摘 要

关系数据库系统自从上世界被发明以来，就成为了企业存储数据的首选数据库系统。但是在移动互联网时代，大量非结构化数据的存储和处理并不是关系数据库的强项。针对关系数据库系统的缺点，已经出现了各种各样的非关系型数据库，比如，键值数据库用来存储关系最简单的数据；文档数据库用来存储大量图片或者文档数据；图形数据库系统用来存储用户关联数据。每一种数据库系统都有其用武之地，但是企业用各种不同类型的数据库来存储不同类型的数据，导致了服务器资源的浪费，增加了数据库系统的维护成本，所以，设计和实现一种能存储和管理多种数据类型的数据库管理系统具有重要的意义。

当把所有数据都存储在一个数据库系统的时候，必须保证系统能够部署在分布式环境下以扩展其读写能力，还需要保证系统的数据安全。基于以上需求，本文设计和实现了一个分布式数据库系统JSQL，其结合关系型数据库和非关系数据库系统的优点，能存储结构化和非结构化数据类型，支持多主分布式架构，能够动态扩展读写能力，还从底层加入审计功能以保证数据的安全。

在分析系统需求和相关技术以后，论文首先对系统的功能和架构进行阐述，JSQL采用常用的客服端-服务器架构，分为客服端和服务器端，服务器端又分为分布式管理节点和分布式数据库集群。分布式管理节点实现了分布式数据库系统的负载均衡功能和监控管理功能，分布式数据库集群实现数据的存储和管理。其次，详细设计和实现了客户端和分布式管理节点中的负载均衡模块、数据审计模块和分布式数据库集群节点中的数据库功能模块、分布式集群架构模块和审计日志存储功能等模块。为了均衡服务器资源，论文设计和实现了新的动态负载均衡算法，为了提高系统更新性能，论文提出利用分布式队列实现的多主分布式架构。再次，对分布式数据库的结构化存储功能和审计功能等进行了测试，给出测试结果并进行分析。最后，总结本论文所述系统的不足之处和论文后续工作，展望数据库系统的未来发展。

关键词：分布式数据库，关系数据库，非关系型数据库，数据安全，负载均衡

ABSTRACT

With the rapid development of Internet application and computer technology.

Keywords: Distributed database, relational database, security audit, Mysql, Non-relational database

目 录

第一章 绪论 1
1.1 研究背景和研究意义 1
1.2 分布式数据库研究历史与现状 4
1.3 论文主要工作与创新 7
1.4 论文结构安排 8
第二章 相关理论和技术 10
2.1 数据库系统 10
2.1.1 数据库系统概念 10
2.1.2 关系数据库系统 10
2.1.3 事务与并发控制 11
2.1.4 非关系型数据库系统 13
2.2 分布式数据库 15
2.2.1 分布式数据库概述 15
2.2.2 分布式数据库的特点 15
2.2.3 数据分布和负载均衡 17
2.2.4 数据复制和一致性 19
2.2.5 副本和分布式MVCC技术 20
2.3 本章小结 21
第三章 系统分析 22
3.1 系统需求分析 22
3.1.1 系统功能需求 22
3.1.2 系统功能用例 24
3.2 技术和框架分析 27
3.2.1 系统实现语言选择 27
3.2.2 网络实现技术分析 27
3.2.3 通信协议分析 28
3.2.4 SQL实现分析 29
3.2.5 存储引擎分析 31
3.2.6 分布式架构分析 32
3.2.7 监控审计分析 32

3.3 本章小结 33
第四章 系统设计 34
4.1 系统总体设计 34
4.1.1 系统架构设计 34
4.1.2 系统功能设计 37
4.2 客户端模块设计 39
4.3 分布式管理模块 39
4.4 数据库功能模块详细设计 41
4.4.1 数据库功能操作流程设计 41
4.4.2 网络模块设计 42
4.4.3 通信协议设计 44
4.4.4 SQL引擎设计 44
4.4.5 存储引擎设计 46
4.5 集群架构详细设计 46
4.6 审计模块详细设计 48
4.7 本章小结 49
第五章 系统实现 51
5.1 代码规范和总体结构 51
5.2 客户端功能关键技术实现 52
5.3 分布式负载均衡算法实现 53
5.3.1 管理节点初始化实现 53
5.3.2 选举算法实现 55
5.4 多主分布式架构关键技术实现 57
5.4.1 集群初始化流程 59
5.4.2 集群命令执行过程 62
5.5 数据库基本功能实现 62
5.5.1 网络和协议模块 64
5.5.2 SQL解析模块 69
5.5.3 存储引擎模块 71
5.6 数据审计模块的实现 73
5.7 本章小结 74
第六章 系统测试 75
6.1 测试方法 75

6.2 测试环境 75
6.3 功能测试 76
6.3.1 数据库功能测试 76
6.3.2 集群功能测试 77
6.3.3 审计功能测试 78
6.4 性能测试 79
6.5 测试对比 83
6.6 本章小结 84
第七章 总结与展望 85
7.1 本文工作的总结 85
7.2 未来的工作方向 86
7.3 数据库系统的未来 86
致 谢 88
参考文献 89

第一章 绪论

1.1 研究背景和研究意义

近年来，移动互联网经过了快速的发展，安卓和苹果等智能手机的普及让移动互联网用户快速增长。随着无线网络的发展，智能手机的体验也越来越好，对手机移动应用程序的需求也与日俱增。根据我国最近的调查数据显示，我国移动电话用户的数量已经达到13亿，智能手机用户则达到了11亿。而且随着网络技术和移动电话技术的发展，这些数据还会继续增长，智能手机迟早会取代功能手机，甚至在很多场景下取代传统的计算机。从全世界来看，还有很多非发展中国家的智能手机用户数量占比还很低，发展的空间也更大。智能手机的出现，导致出现了各种各样的移动互联网应用程序，比如微信、QQ等聊天应用，腾讯视频等视频应用。大量移动用户使得这些移动应用程序对存储的需求变着越来越大，这对数据库系统提出了新的要求。

智能手机移动互联网应用的快速发展，伴随着应用程序数据量的快速增长，这就需要存储设备和软件来存储。现在主流大型应用程序的数据都是存储在数据库系统当中的，这方便数据在不同的应用之间共享，方便对数据进行统一的管理。在移动搜索应用程序中，需要利用数据库来存储各种网页和日志数据，对其进行存储和分析；在微信等聊天应用程序中，需要用数据库来存储用户和聊天数据等信息，而且需要对其进行加密和其他处理来满足用户对聊天应用的需求；美团等移动应用程序则需要利用数据库存储全国各地的店铺信息和各种交易信息。随着智能手机用户数量的快速增长，利用传统关系型数据库来存储这些数据已经出现困难。传统关系型数据库不方便在集群中部署，从而不方便对数据库存储系统进行扩展。

为了解决传统关系型数据[1]的问题，出现了各种各样的非关系型数据库系统[2]。在非关系型数据库中，数据库不再满足强一致性[3]和事务[4]要求，更加重视互联网集群[5]的可扩展性和性能的需求，更加适合大量移动应用程序数据存储场景，特别是非关系型数据的存储，比如各种移动应用程序日志等数据存储。在这几年里，非关系型数据库系统得到了非常大的发展，出现了各种各样的非关系型数据库系统。在企业中，常常会出现这样的情况：用键值数据库[6]系统存储用户数据；用文档数据库[7]系统存储图片和文档资料；用图形数据库[8]系统存储推荐引擎等用户关联数据；用关系数据库系统存储在线商城中的交易信息。总而言之，不同的数据库系统适合不同的数据类型，每种数据库系统都有其优点。但是

用多种数据库系统来存储多种企业数据类型，这对服务器资源是严重的浪费，此外，新的数据库系统常常需要新的维护人员，这增加了企业的数据库系统的人力维护成本。所以研究一种能存储所有数据库系统的数据库系统是非常有意义的事情。

非关系型数据库具有很多传统关系型数据库所不具备的优点，非关系型数据库抛弃了关系型数据库中严格的事务和强一致性要求，存储的一般都是没有关系的非结构化数据[9]，所以非关系型数据库更加容易在分布式环境下部署。非关系型数据库没有关系型数据库固有的关系模型[10]，存储的数据不局限于关系型结构化数据，使得更加容易使用。非关系型数据库不必要求强一致性和分布式事务[11]，所以在分布式环境下，更加容易扩展，更加适应大数据[12]的存储和处理。这些特点都使得非关系型数据库变得越来越流行，使用的公司越来越多。虽然新型非关系型数据库在大数据量的存储和处理上有很大的优势，但是关系数据库在很多关键领域又是无可替代的，所以如何结合关系型数据库和非关系型数据库的优点来开发一种新的分布式数据库系统成为当前数据库领域的关键问题。

基于移动互联网的快速发展对数据存储的需求，需要开发一种能结合关系型数据库和非关系型数据库的优点，能同时满足应用开发的便利性和应用程序对数据存储的高性能需求，其能存储多种类型的数据。如何将关系数据库技术、非关系数据库技术和分布式技术的优点相集合来开发一个新的分布式数据库具有极现实的意义。相对于传统的关系型数据库，这样的数据库结合分布式非关系型数据库的优点，能更加容易的部署在集群环境下，能更加容易存储大量的结构化数据，同时也能存储多种非结构化的数据类型；相对于非关系型数据库，这样的数据库结合了关系型数据库的优点，兼容SQL[13]接口，能存储结构化数据，更加容易满足传统应用程序的开发需求。本文设计和实现的分布数据库系统JSQL就是一种将分布式非关系数据库技术和关系数据库技术相结合的系统，和传统的关系型数据库和其他非关系型数据库系统相比，JSQL具有下面这些优点：

1.结合关系数据库和非关系型数据库的优点

JSQL分布式数据库系统是一种将非关系型数据技术和关系数据库技术相结合的系统，首先，它有非关系数据库系统的优点，能够处理非常大的数据，有很好性能和可扩展性，能支持键值模型，文档模型，能存储无结构化的数据对象。然后，JSQL作为关系数据库系统，能满足应用程序的开发要求，为应用程序提供SQL接口，提供关系操作和事务支持。提供的SQL接口让所有数据库管理员能快速入手操作数据库，提高数据库和应用软件开发效率。

2.提高数据库系统资源使用效率

一方面，JSQL是一种多模型数据库系统，其能存储结构化数据库系统代替传统的关系型数据库系统，其能存储很多非关系型数据库系统所能存储的数据类型。将所有数据存储在一个系统中，提高了服务器资源的使用效率。另一方面，JSQL分布式数据库系统具有负载均衡[14]功能，实现了分布式动态负载均衡算法，能根据当前系统的负载信息状态，将客户端的请求路由到正确的分布式数据库节点上面去，能均衡利用分布式集群系统中的数据库的处理能力。采用分布式的负载均衡算法，所有分布式集群中的机器都能参加到整个算法中来，每个集群节点都能成为分布式管理节点来实现分布式负载均衡算法，这样就能避免单点故障问题。通过实现动态负载均衡算法，分布式数据库系统的资源能均衡使用，提高了系统中各种网络和存储资源的高效利用。

3.提高系统的访问效率

本文设计的分布式系统能为客户端选择最合适的分布式数据库节点，这样能就够提高前端客户端的效率，使得其能更加高效的访问分布式数据库的资源。系统为客户端选择最近的节点，同时也节约了系统的网络资源。

4.提高系统的可扩展性

本文设计的分布式数据库系统，其分布式数据库节点集群采用多主分布式架构，能动态的增加节点，而不需要用户任何的配置。在动态扩展能力的前提下，当我们需要增加分布式系统的整体存储和性能时，只需要在当前分布式集群中增加一台计算机节点，就比如淘宝双十一的时候需要暂时增加系统的处理能力，就可以动态增加节点。而当数据库的读写需求降低时，只需要减少分布式数据库的节点就能动态的减少系统的资源，提高系统的使用效率。这种可以动态增加和减少系统节点的能力使得系统可以弹性扩展。

5.提高系统可移植性

本文设计和实现的分布式数据库系统全套代码由JAVA[15]语言实现，而JAVA是跨平台的开发语言，所以用JAVA语言开发的分布式系统JSQL能够部署到更广泛的平台上。使得分布式数据库系统可以在不同的平台上迁移和移植。

6.提高数据库系统安全性和易用性

将所有数据类型存储在一个数据库系统中，能够方便数据库管理员更加容易的维护系统，一个系统发生错误的概率比多个系统发生错误的概率要大的多，也更容易维护。当今数据的重要性越来越重要，如何保证数据库的安全性[16]是现在急切需要解决的问题，如何防止用户随意篡改数据在系统的设计上作为一个重要的功能需求。JSQL从数据库底层实现了数据库的审计功能[17]，提供比其他方法

更高的性能。管理员通过管理客户端能够监视数据库系统的运行状态，能监控数据的更改情况。

7.减少企业和单位的数据库维护成本

JSQL是一个结合关系型和非关系型数据库系统优点的系统，其能存储多种数据类型，能一个分布式的多模型数据库。企业在部署这样的系统以后，就不再需要部署不同的数据库系统来管理不同的数据类型。每种数据库系统都需要新的维护人员，非关系型数据库因为更新非常快，对维护人员的要求也很高，如果所有数据都能存储到一个系统里面，那么将减少企业和单位的人力成本。

1.2 分布式数据库研究历史与现状

数据库技术出现在上世界中期以后。在没有数据库技术的时候，使用系统的用户主要是用操作系统的文件存储数据。但是文件存储有太多的缺点，比如不容易在不同应用程序之间共享，不容易更改数据的格式，文件存储方式的导致数据库管理系统的出现。从60年代开始，出现了各种各样的数据库，其建立在不同的数据模型下，其中最有名的是层数据模型、网格数据模型[18]、对象数据模型[19]和关系数据模型。其中关系数据模型建立在严格的关系代数理论之上，具有数据独立性更高等优点而变得越发流行。随着关系数据库系统的理论逐渐成熟，关系型数据库系统得到了广泛的应用。在1974年，美国学者发明了结构化查询语言，这就是现在非常流行的SQL，SQL是现今大多数数据库系统，特别是关系型数据库系统的标准操作语言。SQL包括数据定义和数据操作等语言，使用这种语言不需要用户具有数据库和计算机程序设计相关的知识，这使得其在数据库管理人员中变得越来越流行，如今，每个数据库管理员都基本掌握这种数据操作语言。在90年代以后，关系型数据库在企业中成为标准，其操作语言的很多优点，使得其更加容易使用，容易被企业数据管理人员所接受。

分布式数据库系统[20]是部署在网络中多台计算机中的数据库系统，其结合现在的互联网网络技术和数据库技术，使得数据库的存储和处理能力在分布式环境下得到了极大的增强。分布式数据库技术的发展开始的很早，但是分布式系统在企业中的应用还是20世纪90年代的事情。因为在90年代以后，互联网才进入高速的发展，对数据存储的需求得到了爆发的增长。在互联网的快速发展下，分布式数据库系统在市面上出现了，在这其中出现了各种各样的分布式理论，这又促进了分布式数据库系统的发展。在2002，美国有学者提出了CAP理论[21]，CAP理论认为，在分布式数据库系统中，不可能同时满足CAP这3个条件，这三个条件分别是一致性、可用性和分区容忍性[22]。分布式系统理论CAP认为，在分布式系统

中，不可能同时满足一致性、可用性和分区容忍性这三个要求。根据CAP理论的指导，我们知道，传统的关系型数据库系统不能很好的满足现状分布式数据库的需求，因为现状分布式系统的发展是为了存储大容量数据而产生的。但是在关系型数据库中，其满足必须满足强一致性，所以分布式关系型数据库系统是不能再同时满足可用性和分区容忍性的。为了满足大数据量的存储，出现了很多非关系型数据库系统。

在二十一世纪以后，移动应用存储的快速发展，导致了非关系型数据库的出现。和关系型数据库不同的是，非关系型数据库抛弃了强一致性，所以根据CAP理论，其可以同时满足可用性和分区容忍性，这样就使得非关系型数据库系统更加容易部署到分布式的环境下，使得其更加适合存储大量的数据。非关系型数据库不支持强一致性，不支持强事务。因为抛弃了强一致性非关系型数据库支持高可用性和分区容忍性，使得其更加容易使用。在CAP理论出现以后，出现了很多非关系型系统，这些系统因为抛弃了强一致性，所以其可用性和性能都比传统的关系型数据库得到了很大的提高。特别使用很多不需要事务和强一致性的应用程序的需求，就比如现在的贴吧和微信的互联网应用，其得事务没有要求。在这些应用中，用户数据的丢失和错误并不会产生很大的影响。

在过去几年，随着移动互联网的快速发展，应用程序对数据存储的需求越来越高，分布式非关系型数据库得到了广发的使用，同时，CAP理论也证明了它的正确性。非关系型数据库抛弃了对强一致性的支持，非关系型数据库可以应用到对数据的强一致性和对事物不高的应用程序上面。但是，那些对强一致性和事物有要求的应用，比如电子商务，就不能使用非关系型数据库，因为其对事物有严格的要求，用户数据要保证正确和保证强一致性。但是，这些应用也发展的毕竟快，对大数据量的存储需求也在提高，所以如何存储这种数据成了一种新的挑战。要存储大量的数据，而单机系统的存储能力是有限度的，所以系统必须要能部署在分布式的环境下，其就必须要满足可用性和分区容忍性，但是分布式系统理论认为，在满足了可用性和分区容忍性这两个特点以后，就不能满足一致性。不能满足一致性对电子商务这样的应用就不能使用。所以现在的做法是尽量减少可用性和一致性，来开发新的分布式关系型数据库。在之前几年，每个大的公司都对这样的数据库开发提高了兴趣，这种数据库不但要满足关系型数据库的强一致性和事物的需求，还要部署在分布式环境下，满足对大数据存储的要求。下面对对目前市面上比较有代表性的分布式关系数据库系统进行简单的介绍。这些系统就是为了克服关系型数据库的缺点，同时能部署在分布式环境下存储大数据量结构化数据的分布式数据库。

1.Google Spanner

Spanner[23]是Google开发的一个可扩展的、全球分布式的数据库。在最高层面，Spanner就是一个分布式数据数据库，其将数据存储在分布式环境下的很多台计算机上，这些机器被部署在全球各地。在这个分布式数据库中，通过分片的复制，满足了数据的可用性。当一个副本失败以后，其他节点会自动恢复副本。在系统中，分布式系统会自动的将用户存储的数据进行分片，然后复制存储到不同的计算机节点。Spanner作为google开发的分布式关系数据库系统，其能部署在全球的各国国家和地区，能够存储海量的关系型结构化数据，还能支持全球化的分布式事务。

通过在每个不同的洲之间复制数据，可以保证，即使发生特大的自然灾害，存储在分布式数据库系统中的数据依然是可以用的。 Spanner是谷歌开发的一种全球化的分布式数据库，其提供了很多关系型数据库的特性。使用Spanner分布式数据库系统的应用程序可以对存储在系统中的应用数据进行详细的规划和配置，可以选择副本和复制策略。应用可以详细规定，哪些数据存储在哪些分区、哪个洲，不同的数据副本保存在什么节点上。还能配置，在副本失效的时候，从什么地方恢复。分布式数据库也可以在不同的分布式机器之间来回迁移数据，达到资源更加平衡的使用。作为一个分布式的数据库，其不但保证了每个数据分配的强一致性，而且通过原子操作，实现了分布式的事务。

2.阿里巴巴OceanBase

OceanBase[24]是一个阿里公司开发的一个分布式数据库系统，其能存储结构化的数据，是一个分布式关系型数据库系统，支持关系数据库系统中的常用操作，比如关系表的连接和事务。阿里巴巴在开发OceanBase的时候，把开发精力集中在数据库的主要功能上，对不常用的数据库功能，比如视图和存储过程，暂时抛弃，其主要关注分布式跨节点的事务功能上。

OceanBase在阿里的内部很多系统中已经得到了应用，比如最为出名的支付宝。支付宝在中国移动支付市场占有很高的比例，拥有上亿的用户，支付宝的成功运行使得OceanBase的可用性得到了证明。等待上线的应用还包括其他很多阿里巴巴内部和外部的应用程序中，每天更新的数据量超过20亿，更新数据量更是超过2.5TB。

OceanBase设计和实现的时候抛弃了很多不关键的数据库功能，使得其能很快的应用于生成实践中，而且根据CAP理论，在满足可用性和分区容忍性的时候，对强一致性要求必须要选择抛弃，阿里巴巴根据自身应用的特点，对不关键的功能进行了选择的抛弃。

3.微软SQL Azure

SQL Azure[25]是微软开发的部署在云上的分布式关系数据库系统，为应用程序提高分布式的存储服务。其主要部署在windows操作系统之上，为企业用户提高服务。

SQL Azure是一个强大的分布式的数据库，其建立的传统关系型数据库技术之上，很多传统关系型系统的功能在这上面得到了支持。SQL Azure是一个云数据库系统，将数据存储基础结构托管在云上，大大地降低了企业在IT方面的资源投入。企业用户按需付费使用分布式关系数据库，不用自己维护分布式数据库的部署，不用自己管理系统。

数据在企业中的变得越来越重要，所以要确保数据的安全，同时也要确保数据的高可用性，这样才能满足现在企业应用发展的需要。SQL Azure完美的支持可用性，在分布式数据库SQL Azure系统中，任何的数据库操作都会同步在其他分布式数据库节点，以保证数据的高可用性;SQL Azure支持分布式集群，支持故障自动转移，而且不需要人工手动配置。

这些数据库系统都是为存储结构化数据而开发的，而没有考虑到当前各种新型数据类型的存储。导致企业在部署这样的系统的时候，还需要再部署其他系统，增加了服务器和人力成本，不能达到资源的高效利用。

1.3 论文主要工作与创新

在学习了有关关系型数据和非关系型数据库的有关理论和技术以后，论文作者基于Java语言设计和实现了一个分布式多模型数据库JSQL，JSQL结合了非关系型数据库和关系型数据库的优点，能存储多种数据类型，能部署在分布式环境下，动态扩展读写能力，从数据库系统的底层加入审计功能以保证系统的安全。在系统的设计和实现过程中做的主要工作包括：

1.利用OrientDB[26]和ElasticSearch[27]非关系型存储引擎设计和实现了关系型数据库的本地存储系统。其结合了非关系型数据库系统的优点，底层采用新的存储结构，能够很好的避免关系型存储引擎在多表连接下性能的严重下降。

2.实现了Mysql[28]通信协议，使得可以通过Mysql客户端来连接JSQL分布式数据库，方便Mysql用户迁移到本数据库系统。在jsql之上，可以实现各种通信协议以连接不同的数据库客户端，考虑到开源mysql数据库系统使用的广泛性，论文暂时只实现了mysql的通信协议。

3.实现了对SQL语句的解析和执行，完成对关系数据库接口的支持。在非关系型存储引擎之上支持关系数据库操作就必须要实现SQL语句的解析和执行。因为SQL是关系型数据库系统通用的操作语言。

4.实现了数据库的多主分布式架构，使得数据可以存储在多台计算机上面，能够动态扩展分布式数据库系统的读写能力。论文利用开源的Hazelcast框架[29]实现集群节点的自动发现，利用新的多队列算法来解决分布式系统的数据一致性问题。

5.实现了系统的审计系统，使得系统的安全性得到增强。系统管理员通过连接数据库管理系统，通过图形化的界面能够看到分布式数据库系统的数据更改历史情况。

6.设计和实现了分布式负载均衡算法，完成了分布式数据库节点的动态负载均衡功能。分布式系统重要的功能就是如何在分布式集群中找到合适的节点，均衡分配服务器资源，已达到最佳系统总体最佳性能水平。

7.在分布式数据库系统开发完成以后，对系统进行了功能和性能测试，验证系统满足需求分析中的功能要求，在数据存储空间占用量的测试上，系统选择和阿里巴巴的OceanBase分布式数据库系统作对比。

与现有的大多数分布式系统相比，论文所述分布式数据库系统具有以下创新：

1.系统结合了非关系型数据库系统和关系型数据库系统的优点，在一个数据库系统里面提供多种数据操纵接口，能够存储多种数据类型的数据。

2.系统利用分布式队列实现多主分布式架构，能够动态增加系统读写能力。而传统的关系型数据库系统在分布式部署环境下，只有复制架构，其只能增加系统的读取能力而不能增加系统的更新能力。

3.在分布式系统内部实现负载均衡，实现分布式集群节点间服务器资源的均衡使用。能够提高系统的整体性能。

4.系统从数据库引擎层开发数据库的安全审计功能。现有的数据库安全产品是在数据库系统之上做的另外一套系统，其需要通过网络连接数据库系统，需要解析数据库的通信协议，所以其整体监控审计性能会受到影响。

1.4 论文结构安排

第一章作为论文的绪论，在这一章里，介绍了论文的选题背景和本论文所述系统研究和开发的意义。然后阐述了关系数据库和非关系型数据库的发展历史和

现状，对当前几个流行的分布式数据库系统进行了简单的介绍。在最好一节，对论文的主要工作与创新进行了阐述。

第二章，主要介绍了本论文相关的理论基础和相关的技术。这一掌首先对数据库系统方面的理论知识进行了介绍，包括数据库系统和关系数据库系统的概念，简单的讨论了NoSQL数据库系统；然后简单介绍了分布式数据库相关理论，对分布式系统中数据分布和数据复制技术进行了介绍，最后介绍了常用的动态负载均衡算法和分布式MVCC技术。

第三章，作为分布式系统JSQL的分析。本章首先给出分布式数据库JSQL的实现目标，并对系统进行了详细的需求分析。站在分布式数据库系统的使用者角度，对分布式数据库JSQL进行功能说明，并用用例图的形式对其进行进一步说明。在对重要模块进行分析以后，本章后面对分布式系统所用技术进行了介绍和说明。

第四章，作为分布式数据库系统的设计部分，本章包括JSQL的总体设计和模块划分，本章对系统的总体架构和各个模块的详细架构给出了阐述。系统主要包括分布式管理节点和分布式数据库节点集群。本章对分布式管理节点和分布式数据库节点的各个模块进行了详细的设计。

第五章，是关于系统的具体实现。根据第四章的总体设计和详细设计，本章对设计部分划分的关键模块的实现进行了说明阐述。重点阐述了客户端关键技术实现、管理节点关键技术实现和分布式架构关键技术的实现。对每个关键技术的实现给出详细的流程图，并加以文字分析。

第六章，作为系统的测试部分，本章对论文所述JSQL分布式数据库进行了测试，测试包括数据库功能测试、集群功能测试和性能测试。对测试的环境和方法步骤进行详细说明，记录测试结果并对其进行分析。

第七章，论文总结了论文所做的工作，分析了所述分布式数据库系统的不足，对未来的工作进行了说明。最后，作者对数据库系统的未来进行了展望。

最后部分，是关于致谢和参考资料。

第二章 相关理论和技术

本章介绍数据库和分布式系统有关的概念和理论，以及本系统所涉及到的关键算法。

2.1 数据库系统

2.1.1 数据库系统概念

数据库系统[10]就是一种管理数据库的软件，其一般建立在操作系统之上，为应用程序和用户提高操作接口。在数据库系统之前，应用程序用文件存储应用程序数据,这有很多缺点，首先，不同的应用程序就有不同的格式，存储的文件不能在不同的应用程序之间共享；然后，用文件存储应用程序数据不能在多应用程序之间并发执行程序，因为操作系统为了保证文件的安全，一般需要应用程序锁定文件，这样就不能让很多线程或者进程去操作应用数据；最后，用文件存储数据不方便应用程序数据格式转化，在应用程序需要更新应用程序数据格式的时候，就非常麻烦。数据库管理系统能很好的避免上面的缺点，其统一管理系统中的所有数据，应用程序通过向数据库系统发生命令来操纵数据，这样保证了数据的安全，更方便数据在不同应用程序之间的共享。

数据库管理系统为应用程序和各种抽象数据提供了一个软件抽象层，应用程序或者用户通过数据库管理系统来操作数据，不需要关心数据的具体物理存储格式，更不需要了解底层磁盘系统如何存储这些数据。不同的应用程序通过数据库系统可以共享数据，使得数据更加容易更新和维护。在更新数据库管理系统的时候，只要这个管理系统的操作接口和之前的数据库管理系统一样，那么应用程序就不需要做任何的更改就能进行运行，这种抽象为应用程序提供了很大的便利。

2.1.2 关系数据库系统

在数据库系统的发展早期出现了其他多种数据库系统，关系数据库系统[1]因为许多优点从其被发明以后就被用在企业的各种应用之中。关系数据库系统建立在关系模型之上，关系模型以关系代数为理论基础，有严格的数学理论基础。在关系数据库系统中，所有数据都被表示成数学上的关系。关系型数据库系统通过表的结构来存储结构化的数据，其不能很好的存储其他无类型的数据。

在关系数据库系统中，数据存储在表中，一个表代表一种结构化数据类型。表的每列代表这种结构化数据类型中的一个属性，表的每行代表一个这样的结构

化数据记录。在数据库管理系统中，数据之间的关系是通过表与表之间的外键相互联系的，而不是在记录中直接存储数据之间的关系。这样的好处是能够减少数据的存储量，在寻找数据的关系数据的时候就需要通过外键遍历其他的表，在表增多的时候，性能就会受到到严重的影响。

