1 绪论  
1.1 论文研究背景和意义

当今社会是一个“信息爆炸”的社会，尤其是进入互联网时代后，人们每天接触的信息和互联网产品远远大于互联网发展初期的年代。随着中国互联网技术的不断发展以及 WEB 应用开发成本的不断降低，人们对于 WEB 应用的需求不再满足于通过缓慢的加载和复杂的操作获取资讯，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 应用程序被广大开发者所使用，目前主流的 WEB 应用程序 均在使用此框架。使用 MVC 框架，可以将 WEB 应用进行清晰的分层开发，前端 开发人员主要负责页面的呈现方式和用户体验，后台人员则主要负责应用的逻辑 实现以及数据结构的搭建，极大的降低了研发成本。

随着信息化的进一步普及，数据库的使用越来越广泛，数据成为一个应用甚 至也个企业最重要的价值体现，越来越多的企业发展离不开数据库。因此数据库 的稳定性和安全性也称为很多企业研发的重点。通过设计数据的管理和使用机制， 在提高数据库使用效率的同时，保障数据的完整性和安全性是企业运维人员的在 运维过程中的重要任务。

除此之外，如何实现WEB应用的高可用性和分布式服务也是保障用户体验的 一个很重要的部分，通过设计基于应用和服务器的不同层级不同纬度的优化策略， 提升应用的高可用性也是在 WEB 应用开发中必须要注意的。

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

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 论文主要的研究内容

1.3.1 论文的主要研究内容

论文在满足应用正常运行的基础上，通过不同维度的优化实践研究基于 Spring MVC 架构 WEB 应用的系统优化策略。首先通过搭建基于 Spring MVC 的 WEB 测 试应用，在实际用户使用的过程中通过不断开发和调整系统的优化策略，提升应 用的用户体验和应用的稳定性，探索行之有效的系统优化策略。

本文的研究对象主要有以下内容:

(1) 对 Jenkins 持续集成环境研究。自动化部署和代码检测是保证基于 WEB 的 应用产品质量的一个重要环节，通过研究部署 Jenkins 集成测试环境，对编写的代 码进行版本控制、自动化构建和代码测试，研究持续集成方案对于 WEB 应用系统 优化策略的影响。

(2) 对 Couchbase 缓存机制进行研究。目前大多数的 WEB 应用对于缓存性能 的优化还没有足够的重视，通过开发针对 WEB 应用的 Couchbase 缓存系统，研究 Couchbase 的缓存机制测试有效的缓存机制对于系统性能提升和用户体验的影响， 探索可用的系统优化策略。

(3) 对 Docker 容器编排技术进行研究。随着用户的不断增加，单一节点的 WEB 应用或数据库应用已经无法满足用户的需求，如何快速的部署新的应用节点提升 应用性能成为开发者关注的问题。通过构建基于Docker容器编排技术的应用容器， 在新的服务器快速部署新的应用，并加入到应用集群中去，探索 Docker 机制对于 系统优化策略的影响。

(4) 对数据库的主从复制进行研究。随着应用的发展，数据逐渐成为应用最有 价值的部分，如何更好的保证数据的完整性以及安全性在应用的维护过程中日益 重要。通过研究数据库的主从复制和延时复制方案，探索数据库优化对于系统安 全性优化的策略。

(5) 对基于 API 的监控和应急措施方案进行研究。随着服务节点的不断增加， 服务的高可用和服务的健康性检查时 WEB 应用的运维人员在维护过程中必须要 注意的方面。通过开发基于 API 的服务器、服务监控系统以及基于 API 的应急处 理系统研究 API 操作对于系统的快速检测和快速恢复的影响。   
(6) 其他方面，研究 WEB 应用的搭建过程、Tomcat 的相关配置、负载均衡以 及阿里云的相关配置研究 WEB 应用和服务器自身优化对于系统性能的影响。

