谨以此论文献给我的导师和亲人!

——————————— 武斌

基于 Spring MVC 架构 WEB 应用系统优化研究

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 应用系统优化研究

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
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
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)

1 绪论

1.1 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]。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] 不再满足于通过缓慢的加载和复杂的操作获取资讯[4]，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] 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

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]。

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 论文主要研究内容

1.3.1 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。

1.3.2 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。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。

2 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。

2.1 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]。

2.1.1 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，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。

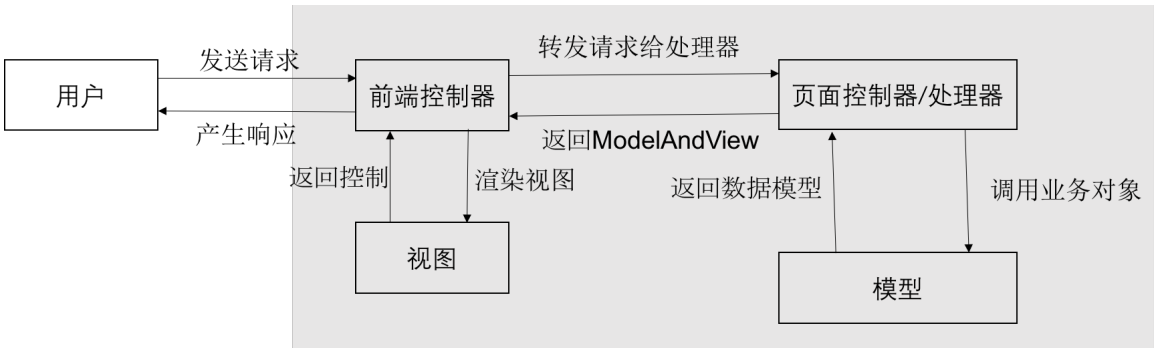

图 2-1 Spring MVC 请求响应过程

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]。

2.1.2 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。
2. 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。
3. 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。

2.2 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。

2.2.1 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]。

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]。

2.2.2 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[10]。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。

2.2.3 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]。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。

2.2.4 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[12]。MySQL 的特点主要有[13]：

- 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口，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。

2.2.5 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]。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]：

- 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 接口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。

2.3 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。

2.3.1 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。

2.3.2 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。

2.3.3 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。

2.4 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]。

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]。

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]。具体的自动构建及部署流程如下：

2.4.1 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]。

将 Jenkins 安装文件 jenkins.war 拷贝到运行中的 Tomcat 应用目录中，访问 Tomcat 地址完成 Jenkins 的相关配置后再次访问 Tomcat 地址进入到图2-2页面表示安装配置成功。

图 2-2　Jenkins 使用界面

2.4.2　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]，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2-3所示：

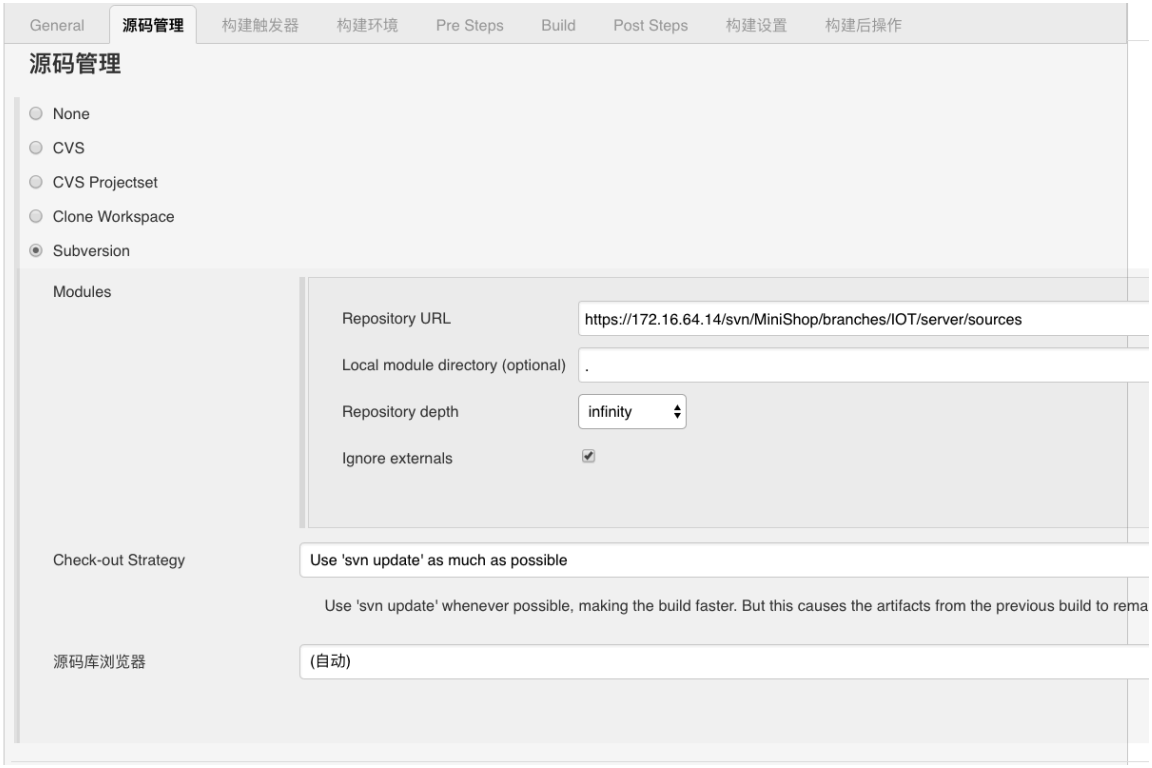

图 2-3　Jenkins SVN 配置

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。

3. 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。

4. 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]，考虑到本项目的实际开发过程在这里选择项目 pom2.xml

5. 在 Post Steps 配置构建完成后的操作，首先需要在"Run only if build succeeds 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。

2.4.3 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 Jenkins“系统管理-系统设置”页面中会出现如图2-4配置：