关系数据库系统有通用的结构化操作语言SQL，在关系数据库系统中，应用程序和用户通过发送SQL命令来操作数据。SQL操作语言非常容易使用，使用者只需要说明操作目的，而不需要说明操作步骤。关系数据库系统会解析SQL，寻找最优化的执行过程。

关系数据库支持严格的事务，要求事务必须要满足原子性、一致性、隔离性和持久性。这些特性能够保证结构化数据的一致性和安全性，在高并发情况下，关系数据库系统也能保证数据处于一致性的状态。数据一旦存储在关系数据库系统中，就永远持久化存储在磁盘中。

2.1.3 事务与并发控制

2.1.3.1 事务

事务能保证关系型数据库中数据的强一致性和正确性。一个事务表示对数据库系统的一系列的操作，关系型数据库管理系统能保证这些操作的正确执行，而不会破坏数据的完整性。关系型数据库的事务有两个主要目的：为了提供一个可靠的工作单元，在这个单元中，允许从故障中正确恢复，并保持数据库在任何情况下都是强一致性的。即使在系统故障的情况下，关系型数据库也能保证事务的正确完成，避免数据的丢失和数据的不一致。事务也能在同时访问数据库的程序之间提供必要的隔离，如果没有提供这种隔离性，程序的执行结果可能是错误的和不可预见的。根据关系数据库的定义，数据库事务必须是原子的、一致的、隔离的和持久的。在一个事务执行单元中，所有步骤必须正确完整的执行完成。此外，关系数据库系统必须将每个事务与其他事务隔离开来，不然可能会产生不一致性的结果，同时，事务的执行结果必须存储在永久性存储介质中，不能丢失。

数据库完整性是事务的一种重要保证，如果不能保证数据的一致性，那么关系型数据库将是不可靠的。每个事务都是不同的执行步骤组成，在事务执行过程中，每个步骤要么全部完成，要么全部失败，没有中间结果。事务的执行必须是可预测的，关系型数据库不准出现不可预测的不一致性的结果，这样数据的完整性不能得到保障，就不能正确的使用数据。

2.1.3.2 并发控制

并发控制[30]在计算机和数据库系统中都是关键技术，操作系统通过并发控制防止硬件和软件资源的错误使用，数据库系统中数据的一致性和正确也需要并发控制技术来保证的。在多并发操作的情况下，如果没有并发控制，多个事务的操作结果是不可预测的，最严重的情况是会产生错误的数据，导致数据的不一致性，这样数据库管理系统就是不可靠的。不管是操作系统和数据库管理系统，其并发算法的执行都需要运行开销，所以如何保证正确性的同时减少系统的开销是数据库系统中重要的话题，所以不同的数据库管理系统都有不同的并发隔离级别，提供给用户选择使用，让用户根据具体场景选择正确的并发控制级别。

并发控制在操作系统和数据库管理系统，或者其他的关键系统中都是正确性的保证。在关系型数据库系统中，不同的事务同时执行时，其结果往往不可预测，如果没有并发控制，数据的正确性也得不到保证。在不同的关系数据库系统中，其采用的并发控制技术会有所不同，但是总结下来，主要的并发控制方法有如下几种：

1.锁定

锁定是并发控制方法中最重要也是最容易理解的方法，在操作系统和数据库管理系统中都有应用。在关系数据库系统中，多个事务同时访问一个数据库对象，就会产生不一致性的结果。所以我们只需要让事务在访问对象的时候，锁定其访问的对象，这个时候其他事务执行过程中就不能访问，只能等待，最终防止数据不一致性的情况出现。

2.串行化

串行化是最严格的并发控制技术，其开销也是最大的。在这种并发控制方法中，禁止事务并发执行，在这种情况下，不会出现数据错误的情况，但是其性能会非常低下。

3.时间戳排序

为每个事务分配不同的时间，让其按照不同的时间执行，这样可以防止同时访问数据库的时候产生不同的结果，影响数据的正确性。其和串行化技术是相识的技术。

4.承诺排序

为不同的事务分配不同的顺序，保证在同一个事务中不可能访问到其他事务的执行结果，防止每个事务的相互影响。这种情况是为了提高并发度，数据库管理系统检查事务的顺序，在保证正确的情况下让事务依次运行。

5.多版本并发控制

通过在每次写入对象时生成新版本的数据库对象，并根据调度方式允许事务对每个对象的最后相关版本的读取操作来增加并发性和性能，但是这种情况会增加数据存储空间的占用。

6.索引并发控制

将访问操作同步到索引，而不是用户数据。这种专业方法提供了显着的性能提升。

每个事务在执行的时候和其他事务保持隔离，事务执行中间结果不会被其他事务看到，这样才能防止很多数据操作的错误。关系型数据库系统一般会提供多种隔离级别，在最高级别的情况下，能保证数据绝对正确，但是并发性能较低，在最低级别的时候，并发性能最高，但是可能会出现数据的不一致性情况的出现。

2.1.4 非关系型数据库系统

随着移动互联网的快速发展，对数据库系统的要求越来越高，要求能存储海量数据，能支持高并发的移动客户端的访问，关系数据库系统不能支持这些新的要求，所以就出现了很多的非关系型数据库系统。非关系型数据库系统由于其非常适合很多大型移动互联网网站数据存储，所以近几年得到了高速的发展，在很多互联网公司中都有不同的应用。非关系型数据库系统是在关系型数据库系统成为主流的时候出现的，其要解决关系型数据库所不能解决的问题，比如分布式环境下部署，大量数据的存储，高并发客户端的访问等等。虽然非关系型数据库的流行还没有多久的时间，但是不可否认，现在已经对关系型数据库产生了非常大的影响。虽然早期非关系型数据库都比较简单，不过现在很多非关系型数据库已经变得更加成熟，越来越适合现在数据的存储。不过，现在很多非关系型数据库不得不重写，已应对新的存储需求，从一开始，其就缺少很多关系型数据库的特性，虽然其抛弃了很多关系型数据库的特性，使得其在一些方法，比关系型数据库更加适合。但是关系型数据库系统的很多功能是很多移动互联网应用所必须要满足的，而要满足这些需求，之前的很多非关系型数据库就必须要经过重写。

非关系型数据库并没有准确的定义，也有不同类型的非关系型数据库系统，比如关系模型最简单的键值数据库系统，存储图片和文档等文档数据库系统，还有推荐引擎中的图形数据库系统。在有些应用程序中，甚至也有对象数据库系统。不管有多少非关系数据库系统，其大多都普遍存在下面一些共同特征：

1.无模式数据存储：在非关系型数据库系统中，不需要像关系型数据库系统一样，在存储数据的时候必须要先确定数据的结果和模式，并建立相应的表结构。数据库系统中的每条记录都可能会有不同的属性和格式。当插入和修改数据时，并不需要定义它们的存储模式。

2.无共享架构：非关系型数据库系统在分布式部署中，其没有共享的架构，存储在分布式集群中不同的存储节点。客户端从最近的网络节点存取数据，比通过网络获取数据能提高系统的性能。

3.弹性可扩展：可以在系统运行的时候，动态增加或者删除结点。不需要停机维护，数据可以自动迁移。

4.分区分片：传统的关系型数据库系统大多都是单机数据库系统，其不能很好的在分布式环境下部署，而非关系型数据库已经放弃对复杂事务的支持，其能方便的部署在分布式环境下，数据不存储在单一节点，存储在分布式环境下的多个节点之上。在数据库集群中，不但要对数据进行分区存储，还要对分片进行复制。这样既提高了数据库系统中的并行性能，又能保证集群系统中没有单点失效的问题。

5.异步复制：和很多关系型数据库系统不同的是，非关系型数据库中的复制，往往就是基于日志的异步复制。异步复制可以让复制更快的完成，不会因为网络原因而阻塞。这样做的缺点是并不总是能保证一致性，而非关系型数据库系统本身就没有保证强一致性，这样的方式在出现节点故障的时候，就可能会丢失少量的数据，在一些场景下，这并不是很严重的问题。

6.BASE：和关系型数据库的强一致性不同，非关系型数据库系统采用的是弱一致性，也就是说，并不能保证数据一定可靠，它只提供必须的一致性，这样就能提高更好的性能。

2.1.4.1 Nosql缺点

虽然NoSQL拥有许多好处，但并非没有缺点。简单的说，非关系型数据库大多有下面这些缺点：

1.技术支持：NoSQL数据库往往是开源项目。当出现问题时，您可能有幸接触到开发人员社区并找到您的答案。不幸的是，今天最强大的企业无法承担那些不够完善，缺乏全面支持的产品。好消息：当今领先的NoSQL公司倾向于提供免费的商业产品以及为倾向于包含企业级支持的组织设计的高级产品。

2.不成熟：虽然NoSQL已经存在了相当长的一段时间，但直到最近NoSQL数据库才开始严重侵入现代企业。因此，许多NoSQL解决方案并不像许多组织那么成熟，特别是与RDBMS相比时更是如此，这也是导致出现各种非关系型数据库系统，并不是一家独大。

3.不符合ACID：特别是公司和电子商务组织，非常重视所谓的ACID合规性。当数据库可以确保每次交易的原子性、一致性、隔离性和持久性时，它们就被认为符合ACID标准。本质上，ACID合规确保数据库按预期处理事务。虽然许多NoSQL数据库不符合ACID标准，但本论文所选的OrientDB支持基本的事务操作。

OrientDB是一种非关系型数据库系统，虽然它不是关系数据库系统，但是其底层的存储引擎支持关系型数据库中的事务。所以本系统在后面就直接用的这个存储引擎减少系统的开发时间，同时也增加系统的稳定性。同时能够结合非关系型存储引擎和关系型数据库功能，能为系统提供更好的实用性。

2.2 分布式数据库

2.2.1 分布式数据库概述

分布式数据库系统[20]也是一个数据库管理系统，只是其管理的数据不存储在单机的计算机主机上面。分布式数据库系统的数据可以存储在相同位置的多个计算机上面，也可以存储在通过网络互连的多个地区的计算机集群上面。分布式数据库系统的节点分布在网络上的各个地方，节点之间没有共享物理组件。分布式数据库相比单机版数据库系统有下面这些特点。

2.2.2 分布式数据库的特点

1.数据独立性

分布式数据库系统中的数据独立性是指，在分布式数据库系统中，集群中的每个节点都独立管理其当前局部数据，就算没有连接互联网，单机情况下，其也能对外提供服务。客户端也能连接这个节点，这个节点为客户端提供本地局部数据操作的能力。分布式数据库系统节点的独立性能够保证系统的高可用性，在分布式系统中，一个节点的失败不会影响到其他的节点，其他节点能正常的为本地客户端提供服务。在分布式数据库系统中，物理位置的独立性和逻辑独立性是两个基本要求，数据物理位置的独立性是指在分布式环境下，数据的位置发现变化的时候，其客户端能保持不变，逻辑独立性是指，在系统的内部存储结构发生变

化的时候，分布式数据库系统对面提供的接口保持不变，这样应用程序就可以和系统保持独立变化。

2.分布透明性

在分布式数据库系统中，分布透明性是指，数据库的客户端不需要了解分布式数据库系统中数据存储的具体物理位置，客户端对分布式系统中数据分布的变化不用了解。有了分布式数据库的分布式透明性，当数据库的物理存储位置发生改变时，其数据库用户应用程序不需要知道变化，能进行保持运行。这样就能保证在分布式数据库系统在系统升级和数据迁移时候，上面的应用程序不需要做任何的改变就能适合新的物理架构。这样我们就能从容的改变分布式的物理分布式，而不用担心会影响到应用程序。

3.节点自治性

节点自治性是指，在分布式数据库系统中，每个分布式数据库节点能够自己管理其本地数据，不需要分布式数据库中其他节点的参与。对于客户端而言，其不必关心连接的是分布式数据库还是分布式数据库中的一个节点。每个数据库节点能够自己管理其中的数据，而且功能和分布式数据库系统本身对外提供的功能一样，这样连接分布式数据库任何节点的客户端，其感受到的功能都是一样的。其所能访问的数据由分布式数据库的所有节点组成，也是一样的。

4.复制透明性

在分布式数据库系统中，复制技术是为了保证系统数据的安全性和可用性。分布式数据库系统中，通过多个数据副本，为客户端选择最适合的读取路线，选择最适合的副本，有利用提高分布式数据库系统中读取性能。在分布式数据库系统中，数据复制多少份，复制副本如何分布式这些问题都是由分布式数据库系统自动完成的，应用程序不需要关系复制的细节。更简单的说，在分布式数据库系统中，客户端根本就不用感受到副本的存在，副本的复制策略和分布式规则都是由分布式数据库系统自动完成。

5.易于扩展性

在分布式数据库系统中，可能会遇到系统性能和存储不能达到要求的情况，这样我们就遇到对分布式数据库系统进行扩展。易于扩展性是指在分布式数据库系统中，很容易对系统的整体性能进行扩展，包括对读取和更新性能的扩展，对存储容量的扩展。在最理想的情况下，分布式数据库系统能够动态增加数据库节点，动态扩展系统的性能。在系统需要暂时进行扩展的时候，只需要在分布式数据库节点集群中加入新的主机就可以了。

2.2.3 数据分布和负载均衡

在分布式数据库系统中，单机服务器不能存储所有的数据，如何把数据分布在分布式环境下的多个服务器上面需要一种数据分布方式。下面将介绍几种不同的数据分布方式[31]。

2.2.3.1 哈希分布

在分布式系统中，哈希分布是指，分布式系统中的所有数据通过计算关键字的哈希来决定其分布的位置，在具体的实现中，其往往用计算的哈希值和分布式数据库节点中的机器数比较，先为集群中的每个节点分配一个序号，然后用关键字的哈希和总节点数取模，最后用取模的结果和机器序列号做比较。在哈希分布式中,分布式系统需要管理的元信息很少,只需要知道哈希函数和计算机集群中节点个数。但是这种方法有个很明显的缺点,那就是其扩展性很差。比如，如果想把集群规模扩大一倍,那么所有集群中的的数据需要被重新迁移到不同的节点中。一种解决哈希分布扩展性差的问题是引进元数据服务器，这种方法的思路是，不再把关键字的哈希值和分布式集群中的分布式节点数取模，而是在元数据服务器中存储数据和分布式节点之间的对应关系。在获取数据的时候，客户端需要先链接元数据服务器得到分布式数据库节点的地址，然后链接分布式数据库节点得到具体的数据。不过，这样一样就需要保存很多的元数据，在分布式集群集群数变得很大时，这样的方法就存储很大的性能问题。

2.2.3.2 顺序分布

除了哈希分布以外，在分布式数据库系统中，也可以按照数据关键字的范围来分配数据，在这种方法中，首先将数据关键字划分成不同的区间，然后将区间和分布式数据库服务器节点映射。比如我们可以将用户划分到不同的区间，分区到分布式系统中不同的节点存储。用数据范围来解决数据的分布式,实现动态划分范围空间,实现负载均衡。这样方式分布数据，需要保存什么区间的数据存储到什么数据库节点上面去，这样的信息可以存储在专门的元数据节点服务器中。使用这样的分布方式，可以在不同的机器之间迁移数据，而不会影响到过多的分布式数据库节点，而只需要在元数据库服务器中，更新分布式区间和数据库节点之间的对应关系，这是哈希分布式方式所不能做到的。按顺序分布的优点还有，在扩展分布式数据库集群的时候，当增加节点的时候，不需要迁移全部的数据，只需要将新的分区迁移到新的数据库服务器节点就可以完成数据库集群的扩容。这样的分布方式问题就是需要存储所有的数据分区到数据库节点之间的对应关系，当数据库的容量增加时，元数据可能会成为分布式数据库系统的瓶颈。

2.2.3.3 一致性哈希

一致性哈希的分布式方式是为了克服简单哈希分布的缺点。在简单哈希分布式中，当增加数据库节点的个数的时候，我们需要迁移非常多的数据。这样就性能系统的扩展时间，影响系统的性能。在一致性哈希中，分布式数据库节点的哈希值保持在一个圆圈上面。然后根据关键字计算哈希值，在圆圈上面找最接近的哈希值。然后就把这个哈希值对应的数据存储在对应哈希点的分布式数据库节点上。在增加或者删除数据库节点的时候，只需要迁移相邻分布式节点之间的数据，其他节点的数据可以保持不移动。这样就能提高分布式系统的性能。使得更加容易扩展其容量和性能。

2.2.3.4 负载均衡

在分布式数据库系统中，负载均衡是用来平衡各种网络和服务器资源的使用的。通过负载均衡，客户端可以选择合适的服务器来操作分布式数据的存储。避免一些分布式节点因为过载而不能提供服务，避免一些分布式数据节点因为得不到客户端的请求，而浪费分布式系统整体的资源使用率。通过使用负载均衡，能使系统高效的完成客户端的请求，同时避免资源的浪费。负载均衡功能可以通过硬件和软件的方式来实现，软件负载均衡是根据不同的负载均衡算法来实现的。有很多种负载均衡的算法可用，下面是常用的一些算法：

1.轮询法

轮询法可能是是负载均衡算法中最常用最简单的算法，这种算法也很容易理解，同时也容易实现。轮询法是指分布式系统中负载均衡服务器将客户端的请求按顺序轮流分配到分布式系统后端服务器节点上，以达到负载均衡后端数据库节点资源的目的。

2.加权轮询法

简单的轮询法没有考虑分布式系统中每个节点性能的差异，也没有考虑分布式系统节点的当前负载状态，所以不是高效的负载均衡算法。加权轮询法则是考虑了分布式数据库节点中每个节点性能的差异，当客户端连接到来的时候，在为其选择分布式节点的时候，考虑到分布式节点的性能和当前的负载状态。为其选择更适合的分布式数据库节点。

3.最小连接数法

最小连接数法也很简单，其根据分布式系统中当前节点的连接数来分配节点。为客户端选择当前连接数最少的数据库节点。这种方法认为，当前连接数最少的服务器节点，其当前的处理能力最强，所以应该分配给客户端。但是这种方法没有考虑到分布式系统中不同节点的性能差异。所以也不是很好的算法。

4.随机法

随机法很简单，其利用数学上的概率知识，随机的从分布式节点中选择一个节点。从概率上来说。这样每个分布式节点被分配的几率是一样的。经过系统的运行，每个分布式节点的连接数应该是差不多的。但是这样的算法，同样是没有考虑系统当前的机器负载状态和性能差异。所以也不是很好的算法。

5.源地址哈希法

这种算法的想法是，根据客户端的一种信息，一般是网络地址。然后计算哈希值。然后和分布式数据库中节点个数取模。就能得到一个值，然后根据这个值。选择分布式节点。这样的方法。首先需要提前排序好分布式数据库节点顺序，所以当其中一台计算机节点失败的时候，就可能需要重新计算所有的哈希值。影响之前的分配。所以这样的方式也不是好的分配算法。

好的负载均衡算法影响是动态的负载均衡算法，其能考虑到不同的分布式节点的性能差异，同时能考虑到当前每个节点的负载信息。这样就能正确的分配的分布式节点。本章后面提出了一种新的分布式负载均衡算法。其不但能够满足这里提出的要求，而且在分布式上运行，能避免单点故障。

2.2.4 数据复制和一致性

2.2.4.1 复制的概述

在分布式数据库系统中，采用数据复制技术来保证数据的安全。复制按照交互方式可以分为同步复制和异步复制，在同步复制方式中，主节点需要等待所有副节点的确认以后才能更新数据，而在异步复制中，主节点收到任何一个节点的确认信息以后就可以更新数据。在分布式数据库系统中，复制是保证数据安全的手段，但是在分布式系统中，不同的复制副本需要保证一致性，不然数据就是不可用的。在分布式数据库系统中。有几种复制方式来保证数据的一致性。

一种保证数据一致性的方式是主从复制，按照交互方式，主从复制也可以分为同步复制和异步复制两种方式，同步复制更加安全，而异步复制更加高效。在MYSQL主从架构中，主从数据库服务器之间默认情况下采用异步复制，虽然这种方式在一般情况下，能够增加性能，减少复制时间，但是也容易引起数据丢失的问题。比如主节点在没有收到从节点复制完成确认信息的时候就挂掉了，从节点的数据就可能和主节点不一致了。即便采用了同步复制，数据也不能保证安全，比如，当主接收写入请求然后发到从节点，从节点写入成功后并发送确认给主，但是这个时候，主节点正准备发送确认信息给客户端时主节点挂了，那么客户端就会认为提交失败，可是这个时候从节点已经提交成功了，如果这个时候从节点

被提升为主，那么就出现问题了，数据不一致了。在主从复制结构里，异步复制有更好的性能。同步复制则相对更加安全，所以分布式数据库系统中，往往结合这两种复制方式。

在分布式系统中，另一种保证数据一致性的复制方式是引进分布式一致性算法Paxos，在这种方式中，Paxos算法本身保证了分布式数据库集群中节点之间数据的一致性，现在很多商业分布式数据库就是采用的这样的方式，但是这样的方式实现困难，容易出现问题。主从复制因为简单部署性，在传统数据库系统中也得到了广泛的应用。

2.2.4.2 一致性和可用性

在分布式数据库理论中的CAP定理，又被称作布鲁尔定理，有特别重要的作用， CAP理论认为，在分布式数据库系统中，不可能同时满足以下三个要求：

1.一致性

2.可用性

3.容忍网络分区

根据CAP理论，任何一个分布式数据库系统都不可能同时满足一致性、可用性和分区容忍性。在实现分布式数据库系统的时候，一般都要选择实现的功能特性。因为不可能一个系统满足三个特性。比如，在很多非关系型数据库系统中，抛弃了强一致性，而选择了可用性和分区容忍性。在关系型数据库中，则必须满足强一致性特性，所以就只能满足可用性和分区容忍性。这也是为什么关系型数据库不能很好的在分布式系统部署的原因。在分布式数据库系统中，一般选择了减少强一致性要求来实现可用性和分区容忍性。从而使得数据库在分布式环境下能部署。

2.2.5 副本和分布式MVCC技术

2.2.5.1 副本的概念

副本是指在分布式数据库系统中为数据提供的冗余存储，以保证数据的安全性。因为在分布式数据库集群中，多个数据库节点存储同一份数据，所以在一个节点的数据丢失时，可以从其他分布式数据节点获取副本数据。在分布式系统中，数据副本是保证数据安全性的重要手段。

2.2.5.2 副本一致性

在分布式数据库系统中，通过一定的分布式副本协议或者方法，使得数据库用户在任何情况下操作和读取数据副本，都能保持每个副本数据之间保持相同，称之为副本一致性。副本一致性是保证分布式数据节点之间数据一致性，而不是针对单个数据库节点。分布式数据库系统为了提高可用性，一定会使用副本的机制，就会引发副本一致性的问题。

2.2.5.3 分布式 MVCC

在数据库系统中，MVCC技术[32]是保证数据一致性和提高数据库更新性能的一种方式。在这种方式中，用存储空间来换取数据操作性能。在MVCC技术中，不但存储数据本身，还要保存数据的版本。在更新数据的时候，更新数据的同时，记录下数据的版本号；在读取数据的时候，这只读取版本号大于当前更新操作版本号的数据；删除数据则不真的删除数据，而只是记录下删除版本号。这种方式能够减少不同的事物之间的冲突，提高高并发情况下数据库系统的更新性能。

分布式MVCC技术则是在分布式数据库中，在多个客户端同时更新数据的时候，保证数据一致性的一种方式。实现的原理同样基于数据库系统中的MVCC技术。每个客户端操作都有一个分布式版本号，更新数据的时候除了更新数据，还要保存分布式版本号。这样多个客户端可以同时更新同一份数据，而不会引起冲突，提高分布式数据系统的性能。在删除数据的时候，只是简单的记录下删除版本号，当客户端检查到这个删除版本号的时候，不会检索当前数据给客户端。

2.3 本章小结

本章对数据库系统和分布式数据库系统中的关键技术进行了介绍，这是论文继续研究下去的理论基础，其内容包括：非关系型和关系型数据库系统的概念和区别，数据库系统中事务和并发控制技术，分布式数据库常用的负载均衡算法，分布式数据库系统中的数据一致性和分布式MVCC技术。在常用的负载均衡算法基础之上，论文在实现部分提出了新的分布式负载均衡算法，在分布式相关技术的基础上，论文在后面提出了多主分布式架构。

第三章 系统分析

分析、设计、开发和测试是软件开发的不同阶段。分析包括需求分析和实现技术分析和选择，软件系统采用的技术是系统设计和实现的重要基础。因为不同的技术需要不同的设计，不同的实现过程。系统需求分析在软件开发生命周期中起到了非常重要的作用，任何一个软件系统在开始设计、开发和测试步骤之前，都必须要明确所开发系统的功能，并用文字记录下来，一旦需求分析出现错误，这将会为后面的开发过程带来非常大的麻烦，需求分析的目的就是明确系统的功能和性能要求[33]。系统设计是描述如何实现需求分析中的功能，给出系统的架构和模块划分图，可见，没有经过仔细的系统分析就不能做好后面的系统设计和编码工作。本章对分布式数据库JSQL进行分析，包括需求分析和技术分析。在对系统进行需求分析以后，本章后面对各个功能模块的技术进行了阐述及分析。

3.1 系统需求分析

需求分析是站在系统使用者的角度上，把系统当作一个黑盒子，分析系统需要满足用户的功能，而不需要考虑这些功能如何实现。在对系统所有功能进行需求分析以后，需要把分析的结果记录下来，为后面的软件设计提供基础。

3.1.1 系统功能需求

本论文所研究的分布式数据库系统，主要是存储大量不同数据类型的数据，所以需要结合关系型数据库和非关系型数据库的优点，系统能够存储不同模式的数据，还能够扩展分布式系统的读写性能，在所有上面这些功能之上，还必须保证存储在分布式数据库系统中数据的安全。本论文所设计和开发的分布式数据库系统JSQL的主要功能如下：

1.提供负载均衡功能

论文所述的分布式系统，为了实现分布式系统中每个集群数据库节点的资源的合理分配，需要系统具有负载均衡功能，即前端客户端发送的数据操作请求消息能够均衡的发送到各分布式数据库节点。因此本文提出了分布式管理几点，该几点的功能构成本文的分布式管理模块的主要功能，其为客户端选择合适的后端分布式数据库几点，从而达到负载均衡的效果。分布式管理节点只是一个功能节点，其上实现分布式负载均衡算法，为客户端选择合适的后端服务器，同时其也能提供后端数据库存储功能。

2.提供数据库功能

论文所述系统作为一个数据库系统，其应该具有存储和管理数据的能力。本文结合非关系型数据库和关系数据库技术。实现了一种分布式数据库系统。对用户来说，本系统提供的功能接口和关系型数据库接口是一样的，也就是就是SQL接口，更准确的说，是Mysql接口，所以系统应该提供关系型数据库常用的功能。同时本系统利用非关系型数据库OrientDB存储，实现数据的高效存储和访问，利用数据库集群实现数据库存储功能的可靠性。

3.提供管理监控功能

JSQL系统在分布式管理节点和分布式数据库节点的结合下，从数据库的底层实现了数据库的安全审计的功能。在分布式管理节点，实现了审计监控的界面功能，管理员通过管理计算机连接分布式管理节点就可以对数据库进行管理和维护。同时也能进行监控报警功能的设置。而监控日志等信息存储在数据库引擎内部，保证监控数据的可靠安全。

从节点功能上分配上，分布式数据库系统JSQL可分为两部分，提供负载均衡功能和管理监控功能的分布式管理节点，提供数据存储和管理的数据库基本功能的分布式数据库集群节点。具体的，提供的数据库功能主要包括：

1.数据定义：分布式数据库系统能提供数据定义功能，数据定义能让用户定义数据模型结构。数据定义保存在数据库字典中。因为本文阐述的是一个多模型数据库，其数据定义相当灵活，可以定义字段，也可以以后增加和删除字段。

2.数据操作：分布式数据库系统提供数据操作功能，供用户实现对数据的追加、删除、更新、查询等操作。用户通过接口向分布式数据库系统发送命令，获取数据操作结果，一般的操作结果采用传统的SQL语言。

3.系统运行控制：分布式数据库系统系统运行控制功能是维持分布式数据库系统正常运行的控制和操纵功能，包括多用户环境下的并发控制和安全管理，这些功能是系统正确运行的基础。

4.数据组织：数据库系统中的数据组织功能是选择数据在底层硬件的存储格式。不同的数据库系统可以选择不同的数据组织和存储方法，系统的目标就是选择合适的组织和存储方式来提高系统存储资源的利用率，选择合适的算法压缩数据也可以进一步的减少数据的存储空间占用。

5.数据保护：数据是所有应用程序运行的基础，保证数据的安全是数据库的重要功能。数据保护的目标是防止数据出现破坏或者丢失，比如防止数据库中出现不一致的数据，防止人为修改数据，在系统故障的时候可恢复数据。

6.数据库的维护：这一功能主要是对数据库系统本身进行监控和管理，使得其一直保持在最佳的状态，这样才能使得数据库系统的性能在最高水平，保证系统数据的安全。数据库维护一般是数据库系统软件以外的工具软件完成，也可以由第三方软件公司开发作为工具软件使用，使得更加容易维护数据库系统。

3.1.2 系统功能用例

用例图用来描述系统用户和系统的关系。一个用例图包括系统功能用例图，系统的每个用例代表系统的一个功能[33]。用例描述的是系统的功能，是站在系统的使用者的角度描述功能，不是描述系统内部功能实现细节。站在用户的角度上，分布式数据库系统应该能够对存储在其中的数据进行增删改查等操作。系统总体的用例图如图3-1所示。