1.3.2 论文目标

本论文致力于分析目前很多 WEB 应用开发过程中存在的问题和不完善的地 方，并探索 WEB 应用性能和服务器性能的优化方案。

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

系统性能优化策略研究主要包含应用层面、数据层面和服务器层面三个层面。 在应用层面，通过持续集成、代码审核实现应用的稳定性和安全性。在数据库层 面，通过多数据库主从复制、负载均衡和备份恢复等策略实现数据的高可用和稳 定性。在服务器层面，通过对服务的监控和服务器的监控、服务器间负载均衡的 配置、基于 API 的自动化 failover 等策略保障服务的正常运行和高负载应对。

1.4 论文组织结构

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

第一章:绪论。本章主要介绍 WEB 应用开发的现状和系统优化策略研究的意义，明确了在 WEB 应用开发过程中对于应用和服务器进行优化的必要性和重要性。介绍了本论文的主要工作内容、目标以及创新点。最后介绍了论文的编排结构。

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

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

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

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

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

2 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 是一种基于 Web 开发的系统架构，目前主流的系统框架还有 Struts，但是 Spring MVC 架构引起优势依然处于不可替代的位置。在 Spring MVC 架构中，系统的终极目标被解聚、分块，并在不同的模块中表达出来，如此系统 的耦合度得到大幅降低。同时，该框架的整体运行也是在模块驱动的基础之上的， 用户首先发出请求，然后系统予以相应，即常规的请求-相应模型。

2.1.1 Spring MVC 架构的特点

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

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

2.1.2 Spring MVC 框架处理流程

Spring MVC 架构是一个基因用户请求的系统架构，其过程也要求有 Web 驱动 的参加。具体过程现有用户发送请求到前端控制器，这时候前端控制器将请求委 托给处理器;于是处理器就开始处理请求，即调用业务对象进行分析，分析完毕后 将模型数据返回给处理器;然后处理器再将处理结果通过模型和视图返回给前端 控制器;此时前端控制器显示的数据用户还无法处理，还需要进行视图渲染，最 终在前端控制器显示出来，即用户发送的请求产生了响应。整个过程参见图 2-1， 此图反映了用户请求-响应的过程。

2.1.2.1 体系的三层架构

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

1. 显示层。主要有 Spring 和 JSP 经过页面 REQUEST 请求，Spring 的 URL Mapping 对应到响应的控制器，通过对页面发送的请求，Spring 自动的去寻 找响应的控制类，并将结构展示在页面上。

1. 控制层。也就是业务层，是系统中最重要的一层，这层主要是接受URL发送 过来的业务请求，然后去访问 Service 层，把需要的数据装入缓存中，任何通 知 Action 进行跳转。
2. 数据访问层。这层最重要的作用是对数据库中数据以及 XML 文件中的数据 进行访问及操作，并把操作的结果反馈到业务逻辑层，将相应的结果交由下 一层去处理。

这个三层架构的体系的应用很好的解决了系统资源的分配问题，而且将系统的耦 合度降到了可以接受的最低水平。同时，关于异步处理问题上，本系统使用了 FDP 框架，FDP 框架是对 AJAX 的一种轻量级的封装，它的目标是使 AJAX 和 XML 6

Http Request 更简单。通过 FDP 框架，我们只需要在相应的配置对应的 JAVA 方法 以及相对于的实体类，我们很容易的就可以实现许多异步操作，使一些操作不用 刷新页面即可达到相应的效果。可以认为 FDP 框架是对 MVC 三层框架的一种拓 展，其本质还是在 MVC 框架内的。由此可见，笔者在系统体系的选择上更倾向于 架构简单、可移植性好的系统框架模型。

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

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

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。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 服务器采用的是有 Apache 软件基金会 (Apache Software Foundation) 下属的 Jakarta 项目开发的一个Servlet 容器，由于其本身也包含一个 HTTP 服务器，因此也常常被用作一个单独的 Web 服务器。Tomcat7 支持最新 的 Servlet 3.0 规范，而且技术先进、稳定性强一级免费的特点，深受 Java 爱好者 的喜爱并得到了部分软件开发商的认可，成为目前比较流行的 Web 应用服务器。

2.2.4 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。

MySQL 的特点主要有:

* 为多种編程语言提供了API。这些編程语言包括C、C++、C♯、Java、Perl、PHP、Python、Ruby 等。
* 支持多线程，充分利用 CPU 资源，支持多用户。
* 提供 TCP/IP、ODBC 和 JDBC 等多种数据库连接途径。
* 提供用于管理、检查、優化数据库操作的管理工具。
* 可以处理拥有上千万条记录的大型数据库。

2.2.5 Couchbase

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

* Cluster 支持不够。在扩容、负载均衡、高可用等方面存在明显不足。
* 持久化支持不好，出现问题后恢复的代价大。memcached完全不支持持久化， redis 的持久化会造成系统间歇性的负载很高。  
  Couchbase 是一个 NoSQL 数据库，它是世界各国的开发者在 2011 年推出的，由于 它有良好的 cluster 支持、异步持久化的支持得到了众多开发者的青睐，他的特点 主要有:
* Couchbase 的对等网设计，smart client 直接获取整的集群的信息，在客户端 实现负载均衡，整个集群没有单点失效，并且完全支持平行扩展。
* vBucket 的引入，完全实现了 auto sharding，可以方便灵活的把数据的子集在 不同节点上移动，以实现集群动态管理。
* Couchbase 有一个非常专业的 web 管理界面，并且支持通过 RESTful API 管理，这也是 memcached, redis 不能企及的。

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

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 。

Jenkins 是一个用 Java 编写的开源的持续集成工具，它提供了软件开发的持续 集成服务，运行在 Servlet 容器中，它可以执行基于 Apache Ant 和 Apache Maven 的项目，以及任意的 Shell 脚本和 Windows 批处理命令。

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。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。

1. 访问http://mirrors.jenkins.io/war-stable/2.7.1/jenkins.war下载2.7.1版本的jenk-ins 安装包。  
2. 将 jenkins.war 文件放到本地构建服务器(10.18.8.19)的 Tomcat 应用目录 (/opt/tomcat/apache-tomcat-7.0.68/webapps)中。

3. 配置并启动 Tomcat 即可完成安装，端又配置为 8080，启动命令配置为: systemctl start/stop/status tomcat\_jenkins.service

4. 访问 jenkins 地址 (http://10.18.8.19:8080/jenkins/) 初始化 Jenkins:

* 完成认证,按照页面提示，将对应文件内容中的密码输入到文本框中，点 击 continue，如图2-2所示
* 在插件选择处选择安装建议的插件
* 安装完成后在新的页面中配置用户名和密码完成安装，如图2-3
* 进入图2-4页面表示安装成功

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2-5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:

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，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Jenk- ins“系统管理-系统设置”页面中会出现如图2-6配置:

按照不同项目配置 SSH Server 信息和登录验证信息等。

1. 插件配置完成后需要配置自动部署，插件安装完成后在“构建后操作”的选项

中会出现“Send build artifacts over SSh”的选项，点击之后会出现配置:

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。

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。

本论文中的WEB应用的数据包含基础数据和应用数据两部分，其中基础数据 主要包括系统的基本配置数据、系统的常用变量数据、用户权限数据、枚举类型等 相对固定的数据，这些数据在用户访问应用的时候只需要加载一次即可，不需要重复的从数据中读取，而应用数据则主要包含应用内的用户信息、商户信息、交易数据、库存数据等等不固定的数据，这些数据随着用户的时候会不断的发生变 化，而且新的数据从数据库重新读取，这些数据在用户使用时需要多次加载。针对 于以上不同数据的特点，将基础数据在用户首次登陆时加载到 Couchbase 缓存中， 以后再读取时直接从缓存中读取。

3.1.1 Couchbase 安装

1. 直接安装

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

命令将 rpm 安装文件下载到制定路径中。  
然后，通过 root 用户访问 Linux 服务器，在终端中进入到 couchbase 安装文 件所在路径，通过如下命令安装 couchbase:

安装完成后自动启动 couchbase，并且会有如下提示信息:

couchbase 默认安装位置为“/opt/couchbase/” 2. Docker 安装

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

3.1.2 Couchbase 集群配置

Couchbase 服务器及可以单独运行，也可以将多个服务器组成一个集群，作为集群来运行。通过 Couchbase 集群，可以实现缓存数据的分布式存储及负载均衡，