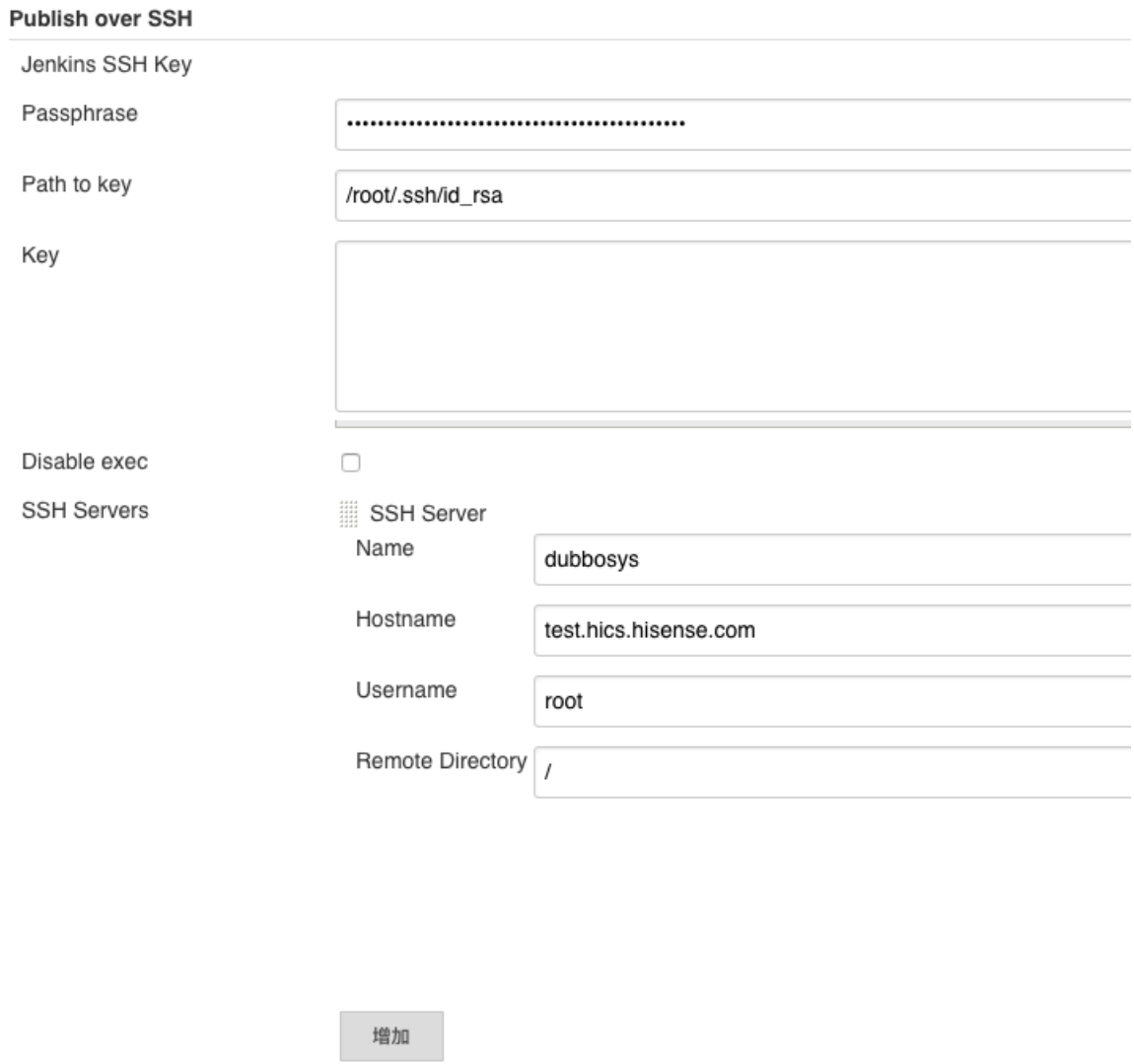

图 2-4　Publish Over SSH 插件配置

按照不同项目配置 SSH Server 信息和登录验证信息等。

3. 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：

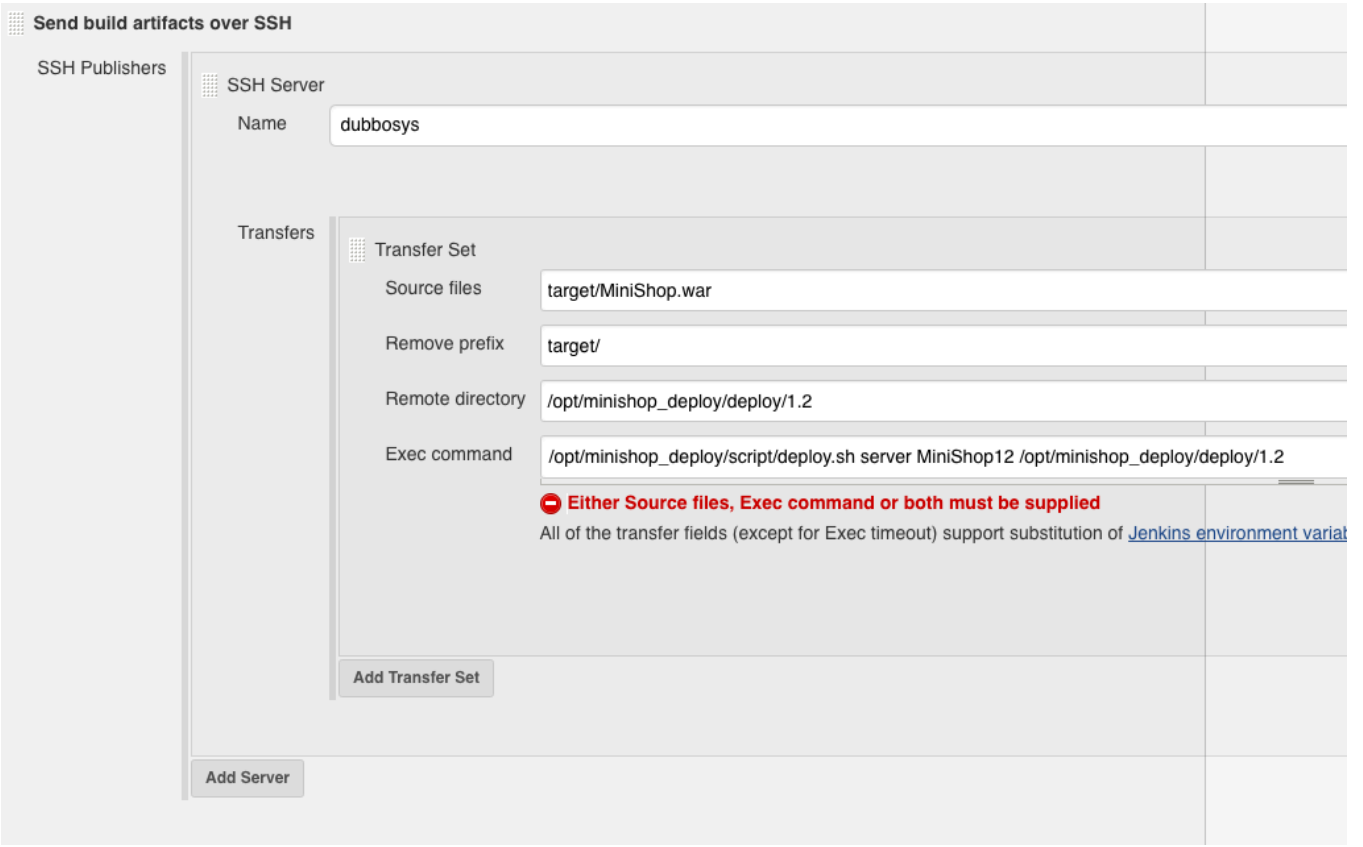

图 2-5　Jenkins 自动部署配置

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。

4. 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。

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。

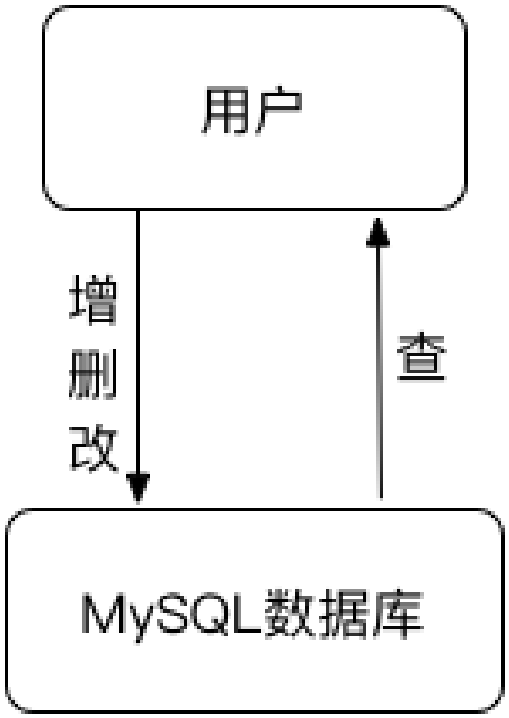

图 3-1 通常数据操作流程

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。

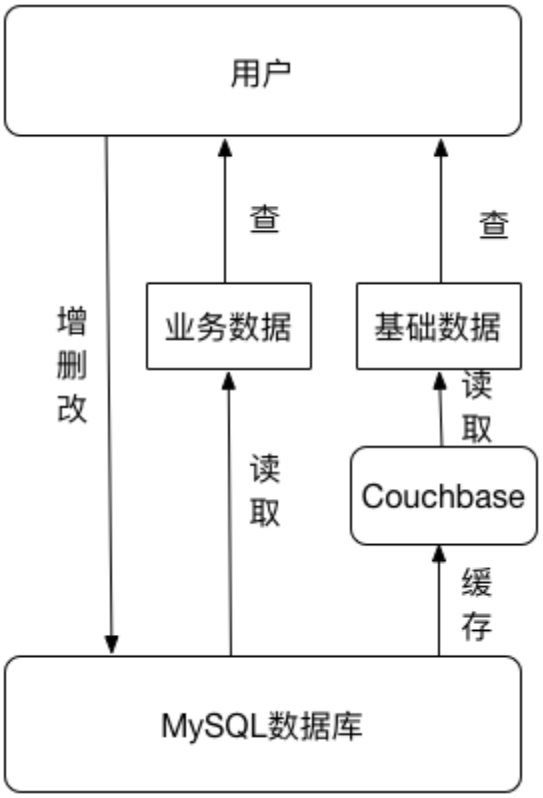

图 3-2　通常数据操作流程

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 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[22]。