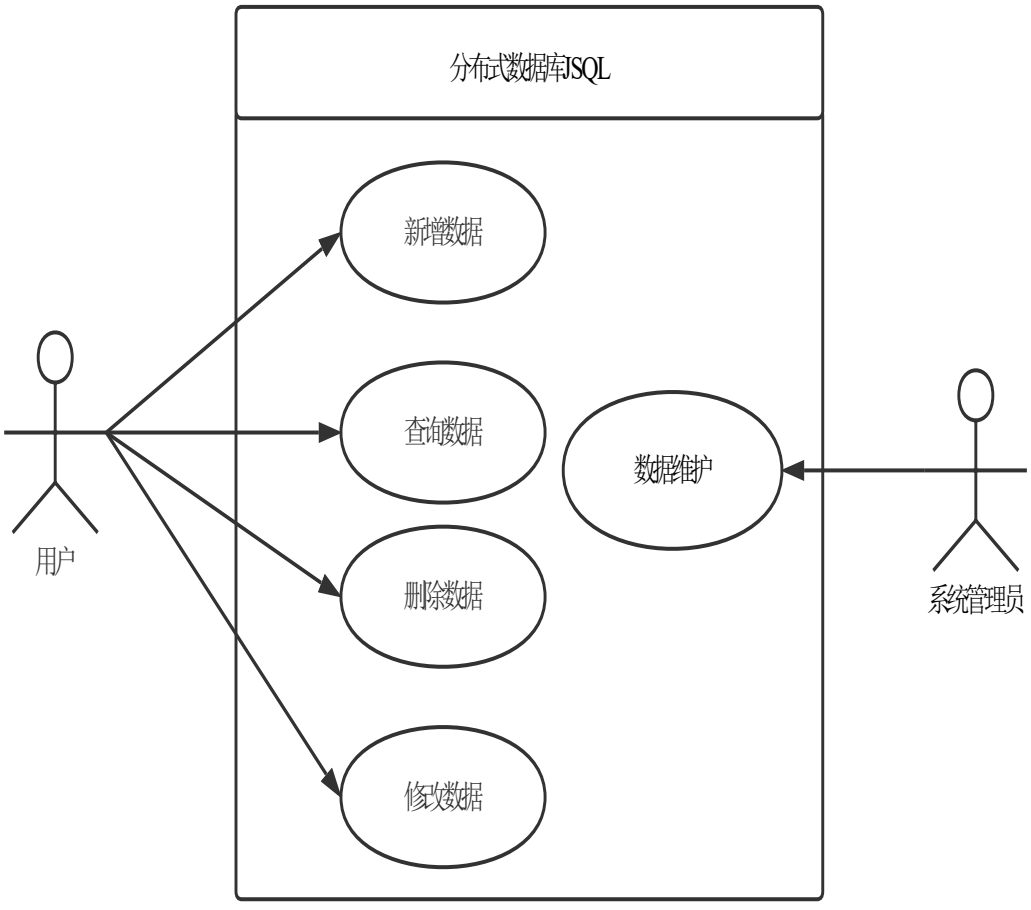


图 3-1 系统用例图

分布式数据库的主要功能用例为新增数据、查询数据、修改数据、删除数据等，下面对这几种重要操作的用例进行说明。

1.新增数据是用户从客户端向系统发起新增数据请求，在数据库中存储新的记录。具体用例见表3-1。

表 3-1 新增数据用例

用例名称	新增数据
用例描述	客户端发送新增数据的请求到分布式数据库
参与者	分布式数据库的用户
前置条件	1、 系统各个模块运行正常。 2、 用户已经成功成功登陆系统。
后置条件	分布式数据库系统能够正确处理客户端请求，成功保存数据。
基本操作	1、用户运行客户端，用户登陆。 2、在客户端上发送新增数据的命令。 3、在客户端上确认发送信息后，请求消息被发送到分布式数据库。 4、数据库存储成功后返回操作成功，如果失败，返回失败原因。
业务规则	如果数据信息已经存在，重复发送命令会增加新的数据

2.数据查询是通过记录的其中一个关键属性查询其整个记录的操作，用户一般是用SQL语言向数据库系统表达命令请求。具体用例见表3-2。

表 3-2 查询数据用例

用例名称	查询数据
用例描述	用户通过客户端发起数据查询的请求
参与者	分布式数据库用户
前置条件	1、 分布式数据库系统各模块运行正常。 2、 数据信息已经存入数据库。 3、 用户已经成功登陆客户端。
后置条件	分布式数据库系统能够处理客户端请求，正确返回用户查询数据。
基本操作	1、 运行客户端，登陆分布式数据库。 2、 在客户端上发送查询数据的命令。 3、 在客户端上确认信息后，请求消息被发送到存储该数据的数据库。 4、 数据库服务器查询成功后返回数据信息，如果失败，返回失败原因。
业务规则	客户端发送查询命令时，携带数据标识字段作为查找关键字。

3.当数据不再需要存储的时候，用户可以向数据库系统发送删除数据的命令，数据库系统将从底层把数据记录删除掉，具体用例见表3-3。

4.修改数据是指当用户需要更加记录的一些属性的时候，向数据库发送修改请求，底层数据库把新的属性存储在数据库系统中，具体用例见表3-4。

表 3-3 删除数据用例

用例名称	删除数据
用例描述	管理员或者用户通过客户端发起数据删除的请求
参与者	分布式数据库管理员和用户
前置条件	1、 分布式数据库系统运行正常。 2、 数据信息已经存在。 3、 管理员或者用户已成功登陆系统。
后置条件	分布式数据库系统能够正确处理客户端请求，删除该条数据。
基本操作	1、 运行客户端，登陆数据库服务器。 2、 在客户端上发送删除数据库的命令。 3、 在客户端上确认信息后，请求消息被发送到分布式数据库。 4、 数据库处理成功后返回删除成功响应，如果失败，返回失败原因。
业务规则	1、 客户端发送删除命令是，必须要指定关键字。 2、 有权限的用户才能删除数据。

表 3-4 修改数据用例

用例名称	修改数据
用例描述	管理员或者用户通过客户端发起数据修改请求
参与者	分布式数据库管理员和用户
前置条件	1、 分布式数据库系统运行正常。 2、 数据信息已经存在。 3、 管理员或者用户已成功登陆系统。
后置条件	分布式数据库系统能够处理客户端请求，正确修改该条数据发生变化的部分。
基本操作	1、 运行客户端，登陆分布式数据库服务器。 2、 在客户端上发送修改数据的命令。 3、 在客户端上确认信息后，请求消息被发送到分布式数据库。 4、 数据库查询成功后返回修改成功响应，如果失败，返回失败原因。
业务规则	1、 客户端填写数据信息时，必须要指定关键字。 2、 不能修改数据标识，其余修改的值在有效范围内。 3、有权限的用户才能修改数据。

3.2 技术和框架分析

3.2.1 系统实现语言选择

本数据库系统选择了JAVA语言和KOTLIN语言作为开发语言，之所以这样选择，主要基于以下几个原因：

1.JAVA是跨平台的开发语言，用JAVA开发实现数据库系统，可实现所有主流平台的数据库部署，方便数据库系统在不同系统间迁移。

2.JAVA是企业开发的首选语言，其安全性比本地语言更高。用C等本地语言开发系统，虽然性能比较好，但是很容易出现内存泄漏，容易被攻击。作为企业开发语言，JAVA开发的软件更加容易维护，不容易出错误，也就更安全。

3.本系统实现相关的非关系型数据库引擎OrienDB和Elasticsearch也是用JAVA开发的，为了方便集成它们。本系统也应该使用同一种开发语言。

4.JAVA易于开发和调试，作为学生时期个人开发的数据库系统，一个人的精力是有限的，如何选择高效的开发工具和语言对我们来说非常重要。

3.2.2 网络实现技术分析

所有的服务器软件都需要实现网络模块，这样才能接受客户端的连接请求。JSQL用java语言开发，主要用到的是java语言的网络开发技术。

在JAVA网络开发技术中，网络IO的方式分为同步阻塞、同步非阻塞和异步非阻塞方式。

在网络编程里面，同步阻塞是最简单的编程模型。在这个模型中，网络中的服务器端和客户端在收到对面返回结果时都需要等待对方，在等待的时候不能干其他时候。同步就是服务器和客户端的操作序列同步，在前一个步骤没有完成的时候，后面的步骤必须等待，阻塞就是在系统等待的时候，系统阻塞运行，不能运行其他代码。这样对系统的资源是一个浪费。虽然在操作系统中，可以通过多线程来避免大部分问题，但是建立线程是需要代价的，而且操作系统的线程数量是有限的，不能支持成千上万的客户端服务器连接。

NIO编程模型[34]就是为了解决上面同步阻塞编程模式的缺点而发明的，在NIO编程模型中，服务器的一个线程可以处理多个客户端连接事件，在等待客户端返回请求的时候，服务器可以接受来自其他客户端的连接请求。这样一个线程就可以维护多个客户端的连接，增加系统的并发性能。而同步阻塞要完成相同的功能，服务器就必须要使用多线程，而多线程是有开销的。所以NIO提高了高并发服务器的性能。

Netty是用JAVA语言开发的网络框架，其主要目的是建立基于NIO编程模型的高性能协议服务器，用Netty开发网络服务器比直接用Java语言开发更加简单，性能也更高。 Netty的线程模型如如4-7所示。 Netty是一个非阻塞框架。与阻塞IO相

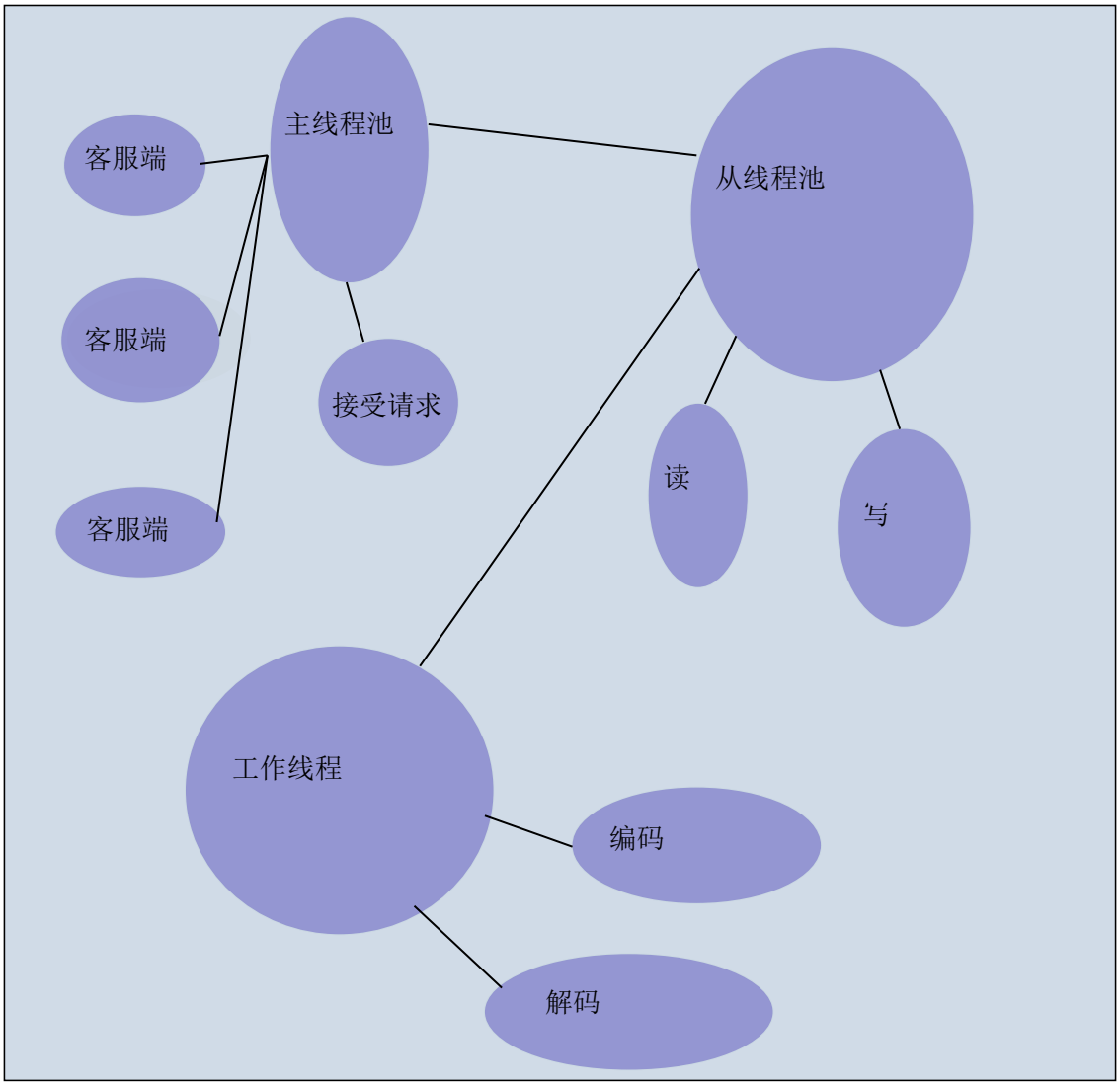


图 3-2 Netty

比，这可以大大提高系统的吞吐量和并发性能。 Netty使用事件驱动的应用程序范例，在Netty中，网络数据包的处理流水线是一系列事件处理程序，开发人员通过编写自己的事件处理程序就能自己开发网络协议。因为Netty的这些特效，利用它可以实现一个高性能的网络应用程序，所以本系统的网络模块也是用的Netty来实现的，它支持高并发的客户端访问。

3.2.3 通信协议分析

每个服务器和客户端通信都要实现自己的通信协议，在本论文所述分布式系

统中，结构化存储需要用到SQL操作需要，就需要实现关系型数据库网络协议，考虑到Mysql使用的广泛性，Jsql采用Mysql的通信协议。这样就能支持更多的遗留客户端软件。在MySQL数据库通信过程中。一共有两个阶段，第一个阶段是在客户端连接服务器的时候，服务器对客户端的身份和权限进行检查，检查成功以后连接才会进行，服务器才会让客户端进行下一个阶段。mysql通信的基本单位是应用程序包。多个指令可以合成一个包；答复可以包含多个包。 MySQL客户端与服务器的交互过程主要有两个阶段：握手认证阶段和命令执行阶段：

1.握手认证阶段

在客户端通过网络接口连接服务器的时候，握手阶段就开始了，其交互过程如下：

(a) 服务器发送给客户端握手初始化报文。

(b) 客户端回复服务器端登陆认证报文。

(c) 服务器发送给客户端认证结果报文。

2.命令执行阶段

在握手阶段，服务器对客户端进行身份和权限检查，所有检查通过以后就会进入命令阶段，命令阶段客户端服务器交互过程如下：

(a) 客户端发送服务器执行命令报文。

(b) 服务器发送给客户端命令执行结果。

MySQL客户端和服务器之间在网络通信中完整的交互过程如图4-8所示。

通过实现mysql通信协议，分布式数据库系统使用者可以用mysql的所有客户端连接本系统，存储结构化数据。在实现mysql协议的同时，也需要对二进制协议进行接口定义，以便用户可以存储其他数据类型的数据。系统选择实现mysql协议，是因为mysql是最广泛使用的免费的数据库系统，其用户非常之多，实现mysql协议以后，mysql的所有工具和客户端都可以直接连接JSQL分布式数据库系统。

3.2.4 SQL实现分析

SQL是一种通用的数据库操纵语言，数据库和应用程序用户将数据库操作请求发送到数据库服务器，只需要说明操作目的和结果，不需要说明具体操作的执行过程。同样的语句，不同的数据库服务器会选择合适自己存储引擎的执行过程。这样每个数据库服务器就能实现不同的操作过程，实现不同的性能。

在关系数据库系统中，SQL是通用的操作语言，用户和数据库系统通过SQL语句交流数据和发送命令请求。用户发送SQL命令，然后数据库系统解析命令，返回命令结构。虽然所有的关系数据库系统都支持SQL语句，但是不同的数据库系

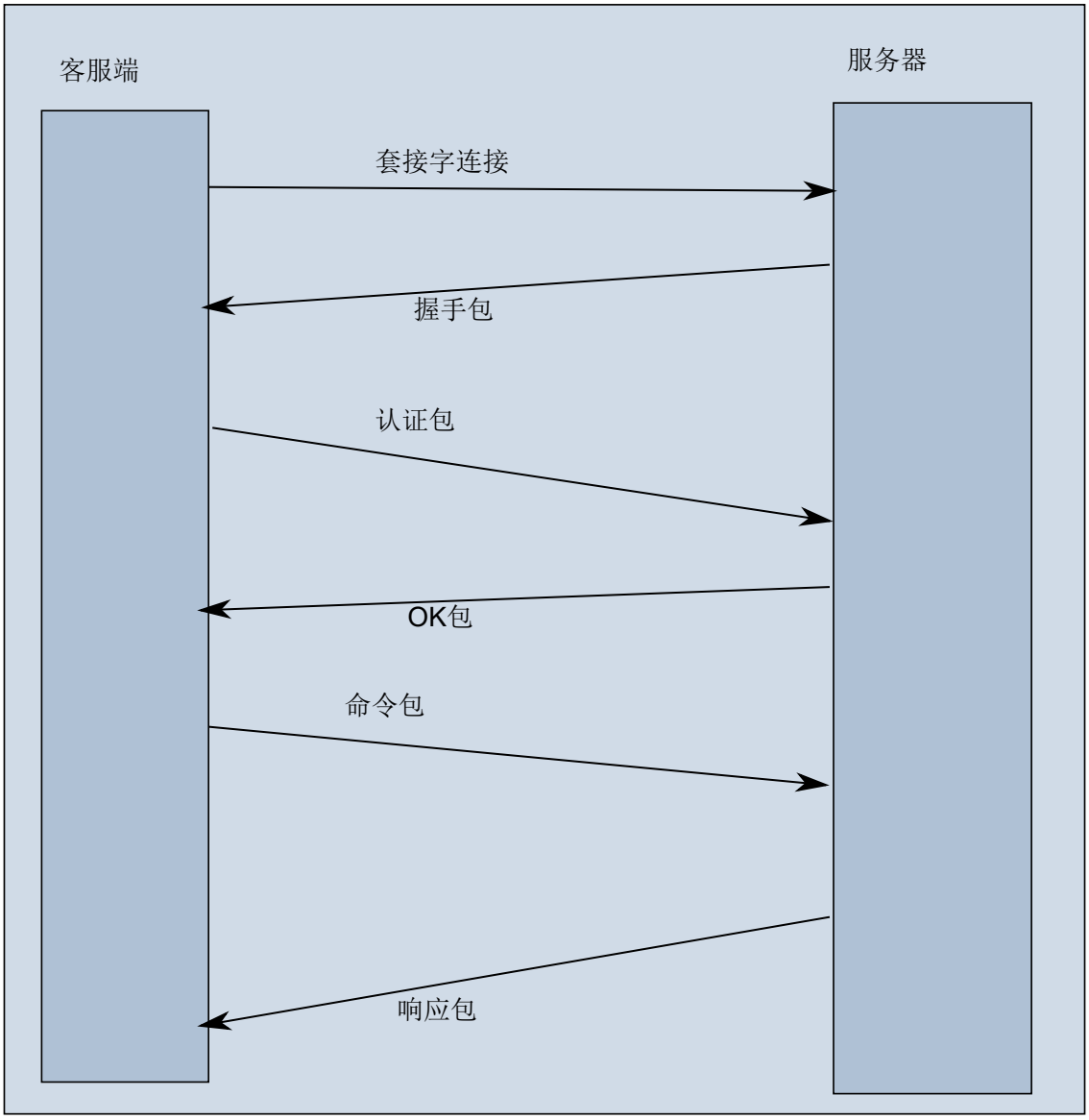


图 3-3 协议交互流程图

统支持的语句类型不一定相同。本论文所述的分布式系统选择实现和mysql一样的sql语句。

Mysql支持的sql语句和标准的sql语句不全一样，具体的，它支持下面这几种语句类型：

1.数据定义语句

2.数据操作语句

3.交易和锁定声明

4.复制语句

5.准备的SQL语句语法

6.复合语句语法

7.数据库管理语句

8.效用声明

其中每种语句类型又分为很多种sql语句，所以mysql支持的sql语句非常的多，因为本系统选择了兼容mysql的协议，所以我们也要解析这些不同的sql语句，本系统利用Druid框架实现了对SQL语句的解析功能。其对客户端的命令进行词法分析后语法分析以后，生成预定的JAVA对象格式，这样我们就能对其进一步的处理。当然，并不是所有的语句都能这样检查的处理，还有其他Druid不支持的语句，我们只能自己对其进行解析。在对sql语句进行语法解析以后，接下来就需要进行语义分析。结合分布式数据库系统元数据，对sql语句进行合法性检查。查找适合的数据库存储接口调用。返回客户端结果。

3.2.5 存储引擎分析

在任何数据库系统中，存储引擎都是最重要最难实现的部分，存储引擎设计的好坏直接影响系统的整体性能。存储引擎决定数据在底层的存储格式，解决数据的并发性和一致性问题。每种存储引擎底层都基于一种数据结构。比如常用的哈希表结构和B+树结构。本系统所用的OrientDB存储引擎底层就是用的B+树结构。利用非关系数据库OrientDB实现可靠的数据存储，结合非关系型数据库的优点，使得数据库系统更加容易实现集群，更加容易扩展。

和传统关系型数据库系统不同的是，在orientdb存储引擎中，关系是直接存储在每个记录的对象里面的。而在关系数据库系统中，数据之间的关系是通过外键关联的。这样在获取一个记录相关联的时候，就必须去查找另外的一个表记录。这样就会对磁盘进来查找。最严重的就是顺序扫描，其性能非常低下，这也就是为什么关系型数据库在外键很多的时候，性能严重下降的原因。而在orientdb数据

库引擎中，要查找一个记录对应的记录，只需要按照记录中记录的地址去查找就可以了。关系直接存储在记录中，不需要再去查找外键。对提高性能非常重要。

3.2.6 分布式架构分析

数据库系统的分布式架构决定其扩展性，在传统的关系型数据库系统中，要扩展数据库系统的性能，一般通过复制主从架构，其能扩展读取性能，但是不能提高系统的更新性能。虽然现在很多NOSQL数据库都实现了分布式，但是本系统还是利用Hazelcast 自己开发了分布式功能，论文所述分布式数据库系统实现的是多主分布式架构，每个集群节点都是主节点，都可以接受更新请求，这样就能扩展更新能力。

Hazelcast是一个用JAVA语言开发的开源的内存数据网格框架，提供了一些在分布式系统中非常重要的基础功能，包括：

1.集群节点发现和选举

2.分布式数据结构

3.分布式计算

4.分布式查询

5.聚类

6.高速缓存

7.多语言绑定

8.轻松嵌入到Java应用程序中

这些功能使Hazelcast成为应用程序中的多用途工具。它可以用来作为非关系型数据库系统，能存储多种结果的数据，还可以用来解决分布式应用程序中的多个问题。Hazelcast从一开始就被设计为一个分布式内存网格框架，解决了主选举、网络弹性以及最终一致性等许多潜在的难题。在Hazelcast框架中，提供了很多分布式的数据结构，利用这些分布式数据结构可以开发分布式数据库系统，这个框架用JAVA语言开发，能在多个平台部署，利用Hazelcast可以轻松开发出稳定可靠的分布式集群功能。

3.2.7 监控审计分析

监控功能主要在分布式管理节点部署，管理员通过连接分布式管理节点，就能对分布式数据库进行监控。对数据进行操作。本系统实现最简单的监控模块，一个监控模块首先要存储所有的日志数据，而且这个数据不能随意的更改，所以我改了Elasticsearch的源代码，让它来存储所有的sql更新日志信息，其中的日志信

息不能被篡改和删除，同时对其进行分析和统计，然后用可视化的框架来显示出结果。之所以用这个框架，主要是因为它是一种搜索引擎，能够非常快速的检束出我们需要的各种信息，这对于信息审计来说非常的重要。利用这个框架我们实现了实时的安全审计功能，提高了数据库的安全性。

3.3 本章小结

在这一章中，对所述分布式数据库系统进行了需求分析和技术分析。需求分析是指分析系统的功能和性能要求，本论文所述分布式数据库系统作为一个数据库系统，其具有存储和操纵数据的功能。在对系统进行需求分析以后，论文分析了系统每个功能模块所需要的技术，每个模块的实现都要用到不同的技术，对每种技术进行了分析。

第四章 系统设计

在软件开发流程中，需求分析之后就是设计阶段，设计分为总体概要设计和详细设计。首先，开发者需要对软件系统进行总体概要设计，即总体概要和框架设计。总体概要设计就是设计系统的总体功能，框架设计就是在系统功能的基础上，设计系统的模块，为软件的详细设计提供基础。在概要设计以后就要对系统的每个模块进行详细设计，详细设计设计每个模块功能实现流程，对每个模块需要的技术和算法进行设计。详细阶段需要对每个模块进行详细的设计，才能根据设计编写代码。只有经过仔细的系统设计，在编码阶段才能提前减少不必要的错误。

4.1 系统总体设计

4.1.1 系统架构设计

在系统设计中，先进行原型化[35]设计可以对系统整体功能进行大概的说明，有利于对整体的功能把握。原型化设计就是先构建一个系统的小版本，通常只包括整体功能中有限的功能，突出整体中的关键功能，其可用于：

1.帮助用户和客户标识系统的关键功能需求，忽视具体的细节，从而利于把握整体。

2.证明设计或方法的可行性，方便对系统做进一步的总体设计。

通常，原型化设计方法是一个迭代的过程：首先构建一个原型，然后对这个原型进行评估，考虑如何对原型设计进行改进，之后再构建另外一个原型，如此循环迭代。当客户认为原型解决方案满意时，迭代过程就终止了，可以进一步完成系统的设计工作。由于本文所涉及的分布式数据库系统功能非常大，结构非常，代码工作量大，因此采用了原型化模型的软件开发方式开发一个数据库原型，然后后期逐渐对这个原型进行改进。作为一个分布式数据库系统，系统要实现的首要功能就是数据库的增加、查询、修改、删除等功能，为了对系统进行扩展，论文设计了分布式数据库系统的分布式架构。当JSQL部署在分布式网络上的时候，系统的网络拓扑结构如图4-1所示。 在系统网络拓扑图中，主要有下面几种节点：

1.客户端节点

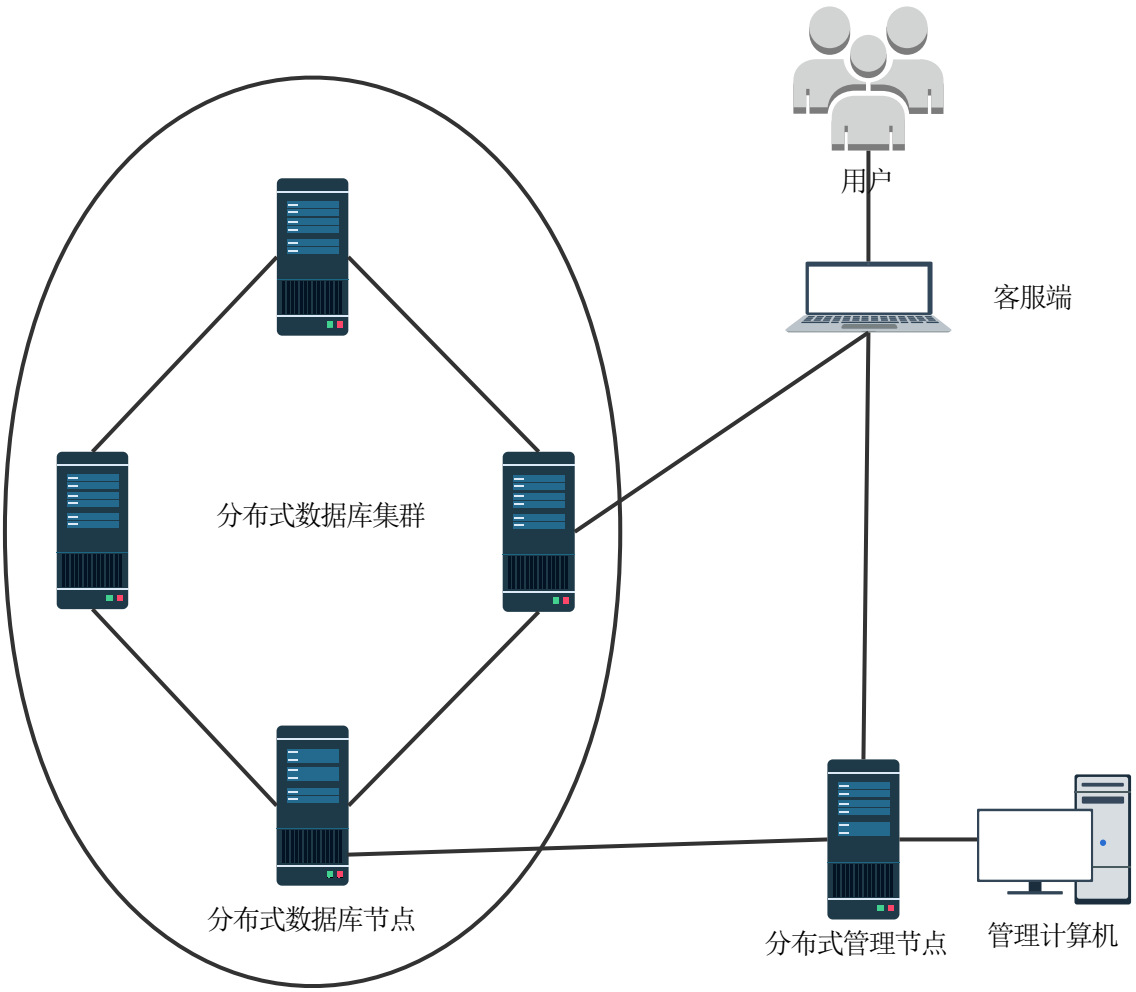


图 4-1 系统网络拓扑结构图

客户端节点作为连接分布式数据库服务器的代理，发送SQL命令请求到数据库服务器，请求数据，客户端节点可以是原生的Mysql节点，也可以是自己开发的具有负载均衡的JDBC[?]客户端。如果是用Mysql客户端，就可以直接连接分布式数据库节点，可以任意选择一台分布式数据库节点执行数据库操作命令，其中数据库会自动同步到其他数据库节点。当用自己开发的JDBC客户端的时候，客户端就会首先连接分布式管理节点，得到具体的分布式数据库节点后，才能连接数据库节点执行具体的数据库命令。