提升缓存的高可用以及系统的性能 。

Couchbase 数据分布是按计算分配到多个节点上，每个节点都储存两部分数据 有效数据和副本数据，客户端对数据的操作主要是按照节点中对应的有效数据进 行操作，执行压力会部分到不同的节点，如 3-3所示:

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)

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

通过测试可以发现，开启 Couchbase 后系统的网络吞吐率在 242.34reqs/s, 而不使用 Couchbase 时吞吐量在 202.32reqs/s，开启 Couchbase 对于系统性能的提升大约在20% 左右。

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。

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:

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:

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:

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。

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

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

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

3.2.1开启 APR 模式

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

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。

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，执行./configure && make && make install 命令，动态库默认 安装在/usr/local/apr/lib 目录下。

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 JAVA\_OPTS 中添加 java.library.path 参数，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:

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3-1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。

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。

同传统的虚拟机技术相比，Docker 容积技术具有非常多的优势，这也是本论 文中使用 Docker 技术作为 WEB 应用开发过程中服务器扩展的主要技术。

* 更高效的利用系统资源 由于容器不需要进行硬件虚拟以及运行完整操作系统等额外开销，Docker 对 系统资源的利用率更高。无论是应用执行速度、内存损耗或者文件存储速度， 都要比传统虚拟机技术更高效。因此，相比虚拟机技术，一个相同配置的主 机，往往可以运行更多数量的应用。
* 更快速的启动时间  
  传统的虚拟机技术启动应用服务往往需要数分钟，而 Docker 容器应用，由于 直接运行于宿主内核，无需启动完整的操作系统，因此可以做到秒级、甚至 毫秒级的启动时间。大大的节约了开发、测试、部署的时间。
* 一致的运行环境 开发过程中一个常见的问题是环境一致性问题。由于开发环境、测试环境、 生产环境不一致，导致有些 bug 并未在开发过程中被发现。而 Docker 的镜像 提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会 再出现“这段代码在我机器上没问题啊”这类问题。
* 持续交付和部署 对开发和运维(DevOps)人员来说，最希望的就是一次创建或配置，可以在 任意地方正常运行。  
  使用 Docker 可以通过定制应用镜像来实现持续集成、持续交付、部署。开 发人员可以通过 Dockerfile 来进行镜像构建，并结合持续集成 (Continuous Integration) 系统进行集成测试，而运维人员则可以直接在生产环境中快速部 署该镜像，甚至结合持续部署 (Continuous Delivery/Deployment) 系统进行自动部署。  
  而且使用 Dockerfile 使镜像构建透明化，不仅仅开发团队可以理解应用运行 环境，也方便运维团队理解应用运行所需条件，帮助更好的生产环境中部署 该镜像。
* 更轻松的迁移 由于Docker确保了执行环境的一致性，使得应用的迁移更加容易。Docker可 以在很多平台上运行，无论是物理机、虚拟机、公有云、私有云，甚至是笔 记本，其运行结果是一致的。因此用户可以很轻易的将在一个平台上运行的 应用，迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常 运行的情况。
* 更轻松的维护和扩展  
  Docker 使用的分层存储以及镜像的技术，使得应用重复部分的复用更为容 易，也使得应用的维护更新更加简单，基于基础镜像进一步扩展镜像也变得 非常简单。此外，Docker 团队同各个开源项目团队一起维护了一大批高质量 的官方镜像，既可以直接在生产环境使用，又可以作为基础进一步定制，大 大的降低了应用服务的镜像制作成本。

• 对比传统虚拟机总结 对比参数 容器技术

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(Definingand 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:

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

3.3.2 应用容器化现状

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

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

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

3.4 SLB 负载均衡优化

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

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

使用带有负载均衡的多个服务器组件，取代单一的组件，可以通过冗余提高 可靠性。

负载平衡最重要的一个应用是利用多台服务器提供单一服务，这种方案有时 也称之为服务器农场。通常，负载平衡主要应用于 Web 网站，大型的 Internet Relay Chat 网络，高流量的文件下载网站，NNTP(Network News Transfer Protocol)服务 和 DNS 服务。现在负载平衡器也开始支持数据库服务，称之为数据库负载平衡器。 对于互联网服务，负载平衡器通常是一个软体程序，这个程序侦听一个外部端又， 互联网用户可以通过这个端又来访问服务，而作为负载平衡器的软体会将用户的 请求转发给后台内网服务器，内网服务器将请求的响应返回给负载平衡器，负载 平衡器再将响应发送到用户，这样就向互联网用户隐藏了内网结构，阻止了用户 直接访问后台(内网)服务器，使得服务器更加安全，可以阻止对核心网络栈和运 行在其它端又服务的攻击。当所有后台服务器出现故障时，有些负载平衡器会提 供一些特殊的功能来处理这种情况。例如转发请求到一个备用的负载平衡器、显 示一条关于服务中断的消息等。负载平衡器使得 IT 团队可以显著提高容错能力。 它可以自动提供大量的容量以处理任何应用程序流量的增加或减少 。

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

考虑到本项目中的所有服务器节点均为阿里云云服务器，故选择通过使用阿 里云的负载均衡服务器服务 (SLB) 来进行应用的负载均衡，通过创建一个负载均 衡服务，即可获取到一个公网的 IP，通过配置监听，将 WEB 应用的 http 协议端又 (80) 和 https(443) 协议分别转发到生产环境各应用服务器节点的 Tomcat 对应端又， 通过配置服务器节点的权重来决定负载均衡在转发用户请求时向后端服务器转发 的比重。

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:

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:

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

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

5. 性能相关的改进

* 只读事务性能改进

众所周知，在传统的 OLTP 应用中，读操作远多于写操作，并且，读操 作不会对数据库进行修改，如果是非锁定读，读操作也不需要进行加锁。 因此，对只读事务进行优化，是一个不错的选择。  
MySQL 5.6 中，已经对只读事务进行了许多优化。例如，将 MySQL 内 部实现中的事务链表分为只读事务链表和普通事务链表，这样在创建 ReadView 的时候，需要遍历事务链表长度就会小很多。

在 MySQL 5.7 中，首先假设一个事务是一个只读事务，只有在该事务发 起了修改操作时，才会将其转换为一个普通事务。MySQL 5.7 通过避免 为只读事务分配事务 ID，不为只读事务分配回滚段，减少锁竞争等多种 方式，优化了只读事务的开销，提高了数据库的整体性能。

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

MySQL 的复制延迟是一直被诟病的问题之一，欣喜的是，MySQL 5.7 版本已经支持”真正”的并行复制功能。MySQL 5.7 并行复制的思想简单 易懂，简而言之，就是”一个组提交的事务都是可以并行回放的”，因为 这些事务都已进入到事务的 prepare 阶段，则说明事务之间没有任何冲 突(否则就不可能提交)。经过对比测试，MySQL 5.7 采用新的并行复制 后，仍然会存在一定程度的延迟，只不过相比 5.6 版本减少了 86%，相 比 MariaDB 的并行复制延迟也小不少。复制延迟问题得到极大改善。 除此之外复制性能的改进还包括多源复制，多从线程增强，在线 GTIDs， 和增强的半同步复制等功能，这些功能均为保证数据的完整性可有效性 提高了保障。

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

Oracle 和 SQL Server 等数据库中只有一种存储引擎，所有数据存储管理机制都 一样。而 MySQL 数据库提供了多种存储引擎。用户可以根据不同的需求为数据表 选择不同的存储引擎，用户也可以根据具体的需求编写自定义存储引擎。MySQL 数据库提供的存储引擎如表 4-1所示。

这些数据引擎中使用最广泛的是 MyISAM 和 InnoDB 两种存储引擎。MyISAM 是 MySQL 早期的 ISAM 存储引擎的升级版本，也是 MySQL 默认的存储引擎，而 InnoDB 是由第三方软件公司 Innobase 所开发，其最大的特点是提供事务控制的特性。

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

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

1. 数据库多版本读取。InnoDB在事务支持的同时，为了保证数据的一致性以及 并发时刻的性能，通过对 un-do 信息的聚簇索引实现对数据的多版本读取。
2. 锁定机制的改进。InnoDB 改变了 MyISAM 的锁机制，实现了行锁。虽然