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：

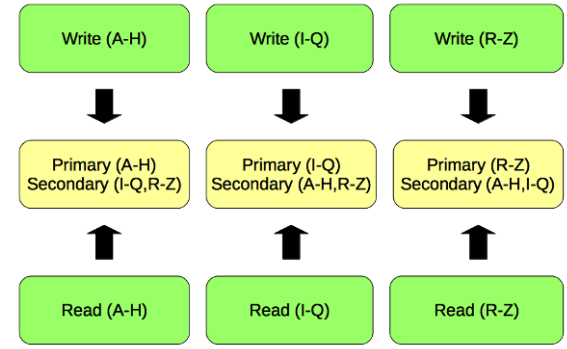

图 3-3　Couchbase 集群数据操作模型

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)

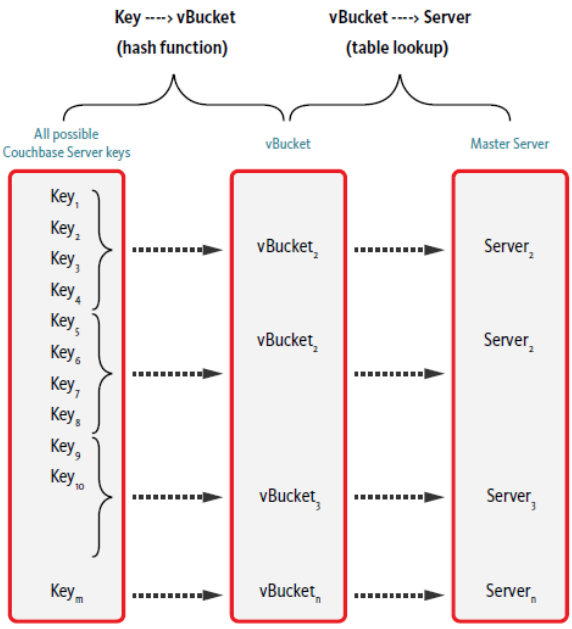

图 3-4　Couchbase 负载均衡模型

在设置标签页中的集群标签页下新建和配置集群信息

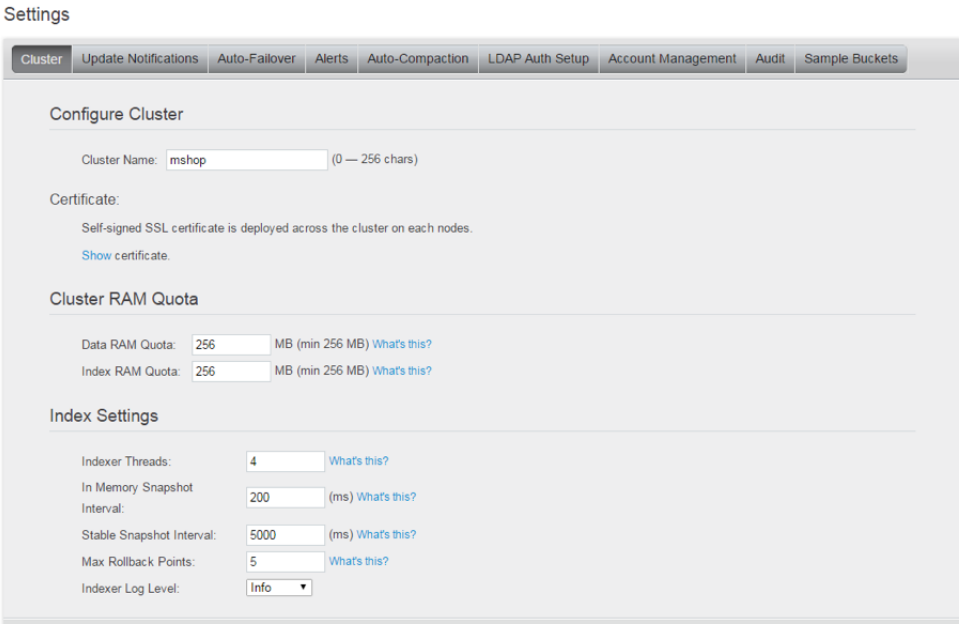

图 3-5　Couchbase 创建集群

在服务器节点标签页下新增服务器节点

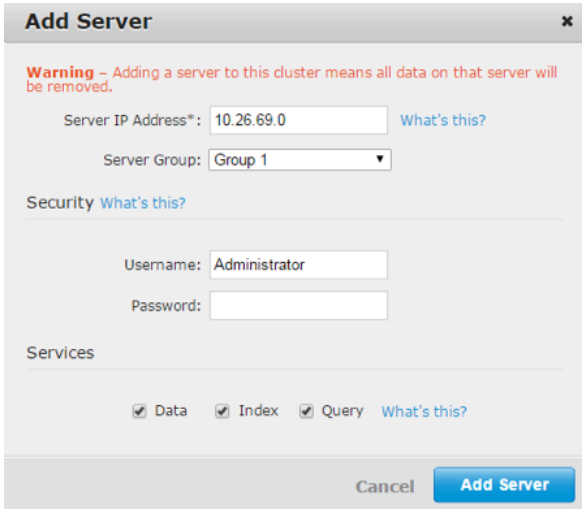

图 3-6　Couchbase 增加节点

增加完服务器节点后可以看到集群的基本信息

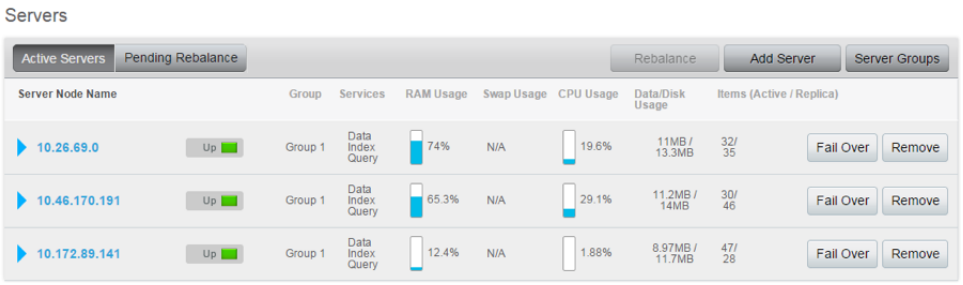

图 3-7　Couchbase 集群概览

3.1.2　Couchbase 缓存对系统性能影响

通过 ab 命令对测试应用进行测试，在开启 Couchbase 和关闭 Couchbase 的情况下，分别模拟 10000 个需要向后台请求数据的请求 (例如获取商户信息的请求 getUserInfo.action)，根据测试的参数及结果来对比分析 Couchbase 缓存对于应用系统性能的影响, 结果如表3-1所示：

