_bookmark52)................................................................... [30](#_bookmark52)

[3.4 SLB 负载均衡优化](#_bookmark53) [30](#_bookmark53)

[3.5 本章总结](#_bookmark55).............................................................................. [31](#_bookmark55)

[4 数据优化](#_bookmark56) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [32](#_bookmark56)

[4.1 InnoDB 引擎参数优化](#_bookmark58) [36](#_bookmark58)

[4.2 主从复制和延迟复制优化](#_bookmark59) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [40](#_bookmark59)

[4.2.1 数据库复制流程](#_bookmark61)................................................................... [41](#_bookmark61)

[4.2.2 双主复制设计](#_bookmark63)...................................................................... [42](#_bookmark63)

[4.2.3 延迟复制设计](#_bookmark64)...................................................................... [44](#_bookmark64)

[4.3 数据库备份](#_bookmark65)........................................................................... [44](#_bookmark65)

[4.4 本章总结](#_bookmark66).............................................................................. [47](#_bookmark66)

[5 服务监控与应急措施优化](#_bookmark67) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [48](#_bookmark67)

[5.1 阿⾥云云监控应⽤](#_bookmark68)................................................................... [48](#_bookmark68)

[5.2 ⾃定义服务监控](#_bookmark73) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [51](#_bookmark73)

[5.2.1 ⼼跳监听](#_bookmark74) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [51](#_bookmark74)

[5.2.2 Tomcat 监控](#_bookmark75) [58](#_bookmark75)

[5.2.3 数据库监控](#_bookmark76) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [59](#_bookmark76)

[5.3 短信通知](#_bookmark77).............................................................................. [59](#_bookmark77)

[5.4 ⽇志备份](#_bookmark78).............................................................................. [61](#_bookmark78)

[5.5 本章总结](#_bookmark80).............................................................................. [63](#_bookmark80)

[6 总结与展望](#_bookmark81) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [64](#_bookmark81)

[6.1 总结](#_bookmark82) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [64](#_bookmark82)

[6.2 展望](#_bookmark84) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [65](#_bookmark84)

[附录 A Jenkins ⾃动部署脚本](#_bookmark133) [69](#_bookmark133)

[附录 B Tomcat 健康监控脚本](#_bookmark134) [72](#_bookmark134)

[附录 C Tomcat 故障恢复脚本](#_bookmark135) [75](#_bookmark135)

[附录 D MySQL 健康监控脚本](#_bookmark136) [78](#_bookmark136)

[附录 E MySQL 同步状态监控脚本](#_bookmark137) [83](#_bookmark137)

[附录 F Tomcat ⽇志备份脚本](#_bookmark138) [87](#_bookmark138)

[致 谢](#_bookmark139) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . [92](#_bookmark139)

[个⼈简历、在学期间发表的学术论⽂与研究成果](#_bookmark140) . . . . . . . . . . . . . . . . . . . . . . . . . . [93](#_bookmark140)

# 主要符号对照表

Internet 互联⽹

FDP 快速开发平台 Fast Development Platform

SLB 服务器负载均衡 (Server Load Balancing) Load balancing 负载均衡

IDE 集成开发环境

cluster 集群

RBR 基于⾏的复制 Row Based Replication

SMP 对称多处理

API 应⽤程序编程接⼜

master 主数据库

slave 从数据库

binary log ⼆进制⽇志

relay log 中继⽇志

GTID 全局事物标识 (global transaction identifieds)

OSS 对象存储

OAS 归档存储

Bucket 存储空间

Failover 故障转移

Cloud Monitor 云监控

CDN 内容分发⽹络（Content delivery network）

ECS 云服务器（Elastic Compute Service）

HA ⾼可⽤ (Highly-Available)

POM 项⽬⼯程模型 (project object model)

# 绪论

## 论文研究背景及意义

随着信息技术的不断发展，21 世纪我们已经进⼊了⼀个“信息爆炸”的社会，⽽且在近⼏年我们的社会⼏乎进⼊互联⽹时代后，社会中的每个⼈每天接收到的咨询和信息以及各种各样的互联⽹产品同互联⽹发展的初期相⽐，超出了太多太多。

[[2](#_bookmark87)]。另外，我国的互联⽹技术在近⼗⼏年内不断的发展，开发 WEB 产品的⼈⼒成本和技术成本不断降低，⽤户对于 WEB 应⽤的需求[[3](#_bookmark88)] 不再满⾜于通过缓慢的加载和复杂的操作获取资讯[[4](#_bookmark89)]，许多的创业者希望通过开发⼀个 WEB 应⽤实现⾃⼰创业梦想的时代已经⽆法获得投资⼈的青睐。⽽且，由于近⼏年 WEB 信息泄露的新闻不断进⼊消费者的视野，⽤户对于 WEB 应⽤的安全性体验也提出了更⾼的要求。

随着互联⽹技术进⼀步普及，基于模型-视图-控制器 (Model-View-Controller, MVC) 模式的 WEB 应⽤程序[[5](#_bookmark90)] 被⼴⼤开发者所使⽤，⽬前主流的 WEB 应⽤程序均在使⽤此框架。使⽤ MVC 框架，可以将 WEB 应⽤进⾏清晰的分层开发，前端开发⼈员主要负责页⾯的呈现⽅式和⽤户体验，后台⼈员则主要负责应⽤的逻辑实现以及数据结构的搭建，极⼤的降低了研发成本。

随着信息化的进⼀步普及，数据库的使⽤越来越⼴泛，数据成为⼀个应⽤甚

⾄也个企业最重要的价值体现，越来越多的企业发展离不开数据库。因此数据库的稳定性和安全性也称为很多企业研发的重点。通过设计数据的管理和使⽤机制， 在提⾼数据库使⽤效率的同时，保障数据的完整性和安全性是企业运维⼈员的在运维过程中的重要任务。

除此之外，如何实现 WEB 应⽤的⾼可⽤性和分布式服务也是保障⽤户体验的

⼀个很重要的部分，通过设计基于应⽤和服务器的不同层级不同纬度的优化策略， 提升应⽤的⾼可⽤性也是在 WEB 应⽤开发中必须要注意的。

## 国内外研究现状

从上世纪九⼗年代开始，计算机技术和 Internet 互联⽹技术开始迅速发展，随着这些技术的不断发展，在互联⽹络中⽹络信息的存储量级和访问数量都是以⼏何数级进⾏增长，随之⽽来导致的问题就是⽹络访问的拥塞问题和⽹络服务的超载运⾏[[6](#_bookmark91)]。

2006 年新华⽹被⿊事件、2010 年的百度域名劫持时间、2011 年的CSDN ⽤户数据泄漏事件以及 2015 年⽹易邮箱密码泄漏事件等事件⽆不说明系统优化的作⽤和意义。

⽬前⼤部分互联⽹产品在开始发展的阶段都重视了产品的设计和功能，然⽽在产品的稳定性和安全性⽅⾯却考虑不⾜，这在⼀定程度上增加了互联⽹产品在推⼴过程中潜在的风险。因此，在开发互联⽹产品的同时，通过⼀系列的优化策略，提升应⽤的体验，增强应⽤的稳定性和安全性必须得到充分的认识。

## 论文主要研究内容

* + 1. 论⽂主要⼯作内容

本篇论⽂通过在满⾜本⼈参与的 WEB 应⽤正常运⾏的基础上，通过应⽤、数据和服务等不同维度的的优化实践研究基于 Spring MVC 架构 WEB 应⽤的系统优化策略。⾸先通过搭建基于 Spring MVC 的 WEB 测试应⽤，在实际⽤户使⽤的过程中通过不断开发和调整系统的优化策略，提升应⽤的⽤户体验和应⽤的稳定性， 探索⾏之有效的系统优化策略。

本⽂的研究对象主要有以下内容：

* + - 1. 对 Jenkins 持续集成环境研究。⾃动化部署和代码检测是保证基于 WEB 的应⽤产品质量的⼀个重要环节，通过研究部署 Jenkins 集成测试环境，对编写的代码进⾏版本控制、⾃动化构建和代码测试，研究持续集成⽅案对于 WEB 应⽤系统优化策略的影响。
      2. 对 Couchbase 缓存机制进⾏研究。⽬前⼤多数的 WEB 应⽤对于缓存性能的优化还没有⾜够的重视，通过开发针对 WEB 应⽤的Couchbase 缓存系统，研究Couchbase 的缓存机制测试有效的缓存机制对于系统性能提升和⽤户体验的影响， 探索可⽤的系统优化策略。
      3. 对Docker 容器编排技术进⾏研究。随着⽤户的不断增加，单⼀节点的 WEB 应⽤或数据库应⽤已经⽆法满⾜⽤户的需求，如何快速的部署新的应⽤节点提升应⽤性能成为开发者关注的问题。通过构建基于 Docker 容器编排技术的应⽤容器， 在新的服务器快速部署新的应⽤，并加⼊到应⽤集群中去，探索 Docker 机制对于系统优化策略的影响。
      4. 对数据库的主从复制进⾏研究。随着应⽤的发展，数据逐渐成为应⽤最有价值的部分，如何更好的保证数据的完整性以及安全性在应⽤的维护过程中⽇益重要。通过研究数据库的主从复制和延时复制⽅案，探索数据库优化对于系统安全性优化的策略。
      5. 对基于 API 的监控和应急措施⽅案进⾏研究。随着服务节点的不断增加， 服务的⾼可⽤和服务的健康性检查时 WEB 应⽤的运维⼈员在维护过程中必须要注意的⽅⾯。通过开发基于 API 的服务器、服务监控系统以及基于 API 的应急处理系统研究 API 操作对于系统的快速检测和快速恢复的影响。
      6. 其他⽅⾯，研究 WEB 应⽤的搭建过程、Tomcat 的相关配置、负载均衡以及阿⾥云的相关配置研究 WEB 应⽤和服务器⾃⾝优化对于系统性能的影响。
    1. 论⽂⽬标

本论⽂致⼒于分析⽬前很多 WEB 应⽤开发过程中存在的问题和不完善的地

⽅，并探索 WEB 应⽤性能和服务器性能的优化⽅案。

在 Linux 平台上，使⽤ JAVA 语⾔和 MySQL 数据搭建⼀个基于 Spring MVC 架构的 WEB 应⽤，通过 Tomcat 实现应⽤的访问和测试，通过设计不同的优化⽅案对 WEB 进⾏测试，研究在实际应⽤中有价值的优化策略。

系统性能优化策略研究主要包含应⽤层⾯、数据层⾯和服务器层⾯三个层⾯。在应⽤层⾯，通过持续集成、代码审核实现应⽤的稳定性和安全性。在数据库层

⾯，通过多数据库主从复制、负载均衡和备份恢复等策略实现数据的⾼可⽤和稳定性。在服务器层⾯，通过对服务的监控和服务器的监控、服务器间负载均衡的配置、基于 API 的⾃动化 failover 等策略保障服务的正常运⾏和⾼负载应对。

1.4 **论文组织结构**

基于 Spring MVC 架构 WEB 应⽤的系统优化策略的研究是按照计划逐步完成的，本⼈将此研究分为六个部分：

第⼀章：绪论。本章主要介绍 WEB 应⽤开发的现状和系统优化策略研究的意义，明确了在 WEB 应⽤开发过程中对于应⽤和服务器进⾏优化的必要性和重要性。除此之外，本章还介绍了论⽂主研究和实验的主要内容、论⽂的⽬标、论⽂创新点以及本篇论⽂的主要结构。

第⼆章：WEB 应⽤开发介绍。本章主要介绍了前端和后端的开发框架和开发流程以及数据库的搭建过程。然后介绍了 Tomcat 的配置过程。之后介绍了 Jenkins 持续集成环境的搭建和使⽤过程。

第三章：应⽤性能优化介绍。本章主要介绍了在 WEB 应⽤性能优化⽅⾯册主要⽅法和策略，主要包括 Couchbase 缓存优化，Tomcat ⾼性能 Apr 配置，Docker 分布式优化以及 SLB 负载均衡优化等⽅⾯。同时对应⽤的安全性和稳定性进⾏了分析。

第四章：数据优化。本章主要介绍了 MySQL 数据库在设计和开发过程的优化策略，主要包括基于数据稳定性的复制⽅案，基于⾼可⽤的负载均衡⽅案，基于数据完整性的备份和恢复⽅案。同时总结了出现问题时的解决⽅案。

第五章：服务监控与应急措施优化。本章主要介绍了服务器层⾯的优化策略， 主要包括多服务器的⼼跳监听服务配置，服务性能的监控脚本实现，基于 API 的应急措施处置以及服务器安全性配置等⽅⾯的优化策略。同时总结了多种⽅案整合使⽤的策略。

第六章：总结与展望。本章主要总结了不同优化策略对于系统性能的影响以及在研究过程中获得的经验，并对优化策略的进⼀步研究做了简单的展望。

1. WEB **应用开发及持续集成**

本论⽂的研究⽬的是对于开发的 WEB 应⽤进⾏应⽤、数据库和服务器层级进

⾏优化，通过对⽐不同优化策略对于系统性能提升的影响研究⼀套可⾏的系统优化策略。为了实现优化的⽬的，⾸先需要创建⼀个 WEB 应⽤或者对现有的应⽤进

⾏优化，本⼈在研究⽣期间，参与了海信集团智能商⽤公司的⼩微商户平台系统的开发，主要⼯作是后台开发和服务器运维⼯作，因此本论⽂将基于参与的⼩微商户 WEB 平台进⾏系统优化策略的研究。

在优化之前，⾸先需要描述作为优化原型的⼩微商户 WEB 平台的平台的开发框架以及应⽤部署流程。

* 1. Spring MVC **开发框架**

Spring MVC 开发框架是⼀种针对 Java 语⾔开发的 WEB 系统架构，该框架的设计理念是的开发的系统整体进⾏分解，通过不同的⽅式对分解后的⼯作进⾏处理，以更加专业的技术解决具体的问题，最中在不同的模块中通过调⽤实现系统的整体呈现，通过这种⽅法系统的耦合度同其他的框架如Struts 相⽐⼤⼤的降低。[[2](#_bookmark87)]。

* + 1. Spring MVC 框架处理流程

本框架是⼀个基于⽤户请求的系统架构。框架的处理过程参见图 [2-1](#_bookmark10)，图形反映了⼀个简单的从⽤户请求开始到获得响应的过程。

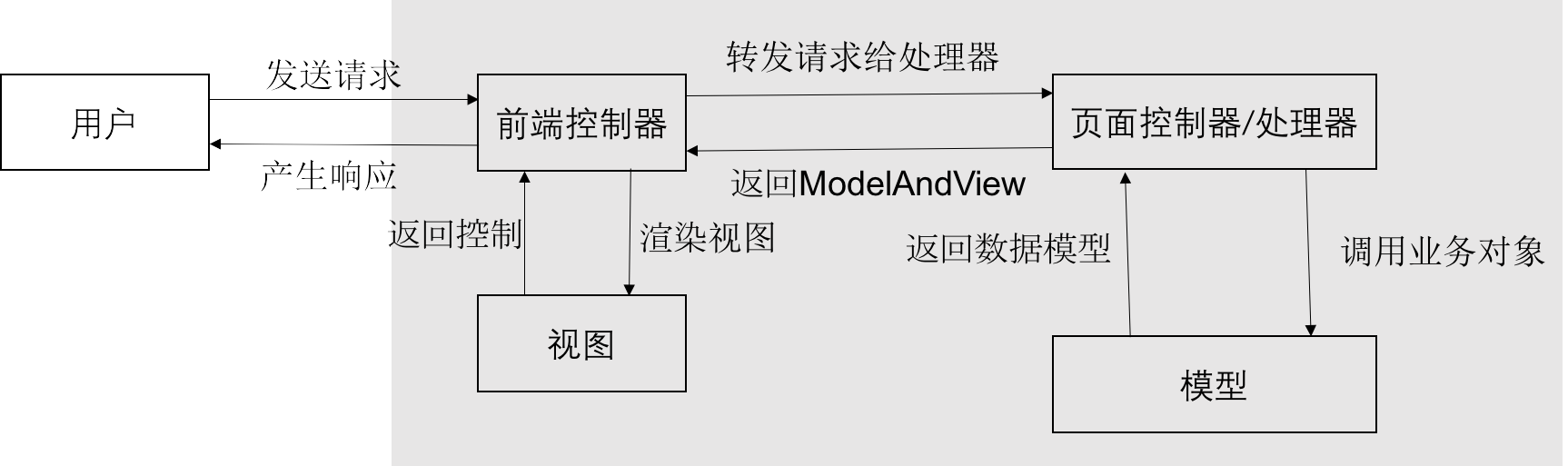


图 2-1 Spring MVC 请求响应过程

通过图可以看出，⾸先需要⽤户发送请求到框架的前端控制器，前端控制器接收到⽤户请求后会对⽤户的请求内容进⾏分析，获取到请求的控制器地址和请

求参数，根据请求地址将⽤户请求转发到对应的后端控制器即处理器；处理器在接收到请求的参数之后开始调⽤后台的业务对象对请求进⾏分析和处理，在处理过程中会通过模型对存在数据库中的数据进⾏增、删、改、查的操作，处理完毕后将最新的数据返回给处理器；在请求的处理⼯作完成返回给前端控制器，之后对需要显⽰的结果进⾏视图渲染⽣成显⽰页⾯，展⽰给⽤户，对⽤户的请求产⽣了响应[[7](#_bookmark92)]。

* + 1. Spring MVC 体系的三层架构

Spring MVC 开发框架的架构主要体现在 MVC(Model View Controller)，它是模型 (M)、视图 (V) 和控制 (C) 三个英⽂单词的⾸字母缩写，根据字⾯意思我们也可以本系统的架构分为模型层（数据访问层）、视图层和控制层三层。

1. 视图层。主要负责前端页⾯的⽤户请求，将⽤户请求按照 URL Mapping ⽅式映射到相对应的控制器，通过分析⽤户请求，Spring 可以⾃动的去寻找响应该请求的控制类，当控制类处理完请求后会将请求的结果以⽤户指定⽅式显

⽰在前端页⾯上。

1. 控制层。控制层是架构中承上启下的⼀层，这⼀层的作⽤是接受视图层传来的⽤户请求，然后针对请求设计实现⽤户请求的⽅法，需要⽤到数据的则会通过模型层获取数据结果，经过⽅法的处理获得⽤户请求的结果，并且将结果返回视图层。
2. 模型层，也可以成为数据访问层。这⼀层的作⽤主要是实现数据的访问⽅法和数据的返回⽅法，通过配置数据模型访问数据库并且获得需要的数据，将数据返回给控制层。

以上的 Spring MVC 三层架构体系在很⼤程度上根据开发需求实现了业务的剥离， 解决了开发过程中的开发资源和业务资源混乱的问题，⽽且降低了系统的耦合度。

## 应用开发工具

在开发基于 Spring MVC 的 WEB 应⽤过程中，需要⽤到的基础编程语⾔是

JAVA，系统的架构采⽤的 MVC 三层架构。但是在架构之外，程序本⾝的开发对于开发⼈员也⾮常重要，因此通过选择和使⽤⼀些⽐较好⽤的软件和⼯具，对于缩短开发周期提升开发效率来说⾮常重要。

* + 1. IntelliJ IDEA

IntelliJ IDEA 是⼀款 Java 集中开发环境⼯具软件，由捷克软件公司 JetBrains

发布和维护[[8](#_bookmark93)]。

随着⽤户数量的增加和软件⾃⾝的优良特性，⽬前以及成为 Java 语⾔开发过程中效率最好的集成开发环境之⼀。由于它本⾝已经集成和⾮常多的使⽤功能和快捷键，因此在开发⼈员的使⽤过程中⼏乎可以摆脱⿏标，⽽且开发效率不降反升[[9](#_bookmark94)]。

* + 1. Maven

Apache Maven 是由 Apache 软件基⾦会所提供德⼀个项⽬管理及⾃动构建⼯具[[10](#_bookmark95)]。它基于项⽬对象模型概念、能够实现⼀个 Java 项⽬的构建和依赖管理。本论⽂所涉及的 WEB 应⽤就是使⽤ Maven 来构建 Java Web 项⽬。

* + 1. Tomcat

WEB 应⽤的 Web 服务器采⽤的是由 Apache 软件基⾦会 (Apache Software Foundation) 开发的⼀个 Servlet 容器，由于其本⾝也包含⼀个HTTP 服务器，因此也常常被⽤作⼀个单独的Web 服务器[[11](#_bookmark96)]。Tomcat7 ⽀持最新的 Servlet 3.0 规范，⽽且技术先进、稳定性强，最重要的⼀点就是免费，因此获得了众多 Java 语⾔开发者的喜欢和中时，渐渐的已经成为主流的 Web 应⽤服务器之⼀。

* + 1. MySQL

MySQl 是个⼩型关系型数据库管理系统，之所以使⽤MySQL 是因为MySQL 是⼀款免费的数据库管理系统，⽽且其建议的操作以及其兼容性都是其优点[[12](#_bookmark97)]。MySQL 的特点主要有[[13](#_bookmark98)]：

* 为 C 语⾔、Java 语⾔、PHP 语⾔以及 Python 语⾔等多种编程语⾔提供了接

⼜，可以实现多语⾔环境对 MySQL 数据库的调⽤。

* 除了对语⾔的⽀持，MySQL 还⽀持多线程处理，可以更加充分的利⽤服务器资源。
* 除了 TCP/IP 协议外，还⽀持 JDBC 等多途径的数据库连接协议。
* 除了对于数据和语⾔的⽀持外，在管理⽅⾯也有⾮常好的数据库管理⼯具， 除了⾃⼰开发的Workbench 还⽀持许多第三⽅⼯具，这些都可以对数据的数据进⾏管理和优化以及检查数据库的异常。
  + 1. Couchbase