InnoDB 的行锁机制是通过索引来完成的，但是由于数据库中 99% 的 SQL 语 句都是通过索引来检索数据，所以行锁机制为 InnoDB 在承受高并发下的环 境下增强了竞争力。

1. 实现外键。InnoDB 实现了外键引用这一数据库的重要特性，使得在数据库 端控制部分数据的完整性成为可能。

4.1 InnoDB 引擎参数优化

目前来说，InnoDB 是为 Mysql 处理巨大数据量时的最大性能设计。它的 CPU 效率可能是任何其它基于磁盘的关系数据库引擎所不能匹敌的。在数据量大的网 站或是应用中 Innodb 是倍受青睐的。另一方面，在数据库的复制操作中 Innodb 也是能保证 master 和 slave 数据一致有一定的作用。由于 InnoDB 数据引擎在事务安全、支持外键以及数据恢复成本的综合性能表现，本论文中的 WEB 应用数据存 储引擎使用的是 InnoDB 引擎。

为了保证在使用 InnoDB 引擎过程中，使 WEB 应用的数据稳定性和服务器 负载达到最优的使用体验，还需要基于目前的 WEB 开发架构和服务器现状对

MySQL 的配置进行一定的参数调优。优化主要包括内存、IO、日志以及其它方面。

1. 内存利用方面  
   在内存利用方面，innodb\_buffer\_pool\_\* 相关的参数主要负载调控数据库在运 行过程中数据缓存到内存的数据。其中 innodb\_buffer\_pool\_size 是 InnoDB 最重要的一个配置参数，主要缓存 innodb 表的索引，数据，插入数据时的缓冲。 为 Innodb 加速优化首要参数。参数的默认分配只有 8M，如果在配置过程中 不调整这个值的话，会导致数据库的性能体验过差。 除此之外，还可以通过调整 innodb\_buffer\_pool\_instances 来修改数据库缓冲 池的实例数量，通过开启多个内存缓冲池，把需要缓冲的数据 hash 到不同的 缓冲池中，这样可以并行的内存读写，在高 IO 负载时保持非常稳定的吞吐。 一般来说 pool size 参数和 pool instance 参数之前相互配置，通过不断的测试 来调优二者的值，最终使数据库的内存利用方面达到最高。
2. IO 控制方面  
   对于数据库的空间占用，可以通过修改 innodb\_file\_\* 参数来配置，考虑到阿里云的磁盘扩展和读取的性能，这个参数的配置对于数据库的的性能提升效 果不明显，因此没有必要做配置。  
   对于数据的 IO 分配来说，需要进行一定的优化，可以通过 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。
3. 日志控制方面  
   通过修改 innodb\_log\_file\_size 参数可以调整每个日志文件的大小，这个值分 配的大小和数据库的写入速度，事务大小，异常重启后的恢复有很大的关系， 由于当前的数据库服务器是阿里云数据库，硬盘为高速硬盘，数据的存取效 率和 IO 压力均表现良好，故这个参数对于当前项目来说没有优化的必要。 通过修改 innodb\_log\_buffer\_size 参数可以日志缓冲区的大小，这个值分配的 大小与内存中缓存的日志大小很大的关系，适当修改这个值可以降低内存的 压力。  
   除了正常的日志外，InnoDB 还有 undo log，它记录某数据被修改前的值，可以 用来在事务失败时进行回滚，因此通过调整 undo log 的相关参数可以在很大 程度上保证数据的有效性。通过修改 innodb\_undo\_log\_truncate 参数值为 1 来 开启在线回收 undo 日志文件，通过修改 innodb\_max\_undo\_log\_size 参数值来 配置触发回收 undo 日志的阈值，通过 innodb\_purge\_rseg\_truncate\_frequency 参数来配置回收 undo 日志的频率。
4. 其它方面优化 除了以上三个方面的优化之外，还有许多优化项目可以应用于本项目的数据 库优化中。考虑到阿里云磁盘的性能，可以通过 innodb\_flush\_method 参数将 数据和日志文件不经过缓存直接写入到文件中;通过 innodb\_strict\_mode 参 数开启严格模式，对于写法错误的 SQL 语句跳过警告直接提示错误，提升数 据库的稳定性;通过脏页的相关配置参数调整数据库对于脏页的刷新方式和 效率，提升数据的有效性。