表 3-1　APR 模式性能对比表

测试参数	开启 Couchbase	关闭 Couchbase
单次请求用时	0.670(s)	0.113(s)
请求失败次数	82	7

通过测试可以发现，开启 Couchbase 后系统的单次请求用时为 0.670 秒, 而不

使用 Couchbase 时单次请求用时为 0.113 秒，效率提升在 83% 左右，这表示开启缓存后进行数据请求时，只需要耗费一次长时间的请求，数据建立缓存后获取效率将大大提升。除此之外，请求的失败次数也大大降低，这对于系统的稳定性来说提升特别明显。

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[23]。

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：

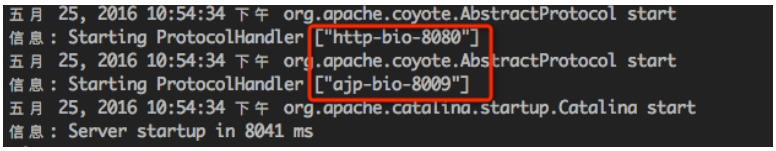

图 3-8 BIO 模式日志

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：

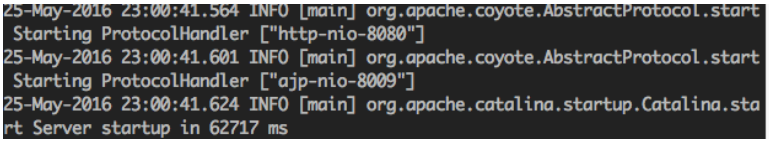

图 3-9 NIO 模式日志

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 APR 模式：

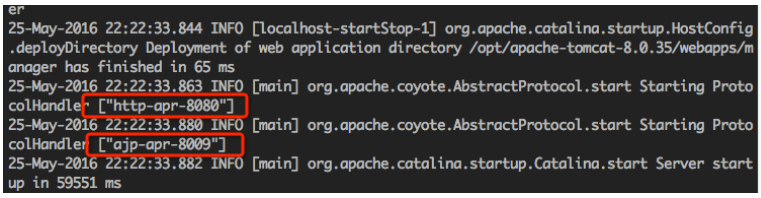

图 3-10　APR 模式日志

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[24]。

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，例如内存配置和接口；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[25]。

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[26]，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。

3.2.1　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 接口，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。

2. 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。

3. 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.library.path=/usr/local/
   apr/lib"
```

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 模式：

```
#禁用SSl
<Listener className="org.apache.catalina.core.
   AprLifecycleListener" SSLEngine="off" />
```

除此之外，还需要修改 Tomcat 连接器的参数，通过调整优化 Tomcat 连接器，可以使 Tomcat 对于用户的请求处理更佳适合于本地的服务器现状，对于并发的效果更好，具体的参数如表 3-2所示：

表 3-2　Tomcat 连接器调整参数

参数项	参数内容	参数描述
port	8089	访问端口
maxHttpHeaderSize	8192	http 请求头信息的最大程度
maxThreads	300	可以创建的最大线程数量
processorCache	1000	协议处理程序缓存
acceptCount	400	队列请求数量
minSpareThreads	50	最小备用线程数
URIEncoding	utf-8	Tomcat 容器的 URL 编码格式
acceptorThreadCount	8	接收线程的进程数
enableLookups	false	返回客户端真实的 IP 地址信息
redirectPort	8443	请求转发端口
connectionTimeout	120000	连接超时时间，毫秒
keepAliveTimeout	120000	长连接的超时时间，毫秒
maxKeepAliveRequests	65535	最大长连接个数
disableUploadTimeout	true	上传时是否使用超时机制
compression	on	启用压缩
compressionMinSize	2048	当超过最小数据大小才进行压缩
noCompressionUserAgents	gozilla, traviata	哪些客户端发出的请求不压缩

4. 配置完成后，重启 Tomcat，在日志文件中包含图 3-10中的日志信息表示 APR 模式开启成功。

3.2.2 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3-3所示。根据统计结果可以发现，在

表 3-3 APR 模式性能对比表

请求次数	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。

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[27]。

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[28]。

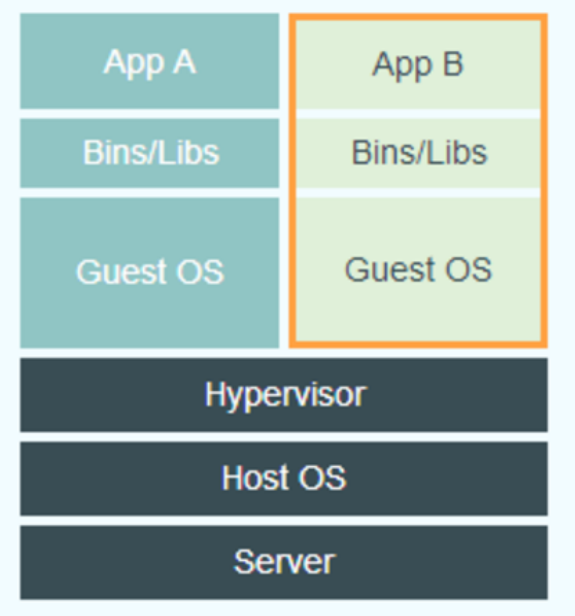

图 3-11　传统虚拟机模型

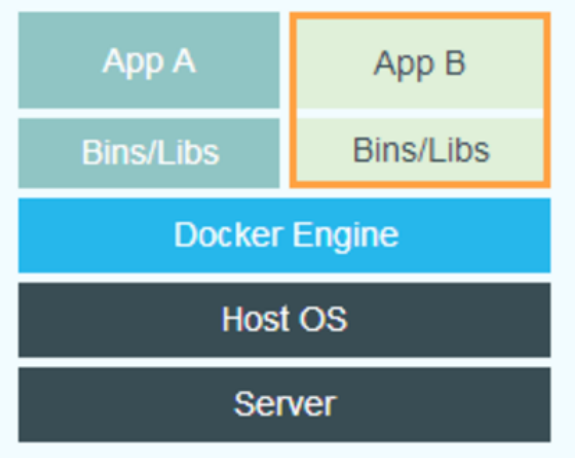

图 3-12　Docker 容器模型

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[29]。

- 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[30]。

- 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。

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[31]。

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口，将访问该端口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[32]。通过这种方式，增强了应用的安全性，具体流程如图3-13所示。

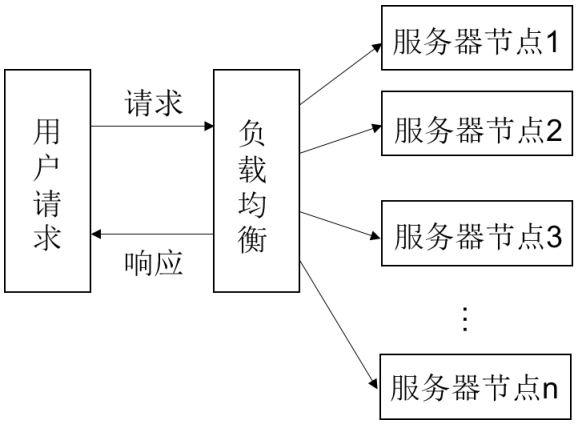

图 3-13　负载均衡流程

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。

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[33]。

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[33]。

4.1　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。[34]。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[35]。

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。

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，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[36]。

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 引擎优化参数如表 4-2所示：

表 4-2　InnoDB 优化参数

参数项	参数内容	参数描述
innodb_buffer_pool_size	800M	缓冲池字节大小
innodb_buffer_pool_instances	8	缓冲池实例数量
innodb_buffer_pool_load_at_startup	1	启动时将热数据加载到内存
innodb_buffer_pool_dump_at_shutdown	1	关闭时将热数据 dump 到本地磁盘
innodb_lru_scan_depth	2000	page cleaner 线程每次刷新脏页的数量
innodb_lock_wait_timeout	5	事务等待获取资源等待的最长时间
innodb_io_capacity	4000	调整刷新脏页的数量
innodb_io_capacity_max	8000	刷新脏页的最大值
innodb_flush_method	O_DIRECT	数据和日志写入磁盘的方式-直接写入磁盘
innodb_file_format	Barracuda	文件格式，Barracuda 支持压缩页，新格式
innodb_file_format_max	Barracuda	设置文件格式最高版本
innodb_flush_neighbors	1	刷新脏页临近页
innodb_log_buffer_size	1M	用来缓冲日志数据的缓冲区大小
innodb_purge_threads	4	单独的清除线程数量
innodb_large_prefix	1	为字段创建索引时限制的字节长度，超过直接报错
innodb_thread_concurrency	64	线程并发数
innodb_print_all_deadlocks	1	将发生的所有死锁信息都记录到错误日志中
innodb_strict_mode	1	严格检查模式，写法有错误直接报错，不警告

续下页

续表 4-2　InnoDB 优化参数

参数项	参数内容	参数描述
innodb_sort_buffer_size	10485760	建立索引时用于排序数据的排序缓冲区大小-10M
innodb_buffer_pool_dump_pct	40	转储缓冲池中的最近使用的页的占比
innodb_page_cleaners	4	page cleaner 线程数量
innodb_undo_log_truncate	1	开启在线回收 undo log 日志文件
innodb_max_undo_log_size	2G	超过这个阈值时触发回收
innodb_purge_rseg_truncate_frequency	128	回收 undo 日志的频率

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[37]。

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。

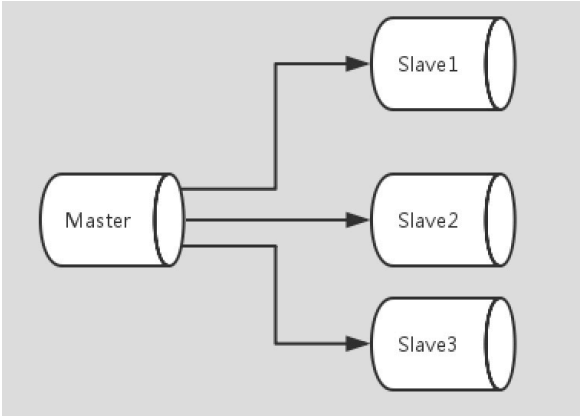

图 4-1　MySQL 复制示意图

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。

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[38]。

图 4-2描述了复制的过程

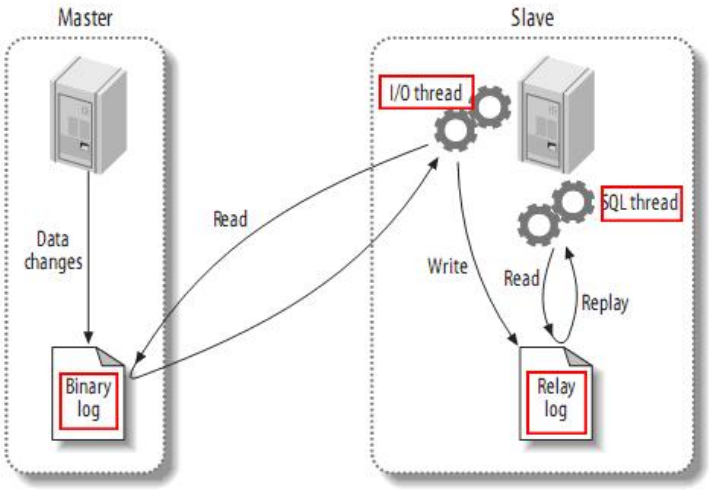

图 4-2　MySQL 复制流程

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 '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;
```