现代的互联⽹产品开发过程中，随着⽤户数量和要求的不断提⾼，我们需要我们的 WEB 产品可以同时⽀持更多的客户端节点并且是其保持在很低的请求延迟[[14](#_bookmark99)]。为了实现这⼀⽬的，我们就需要对平台的数据开发更强⼤的缓存机制以提

⾼数据的读写速度，在近⼏年主流的缓存系统有 memcached 和 redis，虽然它们有很成熟的解决⽅案，但是也都有其局限性[[15](#_bookmark100)]：

* 对于集群的⽀持不够好，只可以实现单个服务器的配置，不⽀持多服务器集群，这样就导致了在缓存扩容和负载均衡以及系统的⾼可⽤等多⽅⾯的缺点。
* 数据持久化和故障转移表现很差，在缓存系统出现问题后修复的成本⾼。memcached 缓存系统不⽀数据持持久化，redis 缓存系统的持久化配置会导致服务器的负载不均衡，可能会出现间歇性负载过⾼的现象。

Couchbase 是⼀个NoSQL 数据库，它是世界各国的开发者在 2011 年推出的，由于它有良好的 cluster ⽀持、异步持久化的⽀持得到了众多开发者的青睐，他的特点主要有：

* Couchbase 缓存系统对于⾃⾝的缓存配置有⼀个专业的 WEB 管理界⾯，除了通过页⾯管理，还可以通过API 接⼜对缓存系统进⾏配置和管理，这些是memcached, redis 不能企及的。
* Couchbase 引⼊了虚拟 Bucket 的概念，这是建⽴在集群和负载均衡的基础上的，通过它可以把数据⾮常灵活的部署到各个集群节点中，这样对于集群就可以进⾏灵活的动态的管理。
* Couchbase 的对等⽹设计实现了集群的负载均衡，通过智能的客户端⽅式可以获取集群的信息和各节点的信息，⽽且还⽀持集群节点的横向扩展。

## 应用开发流程

WEB 应⽤在开发的时候设计为前后端分离，通过 FDP 平台实现前端到后端的请求，所以本论⽂测试 WEB 应⽤的开发主要分为前端开发、后端开发以及数据库搭建三个⽅⾯。

* + 1. 前端开发

应⽤的前端相当于MVC 架构中的视图层，主要实现⽤户的交互，包括页⾯信息的展⽰以及⽤户请求的转发和响应，在前端开发中，通过 Html 和 JavaScript 来设计实现应⽤的页⾯展⽰，通过 FDP 的 Ajax 请求将前端的⽤户请求转发到应⽤的后端。

* + 1. 后端开发

应⽤的后端主要分成了 Controller、Service、Pst 三层，其中Controller 层负责处理⽤户的请求然后将业务转发给 Service 层，Service 层负责实现⽤户的请求，通过设计不同的业务逻辑将⽤户需要的数据返回，涉及到数据的读取则通过 Pst 层，

Pst 层主要负责对数据库的增删改查，将结果返回到 Service 层。

* + 1. 数据库搭建

应⽤的数据主要分为基础数据也业务数据，其中基础数据主要包括页⾯的功能板块、系统的定时任务、数据的权限设置、页⾯的功能逻辑等数据，业务数据主要包括项⽬使⽤中需要存储的商户信息和操作记录、商户的商店、商品以及销售记录等。

* 1. **基于** Jenkins **的持续集成方案开发**

在每⼀次 WEB 应⽤的开发、上线过程中，不可避免的要将本地环境打包上传到⽣产环境或者是测试环境进⾏解压，每⼀次⼈⼯的⼲预⽆疑增加了时间成本和错误率，通过 Jenkis 设计实现应⽤的持续集成，这在很⼤程度上能够帮助开发着实现快速的应⽤部署和错误重现[[16](#_bookmark101)]。

Jenkins 是⼀个⽤ Java 编写的开源的持续集成⼯具，它提供了软件开发的持续集成服务，运⾏在 Servlet 容器中，通过 Jeknins 可以构建基于Apache Ant 和Apache Maven 的项⽬，除了构建的功能以外，Jenkins 还可以执⾏ Linux 环境下的 Shell 命令或者脚本以及 Windows 环境下的 bat 批处理命令[[17](#_bookmark102)]。

由于⼩微平台是通过 Java 语⾔开发的，因此可以通过 Jenkins 的 Maven ⼯具进⾏构建并进⾏语法检查，通过 SSH 插件上传构建后的 war 包以及前端页⾯到服务器端实现部署[[18](#_bookmark103)]。具体的⾃动构建及部署流程如下：

* + 1. Jenkins 软件安装配置

由于 Jenkins 运⾏在 Servlet 容器中，因此在安装配置 Jenkins 之前需要保证安装 Jenkins 的服务器中已经安装好了对应版本的 JDK 和相匹配的 Tomcat 软件[[19](#_bookmark104)]。

* + - 1. 访问<http://mirrors.jenkins.io/war-stable/2.7.1/jenkins.war> 下载 2.7.1 版本的jenk- ins 安装包。
      2. 将 jenkins.war ⽂件放到本地构建服务器（10.18.8.19）的 Tomcat 应⽤⽬录

（/opt/tomcat/apache-tomcat-7.0.68/webapps）中。

* + - 1. 配置并启动 Tomcat 即可完成安装，端⼜配置为 8080，启动命令配置为：

systemctl start/stop/status tomcat\_jenkins.service

* + - 1. 访问 jenkins 地址 (http://10.18.8.19:8080/jenkins/) 初始化 Jenkins:
         * 完成认证, 按照页⾯提⽰，将对应⽂件内容中的密码输⼊到⽂本框中，点击 continue，如图[2-2](#_bookmark24)所⽰

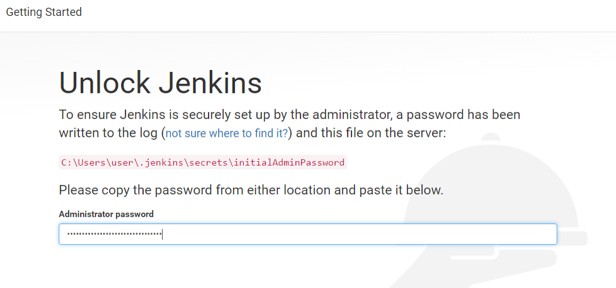


图 2-2 认证 Jenkins 界⾯

* + - * + 在插件选择处选择安装建议的插件
        + 安装完成后在新的页⾯中配置⽤户名和密码完成安装，如图[2-3](#_bookmark25)

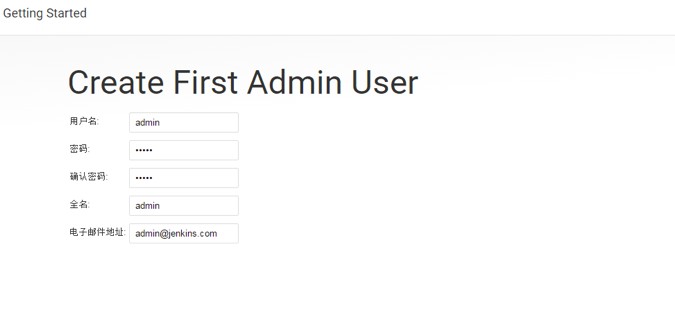


图 2-3 创建⽤户名和密码

* + - * + 进⼊图[2-4](#_bookmark26)页⾯表⽰安装成功



图 2-4 Jenkins 使⽤界⾯

* + 1. ⾃动构建及检查⽅案设计

由于⼩微平台通过 Maven ⼯具来解决代码使⽤过程中的函数依赖和本地项⽬构建[[20](#_bookmark105)]，所以在进⾏⾃动构建的时候同样可以通过使⽤ Maven Integration plugin 插件实现项⽬的构建和代码检查，另外由于⼩微平台在开发过程中使⽤SVN 来进⾏代码的版本控制，因此也需要在 Jenkins 中配置 SVN 插件来进⾏代码的同步和版本管理，除此之外，需要在安装 Jenkins 的服务器中提前安装好 Maven 软件。

⼩微项⽬在开发过程中通过前后端分离进⾏的开发，因此在构建过程中需要新建前端项⽬和后端项⽬，考虑到前端项⽬的构建相对后端项⽬⽐较简单，本论

⽂只介绍后端项⽬的创建和构建过程。

* 新建 Jenkins 项⽬
  + - 1. 在 Jenkins 主页⾯新建项⽬后，⾸先填写项⽬名称（以 ServerDeployfor- PROD1.2 为例）
      2. 源码管理配置为 SVN，具体如图[2-5](#_bookmark28)所⽰：

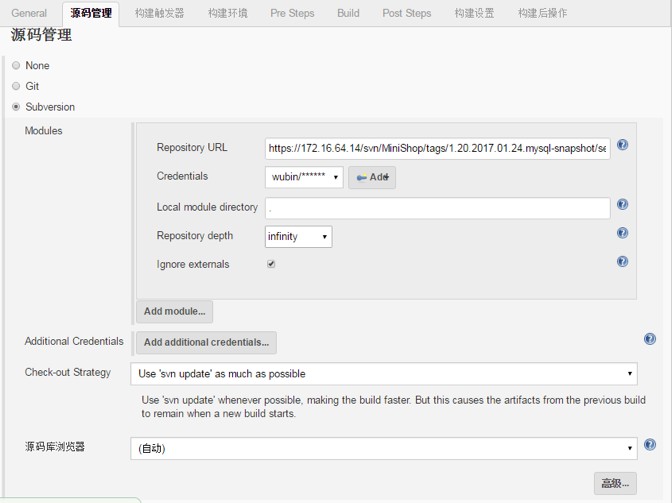


图 2-5 Jenkins SVN 配置

在对应的框中填写代码库的地址、通过Add 按钮实现SVN 的权限认证， 其它选项选择默认即可。

* + - 1. 在 Pre Steps 处增加 shell 命令为:

rm -rf target/MiniShop

确保每次构建时清除前⼀次构建信息。

* + - 1. 在build 标签页下配置构建时的操作，主要包括配置POM ⽂件(pom.xml) 的位置,pom.xml ⽂件主要描述本 Maven ⼯程的整个⽣命周期所需要执⾏的功能和特性[[21](#_bookmark106)]，考虑到本项⽬的实际开发过程在这⾥选择项⽬pom2.xml
      2. 在 Post Steps 配置构建完成后的操作，⾸先需要在"Run only if build suc- ceeds or is unstable" 出打勾，保证在只有构建成功后才执⾏构建完成后的操作，其次需要配置构建完成对⽂件的操作：

cp target/MiniShop.war /data/**test**/server/PROD/V1.2/ MiniShop1.2\_$(date +%Y%m%d)\_$BUILD\_NUMBER.war && chmod 777 /data/**test**/server/PROD/V1.2/MiniShop1

.2\_$(date +%Y%m%d)\_$BUILD\_NUMBER.war

通过上述命令将每次构建完成后的⽂件进⾏备份保存和修改响应权限。

所有项⽬信息配置完成后保存即可。

* 构建 Jenkins 项⽬进⼊项⽬主页，在页⾯左侧点击“⽴即构建”按钮即可进⾏构建，构建完成后将构建完成的 war 包上传到服务器即可，如果构建失败则表

⽰在代码的书写过程中存在错误或者项⽬的库存在异常，需要在项⽬构建页

⾯的构建信息页⾯中查看错误信息，并且根据错误信息来解决问题。

* + 1. ⾃动部署⽅案设计

Jenkins ⾃动部署的⽅案是在每次构建完成后，让 Jenkins 可以⾃动的通过 SSH 协议访问远程服务器并将构建完成后的⽂件上传到服务器，并且执⾏服务器中的相应脚本来实现⾃动备份旧项⽬和部署新项⽬的⽬的。

* + - 1. 在具体配置⾃动部署之前，需要先安装 Publish Over SSH 插件，通过这个插件，Jenkins 可以实现通过 SSH 协议对远程服务器的访问。
      2. 在插件安装完成之后，需要配置插件来配置访问 SSH 的密钥和密码，在 Jenk- ins“系统管理-系统设置”页⾯中会出现如图[2-6](#_bookmark30)配置：

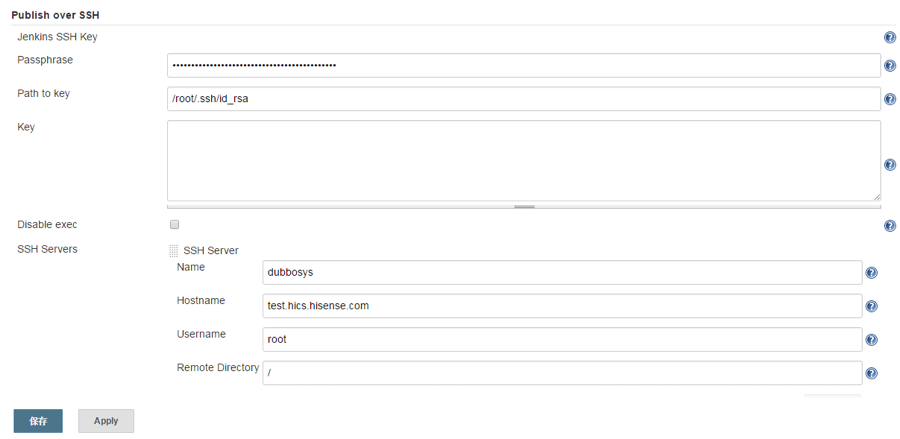


图 2-6 Publish Over SSH 插件配置

按照不同项⽬配置 SSH Server 信息和登录验证信息等。

* + - 1. 插件配置完成后需要配置⾃动部署，插件安装完成后在“构建后操作”的选项中会出现“Send build artifacts over SSh”的选项，点击之后会出现配置：

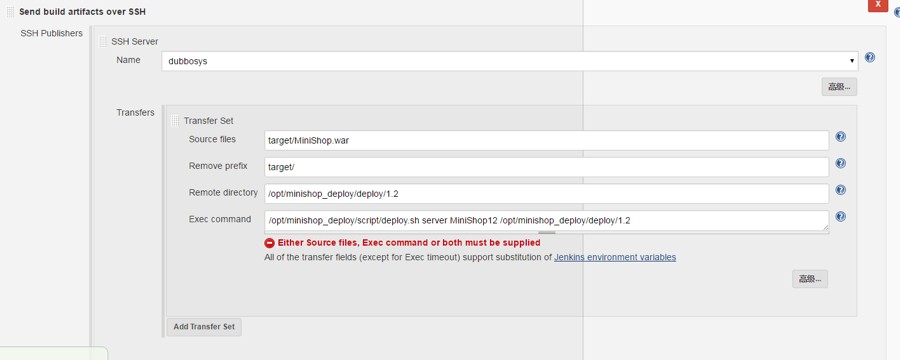


图 2-7 Jenkins ⾃动部署配置

配置上传的⽂件名，上传后的路径，上传后需要执⾏的脚本和参数等，脚本可以参考附录[A](#_bookmark133)。

* + - 1. 配置完成后再进⾏构建，Jenkins 会⾃动的在构建后将构建⽣成的⽂件上传到服务器端指定路径，通过执⾏指定的脚本和参数将⽂件部署到 Tomcat 中，实现⾃动部署。

2.5 **本章总结**

本章主要介绍了本⼈参与的⼩微项⽬的 WEB 平台所使⽤的开发框架、开发⼯具、开发过程以及系统上线的持续集成⽅案，通过持续集成⽅案增强了代码上线的稳定性，在本⽂后⾯的三章将会对本章开发的平台进⾏不通层级的优化，来实现本项⽬的安全性和⾼性能。

# 应用性能优化

* 1. Couchbase **缓存优化**

⽬前⼤多数的 WEB 应⽤产品在设计开发的过程中对数据的读取还依旧通过直接对数据库进⾏增、删、改、查来实现。然⽽随着⽤户数量的增加，⽤户的请求也不断的增多，这对于数据库的压⼒是⾮常⼤的。之前的数据操作流程参见图 [3-1](#_bookmark34)。

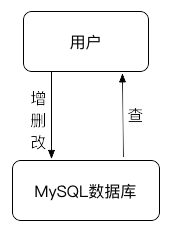


图 3-1 通常数据操作流程

为了缓解数据库的压⼒，在数据的读取过程中将系统的基础数据读取到Couchbase 缓存中，这样只需要对数据的基础数据进⾏⼀次读取即可完成数据的加载，降低了数据库的压⼒。通过缓存的数据操作流程参见图 [3-2](#_bookmark35)。

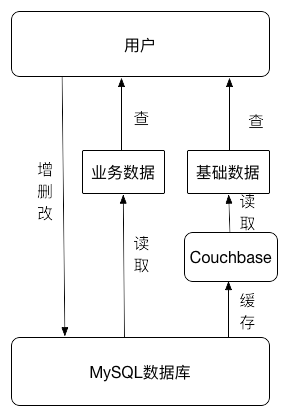


图 3-2 通常数据操作流程

本论⽂中的 WEB 应⽤的数据包含基础数据和应⽤数据两部分，其中基础数据主要包括系统的基本配置数据、系统的常⽤变量数据、⽤户权限数据、枚举类型等相对固定的数据，这些数据在⽤户访问应⽤的时候只需要加载⼀次即可，不需要重复的从数据中读取，⽽应⽤数据则主要包含应⽤内的⽤户信息、商户信息、交

易数据、库存数据等等不固定的数据，这些数据随着⽤户的时候会不断的发⽣变化，⽽且新的数据从数据库重新读取，这些数据在⽤户使⽤时需要多次加载。针对于以上不同数据的特点，将基础数据在⽤户⾸次登陆时加载到 Couchbase 缓存中， 以后再读取时直接从缓存中读取。

* + 1. Couchbase 安装
       1. 直接安装[[14](#_bookmark99)]

⾸先，通过浏览器访问<https://www.couchbase.com/nosql-databases/downloads> ， 在页⾯中选择 4.1.0 版本、⽀持 Red Hat 7 平台的 Couchbase Server 安装⽂件couchbase-server-enterprise-4.1.0-centos7.x86\_64.rpm, 将 rpm ⽂件复制到服务器中。或者直接在 shell 中通过

$ wget https://packages.couchbase.com/releases/4.1.0/

couchbase-server-enterprise-4.1.0-centos7.x86\_64.rpm

命令将 rpm 安装⽂件下载到制定路径中。

然后，通过 root ⽤户访问 Linux 服务器，在终端中进⼊到 couchbase 安装⽂件所在路径，通过如下命令安装 couchbase：

$ rpm --install couchbase-server-enterprise-4.1.0-

centos7.x86\_64.rpm

安装完成后⾃动启动 couchbase，并且会有如下提⽰信息：

Warning: Transparent hugepages looks to

should not be.

Please look at <http://bit.ly/1ZAcLjD> as PERMANENTLY alter this setting.

Minimum RAM required : 4 GB

System RAM configured : 3.70 GB

be active and

**for** how to

Minimum number of processors required : Number of processors on the system

:

4 cores

1 cores

Reloading systemd:

[ OK ]

Starting couchbase-server (via systemctl):

[ OK ]

You have successfully installed Couchbase Server.

Please browse to http://iZ257vyhrvzZ:8091/ to configure your server.

Please refer to [http://couchbase.com](http://couchbase.com/) **for** additional resources.

Please note that you have to update your firewall configuration to

allow connections to the following ports: 11211, 11210, 11209, 4369,

8091, 8092, 8093, 9100 to 9105, 9998, 18091, 18092,

11214, 11215 and

from 21100 to 21299.

By using this software you agree to the End User License Agreement.

