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](#bookmark128)]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](#bookmark129)]不再满足于通过缓慢的加载和复杂的操作获取资讯[[4](#bookmark130)]，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](#bookmark131)]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](#bookmark132)]。

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 SpringMVC开发框架

SpringMVC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[。](#bookmark128)

2.1.1 SpringMVC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](#bookmark21)，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。

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](#bookmark133)]。

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](#bookmark134)]。

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](#bookmark135)]。

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](#bookmark136)]。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](#bookmark137)]。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](#bookmark138)]。MySQL的特点主要有[[13](#bookmark139)]:

•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](#bookmark140)]。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](#bookmark141)]:

•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](#bookmark142)]。

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](#bookmark143)]。

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](#bookmark144)]。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](#bookmark145)]。

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。

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，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：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: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 suc-ceeds or is unstable" 出打勾，保证在只有构建成功后才执⾏构建完成后的操作，其次需要配置构建完成对⽂件的操作：

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](#bookmark41) 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：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](#bookmark173)。
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](#bookmark48)。

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](#bookmark49)。

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](#bookmark148)]。

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](#bookmark51)所示：

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，以最小化数据迀移量进行复制。

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](#bookmark52)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)在设置标签页中的集群标签页下新建和配置集群信息，在服务器节点标签页下新增服务器节点，增加完服务器节点后可以看到集群的基本信息。

* + 1. 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](#bookmark54)所示：

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](#bookmark149)]。

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](#bookmark57)所示日志，表示使用的是BIO模式：

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](#bookmark58)所示日志，表示使用的是NIO模式：

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](#bookmark59)所示日志，表示使用的是APR模式：

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](#bookmark151)]。

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](#bookmark152)]，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：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：

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:

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：

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](#bookmark66)所示。根据统计结果可以发现，在低并发的情况下，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](#bookmark153)]。

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](#bookmark69)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](#bookmark70)所示[2[8](#bookmark153)]。

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](#bookmark155)]。

•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。

•更轻松的迀移

由于Docker确保了执行环境的一致性，使得应用的迀移更加容易。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，迀移到另一个平台上，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](#bookmark156)]。

•对比传统虚拟机总结

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): —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：

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](#bookmark157)]。

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](#bookmark158)]。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。

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。

1. 增强数据存储的灵活性

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

功能。

随着非结构化数据存储需求的持续增长，各种非结构化数据存储的数据库应运而生（如MongoDB）。从最新的数据库使用排行榜来看，MongoDB已经超过了PostgreSQL,其火热程度可见一斑。各大关系型数据库也不甘示弱，纷纷提供对JSON的支持，以应对非结构化数据库的挑战。MySQL数据库从5.7.8版本开始，也提供了对JSON的支持。使用方式如下：

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，MySQL5.7的JSON支持与存储引擎没有关系，MyISAM存储引擎也支持JSON格式。

MySQL支持JSON以后，总是避免不了拿来与MongoDB进行一些比较。但是，MySQL对JSON的支持，至少有两点能够完胜MongoDB:

1. 可以混合存储结构化数据和非结构化数据，同时拥有关系型数据库和非关系型数据库的优点
2. 能够提供完整的事务支持generated column 是 MySQL 5.7引入的新特性，所谓generated column,就是数据库中这一列由其他列计算而得。
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 —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：

1. 提升了数据库的可用性

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

在线开启GTID,在之前的版本中，由于不支持在线开启GTID,用户如果希望将低版本的数据库升级到支持GTID的数据库版本，需要先关闭数据库，再以GTID模式启动，所以导致升级起来特别麻烦。MySQL 5.7以后，这个问题不复存在。在线修改buffer pool的大小，MySQL 5.7为了支持online buffer pool 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inn- odb\_buffer\_pool\_size的配置有了一定的影响。innodb要求buffer pool size是innodb\_buffer\_pool\_chunk\_size\* innodb\_buffer\_pool\_instances 的倍数，如果不是，将会适当调大innodb\_buffer\_pool\_size,以满足要求，因此，可能会出现buffer pool的实际分配比配置文件中指定的size要大的情况.