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：

```
#!/usr/bin/python2.7
```

```
import oss2
import sys
auth = oss2.Auth('AccessKeyId', 'AccessKeySecret ')
endpoint='http://oss-cn-beijing-internal.aliyuncs.com'
bucket = oss2.Bucket(auth, endpoint, 'mysqlbk')
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 Shell 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。

数据库备份脚本的设计流程为：

(a) 为了保障跟数据库服务器以及数据库服务的连接，首先需要配置服务器的基本信息和数据库相关信息，主要包括数据库服务器的 IP 地址以及数据库的服务端口号、数据库名称以及用于认证的用户名和密码等信息。

(b) 为了记录每次数据库备份的信息以及备份结果，需要在指定目录生成数据库备份日志文件，并且设计日志记录的内容和格式，确保每次备份都能准确的记录相关的信息，包括备份的时间、备份的结果、备份文件的路径及名称等信息。

(c) 为了保证数据库备份文件的有效空间占用和统一管理，配置备份文件的保存路径以及文件命名格式。

(d) 为了实现对多个数据表的分别备份，设计变量 dbname 数组，存放需要备份的数据表名称。

(e) 通过循环对需要备份的数据表进行备份，备份方式为：

```
mysqldump -u${dbuser} -p${dbpasswd} -hdbip -
   Pdbport ${dbn} --set-gtid-purged=OFF > ${
   logpath}/${backtime}.sql
```

通过 mysqldump 命令实现数据库的备份

(f) 为了节约磁盘空间，对备份完成后的数据库进行压缩，将 sql 文件压缩为 bz2 格式的文件，压缩比为 1:12。

(g) 为了保证备份文件长久存在，通过 OSS 文件上传脚本将压缩后的备份文件上传到阿里云的 OSS。

(h) 为了合理的配置本地的数据恢复以及磁盘空间，对七天以上的备份进行删除，只保留近其他的备份数据。

备份脚本参见附录 B：

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 分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[39]。

5.1 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[40]。

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

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	WEB 应用访问域名
	img	图片服务器域名
	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：

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[41]。

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[42]。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. 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 主配置文件配置项如表 5-7所示：

表 5-7　心跳监听主配置文件参数

参数项	参数内容	参数描述
logfile	/var/log/ha-log	日志路径口
keepalive	2	发送心跳报文的间隔，默认单位为秒
deadtime	30	认为对方宕掉的间隔时间
warntime	10	认为对方可能宕掉的间隔时间
initdead	120	等待对方启动的最大时间
udpport	694	表示 heartbeat 广播/单播通讯使用的 udp 端口
bcast	eth0	心跳所使用的网络接口
ucast	eth0 10.172.89.141	单播通讯，对方网络接口及 IP 地址
auto_failback	on	表示当主节点正常之后是否将资源/服务切换回来
node	app1,app2	就是 heartbeat 集群中的节点信息

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[43]。

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。

2. 配置监控脚本，实现服务的监控。

目前系统服务的监控主要包括系统服务状态的监控和系统数据库状态的监控，通过心跳监听可以保证上述监控的高可用性，首先开发系统的监控脚本 system_monitor 脚本，脚本的具体内容参见附录 C, 通过脚本的内容可以看出在保证监控程序高可用的情况下，在监控频率上也进行可响应的配置，将监控的频率设置为 30s，表示监控程序每 30 秒会对所有的服务状态和数据库状态进行一次轮询检查，出现异常进行响应的处理，大大保证了监控的有效性。监控脚本的主要流程和策略可以参见图 5-2, 服务的监控主要分为 Tomcat 服务和 MySQL 服务的运行状态，数据库状态的监控主要是对数据库的主从同步状态进行监控，确保数据的完整性。

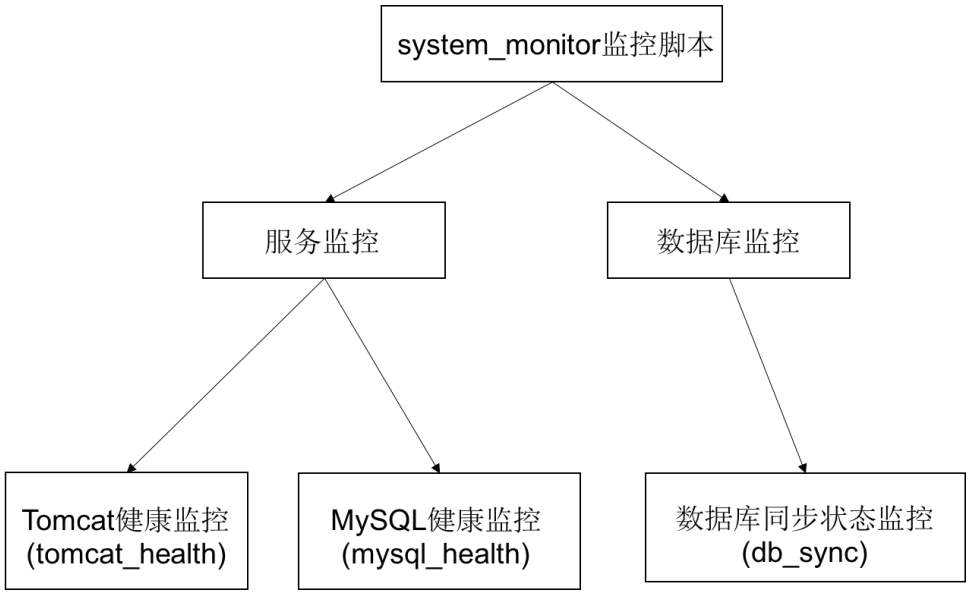

图 5-2　系统监控方案

监控脚本设计和开发完成后，通过在 app1 和 app2 两台服务器上同时部署监控脚本，这样在 app1 服务器出现故障的情况下，app2 会获得 app1 故障的消息，自身启动监控脚本确保监控的正常运行，由于在上一部分我们在配置心跳监听时，将 auto_failback 参数配置为 on，这表示当 app1 服务器状态恢复正常后，app2 会主动停止自身的监控程序，将监控程序移交给 app1 服务器，这样既保证了监控的正常运行也实现了监控的高可用。

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 监控方案设计

Tomcat 是应用运行的容器，要保证 WEB 应用的正常运行必须要先保证 Tomcat 服务的正常运行，正常情况下我们在小型项目的开发过程中可以通过访问服务器，执行 CentOS 系统自带的服务状态命令查看 Tomcat 服务的状态：

