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]。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 continuous 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 virtual 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 efficiency 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 implementation 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 Monitoring and Response Strategies

目 录

主要符号对照表

Internet	互联网
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	应用程序编程接口
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
CloudMonitor	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)

1 绪论

1.1 课题研究背景及意义

当今社会是一个“信息爆炸”的社会，尤其是进入互联网时代后，人们每天接触的信息和互联网产品远远大于互联网发展初期的年代[2]。随着中国互联网技术的不断发展以及 WEB 应用开发成本的不断降低，人们对于 WEB 应用的需求不再满足于通过缓慢的加载和复杂的操作获取资讯[3]，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[4] 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。

1.2 国内外研究现状

近几年，随着计算机技术和 Internet 的迅速发展，网络的信息量和访问量成几何数级增长，网络拥塞和服务超载日益成为网络及其服务器必须面对的严峻问题[5]。

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。

1.3 本文主要研究内容

1.3.1 主要工作

论文在满足应用正常运行的基础上，通过不同维度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。

1.3.2 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。介绍了本论文的主要工作内容、目标以及创新点。最后介绍了论文的编排结构。

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。

2 基于 Spring MVC 框架的 WEB 应用开发

本章首先介绍 WEB 应用开发的主要程序语言，然后对 Spring MVC 框架进行了详细的介绍。同时，对各种技术的应用反问做了初步的探讨，为后文系统性能优化工作做好理论工具上的准备。

2.1 应用开发语言

目前 WEB 应用开发主要分为前端开发和后端开发，其中前端开发以 HTML 和 JavaScript 结合为主，后端开发语言则比较多，主流的有 C 语言、PHP 语言、JAVA 语言以及 Python 语言等语言，不同语言之间的作用和特点都不同，但在提高系统稳定性和系统兼容性方面 JAVA 语言更有优势[6]。

2.1.1 JAVA 语言

2.1.1.1 JAVA 语言产生及发展

JAVA 语言是在 20 世纪 90 年代产生的，任职于太阳微系统的詹姆斯·高斯林等人于 1990 年代初开发 Java 语言的雏形，最初被命名为 Oak，目标设置在家用电器等小型系统的程序语言，应用在电视机、电话、闹钟、烤面包机等家用电器的控制和通信[7]。由于这些智能化家电的市场需求没有预期的高，Sun 公司放弃了该项计划。随着 1990 年代互联网的发展，Sun 公司看见 Oak 在互联网上应用的前景，于是改造了 Oak，于 1995 年 5 月以 Java 的名称正式发布。Java 伴随着互联网的迅猛发展而发展，逐渐成为重要的网络编程语言[8]。

在所有的编程语言中，应用最广泛的莫过于 JAVA 语言，该语言在面向对象的程序开发中一直有着突出的表现。因此在最近 10 几年间，JAVA 语言一直保持在编程语言排行的前两名，参见图 2-1。

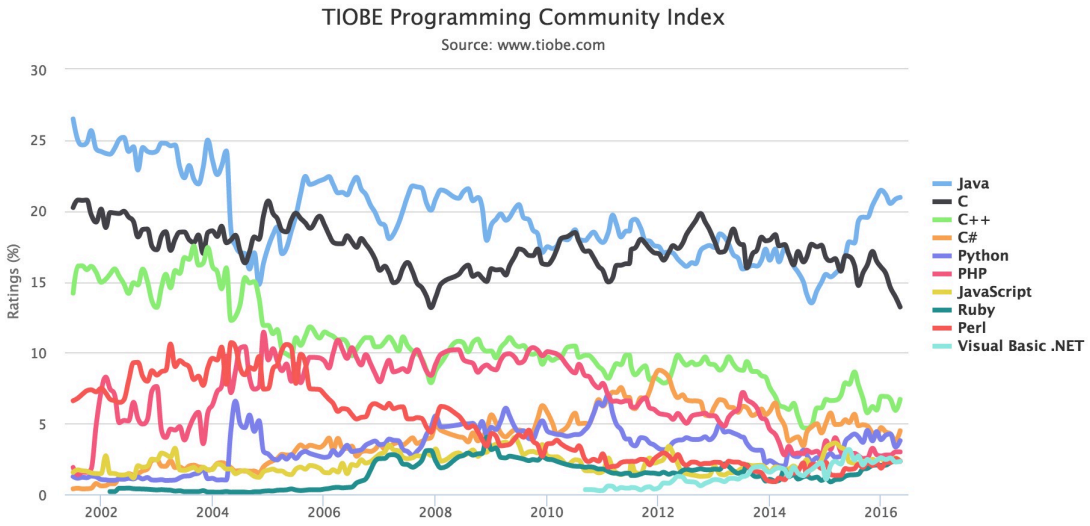

图 2-1　编程语言近年排行

2.1.1.2　JAVA 语言特点

因特网的普及推动了 JAVA 语言的发展，同时 JAVA 语言在网页制作上的应用也丰富了因特网的内容。两者可谓是相辅相成、相互促进。JAVA 语言的特点如表2-1。

表 2-1　模板文件。如果表格的标题很长，那么在表格索引中就会很不美观，所以要像 chapter 那样在前面用中括号写一个简短的标题。这个标题会出现在索引中。

特点	描述
可移植性	能移植于不同的使用平台，扩大了系统的使用范围
面向对象	集成了 C++ 的优点，并且在面向对象方面有新的扩展
多线程	可以让多个不同处理器同时进行工作，是数据库系统开发的最佳选择
分布式	在处理 TCP 等协议的同时也支持网络编程

2.2　Spring MVC 开发框架

Spring MVC 是一种基于 Web 开发的系统架构，目前主流的系统框架还有 Struts，但是 Spring MVC 架构引起优势依然处于不可替代的位置。在 Spring MVC

架构中，系统的终极目标被解聚、分块，并在不同的模块中表达出来，如此系统的耦合度得到大幅降低。同时，该框架的整体运行也是在模块驱动的基础之上的，用户首先发出请求，然后系统予以相应，即常规的请求-相应模型。