2.分布式管理节点

分布式管理节点管理分布式数据库节点的元数据和实现负载均衡算法。管理计算机通过连接分布式管理节点，可以对分布式节点进行管理，对系统进行监控。这样当客户端请求到来的适合，分布式管理节点就可以返回适合的分布式数据库集群节点，从而达到均衡集群服务器资源的效果。

3.分布式数据库节点

分布式节点存储具体的数据，数据库系统功能主要在这个节点上面执行，响应客户端的增删改查的命令请求。当客户端连接分布式数据库节点以后，会检查客户端的权限，只有有权限的用户才能执行命令。在JSQL中，每个分布式数据库节点的功能都是独立的，都可以独立运行，管理本地存储的数据。

4.管理计算机

管理员通过管理计算机管理分布式数据库几点，也可以通过管理计算机来查看审计信息，设置监控报警功能。

在系统网络拓扑结构图中，客户端是数据库系统用户发送命令的终端设备，各个分布式数据库节点和分布式管理节点构成了系统的服务器端。其中，分布式管理节点作为中心节点接收所有的控制消息，通过分布式管理节点才能找到分布式数据库节点的IP地址。

在图4-1中，客户端是用户向分布式数据库系统发送数据操纵命令的软件。分布式管理节点实现分布式负载均衡算法，把客户端的请求均衡的分配到每个分布式数据库节点上面去，分布式数据库节点则是存储数据的服务器节点。客户端通过分布式管理节点得到分布式数据库节点的IP网络地址，得到地址以后，客户端再连接数据库节点，得到具体的数据，分布式管理节点通过选举产生，也就是第一次启动的服务器节点，同时也能存储数据库数据。本论文所述分布式数据库系统JSQL在分布式环境下具有两种节点，客户端节点和分布式数据库集群节点。客户端节点是用来连接分布式集群的，在系统中，可以用mysql节点来连接分布式数据库系统。在分布式数据库集群中，根据功能可以分为分布式管理节点和分布

式数据库节点，在一个分布式系统中，只有一个分布式管理节点，其是通过选举产生的。当一个分布式管理节点失败的时候，通过选举选择其他的分布式数据节点作为管理节点，剩下的节点作为分布式数据库节点，用来存储具体的数据。在JSQL中，每个分布式管理节点都是独立的，可以自己管理其本地局部数据，通过和其他节点连接，可以动态扩展系统整体的读写性能。。本系统的架构图如图4-2所示。 本系统中，客户端是用户发送命令的终端设备。分布式管理节点收到客

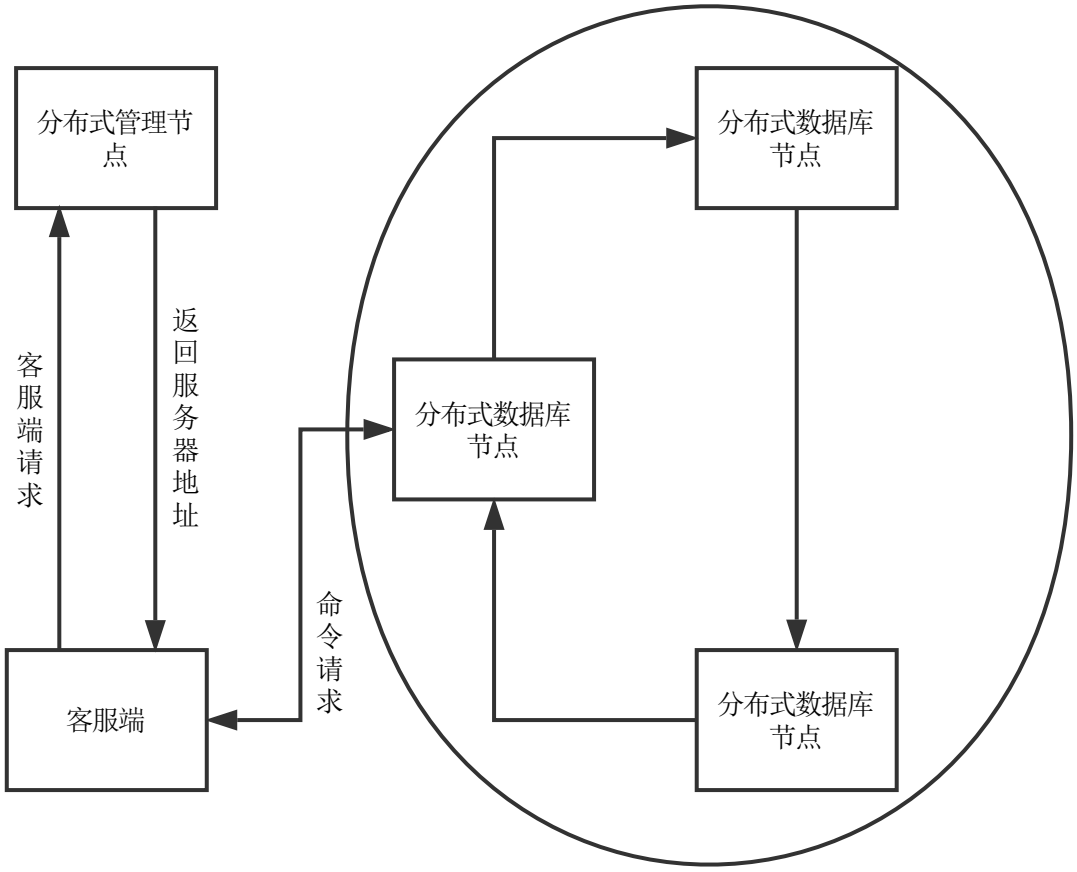


图 4-2 系统总体架构图

户端请求以后，返回给客户端数据库节点的地址，然后客户端再连接数据库节点，分布式数据库节点负责对客户端的请求消息进行处理，并将处理后的响应消息返回给客户端。在系统中，数据因为在不同的计算机上面都有备份，所以不存在丢失的危险。

4.1.2 系统功能设计

在需求分析和架构设计的基础之上，按照系统的节点对其进行功能划分，解决每个节点具有什么样的功能，每个功能由哪些模块组成。系统可以分为客户端和服务器两个功能节点，它们的主要功能模块如图4-3所示。

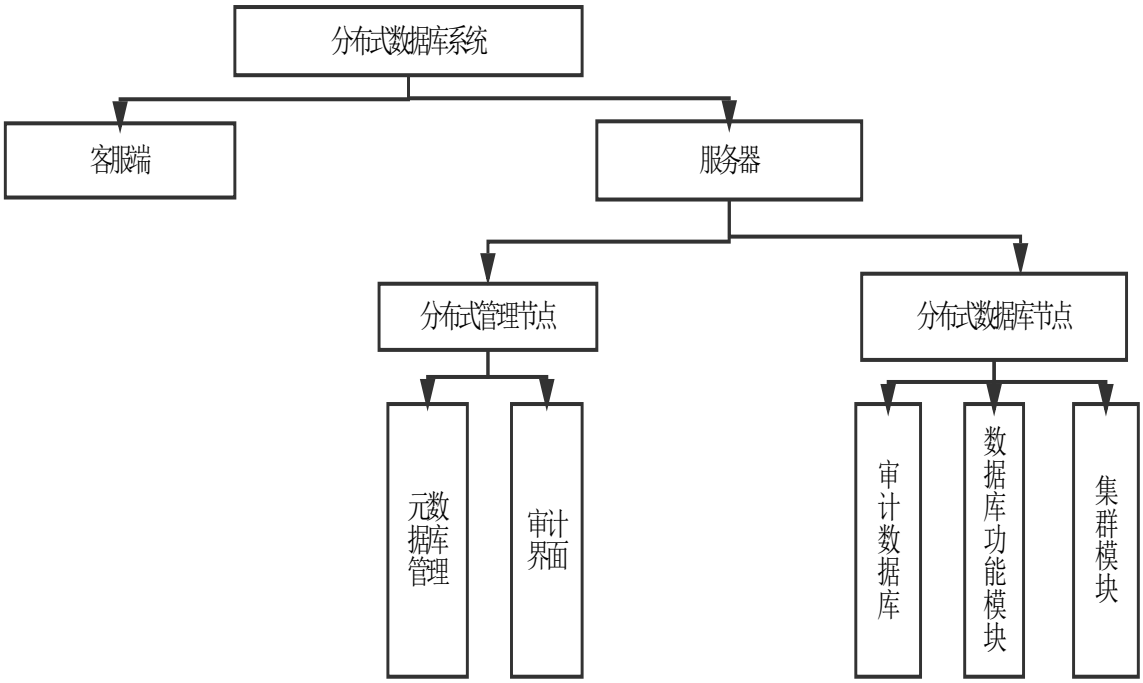


图 4-3 系统模块划分

在图4-3中，分布式管理节点有元数据管理模块和审计界面功能模块这两个功能模块。以下是对它们的更详细的介绍：

1.元数据管理模块管理服务器的所有元数据，包括每个数据库节点的 IP地址和端口号。分布式管理节点利用这些元数据来完成分布式的负载均衡算法，为客户端选择合适的分布式数据库节点。

2.审计界面管理模块对管理员使用，它可以让管理员管理所有的数据节点。审计功能除了审计界面以外，还包括分布式数据库节点的审计数据库，而审计数据库本身存储在系统内只有结合了分布式数据库的审计模块和分布式管理节点的审计模块，管理员才能对分布式系统进行审计管理。

分布式数据库节点由审计数据库模块、数据库功能模块、集群功能模块组成，下面分别对 3 个模块的功能进行介绍。

1.审计数据库模块用来收集所有的日志信息，作为审计使用。所有的数据库节点在接受到数据更改请求以后，就会把这些日志信息发送到审计数据库。然后分布式管理节点就能利用这个数据为管理员提供审计功能。

2.数据库功能模块完成具体的数据库功能。为用户提供操作接口，数据库功能模块可以分为网络模块，协议模块，解析模块和存储模块。在收到用户的请求命令以后。数据库功能模块就会对用户返回正确的信息。

3.集群功能实现每个数据库节点的通信。本系统利用Hazelcast实现了数据库的集群。每个分布式数据库节点的数据都能自动同步。同时，利用分布式多版本并发控制技术解决了数据的一致性问题。

4.2 客户端模块设计

分布式数据库系统应该能够为客户端选择合适的网络存取路线，选择合适的分布式集群节点，在节点发生变化的时候，还能动态的调整分配和连接策略。本文采用的分布式管理节点较好地满足了这两点。分布式管理节点存储所有数据库节点的元数据信息，实现了分布式负载均衡算法。在分布式数据库系统中，负载均衡的功能是非常重要的，每个节点的性能和存储容量都不一样，选择的负载均衡算法必须要考虑到这些节点的差异，均衡的为客户端选择最合适的集群节点，这样才能利于系统整体性能的提高。分布式管理节点实现了负载均衡功能，所以当我们想要使用负载均衡的时候我们必须要先连接管理节点，再连接数据库节点。客户端连接的流程图如图4-4所示。 每个流程解释如下：

1.首先客户端连接到分布式管理节点，管理节点返回给客户端正确的数据库节点的地址。

2.然后客户端连接到正确的数据库节点。

3.最后数据库节点返回给客户端具体的数据。

当然我们也可以直接连接分布式数据库节点，其中的数据会自动同步到其他的分布式数据库节点，这样可以直接用Mysql的客户端，减少用户的使用成本。使用Mysql客户端，可以执行几乎所有的Mysql命令。当存储结构化数据的时候，在用户看来，分布式数据库节点就像一个Mysql数据库服务器一样。只是本系统能自动同步每个分布式数据库节点的数据。

4.3 分布式管理模块

要实现负载均衡和其他管理功能，只实现客户端是不够的。还需要一个分布式管理节点。分布式管理模块存储所有数据库节点的元数据和实现负载均衡算法，当启动分布式管理节点的时候，它会去查找所有的数据库节点，然后存储到数据库里面保存。分布式管理节点的启动流程图如图4-5所示。

分布式管理节点和分布式数据库节点一样，都是服务器端的节点，都实现了同一个分布式功能。他们之间可以通过多播通信来交换信息。分布式管理节点得到这些信息以后。就能实现正确的负载均衡算法来选择合适的分布式数据库节点。

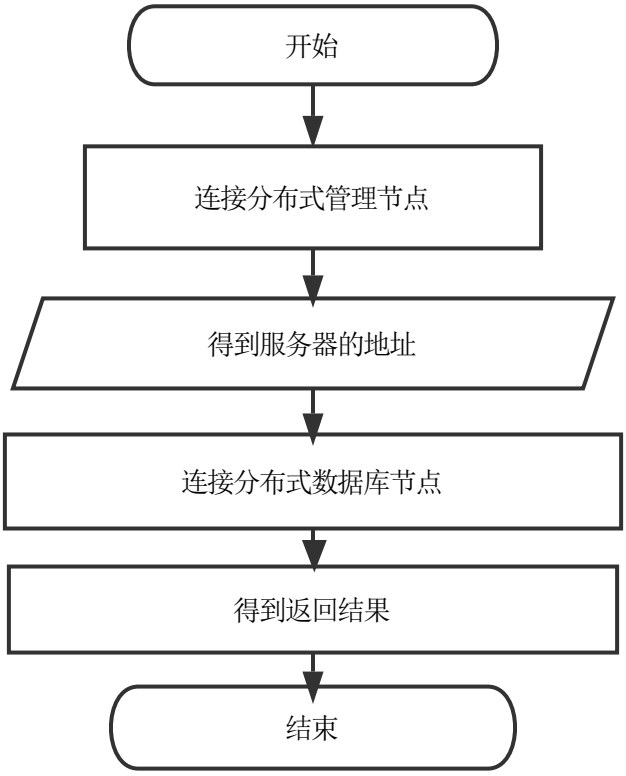


图 4-4 客户端连接流程图

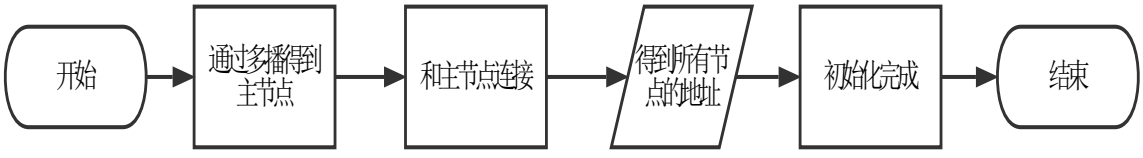


图 4-5 分布式管理节点初始化过程

分布式管理节点初始化成功以后，客户端就可以通过自己开发的基于JDBC的客户端来连接后端的分布式数据库节点，发送数据查询命令。

4.4 数据库功能模块详细设计

作为一个分布式数据库系统，其首要的功能就是数据库功能，也就是能完成对数据的存储和操纵。分布式数据库系统接收到客户端的命令，然后解析命令，为客户端完成数据操纵，最后返回结果给客户端。在数据库功能中，数据的增加、删除。修改和查询是其最主要的功能。下面设计出每个数据库功能的操作处理流程。

4.4.1 数据库功能操作流程设计

4.4.1.1 新增数据过程

第一步，客户端首先连接分布式管理节点，由分布式管理节点返回客户端的连接会话信息，客户端得到服务器的地址。第二步：客户端连接具体的数据库节点，发送新增数据请求，请求发送到具体的数据库节点。数据库节点经过网络模块和通信模块，得到请求信息以后，调用数据库引擎的接口，得到具体的数据，返回给客服。

4.4.1.2 数据查询和修改过程

第一步：客户端首先连接分布式管理节点，由分布式管理节点返回客户端的连接会话信息，客户端得到服务器的地址。第二步：客户端连接具体的数据库节点，发送新增数据请求，请求发送到具体的数据库节点。数据库节点经过网络模块和通信模块，得到请求信息以后，调用数据库引擎的接口，得到具体的数据，返回给客户端。

4.4.1.3 数据删除过程

第一步：客户端首先连接分布式管理节点，由分布式管理节点返回客户端的连接会话信息，客户端得到服务器的地址。第二步：客户端连接具体的数据库节点，发送新增数据请求，请求发送到具体的数据库节点。数据库节点经过网络模块和通信模块，得到请求信息以后，调用数据库引擎的接口，删除具体的数据，返回给客服操作成功消息。

数据库模块是本系统的主要开发模块，其详细模块图如图4-6所示。 下面对每个功能模块进行详细的说明。

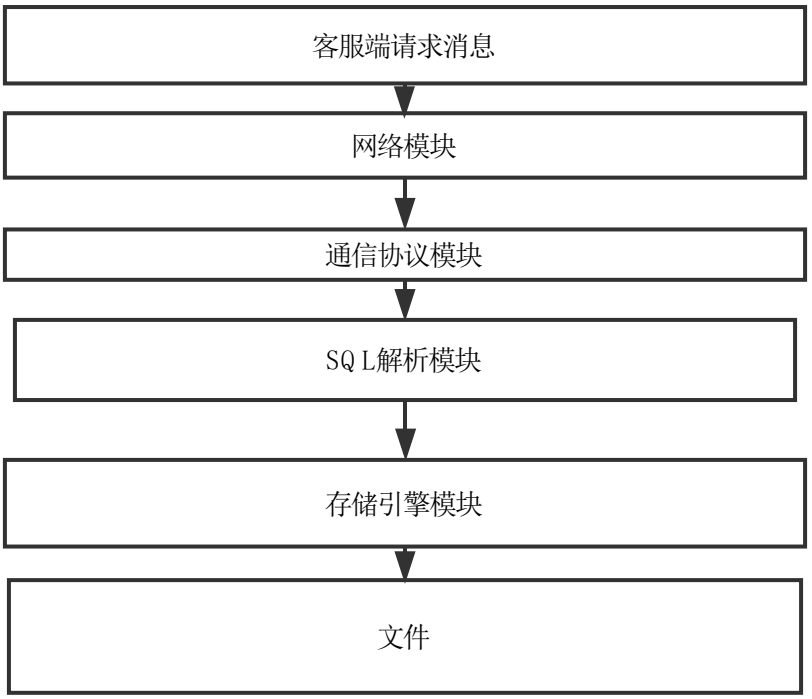


图 4-6 数据库功能模块图

4.4.2 网络模块设计

所有的服务软件都需要实现网络模块，这样才能连接客户端的请求。JSQL用java语言开发，主要用到的是java的网络开发模块。但是直接用java语言开发网络功能非常的麻烦，所以本文基于Netty[36]来开发直接的网络模块。具体的网络实现，包括线程池的规划都按照Netty的建议来开发。Netty的线程模型如图4-7所示。 根据图4-7所示，netty主要有三种线程：

1.主线程池主要接受客户端的网络连接请求，当主线程接受到客户端的请求以后，就可以把客户端的请求交个从线程池来处理。这样主线程就可以接受更多的其他客户端的连接请求。使得服务器的吞吐量大大的提高。

2.从线程从主线程得到客户端的连接以后，执行具体的读取操作。如果需要执行其他的事务逻辑，比如说读取磁盘，那么就需要把这个请求发送给工作线程来处理，这样客户端才不会因为从线程的忙碌而堵塞。

3.工作线程主要用来执行时间消耗比较大的任何，比如任何和磁盘文件有关的操作都应该放在工作线程来处理。

通过使用合适的线程模型，服务器具有很高的吞吐量，能接受成千上万的客户端连接。在设计系统的网络功能的时候，通过选择合适的框架和线程模型，设计出高性能的网络功能模块。

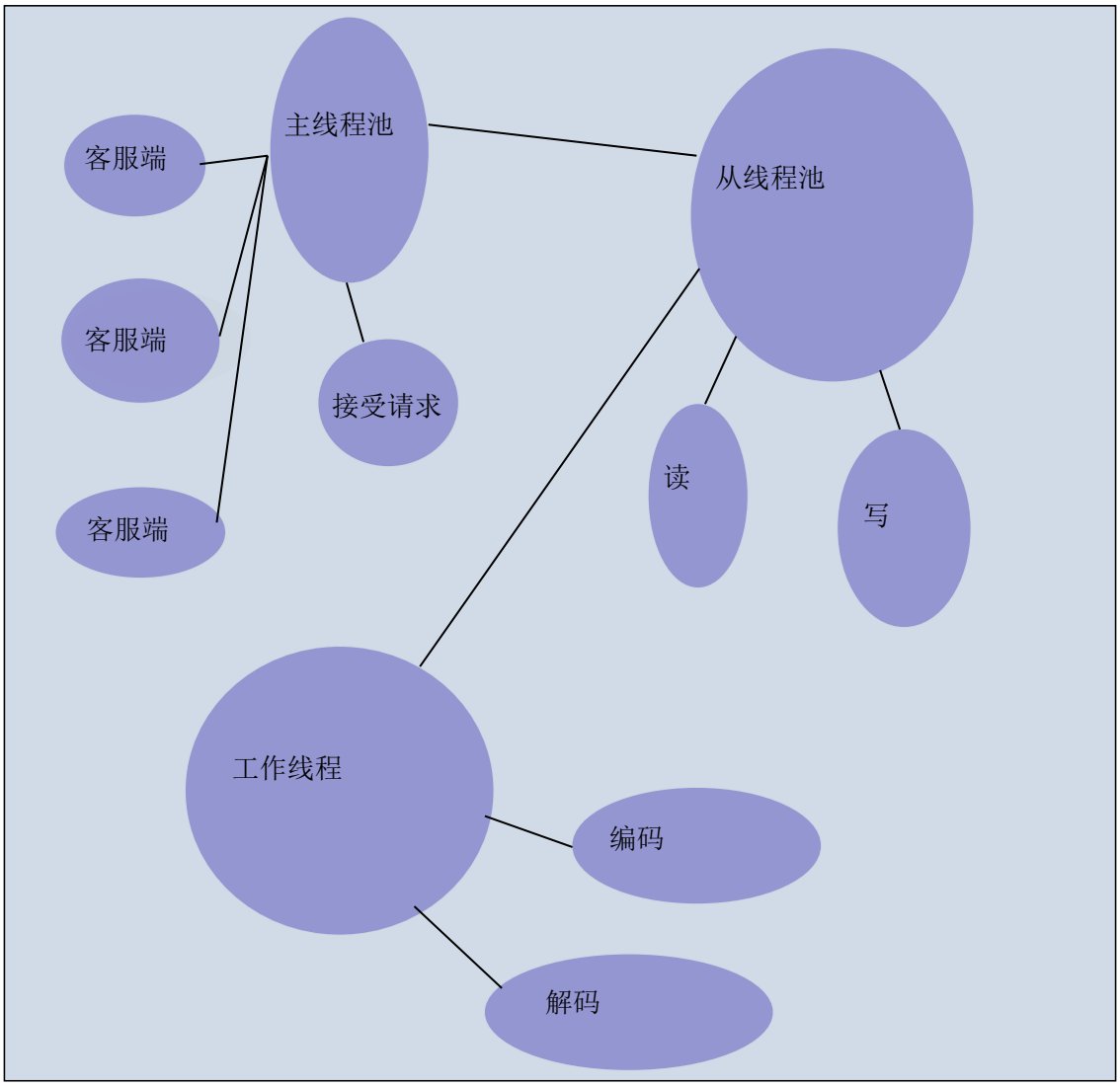


图 4-7 Netty线程模型

4.4.3 通信协议设计

每个服务器和客户端通信都要实现自己的通信协议，考虑到mysql使用的广泛性。jsql采用mysql的通信协议[37]。这样就不用自己开发协议了。在mysql服务器客户端交互生命过程中，主要有握手阶段和命令执行阶段。当客户端通过网络接口连接服务器的时候，握手阶段就开始了，在服务器对客户端进行身份和权限检查以后，双方就会进入命令阶段。在mysql客户端和服务器端的交互生命过程中，一共有握手和命令执行两个阶段。握手节点开始客户端的认证和双方的能力交互，命令阶段就是客户端向服务器发送命令，服务器执行命令然后客户端结果的执行阶段。

1.握手认证阶段

当客户端通过网络连接服务器的时候就进入握手阶段，其交互过程如下：

(a) 服务器发送给客户端握手初始化报文。

(b) 客户端回复服务器端登陆认证报文。

(c) 服务器发送给客户端认证结果报文。

2.命令执行阶段

在服务器认证客户端成功后，双方会进入命令执行阶段，命令执行阶段的交互过程如下：

(a) 客户端发送服务器执行命令报文。

(b) 服务器发送给客户端命令执行结果。

MySQL客户端与服务器之间的完整交互过程如图4-8所示。

在mysql的客户端和服务器的交互过程中，都是通过发送网络数据包来通信。每个数据包分为数据包头部和具体数据两个部分，数据报文的头部说明数据包的大小和序列号信息，数据部分则是具体的数据。在每个数据库报文头中都有两个字节用于表明实际数据的长度。在报文头中还有一个关键字段就是序列号，在每次客户端服务器连接过程中，序列号就会被初始化为0。在每个数据包的后面就是具体的报文数据，其具体的内容和报文类型有关，可以报考请求的内容、服务器响应的数据和其他的服务器操纵命令，这部分的长度由每个数据包的头部指定。

4.4.4 SQL引擎设计

SQL引擎模块就是对客户端发送过来的SQL语句进行语法和语意解析，理解客户端的操作目的。终于成本和性能的考虑，在设计SQL引擎功能的时候，论文选择基于成熟的开源框架Druid。SQL的解析流程如图4-9所示。 SQL引擎层从协议层解析到语句以后，就要对这个语句进行解析，才能交给下面的存储引擎处理