```
systemctl status tomcat
```

执行上述命令后，终端中会显示 tomcat 服务的运行状态情况，如图 5-3所示：

```
$systemctl status tomcat
● tomcat.service - Apache Tomcat Web Application Container
   Loaded: loaded (/usr/lib/systemd/system/tomcat.service; enabled; vendor preset: disabled)
   Active: active (running) since 日 2017-03-05 02:00:04 CST; 18h ago
  Process: 9873 ExecStop=/opt/tomcat_minishop/bin/shutdown.sh (code=exited, status=0/SUCCESS)
  Process: 9915 ExecStart=/opt/tomcat_minishop/bin/startup.sh (code=exited, status=0/SUCCESS)
 Main PID: 9923 (java)
   CGroup: /system.slice/tomcat.service
           └─9923 /usr/java/jdk1.7.0_80/bin/java -server -showversion -Djdk.tls.ephemeralDHKeySiz...

3月 05 02:00:04 miniapp1 systemd[1]: Starting Apache Tomcat Web Application Container...
3月 05 02:00:04 miniapp1 startup.sh[9915]: Tomcat started.
3月 05 02:00:04 miniapp1 systemd[1]: Started Apache Tomcat Web Application Container.
```

图 5-3　tomcat 服务状态

active(running) 表示服务正常运行，除此之外还有其他的几种状态如表 5-8所示：

表 5-8　Tocmat 状态

状态	描述
active	运行正常
failed	启动失败
unknown	未知错误

为了实现对于 Tocmat 的远程监控以及脚本监控，我们需要根据以上的几种运行状态去设计监控过程中对于 tomcat 运行状态的判断和故障通知，除此之外，还需要使用可以通过 SSH 协议执行远程命令的 Python 库来实现在第三方服务器上开发 Tomcat 状态监控和通知的脚本。

针对以上的两个问题，本论文选择 Python 的 ssh 库实现对于远程服务的 ssh 访问，通过 ssh 库中 SSHClient() 对象的 connect 方法可以建立本地对于远程服务器的连接，建立连接之后我们就可以执行远程命令了，这一部分同样通过 ssh 库的 exec_command 函数来实现，通过该函数在被监控服务器端执行 systemctl is-active tomcat.service 命令来监测 Tomcat 服务的健康状态，根据不同的服务运行状态设计 Python 脚本返回的服务运行状态码，如表 5-9所示：

在 Python 监控脚本中，如果出现可以自动解决的问题比如服务停止的异常，可以通过执行远程命令 systemctl restart tomcat.service 命令来重新启动 Tomcat 服务，最终 Python 监控脚本将状态码返回到 SHELL 监控脚本中。

以上提到的 Python 监控脚本仅仅是对于 Tomcat 状态监控的脚本，在实际应用中除了状态监控以外我们还需要考虑到日志存储和短信通知的问题，这些问题我们通过设计一个 shell 监控脚本来实现日志记录和短信通知的功能。首先我们通过调用 Python 监控脚本，根据脚本返回的状态码来进行判断，然后根据判断结果

表 5-9　自定义返回状态码

状态码	描述
100	表示 WEB 访问正常
101	表示 WEB 应用异常，需要查看日志
102	Tomcat 服务停止，需要重启
103	服务不存在，需要查看日志
104	服务器连接失败，需要查看是网络问题还是服务器异常
105	服务重启
106	其他异常
200	重启成功
201	重启失败

进行日志的记录和短信通知。

综合上述介绍的两部分，Tomcat 监控的框架可以描述为图 5-4所示：

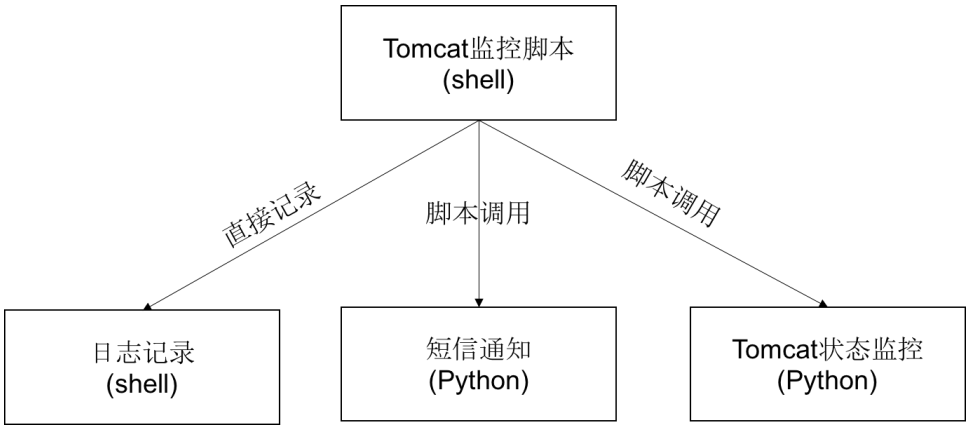

图 5-4　Tomcat 监控框架

具体的监控和故障处理脚本可以通过附录 D和附录 E来查看。

5.2.3　数据库监控方案设计

除了 WEB 应用容器 Tomcat 容器的监控，对数据库的监控对于系统的稳定性来说也是非常重要的，为了最大程度的保证数据库的稳定性和数据的完整性，本论文对于数据库部分设计开发了相对应的监控方案，为了保证数据库服务的正常运行设计开发数据库健康状态监控方案，针对于数据的完整性和高可用性，对于

数据库的主从数据看复制设计开发了主从复制状态监控。

1. 数据库健康状态监控方案

   对于数据库的健康状态监控，可以通过跟 Tomcat 监控方案类似的方法进行监控，但是通过执行 SSH 命令对于系统的安全性来说有一定程度的威胁，而且考虑到 MySQL 服务本身可以实现远程的连接与访问，因此本部分没有使用基于 SSH 协议的远程命令执行方案对数据库健康状态进行监控，本部分使用的是执行本地 mysql 命令访问远程的数据库，通过执行 sql 语句的命令来实现数据库健康状态监控。这样做的优势在于"systemctl status " 命令只能判断 MySQL 服务是否正常运行，无法检查 MySQL 内部异常导致服务异常但是运行正常的现象。