2.2.1 Spring MVC 架构的特点

Spring MVC 架构不同于传统的体系架构，在其发展的过程中吸收其他系统的特点，如支持 Userful 风格，灵活的本地化等。这些特征保证了 Spring MVC 框架在发展过程中，更加受到开发者的青睐。同时，吸收了前人的经验以后，Spring MVC 框架也具备专属的特点，比如用 Spring MVC 框架设计的 WEB 系统其耦合度较低，系统总体表现出简洁明了的风格，这样用户就能更快捷的掌握数据库系统的操作[2]。

Spring MVC 框架是在 Spring 框架的基础上发展起来的，因此 Spring MVC 框架集成了 Spring 框架的所有功能，而且两者在相互继承方面表现出很好的融合性。这一特点给系统开发带来了很多方便。除此之外，Spring MVC 模型在图形兼容性和数据验证的灵活性上都有先天的优势。正式因为这些特点，使 Spring MVC 模型在实际应用中发挥出巨大的优势，同时也在程序员中受到青睐。本文的 WEB 应用就是建立在该框架之上的。

2.2.2 Spring MVC 框架处理流程

Spring MVC 架构是一个基因用户请求的系统架构，其过程也要求有 Web 驱动的参加。具体过程现有用户发送请求到前端控制器，这时候前端控制器将请求委托给处理器；于是处理器就开始处理请求，即调用业务对象进行分析，分析完毕后将模型数据返回给处理器；然后处理器再将处理结果通过模型和视图返回给前端控制器；此时前端控制器显示的数据用户还无法处理，还需要进行视图渲染，最终在前端控制器显示出来，即用户发送的请求产生了响应。整个过程参见图 2-2，此图反映了用户请求-响应的过程。

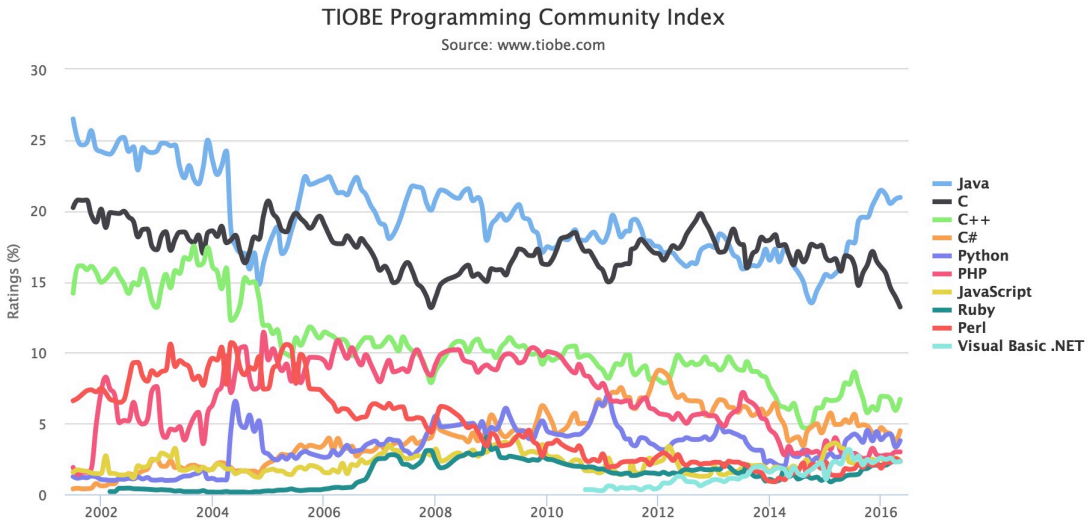

图 2-2　编程语言近年排行

2.2.2.1　体系的三层架构

MVC(Model View Controller), 即视图、模型和控制器三个英文单词的首字母缩写，按照应用的流程本系统可以在 Spring MVC 架构下分为三层，具体的架构为显示层、控制层和数据访问层。

1. 显示层。主要有 Spring 和 JSP 经过页面 REQUEST 请求，Spring 的 URL Mapping 对应到响应的控制器，通过对页面发送的请求，Spring 自动的去寻找响应的控制类，并将结构展示在页面上。
2. 控制层。也就是业务层，是系统中最重要的一层，这层主要是接受 URL 发送过来的业务请求，然后去访问 Service 层，把需要的数据装入缓存中，任何通知 Action 进行跳转。
3. 数据访问层。这层最重要的作用是对数据库中数据以及 XML 文件中的数据进行访问及操作，并把操作的结果反馈到业务逻辑层，将相应的结果交由下一层去处理。

这个三层架构的体系的应用很好的解决了系统资源的分配问题，而且将系统的耦合度降到了可以接受的最低水平。同时，关于异步处理问题上，本系统使用了 FDP 框架，FDP 框架是对 AJAX 的一种轻量级的封装，它的目标是使 AJAX 和 XML Http Request 更简单。通过 FDP 框架，我们只需要在相应的配置对应的 JAVA 方法以及相对于的实体类，我们很容易的就可以实现许多异步操作，使一些操作不用刷新页面即可达到相应的效果。可以认为 FDP 框架是对 MVC 三层框架的一种拓展，其本质还是在 MVC 框架内的。由此可见，笔者在系统体系的选择上更倾向于架构简单、可移植性好的系统框架模型。

2.3 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。这些都属于手段，但是想要达到最终目的则需要用到一些基础的软件工具，这些工具的使用将加快工作效率。

2.3.1 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 在 2001 年 1 月时推出最初版[9]。该软件被认为是当前 Java 开发效率最快的 IDE 工具。它集成了开发过程中实用的众多功能，几乎可以不用鼠标可以方便的完成你要做的任何事情，最大程度的加快开发的速度。简单而又功能强大。与其他的一些繁冗而复杂的 IDE 工具有鲜明的对比[10]。

2.3.2 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[11]。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。

2.3.3 Tomcat

WEB 应用的 Web 服务器采用的是有 Apache 软件基金会 (Apache Software Foundation) 下属的 Jakarta 项目开发的一个 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[12]。Tomcat7 支持最新的 Servlet 3.0 规范，而且技术先进、稳定性强一级免费的特点，深受 Java 爱好者的喜爱并得到了部分软件开发商的认可，成为目前比较流行的 Web 应用服务器。