See /opt/couchbase/LICENSE.txt.

couchbase 默认安装位置为“/opt/couchbase/”

* + - 1. Docker 安装[[22](#_bookmark107)]

# 下载镜像

$ docker pull couchbase:4.1.0

# 运行容器

$ docker run -d --name db -p 8091-8094:8091-8094 -p 11210:11210 couchbase

Couchbase 安装完成后，需要在阿⾥云的安全策略中配置外⽹访问的端⼜和应⽤访问的端⼜，相关配置参见[5.4](#_bookmark79)

* + 1. Couchbase 集群配置

Couchbase 服务器及可以单独运⾏，也可以将多个服务器组成⼀个集群，作为集群来运⾏。通过 Couchbase 集群，可以实现缓存数据的分布式存储及负载均衡， 提升缓存的⾼可⽤以及系统的性能[[23](#_bookmark108)]。

Couchbase 数据分布是按计算分配到多个节点上，每个节点都储存两部分数据有效数据和副本数据，客户端对数据的操作主要是按照节点中对应的有效数据进

⾏操作，执⾏压⼒会部分到不同的节点，如 [3-3](#_bookmark38)所⽰：

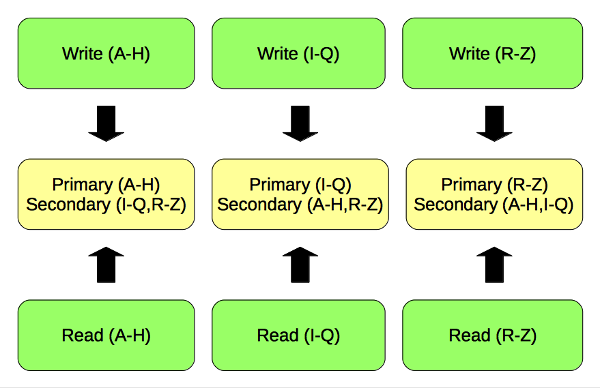


图 3-3 Couchbase 集群数据操作模型

Couchbase 的集群管理是由 erlang/otp 进⾏集群通信管理，集群之间使⽤⼼跳机制进⾏监测服务器节点健康监测，配置参数信息是同步到每⼀个节点上进⾏储存。整个集群以vbucket 为单位划分映射到不同服务器节点中进⾏储存，划分规则如下：

* + - 1. 均匀的分配有效 vbucket 和副本 vbucket 到不同服务器节点中；
      2. 把有效数据与副本数据划分到不同物理节点中；
      3. 在复制多份数据时，尽量有其它节点进⾏数据传播；
      4. 扩展时，以最⼩化数据迁移量进⾏复制。

在 Couchbase 负载均衡中，我们所操作的每⼀个 bucket 会逻辑划分为 1024 个 vbucket，其数据的储存基于每个 vbucket 储存并且每个 vbucket 都会映射到相对应的服务器节点，这种储存结构的⽅式叫做集群映射。如 [3-4](#_bookmark39)所⽰，当应⽤与Couchbase 服务器交互时，会通过 SDK 的与服务器数据进⾏交互，当应⽤操作某

⼀个的bucket 的key 值时，在 SDK 中会通过哈希的⽅式计算，使⽤公式 crc32(key)

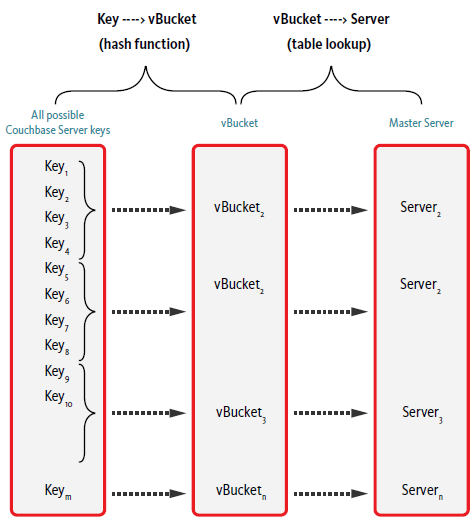


图 3-4 Couchbase 负载均衡模型

在设置标签页中的集群标签页下新建和配置集群信息

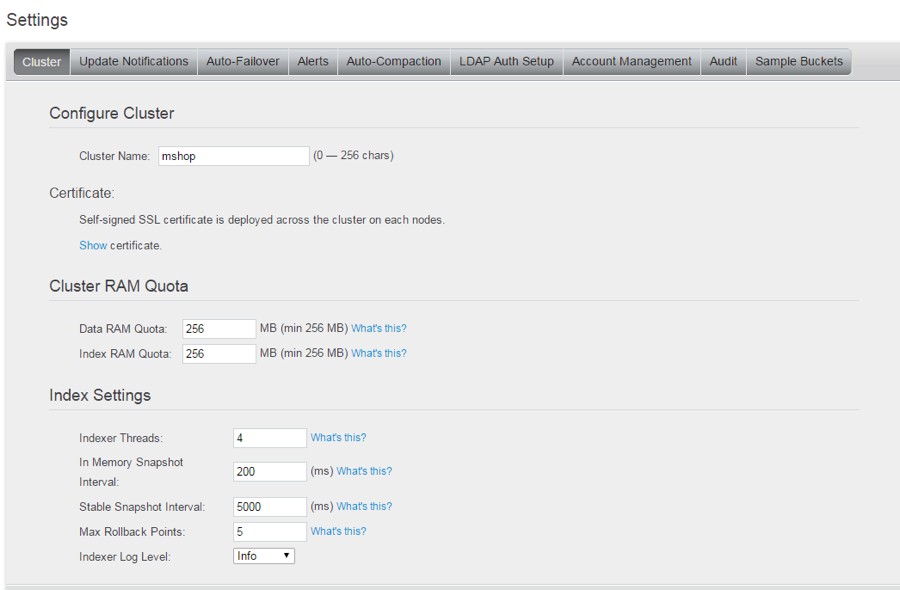


图 3-5 Couchbase 创建集群

在服务器节点标签页下新增服务器节点

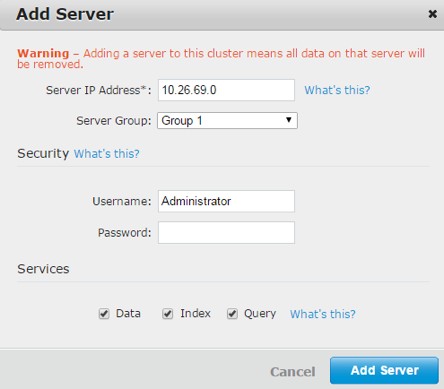


图 3-6 Couchbase 增加节点

增加完服务器节点后可以看到集群的基本信息

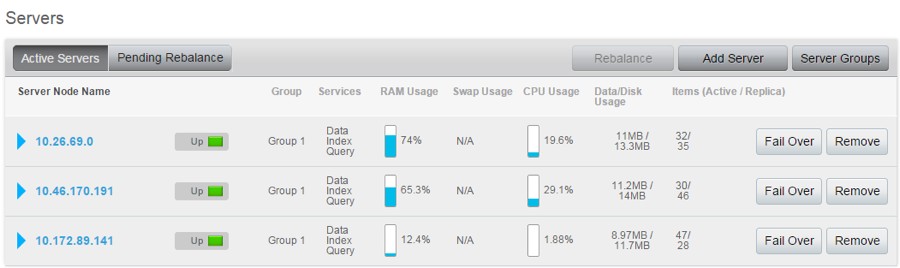


图 3-7 Couchbase 集群概览

* + 1. Couchbase 缓存对系统性能影响

通过ab 命令对使⽤应⽤进⾏压⼒测试，在开启 couchbase 和关闭couchbase 的情况下分别模拟 5 万个请求，每次并发 10 个请求来对⽐缓存对系统性能的影响：

$ ab -n 50000 -c

mnsindex.html

10

-k [http://**test**.hics.hisense.com/client/](http://test.hics.hisense.com/client/)

Server Software:

Server Hostname: Server Port:

Apache-Coyote/1.1

**test**.hics.hisense.com 80

Document Path:

Document Length:

/client/mnsindex.html

36028 bytes

Concurrency Level:

Time taken **for** tests:

10

206.318 seconds

Complete requests:

Failed requests: Keep-Alive requests: Total transferred: HTML transferred: Requests

Time per

Time per

per second:

request: request:

50000

0

49504

1816897520 bytes

1801400000 bytes

242.34 [#/sec] (mean)

41.264 [ms] (mean)

4.126 [ms] (mean, across all

concurrent requests)

Transfer rate: 8599.92 [Kbytes/sec] received

$ ab -n 50000 -c

mnsindex.html

10

-k [http://**test**.hics.hisense.com/client/](http://test.hics.hisense.com/client/)

Server Software: Apache-Coyote/1.1

Server Hostname: **test**.hics.hisense.com Server Port: 80

Document Path:

Document Length:

/client/mnsindex.html

36028 bytes

Concurrency Level:

Time taken **for** tests: Complete requests: Failed requests:

Keep-Alive requests: Total transferred: HTML transferred: Requests

Time per

Time per

per second:

request: request:

10

247.137 seconds 50000

0

49505

1814097525 bytes

1801400000 bytes

202.32 [#/sec] (mean)

49.427 [ms] (mean)

4.943 [ms] (mean, across all

concurrent requests)

Transfer rate: 7168.40 [Kbytes/sec] received

通过测试可以发现，开启 Couchbase 后系统的⽹络吞吐率在 242.34reqs/s, ⽽不使⽤Couchbase 时吞吐量在 202.32reqs/s，开启 Couchbase 对于系统性能的提升⼤约在

20% 左右。

* 1. Tomcat **高并发** APR **优化**

Tomcat 对⽤户请求的处理⽅式有BIO、NIO 以及APR 三种⽅式，其中 BIO 模式为 Tomcat 的默认处理⽅式[[24](#_bookmark109)]。

BIO 模式是⼀种阻塞式 I/O 操作，这种模式表⽰ Tomcat 使⽤的是传统 Java I/O 操作。在这种模式下对于每个请求都要创建⼀个线程来处理，线程开销较⼤，不能处理⾼并发的场景，在三种模式中性能也最低。启动 tomcat 看到如图 [3-8](#_bookmark42)所⽰⽇志，表⽰使⽤的是 BIO 模式：

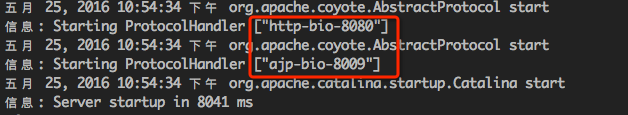


图 3-8 BIO 模式⽇志

NIO 模式是 Java SE 1.4 及后续版本提供的⼀种新的 I/O 操作⽅式。该模式是⼀个基于缓冲区、并能提供⾮阻塞 I/O 操作的 Java API，它拥有⽐传统 I/O 操作 (bio) 更好的并发运⾏性能。启动 tomcat 看到如图 [3-9](#_bookmark43)所⽰⽇志，表⽰使⽤的是 NIO 模式：

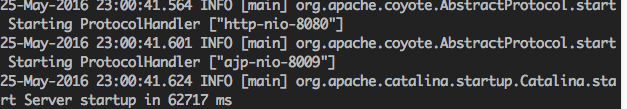


图 3-9 NIO 模式⽇志

APR 模式是从操作系统级别解决异步 IO 问题，⼤幅度的提⾼服务器的处理和响应性能，也是 Tomcat 运⾏⾼并发应⽤的⾸选模式。启动 tomcat 看到如图 [3-10](#_bookmark44)所

⽰⽇志，表⽰使⽤的是 APR 模式：

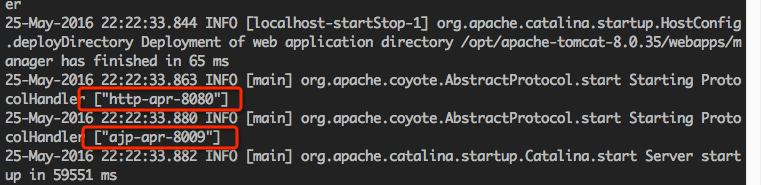


图 3-10 APR 模式⽇志

考虑到应⽤在⼤量⽤户访问的情况下必然会出现⾼并发的现象，因此调整Tomcat 的请求处理⽅式为 APR 模式对于提升系统的⾼并发处理能⼒时⾮常必要的[[25](#_bookmark110)]。

APR(Apache Portable Runtime) 是⼀个基于 Apache 协议的⾼可移植库。它可以实现本地的进程管理，例如内存配置和接⼜；可以访问系统的 IO 例如使⽤OpenSSL；⽽且可以访问系统的⼀些命令和功能，例如获取系统的运⾏状态等。以 上的⼀些功能可以将 Tomcat 打造成⼀个更加通⽤的 WEB 应⽤服务器，通过与本地环境的⾼度集成，发挥⽹络应⽤的⾼性能[[26](#_bookmark111)]。

在不配置 APR 模式的 WEB 应⽤中，300 个以上的并发访问会让服务器的系统进⾏很快⽤满，后期的请求会出现⽆限等待和丢包的情况，当配置 APR 模式后， 会发现服务器的压⼒⼤⼤降低[[27](#_bookmark112)]，⼏乎在短时间内就可以完成请求，因此在⽣产环境中，使⽤ APR 模式对于系统性能的提升尤为重要。

* + 1. 开启 APR 模式
       1. 安装依赖库, 因为 APR 模式是使⽤ JNI 技术调⽤操作系统 IO 接⼜，需要⽤到相关 API 的头⽂件，所以需要安装必须的依赖库：

yum install apr-devel

yum install openssl-devel yum install gcc

yum install make

注意：openssl 库要求在 0.9.7 以上版本，APR 要求在 1.2 以上版本，⽤ rpm -qa

| grep openssl 检查本机安装的依赖库版本是否⼤于或等于 apr 要求的版本。

* + - 1. 在安装完依赖之后，需要安装 APR 的动态库，⾸先进⼊ Tomcat ⽬录下的 bin ⽬录中，解压其中的 tomcat-native.tar.gz ⽂件，并进⼊ tomcat-native- 1.2.10-src/native ⽬录，通过 configure 命令监测安装平台，通过 make 进⾏编译，最后通过 make install 命令完成安装，安装完成后动态库的安装路径为/usr/local/apr/lib ⽬录。
      2. APR 动态库安装完成后，需要配置 APR 本地库到系统共享库搜索路径中，编辑 Tomcat ⽬录中 bin ⽬录下的 catalina.sh ⽂件，在虚拟机启动参数配置中增加 JAVA\_OPTS 变量的值中添加 java 的 library 的路径，指定 apr 库的路径：

JAVA\_OPTS= *"$JAVA\_OPTS* −*Djava . l i b r a r y . path =/ usr / l o c a l /*

*apr / l i b "*

Tomcat8 以下版本，需要指定运⾏模式，将 protocol 从 HTTP/1.1 改成org.apache.coyote.http11.Http11AprProtocol，修改 Tomcat ⽬录下 conf ⽬录中的 server.xml ⽂件:

#禁用SSl

<!--APR library loader. Documentation at /docs/apr. html -->

<Listener className="org.apache.catalina.core.

AprLifecycleListener" SSLEngine="off" />

#修改链接器为

<Executor name="tomcatThreadPool" namePrefix="catalina

-exec-"

maxThreads="200" maxIdleTime="60000" minSpareThreads="20"/>

<Connector executor="tomcatThreadPool"

port="8089" protocol="org.apache.coyote

.http11.Http11AprProtocol" maxHttpHeaderSize="8192" maxThreads

="300" processorCache="1000" acceptCount="400"

minSpareThreads="50" URIEncoding=’utf

-8’ acceptorThreadCount="8" enableLookups="false" redirectPort

="8443"

connectionTimeout="120000" keepAliveTimeout="120000"

maxKeepAliveRequests="65535" disableUploadTimeout="true"

compression="on" compressionMinSize

="2048"

noCompressionUserAgents="gozilla, traviata"

compressableMimeType="text/html,text/ xml,text/javascript,text/css,text/ plain,application/json,

application/x-javascript "/>

* + - 1. 配置完成后，重启 Tomcat，在⽇志⽂件中包含图 [3-10](#_bookmark44)中的⽇志信息表⽰APR

模式开启成功。

* + 1. APR 模式对于系统性能的影响

为了测试 APR 对于系统性能提升的影响，同样通过 ab ⼯具对系统进⾏压⼒测试，分别模拟 10000 次、50000 次和 100000 次请求，每⼀次都并发 10/1000 次两种情况下分别测试系统的压⼒，结果如表[3-1](#_bookmark47)所⽰。根据统计结果可以发现，在

|  |  |  |  |
| --- | --- | --- | --- |
|  | 表 3-1 | APR 模式性能对⽐表 |  |
| **请求此时** | **并发数量** | APR **吞吐量** | BIO **吞吐量** |
| 10000 | 10 | 6612.60 | 8255.48 |
| 10000 | 1000 | 6966.93 | 6155.21 |
| 50000 | 10 | 7480.96 | 8841.41 |
| 50000 | 1000 | 9187.63 | 1957.35 |
| 100000 | 10 | 6588.83 | 8355.85 |
| 100000 | 1000 | 7751.47 | - |

低并发的情况下，APR 模式同默认模式BIO 模式相⽐，APR 模式吞吐量较低，但相差不⼤；然⽽在⾼并发的情况下，随着请求次数的增加，APR 模式的优势逐渐明显显现出来，在 50000 次请求，每次并发 1000 个请求的情况下，APR 模式远⾼于 BIO 模式，在 100000 次请求，每次并发 1000 个请求的情况下，BIO 模式⽆法完成测试，通过数据可以看出，开启 APR 模式对于 WEB 平台应对⾼并发请求有着⾮常重要的作⽤。

* 1. Docker **分布式优化**

随着⽤户数量的增加以及应⽤业务的扩展，⽬前 WEB 应⽤的数据库服务器已经由⼀台服务器扩展为两台服务器，应⽤服务器也已经由⼀台服务器扩展为两台服务器，加上⽬前的测试服务器，⼀共有 5 台服务器服务于⼀个 WEB 应⽤，⽽且在未来的发展过程中，⽆论是数据节点还是应⽤节点，数量肯定时不断增加的。在节点不断增加的情况下，如何快速部署⼀个数据库节点或者⼀个应⽤节点必然是运维⼈员在新增节点时必须要⾯对的问题[[28](#_bookmark113)]。

在新增节点上部署服务的⽅式主要有直接安装以及容器技术两种⽅式，其中直接安装的⽅式依靠⼿动的在服务器中安装各种库以及依赖，然后安装应⽤软件病进⾏配置，这种⽅式在快速部署⼀个节点的过程中的劣势⾮常明显。

⽬前实现快速部署新增节点的解决⽅案主要有虚拟机技术和容器技术两种技术，其中传统虚拟机技术是虚拟出⼀套硬件后，在其上运⾏⼀个完整操作系统，在该系统上再运⾏所需应⽤进程，如图 [3-11](#_bookmark49)所⽰；⽽容器内的应⽤进程直接运⾏于宿主的内核，容器内没有⾃⼰的内核，⽽且也没有进⾏硬件虚拟。因此容器要⽐传统虚拟机更为轻便如图 [3-12](#_bookmark50)所⽰[[29](#_bookmark114)]。

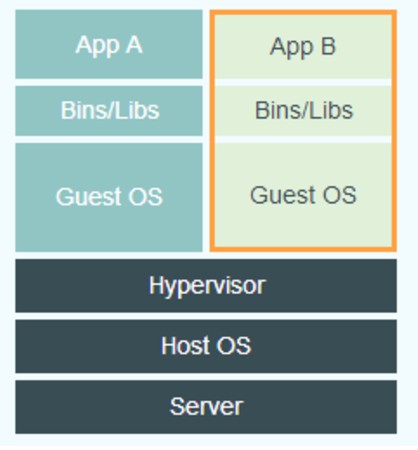


图 3-11 传统虚拟机模型

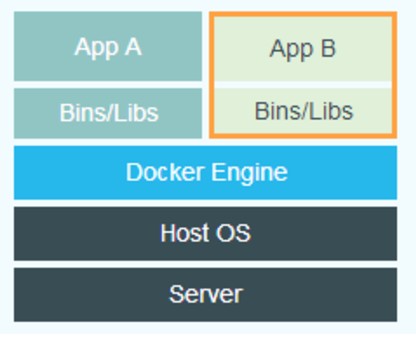


图 3-12 Docker 容器模型

同传统的虚拟机技术相⽐，Docker 容积技术具有⾮常多的优势，这也是本论

⽂中使⽤ Docker 技术作为 WEB 应⽤开发过程中服务器扩展的主要技术[[30](#_bookmark115)]。

* 更⾼效的利⽤系统资源

由于容器不需要进⾏硬件虚拟以及运⾏完整操作系统等额外开销，Docker 对系统资源的利⽤率更⾼。⽆论是应⽤执⾏速度、内存损耗或者⽂件存储速度， 都要⽐传统虚拟机技术更⾼效。因此，相⽐虚拟机技术，⼀个相同配置的主机，往往可以运⾏更多数量的应⽤。

* 更快速的启动时间

传统的虚拟机技术启动应⽤服务往往需要数分钟，⽽ Docker 容器应⽤，由于直接运⾏于宿主内核，⽆需启动完整的操作系统，因此可以做到秒级、甚⾄毫秒级的启动时间。⼤⼤的节约了开发、测试、部署的时间。

* ⼀致的运⾏环境

开发过程中⼀个常见的问题是环境⼀致性问题。由于开发环境、测试环境、

⽣产环境不⼀致，导致有些 bug 并未在开发过程中被发现。⽽ Docker 的镜像提供了除内核外完整的运⾏时环境，确保了应⽤运⾏环境⼀致性，从⽽不会再出现“这段代码在我机器上没问题啊”这类问题。

* 持续交付和部署

对应⽤的开发和运维 (DevOps) ⼈员来说，可以将第⼀次的服务器创建和配置⼯作保存下来，并且可以扩展到其他任何地⽅同样可以运⾏并提供服务是最好的⽅案。

使⽤ Docker 可以通过定制应⽤镜像来实现持续集成、持续交付、部署。开发⼈员可以通过 Dockerfile 来进⾏镜像构建，并结合持续集成 (Continuous

Integration) 系统进⾏集成测试，⽽运维⼈员则可以直接在⽣产环境中快速部署该镜像，甚⾄结合持续部署 (Continuous Delivery/Deployment) 系统进⾏⾃动部署。

⽽且使⽤ Dockerfile 使镜像构建透明化，不仅仅开发团队可以理解应⽤运⾏环境，也⽅便运维团队理解应⽤运⾏所需条件，帮助更好的⽣产环境中部署该镜像。

* 更轻松的迁移

由于Docker 确保了执⾏环境的⼀致性，使得应⽤的迁移更加容易。Docker 除了可以在我们的物理服务器中运⾏外，还可以在云资源服务器以及虚拟机中运⾏，甚⾄是笔记本, 其运⾏结果是⼀致的。因此⽤户可以很轻易的将在⼀个平台上运⾏的应⽤，迁移到另⼀个平台上，⽽不⽤担⼼运⾏环境的变化导致应⽤⽆法正常运⾏的情况。

* 更轻松的维护和扩展

Docker 使⽤的分层存储以及镜像的技术，使得应⽤重复部分的复⽤更为容

易，也使得应⽤的维护更新更加简单，基于基础镜像进⼀步扩展镜像也变得

⾮常简单。此外，Docker 团队同各个开源项⽬团队⼀起维护了⼀⼤批⾼质量的官⽅镜像，既可以直接在⽣产环境使⽤，又可以作为基础进⼀步定制，⼤

⼤的降低了应⽤服务的镜像制作成本[[31](#_bookmark116)]。

* + 对⽐传统虚拟机总结

|  |  |  |
| --- | --- | --- |
| **对比参数** | **容器技术** | **虚拟机技术** |
| 启动时间 | 秒级 | 分钟级 |
| 硬盘使⽤ | ⼀般为 MB | ⼀般为 GB |
| 应⽤性能 | 接近原⽣ | 弱于 |
| 系统⽀持量 | 单机⽀持上千个容器 | ⼀般⼏⼗个 |

