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

1.

Couchbase 安装 直接安装[12]

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[20]

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 [21] 。

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)

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

3.1.3 Couchbase 缓存对系统性能影响

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

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[22]。

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[23]。

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 [24] 。

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

1.

开启 APR 模式安装依赖库, 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。

yum install apr-devel

yum install openssl-devel

yum install gcc

yum install make

2.

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[25? ]。

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[26]。

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[27]。

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 [28] 。

• 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(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 [29] 。

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。