2.3.4 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[13]。MySQL 的特点主要有:

- 为多种編程语言提供了 API。这些編程语言包括 C、C++、C♯、Java、Perl、PHP、Python、Ruby 等。
- 支持多线程，充分利用 CPU 资源，支持多用户。
- 提供 TCP/IP、ODBC 和 JDBC 等多种数据库连接途径。
- 提供用于管理、检查、優化数据库操作的管理工具。

- 可以处理拥有上千万条记录的大型数据库。

2.3.5 Couchbase

在互联网时代，我们面对的是更多的客户端，更低的请求延迟，这就需要对数据做大量的 Cache 以提高读写速度[14]。目前主流的 Cache 系统 memcached 和 redis 虽然有很熟的解决方案，但是也都有其局限性[15]：

- Cluster 支持不够。在扩容、负载均衡、高可用等方面存在明显不足。
- 持久化支持不好，出现问题后恢复的代价大。memcached 完全不支持持久化，redis 的持久化会造成系统间歇性的负载很高。

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 的对等网设计，smart client 直接获取整的集群的信息，在客户端实现负载均衡，整个集群没有单点失效，并且完全支持平行扩展。
- vBucket 的引入，完全实现了 auto sharding，可以方便灵活的把数据的子集在不同节点上移动，以实现集群动态管理。
- Couchbase 有一个非常专业的 web 管理界面，并且支持通过 RESTful API 管理，这也是 memcached, redis 不能企及的。

2.4 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 框架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三个方面。

2.4.1 前端开发

应用的前端相当于 MVC 架构中的视图层，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。

2.4.2 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。

2.4.3 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，业务数据主要包括用户的信息、用户操作记录、用户的商店、商品、销售信息等。

2.5 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，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。

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，它可以执行基于 Apache Ant 和 Apache Maven 的项目，以及任意的 Shell 脚本和 Windows 批处理命令。

2.6 本章总结

3 应用性能优化

3.1 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。

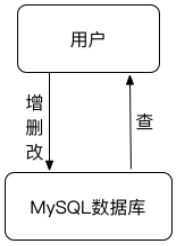

图 3-1 通常数据操作流程

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。

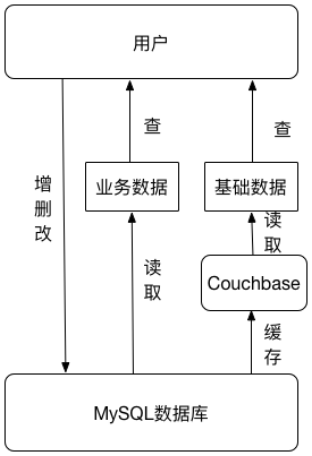

图 3-2 通常数据操作流程

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。

3.1.1 Couchbase 安装

1. 直接安装[14]

   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 be active and
     should not be.
   Please look at http://bit.ly/1ZAcLjD as for how to
     PERMANENTLY alter this setting.
   Minimum RAM required  : 4 GB
   System RAM configured : 3.70 GB

   Minimum number of processors required : 4 cores
   Number of processors on the system    : 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 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/”