   对于 MySQL 而言，其自身的"show status" 命令可以获取到 MySQL 运行时的相关参数，包括进程状态、连接状态、存储引擎状态等相关信息，通过对有效参数进行判断可以准确的获取 MySQL 的健康状态，如果有正常的数据返回则说明数据库运行正常，否则数据库的健康状态出现问题。当出现问题时，首先需要通知运维人员，为了实现通知功能本论文开发了短信通知的功能，这部分可以参考下一节，其次需要对当前的问题进行分析和判断，如果主从两台服务器的数据库的健康状态均出现问题，那么只能通过通知运维人员人工介入的方式来解决问题，如果只有一台服务器的数据库健康状态出现问题，那么需要通过调整数据库负载均衡，将出现问题的数据库的权重设置为 0，正常的数据库权重设置为 100，保证数据库服务的正常运行，然后尝试重启异常的数据库服务进行故障恢复。

   为了实现数据库负载均衡的动态调整，需要开发针对于阿里云负载均衡的权重调整脚本，通过分析和使用阿里云的负载均衡 SDK 来开发权重调整脚本，通过 SDK 的 DescribeLoadBalancerAttributeRequest 包来获取当前数据库负载均衡的状态，通过 SetBackendServersRequest 开对负载均衡的权重进行调整，最终返回权重调整的结果。

   为了实现数据库服务的远程重启，需要开发针对于 Docker API 的容器启动脚本，通过 docker SDK 的 Client 方发同远程服务器进行连接，通过 inspect_container 方法获取远程容器的运行状态，如果 MySQL 容器的状态异常则通过 restart 方法进行重启。

   综上所述，数据健康健康脚本的执行流程和顺序如图 5-5所示。

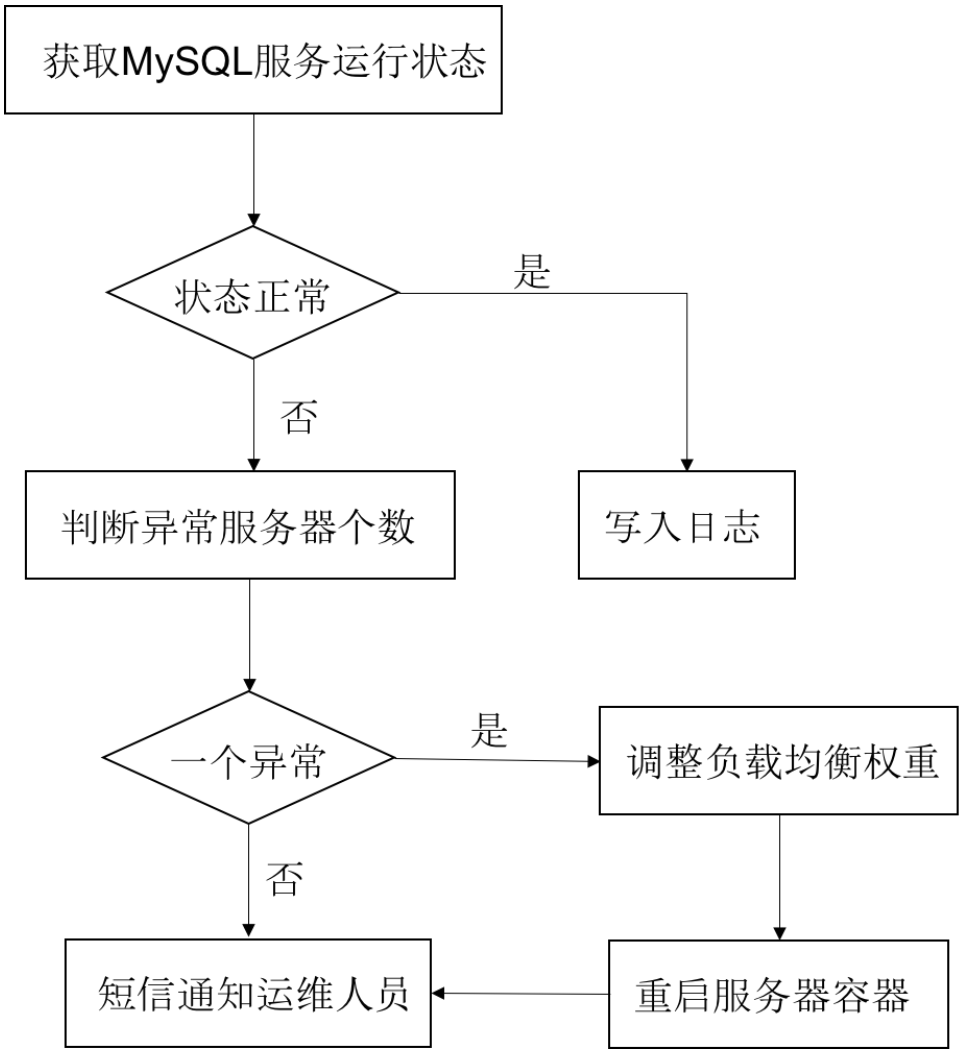

图 5-5　数据库健康状态监控方案

完整的数据库健康监控脚本可以参考附录 F。

2. 数据库主从复制状态监控方案

针对于数据库的主从数据库复制的问题性和数据一致性，需要设计数据库主从复制状态监控方案，方案的目的就是通过执行主从复制的 sql 语句 show slave status 来获取数据库主从复制的状态，根据返回的参数值判断数据主从复制是否正常，如果正常则记录正常日志，否则需要将错误信息记录到日志文件中并短信通知运维人员。

对于执行 show slave status 语句后返回的数据库主从复制状态参数，主要通过以下三个方面判断数据库状态是否正常：

- 通过 Slave_IO_Running 和 Slave_SQL_Running 两个参数的值来判断从库的复制状态是否正常，第一个参数负责与主机 IO 通信，第二个参数负责自己的 slave mysql 进程，如果两个参数的值均为 YES 则说明从库

复制状态正常，否则说明复制状态不正常，需要检查从库的日志来进行问题的定位。这是进行数据库复制状态检查的第一步，只有以上两个参数值均为 YES 是才会进行下面的判断，否则直接将错误信息写入日志并短信通知运维人员。

- 通过 Seconds_Behind_Master 参数的值来判断主从复制的延迟，如果延迟时间大于 300 毫秒则说明主从复制状态异常，需要记录错误信息并短信通知运维人员，否则说明复制状态正常，需要进行下一步的判断。
- 通过 Master_Log_File,Relay_Master_Log_File,Read_master_log_Pos 和 Exec_Master_log_pos 四个参数的值来判断主从复制的最终状态，其中 Master_Log_File 表示主库的数据库当前写入的日志文件，Relay_Master_Log_File 表示从库读取到的主库日志文件，如果两个参数的一致则表示日志文件正常；Read_master_log_Pos 表示当前主库操作的日志位置，Exec_Master_log_pos 表示当前从库执行的主库日志位置，如果这两个参数一致则表示从库执行的最新操作是主库的最后操作，不存在延迟情况，只有当以上两种情况同时正常时，才会最终确认数据库主从复制正常，否则会将错误信息记录到日志文件中，并通知运维人员人工干预修复异常。

数据库主从复制状态监控的流程如图 5-6所示。

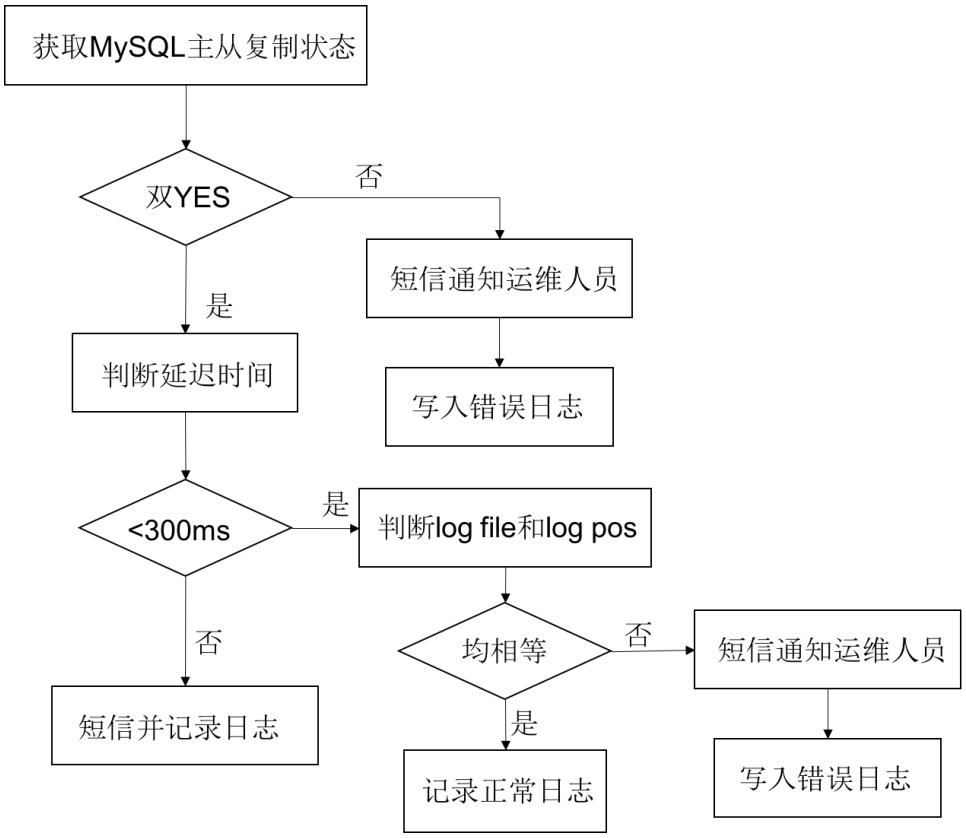

图 5-6　数据库主从复制状态监控方案

数据库同步状态监控的脚本可以参考 G。

5.3　短信通知方案设计

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[44]。

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, 短信通知的脚本参考附录 I。

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 日志备份方案设计

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参考附录 J。

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 H, 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

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。

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。

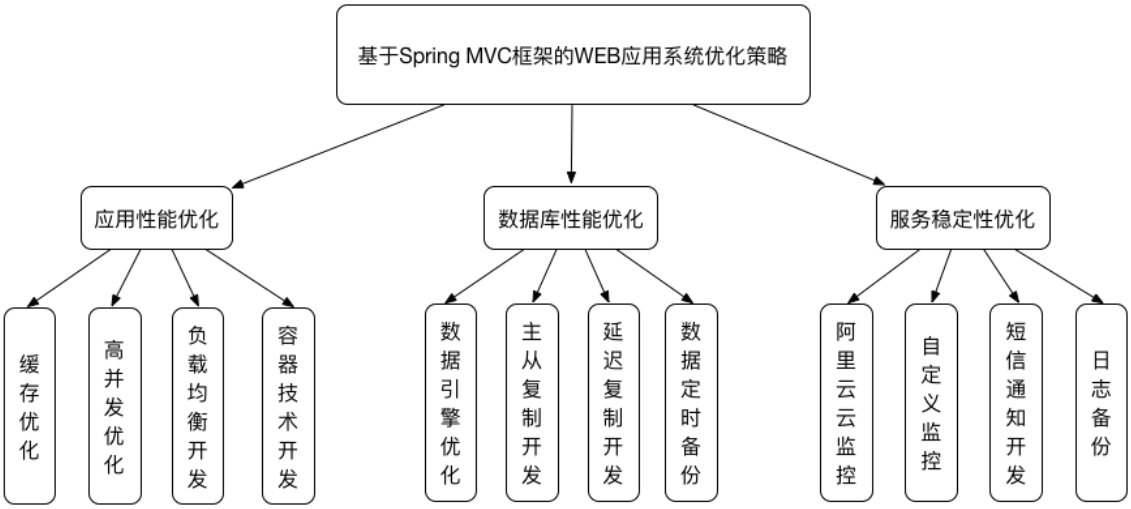

图 6-1 系统优化研究总结

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[45]。
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[46]。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。

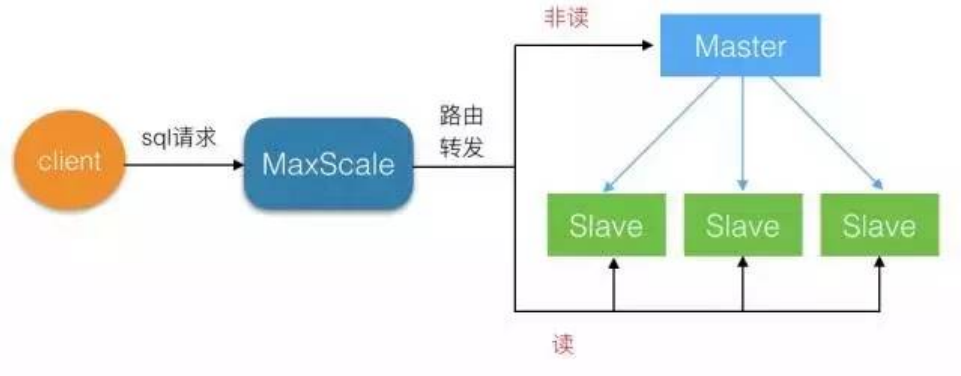

图 6-2　数据库读写分离流程

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] 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. 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] Brown M C. Developing with Couchbase Server. O'Reilly, 2013.