OnlineDDLMySQL 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。

1. 性能相关的改进

•只读事务性能改进

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

MySQL5.6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MySQL5.7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垢病的问题之一，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，MySQL5.7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](#bookmark159)]。

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](#bookmark80)所示。

这些数据引擎中使用最广泛的是MylSAM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](#bookmark159)]。

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](#bookmark160)]。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](#bookmark161)]。

1. 内存利用方面

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

除此之外，还可以通过调整innodb\_buffer\_pool\_instances来修改数据库缓冲池的实例数量，通过开启多个内存缓冲池，把需要缓冲的数据hash到不同的缓冲池中，这样可以并行的内存读写，在高IO负载时保持非常稳定的吞吐。一般来说pool size参数和pool instance参数之前相互配置，通过不断的测试来调优二者的值，最终使数据库的内存利用方面达到最高。

1. IO控制方面

对于数据库的空间占用，可以通过修改innodb\_file\_\*参数来配置，考虑到阿里云的磁盘扩展和读取的性能，这个参数的配置对于数据库的的性能提升效果不明显，因此没有必要做配置。

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

1. 日志控制方面

通过修改innodb\_log\_file\_size参数可以调整每个日志文件的大小，每个日志文件的大小对于服务器磁盘的写入和异常恢复后的恢复效率有非常大的影响，因此需要合理配置该参数值，由于当前的数据库服务器是阿里云数据库，硬盘为高速硬盘，数据的存取效率和IO压力均表现良好，故这个参数对于当前项目来说没有优化的必要。通过修改innodb\_log\_buffer\_size参数可以日志缓冲区的大小，这个值分配的大小与内存中缓存的日志大小很大的关系，适当修改这个值可以降低内存的压力。

除了正常的日志外，InnoDB还有undolog，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](#bookmark162)]。

1. 其它方面优化

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

综合以上各个方面的优化之后，本论文中WEB应用的数据库InnoDB引擎优化参数如表[4-2](#bookmark83)所示：

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](#bookmark163)]。

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](#bookmark86)所示，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](#bookmark164)]。

图4-2描述了复制的过程

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masterl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。

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，masterl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

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

其中repl为⽤户名，replpassword为密码

1. 配置master信息

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。

1. 启动复制，并查看状态

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。

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：

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AccessKey-Secret。配置完成之后即可完成访问OSS的认证⼯作。

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

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

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：

1. 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。
2. 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。
3. 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。
4. 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。
5. 通过循环对需要备份的数据表进行备份，备份方式为：通过mysqldump命令实现数据库的备份，备份⽅式为：
6. 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。
7. 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。
8. 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](#bookmark175):

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：

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：

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

1.站点监控

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

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

监控的状态如图5-1所⽰：

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：

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。

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:

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主配置⽂件配置如下：

在配置过程中将app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:

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:

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。

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：

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：

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5-7所⽰：

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-activetomcat.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5-8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：

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"systemctlstatus"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in-spect\_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。

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showslave 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，Re-lay\_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。

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,短信发送脚本如下所⽰：

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：

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。

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](#bookmark171)]。
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](#bookmark172)]。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。

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

1. 除了通过读写分离的方式提升数据库的效率以外，在数据库稳定性方面可以通过LiquiBase软件来实现数据库重构和迀移的操作，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迀移，避免了人工干预导致的其它问题。
2. 本论文虽然用了阿里云的云监控实现了服务状态的监控，但是服务的自动failover工作却没有实现，目前还是通过人工干预来进行故障恢复。在以后的项目开发中，可以充分开发阿里云API的故障检测和自动failover系统，通过云监控的API去监控服务的状态，当出现故障时，自动调用failover程序通过故障服务的API接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。