* + 1. 使⽤ Docker Compose 管理 Docker 容器

在Docker 安装完成后，可以通过 docker pull 命令下载相应的镜像，通过 ducker run 运⾏镜像，⽣成⼀个运⾏中的容器。除了通过 ducker run 命令运⾏容器以外， 还可以通过 Docker Compose ⼯具来快速在集群中部署分布式应⽤以及容器的管理

⼯作。

使⽤ Docker Compose 项⽬是 Docker 官⽅的开源项⽬，负责实现对 Docker 容器集群的快速编排。它的定位是“定义和运⾏多个 Docker 容器的应⽤（Defining and running multi-container Docker applications），它允许⽤户通过⼀个单独的 docker- compose.yml 模板⽂件（YAML 格式）来定义⼀组相关联的应⽤容器为⼀个项⽬

（project）。

Compose 中有两个重要的概念：

* 服务（service）：⼀个应⽤的容器，实际上可以包括若⼲运⾏相同镜像的容器实例。
* 项⽬ (project)：由⼀组关联的应⽤容器组成的⼀个完整业务单元，在 docker- compose.yml ⽂件中定义。

Compose 的默认管理对象是项⽬，通过⼦命令对项⽬中的⼀组容器进⾏便捷地⽣命周期管理。

Compose 项⽬由 Python 编写，实现上调⽤了 Docker 服务提供的API 来对容器进⾏管理。因此，只要所操作的平台⽀持 Docker API，就可以在其上利⽤ Compose 来进⾏编排管理。

对于 Mysql 可以通过修改 docker-compose.yml ⽂件来实现数据库镜像的运⾏以及容器的相关控制：

version: ’2’ services:

mysql\_master:

image: docker.io/mysql restart: always container\_name: mysql-master ports:

- *" 3306 : 3306 "*

volumes:

* ./master/conf:/etc/mysql/conf.d
* ./master/data:/var/lib/mysql
* /etc/localtime:/etc/localtime:ro environment:

MYSQL\_ROOT\_PASSWORD: *" ∗∗∗∗∗∗∗∗∗"*

mysql\_slave:

image: docker.io/mysql restart: always container\_name: mysql-slave ports:

- *" 3307 : 3306 "*

volumes:

* ./slave/conf:/etc/mysql/conf.d
* ./slvae/data:/var/lib/mysql
* /etc/localtime:/etc/localtime:ro environment:

MYSQL\_ROOT\_PASSWORD: *" ∗∗∗∗∗∗∗∗∗"*

上述配置⽂件通过运⾏docker.io/mysql 镜像⽣成两个mysql 容器，其中⼀个的端⼜为 3306，另⼀个的端⼜为 3307，这样能够很⽅⾯的创建两个容器。同样，根据这个⽅法也可以创建多个相互关联的容器，⽐如 Tomcat 和 Mysql 的相互关联。

* + 1. 应⽤容器化现状

根据现阶段的项⽬开发和服务器运维需求，已经在⽣产环境的两个数据库服务器中通过 Docker 实现了 Mysql 数据库容器的运⾏，在⽣产环境的两个应⽤服务器中实现了 Couchbase 应⽤的容器化，在测试环境的服务器中实现了测试环境数据库、测试环境Couchbase、⽣产环境服务器Couchbase 节点、⽣产环境延时备份数据库的容器化。这样，当部署⼀个新的服务器节点的时候，可以通过 Docker 容器快速部署数据库应⽤、Couchbase 应⽤。

⽬前考虑到 Tomcat 在运⾏过程中需要⾯临频繁修改的配置⽂件、库⽂件以及其他⼯具的配置等问题，Docker Hub 中官⽅的 Tomcat 镜像⽆法满⾜项⽬开发的实际需求，暂时没有实现 Tomat 的容器化。为了后期 Tomcat 的容器化，⽬前也正在预研通过 Dockerfile 定制镜像的⽅式定制符合项⽬需求的 Tomcat 镜像，预研⼯作完成后即进⾏ Tomcat 的容器话需求。

由于 Docker Hub 的官⽅镜像服务器在国外，在本地服务器中通过 docker pull 命令下载镜像的速度太慢，难以满⾜快速部署的需求，故在⽣产环境的⼀个服务器节点中部署了本地的 Docker 镜像源，后期在新节点可以通过 docker pull 本地的镜像源来下载官⽅的以及定制的 docker 镜像。

* 1. SLB **负载均衡优化**

随着⽤户访问的增加以及服务⾼可⽤的需求，单个应⽤节点⽆法满⾜项⽬的需求，因此需要通过增加应⽤服务器节点来实现 WEB 应⽤的⾼可⽤，并且通过负载均衡将⽤户的请求转发到不同的服务器，降低单个服务器节点的压⼒。

服务负载均衡 (Server Load Balancing) 是在计算机⽹络的发展过程中新⽣的⼀项技术，该技术通过在多个服务器节点或其它互联⽹资源中动态的分配负载，将应⽤的压⼒转发到各个服务器节点中，实现对于资源的使⽤和⽹络吞吐率的最佳分配，同时也在⼀定程度上解决了服务器过载的问题[[32](#_bookmark117)]。

负载均衡主要应⽤在⽤户访问量⽐较⼤的 Web ⽹站、流量很⾼的⽂件访问和下载应⽤以及DNS 服务中。随着负载均衡的突出表现，现在的⼀些数据库服务也可以实现负载均衡了，实现了数据的稳定性和⾼可⽤性。负载均衡本⾝是⼀个软件或服务，它负责监听访问服务器的外部地址和端⼜，将访问该端⼜的请求转发到后台的内⽹服务器节点，服务器将收到的请求进⾏处理后将结果或响应返回到负载均衡服务，负载均衡将收到的信息返回给⽤户[[33](#_bookmark118)]。通过这种⽅式，增强了应

⽤的安全性，具体流程如图[3-13](#_bookmark54)所⽰。

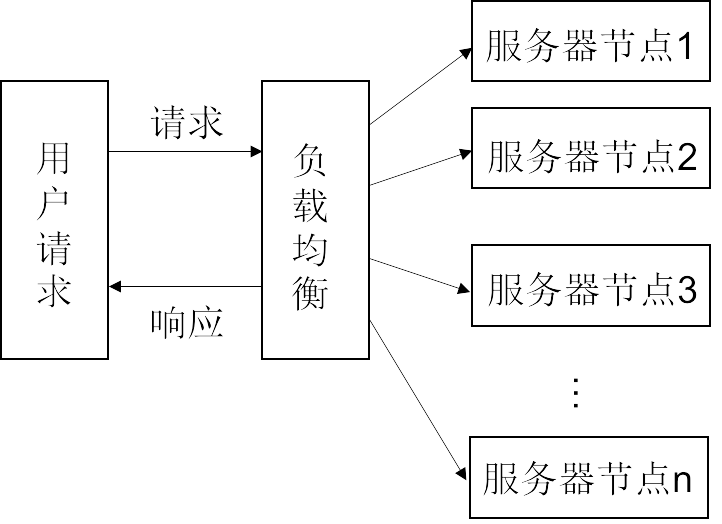


图 3-13 负载均衡流程

负载均衡的主要特点有：

* + 1. 通过负载均衡可以动态的增加或者减少提供服务的服务器的数量，可以更佳灵活的为应⽤提供扩展⽀持;
    2. 通过多服务器同时提供服务，在⾼并发的情况下将⽤户请求分发到各个服务器节点中，增加了应⽤应对并发的能⼒，提⾼了处理的性能;
    3. 通过负载均衡可以对⽤户的请求进⾏⼀定程度的过滤，对于恶意的访问和攻击可以通过⿊名单等⽅式进⾏处理，增加了应⽤的安全性;
    4. 在多节点情况下，当⼀个服务器节点损坏后依旧可以保证服务的正常运⾏， 实现了应⽤的⾼可⽤性;

考虑到本项⽬中的所有服务器节点均为阿⾥云云服务器，故选择通过使⽤阿

⾥云的负载均衡服务器服务 (SLB) 来进⾏应⽤的负载均衡，通过创建⼀个负载均衡服务，即可获取到⼀个公⽹的 IP，通过配置监听，将 WEB 应⽤的http 协议端⼜

(80) 和 https(443) 协议分别转发到⽣产环境各应⽤服务器节点的 Tomcat 对应端⼜， 通过配置服务器节点的权重来决定负载均衡在转发⽤户请求时向后端服务器转发的⽐重。

## 本章总结

本章主要是对 WEB 应⽤的应⽤性能进⾏优化，主要包括通过配置 Couchbase 缓存机制，将应⽤的基础数据加载到缓存中，在提升应⽤的相应速度的同时降低了数据库的压⼒；通过调整 Tomcat 的请求处理⽅式为 APR 模式，提升 Tomcat 对于⾼并发的处理能⼒；通过 Docker 容器编排技术将应⽤通过容器的⽅式运⾏在服务器中，实现了节点快速部署应⽤的能⼒；通过配置应⽤负载均衡，在提升服务的⾼可⽤性的同时降低了服务器节点的压⼒。

# 数据优化

在 WEB 应⽤的开发和使⽤过程中，数据对于应⽤的价值越来越重要，因此在应⽤的使⽤过程中，保证数据库的正常⼯作以及数据的完整性将成为开发过程中数据库优化的主要⽬标。

⽬前应⽤开发过程中使⽤的数据库是 MySQL，MySQL 是⼀个开源的关系数据库管理系统，通过关系模型为⽤户提供数据的存储和修改等操作。

最新的稳定版为 5.7.17，项⽬所使⽤的MySQL 版本为 5.7.11。较之前的版本，

* 1. 版本的主要改进包括：
     1. 提升安全性

为了增强数据库数据的安全性，在完成 MySQL 安装时，默认的 root 密码不再为空，⽽是随机⽣成⼀个密码，⽤户可以通过随机密码登录 MySQL 后修改密码。除此之外，新版本删除了 test 数据库，并且对⽤户创建的 test 数据库的权限进⾏了控制，同时提供了更为简单 SSL 安全访问配置，对于⽤户的密码可以设置有效期策略，超过有效期是强制⽤户修改密码来提升数据库安全，新版本还新增了对⽤户的暂时禁⽤功能。

* + 1. 增强数据存储的灵活性

新版本在提升数据存储的灵活性⽅⾯增加了 JSON 和generate column 两个新功能。

随着⾮结构化数据存储需求的持续增长，各种⾮结构化数据存储的数据库应运⽽⽣（如 MongoDB）。从最新的数据库使⽤排⾏榜来看，MongoDB 已经超过了PostgreSQL，其⽕热程度可见⼀斑。各⼤关系型数据库也不⽢⽰弱，纷 纷提供对 JSON 的⽀持，以应对⾮结构化数据库的挑战。MySQL 数据库从

* + 1. 版本开始，也提供了对 JSON 的⽀持。使⽤⽅式如下：

**CREATE TABLE** t1 (jdoc JSON);

**INSERT INTO** t1 **VALUES**( *’ {" key 1 " : " value 1 " , " key 2 " : "*

*value 2 "} ’* );

MySQL 对⽀持 JSON 的做法是，在 server 层提供了⼀堆便于操作 JSON 的函数，⾄于存储，就是简单地将 JSON 编码成 BLOB，然后交由存储引擎层进

⾏处理，也就是说，MySQL 5.7 的 JSON ⽀持与存储引擎没有关系，MyISAM

存储引擎也⽀持 JSON 格式。

MySQL ⽀持 JSON 以后，总是避免不了拿来与MongoDB 进⾏⼀些⽐较。但

是，MySQL 对 JSON 的⽀持，⾄少有两点能够完胜 MongoDB：

* + - 1. 可以混合存储结构化数据和⾮结构化数据，同时拥有关系型数据库和⾮关系型数据库的优点
      2. 能够提供完整的事务⽀持

generated column 是 MySQL 5.7 引⼊的新特性，所谓 generated column，就是数据库中这⼀列由其他列计算⽽得。

* + 1. 提升数据库运⾏的易⽤性

在开发或者运维⼈员进⾏数据库使⽤和状态检查的过程中，MySQL 5.7 可以 explain ⼀个正在运⾏的 SQL，这对于 DBA 分析运⾏时间较长的语句将会⾮常有⽤，同时 performance\_schema 提供了更多监控信息，包括内存使⽤， MDL 锁，存储过程等。

除此之外，MySQL 5.7.7 中引⼊了⼀个系统库 sys schema，它包含了⼀系列视图、函数和存储过程，该项⽬专注于 MySQL 的易⽤性。例如，我们可以通过 sys schema 快速的知道，哪些语句使⽤了临时表，哪个⽤户请求了最多的 io，哪个线程占⽤了最多的内存，哪些索引是⽆⽤索引等

sys schema 中包含了⼤量的视图，那么，这些视图的信息来⾃哪⾥呢？视图中的信息均来⾃performance schema 统计信息。也就是说，performance schema 提供了信息源，但是，没有很好的将这些信息组织成有⽤的信息，从⽽没有很好的发挥它们的作⽤。⽽ sys schema 使⽤ performance schema 信息，通过视图的⽅式给出解决实际问题的答案。

例如，下⾯这些问题，在MySQL 5.7 之前，需要借助外部⼯具才能知道，在MySQL 5.7 中，直接查询 sys 库下相应的表就能得到答案：

# 如何查看数据库中的冗余索引

**select from** sys.schema\_redundant\_indexes;

\*

# 如何获取未使用的索引

**select from** schema\_unused\_indexes;

\*

# 如何查看使用全表扫描的SQL语句

**select from** statements\_with\_full\_table\_scans

\*

* + 1. 提升了数据库的可⽤性

在以往的版本中，许多数据库的设置修改都需要重启服务使配置⽣效，在 5.7 版本中增加了许多改进，可以时⼀些必要的配置⽆需重启服务即可⽣效。 在线设置复制的过滤规则不再需要重启MySQL，只需要停⽌ SQL thread，修改完成以后，启动 SQL thread。

在线开启GTID，在之前的版本中，由于不⽀持在线开启 GTID，⽤户如果希望将低版本的数据库升级到⽀持 GTID 的数据库版本，需要先关闭数据库， 再以 GTID 模式启动，所以导致升级起来特别⿇烦。MySQL 5.7 以后，这个问题不复存在。

在线修改 buﬀer pool 的⼤⼩，MySQL 5.7 为了⽀持 online buﬀer pool re-

size，引⼊ chunk 的概念，每个 chunk 默认是 128M，当我们在线修改 buﬀer pool 的时候，以 chunk 为单位进⾏增长或收缩。这个参数的引⼊，对 inn- odb\_buﬀer\_pool\_size 的配置有了⼀定的影响。innodb 要求 buﬀer pool size 是innodb\_buﬀer\_pool\_chunk\_size\* innodb\_buﬀer\_pool\_instances 的倍数，如果不是，将会适当调⼤innodb\_buﬀer\_pool\_size，以满⾜要求，因此，可能会出现buﬀer pool 的实际分配⽐配置⽂件中指定的 size 要⼤的情况.

Online DDL MySQL 5.7 ⽀持重命名索引和修改 varchar 的⼤⼩，这两项操作在之前的版本中，都需要重建索引或表.

**ALTER TABLE** t1 ALGORITHM=INPLACE, CHANGE **COLUMN** c1 c1

**VARCHAR**(255);

* + 1. 性能相关的改进
       - 只读事务性能改进

众所周知，在传统的OLTP 应⽤中，读操作远多于写操作，并且，读操作不会对数据库进⾏修改，如果是⾮锁定读，读操作也不需要进⾏加锁。因此，对只读事务进⾏优化，是⼀个不错的选择。

MySQL 5.6 中，已经对只读事务进⾏了许多优化。例如，将 MySQL 内部实现中的事务链表分为只读事务链表和普通事务链表，这样在创建ReadView 的时候，需要遍历事务链表长度就会⼩很多。

在MySQL 5.7 中，⾸先假设⼀个事务是⼀个只读事务，只有在该事务发起了修改操作时，才会将其转换为⼀个普通事务。MySQL 5.7 通过避免为只读事务分配事务 ID，不为只读事务分配回滚段，减少锁竞争等多种

⽅式，优化了只读事务的开销，提⾼了数据库的整体性能。

* + - * 加速连接处理

在MySQL 5.7 之前，变量的初始化操作（THD、VIO）都是在连接接收线程⾥⾯完成的，现在将这些⼯作下发给⼯作线程，以减少连接接收线程的⼯作量，提⾼连接的处理速度。这个优化对那些频繁建⽴短连接的应⽤，将会⾮常有⽤。

* + - * 复制性能的改进

MySQL 的复制延迟是⼀直被诟病的问题之⼀，欣喜的是，MySQL 5.7

版本已经⽀持”真正”的并⾏复制功能。MySQL 5.7 并⾏复制的思想简单易懂，简⽽⾔之，就是”⼀个组提交的事务都是可以并⾏回放的”，因为这些事务都已进⼊到事务的 prepare 阶段，则说明事务之间没有任何冲突（否则就不可能提交）。经过对⽐测试，MySQL 5.7 采⽤新的并⾏复制后，仍然会存在⼀定程度的延迟，只不过相⽐ 5.6 版本减少了 86%，相

⽐ MariaDB 的并⾏复制延迟也⼩不少。复制延迟问题得到极⼤改善。除此之外复制性能的改进还包括多源复制，多从线程增强，在线 GTIDs， 和增强的半同步复制等功能，这些功能均为保证数据的完整性可有效性提⾼了保障。

在数据库开发过程中，存储引擎对于数据库性能也⾮常重要，它是我们在进

⾏数据的存储、建⽴数据索引、数据的更新以及数据查询的内部实现⽅法[[34](#_bookmark119)]。 同 MySQL 数据库相⽐，Oracle 数据库和 SQL Server 数据库只使⽤⼀种数据

库引擎，所有的数据操作⽅法都是采⽤⼀种⽅法。MySQL 数据库为了改变这种局

⾯，为⽤户提供了多种引擎，开发者在开发过程中可以根据应⽤的⾃⾝需求为数据配置更加合适的数据库引擎。MySQL 数据库提供的存储引擎如表 [4-1](#_bookmark57)所⽰。

表 4-1 MySQL 数据库存储引擎

### 存储引擎 描述

InnoDB ⽀持事务和⾏级锁，是 Mysql 上唯⼀⼀个提供了外键约束的引擎

MyISAM 基于 ISAM 存储引擎，常⽤的引擎，但是不⽀持事务、⾏级锁、⽽且崩溃后不能保证完全恢复。

ARCHIVE 仅⽀持插⼊和查询以及索引功能，拥有很好的压缩机制，适⽤于存储⽇志信息或其他按时间序列实现的数据采集类的应⽤场景中

CSV 将数据⽂件保存为 CSV 格式的的⽂件，可以

⽅便的导⼊到其他数据库中去，但是不⽀持索引

BLACKHOLE 没有存储机制，任何数据都会被丢弃，但是会记录⼆进制⽇志

FEDERATED 可以访问远程服务器上数据的存储引擎，所以说它不再本地创建数据只会⾃动的建⽴⼀个连接到其他服务器上链接，有点类似于代理的功能

MEMORY 使⽤存储在内存中的内存来创建表，⽽且所有数据保存在内存中，数据安全性很低，但是查找和插⼊速度很快，通常⽤于临时表

MRG\_MYISAM 将多个 MyISAM 合并为⼀个

这些数据引擎中使⽤最⼴泛的是 MyISAM 和 InnoDB 两种存储引擎。前者是MySQL 数据库的默认存储引擎，后者则是第三⽅公司开发的，它跟前者相⽐，后者具有⽀持事务、⽀持⾏级锁以及⽀持外键约束等⾮常实⽤的功能[[34](#_bookmark119)]。

* 1. InnoDB **引擎参数优化**

设计 InnoDB 引擎的⽬的是解决MySQL 在处理⾮常⼤的数据量时表现出的性能不⾜。通过使⽤ InnoDB 引擎，对于服务器的 CPU 运⾏效率来说性能远远超过了其它关系型数据库引擎，因此在开发数据量⽐较⼤的应⽤时，InnoDB 引擎已经

受到了很多开发者的认可。[[35](#_bookmark120)]。由于 InnoDB 数据引擎在事务安全、⽀持外键以及数据恢复成本的综合性能表现，本论⽂中的 WEB 应⽤数据存储引擎使⽤的是

InnoDB 引擎。

为了保证在使⽤ InnoDB 引擎过程中，使 WEB 应⽤的数据稳定性和服务器负载达到最优的使⽤体验，还需要基于⽬前的 WEB 开发架构和服务器现状对MySQL 的配置进⾏⼀定的参数调优。优化主要包括内存、IO、⽇志以及其它⽅

⾯[[36](#_bookmark121)]。

* + 1. 内存利⽤⽅⾯

在内存利⽤⽅⾯，innodb\_buﬀer\_pool\_\* 相关的参数主要负载调控数据库在运

⾏过程中数据缓存到内存的数据。其中 innodb\_buﬀer\_pool\_size 是 InnoDB 最重要的⼀个配置参数，也是在进⾏ Innodb 引擎优化时第⼀个需要调整的参数，它的主要作⽤是对 Innodb 的索引进⾏缓存。参数的默认分配只有 8M， 如果在配置过程中不调整这个值的话，会导致数据库的性能体验过差。

除此之外，还可以通过调整 innodb\_buﬀer\_pool\_instances 来修改数据库缓冲池的实例数量，通过开启多个内存缓冲池，把需要缓冲的数据 hash 到不同的缓冲池中，这样可以并⾏的内存读写，在⾼ IO 负载时保持⾮常稳定的吞吐。