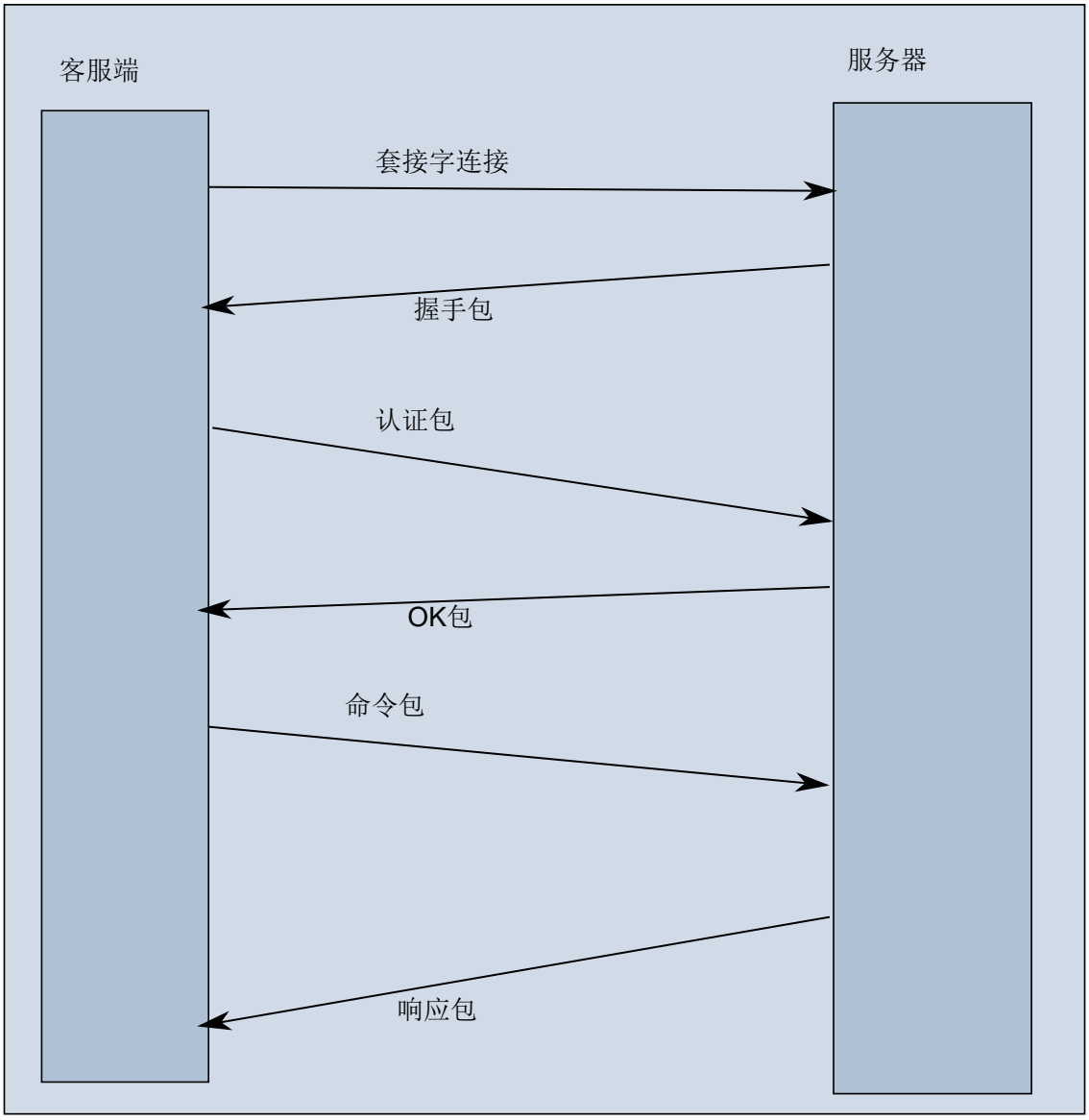


图 4-8 协议交互流程图

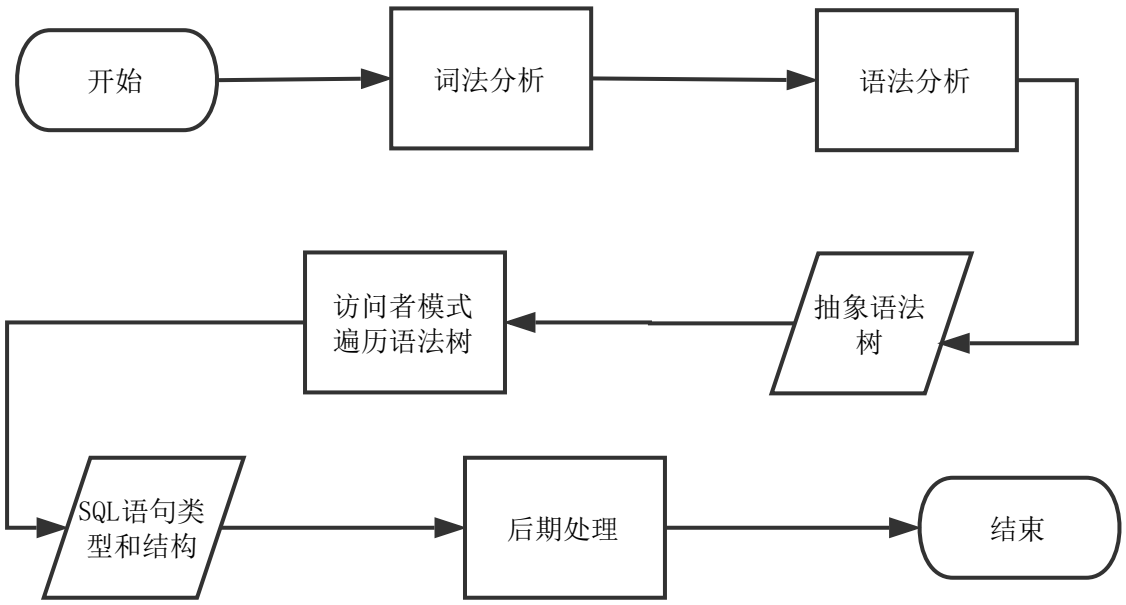


图 4-9 SQL解析流程图

具体的请求。解析模块首先对语句进行词法分析和语法分析，得到抽象语法树，然后用访问者模式去遍历抽象语法树，得到我们需要的数据结构。这个结构被封装成类，每种语句都有不同的类对应。得到语句类型以后，我们就可以对他进行具体的分析和处理。最后交个存储引擎层处理具体的数据操纵。

4.4.5 存储引擎设计

存储引擎是数据库系统最重要的功能部件，存储引擎的功能就是为数据选择合适的存储组织结构，使得其更加高效的操作数据，不同数据库系统都有其特有的存储引擎。每种存储引擎底层都基于一种数据结构。比如常用的哈希表结构和B+树结构。本系统采用了一种NoSQL数据库引擎[38]，这样可以结合非关系型数据库的优点。在OrientDB这个存储引擎基础之上，封装关系型的操作。完成关系型表和NoSQL数据库的转化。对上一层来说，我们提供了关系型存储引擎的接口调用。在存储引擎的底层，我们调用OrientDB的接口来完成具体的功能。这样能结合关系型操作的便利和非关系型数据库的优点。

4.5 集群架构详细设计

要实现集群功能，肯定要用到网络功能，还要考虑各种不可靠的因素。为了系统的稳定性和可靠性，本文用到了一个开源的分布式的开发框架hazelcast，基于hazelcast的应用程序的结构图如图4-10所示。 分布式数据库节点和分布式管理

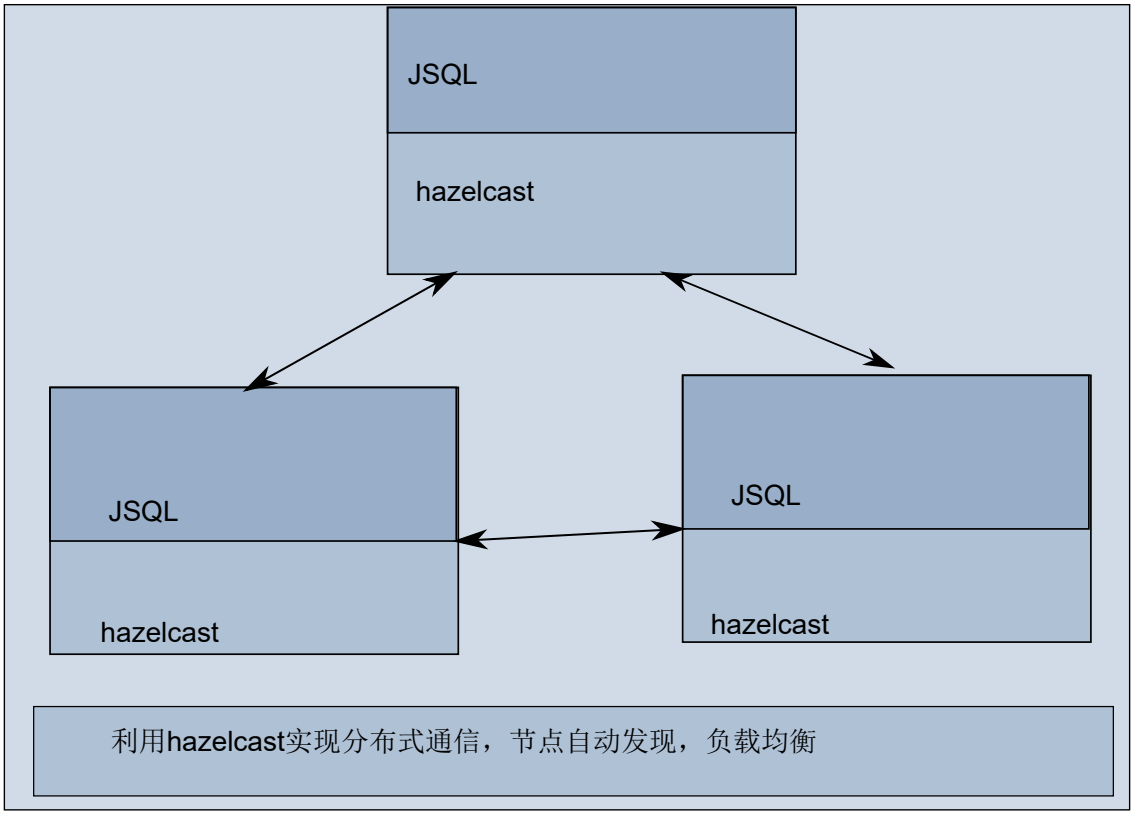

JSQL
hazelcast
JSQL
hazelcast
JSQL
hazelcast
利用hazelcast实现分布式通信，节点自动发现，负载均衡

图 4-10 集群架构图

节点都嵌入了Hazelcast模块，这样分布式管理节点就能很容易的得到每个数据库节点的详细信息，直接调用框架的接口就可以得到这些信息。利用hazelcast的分布式数据结构，论文设计了分布式系统的多主分布式架构，在后面的具体实现部分会给出详细说明。

hazelcast分布式原理如图4-11所示。 在分布式数据库集群中，不用设计主节

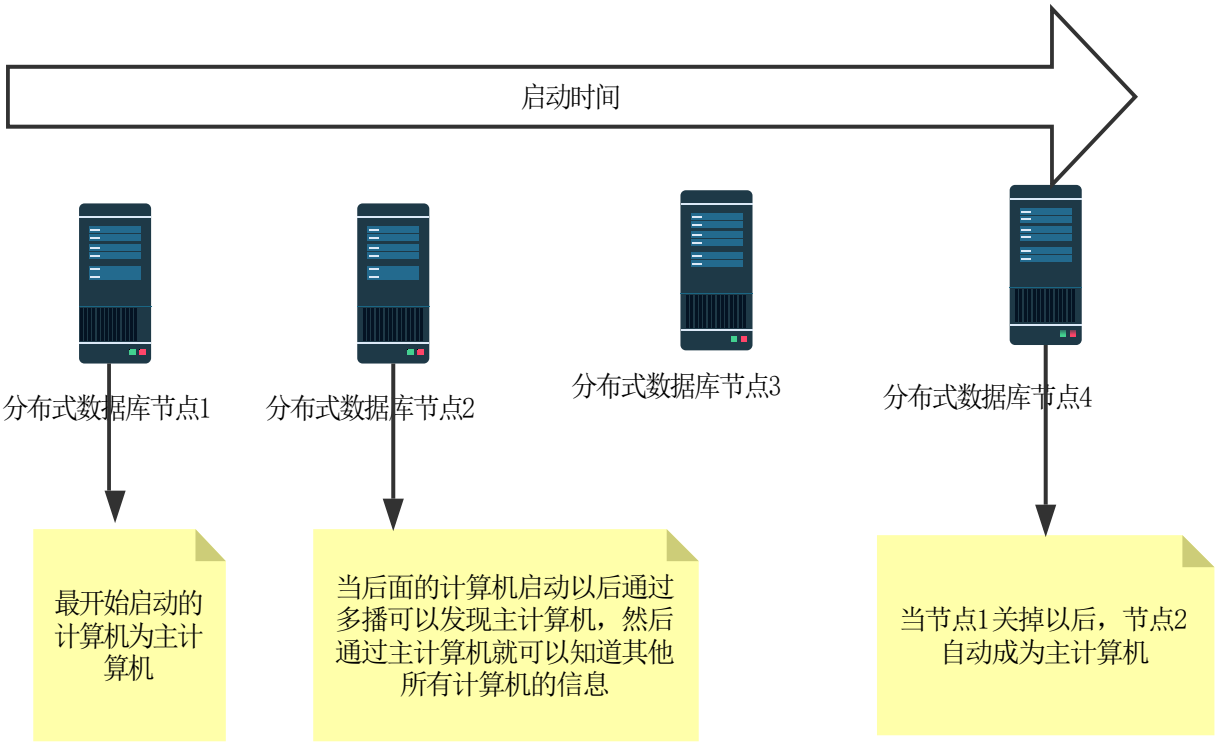


图 4-11 集群架构图

点，集群会自己发现和设置主节点。在一个集群中，最先启动的计算机就是主节点，主节点拥有其他所有节点的信息。当其他节点启动的时候，通过多播技术，加入多播网络。主节点发现以后，就会把自己的信息和集群所有其他计算机的信息发送给这个新的计算机，同时更新集群的元数据。当第一个主节点关掉的时候，再一次启动的计算机就自动成为主节点。这样就不需要用户频繁的设置和更新元数据。分布式数据库集群在增加和删除节点的时候，不需要任何人为的干预，集群能够动态增加分布式数据库集群节点。

4.6 审计模块详细设计

审计模块部署在管理计算机和分布式管理节点上面。管理员通过管理计算机连接分布式管理节点就可以管理所有的数据。监控所有数据的更改情况。图4-12显示了审计模块的实现层次结构图。从上大下一共可以分为下面4个模块：

1.grafana[39]可视化和报警模块，主要用到了开源的图形框架，来显示底层的审计数据，同时也可以让用户设定预警数据，这样以后就可以在一定的情况下向用户报警。

2.elasticsearch模块用来存储监控数据，监控数据为了登陆日志和sql执行日志我们需要更改它的源代码才能用来作为审计数据库，因为审计数据库不能随意的更改和删除，只能查询和增加。

3.JSQL接口层，主要是为上面和下面的各层提供连接功能，为上面层提供数据接口，为下面层提供显示接口，使得更加容易使用。

4.本地日志层，主要是在文件系统中存储所有需要的日志记录，这样可以随时和审计数据库的数据来对比。审计数据的安全和一致性。

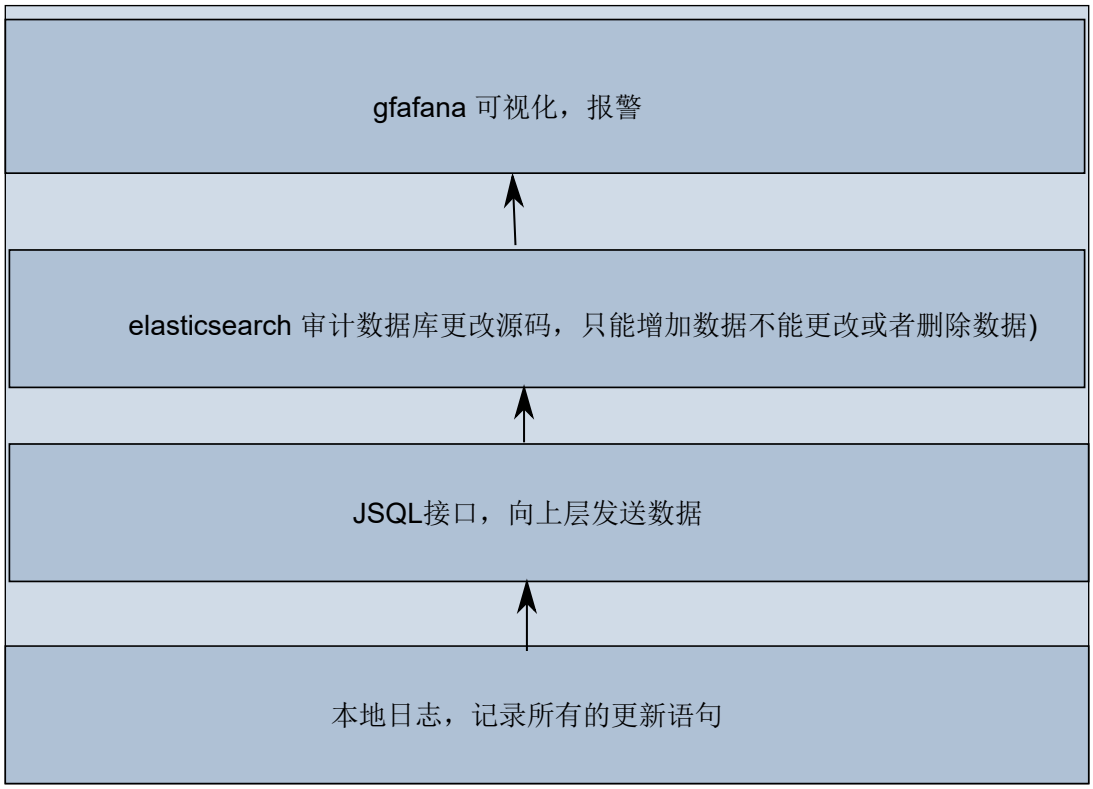


图 4-12 监控模块

4.7 本章小结

任何软件系统只有经过仔细的系统设计，在编码阶段才能提前减少不必要的错误。在本章分别对系统进行了总体设计和详细设计，在总体设计部分，首先对分布式数据库系统的部署架构进行说明，然后针对每个节点的功能进行模块划分。在系统的详细设计阶段，对总体设计阶段所划分的模块进行详细设计，包括客户

端模块、分布式管理模块、数据库模块、集群架构模块和审计模块。对系统进行详细设计方便系统的编码和测试。

第五章 系统实现

在对系统进行总体设计和详细设计以后，就是编码实现阶段。本章按照前一章的功能模块设计，给出其中关键功能模块的具体实现。详细分析关键功能实现流程图，对系统所用关键算法进行说明。

5.1 代码规范和总体结构

本数据库系统采用和JAVA语言类似的 Kotlin语言开发。JAVA中的代码以包为组织单位，这样组织代码的好处是逻辑结构分明。表5-1描述了本系统所有的包和每个包的详细功能。其中每个包实现具体的功能，这样分配代码在大型工程实践中很常见。除了逻辑清晰，方便管理以外，同时也方便团队协作。 在所有包里

表 5-1 本系统所有包和各个包的作用

包	作用
io.jsql	启动或者关闭服务器
io.jsql.audit	审计监控功能
io.jsql.cache	数据库缓存
io.jsql.config	配置功能，读取外部配置文件
io.jsql.hazelcast	集群功能实现
io.jsql.mysql	实现mysql通信协议
io.jsql.netty	Netty实现高性能网络前端
io.jsql.orientstorage	嵌入式存储引擎模块
io.jsql.shutdown	关闭数据库功能
io.jsql.springaop	面向切面，实现日志等功能
io.jsql.sql	实现SQL解析模块
io.jsql.storage	存储引擎接口
io.jsql.test	测试包
io.jsql.util	所有常用的工具类

面，config包没有实现具体的功能，这个包主要包括了系统常用的配置信息。

图5-2给出了config包下面每个类的具体作用，这个包主要是让用户可以配置数据库的各个方面，比如配置数据库的端口号，配置最大的连接数等等。为了方便对软件系统的代码进行管理，本论文所述系统全部实现代码都放在github上面进行保存和管理。 在本章的后面会给出其他关键功能模块的详细实现。

表 5-2 config包下面每个类的作用

类	作用
Capabilities	处理能力标识定义
ErrorCode	所有的错误代码
Fields	字段类型及标识定义
Isolations	事务隔离级别定义
Versions	本系统的版本号，通讯包

5.2 客户端功能关键技术实现

在分布式数据库系统中，客户端模块的功能主要是连接分布式数据库节点服务器，在分布式管理节点的配合下，为用户选择合适的后端分布式数据库集群节点，实现服务器资源的均衡利用。客户端在得到分布式数据库节点网络地址以后，就可以通过JDBC连接后端的分布式数据库节点，也可以通过其他软件接口存储文档模型的数据类型。

利用网络连接到分布式管理节点得到分布式数据库节点的IP地址。在分布式管理节点为客户端返回正确数据库地址以后，客户端就可以通过JDBC连接数据库节点。客户端连接功能流程图如图5-1所示。

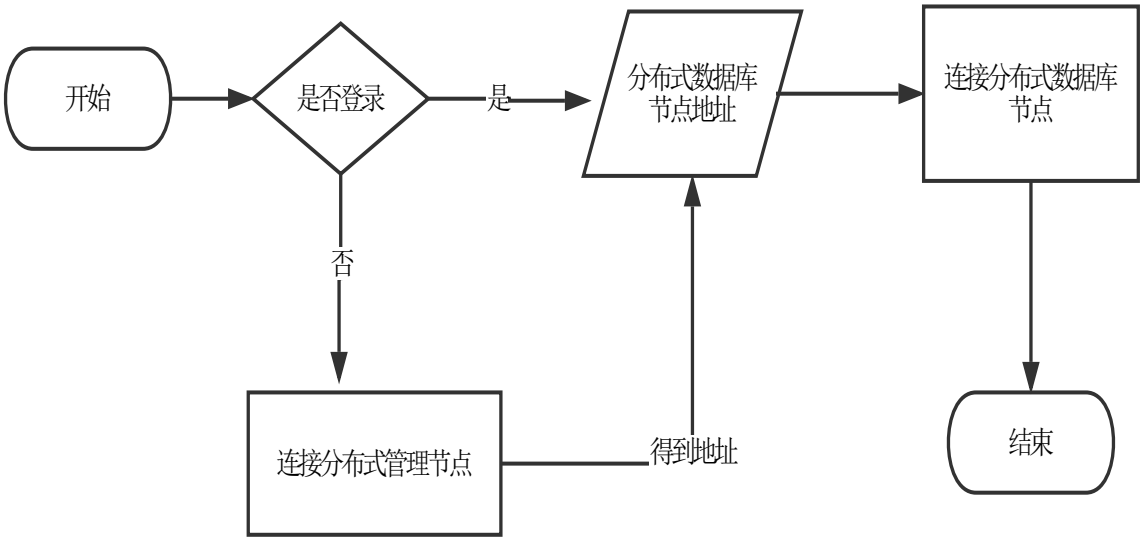


图 5-1 客户端连接功能流程图

首先，判断客户端有没有已经登录的会话，如果已经有已经登录的会话，那么就可以直接和对应的分布式数据库节点连接。发送数据库命令并且得到返回结果。当客户端没有连接会话的时候，客户端就首先连接分布式管理节点，分布式管理为客户端返回合适的数据节点地址，然后客户端利用这个地址再连接数据库

服务器。通过这样一个步骤，分布式管理节点能利用合适的负载均衡算法为客户端选择合适的数据库节点，有利于服务器资源的均衡利用。

5.3 分布式负载均衡算法实现

管理节点存储数据库集群的元数据，实现分布式负载均衡功能。同时也为管理员提供管理和监控功能。本节主要对分布式管理节点的负载均衡的实现进行说明。

负载均衡就是根据当前分布式系统节点的性能和负载状态信息，为客户端选择合适的分布式节点。达到均衡分配系统资源的目的。有利于系统资源的高效使用，提高系统的整体效率。同时也提高了系统的性能，为客户端选择的合适节点，也能节约网络负载资源。

根据算法考虑当前服务器的负载状态信息与否，负载均衡算法可以分为动态算法和静态算法目前, 静态负载均衡因为负载均衡效果已越来越不能满足需要, 动态负载均衡技术有取而代之的趋势。动态均衡技术又可以分为集中式方法和分布式方法，在集中方法中，单个节点负责管理内部整个节点的负载状态信息，有单点故障的危险；分布式方法中，每个节点，通过收集独立构建自己的负载状态信息，然后把负载信息分享给其他节点，同时能在集群中保存元数据，负者分布式管理功能的节点挂掉的时候，可以通过选择一个新的管理节点。

本论文实现了一种新的分布式负载均衡算法，其采用分布式集群来管理动态负载均衡信息，避免单点故障，负载效果良好。分布式负载均衡算法包括初始化和选举两个阶段。

5.3.1 管理节点初始化实现

在集群启动的时候，初始化管理节点，其算法流程如图5-2所示。

第一步，分布式数据库集群中的节点根据一定的方法选出一个节点作用分布式管理节点。管理节点除了能存储数据，响应用户的数据操作请求以外，还能作为分布式管理节点。分布式管理节点的作用就是为数据库客户端选择正确的后端数据库服务器，作为负载均衡功能的总代理。

第二步，分布式数据库集群中每个服务器节点根据公式5-1得到自己的权重值P，然后发送到分布式管理节点存储。其中 L_c为服务器节点CPU的个数， L_r为当前服务器总内存。

$$P = L_c * L_r \tag{5-1}$$

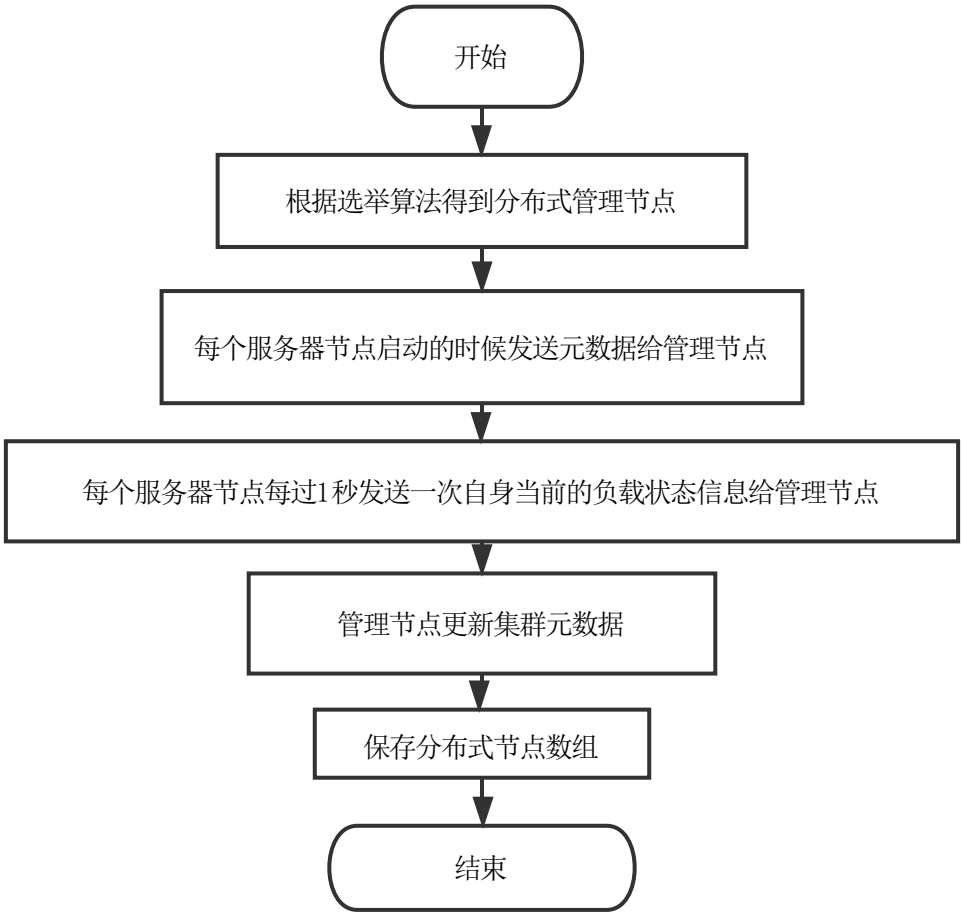


图 5-2 分布式管理节点初始化流程

第三步，分布式数据库节点每过1秒就根据公式5-2得到自己当前的负载S，同时发送到分布式管理节点。其中C为当前CPU使用率， R为当前服务器内存剩余量，单位为兆。

$$S = \begin{cases} 1 & \text{if } C > 0.9 \text{ or } R < 300, \\ 0 & \text{if other.} \end{cases} \tag{5-2}$$

第四步，分布式管理节点根据根据每个服务器节点的权重排序，然后使其排列成一个圆形。最后一个服务器下一个服务器为第一个服务器，保存在数据库的元数据中。这样分布式负载均衡就初始化完成了。

5.3.2 选举算法实现

在管理节点初始化完成以后，分布式集群中的服务器通过心跳定时向管理节点发送自身服务器信负载信息。在正确维护集群元数据以后，就可以利用选举算法为客户端选择合适的分布式数据库节点。每当客户端发出连接请求，分布式管理节点就会为客户端选择正确的服务器地址，其流程如图5-3所示。

当分布式管理节点收到客户端请求以后，遍历数据库的元数据库，依次检查其中的每个服务器节点，如果没有更多的服务器就返回错误信息；如果有的话，检查当前遍历节点的负载S。如果$S = 0$就返回当前服务器的地址，算法结束，下一次遍历的时候从下一个服务器节点开始检查；如果$S = 1$就跳过这个节点，再检查下一个节点，如此循环，直到算法正确结束或者返回给客户端错误消息。

按照上面的动态负载均衡算法来实现管理节点负载均衡功能，有如下好处：

1.管理节点其实和分布式数据库节点一样，都是分布式集群中的一台服务器，只是选择出来一个作为管理节点。分布式负载信息保存在所有服务器里面，所以当当前管理节点下线的时候，可以通过hazelcast选举出下一个管理节点，避免单点故障。

2.考虑到每个服务器的处理器和内存信息，根据每个服务器的能力进行排序。每次选举的时候都是从元数据数组的下一个节点选择，这样就可以避免一个大服务器全部处理客户端请求的情况。当处理器和内存到达一定的时候，就暂时放弃当前服务器。这样可以获取更好的均衡效果。