2. Docker 安装[16]

```
# 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口和应用访问的端口，相关配置参见5.4

3.1.2 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。

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：

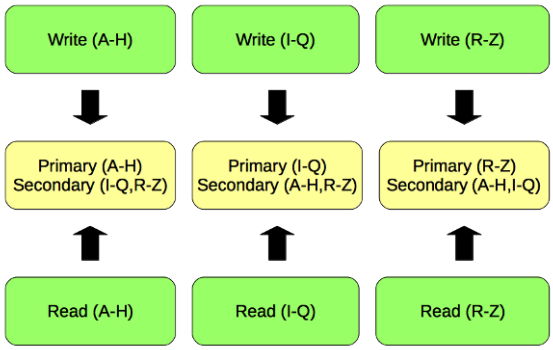

图 3-3　Couchbase 集群数据操作模型

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)

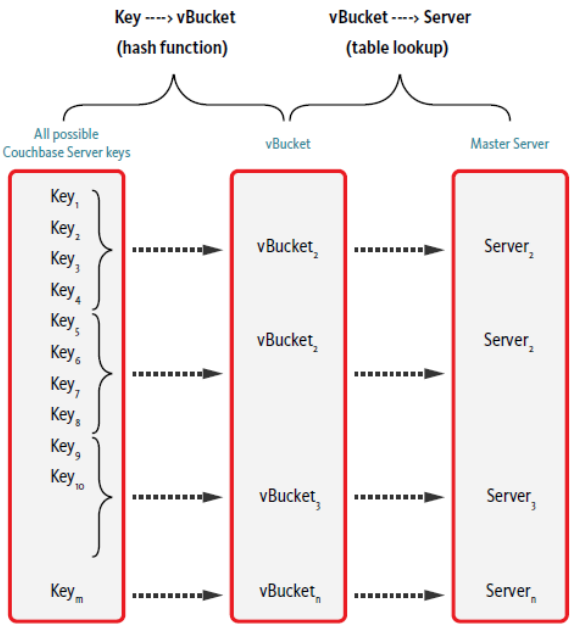

图 3-4　Couchbase 负载均衡模型

在设置标签页中的集群标签页下新建和配置集群信息

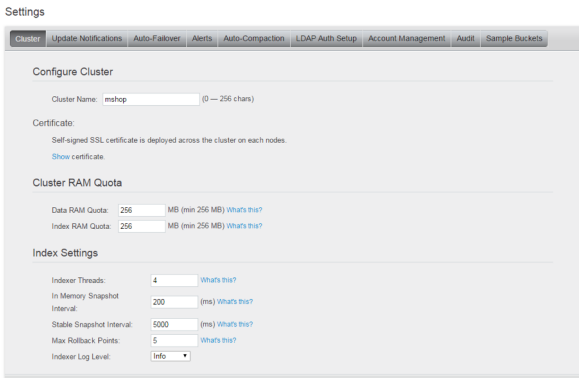

图 3-5　Couchbase 创建集群

在服务器节点标签页下新增服务器节点

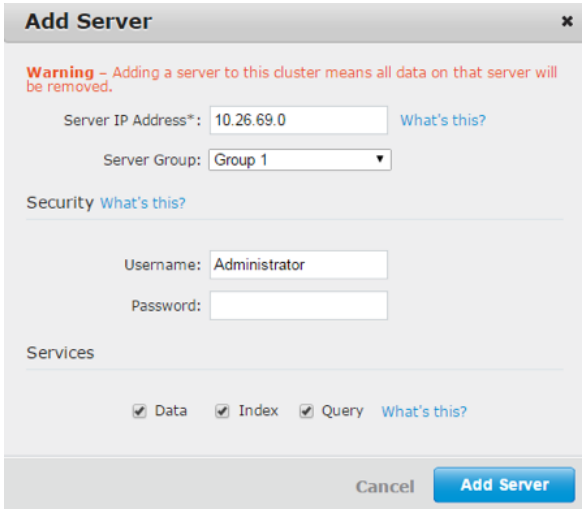

图 3-6　Couchbase 增加节点

增加完服务器节点后可以看到集群的基本信息

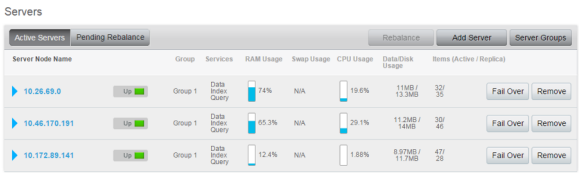

图 3-7　Couchbase 集群概览

3.1.3　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 10 -k http://test.hics.hisense.com:8081/client/mnsindex.html
This is ApacheBench, Version 2.3 <$Revision: 1748469 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking test.hics.hisense.com (be patient)
Completed 5000 requests
Completed 10000 requests
Completed 15000 requests
```

```
Completed 20000 requests
Completed 25000 requests
Completed 30000 requests
Completed 35000 requests
Completed 40000 requests
Completed 45000 requests
Completed 50000 requests
Finished 50000 requests


Server Software:        nginx/1.9.12
Server Hostname:        test.hics.hisense.com
Server Port:            8081

Document Path:          /client/mnsindex.html
Document Length:        36028 bytes

Concurrency Level:      10
Time taken for tests:   206.318 seconds
Complete requests:      50000
Failed requests:        0
Keep-Alive requests:    49504
Total transferred:      1816897520 bytes
HTML transferred:       1801400000 bytes
Requests per second:    242.34 [#/sec] (mean)
Time per request:       41.264 [ms] (mean)
Time per request:       4.126 [ms] (mean, across all
   concurrent requests)
Transfer rate:          8599.92 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       5
```

```
Processing:    20   41  19.1     38    1146
Waiting:       16   31   8.9     30    1058
Total:         20   41  19.2     38    1147

Percentage of the requests served within a certain time (ms)
  50%     38
  66%     44
  75%     47
  80%     49
  90%     54
  95%     62
  98%     75
  99%     86
 100%   1147 (longest request)
```

```
$ ab -n 50000 -c 10 -k http://test.hics.hisense.com/client/mnsindex.html
This is ApacheBench, Version 2.3 <$Revision: 1748469 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking test.hics.hisense.com (be patient)
Completed 5000 requests
Completed 10000 requests
Completed 15000 requests
Completed 20000 requests
Completed 25000 requests
Completed 30000 requests
Completed 35000 requests
Completed 40000 requests
Completed 45000 requests
```

```
16 Completed 50000 requests
17 Finished 50000 requests
18
19
20 Server Software:        Apache-Coyote/1.1
21 Server Hostname:        test.hics.hisense.com
22 Server Port:            80
23
24 Document Path:          /client/mnsindex.html
25 Document Length:        36028 bytes
26
27 Concurrency Level:      10
28 Time taken for tests:   247.137 seconds
29 Complete requests:      50000
30 Failed requests:        0
31 Keep-Alive requests:    49505
32 Total transferred:      1814097525 bytes
33 HTML transferred:       1801400000 bytes
34 Requests per second:    202.32 [#/sec] (mean)
35 Time per request:       49.427 [ms] (mean)
36 Time per request:       4.943 [ms] (mean, across all
      concurrent requests)
37 Transfer rate:          7168.40 [Kbytes/sec] received
38
39 Connection Times (ms)
40               min  mean[+/-sd] median   max
41 Connect:        0    0   0.1      0      12
42 Processing:    21   49  17.5     45     528
43 Waiting:       17   27  12.0     24     504
44 Total:         21   49  17.6     45     528
45
46 Percentage of the requests served within a certain time (ms
      )
```

```
 50%     45
 66%     49
 75%     51
 80%     53
 90%     63
 95%     69
 98%     88
 99%    104
100%    528 (longest request)
```

3.2 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[17]。

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：

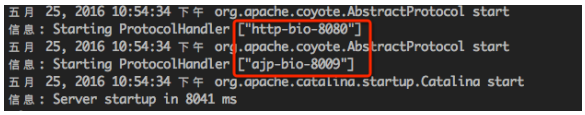

图 3-8　BIO 模式日志

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：

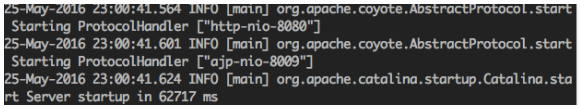

图 3-9　NIO 模式日志

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所示日志，表示使用的是 NIO 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[18]。

APR(Apache Portable Runtime) 是一个高可移植库，它是 Apache HTTP Server 2.x 的核心。APR 有很多用途，包括访问高级 IO 功能 (例如 sendfile,epoll 和 OpenSSL)，OS 级别功能 (随机数生成，系统状态等等)，本地进程管理 (共享内存，NT 管道和 UNIX sockets)。这些功能可以使 Tomcat 作为一个通常的前台 WEB 服务器，能更好地和其它本地 web 技术集成，总体上让 Java 更有效率作为一个高性能 web 服务器平台而不是简单作为后台容器。

在产品环境中，特别是直接使用 Tomcat 做 WEB 服务器的时候，应该使用 Tomcat Native 来提高其性能，要测 APR 给 tomcat 带来的好处最好的方法是在慢速网络上（模拟 Internet），将 Tomcat 线程数开到 300 以上的水平，然后模拟一大堆并发请求。