⼀般来说 pool size 参数和 pool instance 参数之前相互配置，通过不断的测试来调优⼆者的值，最终使数据库的内存利⽤⽅⾯达到最⾼。

* + 1. IO 控制⽅⾯

对于数据库的空间占⽤，可以通过修改innodb\_file\_\* 参数来配置，考虑到阿

⾥云的磁盘扩展和读取的性能，这个参数的配置对于数据库的的性能提升效果不明显，因此没有必要做配置。

对于数据的 IO 分配来说， 需要进⾏⼀定的优化， 可以通过 inn- odb\_file\_format 配置⽂件的格式，提升存储数据的压缩⽐，可以通过修改 inn- odb\_thread\_concurrency 参数来配置线程的并发数，提升效率。

* + 1. ⽇志控制⽅⾯

通过修改 innodb\_log\_file\_size 参数可以调整每个⽇志⽂件的⼤⼩，每个⽇志

⽂件的⼤⼩对于服务器磁盘的写⼊和异常恢复后的恢复效率有⾮常⼤的影响，因此需要合理配置该参数值，由于当前的数据库服务器是阿⾥云数据库， 硬盘为⾼速硬盘，数据的存取效率和 IO 压⼒均表现良好，故这个参数对于当前项⽬来说没有优化的必要。

通过修改 innodb\_log\_buﬀer\_size 参数可以⽇志缓冲区的⼤⼩，这个值分配的

⼤⼩与内存中缓存的⽇志⼤⼩很⼤的关系，适当修改这个值可以降低内存的

压⼒。

除了正常的⽇志外，InnoDB 还有undo log，它记录某数据被修改前的值，可以

⽤来在事务失败时进⾏回滚，因此通过调整undo log 的相关参数可以在很⼤程度上保证数据的有效性。通过修改 innodb\_undo\_log\_truncate 参数值为 1 来开启在线回收undo ⽇志⽂件，通过修改 innodb\_max\_undo\_log\_size 参数值来配置触发回收 undo ⽇志的阈值，通过 innodb\_purge\_rseg\_truncate\_frequency 参数来配置回收 undo ⽇志的频率[[37](#_bookmark122)]。

* + 1. 其它⽅⾯优化

除了以上三个⽅⾯的优化之外，还有许多优化项⽬可以应⽤于本项⽬的数据库优化中。考虑到阿⾥云磁盘的性能，可以通过 innodb\_flush\_method 参数将数据和⽇志⽂件不经过缓存直接写⼊到⽂件中；通过 innodb\_strict\_mode 参数开启严格模式，对于写法错误的 SQL 语句跳过警告直接提⽰错误，提升数据库的稳定性；通过脏页的相关配置参数调整数据库对于脏页的刷新⽅式和效率，提升数据的有效性。

综合以上各个⽅⾯的优化之后，本论⽂中 WEB 应⽤的数据库 InnoDB 引擎优化参数为：

# 缓冲池字节大小

innodb\_buffer\_pool\_size = 800M

# 缓冲池实例数量

innodb\_buffer\_pool\_instances = 8

# 启动时将热数据加载到内存

innodb\_buffer\_pool\_load\_at\_startup = 1

# 关闭时将热数据dump到本地磁盘

innodb\_buffer\_pool\_dump\_at\_shutdown = 1

# page cleaner线程每次刷新脏页的数量 innodb\_lru\_scan\_depth = 2000

# 事务等待获取资源等待的最长时间

innodb\_lock\_wait\_timeout = 5

# 调整刷新脏页的数量

innodb\_io\_capacity = 4000

# 刷新脏页的最大值

innodb\_io\_capacity\_max = 8000

# 数据和日志写入磁盘的方式-直接写入磁盘

innodb\_flush\_method = O\_DIRECT

# 文件格式， Barracuda支持压缩页， 新格式

innodb\_file\_format = Barracuda

# 设置文件格式最高版本

innodb\_file\_format\_max = Barracuda

# 刷新脏页临近页

innodb\_flush\_neighbors = 1

# 用来缓冲日志数据的缓冲区大小

innodb\_log\_buffer\_size = 1M

# 单独的清除线程数量-0不适用单独线程

innodb\_purge\_threads = 4

# 为字段创建索引时， 限制的字节长度， 超过直接报错

innodb\_large\_prefix = 1

# 线程并发数

innodb\_thread\_concurrency = 64

# 将发生的所有死锁信息都记录到错误日志中

innodb\_print\_all\_deadlocks = 1

# 严格检查模式， 写法有错误直接报错， 不警告

innodb\_strict\_mode = 1

# 建立索引时用于排序数据的排序缓冲区大小-10M innodb\_sort\_buffer\_size = 10485760

# 转储缓冲池中**read** out **and** dump 的最近使用的页的占比

innodb\_buffer\_pool\_dump\_pct = 40

# page cleaner线程数量innodb\_page\_cleaners = 4

# 开启在线回收undo log日志文件

innodb\_undo\_log\_truncate = 1

# 超过这个阈值时触发回收

innodb\_max\_undo\_log\_size = 2G

# 回收undo日志的频率

innodb\_purge\_rseg\_truncate\_frequency = 128

* 1. **主从复制和延迟复制优化**

除了通过 InnoDB 引擎的配置来保证数据的有效性和稳定性之外，在数据库的开发是使⽤过程中还可以通过配置主从复制和延迟复制来保证数据。在 5.7 以前的MySQL 版本中，由于主从复制的延迟问题，通常会选择第三⽅⼯具来进⾏数据的同步，但是通过第三⽅⼯具，对于数据同步的问题性由带来了问题，这些问题⼀直是运维⼈员在数据库开发过程中⽐较头疼的问题。随着 5.7 版本的发布，新版本在数据库主从复制⽅⾯进⾏了很多改进，包括降低了复制的延迟，通过 looseless 半同步⽅式提升了复制的稳定性[[38](#_bookmark123)]。

MySQL 数据库复制技术是把数据从⼀个数据库节点拷贝到其它⼀个或者多个数据库节点中，前者通常被称为主库 (Master)，后者通常被称为从库 (Slave)，如图[4-1](#_bookmark60)所⽰。

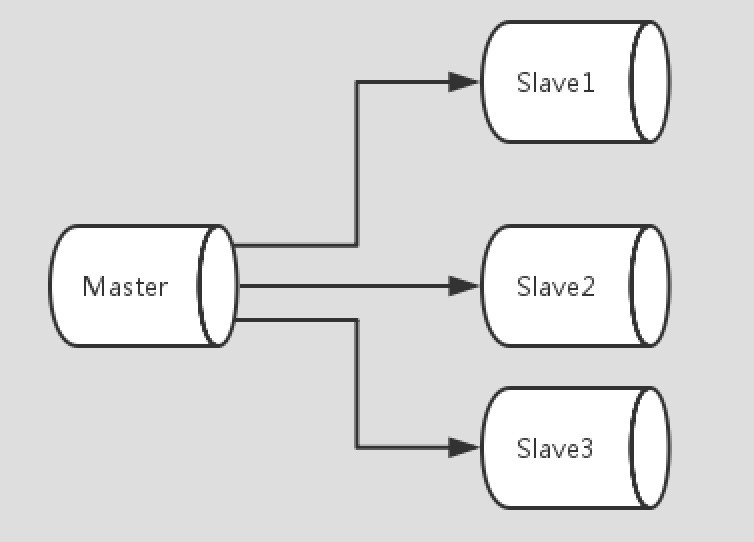


图 4-1 MySQL 复制⽰意图

复制的结果是集群（Cluster）中的所有数据库服务器得到的数据理论上都是

⼀样的，都是同⼀份数据，只是有多个copy。MySQL 默认内建的复制策略是异步的，基于不同的配置可以调整复制策略，Slave 不⼀定要⼀直和Master 保持连接不断的复制或等待复制，我们可以指定复制所有的数据库，⼀部分数据库，或者是某个数据库的某部分的表。

MySQL 复制⽀持多种不同的复制策略，包括同步、半同步、异步和延迟策略等。

* + 1. 同步策略：Master 要等待所有 Slave 应答之后才会提交（MySql 对 DB 操作的提交通常是先对操作事件进⾏⼆进制⽇志⽂件写⼊然后再进⾏提交）。
    2. 半同步策略：Master 等待⾄少⼀个 Slave 应答就可以提交。
    3. 异步策略：Master 不需要等待 Slave 应答就可以提交。
    4. 延迟策略：Slave 要⾄少落后 Master 指定的时间。

MySQL 复制同时⽀持多种不同的复制模式：

1. 基于语句的复制，Statement Based Replication（SBR）。
2. 基于⾏的复制 Row Based Replication（RBR）。
3. 混合复制（Mixed）。

使⽤ MySQL 复制，对于系统性能的提升主要表现在以下⼏个⽅⾯：

1. 性能⽅⾯：MySQL 复制是⼀种 Scale-out ⽅案，也即“⽔平扩展”，将原来的单点负载扩散到多台 Slave 机器中去，从⽽提⾼总体的服务性能。在这种⽅式下，所有的写操作，当然包括 UPDATE 操作，都要发⽣在 Master 服务器上。读操作发⽣在⼀台或者多台 Slave 机器上。这种模型可以在⼀定程度上提⾼总体的服务性能，Master 服务器专注于写和更新操作，Slave 服务器专注于读操作，我们同时可以通过增加 Slave 服务器的数量来提⾼读服务的性能。
2. 防腐化：由于数据被复制到了 Slave，Slave 可以暂停复制进程，进⾏数据备份，因此可以防⽌数据腐化。
3. 故障恢复：同时多台Slave 如果有⼀台Slave 挂掉之后我们还可以从其他Slave

读取，如果配置了主从切换的话，当 Master 挂掉之后我们还可以选择⼀台

Slave 作为 Master 继续提供写服务，这⼤⼤增加了应⽤的可靠性。

1. 数据分析：实时数据可以存储在 Master，⽽数据分析可以从 Slave 读取，这样不会影响 Master 的性能。
   * 1. 数据库复制流程

MySQL 复制最常⽤的复制⽅式是通过⼆进制⽂件的⽅式进⾏复制，因此在配置数据库复制时，需要在主服务器和从服务器中开启⼆进制⽇志的配置。

数据库的复制过程主要可以分为三步:

* + - 1. 当主数据库的数据发⽣改变时，主数据库会将数据库的数据记录的事务过程写到已经开启的⼆进制⽇志⽂件中；
      2. 主库将事务写如⼆进制⽂件后会通知从库数据，此时从数据库会链接主数据库将改变的事务⽇志下载到从数据库的中继⽇志中；
      3. 从数据库对于中继⽇志中的新事务进⾏重放，根据⽇志操作⽅式修改从数据库的对应数据，保证数据的⼀致[[39](#_bookmark124)]。

图 [4-2](#_bookmark62)描述了复制的过程

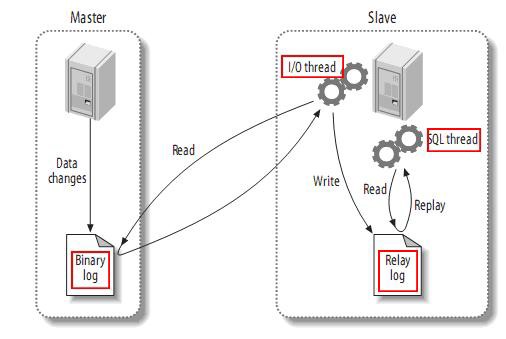


图 4-2 MySQL 复制流程

* + 1. 双主复制设计

⽬前⽣产环境的数据库有两个专有的服务器提供服务，为了保证数据的有效性和⾼可⽤性，两个服务器在同⼀时间只有⼀个提供数据业务服务，扮演主要的数据服务器⾓⾊，另外⼀个服务器复制主服务器的数据确保数据完整，扮演备服务器或从服务器⾓⾊。但是当主服务器出现问题⽽宕机时，需要快速的将从数据库切换为主数据库，在问题服务器恢复正常时作为从数据库，从新的主服务器复制数据。根据这个需求，将⽬前的两个数据库配置为双主数据库，将两个主数据库标⽰为 master1 和 master2。

为了保证数据库的顺利复制，⾸先需要在两个数据库中通过 stop slave 命令来停⽌复制，并且保持两者的数据完全⼀致，通过 reset master 以及 reset slave 命令初始化。

然后在MySQl 的配置⽂件中开启⼆进制⽇志，并为每⼀个数据库配置⼀个唯

⼀的 server-id，以及必要的配置。

# 服 务 器 ID server-id = 1

# 开启二进制日志并配置日志名

log\_bin = db2.bin

# 每一次事物提交都将binlog\_cache中的数据强制写到磁盘

sync\_binlog = 1

根据如上配置项，将两个数据库的⼆进制⽇志的⽂件名均配置为 db2.bin，master1 的 server-id 为 1，master2 的 server-id 为 2，配置 sync\_binlog 参数对 MySQL 数据库的运⾏性能和数据完整性⾮常重要，参数值设置为 1 表⽰在数据操作过程中我们每提交⼀次事务，数据库就会进⾏⼀次磁盘同步，将缓存中的数据写⼊磁盘中, 提⾼数据的持久性。

除此之外，需要在配置⽂件中开启 GTID

# 开启gtid工作模式

gtid\_mode = **on**

# 只允许能保障事物安全， 且能够被日志记录的SQL语句被执行

enforce\_gtid\_consistency = 1

# 从库从主库复制数据时的操作也写入binlog log\_slave\_updates

# 重启和启动时， 如何迭代使用binlog文件

binlog\_gtid\_simple\_recovery = 1

完成基本配置后， 需要在数据库中配置和连接master。

GTID 是全局事务标识 (global transaction identifieds)，在数据库中⼀个事务对应⼀个GTID，⽽且⼀个 GTID 在⼀个服务器上只执⾏⼀次，避免重复执⾏导致数据混乱或者主从不⼀致，通过 GTID 可以保证⽇志⽂件中每⼀次事务都对应⼀个唯⼀的标志，对于从库拉取⽇志和⽇志分析具有很重要的意义。

* + - 1. 创建⽤于主从复制的⽤户

**GRANT** REPLICATION SLAVE **ON**

\*.\*

TO *’ r e p l ’* @ *’%’*

IDENTIFIED **BY** *’ r eplpassword ’* ;

其中 repl 为⽤户名，replpassword 为密码

* + - 1. 配置 master 信息

change master to master\_host = *’ i p ’* , master\_port = port

, master\_user = *’ r e p l ’* , master\_password = *’ r eplpassword ’* , master\_auto\_position =1;

变量 master\_host 值为 master 所在服务器的 IP 地址，master\_port 为 master 服务器的数据库连接端⼜，配置好⽤户名和密码。master\_auto\_position 让从库根据 GTID ⾃动选择适当的事务点进⾏复制，基本上⽆需关注和担⼼主从不

⼀致的问题。

* + - 1. 启动复制，并查看状态。

start slave；

show slave status；

在查询结果的字段中通过 Slave\_IO\_Running 和 Slave\_SQL\_Running 两个变量的值为 YES 来判断主从复制是否成功启动。

在 master1 和 master2 上均按照上述步骤进⾏操作，完成⾼可⽤的双主复制的部署。

* + 1. 延迟复制设计

为了避免上述服务器同时出现问题后⽆法提供数据服务的现象，论⽂考虑在本系统所使⽤的测试服务器中搭建⼀个延时复制服务器，延时复制的主库配置为master1，将复制策略配置为延迟 1 ⼩时复制，然后通过⼆进制⽇志进⾏近⼩时内数据的恢复，这样能够最⼤程度保证数据的完整。

配置的步骤基本类似于双主的配置，但是在配置 master 信息后需要增加⼀步， 调整复制的延迟时间，单位是 ms，因此⼀⼩时延时的变量值为 3600。

CHANGE MASTER TO MASTER\_DELAY = 3600;

## 数据库备份

除了通过延迟复制来保证数据之外，服务器还将每天对数据库进⾏⼀次备份， 并且将备份的数据库上传到阿⾥云的对象存储OSS 中，⽬前 master1 和 master2 数据库的数据是⼀致的，因此只需要对 master1 的数据库进⾏备份即可。

备份通过mysqldump 命令进⾏备份，备份完成之后通过阿⾥云OSS 的 Python

SDK 将备份⽂件上传到 OSS 中。

* + 1. 开发 OSS ⽂件上传脚本

⾸先需要通过 pip install oss2 命令在备份服务器中安装 Python 版本的 OSS

SDK，安装完成后在服务器的/mnt/sh 中新建 oss ⽬录，并在 oss ⽬录下创建上传脚本 dbuposs.py ⽂件。修改⽂件如下：

1

#!/usr/bin/python2.7

# -\*- coding: utf-8 -\*-

**import** oss2

**import** sys

auth = oss2.Auth( *’ Access Key Id ’* , *’ Access Key Secret ’* ) endpoint= *’ h t t p : / / oss* −*cn* − *b e i j i n g* − *i n t e r n a l . a l i y u n c s . com ’* bucket = oss2.Bucket(auth, endpoint, *’ mysqlbk ’* ) bucket.put\_object\_from\_file(sys.argv[1], *’ / mnt /*

*mysqldump / ’* +sys.argv[1])

2

3

4

5

6

7

8

在脚本中配置阿⾥云 OSS 的访问域名 (本项⽬的 OSS 访问域名为 [http://oss-](http://oss-/) cn-beijing-internal.aliyuncs.com)，以及访问 OSS 的AccessKeyId 和AccessKey-

Secret。配置完成之后即可完成访问 OSS 的认证⼯作。

通过 oss2.Bucket 函数连接 OSS 的 mysqlbk 存储空间获取 bucket 对象。

通过bucket 对象的put\_object\_from\_file 函数将指定的⽂件上传到OSS 中，完成⽂件上传。

* + 1. 开发数据库备份脚本

数据库脚本可以通过 Bash 脚本来实现，前提是需要在服务器中安装 mysql， 脚本的存放位置为/mnt/sh/mysqlbak.sh。

备份的流程主要为：⾸先要配置被备份数据库的相关信息，包括⽤户名、密码、端⼜、IP 地址、数据库名称等变量；然后配置备份数据的保存位置和命名⽅式，以及⽇志的相关配置；之后通过执⾏ mysqldump 备份数据库到本地， 通过 tar 命令将备份的 sql ⽂件压缩；最后通过调⽤ OSS 上传脚本将压缩后的备份⽂件上传到OSS 中。同时，本地⽬录中只保存七天内的数据库备份⽂件，超过七天的⽂件会被删除。

备份脚本如下：

1

#!/bin/bash PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/**local**/bin:/usr

/**local**/sbin

**export** PATH

#数据库用户名dbuser=’root’

#数据库用密码

dbpasswd=’password’

#需要备份的数据库， 多个数据库用空格分开dbname=’minishop’

#备份时间

backtime= (date +%Y%m%d%H%M%S)

#日志备份路径logpath=’/mnt/mysqldump’

#数据备份路径

datapath=’/mnt/mysqldump’

#日志记录头部

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

**echo** *"* 备份时间为*$ { backtime } ,* 备份数据库表 *$ { dbname }* 开始 *"* >> ${logpath}/log.log

#正式备份数据库

**for** dbn **in** $dbname; **do**

**source**=(mysqldump -u${dbuser} -p${dbpasswd} -hdbip - Pdbport ${dbn} --**set**-gtid-purged=OFF > ${logpath}/${ backtime}.sql) 2>> ${logpath}/mysqllog.log;

#备份成功以下操作

**if** [ *"$? "* == 0 ];**then cd** $datapath

#为节约硬盘空间， 将数据库压缩

tar jcf ${dbn}${backtime}.tar.bz2 ${backtime}.sql > / dev/null

#删除原始文件， 只留压缩后文件

rm -f ${datapath}/${backtime}.sql

#上传到oss

/mnt/sh/oss/dbuposs.py ${dbn}${backtime}.tar.bz2

#删除七天前备份， 也就是只保存7天内的备份

find $datapath -name *" ∗ . t a r . bz 2 "* -**type** f -mtime +7 -

**exec** rm -rf {} \; > /dev/null 2>&1

**echo** *"* 数据库表 *$ { dbname }* 备份成功 *! ! "* >> ${logpath}/ mysqllog.log

**else**

#备份失败则进行以下操作

**echo** *"* 数据库表 *$ { dbname }* 备份失败 *! ! "* >> ${logpath}/ mysqllog.log

**fi**

**done**

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

* + 1. 定时任务设置为了保证每天都进⾏⼀次数据库备份操作，需要通过定时任务来保证数据库备份脚本每天执⾏⼀次。在 Linux 环境中，通过 crontab 软件来执⾏定时任务。crontab 是⼀个在类 Unix 操作系统上的任务计划程序，它可以让⽤户在指定时间段周期性地运⾏命令或者 shell 脚本，通常被⽤在系统的⾃动化维护或者管理。

在本项⽬中，cron 定时任务的配置⽂件在/var/spool/cron 中的 root ⽂件，在

root ⽂件中添加如下定时任务：

#< 分钟> < 小时> <日> < 月份> < 星期> < 命令>

00 00

\*

\* \*

/mnt/sh/mysqlbak.sh

设定在每天的凌晨 0 点 0 份执⾏数据库备份脚本。

## 本章总结

本章是对本论⽂中 WEB 项⽬的数据库进⾏优化的⼀些策略，主要包括通过

InnoDB 配置参数调整提升数据库⾃⾝的运⾏性能，通过配置主从复制以及延迟复制提升数据的稳定性和有效性，通过开发数据库定时备份脚本来实现数据库的每

⽇备份。通过这些策略，在提升 MySQL 运⾏性能同时也保证了数据的完整性和有效性，对于 WEB 应⽤的作⽤⾄关重要。

# 服务监控与应急措施优化

在本论⽂的第三章和第四章分别对应⽤性能⽅⾯和数据库⽅⾯进⾏了优化， 在系统优化的过程中，除了提升应⽤性能和数据库性能之外，对于服务器中运⾏的服务进⾏监控检查和报警，并且能够实现⾃动化的 Failover 也是系统优化的重点，本章将对于⽬前项⽬开发过程中使⽤的⼯具、软件设计开发相应的监控脚本， 并且探索在服务监控过程中的报警和故障恢复模式[[40](#_bookmark125)]。