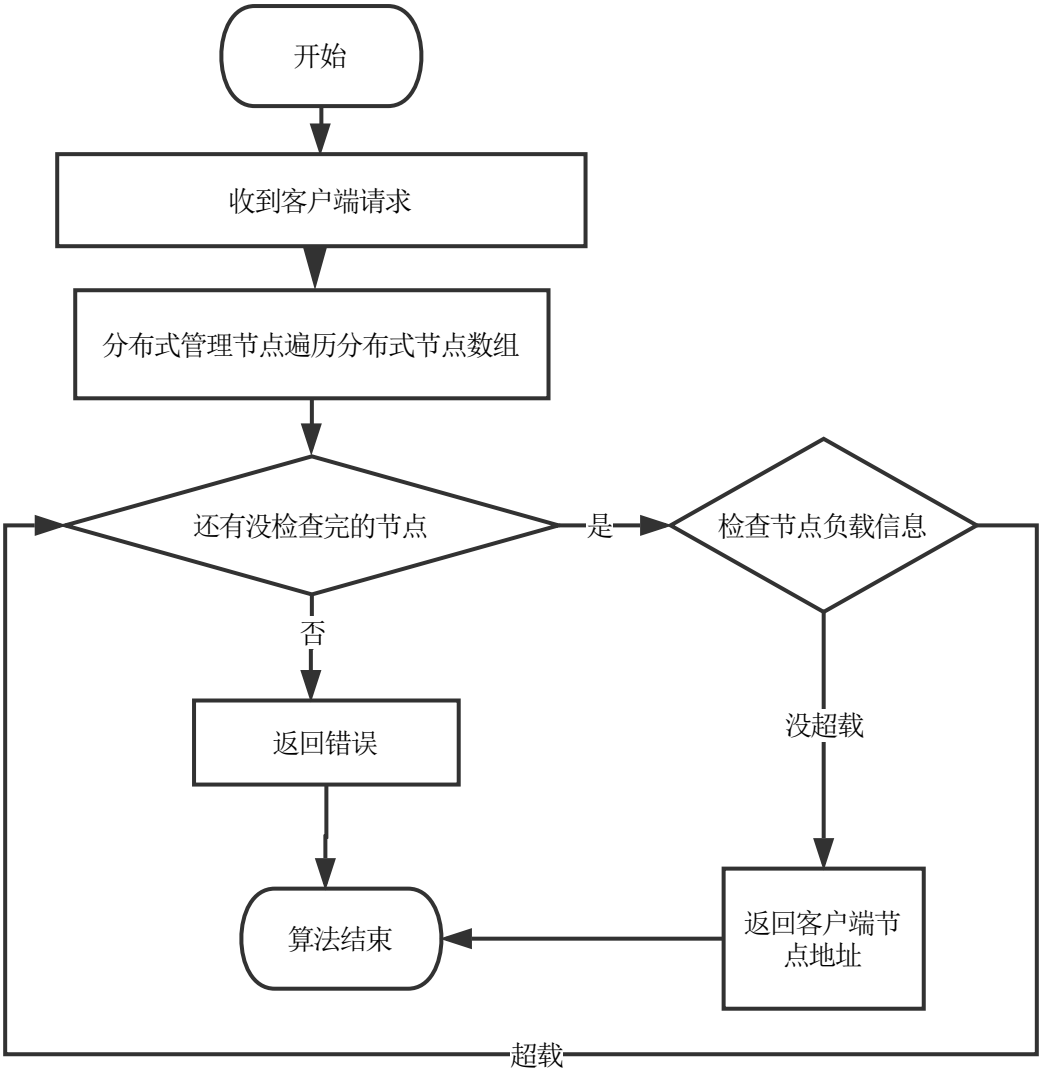


图 5-3 负载均衡选举流程图

5.4 多主分布式架构关键技术实现

传统的关系数据库从发明以来就是一个单机版系统，为了扩展数据库的读写能力，很多传统的关系型数据库系统支持复制。比如在mysql中，可以通过主从复制来扩展服务器的读取能力,如图5-4所示，在主从架构中，只有一个主服务器接受客户端的写请求，然后主节点传播这些更新请求到从服务器，从而达到每个服务器数据保持一致的状态。传统关系数据库系统的复制分为物理复制和命令复制，物理复制就是把底层数据存储引擎的更改情况传播到其他的从节点。命令复制就只需要传播SQL命令到从节点再执行一下。主从数据就一样了。主从复制虽然能够提高数据库系统的读取能力，但是其不能扩展服务器的写能力，因为多个服务器同时写会引起数据不一致的问题。因为主从复制相对简单容易实现，大多数传统关系数据库系统都只支持主从架构。

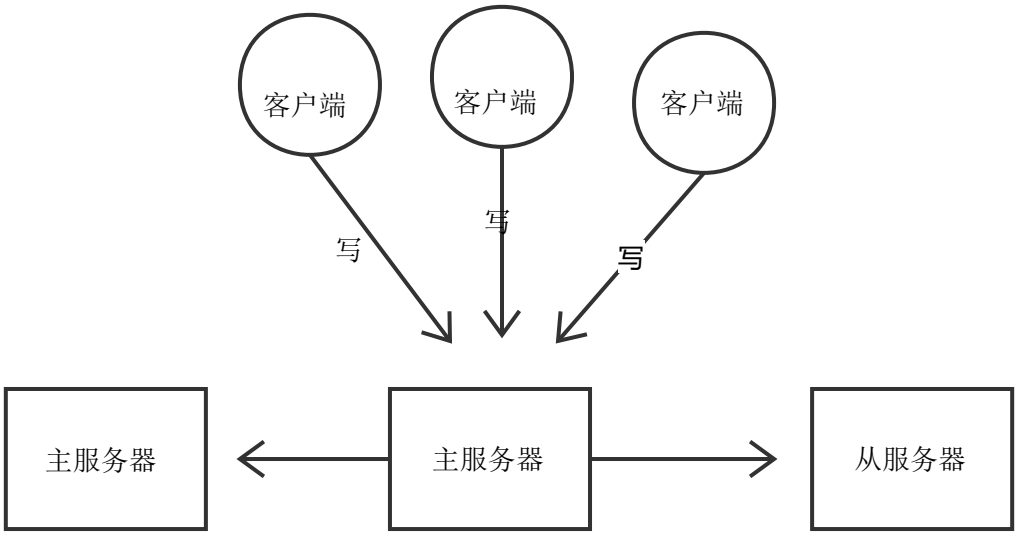


图 5-4 主从架构原理示意图

在本论文所述分布式系统中，论文实现了多主复制架构，改架构示意图如图5-5 所示，该架构和主从复制相比，不但能扩展数据库的读取能力，也能扩展数据库服务器的写能力，基本达到线性扩展的理想性能。不但能提高分布式系统的读取能力，也能提高分布式系统的写性能。

在分布式多主架构集群系统中，系统利用Hazelcast的多播节点自动发现功能，这样就可以做到零配置分布式部署，当增加服务器的时候，新的服务器节点会自动加到已有的分布式集群，在这个集群中，最先启动的服务器作为主节点，但主节点挂掉的时候，次启动的服务器代替主节点，如此循环，当服务器集群中只有一个服务器的时候，这个服务器就作为唯一的主节点。

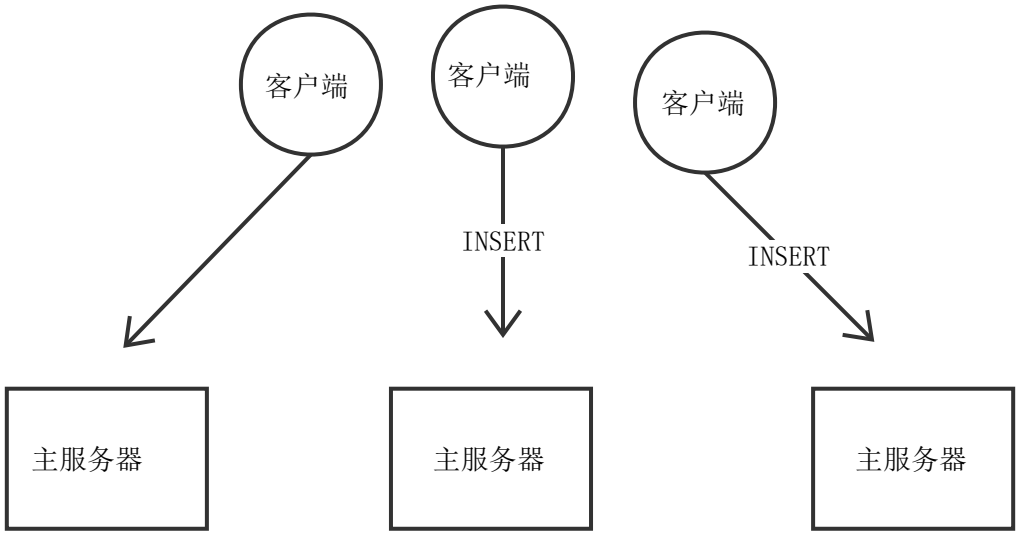


图 5-5 多主分布式架构原理图

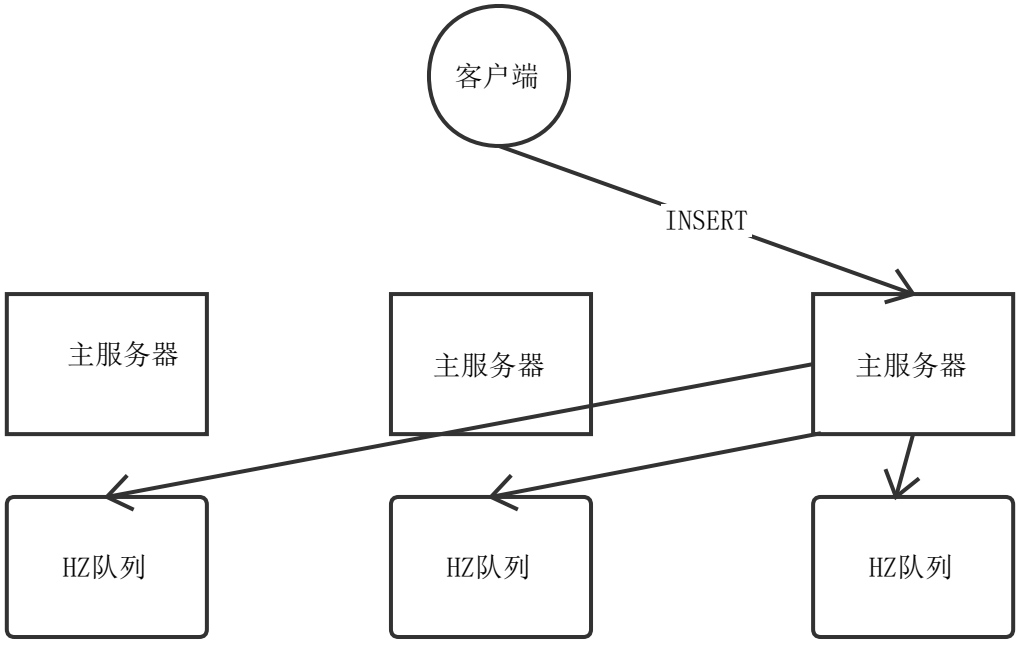


图 5-6 多主架构更新请求

在分布式多主架构中，系统的更新请求如图5-6 所示。系统利用Hazelcast的分布式队列功能保存分布式系统的命令请求，当前服务器处理完成以后就直接返回给客户端，这样可以减少服务器系统的网络延迟，提高系统的性能。当其他服务器都返回成功消息以后，就会把这个命令请求从分布式队列里面删除，如图5-7所示； 当有服务器执行失败的时候，就会把当前命令放到一个恢复队列里面进行执

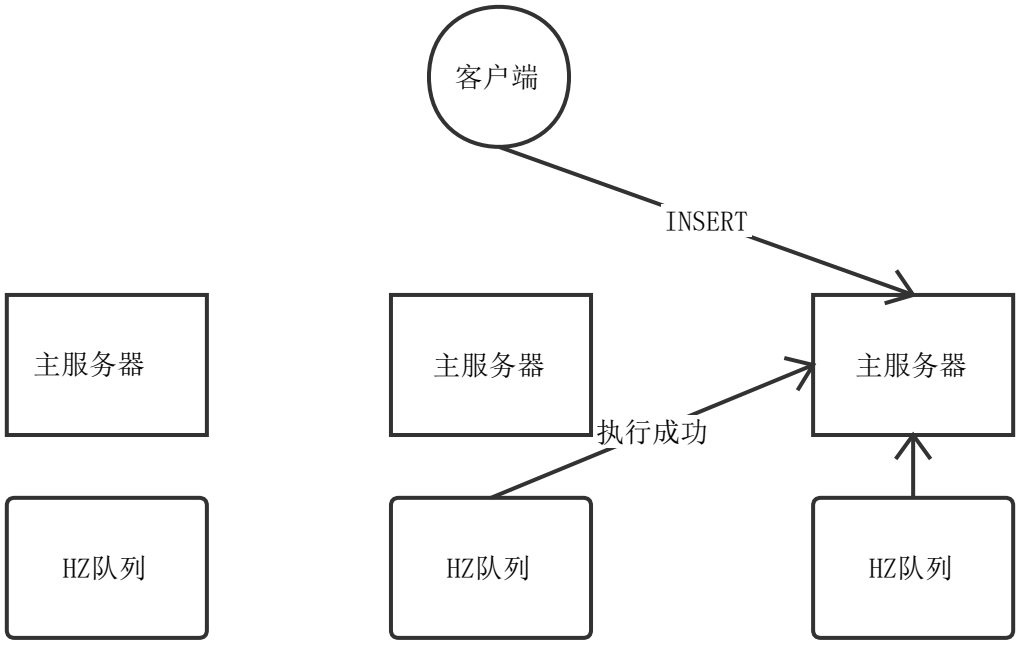


图 5-7 多主架构更新成功图

行。

JSQL分布式系统选择逻辑复制，这样可以减少网络中数据的传输数量，减少客户端的等待时间，提高服务器的整体性能。在分布式架构实现中，主要有三个关键的类，其类结构定义如图5-8所示。其中SqlUpdateLog作为数据库操作日志类，LSN为日志序列号，sql为操作命令语句，db为当前操作的逻辑数据库；ReplicationCMD对象为复制命令对象，当一个服务器节点启动的时候，如果其数据不是最新的，就需要发送复制命令给其他服务器，使得其数据可以和集群保存一致；MyHazelcast类图中的一些关键变量是实现分布式架构的基础，这些变量的作用后面会详细说明。在分布式数据库集群中，每个节点在启动的时候，都会经过初始化和命令执行两个阶段，下面对这两个阶段的实现流程进行说明。

5.4.1 集群初始化流程

在每个服务器启动的时候，就会执行集群初始化。集群初始化流程如图 5-9所示。 根据集群初始化流程图，服务器初始化过程包括以下步骤：

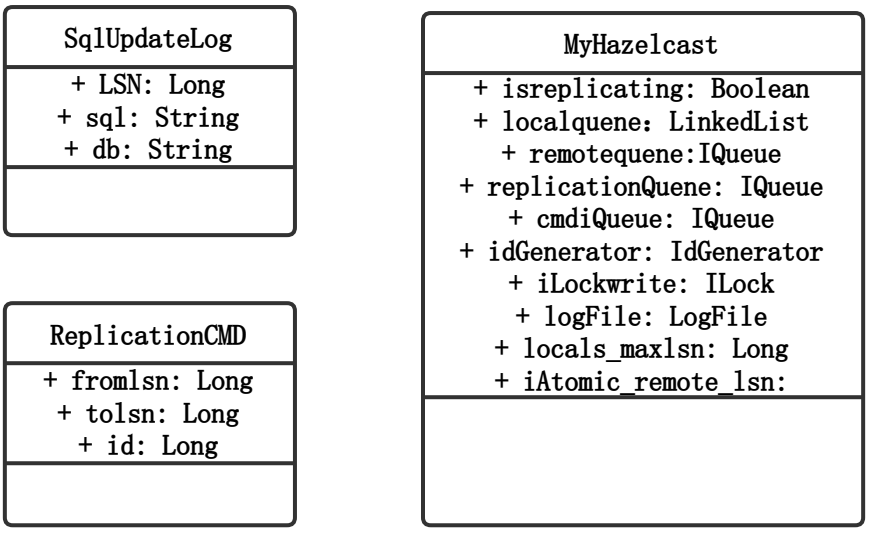


图 5-8 分布式架构相关类图

第一，获取当前服务器节点的本地日志序列号locals_maxlsn，locals_maxlsn代表当前服务器本地存储引擎已经保存的最新数据的日志记录，通过logFile类可以得到当前服务器的本地日志序列号。如果当前服务器没有存储任何的数据，那么这个locals_maxlsn就为零。

第二，获取当前集群的日志序列号iAtomic_remote_lsn,iAtomic_remote_lsn记录当前的集群最新的日志序列号。在获取集群日志序列号的过程中，如果当前服务器是最先启动的服务器，那么集群的日志序列号就还没有设置，当前服务器节点作为主节点，设置集群的日志序列号为当前服务器的本地序列号。在实现的过程中，用到了Hazelcast的分布式原子数据结构，能够保持分布式集群环境下，日志序列号数据的全局一致性。

第三，比较当前服务器的本地日志序列号locals _maxlsn和集群日志序列号iAtomic_remote_lsn,如果iAtomic_remote_lsn大于locals_maxlsn，就设置变量isreplicating为true，表示当前服务器节点进入复制阶段。在复制阶段。只能执行复制命令。不能执行其他服务器节点接受到的客户端命令。当前节点需要从集群更新复制数据的时候，就向cmdQuene队列发送复制命令请求，集群中的服务器接受到这个命令请求以后，就通过iLockwrite锁来得到这个命令。这个命令请求的服务器节点就会把最新的数据发送给当前服务器节点。当前服务器节点更新数据完成以后。才进行远程队列数据的同步。

第四，在当前节点进入复制阶段以后，有可能会收到客户端的命令请求，或者是收到其他服务器节点的远程命令请求。但是在复制阶段，当前服务器不能执

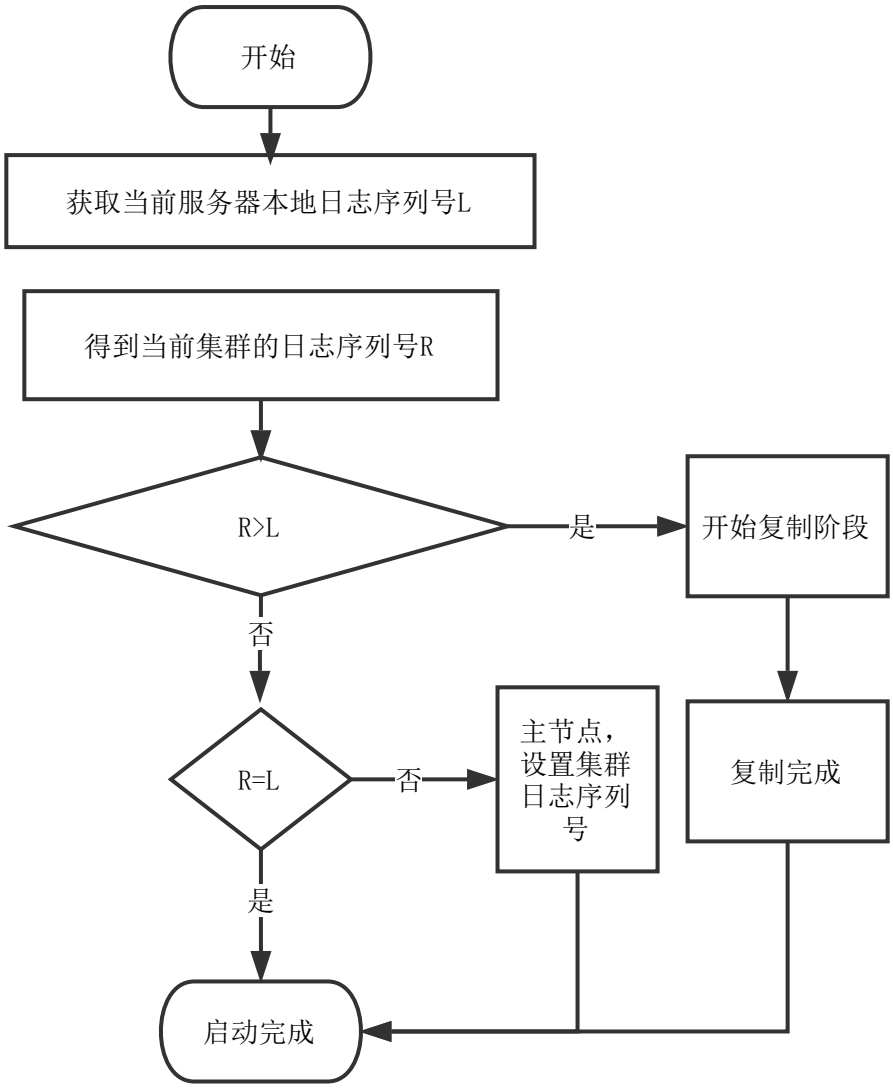

开始
获取当前服务器本地日志序列号L
得到当前集群的日志序列号R
R>L
是
开始复制阶段
否
R=L
否
主节点，设置集群日志序列号
复制完成
是
启动完成

图 5-9 集群初始化流程

行客户端的命令请求，所以需要localquene保存接收到的命令请求。

第五，在第三步完成以后，服务器节点执行localquene里面的命令。所以步骤完成以后。当前服务器节点的数据就和集群中的数据保持一致了。以后就可以接收客户端的命令，直接执行，返回结果。也可以同步执行远程队列里面其他服务器发送过来的命令。

5.4.2 集群命令执行过程

当服务器启动的时候，就可以接受客户端的命令请求，但是只有在服务器初始化完成以后，才会对命令请求进行处理。每个服务器节点启动的时候有两个阶段：初始化阶段和复制阶段。在这两个阶段，都需要暂时保存接受到的客户端命令。一旦初始化完成以后，就可以对客户端的命令进行处理，具体处理流程如图5-9所示。其具体的算法步骤如下：

1.客户端通过负载均衡选择到了当前服务器，然后向当前服务器发送命令请求。判断当前服务器的状态，如果在复制阶段，则说明当前服务器节点正在初始化阶段，还不能执行命令。

2.当服务器处于复制状态的时候，需要获取集群的当前日志序列号，然后根据当前的日志序列号和命令请求，生成日志对象。暂时保持在本地队里面。待复制阶段完成以后，依次执行本地队列里面的命令请求。

3.如果当前服务器初始化已经完成，没有处于复制状态，那么说明本地数据和集群数据已经保持一致，可以执行命令。

4.本地日志序列号增加1，代表本地数据已经更新。

5.集群日志序列号增加1，向集群中所有的服务器节点广播，表示已经有新的数据更新。

6.当前服务器节点保存日志记录，已持久化数据存储，然后处理命令请求。返回数据给客户端。

7.发送日志记录到远程队列。其他服务器节点就可以通过远程队列来同步更新里面的数据记录。保持和当前服务器的数据一致性。

5.5 数据库基本功能实现

数据库系统功能在是分布式数据库节点中最重要的功能模块，提供数据的更删改查等数据库最基本的功能。在源代码实现里面， SQL包下面的类主要是作为数据库系统管理类，让用户启动数据库服务器或者关闭数据库服务器。表5-3描述了SQL包下每个类的作用。 其中SpringMain类是数据库系统功能的入口，这个

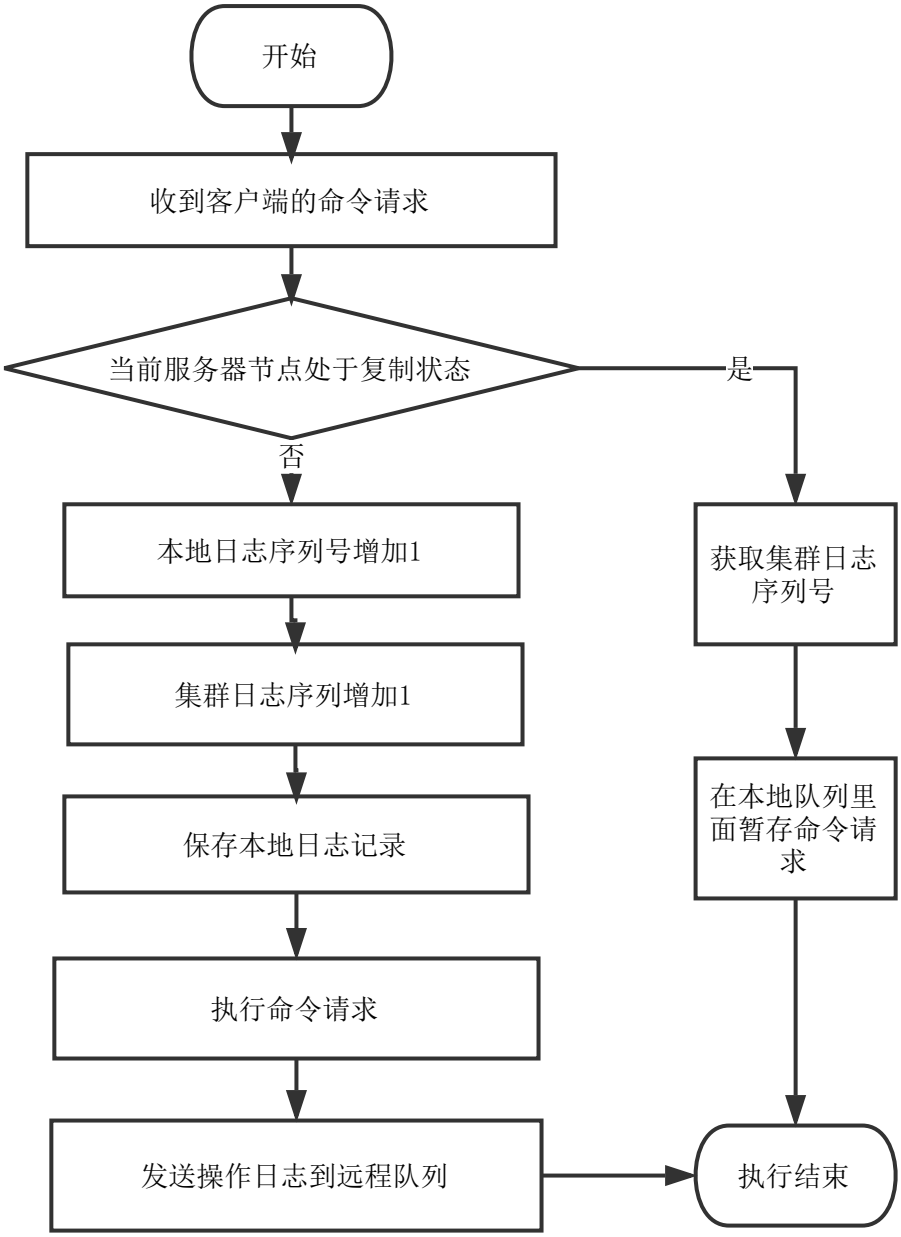


图 5-10 集群命令执行流程

表 5-3 SQL包的每个类的作用

类	功能
ShutdownMain	关闭本机服务器
SpringMain	启动类，通过spring boot启动。 ioc，aop功能

类主要调用其他模块，完成数据库系统的启动。 ShutdownMain类用来关闭数据库系统。按照客户端请求信息的处理流程，数据库功能模块主要包括网络模块，SQL解析模块和存储引擎模块，下面分别描述每个模块具体的实现。

5.5.1 网络和协议模块

在网络模块，处理网络套接字的连接和网络字节的处理。在分布式数据库节点接收到客户端的套接字连接请求以后，网络模块服务器端套接字接收请求，完成三次握手建立连接以后。交个Netty来处理。网络字节处理模块主要是用了Netty来实现的，所有和网络字节处理有关的代码都在netty包下面。表5-4描述了每个类的功能。 在接收到客户端套接字连接以后，客户端信息会依次经过下面四个类的处

表 5-4 网络模块中各个类的作用

类	作用
ByteToMysqlDecoder	接受客服端的字节消息，转化为mysql的消息格式
ByteToMysqlPacket	包字节格式转化为包对象
MysqlPacketHander	解码器，处理接受到的包对象
NettyServer	前端线程池，接受客服端的请求

理：

1.ByteToMysqlDecoder

2.ByteToMysqlPacket

3.MysqlPacketHander

4.NettyServer

具体处理流程如图5-11所示。 网络模块收到客户端的套接字连接以后需要包网络字节包装成Mysql的消息包格式，如果错误就说明客户端不是我们的客户端或者客户端发送了其他的错误。当截取到消息包格式以后，还需要解析成我们的数据结构，如果解析成功就包数据结构发送到解析模块，如果解析错误，就断开连接。

除了实现网络模块以外，我们还要实现通信协议，系统采用了和mysql一样的通信协议，在mysql包下面实现了mysql的通信协议。表5-5给出了实现通信协议所用到的所有的类。

网络服务器接受到客户端的连接以后，就要解析mysql的通信协议，把每一个消息包封装到具体的对象里面， mysql协议里面有很多的包，每个协议包都需要一个类来实现，图5-12是握手包的类图， 其他包的实现大体相同，所以在这里不再重复说明。