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

4.2 主从复制和延迟复制优化

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

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

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。

1. 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。
2. 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。
3. 数据分析:实时数据可以存储在 Master，而数据分析可以从 Slave 读取，这 样不会影响 Master 的性能。

4.2.1 数据库复制流程

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

复制的过程主要分为三步:  
1. master 将改变记录到二进制日志 (binary log) 中(这些记录叫做二进制日志事 件，binary log events);  
2. slave 将 master 的 binary log events 拷贝到它的中继日志 (relay log); 3. slave 重做中继日志中的事件，将改变反映它自己的数据。

图 4-2描述了复制的过程

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

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

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

此外，在 master 中也有一个工作线程:和其它 MySQL 的连接一样，slave 在 master 中打开一个连接也会使得 master 开始一个线程。复制过程有一个很重要的限制——复制在 slave 上是串行化的，也就是说 master 上的并行更新操作不能在slave 上并行操作。

4.2.2 双主复制设计

目前生产环境的数据库有两个专有的服务器提供服务，为了保证数据的有效性，需要这两个服务器一个主服务器提供数据库的操作，另一个服务器作为从服务器，复制主数据库的数据。但是当主服务器出现问题而宕机时，需要快速的将 从数据库切换为主数据库，在问题服务器恢复正常时作为从数据库，从新的主服 务器复制数据。根据这个需求，将目前的两个数据库配置为双主数据库，将两个 主数据库标示为 master1 和 master2。

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

然后在 MySQl 的配置文件中开启二进制日志，并为每一个数据库配置一个唯 一的 server-id，以及必要的配置。

根据如上配置项，将两个数据库的二进制日志的文件名均配置为 db2.bin，master1 的 server-id 为 1，master2 的 server-id 为 2，配置 sync\_binlog 参数为 1 表示每进行 1 次事务提交之后，MySQL 将进行一次 fsync 之类的磁盘同步指令来将 binlog\_cache 中的数据强制写入磁盘, 提高数据的持久性。

除此之外，需要在配置文件中开启 GTID

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又，配置好用户名和密码。master\_auto\_position 让从库 根据 GTID 自动选择适当的事务点进行复制，基本上无需关注和担心主从不 一致的问题。

1. 启动复制，并查看状态。

在查询结果的字段中通过 Slave\_IO\_Running 和 Slave\_SQL\_Running 两个变量 的值为 YES 来判断主从复制是否成功启动。  
在 master1 和 master2 上均按照上述步骤进行操作，完成高可用的双主复制的 部署。

4.2.3 延迟复制设计

为了避免上述双主服务器均出现问题后无法提供数据服务的现象，在测试服 务器中搭建一个延时复制服务器，延时复制的主库配置为 master1，将复制策略配 置为延迟 1 小时复制，然后通过二进制日志进行近小时内数据的恢复，这样能够 最大程度保证数据的完整。

配置的步骤基本类似于双主的配置，但是在配置 master 信息后需要增加一步，

调整复制的延迟时间，单位是 ms，因此一小时延时的变量值为 3600。

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

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

备份脚本如下:

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 份执行数据库备份脚本。

4.4 本章总结

本章是对本论文中 WEB 项目的数据库进行优化的一些策略，主要包括通过 InnoDB 配置参数调整提升数据库自身的运行性能，通过配置主从复制以及延迟复 制提升数据的稳定性和有效性，通过开发数据库定时备份脚本来实现数据库的每 日备份。通过这些策略，在提升 MySQL 运行性能同时也保证了数据的完整性和有 效性，对于 WEB 应用的作用至关重要。

5 服务监控与应急措施优化

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

5.1 阿里云云监控应用

云监控(CloudMonitor)是一项针对阿里云资源和互联网应用进行监控的服务。 云监控服务可用于收集获取阿里云资源的监控指标，探测互联网服务可用性，以及 针对指标设置警报。由于本项目的所有服务器均为阿里云服务器，所以一部分的监 控可以通过配置阿里云的云监控来实施。通过云监控服务能够监控云服务器 ECS、 云数据库 RDS 和负载均衡等各种阿里云服务资源，同时也能够通过 HTTP,ICMP 等通用网络协议监控互联网应用的可用性。