如果不配 APR，基本上 300 个线程狠快就会用满，以后的请求就只好等待。但是配上 APR 之后，并发的线程数量明显下降，从原来的 300 可能会马上下降到只有几十，新的请求会毫无阻塞的进来。在局域网环境测，就算是 400 个并发，也是一瞬间就处理/传输完毕，但是在真实的 Internet 环境下，页面处理时间只占 0.1% 都不到，绝大部分时间都用来页面传输。如果不用 APR，一个线程同一时间只能处理一个用户，势必会造成阻塞。所以生产环境下用 apr 是非常必要的。

3.2.1　开启 APR 模式

3.2.2　APR 模式对于系统性能的影响

3.3　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。

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所示[19]。

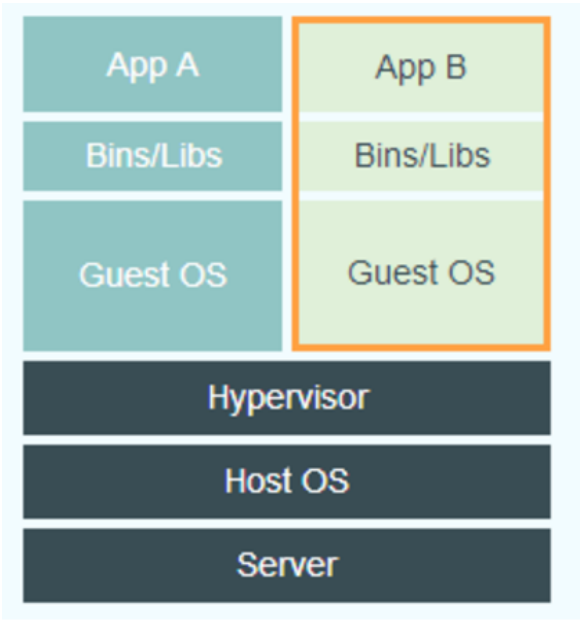

图 3-11　传统虚拟机模型

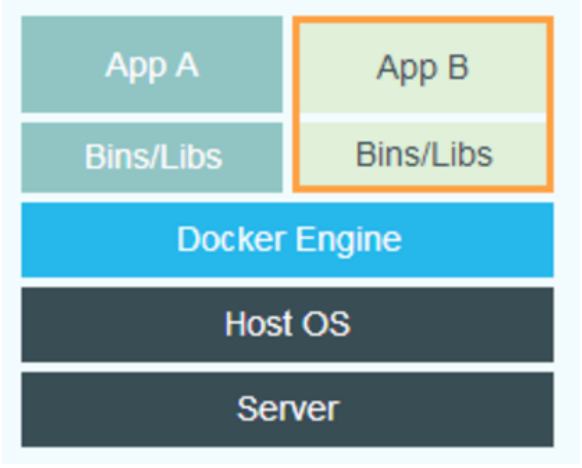

图 3-12　Docker 容器模型

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。

- 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

  对开发和运维（DevOps）人员来说，最希望的就是一次创建或配置，可以在任意地方正常运行。

  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 可以在很多平台上运行，无论是物理机、虚拟机、公有云、私有云，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。
- 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

3.3.1 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，其中一个的端口为 3306，另一个的端口为 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。

3.3.2 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。

3.4 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。

负载均衡（Load balancing）是一种计算机网络技术，用来在多个计算机（计算机集群）、网络连接、CPU、磁盘驱动器或其他资源中分配负载，以达到最佳化资源使用、最大化吞吐率、最小化响应时间、同时避免过载的目的。

使用带有负载均衡的多个服务器组件，取代单一的组件，可以通过冗余提高可靠性。

负载平衡最重要的一个应用是利用多台服务器提供单一服务，这种方案有时也称之为服务器农场。通常，负载平衡主要应用于 Web 网站，大型的 Internet Relay Chat 网络，高流量的文件下载网站，NNTP（Network News Transfer Protocol）服务和 DNS 服务。现在负载平衡器也开始支持数据库服务，称之为数据库负载平衡器。对于互联网服务，负载平衡器通常是一个软体程序，这个程序侦听一个外部端口，互联网用户可以通过这个端口来访问服务，而作为负载平衡器的软体会将用户的请求转发给后台内网服务器，内网服务器将请求的响应返回给负载平衡器，负载

平衡器再将响应发送到用户，这样就向互联网用户隐藏了内网结构，阻止了用户直接访问后台（内网）服务器，使得服务器更加安全，可以阻止对核心网络栈和运行在其它端口服务的攻击。当所有后台服务器出现故障时，有些负载平衡器会提供一些特殊的功能来处理这种情况。例如转发请求到一个备用的负载平衡器、显示一条关于服务中断的消息等。负载平衡器使得 IT 团队可以显著提高容错能力。它可以自动提供大量的容量以处理任何应用程序流量的增加或减少[20]。

负载均衡的主要特点有:

1. 解决并发压力，提高应用处理性能（增加吞吐量，加强网络处理能力）；
2. 提供故障转移，实现高可用;
3. 通过添加或减少服务器数量，提供网站伸缩性（扩展性）；
4. 安全防护;（负载均衡设备上做一些过滤，黑白名单等处理）；

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 协议端口 (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 对应端口，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。

3.5 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。

4 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，5.7 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。

2. 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 5.7.8 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('{"key1": "value1", "key2": "
    value2"}');
```

   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：

(a) 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

(b) 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。

3. 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
```