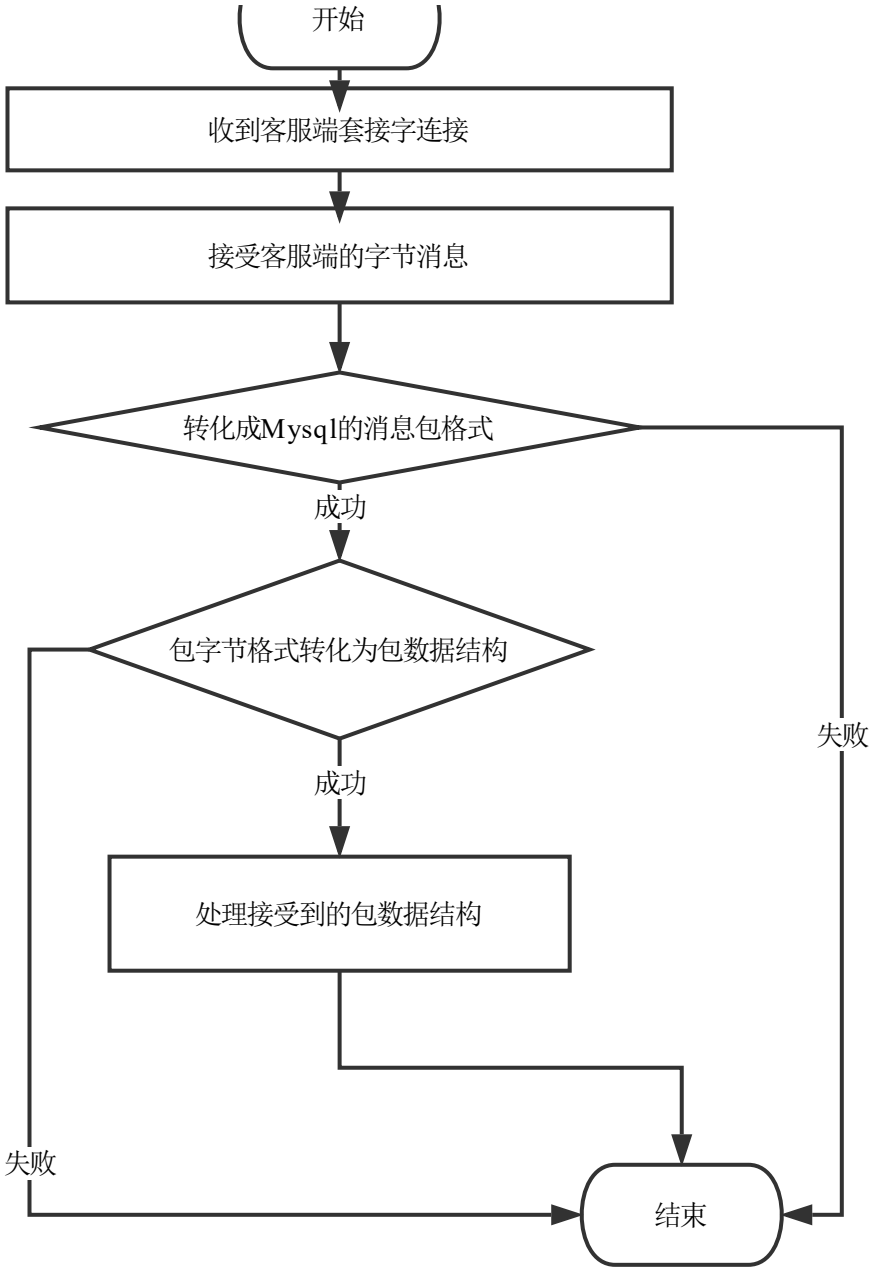


图 5-11 网络模块功能图

表 5-5 通信协议实现所用到的类

文件	类
BindValue	class BindValue
BindValueUtil	object BindValueUtil
BufferUtil	object BufferUtil
ByteUtil	object ByteUtil
CharsetUtil	object CharsetUtil
MBufferUtil	object MBufferUtil
MySQLMessage	class MySQLMessage
PacketUtil	object PacketUtil
PreparedStatement	class PreparedStatement
SecurityUtil	object SecurityUtil
StreamUtil	object StreamUtil

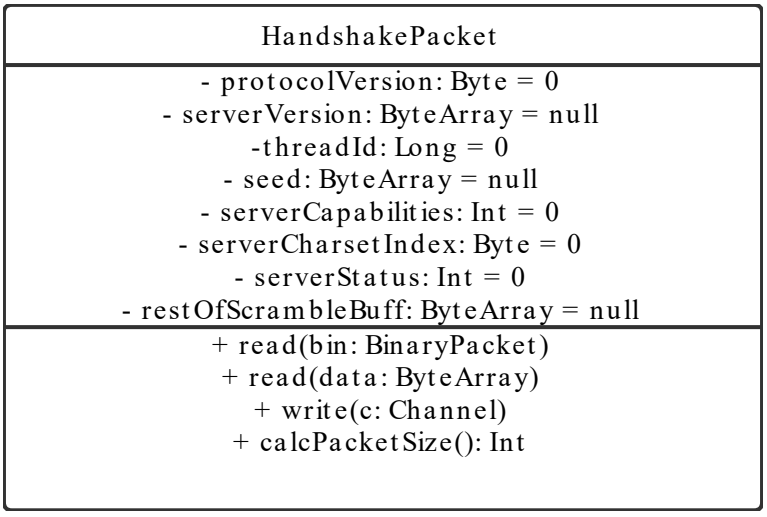


图 5-12 握手包实现类图

解析到协议包以后我们就要对协议包进行处理，协议包的主要有两个阶段，一个是认证阶段，一个是命令阶段，认证阶段就是接受客户端的请求，然后检查用户的认证信息，比如用户名和密码，图5-13显示了认证流程过程。 如果认证成

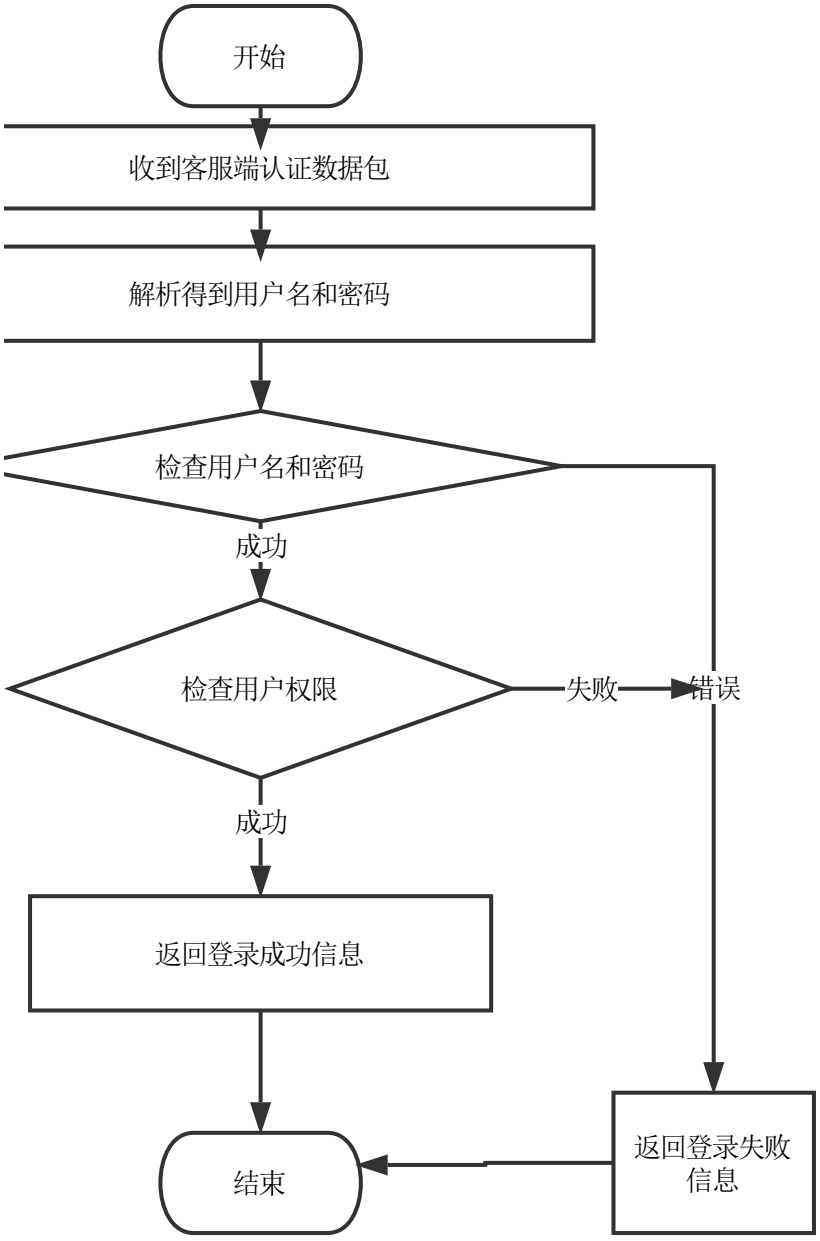


图 5-13 认证流程图

功以后，就建立连接会话，然后客户端就可以进行向服务器发送命令请求向分布式数据库节点处理数据。如果失败失败就返回给客户端失败原因。认证成功以后就是命令阶段，服务器接受客户端的命令，然后处理，图5-14显示了命令处理流程。 在接收到客户端的命令请求以后，网络模块就会调用下面模块的功能接口来对这个命令进行处理。如果命令处理过程中没有发生异常，那么就把命令处理结

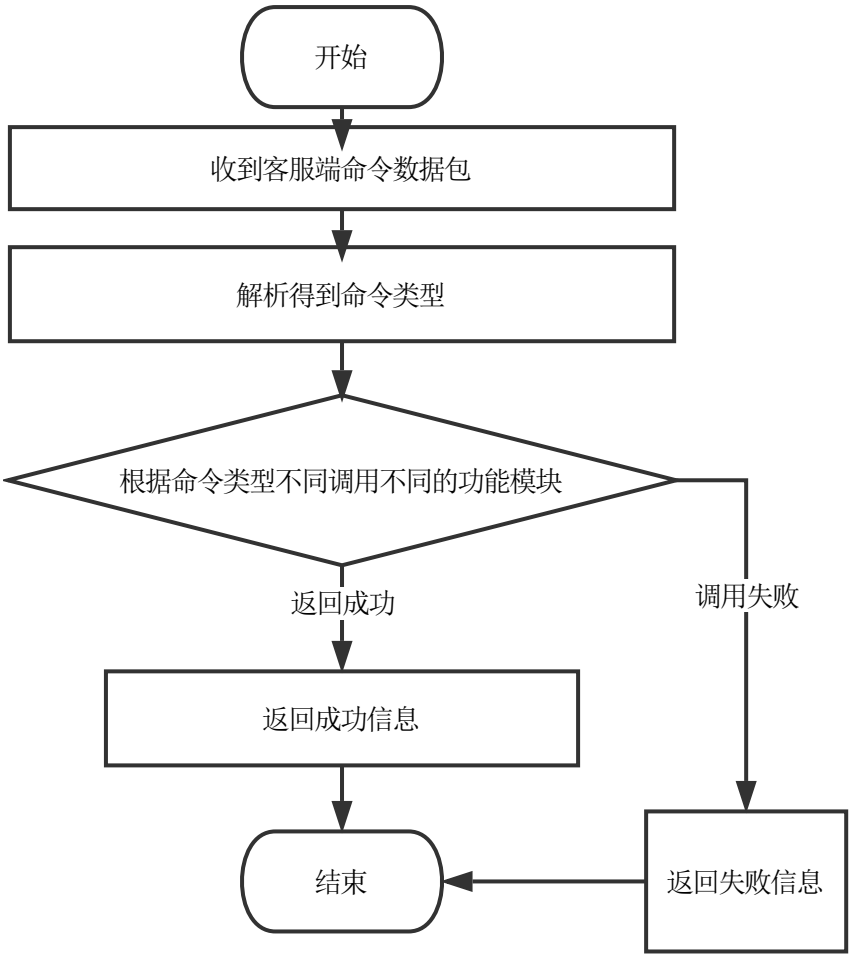


图 5-14 命令处理流程图

果发送给客户端，如果命令处理工程中发送了异常，就返回失败原因。

5.5.2 SQL解析模块

经过网络模块的处理，我们建立了客户端的连接会话，接下来我们就要对客户端发送过来的命令进行判断，如果是SQL命令，就要对这个命令进行解析，进行词法分析，语法分析和语意检查。在代码实现中，sql包下面的类主要完成对客户端连接和会话的封装和管理，表5-6显示了sql包下面每个类的具体作用。在sql解析功能模块里面系统用到了druid开源框架，用来解析sql，解析sql以后就

表 5-6 SQL前端连接模块相关的类

类	作用
MysqlSQLhander	接受前端的语句，处理用户的请求
OConnection	前端连接对象，一个客服端一个连接
ONullConnection	继承连接对象，但是不处理具体任务，用在分布式中
OconnectionPool	管理前端的连接，作为一个连接池对象

要做具体的数据处理，图5-15显示了SQL模块的处理流程图。 在sql解析流程中，如果遇到Druid框架不支持的语句，我们就要自己对这个语句进行解析。再这个工程中。任何一个步骤发送错误就返回错误信息给客户端。不然就进行发送给下一个功能模块进行具体的数据操作。 mysql当中的sql语句有很多种，每一种语句都要做不同的处理。每一种类型语句下面又回有很多种具体语句的实现，表5-7给出了数据操纵类型语句实现的所有相关的类和响应的功能。其他类型的语句也需要进行相同的处理，这里就不再进行详细表述。

表 5-7 数据操纵数据实现所相关的类

文件	作用
MSelectHandler	处理查找语句
Mcall	处理函数调用语句
Mdelete	处理删除语句
Mdo	处理mysql中的do语句
Mhandler	所有类的基类
Minsert	处理插入语句
MloaddataINfile	处理导入文件的语句
Mloadxml	处理导入xml的语句
Mrepelace	处理解释语句
MselectVariables	处理查找变量的语句
Msubquery	处理子查询
Mupdate	处理更新语句

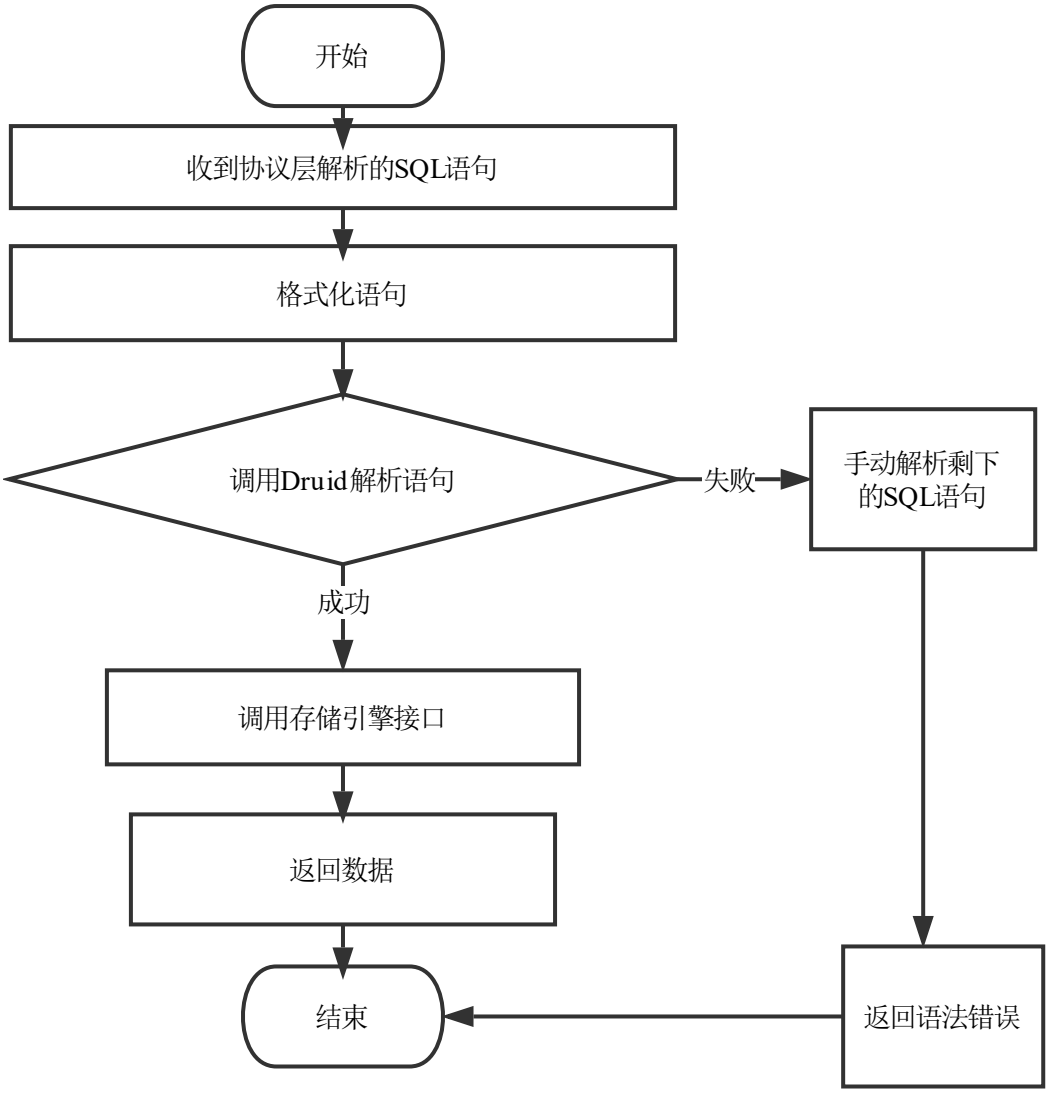


图 5-15 SQL处理流程图

5.5.3 存储引擎模块

存储引擎有关的功能主要在storage包下面实现，表5-8给出了 storage包下面各种文件的功能。 其中DB类作为底层存储引擎的接口，他的功能接口如表5-9所示。 和表有关的功能接口如表5-10所示。

表 5-8 storage包下面各种文件的作用

文件	功能
DB	接口，规定所有的存储引擎应该实现的和数据库有关的功能
Table	接口，规定所有的存储引擎应该实现的和表有关的功能

表 5-9 数据库存储引擎的函数接口

接口	函数签名
close	abstract fun close(): Unit
createdbAsyn	abstract fun createdbAsyn(dbname: String): Unit
createdbSyn	abstract fun createdbSyn(dbname: String): Unit
deletedbAyn	abstract fun deletedbAyn(dbname: String): Unit
deletedbSyn	abstract fun deletedbSyn(dbname: String): Unit
exe	abstract fun exe(sql: String, db: String): Unit
exesqlNoResultAsyn	abstract fun exesqlNoResultAsyn(sql: String, dbname: String): Unit
exesqlforResult	abstract fun exesqlforResult(sql: String, dbname: String):Oresult
getallDBs	abstract fun getallDBs(): List<String>
getdb	abstract fun getdb(dbname: String): ODatabaseDocument
query	abstract fun query(sqlquery: String, dbname: String): Stream<OElement>

表 5-10 数据库引擎和表有关功能的接口

函数	函数签名
createtableSyn	abstract fun createtableSyn(dbname: String, createTableStatement
droptableSyn	abstract fun droptableSyn(dbname: String, table: String): Unit
getalltable	abstract fun getalltable(dbname: String): List<String>
gettableclass	abstract fun gettableclass(tablename: String, db: String): OClass
selectSyn	abstract fun selectSyn(oClass: OClass, dbname: String): Stream<OElement>

存储引擎利用非关系型数据存储，为上面模块提供操作接口。其中有关数据库操作的类接口如图5-16b所示，有关表操作的类接口如图5-17所示。类的实现主要是封装了OrientDB存储引擎的功能，为上一次提供关系操作的接口。在关系数据库中，操作接口就是SQL语言。JSQL因为选择了兼容Mysql协议，所以实现了大部分的Mysql语句功能。这样上层功能模块就可以当做Mysql一样的使用本数据库引擎。

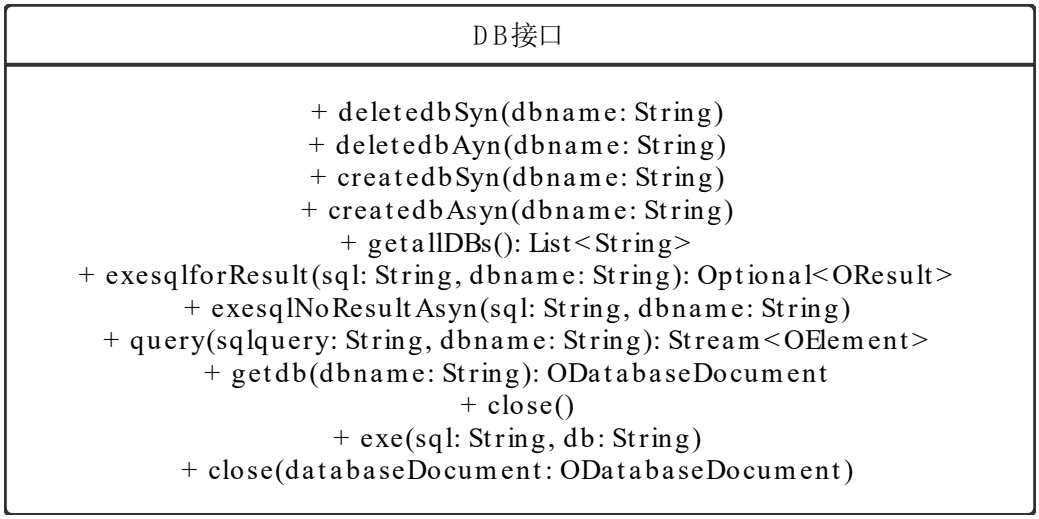


图 5-16 数据库操作接口类图

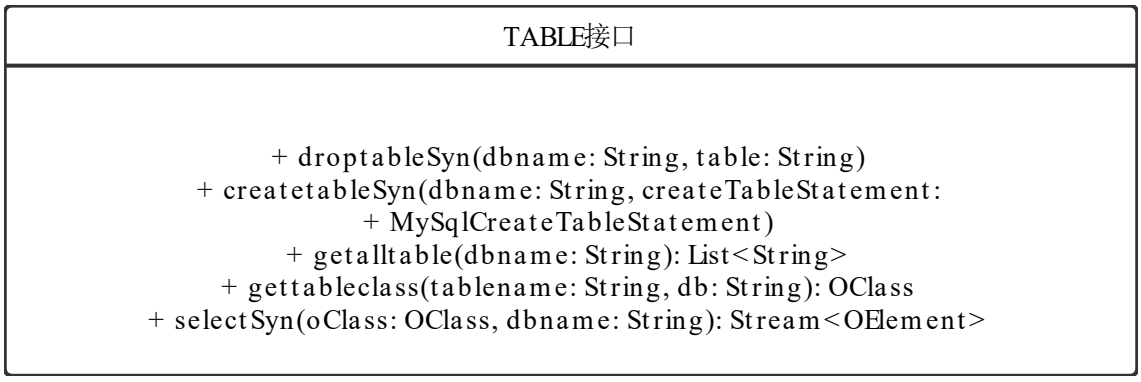


图 5-17 表操作接口类图

5.6 数据审计模块的实现

JSQL除了结合关系数据库和非关系型数据库以外，还从数据库的底层考虑数据的安全问题，对数据审计功能进行了简单的实现，在代码实现中，审计模块有关的功能全部在audit包下面实现，表5-11给出了audit包下面每个类的具体的作用。其中LoginLog和SQLlog类主要封装了两种日志类型，一种是客户端的登录记录日志，一种是客户端的命令执行日志。通过封装这两种日志记录，很容易的就可以实现审计数据的收集和发送功能。图5-18显示了审计功能演示图。 审计功能主要

表 5-11 audit包下面每个类的作用

类	作用
LoginLog	简单对象，记录登陆日志
SqlLog	简单对象，记录SQL执行日志
elasticUtil	工具类，包日志记录发送到ELK

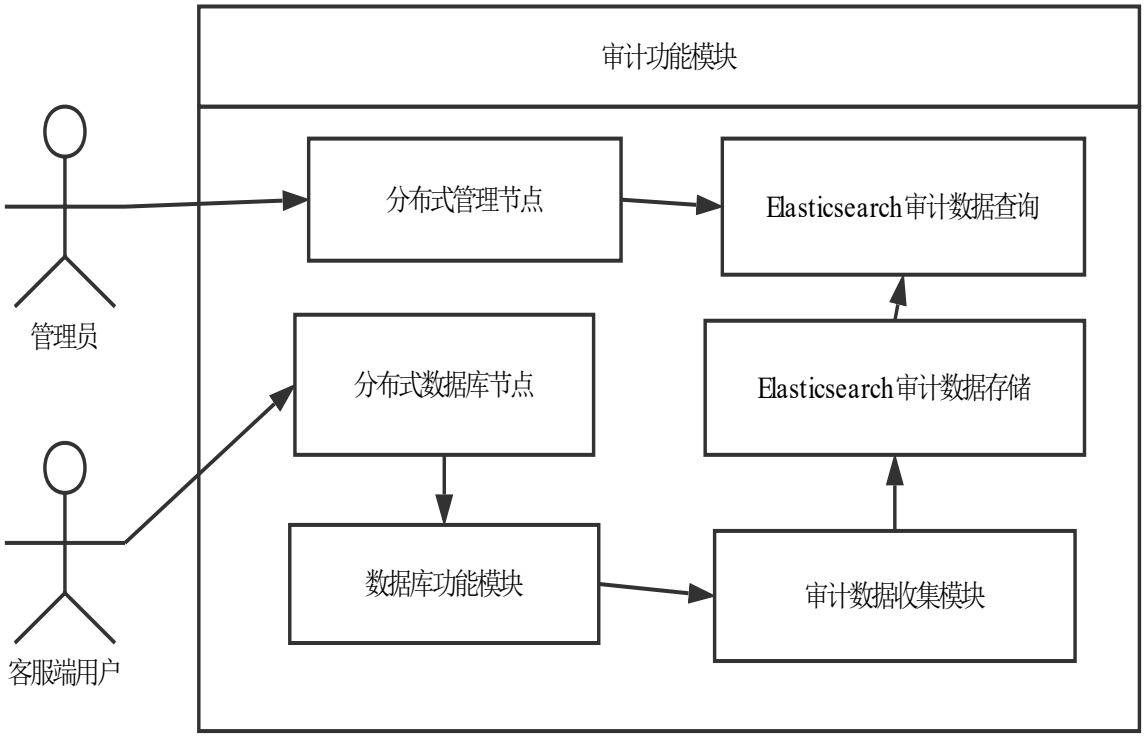


图 5-18 审计功能演示图

包括审计数据库，审计管理器和审计可视化功能模块，下面分别对每个模块进行说明

1.审计数据库: 审计数据库用Elasticsearch来实现，作为审计数据库，不能让用户随意的更改，所以本系统更改了它的源代码，使得它只能增加数据和查找数据不能更改和删除数据。其中主要存储的对象如图5-11所示。

2.审计管理器: 审计管理功能全部在audit下面实现，主要是存储本地的日志文件，然后发布到审计数据库。提供给前端可视化的接口。

3.审计可视化模块的实现: 审计可视化模块的实现用到了grafana，主要用来监控ELK中的数据。管理员通过连接分布式管理节点可以对审计数据进行查询。从而对数据的更改情况进行掌控。

5.7 本章小结

对系统进行详细设计以后就是编码阶段，分布式数据库系统JSQL完全采用JAVA语言开发，本章首先对系统代码的结构进行说明，然后给出系统设计阶段所划分的关键功能模块的实现，主要包括客户端功能模块、数据库功能模块和数据审计功能模块，对每个功能模块，通过流程图或者代码的方法描述其功能实现。本章重点对论文所用分布式负载均衡算法和多主分布式架构的实现进行了详细的阐述。

第六章 系统测试

系统测试是软件开发过程的重要步骤，用来检验所开发的系统是否满足需求分析步骤中所说的要求。根据测试目的，系统测试可以分为功能测试和性能测试，功能测试主要是对需求分析说明中的功能进行验证，验证系统是否达到基本的功能要求，性能测试主要从时间和空间两个方面来看系统的能力。作为分布式多多模型数据库系统，其时间主要是响应客户端的命令请求时间，而空间则主要是运行系统所需要的环境要求。

6.1 测试方法

在软件测试方法中，分为黑盒测试和白盒测试，白盒测试是开发人员对系统内部做仔细的测试。对系统中关键算法部分进行数据测试等等；黑盒测试是把整个系统当成一个黑盒子，测试者不需要了解其具体的结构，站在系统使用者的角度对系统进行功能和性能测试。本章选择用黑盒测试方法对系统进行功能和性能测试。测试方法为，在一定的硬件和软件平台上运行分布式数据库系统，然后用mysql客户端对进进行功能测试。在性能测试的部分，用到了常用的性能测试软件，以可视化的界面显示结果。为了测试的方便，作者写了一些测试代码，更加方便的对数据库进行大量重复的操作。

6.2 测试环境

因为论文所述的系统是纯java语言开发，所以其能在大多数的操作系统和计算机硬件上运行。本章测试所用计算机为实验室个人台式机电脑，它的硬件和软件参与如下

1.硬件参数：

(a) CPU：intel(R) i5-4460

(b) 内存：8G

(c) 磁盘：120G固态硬盘和500G磁盘

(d) 网络：100Mbit/s

2.软件参数：

(a) 操作系统：windows10

(b) java版本：8.0

6.3 功能测试

功能测试就是按照前面需求分析阶段的结果对系统进行功能性验证，验证所开发的系统是否满足需求分析中所述的功能要求。作为一个分布式多模型数据库系统，其系统规模大，功能点非常多，本测试主要是测试系统的结构化数据存储功能上，这个功能是关系型数据库的基本功能，选择测试这个功能也方便和其他主要关系数据库系统进行对比。

6.3.1 数据库功能测试

作为一个分布式数据库系统，其首要的功能就是能存储数据，根据用户的操作请求对数据进行增加、删除和更改等操作。这里将给出几个关键的功能测试的条件、目的、步骤和结果。其他更为细节的，比如对失败的查询的测试，或其他类似的，比如与插入、更新类似的删除数据，删除表或者数据库等功能，由于篇幅原因，就不再细述。