## 阿里云云监控应用

云监控 (CloudMonitor) 服务是由阿⾥云推出的，它的⽬的是对阿⾥云上的资源和互联⽹应⽤进⾏状态的监控。通过云监控服务，可以根据⾃定义规则获取阿

⾥云资源的各项监控指标，可以通过设置服务的访问⽅式来探测服务的可⽤性，除此之外还可以对于监控指标设置报警，及时通知运维⼈员。由于本项⽬的所有服务器均为阿⾥云服务器，所以⼀部分的监控可以通过配置阿⾥云资源的监控规则对部分服务和资源状态进⾏监控，主要可以实现云服务器、云数据库和负载均衡等资源的监控，也可以实现对使⽤ HTTP 和 ICMP 这些⽐较通⽤的⽹络协议的服务进⾏监控，监控⽹络服务的可⽤性[[41](#_bookmark126)]。

在正式进⾏阿⾥云监控配置，需要对⽬前阿⾥云资源进⾏统计, 如表 [5-1](#_bookmark69)所⽰：

表 5-1 项⽬阿⾥云资源整理

**资源类别 资源实例 资源介绍**

站点

云服务器 ECS

负载均衡

⽣产环境地址 ⽣产环境应⽤的负载均衡⼊⼜地址

测试环境地址 测试环境应⽤的负载均衡⼊⼜地址

主数据库地址 ⽣产环境主数据库的访问地址和端⼜

从数据库地址 ⽣产环境从数据库的访问地址和端⼜

Couchbase 地址 ⽣产环境 Couchbase ⼊⼜地址

APP1 WEB 应⽤服务器，提供 WEB 应⽤服务

APP2 WEB 应⽤服务器，提供 WEB 应⽤服务

DB1 主数据库服务器，WEB 请求的主服务器

DB2 从数据库服务器，同 DB1 主从复制，DB1 出现问题时切换为主数据库

TEST 测试服务器，提供测试环境相关的服务

prod ⽣产环境 WEB 应⽤负载均衡

dbslb ⽣产环境数据库负载均衡

test 测试环境负载均衡

续下页

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 续表 5-1 | 项⽬阿⾥云资源整理 |
| **资源类别** | **资源实例** |  | **资源介绍** |
| CDN | web img  fileserver |  | WEB 应⽤访问域名图⽚服务器域名  ⽂档服务器域名 |

为了保证各个阿⾥云资源的正常使⽤，通过阿⾥云的云监控配置各服务的监控项，并且增加报警联系⼈，在监控到问题的时候能够通过短信或邮件的⽅式及时通知运维⼈员，以便在短时间内解决问题。

* + 1. 站点监控

站点监控通过 HTTP 协议和 TCP 协议根据设置的监控频率去检测各个站点的访问时间，以此来判断站点是否正常。

对于⽣产环境的 WEB 应⽤、测试环境的 WEB 应⽤以及⽣产环境的 Couch- base，通过HTTP 协议去访问对应的地址，对于主从数据库，通过 TCP 去测试数据库的链接是否正常。

监控的状态如图 [5-1](#_bookmark70)所⽰：



图 5-1 站点监控状态图

* + 1. 云服务器监控

云服务器的监控则通过安装在服务器中的阿⾥云监控插件获取云服务器的状态，对于所有的云服务监控的规则都是⼀样的，均为 CPU 使⽤率、内存使

⽤率和磁盘使⽤率三个⽅⾯，监控的具体规则如表 [5-2](#_bookmark71)所⽰：

表 5-2 云服务器监控规则

### 监控项 监控描述

CPU 使⽤率 5 分钟 CPU 使⽤率平均值 >85% 则报警内存使⽤率 5 分钟内存使⽤率最⼤值 >95% 则报警磁盘使⽤率 5 分钟磁盘使⽤率平均值 >70% 则报警

* + 1. 负载均衡监控负载均衡主要是监控负载均衡内的各个服务器的监控状态以及负载均衡的带宽状态，当负载均衡出现异常时，可以通知报警联系⼈及时作出应对。
       - ⽣产环境应⽤负载均衡监控规则

表 5-3 ⽣产环境应⽤负载均衡监控规则

### 监控项 监控描述

流出带宽 5 分钟流出带宽平均值 >3M/s 则报警

流⼊带宽 5 分钟流⼊带宽平均值 >3M/s 则报警 后端异常 ECS 实例数 1 分钟后端异常 ECS 实例数平均值 >0

个则报警

* + - * ⽣产环境数据库负载均衡监控规则

表 5-4 ⽣产环境数据库负载均衡监控规则

### 监控项 监控描述

后端健康 ECS 实例数 1 分钟后端健康 ECS 实例数平均值 <1

个则报警

* + - * 测试环境负载均衡监控规则

表 5-5 测试环境负载均衡监控规则

### 监控项 监控描述

流出带宽 5 分钟流出带宽平均值 >3M/s 则报警

流⼊带宽 5 分钟流⼊带宽平均值 >3M/s 则报警 后端异常 ECS 实例数 1 分钟后端异常 ECS 实例数平均值 >0

个则报警

* + 1. CDN 监控 CDN 监控时通过监控每⼀个域名的流量状态来判断应⽤的访问是否正常，当遇到DDos 攻击时，CDN 的流量会出现明显异常，通过监控这些异常，及时向运维⽤户报警，在很⼤程度上可以减少 CDN 流量的损失和降低攻击的风险。

⽬前 CDN 监控的规则如表 [5-6](#_bookmark72)所⽰:

表 5-6 CDN 监控规则

|  |  |  |
| --- | --- | --- |
| **报警维度** | **监控项** | **监控描述** |
| fileserver | ⽹络带宽峰值 | 5 分钟宽带峰值平均值 |
|  |  | >4000000bit/s 则报警 |
| fileserver | 公⽹下⾏流量 | 5 分钟公⽹⽹络出流量求 |
|  |  | 和值 >100M/s 则报警 |
| img | ⽹络带宽峰值 | 5 分钟宽带峰值平均值 |
|  |  | >4000000bit/s 则报警 |
| web | ⽹络带宽峰值 | 5 分钟宽带峰值平均值 |
|  |  | >5000000bit/s 则报警 |

## 自定义服务监控

虽然阿⾥云的云监控功能很完善，但是对于错误恢复和特殊需求的监控做的还相对不⾜，因此需要在本地的服务器中⾃⼰搭建监控的环境，以提升系统的稳定性。

* + 1. ⼼跳监听

为了保证监控系统的⾼可⽤性，需要在 APP1 和APP2 两个应⽤服务器同时搭建监控系统，为了保证两套监控系统在同⼀时间只有⼀个监控系统在运⾏需要配

置⼼跳监听，从监控节点通过⼼跳监听来监听主监控节点的运⾏状态，当主监控节点出现故障时，从监控节点运⾏监控进程，保证监控的正常[[42](#_bookmark127)]。

Heartbeat 是⼀款开源提供⾼可⽤（Highly-Available）服务的软件，通过Heartbeat可以将资源（IP 及程序服务等资源）从⼀台已经故障的计算机快速转移到另⼀台可以正常运转的机器上继续提供服务，⼀般称之为⾼可⽤服务。在实际⽣产应⽤场景中，heartbeat 的功能和keepalived 有很多相同之处，但在⽣产中，对实际的业务应⽤也是有区别的。如：keepalived 主要是控制ip 的漂移，配置、应⽤简单，⽽heartbeat 则不但可以控制 ip 漂移，更擅长对资源服务的控制，配置、应⽤⽐较复杂[[43](#_bookmark128)]。由于 Heartbeat 能够对资源服务进⾏控制，所以本论⽂使⽤ Heartbeat 作为

⼼跳监听的⼯具。

在服务器中配置⼼跳监听的步骤主要有以下步骤：

* + - 1. 下载和安装 Heartbeat 软件

访问<http://www.linux-ha.org/wiki/Downloads> 下载Heartbeat 软件，解压完成后进⼊软件的⽬录中，执⾏以下命令完成安装：

./bootstrap

#进入源码目录生成配置文件：

./ConfigureMe configure

#编译安装

make && make install

安装完成后出现下⾯的信息表⽰安装成功：

heartbeat configuration:

Version Executables Man pages Libraries Header files

Arch-independent

files

Documentation files

heartbeat" State information

System configuration Init (rc) scripts

Init (rc) defaults

= "3.0.6"

= "/usr/sbin"

= "/usr/share/man"

= "/usr/lib64"

= "/usr/include"

= "/usr/share"

= "/usr/share/doc/

=

=

=

=

"/var"

"/etc" "/etc/rc.d/init.d" "/etc/sysconfig"

Use system LTDL = "yes"

HA group name = "haclient"

HA group id = "987"

HA user name = "hacluster"

HA user user id = "991"

Build dopd plugin = "yes" Enable times kludge = "yes"

CC\_WARNINGS = " -Wall -Wmissing- prototypes -Wmissing-declarations -Wstrict- prototypes -Wdeclaration-after-statement - Wpointer-arith -Wwrite-strings -Wcast-qual - Wcast-align -Wbad-function-cast -Winline - Wmissing-format-attribute -Wformat=2 -Wformat- security -Wformat-nonliteral -Wno-long-long -Wno

-strict-aliasing -Werror "

Mangled CFLAGS = "-g -O2 -Wall - Wmissing-prototypes -Wmissing-declarations - Wstrict-prototypes -Wdeclaration-after-statement

-Wpointer-arith -Wwrite-strings -Wcast-qual - Wcast-align -Wbad-function-cast -Winline - Wmissing-format-attribute -Wformat=2 -Wformat- security -Wformat-nonliteral -Wno-long-long -Wno

-strict-aliasing -Werror -ggdb3 -funsigned-char "

Libraries = "-lbz2 -lz -lc -luuid - lrt -ldl -lltdl"

RPATH enabled = "no"

Distro-style RPMs = "no"

Note: If you use the ’make install’ method for installation you

also need to adjust ’/etc/passwd’ and ’/etc/group’ manually.

* + - 1. 配置 Heartbeat 软件，实现两个服务器的⼼跳监听

heartbeat 软件在使⽤时主要需要配置 3 个⽂件，分别是认证⽂件 authkeys, 配置⽂件ha.cf 和资源⽂件haresources，authkeys 配置⽂件主要配置节点之间的认证⽅式，ha.cf 时主要的配置⽂件，主要配置节点信息、⽹络信息、监听频率等信息，haresources ⽂件主要配置需要运⾏的程序或脚本。这⾥以 app1/app2 两节点为例，其 IP 分别是 10.46.170.191/10.172.89.141，这⾥⽰例的配置⽂件为 app1 的，app2 的参考 app1 配置即可。

ha.cf 主配置⽂件配置如下：

# 日志路径

logfile /var/log/ha-log

# 日志级别， 默认是用系统日志

#logfacility local0

# 发送心跳报文的间隔， 默认单位为秒， 也可以使用500ms来代理500毫秒， 等同于0.5。

keepalive 2

# 认为对方宕掉的间隔时间， 超过这个时间， 则认为对方已经宕掉。

deadtime 30

# 认为对方可能宕掉的间隔时间。

warntime 10

# 等待对方启动的最大时 间， （ 超 过这个时间会自动认为对方已经启动成 功？ ） 。

initdead 120

# 表示heartbeat广播/单播通讯使用的udp端口。

udpport 694

# 心跳所使用的网络接口。

bcast eth0

# 单播通讯， 对方网络接口及IP地址。

ucast eth0 10.172.89.141

# 表示当主节点（ 即提供资源/服务的节点） 正常之后是否将资源/服务切换回来。

auto\_failback on

# 看门狗定时器， 如果节点一分钟内没有心跳， 则重启节点。

#watchdog /dev/watchdog

# 就是heartbeat集群中的节点信息（ 节点的主机名: uname -

n） 。

node app2

node app1

在配置过程中将app1 配置为主节点，当配置auto\_failback 为off 时，app1 节点异常时 app2 会接管继续执⾏监控程序，app1 节点恢复后也不会移交监控程序，当配置auto\_failback 为on 时，app1 节点恢复后app2 回将资源移交回app1.

authkeys 主配置⽂件配置如下:

auth 2

#1 crc

2 sha1 mimaminishop

#3 md5 somewords

此⽂件为不同集群中heartbeat 节点进⾏连接的认证⽂件，不同集群中该⽂件采⽤的算法和密钥必须相同，认证⽅式有 3 种算法:crc ⽅式、md5 加密⽅式和 sha1 哈希⽅式。三种⽅式中crc ⽅式只能够校验节点间通信的校验数据包是否损坏，不能进⾏安全性认证；sha1/md5 两种⽅式需要进⾏安全性认证， 它们通过⼀个密钥来进⾏认证。sha1 和 md5 两种⽅式相⽐，sh1 ⽅式的资源消耗远⼩于 md5 ⽅式，因此在⼤多数应⽤中建议使⽤ sha1 ⽅式[[44](#_bookmark129)]。

可以看出，⽰例中使⽤的是 sha1 算法，如果要换⽤其他算法只需要修改 auth 指令后⾯的数字，然后取消相应⾏的注释即可。另外，该⽂件的属性必须为600，否则 heartbeat 启动将失败。

haresources 主配置⽂件配置如下:

app1 system\_monitor

这表⽰Heartbeat 启动时会执⾏资源路径中的 system\_monitor 脚本，实现服务的监控。

* + - 1. 配置监控脚本，实现服务的监控。

#!/bin/bash

# 基本变量配置MONITOR\_LOG\_PATH="/tmp/system\_monitor.log" MONITOR\_PID=’/tmp/system\_monitor.lock’ mysql\_heal\_script="/opt/sh/mysql/mysql\_health\_stat.sh"

echo "--[OK]Mysql sync is

$MONITOR\_LOG\_PATH

else

echo "--[ERROR]Mysql sync

$MONITOR\_LOG\_PATH

fi

is OK" >>

mysql\_sync\_script="/opt/sh/mysql/mysql\_sync\_stat.sh"

tomcat\_heal\_script="/opt/sh/ssh/tomcat\_health\_stat.sh" heal\_stat=1

sync\_stat=1 CHECK\_TIME=’30s’

export sms\_send\_count=0 export sms\_send\_count\_1=0

export sms\_send\_count\_tomcat=0 function check\_db\_health (){

source $mysql\_heal\_script if [ $? = 0 ] ;then

echo "--[OK]Mysql health is OK" >>

$MONITOR\_LOG\_PATH

else

echo "--[ERROR]Mysql health is ERROR" >>

$MONITOR\_LOG\_PATH

fi

}

function check\_db\_sync (){ source $mysql\_sync\_script if [ $? = 0 ] ;then

OK" >>

is ERROR" >>

}

function check\_tomcat\_health (){ source $tomcat\_heal\_script if [ $? = 0 ] ;then

echo "--[OK]Tomcat health

$MONITOR\_LOG\_PATH

else

echo "--[ERROR]Tomcat health is ERROR" >>

$MONITOR\_LOG\_PATH

fi

}

start(){

echo "Starting system\_monitor ..." if [ -f $MONITOR\_PID ];then

echo "system\_monitor is already started!" exit 0

fi

while [ true ]; do mypid=$$

echo $mypid > $MONITOR\_PID

echo ’####’(date ’+%Y-%m-%d %H:%M:%S’)####’ >>

$MONITOR\_LOG\_PATH

check\_db\_health check\_db\_sync check\_tomcat\_health sleep ${CHECK\_TIME}

done

}

stop(){

if [ -f $MONITOR\_PID ];then

echo "Stopping system\_monitor ..." kill (cat $MONITOR\_PID)

rm -f $MONITOR\_PID else

echo "system\_monitor is already stopped!"

fi

}

status(){

if [ -f $MONITOR\_PID ];then

echo

else

echo

fi

"system\_monitor

is Running!"

"system\_monitor

is

stopped!"

}

case "$1" in

start)

start

;;

stop)

stop

;;

status)

status

;;

restart)

stop start

;;

\*)

echo $"Usage: $0 {start|stop|status|restart}" exit 2

esac

通过 system\_monitor 脚本，每 30 秒对脚本中的监控服务进⾏⼀次监听，如果有新的监控脚本，可以在该脚本中脚本的路径并完成相关配置。

* + 1. Tomcat 监控

对于 Tomcat 的监控监控，主要通过 Python 中 ssh 库的 SSHClient() 对象的connect ⽅法连接到远程服务器中，然后通过 exec\_command 函数在服务器端执⾏systemctl is-active tomcat.service 命令来检测 Tomcat 服务的健康状态，并且将规

定好的状态码返回到监控脚本中，如果检测到异常，则通过 systemctl restart tom- cat.service 命令来重新启动 Tomcat 服务。具体的监控和故障处理脚本可以通过附录 [B](#_bookmark134)和附录 [C](#_bookmark135)来查看

* + 1. 数据库监控

对于数据库的监控主要包括数据库监控状态监测和数据库主从复制状态监控。

* + - 1. 数据库健康状态监控

对于数据库的健康状态，通过 mysql 命令访问服务器数据库执⾏ show status 命令来判断数据库是否运⾏正常，如果有数据返回则说明数据库运⾏正常， 负责数据库健康状态出现问题，出现问题后通过调⽤短信发送命令向运维⼈员发送通知，健康监控脚本可以参考附录 [D](#_bookmark136)。

* + - 1. 数据库同步状态监控

对于数据的同步状态， 通过执⾏ show slave status 命令来获取从库的状态， 然后⾸先查看 SQL 和 IO 线程是否为 YES 状态， 其次通过延迟来判断数据库复制的延迟是否过⼤，最后通过判断 Mas- ter\_Log\_File 和 Relay\_Master\_Log\_File 的值是否相等以及判断同步的偏移量Read\_master\_log\_Pos 和 Exec\_Master\_log\_pos 的值是否相等来判断同步的健康状态。

当数据库同步出现故障时通过发送短信通知运维⼈员进⾏处理。数据库同步状态监控的脚本可以参考 [E](#_bookmark137)

## 短信通知

在以前，当服务出现问题时⽆法及时的通知到运维⼈员，直到运维⼈员巡检活着应⽤的访问出现问题时运维⼈员才会发现故障，⽽现在有了监控的脚本，那么尽快的通知运维⼈员就是需要解决的⼀个主要问题。通知时效性最⾼的就是短信通知，因此在服务器中开发⼀个能够发送短信程序就显得⾮常重要[[45](#_bookmark130)]。

为了实现短信的发送，⾸先需要选择和注册⼀个短信平台，由于云通讯平台短信发送的时间低于五秒，且⽀持⾃定义短信模版的功能，以及⽀持 API 的特点， 本论⽂选择云通讯作为系统的短信发送应⽤。

其次，需要在服务中安装 SDK，同时开发短信发送脚本调⽤ SDK 实现短信的发送, 短信发送脚本如下所⽰：

# -\*- coding: UTF-8 -\*- from CCPRestSDK import REST

import ConfigParser import re

import sys

# 基本参数配置

# 主账号

accountSid= ’主账户Id’;

# 主账号Token

accountToken= ’主账号验证token’;

# 应用ID

appId=’应用标识， 每个创建的应用都对应唯一的id标识’;

# 请求地址

serverIP=’app.cloopen.com’;

# 请求端口

serverPort=’8883’;

# REST版本号softVersion=’2013-12-26’;

# 发送函数

def sendTemplateSMS(to,datas,tempId):

rest = REST(serverIP,serverPort,softVersion) rest.setAccount(accountSid,accountToken) rest.setAppId(appId)

result = rest.sendTemplateSMS(to,datas,tempId) for k,v in result.iteritems():

if k==’templateSMS’ :

for k,s in v.iteritems(): print ’%s:%s’ % (k, s)

else:

print ’%s:%s’ % (k, v)

# 主函数

def main():

if len(sys.argv) == 4:

# 编码调整

reload(sys)

sys.setdefaultencoding(’utf-8’)

# 变量处理to=sys.argv[1] tempId=sys.argv[2]

datas=re.split(’-’,sys.argv[3]) sendTemplateSMS(to,datas,tempId) sys.exit(0)

else:

print "参数错误" sys.exit(1)

if name ==’ main ’:

main()

在脚本中，⾸先需要根据⾃⼰注册的账户获取账户 ID 和验证 Token，然后获取

⾃⼰创建的应⽤的 ID，完成基础的配置和认证。在执⾏脚本时，通过 SendTem- plateSMS.py phone tempID content 命令来发送短信，其中phone 为接受短信的⼀个或者⼀组⼿机号，tempID 为⾃⼰设计的短信模版对于的 ID，contenet 为短信的发送内容。

## 日志备份

在 Tomcat 的运⾏过程中，随着⽤户的访问，每天都会产⽣⼤量的访问⽇志和请求⽇志，随时时间的增加，⽇志的数量和空间占⽤会越来越⼤，为了解决⽇志的空间占⽤问题同时保证⽇志的存储以便后期进⾏⽇志分析，需要设计开发⽇志备份脚本。主要的需求为：

* + - 每个⽉的 1 号会压缩上⼀个⽉的⽇志⽂件到压缩⽂件中
    - 每个⽉清理超过三个⽉的⽇志
    - 每个⽉压缩的⽇志上传到阿⾥云归档存储中进⾏备份针对以上需求，开发的归档存储上传脚本为：

#!/usr/bin/python2.7

# -\*- coding: utf-8 -\*-

from oas.oas\_api import OASAPI from oas.ease.vault import Vault import sys

def main():

if len(sys.argv) == 4:

# 变量处理vault\_name=sys.argv[1] file\_path=sys.argv[2] file\_desc=sys.argv[3]

# 创建OASAPI对象

api = OASAPI(’cn-beijing.oas-internal.aliyuncs.com ’,’AccessKeyId’, ’AccessKeySecret’)