在监控配置之前，需要对目前阿里云资源进行统计, 如表 5-1所示:

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。

• 生产环境应用负载均衡监控规则

• 生产环境数据库负载均衡监控规则

• 测试环境负载均衡监控规则

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。

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。由于 Heartbeat 能够对资源服务进行控制，所以本论文使用 Heartbeat 作为 心跳监听的工具。

在服务器中配置心跳监听的步骤主要有以下步骤:

1. 下载和安装 Heartbeat 软件

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

安装完成后出现下面的信息表示安装成功:

2. 配置 Heartbeat 软件，实现两个服务器的心跳监听

heartbeat 主要的配置文件有 3 个，分别是 authkeys,ha.cf 和 haresources，au- thkeys 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 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:

该文件为 heartbeat 的认证文件，该文件主要是用于集群中两个节点的认证， 采用的算法和密钥 (如果有的话) 在集群中节点上必须相同，目前提供了 3 种算法:crc/md5/sha1。其中 crc 不能够提供认证，它只能够用于校验数据包 是否损坏，而 sha1/md5 需要一个密钥来进行认证，从资源消耗的角度来讲， md5 消耗的比较多，sha1 次之，因此建议一般使用 sha1 算法。 可以看出，示例中使用的是 sha1 算法，如果要换用其他算法只需要修改 auth 指令后面的数字，然后取消相应行的注释即可。另外，该文件的属性必须为 600，否则 heartbeat 启动将失败。

haresources 主配置文件配置如下:

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

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

5.2.2 Tomcat 监控

对于 Tomcat 的监控监控，主要通过 Python 中 ssh 库的 SSHClient() 对象的 connect 方法连接到远程服务器中，然后通过 exec\_command 函数在服务器端执行 systemctl is-active tomcat.service 命令来检测 Tomcat 服务的健康状态，并且将规 定好的状态码返回到监控脚本中，如果检测到异常，则通过 systemctl restart tom-cat.service 命令来重新启动 Tomcat 服务。具体的监控和故障处理脚本可以通过附录 B和附录 C来查看 5.2.3 数据库监控

对于数据库的监控主要包括数据库监控状态监测和数据库主从复制状态监控。

1. 数据库健康状态监控

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

1. 数据库同步状态监控  
   对于数据的同步状态，通过执行 show slave status 命令来获取从库 的状态，然后首先查看 SQL 和 IO 线程是否为 YES 状态，其 次通过延迟来判断数据库复制的延迟是否过大，最后通过判断 Mas- ter\_Log\_File 和 Relay\_Master\_Log\_File 的值是否相等以及判断同步的偏移量 Read\_master\_log\_Pos 和 Exec\_Master\_log\_pos 的值是否相等来判断同步的健 康状态。  
   当数据库同步出现故障时通过发送短信通知运维人员进行处理。 数据库同步状态监控的脚本可以参考 E

5.3 短信通知

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

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

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

在脚本中，首先需要根据自己注册的账户获取账户 ID 和验证 Token，然后获取 自己创建的应用的 ID，完成基础的配置和认证。在执行脚本时，通过 SendTem- plateSMS.py phone tempID content 命令来发送短信，其中 phone 为接受短信的一个 或者一组手机号，tempID 为自己设计的短信模版对于的 ID，contenet 为短信的发 送内容。

5.4 日志备份

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

• 每个月的 1 号会压缩上一个月的日志文件到压缩文件中 • 每个月清理超过三个月的日志  
• 每个月压缩的日志上传到阿里云归档存储中进行备份

针对以上需求，开发的归档存储上传脚本为:

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

开发的日志备份脚本可以参考附录 F, 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。
2. 对于数据库的性能而言，除了自身的引擎提高之外，还可以通过读写分离来 提高数据库的响应和降低数据库的负载。通过使用 MaxScale 插件，实现 数据库请求的读写分离。具体的流程如图 6-2所示。

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

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