4. 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 resize，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 innodb_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);
```

5. 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。

存储引擎是存储数据、为存储的数据建立索引以及更新、查询数据等技术的实现方法。因为在关系数据库中数据的存储是以表的形式存储，所以存储引擎也可以称为表类型[21]。

Oracle 和 SQL Server 等数据库中只有一种存储引擎，所有数据存储管理机制都一样。而 MySQL 数据库提供了多种存储引擎。用户可以根据不同的需求为数据表选择不同的存储引擎，用户也可以根据具体的需求编写自定义存储引擎。MySQL 数据库提供的存储引擎如表 4-1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。MyISAM 是 MySQL 早期的 ISAM 存储引擎的升级版本，也是 MySQL 默认的存储引擎，而 InnoDB 是由第三方软件公司 Innobase 所开发，其最大的特点是提供事务控制的特性[22]。

相比 MyISAM，InnoDB 数据引擎具有支持事务、行级锁和外键约束等功能。

1. 支持事务安全。InnoDB 存储引擎最重要的一点就是对事务安全的支持，这也是让它成为最流行的存储引擎很重要的原因，而且实现了 SQL92 标准所定义的 4 个级别 (READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE).

2. 数据库多版本读取。InnoDB 在事务支持的同时，为了保证数据的一致性以及并发时刻的性能，通过对 un-do 信息的聚簇索引实现对数据的多版本读取。
3. 锁定机制的改进。InnoDB 改变了 MyISAM 的锁机制，实现了行锁。虽然 InnoDB 的行锁机制是通过索引来完成的，但是由于数据库中 99% 的 SQL 语句都是通过索引来检索数据，所以行锁机制为 InnoDB 在承受高并发下的环境下增强了竞争力。
4. 实现外键。InnoDB 实现了外键引用这一数据库的重要特性，使得在数据库端控制部分数据的完整性成为可能。

4.1 InnoDB 引擎参数优化

目前来说，InnoDB 是为 Mysql 处理巨大数据量时的最大性能设计。它的 CPU 效率可能是任何其它基于磁盘的关系数据库引擎所不能匹敌的。在数据量大的网站或是应用中 Innodb 是倍受青睐的。另一方面，在数据库的复制操作中 Innodb 也是能保证 master 和 slave 数据一致有一定的作用[23]。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。

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，主要缓存 innodb 表的索引，数据，插入数据时的缓冲。为 Innodb 加速优化首要参数。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。
2. 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 innodb_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 innodb_thread_concurrency 参数来配置线程的并发数，提升效率。

3. 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，这个值分配的大小和数据库的写入速度，事务大小，异常重启后的恢复有很大的关系，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。

4. 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
```

4.2 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。

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。

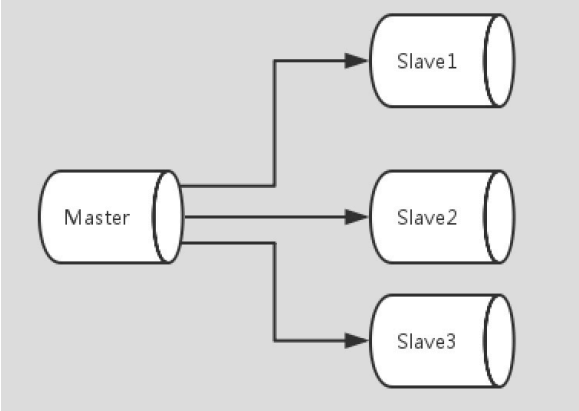

图 4-1 MySQL 复制示意图

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。
4. 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。

4.2.1 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。

复制的过程主要分为三步：

1. master 将改变记录到二进制日志 (binary log) 中（这些记录叫做二进制日志事件，binary log events）；
2. slave 将 master 的 binary log events 拷贝到它的中继日志 (relay log)；
3. slave 重做中继日志中的事件，将改变反映它自己的数据。

图 4-2描述了复制的过程

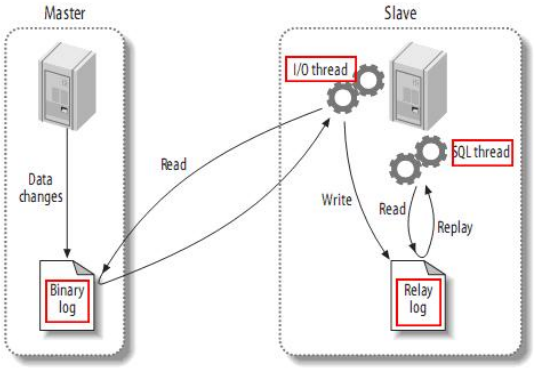

图 4-2　MySQL 复制流程

该过程的第一部分就是 master 记录二进制日志。在每个事务更新数据完成之前，master 在二日志记录这些改变。MySQL 将事务串行的写入二进制日志，即使事务中的语句都是交叉执行的。在事件写入二进制日志完成后，master 通知存储引擎提交事务。

下一步就是 slave 将 master 的 binary log 拷贝到它自己的中继日志 (relay log)。首先，slave 开始一个工作线程——I/O 线程。I/O 线程在 master 上打开一个普通的连接，然后开始 binlog dump process。Binlog dump process 从 master 的二进制日志中读取事件，如果已经跟上 master，它会睡眠并等待 master 产生新的事件。I/O 线程将这些事件写入中继日志。

SQL slave thread 处理该过程的最后一步。SQL 线程从中继日志读取事件，更新 slave 的数据，使其与 master 中的数据一致。只要该线程与 I/O 线程保持一致，中继日志通常会位于 OS 的缓存中，所以中继日志的开销很小。

此外，在 master 中也有一个工作线程：和其它 MySQL 的连接一样，slave 在 master 中打开一个连接也会使得 master 开始一个线程。复制过程有一个很重要的

限制——复制在 slave 上是串行化的，也就是说 master 上的并行更新操作不能在 slave 上并行操作。

4.2.2 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，需要这两个服务器一个主服务器提供数据库的操作，另一个服务器作为从服务器，复制主数据库的数据。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 参数为 1 表示每进行 1 次事务提交之后，MySQL 将进行一次 fsync 之类的磁盘同步指令来将 binlog_cache 中的数据强制写入磁盘, 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 'repl'@'%'
  IDENTIFIED BY 'replpassword';
```

其中 repl 为用户名，replpassword 为密码

2. 配置 master 信息

```
change master to master_host ='ip', master_port = port
  , master_user = 'repl', master_password = '
  replpassword', 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口，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。