# 获取vault

vault = Vault.get\_vault\_by\_name(api,vault\_name)

# 上传后获取archiveid

archive\_id = vault.upload\_archive(file\_path, file\_desc)

print archive\_id sys.exit(0)

else:

print "参数错误" sys.exit(1)

if name ==’ main ’:

main()

通过 oas 库中 oas\_api ⽅法链接获取到 OAS 对象，通过对象的upload\_archive ⽅法上传备份的⽇志⽂件。

开发的⽇志备份脚本可以参考附录 [F](#_bookmark138), 脚本中⾸先通过匹配不同类型的⽇志名称来将⽇志⽂件分类压缩，然后通过执⾏ OAS 上传脚本上传压缩后的⽇志⽂件， 上传完成后删除未压缩的⽇志⽂件。

⽇志备份脚本开发完成后，为了保证脚本每个⽉运⾏⼀次，需要修改 crontab

的配置⽂件，增加：

0 3 1

\* \*

/opt/sh/Prod\_tomcat\_log\_bak.sh

设置脚本执⾏时间为每个⽉ 1 号的凌晨 3 点。

## 本章总结

本章主要对服务器层级的服务器监控措施和应急措施的优化策略研究，主要包括基于阿⾥云平台的⾃有监控和通知⽅案设计、个⼈开发的服务监控⽅案设计、短信通知脚本开发以及⽇志定期备份脚本开发等内容，通过以上的监控和优化措施的开发，在最⼤程度上保证服务器的正常运⾏、服务的正常运⾏、服务故障的⾃动恢复以及服务故障的通知，降低运维⼈员的时间成本，努⼒提升系统的稳定性。

# 总结与展望

## 总结

本论⽂通过针对基于 Spring MVC 框架开发的 WEB 应⽤系统进⾏应⽤、数据库、服务器三个层级的优化研究，如图 [6-1](#_bookmark83)所⽰。

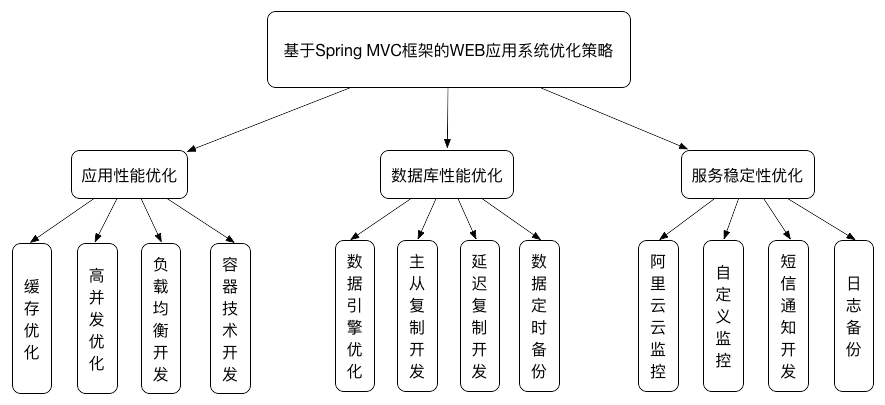


图 6-1 系统优化研究总结

对于应⽤层⾯，论⽂通过开发 Couchbase 技术，将应⽤的基础数据缓存到Couchbase 缓存中，提⾼了数据的加载效率并且降低了数据的压⼒；通过对 Tomcat 进⾏基于⾼并发的 apr 模块优化，⼤⼤提升了 Tomcat 对于⽤户请求的处理，提升了⽤户的访问速度；通过开发负载均衡技术，将⽤户的请求转发到两台应⽤服务器中，降低了服务器的压⼒，同时增强了应⽤的⾼可⽤性；通过开发 Docker 容器技术，将数据库应⽤、Couchbase 应⽤以容器的⽅式提供服务，在新增加服务器节点的过程中提⾼了快速部署应⽤的效率，同时实现了多个服务器同⼀服务的集中管理，⼤⼤提⾼了运维⼈员的⼯作效率。

对于数据库层⾯，⾸先对数据库使⽤的数据引擎 InnoDB 引擎进⾏优化，在内存读取、⽇志、IO 和其它⽅⾯提升了数据库的运⾏效率以及数据的稳定性；通过开发双主数据库的主从复制，提升了数据库的⾼可⽤性以及数据的完整性；通过开发延迟复制，可以最⼤程度的将数据进⾏备份，加上机遇⼆进制⽇志的恢复⽅式，能够在短时间内完成数据的恢复，这对于提⾼数据的完整⾏和应⽤的⽤户体验尤为重要；通过开发数据定时备份脚本，将每⽇的数据进⾏备份，以防数据丢

失带来的数据损伤和经济损失。

对于服务器层⾯，论⽂通过阿⾥云的云监控服务，对系统的云服务器、负载均衡、⽹络站点以及CDN 进⾏监控，当监控项出现异常时通知运维⼈员作出相应的处理，这提⾼了发现问题和解决服务器故障的效率，通过降低了过多流量丢失导致的经济损失；通过开发⾃定义监控实现对 Tomcat 服务的健康状态进⾏监测和重启，对 MySQL 的健康状态进⾏监测和根据监控情况调整数据库负载均衡损害数据库服务器的权重，并且短信通知运维⼈员，对于 MySQL 的主从同步状态进⾏同步状态监测并且通知运维⼈员；通过开发短信通知模块，实现重要信息的短信通知；通过⽇志备份模块的开发对 Tomcat 的运⾏⽇志进⾏定时备份并且上传到阿⾥云归档存储，提升运维⼈员在发⽣故障时的故障原因定位。

通过三个层⾯的优化策略研究，⽬前应⽤服务较优化前相⽐，单个服务器的负载达到了⽐较好的程度，应⽤的访问速度提升了接近 20%, 新版的部署时间节省了 50%。

具体的数据对⽐在后期的论⽂完善过程中进⾏补充。

## 展望

虽然通过本论⽂的优化，WEB 应⽤的性能得到了⼤⼤的提升，但是⽬前的

WEB 应⽤系统依然有很⼤的改进空间。

* + 1. 搭建基于 Nginx 的前端服务器，由于 Nginx 在⾼并发的处理和静态⽂件的处理速度⽅⾯以及安全性⽅⾯的性能均优于 Tomcat 对静态页⾯处理，因此搭建 Nginx 服务器处理⽤户的请求，将前端的页⾯请求直接提供给⽤户，后端的请求通过转发到 Tomcat 获取数据的处理结果。这样对于应⽤的访问速度还能进⼀步的提⾼，对于 DDos 攻击也能起到⼀定程度的防御[[46](#_bookmark131)]。
    2. 对于数据库的性能⽽⾔，除了⾃⾝的引擎提⾼之外，还可以通过读写分离来提⾼数据库的响应和降低数据库的负载[[47](#_bookmark132)]。通过使⽤ MaxScale 插件，实现数据库请求的读写分离。具体的流程如图 [6-2](#_bookmark85)所⽰。

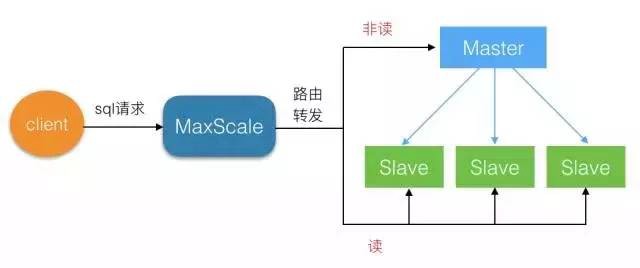


图 6-2 数据库读写分离流程

MaxScale 会根据⽤户的请求将读取的请求和⾮读取的请求转发到不同的数据库中，其中读取的请求转发到 slave 库中，其它请求转发到 master 库中，实现数据的读写分离。

* + 1. 除了通过读写分离的⽅式提升数据库的效率以外，在数据库稳定性⽅⾯可以通过 LiquiBase 软件来实现数据库重构和迁移的操作，它可以在很⼤程度上保证当数据库之间出现数据不⼀致的情况时，能够⾃动的实现数据的同步和迁移，避免了⼈⼯⼲预导致的其它问题。
    2. 本论⽂虽然⽤了阿⾥云的云监控实现了服务状态的监控，但是服务的⾃动failover ⼯作却没有实现，⽬前还是通过⼈⼯⼲预来进⾏故障恢复。在以后的项⽬开发中，可以充分开发阿⾥云 API 的故障检测和⾃动 failover 系统，通过云监控的 API 去监控服务的状态，当出现故障时，⾃动调⽤ failover 程序通过故障服务的API 接⼜进⾏故障的转移或者恢复，这对于提⾼故障的解决和 WEB 应⽤的⾼可⽤有更⼤的价值。

### 参考文献

1. 李梦, 江⼭, 黄幸, et al. 基于前端的 web 优化设计. 电⼦设计⼯程, 2016, 24(9):78–80.
2. 林薇. 基于 spring mvc 架构的数据同步系统的设计与实现 [D]. 吉林⼤学, 2015.
3. 邵志强. 跨媒体检索平台 [D]. ⼭东师范⼤学, 2015.
4. 贺海梁, 袁⽟宇. 基于 rest 的⾯向资源 web 应⽤架构参考模型. 软件, 2012, 33(11):59–63.
5. 宇, 王映辉, 张翔南. 基于 spring 的mvc 框架设计与实现. 计算机⼯程, 2010, 36(4):59–62.
6. 王霜, 修保新, 肖卫东. Web 服务器集群的负载均衡算法研究. 计算机⼯程与应⽤, 2004, 40(25):78–80.
7. 李守振, 张南平, 常国锋. Web 应⽤分层与开发框架设计研究. 计算机⼯程, 2006, 32(22):274–276.
8. Jemerov D. Implementing refactorings in intellij idea. Proceedings of the 2nd Workshop on Refactoring Tools. ACM, 2008. 13.
9. wikipedia. Intellij idea. <https://zh.wikipedia.org/wiki/IntelliJ_IDEA>. Accessed November 16, 2017.
10. Smart J F, et al. An introduction to maven 2. JavaWorld Magazine. Available at: [http://www.](http://www/) javaworld. com/javaworld/jw-12-2005/jw-1205-maven. html, 2005..
11. Brittain J, Darwin I F. Tomcat: The Definitive Guide: The Definitive Guide. " O’Reilly Media, Inc.", 2007.
12. Greenspan J, Bulger B. MySQL/PHP database applications. John Wiley & Sons, Inc., 2001.
13. 王川. 农业机械装备信息管理系统的设计和研究, 2009.
14. Brown M C. Getting Started with Couchbase Server. " O’Reilly Media, Inc.", 2012.
15. Kovács K. Cassandra vs mongodb vs couchdb vs redis vs riak vs hbase vs couchbase vs neo4j vs hypertable vs elasticsearch vs accumulo vs voltdb vs scalaris comparison, 2013.
16. ⾼珺. 以持续集成⽅式进⾏系统⾃动化部署. 华东师范⼤学学报: ⾃然科学版, 2015, (B03):373–377.
17. 王宁. 基于 jenkins 的持续集成系统的设计与实现 [D]. 北京邮电⼤学, 2014.
18. 赵亚楠. 基于 jenkins 的企业持续集成系统的设计与实现 [D]. 西安电⼦科技⼤学, 2013.
19. 赵杰昌, 张良宇. 基于 jenkins 构建持续集成系统. 电脑编程技巧与维护, 2014, (9):9–10.
20. 董晓光, 喻涛. 使⽤ maven 构建 java 项⽬. 电⼦技术与软件⼯程, 2014, (10):105–105.
21. Mileva Y M, Dallmeier V, Burger M, et al. Mining trends of library usage. Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops. ACM, 2009. 57–62.
22. Merkel D. Docker: lightweight linux containers for consistent development and deployment. Linux Journal, 2014, 2014(239):2.
23. Brown M C. Developing with Couchbase Server. O’Reilly, 2013.
24. Vukotic A, Goodwill J. Securing tomcat with ssl. Apache Tomcat 7. Springer, 2011: 141–154.
25. 蒋⽂旭, ⾟阳. ⼤型⾼并发web 应⽤系统架构分析与设计 [D]. 北京: 北京邮电⼤学, 2012.
26. HuaYing. Tomcat 性能调优 [EB/OL]. [2012-10-06]. [http://blog.sina.com.cn/s/blog\_](http://blog.sina.com.cn/s/blog_417b97470100glmi.html) [417b97470100glmi.html](http://blog.sina.com.cn/s/blog_417b97470100glmi.html).
27. Vukotic A, Goodwill J. Embedding tomcat. Apache Tomcat 7. Springer, 2011: 175–184.
28. Fink J. Docker: a software as a service, operating system-level virtualization framework. Code4Lib Journal, 2014, 25.
29. 刘熙, 胡志勇. 基于 docker 容器的 web 集群设计与实现. 电⼦设计⼯程, 2016, 24(8):117– 119.
30. Chamberlain R, Schommer J. Using docker to support reproducible research. DOI: [http://dx.](http://dx/) doi. org/10.6084/m9. figshare, 2014, 1101910.
31. ⾼飞, 李勇. Docker 容器资源管控配置实战. 程序员, 2014, (9):96–101.
32. 互联⽹⽂档资源. 负载均衡 (计算机)[EB/OL]. [2015-07-22]. [http://www.360doc.com/](http://www.360doc.com/content/14/0912/14/15077656_408909322.shtml) [content/14/0912/14/15077656\_408909322.shtml](http://www.360doc.com/content/14/0912/14/15077656_408909322.shtml).
33. Wei C. System and method for performance acceleration, data protection, disaster recovery and on-demand scaling of computer applications, March 4, 2010. US Patent App. 12/717,297.

[34] 胡雯, 李燕. Mysql 数据库存储引擎探析. 软件导刊, 2012, 11(12):129–131.

1. Schwartz B, Zaitsev P, Tkachenko V. High performance MySQL: optimization, backups, and replication. " O’Reilly Media, Inc.", 2012.
2. 顾治华, 忽朝俭. Mysql 存储引擎与数据库性能. 计算机时代, 2006, (10):8–10.
3. Frühwirt P, Huber M, Mulazzani M, et al. Innodb database forensics. Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on. IEEE, 2010. 1028–1036.
4. 朱振. 基于 mysql 复制改进的多主复制数据库扩展实现 [D]. 万⽅数据资源系统, 2013.
5. 秦⾦. 分布式 mysql 副本控制的研究与实现 [D]. 北京邮电⼤学, 2013.

[40] 刘雄辉. 服务器监控管理. ⽹管员世界, 2007, (23):75–77.

1. 梁宇, 杨海波, 李鸿彬, et al. 基于 openstack 资源监控系统. 计算机系统应⽤, 2014, 23(4):44–47.
2. 巩天宁, 周书明. 基于 drbd 的 linux ⾼可⽤集群. 电脑与信息技术, 2012, 20(1):63–65.
3. 郭绪晶. 服务器集群系统⾼可⽤模块设计与实现 [D]. 万⽅数据资源系统, 2012.
4. 百度⽂库. heartbeat 配置说明 [EB/OL]. [2012-09-23]. [http://wenku.baidu.com/view/](http://wenku.baidu.com/view/101c9ac3d5bbfd0a795673df.html) [101c9ac3d5bbfd0a795673df.html](http://wenku.baidu.com/view/101c9ac3d5bbfd0a795673df.html).
5. 陈泰伟, 周振柳, 刘宝旭. 基于短信平台的服务器监控系统关键技术探讨. 核電⼦學與探測技術, 2007, 27(6):1050–1053.
6. Reese W. Nginx: the high-performance web server and reverse proxy. Linux Journal, 2008, 2008(173):2.
7. 刘进京. Mysql 主从复制读写分离. ⽹络安全和信息化, 2016, (8):64–69.

**附录** A Jenkins **自动部署脚本**

#/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/ local/sbin

export PATH

# 公共变量apppath=’/usr/local/tomcat7/webapps’ backpath=’/opt/minishop\_deploy/bakup’ logpath=’/opt/minishop\_deploy/logs/deploy.log’

#公共函数

function Usage()

{

echo "Usage: deploy.sh [server/client]

deploypath]" exit 1

[project

name] [

}

if [ $# != 3 ];then Usage

fi

if [ -d $3 ];then deploypath=$3

else

Usage

fi

backtoday="${backpath}/(date +%Y%m%d)" if [ ! -d $backtoday ];then

mkdir -p $backtoday

fi

echo "####$2- (date +%Y%m%d%H%M%S) -deploy####" >> $logpath echo "开始部署"

if [ $1 == ’server’ ]; then

echo "-备份原始数据-" >> $logpath

#删除原备份文件

if [ -d "$backtoday/$2" ] || [ -f "$backtoday/$2.war"

];then

rm -rf "$backtoday/$2"

rm -rf "$backtoday/$2.war" echo "--原备份文件删除成功--"

echo "--原备份文件删除成功--" >> $logpath

fi

#备份并删除原文件

if [ -d "$apppath/$2" ] || [ -f "$apppath/$2.war" ]; then

cp -rf "$apppath/$2" $backtoday

cp -rf "$apppath/$2.war" $backtoday echo "--备份成功--"

echo "--备份成功--" >> $logpath rm -rf "$apppath/$2"

rm -f "$apppath/$2.war" echo "--原文件删除成功--"

echo "--原文件删除成功--" >> $logpath

fi

#部署新的文件

cp "$deploypath/MiniShop.war" "$apppath/$2.war" echo "--部署成功--"

echo "--部署成功--" >> $logpath elif [ $1 == ’client’ ];then

echo "-备份原始数据-" >> $logpath

#删除原备份文件

if [ -d "$backtoday/$2" ];then rm -rf "$backtoday/$2"

echo "--原备份文件删除成功--"

echo "--原备份文件删除成功--" >> $logpath

fi

#备份并删除原文件

if [ -d $apppath/$2 ];then

cp -rf "$apppath/$2" $backtoday echo "--备份成功--"

echo "--备份成功--" >> $logpath rm -rf "$apppath/$2"

echo "--原文件删除成功--"

echo "--原文件删除成功--" >> $logpath

fi

#部署新的文件

cd "$deploypath" && tar -jxf "client.bz2" cp -rf "$deploypath/client" "$apppath/$2" cd "$deploypath" && rm -rf "client"

echo "--部署成功--"

echo "--部署成功--" >> $logpath

else

Usage

fi

**附录** B Tomcat **健康监控脚本**

#!/bin/bash

# func:执行SHELl命令监控tomcat服务状态并且执行重启操作

# author:武斌

# date:2017-01-23

# 修订历史：

# 创 建 --2017-01-23

# 调整检测时间--2017-02-01

# 基本变量配置APP1\_IP=’10.46.170.191’ APP2\_IP=’10.172.89.141’

Error\_Count=0 LOG\_PATH="/tmp/tomcat\_heal\_stat.log"

#短信通知的手机号码

SMS\_to=’1766793\*\*\*\*’

#短信通知的模版ID SMS\_tempId=’86014’ function send\_sms (){

if [[ $sms\_send\_count\_tomcat -lt 3 ]];then

if [[ $sms\_send\_count\_tomcat -lt 2 ]];then sms\_send\_count\_tomcat=$[ sms\_send\_count\_tomcat

+ 1 ]

else

sms\_result=(python /opt/sh/sms/SendTemplateSMS. py ${SMS\_to} ${SMS\_tempId} $1)

if [ "$?" == 0 ];then statusCode=${sms\_result:0-6}

if [ $statusCode == ’000000’ ];then

echo "----短信发送成功" >> $LOG\_PATH sms\_send\_count\_tomcat=$[

sms\_send\_count\_tomcat + 1 ]

else

statusMsg=${sms\_result##\*Msg:} statusMsg=${statusMsg%%s\*}

echo "---短信发送失败,错误代码： ${

statusCode}， 错误信息： ${statusMsg}"

>> $LOG\_PATH

fi else

echo "---短信发送失败" >> $LOG\_PATH

fi

fi

fi

}

function check\_health () {

check\_result=(python /opt/sh/ssh/tomcat\_fialover.py monitor $1)

check\_code="$?"

if

[[ $check\_code -eq 100 ]];then

echo "--[OK]$2 tomcat is RUNNING" >> TomcatServiceCode=$(curl -I -s -m 10

-w %{http\_code} --connect-timeout

:8089/MiniShop/index.jsp)

$LOG\_PATH

-o /dev/null 10 http://$1

if [ $TomcatServiceCode -eq 200 ];then

echo "--[OK]$2 WEB is OK" >> $LOG\_PATH

sms\_send\_count\_tomcat=0 else

echo "--[ERROR]$2 tomcat页面出错,请注意......状

态码为$TomcatServiceCode" >> $LOG\_PATH

message="$2:WEB页面访问出错， 请查看。状态码为

$TomcatServiceCode" send\_sms $message

fi

elif [[ $check\_code -eq 102 ]]; then

echo "--[ERROR]$2-Tomcat服务不存在"

message="$2：Tomcat服务不存在" send\_sms $message

elif [[ $check\_code -eq 104 ]]; then echo "--[ERROR]$2服务器连接失败" >>

message="$2服务器连接失败" send\_sms $message

elif [[ $check\_code -eq 105 ]]; then echo "--[ERROR]$2Tomcat正在重启" >>

else

>> $LOG\_PATH

$LOG\_PATH

$LOG\_PATH

echo "--[ERROR]$2 tomcat is STOPPING" >> $LOG\_PATH

echo "--[INFO]$2 开 始 重 启 Tomcat" >> $LOG\_PATH restart\_result=(python /opt/sh/ssh/tomcat\_fialover.

py restart $1) restart\_code="$?"

if [[ $restart\_code -eq 200 ]]; then

echo "--[OK]$2 Tomcat重启成功" >> $LOG\_PATH else

echo "--[ERROR]$2 Tomcat重启失败" >> $LOG\_PATH

message="$2:Tomcat重启失败" send\_sms $message

fi

elif [[ $check\_code -eq 103 ]]; then

echo "--[ERROR]$2未知错误： ${check\_result}" >>

$LOG\_PATH

message="$2应用检测未知错误： ${check\_result}" send\_sms $message

fi

}

echo ’####’(date ’+%Y-%m-%d %H:%M:%S’)####’ >> $LOG\_PATH

check\_health ${APP1\_IP} "APP1" check\_health ${APP2\_IP} "APP2"

**附录** C Tomcat **故障恢复脚本**

#!/usr/bin/python2.7

# -\*- coding: utf-8 -\*-

# func:执行SHELl命令监控tomcat服务状态并且执行重启操作

# author:武斌

# date:2017-01-23

# 修订历史：

# 创 建 --2017-01-23

**import** ssh

**import** sys

**def** main():

**try**:

**if len**(sys.argv) == 3:

# 变量处理action=sys.argv[1] host=sys.argv[2]

# 新建一个ssh客户端对象

client=ssh.SSHClient()

# 设置成默认自动接受密钥

client.set\_missing\_host\_key\_policy(ssh.