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

[24] 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.

[25] HuaYing. Tomcat 性能调优 [EB/OL]. [2012-10-06]. http://blog.sina.com.cn/s/blog_417b97470100glmi.html.

[26] Vukotic A, Goodwill J. Embedding tomcat. Apache Tomcat 7. Springer, 2011: 175–184.

[27] Fink J. Docker: a software as a service, operating system-level virtualization framework. Code4Lib Journal, 2014, 25.

[28] 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.

[29] Chamberlain R, Schommer J. Using docker to support reproducible research. DOI: http://dx. doi. org/10.6084/m9. figshare, 2014, 1101910.

[30] 高飞, 李勇. Docker 容器资源管控配置实战. 程序员, 2014, (9):96–101.

[31] 互联网文档资源. 负载均衡 (计算机)[EB/OL]. [2015-07-22]. http://www.360doc.com/content/14/0912/14/15077656_408909322.shtml.

[32] 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.

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

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

[35] 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.

[36] 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.

[37] 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.

[38] 秦金. 分布式 mysql 副本控制的研究与实现 [D]. 北京邮电大学, 2013.

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

[40] 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.

[41] 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.

[42] 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.

[43] 百度文库. heartbeat 配置说明 [EB/OL]. [2012-09-23]. http://wenku.baidu.com/view/101c9ac3d5bbfd0a795673df.html.

[44] 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.

[45] Reese W. Nginx: the high-performance web server and reverse proxy. Linux Journal, 2008, 2008(173):2.

[46] 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] [project name] [
       deploypath]"
    exit 1
}
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　数据库备份脚本

```
#!/bin/bash
#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
```

附录 C　心跳监听系统监控脚本

```
#!/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
```

附录 D　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 E　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 F　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 G　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 H　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
```

附录 I　短信通知脚本

```
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()
```

附录 J　归档存储脚本

```
#!/usr/bin/python2.7
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()
```

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.