3. 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。

4.2.3 延迟复制设计

为了避免上述双主服务器均出现问题后无法提供数据服务的现象，在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;
```

4.3 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
2 # -*- coding: utf-8 -*-
3 import oss2
4 import sys
5 auth = oss2.Auth('AccessKeyId', 'AccessKeySecret ')
6 endpoint='http://oss-cn-beijing-internal.aliyuncs.com'
7 bucket = oss2.Bucket(auth, endpoint, 'mysqlbk')
8 bucket.put_object_from_file(sys.argv[1], '/mnt/
    mysqldump/'+sys.argv[1])
```

   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-cn-beijing-internal.aliyuncs.com)，以及访问 OSS 的 AccessKeyId 和 AccessKeySecret。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。

2. 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、端口、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：

```
#!/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
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 "*.tar.bz2" -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
```

3. 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。

4.4 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。

5 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。

5.1 阿里云云监控应用

云监控（CloudMonitor）是一项针对阿里云资源和互联网应用进行监控的服务。云监控服务可用于收集获取阿里云资源的监控指标，探测互联网服务可用性，以及针对指标设置警报。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的云监控来实施。通过云监控服务能够监控云服务器 ECS、云数据库 RDS 和负载均衡等各种阿里云服务资源，同时也能够通过 HTTP,ICMP 等通用网络协议监控互联网应用的可用性。

在监控配置之前，需要对目前阿里云资源进行统计，如表 5-1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	生产环境地址	生产环境应用的负载均衡入口地址
	测试环境地址	测试环境应用的负载均衡入口地址
	主数据库地址	生产环境主数据库的访问地址和端口
	从数据库地址	生产环境从数据库的访问地址和端口
	Couchbase 地址	生产环境 Couchbase 入口地址
云服务器 ECS	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
负载均衡	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
CDN	web	WEB 应用访问域名
	img	图片服务器域名

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
	fileserver	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 Couchbase，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所示：

监控地址 (全部)	类型 (全部)	监控频率	杭州	青岛	北京
mshop http://mshop.hics.hisense.com/	HTTP	5分钟	正常 35毫秒	正常 14毫秒	正常 3毫秒
dbmaster http://test.hics.hisense.com	TCP	5分钟	正常 34毫秒	正常 12毫秒	正常 2毫秒
mysql http://test.hics.hisense.com	TCP	1分钟	正常 61毫秒	正常 26毫秒	正常 4毫秒
test http://test.hics.hisense.com	HTTP	5分钟	正常 35毫秒	正常 13毫秒	正常 2毫秒
Couchbase http://test.hics.hisense.com:8091/pools/nodes	HTTP	5分钟	正常 67毫秒	正常 31毫秒	正常 12毫秒

图 5-1　站点监控状态图

2. 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

3. 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

4. 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

5.2　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。

5.2.1　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[24]。

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[25]。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                   = "3.0.6"
    Executables               = "/usr/sbin"
    Man pages                 = "/usr/share/man"
    Libraries                 = "/usr/lib64"
    Header files              = "/usr/include"
    Arch-independent files    = "/usr/share"
    Documentation files       = "/usr/share/doc/
       heartbeat"
    State information         = "/var"
    System configuration      = "/etc"
    Init (rc) scripts         = "/etc/rc.d/init.d"
    Init (rc) defaults        = "/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.
```

2. 配置 Heartbeat 软件，实现两个服务器的心跳监听

heartbeat 主要的配置文件有 3 个，分别是 authkeys,ha.cf 和 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
```

该文件为 heartbeat 的认证文件，该文件主要是用于集群中两个节点的认证，采用的算法和密钥 (如果有的话) 在集群中节点上必须相同，目前提供了 3 种算法：crc/md5/sha1。其中 crc 不能够提供认证，它只能够用于校验数据包是否损坏，而 sha1/md5 需要一个密钥来进行认证，从资源消耗的角度来讲，md5 消耗的比较多，sha1 次之，因此建议一般使用 sha1 算法。

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。

3. 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"
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
        echo "--[OK]Mysql sync is OK" >>
           $MONITOR_LOG_PATH
    else
        echo "--[ERROR]Mysql sync is ERROR" >>
           $MONITOR_LOG_PATH
    fi
}
function check_tomcat_health (){
    source $tomcat_heal_script
    if [ $? = 0 ] ;then
        echo "--[OK]Tomcat health is OK" >>
           $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 "system_monitor is Running!"
    else
        echo "system_monitor is stopped!"
    fi
}

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。

5.2.2 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 A和附录 B来查看

5.2.3 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 **??**。

2. 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 Master_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 **??**

5.3 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。

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 SendTemplateSMS.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。

5.4 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 E, 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。

5.5 本章总结

6 总结与展望

6.1 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。

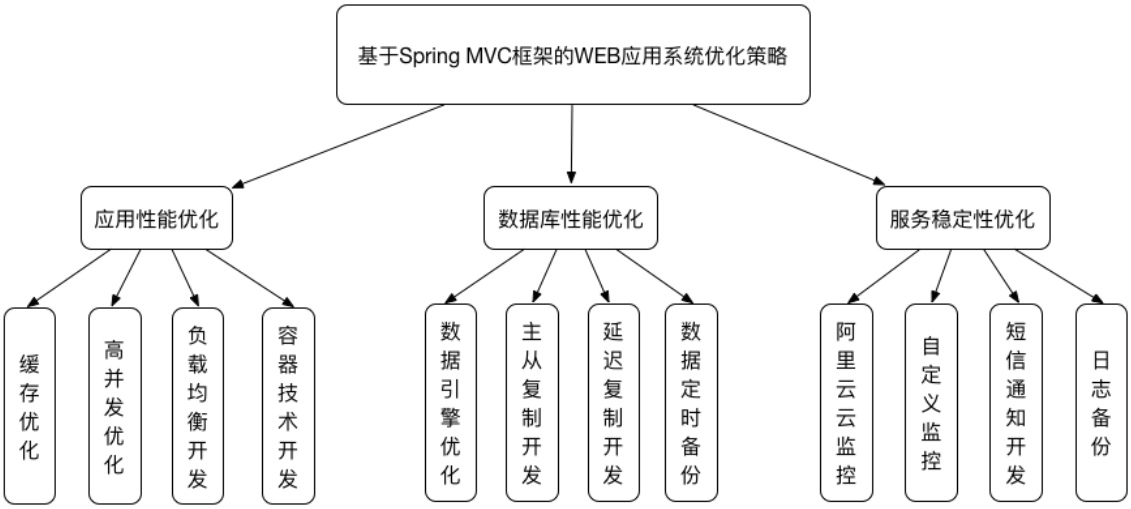

图 6-1 系统优化研究总结

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。

6.2 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。
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。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。