AutoAddPolicy())

# 连接远程主机port=22 username= *’ r o o t ’*

key\_filename = *’ / r o o t / . s sh / i d \_ r s a ’*

client.connect(host, port=port, username= username, key\_filename = key\_filename)

**if** action == *’ monitor ’* :

stdin, stdout, stderr = client.exec\_command (*" s y s t e m c t l i s* − *a c t i v e t omcat . s e r v i c e "* )

**if** stderr.read():

**print** stderr.read() sys.exit(105)

**else**:

status=stdout.read().strip()

**if** status == *’ a c t i v e ’* : **print** *"* 运⾏正常 *"* sys.exit(100)

**elif** status == *’ f a i l e d ’* : **print** *"* 服 务 停 ⽌ *"* sys.exit(102)

**elif** status == *’ unknown ’* : **print** *"* 未 知 服 务 *"* sys.exit(103)

**elif** status == *’ d e a c t i v a t i n g ’* : **print** *"* 服 务 重 启 *"* sys.exit(105)

**else**:

**print** *"* 未知状态 *"*

sys.exit(106)

**elif** action == *’ r e s t a r t ’* :

stdin, stdout, stderr = client.exec\_command (*" s y s t e m c t l r e s t a r t t omcat . s e r v i c e "* )

**if** stderr.read():

**print** stderr.read() sys.exit(201)

**else**:

**print** stdout.read() sys.exit(200)

**else**:

**print** *"* 不是有效参数 *"*

sys.exit(1)

**else**:

**print** *"* 参数错误 *"*

sys.exit(1)

**except** Exception:

**print** *"* 服务器连接出现问题， 请联系管理员 *"*

sys.exit(104)

**if** name == *’ \_\_ main\_\_ ’* : main()

**附录** D MySQL **健康监控脚本**

#!/bin/bash

# 基本变量配置

MYSQL=’/usr/bin/mysql’

#MYSQL=’/usr/local/mysql/bin/mysql’ DB1\_HOST=’hostname’ DB2\_HOST=’hostname’

DB1\_USER=’root’ DB2\_USER=’root’ DB1\_PASSWORD=’password’ DB2\_PASSWORD=’password’ DB1\_PORT=’3307’ DB2\_PORT=’3305’

MySQL\_LOG\_PATH="/tmp/mysql\_heal\_stat.log" DB1\_OK=1

DB2\_OK=1 CHECK\_TIME=’30s’ CHECK\_FLAG=0

DB1\_BASE\_URL=’tcp://hostname:port’ DB2\_BASE\_URL=’tcp://hostname:port’ DOCKER\_API\_VERSION=’1.21’

MYSQL\_CONTAINER\_ID=’dockermysql\_mysql\_1’ SMS\_to=’1766793\*\*\*\*’

SMS\_tempId=’86014’

function check\_db1\_health (){

$MYSQL -h${DB1\_HOST} -u${DB1\_USER} -p${DB1\_PASSWORD} -

P${DB1\_PORT} -e "show status;" 1>/dev/null 2>&1 if [ $? = 0 ] ;then

DB1\_OK=1

else DB1\_OK=0

fi

return $DB1\_OK

}

function check\_db2\_health (){

$MYSQL -h${DB2\_HOST} -u${DB2\_USER} -p${DB2\_PASSWORD} -

P${DB2\_PORT} -e "show status;" 1>/dev/null 2>&1 if [ $? = 0 ] ;then

DB2\_OK=1

else DB2\_OK=0

fi

return $DB2\_OK

}

function send\_sms (){

if

[[ $sms\_send\_count\_1 -lt 1 ]];then

sms\_result=(python /opt/sh/sms/SendTemplateSMS.py

{SMS\_to} ${SMS\_tempId} $1) if [ "$?" == 0 ];then

statusCode=${sms\_result:0-6}

if [ $statusCode == ’000000’ ];then

$

echo "----短信发送成功" >> $MySQL\_LOG\_PATH

sms\_send\_count\_1=$[ sms\_send\_count\_1 + 1 ]

else

statusMsg=${sms\_result##\*Msg:} statusMsg=${statusMsg%%s\*}

echo "---短信发送失败,错误代码： ${

statusCode}， 错误信息： ${statusMsg}" >>

$MySQL\_LOG\_PATH

fi else

echo "---短信发送失败" >> $MySQL\_LOG\_PATH

fi

fi

if [ $DB1\_OK =

echo (date

stop!"

#短信通知

1 ] && [ $DB2\_OK = 0 ] ; then

’+%Y-%m-%d %H:%M:%S’) "

>> $MySQL\_LOG\_PATH

Mysql db2 Current

}

check\_db1\_health check\_db2\_health

# 数据库均正常

if [ $DB1\_OK = 1 ] && [ $DB2\_OK = 1 ] ; then

# 调整短信计数

sms\_send\_count\_1=0

echo (date ’+%Y-%m-%d %H:%M:%S’) " Mysql status Current OK!" >> $MySQL\_LOG\_PATH

#调整阿里云负载均衡DBSLB权重

check\_result=(python /opt/sh/slb/setBackendServers.py 0

100)

if [ "$?" == 0 ];then

echo " 权重python脚本执行成功： ${check\_result}" >>

$MySQL\_LOG\_PATH

else

echo "

fi

权重python脚本执行失败"

>> $MySQL\_LOG\_PATH

fi

# 数据库均不正常

if [ $DB1\_OK =

echo (date Current

#短信通知

0 ] && [ $DB2\_OK = 0 ] ;

’+%Y-%m-%d %H:%M:%S’) "

Stop!" >> $MySQL\_LOG\_PATH

then

Mysql status

SMS\_datas=’DB1和DB2 数据库健康状态异常， 请查看！ ’

send\_sms $SMS\_datas

fi

# DB1 正常， DB2 不正常

SMS\_datas=’DB2数据库健康状态异常， 请查看！ ’

if [ "$?" ==

echo " else

echo "

fi

0 ];then

重启DB2 成功： ${mysql\_restart\_result}"

重启DB2 失败： ${mysql\_restart\_result}"

if [ $DB1\_OK =

echo (date

Stop!"

#短信通知

0 ] && [ $DB2\_OK = 1 ] ; then

’+%Y-%m-%d %H:%M:%S’) "

>> $MySQL\_LOG\_PATH

Mysql db1 Current

send\_sms $SMS\_datas

#调整阿里云负载均衡DBSLB权重

check\_result=(python /opt/sh/slb/setBackendServers.py 100 0)

if [ "$?" == 0 ];then

echo " 权重python脚本执行成功： ${check\_result}" >>

$MySQL\_LOG\_PATH

else

echo " 权重python脚本执行失败" >> $MySQL\_LOG\_PATH

fi

# 重启DB2

mysql\_restart\_result=(python /opt/sh/docker/ mysql\_restart.py ${DB2\_BASE\_URL} ${ DOCKER\_API\_VERSION} ${MYSQL\_CONTAINER\_ID})

fi

# DB2 正常， DB1 不正常

SMS\_datas=’DB1数据库健康状态异常， 请查看！ ’

send\_sms $SMS\_datas

#调整阿里云负载均衡DBSLB权重

check\_result=(python /opt/sh/slb/setBackendServers.py 0

100)

if [ "$?" == 0 ];then

echo " 权重python脚本执行成功： ${check\_result}" >>

$MySQL\_LOG\_PATH

else

echo " 权重python脚本执行失败" >> $MySQL\_LOG\_PATH

fi

# 重启DB1

mysql\_restart\_result=(python /opt/sh/docker/ mysql\_restart.py ${DB1\_BASE\_URL} ${ DOCKER\_API\_VERSION} ${MYSQL\_CONTAINER\_ID})

if [ "$?" == 0 ];then

echo "

else

echo "

fi

重 启 DB1 成 功 ： ${mysql\_restart\_result}"

重 启 DB1 失 败 ： ${mysql\_restart\_result}"

fi

-p${

-p${

**附录** E MySQL **同步状态监控脚本**

#!/bin/sh

# 基本变量配置

MYSQL=/usr/bin/mysql

#MYSQL=’/usr/local/mysql/bin/mysql’ DB1\_HOST=’hostname’ DB2\_HOST=’hostname’

DB1\_USER=’root’ DB2\_USER=’root’ DB1\_PASSWORD=’password’ DB2\_PASSWORD=’password’ DB1\_PORT=’3307’ DB2\_PORT=’3305’ DB1\_OK=1

DB2\_OK=1 DB1\_STAT=0 DB2\_STAT=0 CHECK\_TIME=’30s’

MySQL\_LOG\_PATH="/tmp/mysql\_sync\_stat.log" Connect\_db1="$MYSQL -h${DB1\_HOST} -u${DB1\_USER}

DB1\_PASSWORD} -P${DB1\_PORT}"

Connect\_db2="$MYSQL -h${DB2\_HOST} -u${DB2\_USER} DB2\_PASSWORD} -P${DB2\_PORT}"

SMS\_to=’1766793\*\*\*\*’ SMS\_tempId=’86014’

function check\_db1\_stat(){

db1\_thread\_status=($Connect\_db1 -e ’show slave status\G

’|grep -i yes|wc -l)

db1\_status=($Connect\_db1 -e ’show slave status\G’| egrep -i "Master\_Log\_File|Relay\_Master\_Log\_File| Read\_master\_log\_Pos|Exec\_Master\_log\_pos|

Seconds\_Behind\_Master" |awk -F’:’ ’{print $2}’) echo ${db1\_status[2]}

if [[ $db1\_thread\_status -ne ’2’ ]]; then

DB1\_STAT="DB1 故障， 不是双YES"

DB1\_OK=0

elif [[ "${db1\_status[4]}" > ’300’ ]]; then

DB1\_STAT="DB1 延迟过高"

DB1\_OK=0

elif [[ ${db1\_status[0]} != ${db1\_status[2]} ]] || [[ $

{db1\_status[1]} != ${db1\_status[3]} ]]; then

DB1\_STAT="DB1 从库复制位置异常"

DB1\_OK=0

else

DB1\_OK=1

fi

}

function check\_db2\_stat (){

db2\_thread\_status=$Connect\_db1 -e ’show slave status\G ’|grep -i yes|wc -l

db2\_status=($Connect\_db1 -e ’show slave status\G’| egrep -i "Master\_Log\_File|Relay\_Master\_Log\_File| Read\_master\_log\_Pos|Exec\_Master\_log\_pos| Seconds\_Behind\_Master" |awk -F’:’ ’{print $2}’)

if [[ $db2\_thread\_status -ne ’2’ ]]; then

DB2\_STAT="DB2 故障， 不是双YES"

DB2\_OK=0

elif [[ "${db2\_status[4]}" > ’300’ ]]; then

DB2\_STAT="DB2 延迟过高"

DB2\_OK=0

elif [[ ${db2\_status[0]} != ${db2\_status[2]} ]] || [[ $

{db2\_status[1]} != ${db2\_status[3]} ]]; then

DB2\_STAT="DB2 从库复制位置异常" DB2\_OK=0

else

DB2\_OK=1

fi

}

function send\_sms (){

if [[ $sms\_send\_count -lt 1 ]];then

sms\_result=python /opt/sh/sms/SendTemplateSMS.py ${

SMS\_to} ${SMS\_tempId} $1 if [ "$?" == 0 ];then

statusCode=${sms\_result:0-6}

if [ $statusCode == ’000000’ ];then

echo "----短信发送成功" >> $MySQL\_LOG\_PATH sms\_send\_count=$[ sms\_send\_count + 1 ]

else

statusMsg=${sms\_result##\*Msg:} statusMsg=${statusMsg%%s\*}

echo "---短信发送失败,错误代码： ${

statusCode}， 错误信息： ${statusMsg}" >>

$MySQL\_LOG\_PATH

fi else

echo "---短信发送失败" >> $MySQL\_LOG\_PATH

fi

fi

}

check\_db1\_stat check\_db2\_stat

# 数据库均正常

if [ $DB1\_OK = 1 ] && [ $DB2\_OK = 1 ] ; then

echo date ’+%Y-%m-%d %H:%M:%S’ " Mysql sync status Current OK! Info:$DB2\_STAT" >> $MySQL\_LOG\_PATH

sms\_send\_count=0

fi

# DB1 正常， DB2 不正常

if [ $DB1\_OK = 1 ] && [ $DB2\_OK = 0 ] ; then

echo date ’+%Y-%m-%d %H:%M:%S’ " Mysql db2 sync ERROR! Info:$DB2\_STAT" >> $MySQL\_LOG\_PATH

#短信通知

SMS\_datas="DB2数据库同步异常， 错误原因为$DB2\_STAT, 请查看 ！ "

send\_sms $SMS\_datas

fi

# DB2 正常， DB1 不正常

if [ $DB1\_OK = 0 ] && [ $DB2\_OK = 1 ] ; then

echo date ’+%Y-%m-%d %H:%M:%S’ " Mysql db1 sync ERROR! Info:$DB1\_STAT" >> $MySQL\_LOG\_PATH

#短信通知

SMS\_datas="DB1数据库同步异常， 错误原因为$DB1\_STAT, 请查看 ！ "

send\_sms $SMS\_datas

fi

# 数据库均不正常

if [ $DB1\_OK = 0 ] && [ $DB2\_OK = 0 ] ; then

echo date ’+%Y-%m-%d %H:%M:%S’ " Mysql sync status Current ERROR! Info: $DB1\_STAT--$DB2\_STAT" >>

$MySQL\_LOG\_PATH

#短信通知

SMS\_datas="DB1和DB2 数据库同步异常， 错误原因为$DB1\_STAT,

$DB1\_STAT, 请查看！ " send\_sms $SMS\_datas

fi

**附录** F Tomcat **日志备份脚本**

#/bin/sh PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/

local/sbin export PATH

# 当前月份

tmonth=date +%m

#备份时间

backtime=date +%Y%m%d%H%M%S

# 要备份的日志路径

logpath=’/usr/share/tomcat/logs’

# 归档存储的的vault名称

vault\_name=’tomcat\_prod\_app1\_log’

# 要备份的日志名称catalina\_list=’’ localhost\_list=’’ localhost\_access\_log\_list=’’ minishop\_list=’’

# 根据当前月份判断上一月份的年-月

if [ "$tmonth" == ’01’ ]; then month=12

year=date -d ’last year’ +%Y

else

month=date -d ’last month’ +%m year=date +%Y

fi

# 获取指定路径下的所有日志文件名称

files=$(ls $logpath)

# 正则匹配

catalinaPattern="^catalina.$year-$month-[0-9]\*.log$" localhostPattern="^localhost.$year-$month-[0-9]\*.log$"

localhostAccessPattern="^localhost\_access\_log.$year-$month

-[0-9]\*.txt$"

minishopPattern="^minishop.$year-$month-[0-9]\*.log$"

# 将不同类型的日志文件给对应变量

for i in $files:

do

if [[ "$i" =~ $catalinaPattern ]];then catalina\_list="$catalina\_list $i"

elif [[ "$i" =~ $localhostPattern ]];then localhost\_list="$localhost\_list $i"

elif [[ "$i" =~ $localhostAccessPattern ]]; then

localhost\_access\_log\_list="

$localhost\_access\_log\_list $i" elif [[ "$i" =~ $minishopPattern ]];then

minishop\_list="$minishop\_list $i"

fi

done

# 日志记录头部

echo "-备份时间为${backtime}， 备份$year-$month的日志开始"

>> ${logpath}/log.log cd ${logpath}

if [ ! -z "${localhost\_list}" ];then localhostbz="localhost-${year}-${month}-log.tar.bz2" tar jcf ${localhostbz} ${localhost\_list} > /dev/null if [ "$?" == 0 ];then

localhost\_path="${logpath}/${localhostbz}" localhost\_desc="localhost-${year}-${month}-log" localhostid=python /opt/sh/oas/logupoas.py ${

vault\_name} ${localhost\_path} ${localhost\_desc} if [ "$?" == 0 ];then

echo "--localhost归档存储上传成功,Archiveid为:$

{localhostid}" >> ${logpath}/log.log

else

echo "--localhost归档存储上传失败,原因为:${ localhostid}" >> ${logpath}/log.log

fi

echo "--localhost日志备份完成， 删除原始数据" >> ${ logpath}/log.log

rm -f ${localhost\_list} else

echo "--localhost日志备份失败" >> ${logpath}/log.

log

fi

else

echo "--无localhost日志" >> ${logpath}/log.log

fi

if [ ! -z "${catalina\_list}" ];then catalinabz="catalina-${year}-${month}-log.tar.bz2" tar jcf ${catalinabz} ${catalina\_list}

if [ "$?" == 0 ];then catalina\_path="${logpath}/${catalinabz}" catalina\_desc="catalina-${year}-${month}-log" catalinaid=python /opt/sh/oas/logupoas.py ${

vault\_name} ${catalina\_path} ${catalina\_desc} if [ "$?" == 0 ];then

echo "--catalina归档存储上传成功,Archiveid为:${

catalinaid}" >> ${logpath}/log.log

else

echo "--catalina归档存储上传失败,原因为:${ catalinaid}" >> ${logpath}/log.log

fi

echo "--catalina日志备份完成， 删除原始数据" >> ${ logpath}/log.log

rm -f ${catalina\_list}

else

echo "--catalina日志备份失败" >> ${logpath}/log.log

fi

else

echo "--无catalina日志" >> ${logpath}/log.log

fi

if [ ! -z "${localhost\_access\_log\_list}" ];then accessbz="localhost\_access-${year}-${month}-log.tar.bz2

"

tar jcf ${accessbz} ${localhost\_access\_log\_list} if [ "$?" == 0 ];then

access\_path="${logpath}/${accessbz}" access\_desc="access-${year}-${month}-log" accessid=python /opt/sh/oas/logupoas.py ${

vault\_name} ${access\_path} ${access\_desc} if [ "$?" == 0 ];then

echo "--access归档存储上传成功,Archiveid为:${

accessid}" >> ${logpath}/log.log

else

echo "--access归档存储上传失败" >> ${logpath}/ log.log

fi

echo "--localhost\_access日志备份完成， 删除原始数据"

>> ${logpath}/log.log

rm -f ${localhost\_access\_log\_list} else

echo "--localhost\_access日志备份失败" >> ${logpath

}/log.log

fi

else

echo "--无localhost\_access日志" >> ${logpath}/log.log

fi

if [ ! -z "${minishop\_list}" ];then accessbz="minishop-${year}-${month}-log.tar.bz2"

tar jcf ${accessbz} ${minishop\_list} if [ "$?" == 0 ];then

access\_path="${logpath}/${accessbz}" access\_desc="minishop-${year}-${month}-log" accessid=python /opt/sh/oas/logupoas.py ${

vault\_name} ${access\_path} ${access\_desc} if [ "$?" == 0 ];then

echo "--minishop归档存储上传成功,Archiveid为:${

accessid}" >> ${logpath}/log.log

else

echo "--minishop归档存储上传失败" >> ${logpath

}/log.log

fi

echo "--minishop日志备份完成， 删除原始数据" >> ${ logpath}/log.log

rm -f ${minishop\_list} else

echo "--minishop\_list日志备份失败" >> ${logpath}/

log.log

fi

else

echo "--无minishop\_list日志" >> ${logpath}/log.log

fi

# 致 谢

衷⼼感谢导师郑海永教授副教授对本⼈的精⼼指导。他的⾔传⾝教将使我终

⽣受益。

我在本科期间就开始在郑海永⽼师的指导下进⾏项⽬开发，郑⽼师除了在项

⽬上耐⼼的进⾏指导，在个⼈规划、科研⽔平提⾼以及⽣活⽅⾯都对我有很⼤的帮助，让我认识到了⾃⼰很多⽅⾯的不⾜，再次感谢。

感谢海信智能商⽤系统股份有限公司创新产品研究所的各位同事，在他们的帮助下，我对于商⽤产品的规划、设计、研发和维护流程有了更深刻的认识，⽽这促使我完成了本次论⽂。

感谢ThuThesis，它的存在让我的论⽂写作轻松⾃在了许多，让我的论⽂格式规整漂亮了许多。

# 个人简历、在学期间发表的学术论文与研究成果

## 个人简历

1991 年 7 ⽉ 9 ⽇出⽣于⼭东省诸城市。

2010 年 9 ⽉考⼊中国海洋⼤学电⼦系电⼦信息⼯程专业，2014 年 7 ⽉本科毕业并获得⼯学学⼠学位。

2014 年 9 ⽉免试进⼊中国海洋⼤学电⼦系攻读电⼦与通信⼯程⼯学学位⾄今。

## 发表的学术论文

[1] Zheng H, Wu B, Wei T, et al. Global Exponential Robust Stability of High- Order Hopfield Neural Networks with S-Type Distributed Time Delays[J]. Journal of Applied Mathematics, 2014, 2014.