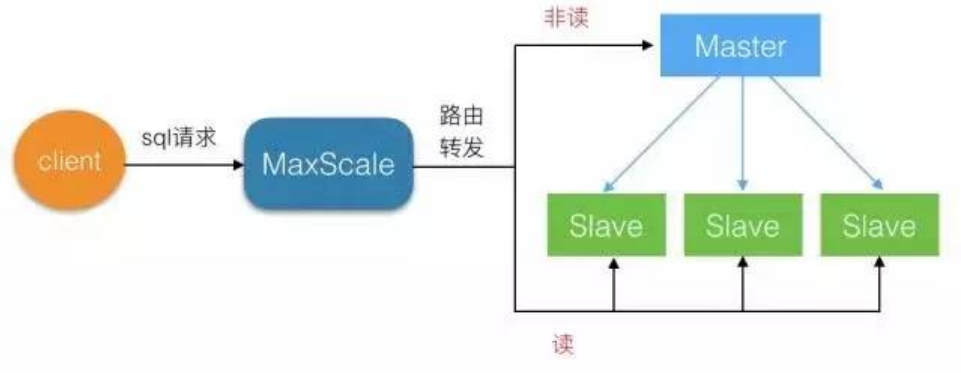

图 6-2　数据库读写分离流程

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。

3. 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。
4. 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 接口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] 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.

[4] 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.

[5] 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.

[6] Suzumura T, Trent S, Tatsubori M, et al. Performance comparison of web service engines in php, java and c. Web Services, 2008. ICWS'08. IEEE International Conference on. IEEE, 2008. 385–392.

[7] Byous J. Java technology: The early years. Sun Developer Network, 1998..

[8] wikipedia. Java. https://zh.wikipedia.org/wiki/Java. Accessed January 6, 2017.

[9] Jemerov D. Implementing refactorings in intellij idea. Proceedings of the 2nd Workshop on Refactoring Tools. ACM, 2008. 13.

[10] wikipedia. Intellij idea. https://zh.wikipedia.org/wiki/IntelliJ_IDEA. Accessed November 16, 2017.

[11] Smart J F, et al. An introduction to maven 2. JavaWorld Magazine. Available at: http://www. javaworld. com/javaworld/jw-12-2005/jw-1205-maven. html, 2005..

[12] Brittain J, Darwin I F. Tomcat: The Definitive Guide: The Definitive Guide. " O'Reilly Media, Inc.", 2007.

[13] Greenspan J, Bulger B. MySQL/PHP database applications. John Wiley & Sons, Inc., 2001.

[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] Merkel D. Docker: lightweight linux containers for consistent development and deployment. Linux Journal, 2014, 2014(239):2.

[17] Vukotic A, Goodwill J. Securing tomcat with ssl. Apache Tomcat 7. Springer, 2011: 141–154.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] Schwartz B, Zaitsev P, Tkachenko V. High performance MySQL: optimization, backups, and replication. " O'Reilly Media, Inc.", 2012.

[24] 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.

[25] 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.

附录 A　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" >> $LOG_PATH
        TomcatServiceCode=$(curl -I -s -m 10 -o /dev/null
           -w %{http_code} --connect-timeout 10 http://$1
           :8089/MiniShop/index.jsp)
        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 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-Tomcat服务不存在" >> $LOG_PATH
        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" >> $LOG_PATH
        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" >> $LOG_PATH
    else
        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 B　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='root'
            key_filename = '/root/.ssh/id_rsa'
            client.connect(host, port=port, username=
               username, key_filename = key_filename)
            if action == 'monitor':
```

```
            stdin, stdout, stderr = client.exec_command
                ("systemctl is-active tomcat.service")
            if stderr.read():
                print stderr.read()
                sys.exit(105)
            else:
                status=stdout.read().strip()
                if status == 'active':
                    print "运行正常"
                    sys.exit(100)
                elif status == 'failed':
                    print "服务停止"
                    sys.exit(102)
                elif status == 'unknown':
                    print "未知服务"
                    sys.exit(103)
                elif status == 'deactivating':
                    print "服务重启"
                    sys.exit(105)
                else:
                    print "未知状态"
                    sys.exit(106)
        elif action == 'restart':
            stdin, stdout, stderr = client.exec_command
                ("systemctl restart tomcat.service")
            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 C　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
```

```
}
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 "  权重python脚本执行失败" >> $MySQL_LOG_PATH
    fi
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 status
       Current Stop!" >> $MySQL_LOG_PATH
    #短信通知
    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
if [ $DB1_OK = 1 ] && [ $DB2_OK = 0 ] ; then
    echo (date '+%Y-%m-%d %H:%M:%S') "    Mysql db2 Current
        stop!" >> $MySQL_LOG_PATH
    #短信通知
    SMS_datas='DB2数据库健康状态异常, 请查看! '
```

```
    send_sms $SMS_datas
    #调整阿里云负载均衡DBSLB权重
    check_result=(python /opt/sh/slb/setBackendServers.py
       100 0)
    if [ "$?" == 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} ${MYSQL_CONTAINER_ID})
    if [ "$?" == 0 ];then
        echo "    重启DB2成功: ${mysql_restart_result}"
    else
        echo "    重启DB2失败: ${mysql_restart_result}"
    fi
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 Current
        Stop!" >> $MySQL_LOG_PATH
    #短信通知
    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 "  重启DB1成功: ${mysql_restart_result}"
    else
        echo "  重启DB1失败: ${mysql_restart_result}"
    fi
fi
```

附录 D　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${DB1_USER} -p${
   DB1_PASSWORD} -P${DB1_PORT}"
Connect_db2="$MYSQL -h${DB2_HOST} -u${DB2_USER} -p${
   DB2_PASSWORD} -P${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 E　Tomcat 健康监控脚本

```
#/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.