1.系统启动测试

(a) 测试条件：代码编码完成，系统功能正常

(b) 测试步骤：

i. 启动系统

ii. 用mysql客户端连接系统

(c) 测试结果：系统能正常启动，而且mysql客户端也可以连接上系统

2.工具语句测试

(a) 测试条件：系统启动成功，客户端连接上系统

(b) 测试步骤：

i. 打开命令行客户端

ii. 输入语句show databases，发送给服务器

(c) 测试结果：系统返回当前所有的数据库。

3.建表语句测试

(a) 测试条件：系统启动成功，客户端连接上系统

(b) 测试步骤：

i. 打开命令行客户端

ii. 输入语句create table test(id int,name varchar(100))

iii. 回头，发送命令给服务器

(c) 测试结果：系统返回正确，而且文件系统里面多了一个文件。

4.插入语句测试

(a) 测试条件：系统启动成功，客户端连接上系统

(b) 测试步骤：

i. 打开命令行客户端

ii. 输入语句insert into test(id,name) values(1,'changhong');

iii. 回头，发送命令给服务器

(c) 测试结果：系统返回正确

5.查询语句测试

(a) 测试条件：系统启动成功，客户端连接上系统

(b) 测试步骤：

i. 打开命令行客户端

ii. 输入语句select * from test;

iii. 回头，发送命令给服务器

(c) 测试结果：系统返回一行数据。

6.更新语句测试

(a) 测试条件：系统启动成功，客户端连接上系统

(b) 测试步骤：

i. 打开命令行客户端

ii. 输入语句update test set name='changhong1'

iii. 回头，发送命令给服务器

iv. 输入语句select * from test

v. 回头，发送命令给服务器

(c) 测试结果：系统执行正确，返回一条数据，名字字段weichangohng1

7.更新语句测试

(a) 测试条件：系统启动成功，客户端连接上系统

(b) 测试步骤：

i. 打开命令行客户端

ii. 输入语句delete from test

iii. 回头，发送命令给服务器

iv. 输入语句select * from test

v. 回头，发送命令给服务器

(c) 测试结果：系统执行正确，没有返回数据

6.3.2 集群功能测试

分布式系统的测试是最难的测试步骤，好在本系统采用java语言开发，一个java虚

拟机就可以当做一个分布式数据库服务器节点，这非常方便对系统进行集群功能测试。分布式数据库集群分和初始化阶段和命令执行阶段，本测试只对数据库集群的命令执行阶段进行测试。测试的方法和单机版数据库功能测试一样，客户端发送命令，验证集群是不是满足功能要求。考虑到操作命令非常的多，不可能一一进行测试。所以，本次测试只测试最常用的insert语句：

1.测试条件：

(a) 数据库1启动成功，没有任何数据

(b) 数据库2启动成功，没有任何数据

(c) 数据库1和数据库2都有一个test表

2.测试步骤：

(a) 打开命令行客户端，连接数据库1

(b) 输入语句insert into test(id,name) values(1,'changhong')并且执行;

(c) 断开连接，连接数据库2

(d) 输入语句select * from test;

(e) 回头，发送命令给服务器

3.测试结果

(a) 数据库1执行语句正确

(b) 语句2执行正确，并且返回一条数据

6.3.3 审计功能测试

分布式系统的审计功能用来对分布式数据库系统进行简单的监控和预警功能，审计功能为管理员提供可视化的界面，方便对其进行查看。所以审计功能的测试部分主要是看审计前端能否正确的显示当前服务器的状态和所有对分布式系统的操作情况。

1.测试条件：

(a) 数据库1启动成功，没有任何数据

(b) 数据库1有一个test表

2.测试步骤：

(a) 打开命令行客户端，连接数据库1

(b) 输入语句insert into test(id,name) values(1,'changhong')并且执行;

(c) 打开grafana web界面

3.测试结果

(a) 数据库1执行语句正确

(b) 通过web界面可以观察到插入的数据和执行者的信息

(c) 图6-1、图 6-2和图 6-3显示了监控的可视化结果

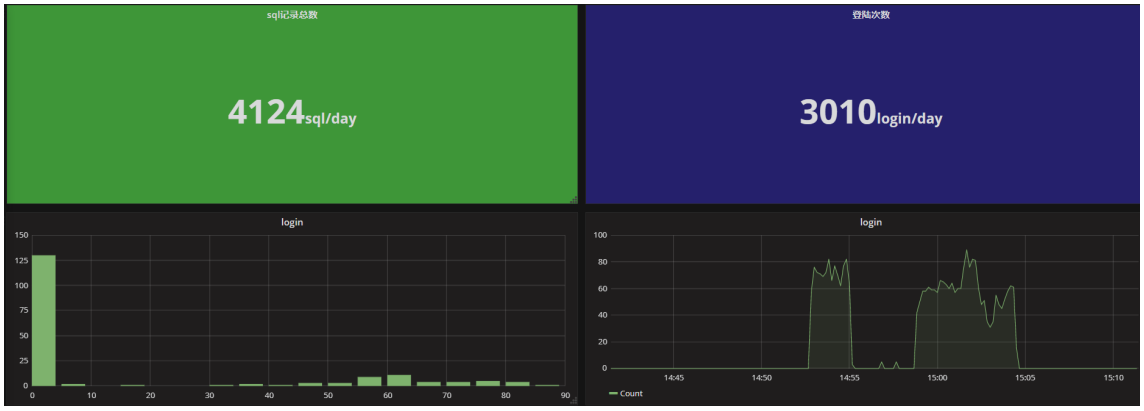


图 6-1 监控可视化结果

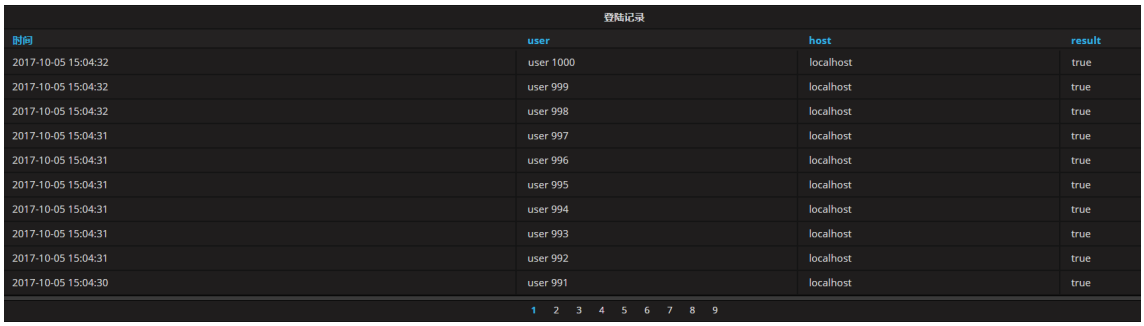
图 6-2 监控可视化结果

通过可视化结果可以看到，本系统的审计功能能正确的显示出用户的登陆和命令操作情况。对所有时间段的命令执行情况进行了记录。方便管理员随时查看数据库的状态。

6.4 性能测试

数据库的更删改查是最常用的操作，其中又以查找使用的场景更多，所以本节主要测试数据库的查询性能。jsql和mysql都可以通过JDBC来连接，下面的源代码是测试代码，用来测试查询次数和执行时间的关系。执行结果如图6-4所示。

```
01    fun test() {
02        //sql执行次数
```



登陆记录

时间	user	host	result
2017-10-05 15:04:32	user 1000	localhost	true
2017-10-05 15:04:32	user 999	localhost	true
2017-10-05 15:04:32	user 998	localhost	true
2017-10-05 15:04:31	user 997	localhost	true
2017-10-05 15:04:31	user 996	localhost	true
2017-10-05 15:04:31	user 995	localhost	true
2017-10-05 15:04:31	user 994	localhost	true
2017-10-05 15:04:31	user 993	localhost	true
2017-10-05 15:04:31	user 992	localhost	true
2017-10-05 15:04:30	user 991	localhost	true

1 2 3 4 5 6 7 8 9

图 6-3 监控可视化结果

```
03          val sqlnumber = intArrayOf(10, 100, 1000, 3000, 5000, 10000)
04          val mysql = "jdbc:mysql://localhost:3306/changhong?" + "user=root&"
+
05                  "password=0000&useUnicode=" + "true&characterEncoding=UTF8"
06          var connection = DriverManager.getConnection(mysql)
07          var statement = connection.createStatement()
08          println("mysql sqlnumber to times:")
09          for (number in sqlnumber) {
10              val start = System.currentTimeMillis()
11              for (i in 1..number) {
12                  statement.executeQuery("select * from test")
13              }
14              val end = System.currentTimeMillis()
15              println("$number      ${end - start}")
16          }
17          connection.close()
18          val jsql = "jdbc:mysql://localhost:9999/changhong?" + "user=root&"
+
19                  "password=0000&useUnicode=" + "true&characterEncoding=UTF8"
20          connection = DriverManager.getConnection(jsql)
21          statement = connection.createStatement()
22          println("jsql sqlnumber to times:")
23          for (number in sqlnumber) {
24              val start = System.currentTimeMillis()
25              for (i in 1..number) {
```

```
26                  statement.executeQuery("select * from test")
27              }
28              val end = System.currentTimeMillis()
29              println("$number      ${end - start}")
30          }
31          connection.close()
32      }
33  }
```

SQL执行次数	MYSQL完成时间（毫秒）	JSQL完成时间（毫秒）
10	5	4
100	31	40
1000	252	315
3000	1207	941
5000	648	1504
10000	1297	4495

图 6-4 mysql和jsql的性能比较

从图6-4可以看出，随着SQL执行次数的增加，mysql和jsql的执行时一般都相应的增加，在3000执行次数为3000以下时，jsql和mysql的性能相当，当执行次数大于5000时，mysql的性能小幅度的超过jsql。再从变化幅度来看，当执行语句是5000次数的时候，mysql的执行时候审计比执行3000次数的时候还要少，这点可能是因为mysql本身的缓存或者优化有关。而jsql就相对来说随着执行次数的增加，时间就随着增加，符合预期。

除了jdbc的测试以外，本次测试还用了一个流行的测试框架JMeter。 Apache JMeter是用java语言开发的测试框架，其主要用来对软件进行压力测试，本论文所述系统用JMeter测试jsql的结果如图6-5所示。用JMeter测试jsql的结果如图6-6所示。

从测试结果来看，在执行简单的SQL语句的时候，jsql相对mysql的性能有一定的差距。其中一个原因是jsql是用纯java语言开发的，而mysql用c语法开发，其中自然有一定的性能损耗；另一个原因是jsql数据库引擎在存储数据的时候还要直接存储数据直接的关系，所以在检查的数据模型的测试下，其性能优势并不能表现出来。当关系表直接的连接非常多的时候，关系型数据库mysql就会因为join操作变着缓慢，而jsql的性能优势才会显示出来。因为实验室没有生产数据可以测试，本章只对简单的语句进行测试。

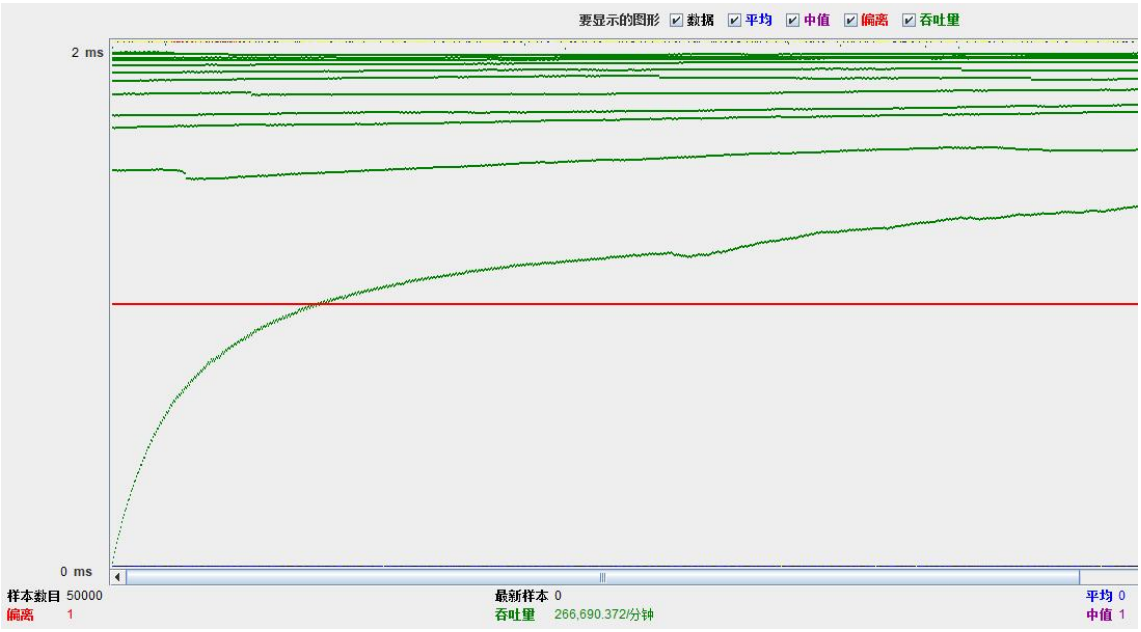


图 6-5 jsql的JMeter性能测试

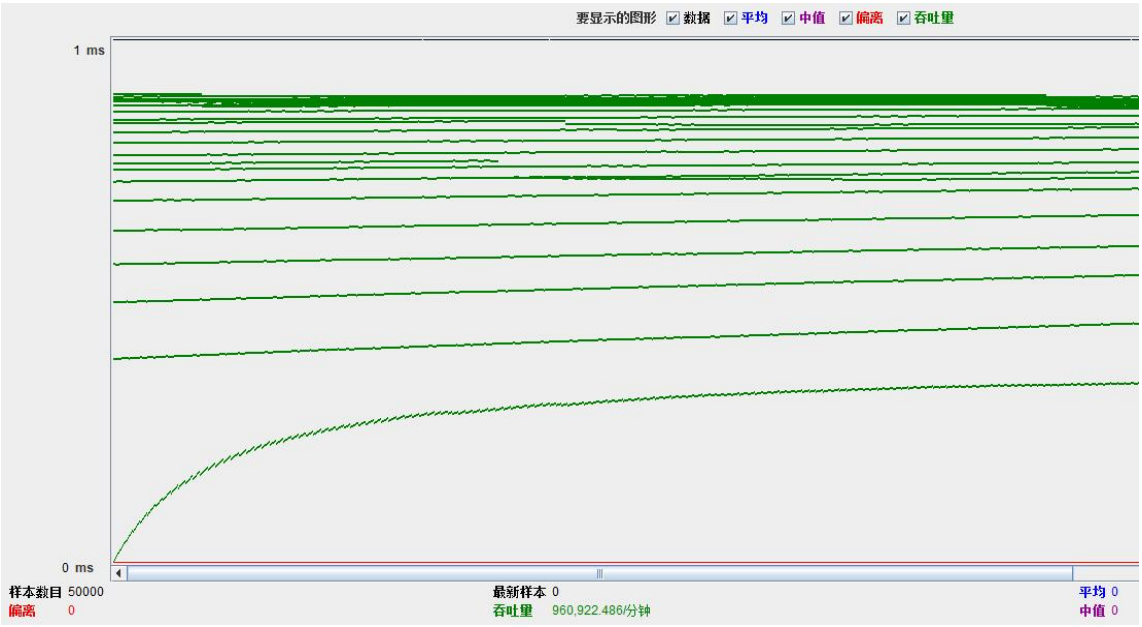


图 6-6 mysql的JMeter性能测试

6.5 测试对比

在这个部分论文选择和OceanBase进行对比，阿里巴巴OceanBase是淘宝开发的一个的分布式数据库系统，其实现了分布式的事务，支持大量数据的存储。是一个分布式的关系数据库系统。测试对比项目可以有很多，常用的有读写和更新速度，数据库存储占用空间容量。本次测试对比选择对JSQL和OceanBase的数据存储占用空间进行对比测试。在当今数据量越来越大的今天，相同数据记录占用的存储空间更小的数据库系统显然有其优势。本次测试的数据库表结构如表**??**所示。JSQL的测试电脑为实验室个人台式计算机，OceanBase因为没有开源，选择阿里云上的分布式数据库系统OceanBase。测试过程为逐渐往这两个数据库系统中写入数据，然后比较JSQL的本地磁盘占用量和阿里云的存储空间。

表 6-1 对比测试所用表结构

字段	类型	主键	备注
id	long	是	自增主键，只用作数据索引
createTimeYmd	date	否	创建日期，格式为yyyy-mm-dd
createTime	date	否	创建时间，格式为yyyy-mm-dd hh:mi:ss
createName	varchar(10)	否	创建者名字
xmlContent	text	否	测试文本，格式为字符数据
refKey	varchar(10)	否	外键关联，本测试为模拟关系数据库中的关系

测试结果如表6-2所示。随着记录数量的增多，数据库系统的存储空间自然也变得更多，当数据记录量达到18万的时候，JSQL存储量达到4.02G，而阿里云存储空间已经不够，在记录数据量继续增多的时候，JSQL还能继续存储，而阿里云已经没有空间写入数据。测试结果表明，在相同数据记录条数的时候，JSQL存储占用量明显低于阿里云分布式数据库OceanBase，在应用程序数据越来越多的今天，存储容量的降低无疑是JSQL相比OceanBase的优势。在测试中，之所以JSQL的存储量这么少，是因为在存储引擎层已经对数据进行了压缩处理，使得存储空间占用量降低。

表 6-2 数据空间占用对比

记录条数(行)	JSQL磁盘占用量	云OceanBase数据库占用量
5000	21.8M	183.51M
15000	47.9M	550.51M
65000	257M	2386M
18w	4.02GB	no more storage
26.8w	7.85GB	no more storage
677621	27.5GB	no more storage

6.6 本章小结

本章为系统的测试部分，在对系统的测试环境和测试方法介绍以后，对系统进行了功能测试和性能测试，功能测试包括数据库功能测试、集群功能测试以及审计功能的测试，其中数据库功能测试选择测试系统对结构化数据存储的测试。性能测试分为JDBC和JMeter测试两部分，对每个部分的测试结构进行分析。在本章后面对阿里巴巴的分布式数据库系统OceanBase和JSQL进行了存储空间占用的对比测试，测试结构表明，在选择压缩后的存储引擎以后，JSQL的存储空间占用明显低于OceanBase。

第七章 总结与展望

7.1 本文工作的总结

现在社会的发展，离不开各种各样的数据库系统。企业中常常出现的情况是，用不同的数据库系统来存储不同数据类型的数据，这显然浪费了资源，增加了人力维护成本。另一方面，单机数据库已经不能满足现在数据存储的需要。最后，在数据越来越重要的今天，从底层加入审计功能对一个数据库系统来说非常有必要，所以研究设计和实现一个分布式多模型安全数据库系统非常有意义。。因此，在导师的指导下，在研究生阶段作者主要做数据库有关的开发工作。最终，将理论知识和实践技能相结合，开发出了这套分布式多模型安全数据库系统。

分布式系统的研发是一个不小的挑战。作为这个系统的开发者，我首先在理论方面做了很多学习和研究，包括硬件软件有关的知识，和数据库事务相关的理论，以及各种分布式相关的协议。理论只能作为指导，而不能成为一个系统。在实现方面，我主要深入学习了JAVA语言，对多线程和高性能网络开发技能也进行了深入的学习。最终实现的这套数据库系统作者认为有如下优点：

1.论文所述系统完全采用JAVA语言编写，具有跨平台和安全的特点。

2.结合了非关系型数据库和关系型数据库的优点。

3.在系统的架构设计上，采用了面向对象的思想，对各个模块进行了很好的。划分，有利于对系统进行进一步的完善。

4.从数据库系统本身加入了审计功能。

5.采用分布式负载均衡算法，实现多主分布式架构，能够动态增加数据库节点。

在设计和实现这套系统的过程中，我的工作主要包括以下几点：

1.利用OrientDB非关系型存储引擎设计和实现了关系型数据库的本地存储系统。

2.实现了Mysql通信协议，使得可以通过Mysql客服端来连接JSQL分布式数据库，方便Mysql用户迁移到本数据库系统。

3.实现了对SQL语句的解析和执行，完成对关系数据库接口的支持。

4.实现了数据库的分布式架构，使得数据可以存储在多台计算机上面。

5.实现了系统的审计系统，使得系统的安全性得到增强。

6.设计和实现了分布式负载均衡算法，完成了分布式数据库节点的动态负载均衡功能。

7.在设计和开发完成系统以后，论文对系统进行了性能和功能测试，在系统存储空间占用上，拿它和阿里巴巴的OceanBase系统进行了对比。

7.2 未来的工作方向

作者设计和开发的分布式数据库系统，从数据存储的基本功能来看，其已基本达到要求。但是这个初步开发的系统还不能和生产环境下那些非常成熟的系统相比较；还有，作者一个人水平也有限，本系统难免还有很多需要完善的地方，总结起来有下面几个方面：

1.系统的数据恢复机制不够完善。虽然现在系统能够对数据进行存储，但是其不能很好的应对很多突发情况。在出现这些情况的时候，其数据又可能会丢失。在这样的问题下，论文在以后的工作中还要对其更好的设计，加入日志恢复功能。

2.分布式数据库中数据迁移目前还没有实现。现在有非常多的数据库系统，对于新的数据库系统来说，要想用户使用其系统，就必须要提供一个安全的迁移工具，使得其能将老的数据迁移到新的数据库系统上。论文后期工作可以加入对常用数据库系统的迁移工具。

3.关系型数据库大都有存储过程和触发器这些功能，本系统作为一个实验室产品，目前还没有完全的实现这些功能。虽然作为一个能存储结构化数据的关系型数据系统而言，这些都是应该有的功能。由于暂时时间有限，而且这些功能在一般情况下用的不多，所以系统目前就没有实现这些功能。在未来的工作中，我会把jsql当作一个关系型分布式数据库系统，进行对其关系型数据库功能进行完善。

4.缺少数据分区分片功能。对于现在的应用程序来说，很少有需要分区分片功能的场景。大多数应用程序只需要对读写性能进行扩展就可以满足要求。但是对于超大应用程序来说。还是要在分布式环境下实现分区分片。所以这个功能也是需要未来来完成的。

7.3 数据库系统的未来

数据库系统出现之前是用文件来存储数据的，其缺点是不能在不同的应用程序中共享数据，不同的应用程序存储文件的格式不同，就很难与其他应用程序分享数据。在数据库系统的发展过程中，出现了非常多的数据库系统。其中上世界发明的关系型数据库系统因为很多优点而一直沿用至今。在当时网络不是很快，移动设备不多的时代，关系型数据库系统能很好的满足企业的要求。

但是随着移动互联网的发展，出现了越来越多的移动设备，数据的增长数据已经远远超过我们的想象。在数据量增加的同时，数据类型也在变多。以前我们主要关注结构化数据的存储。现在出现了非常多的数据类型，就比如推荐引擎中的图数据类型，而关系数据库系统不能很好的存储这类数据，所以就需要不同的数据库系统。

企业需要对不同的数据进行存储和分析处理，就需要不同的数据库系统，这对系统资源是严重的浪费。新的数据库系统需要维护人员，这也增加了企业的人力成本。如果有一个数据库系统能够存储所有数据类型，而且其部署在分布式环境下，能够线性扩展其存储能力和读写能力，就能解决当前企业数据存储的所有问题。

把所有系统都存储在一个系统中也带来了非常大的危险，如果这个系统被攻击了，那么企业的所有数据就会收到损害或者丢失。这就是为什么本论文所设计的分布式系统实现了安全监控功能。

我认为，未来的数据库系统应该会朝着jsql的理想发展。理想的数据库系统应该是一个分布式的数据库系统，其能部署在全球所有地区，能够动态增加数据库节点，自动部署，最理想的情况下还应该能线性扩展其存储能力，能动态扩展其读写能力；理想的数据库系统也系统是一个多模型的数据库系统，其能存储所有数据类型的数据，包括结构化的订单数据，图片和视频等二进制数据，还能对其进行分析处理，产生价值，数据本身没有价值，只有对其进行分析和挖掘，才能找出价值；最后，理想的数据库系统应该是一个安全的数据库系统，其能自动备份，能够抵御自然灾害，能够抵御人为攻击。我们希望出现一个这样的理想的数据库系统，这需要所有工作者的艰苦努力。

致 谢

时间过的真快，转眼间，2年多的研究生生活就快要结束了。回顾自己的研究生学习生涯，感慨万千。论文的完成，除了自己的努力以外，更离不开老师和学弟们的帮助，在论文成稿之际，衷心感谢给予自己悉心指导和热情帮助的各位老师和学弟们。

论文的完成，首先要感谢我的校内导师曹晟老师，在整个论文的的写作过程中，曹老师都给予了我很大的关心和帮助。从作者毕业论文的选题、写作一直到最终完成的过程中，曹老师都是在百忙的工作中以一贯认真负责的态度认真仔细阅读作者的论文，给予作者耐心的指导，使得论文能够顺利的完成。他严肃的科学态度，严谨的治学精神，以及精益求精的工作作风，深深地感染和激励着我。

在这一年多的时间里，我还要感谢我的学弟们。感谢你们陪我一起开发这个分布式数据库系统，我们一起学习，一个努力，才能让本系统顺利完成。任何一个系统都要靠团队合作，更何况分布式系统这个既具有挑战性的工程项目，没有你们的帮助，就不可能按时完成这个系统。对学弟们的帮助，在此表示非常的感谢。

最后，作者非常感谢负责评审论文的教师、专家和教授，感谢你们认真负责的阅读论文，感谢你们为论文提出的宝贵意见和建议。

参考文献

[1] Kim W, 漆永新. 关系数据库系统[J]. 计算机科学, 1982, 1(4):13–24

[2] 黄贤立. NoSQL非关系型数据库的发展及应用初探[J]. 福建电脑, 2010, 26(7):30–30

[3] 潘群华, 吴秋云, 陈宏盛. 分布式数据库系统中数据一致性维护方法[J]. 计算机工程, 2002, 28(9):255–257

[4] 刘娜. 关系数据库事务操作及并发控制机制[J]. 电脑知识与技术, 2009, 5(11):2807–2808

[5] 王婉菲, 王欣, 张志浩. 数据库集群系统的研究与实施[J]. 微型电脑应用, 2003, 19(10):31–33

[6] 赵鑫. 键值数据库在云计算中的应用与实现[D]. 成都: 电子科技大学, 2015, 51–63

[7] 金天荣. 文档数据库与关系数据库研究[J]. 微计算机信息, 2008, 24(3):28+179–180

[8] 程耀东, 赵建昌, 徐军. 图形数据库应用技术研究[J]. 图学学报, 2006, 1(1):143–148

[9] 陈金水, 王鉴. 非结构化数据存储管理的实用化方法[J]. 计算机与现代化, 2006, 1(8):25–28

[10] 王珊. 数据库系统原理教程[M]. 北京: 清华大学出版社, 1998, 56–60

[11] 齐勇, 马莉. 分布式事务处理技术及其模型[J]. 计算机工程与应用, 2001, 37(9):60–62

[12] 王元卓, 靳小龙, 程学旗. 网络大数据:现状与展望[J]. 计算机学报, 2013, 36(6):1125–1138

[13] C. J. Date, H. Darwen. A guide to the SQL Standard: a user's guide to the standard relational language SQL[M]. The United States: Addison-Wesley, 1989, 10–20

[14] 薛军, 李增智, 王云岚. 负载均衡技术的发展[J]. 小型微型计算机系统, 2003, 24(12):2100–2103

[15] B. Eckel. Thinking in Java[M]. The United States: China Machine Press, 2007, 212

[16] 吴溥峰, 张玉清. 数据库安全综述[J]. 计算机工程, 2006, 32(12):85–88

[17] 黄志国. 数据库安全审计的研究[D]. 太原: 中北大学, 2006, 20–60

[18] 夏玉萍, 赵焕平, 张莉,等. 网格数据库技术的分析及应用[J]. 重庆理工大学学报, 2007, 21(12):98–101

[19] 左凤朝, 王文德. 面向对象数据模型的研究[J]. 计算机工程与应用, 2001, 37(16):110–112

[20] 邵佩英. 分布式数据库系统及其应用[M]. 北京: 科学出版社, 2005, 56–60

[21] J. Han, E. Haihong, G. Le, et al. Survey on NoSQL database[C]. IEEE, The United States, 2011, 363–366

[22] 孙建良. 分布式存储系统可用性与一致性研究[D]. 武汉: 华中科技大学, 2013, 20–60

[23] J. C. Corbett, J. Dean, M. Epstein, et al. Spanner: Google's Globally Distributed Database[J]. Acm Transactions on Computer Systems, 2013, 31(3):8

[24] Z. Yang. The architecture of OceanBase relational database system[J]. Journal of East China Normal University (Natural Science), 2014, 9(5):141–148

[25] T. Redkar. SQL Azure[M]. The United States: Apress, 2009, 505–584

[26] C. Tesoriero. Getting Started with OrientDB[M]. The United States: Packt Publishing Ltd, 2013, 101–114

[27] C. Gormley, Z. Tong. Elasticsearch: The Definitive Guide[M]. The United States: O'Reilly Media, Inc., 2015, 103–116

[28] M. Widenius. Mysql Reference Manual[M]. The United States: O'Reilly & Associates, Inc., 2002, 1–40

[29] M. Johns. Getting Started with Hazelcast[M]. The United States: Packt Publishing Ltd, 2013, 32–56

[30] 萧美阳, 叶晓俊. 并发控制实现方法的比较研究[J]. 计算机应用研究, 2006, 23(6):19–22

[31] 陈楠. 分布式数据库系统数据分布策略分析[J]. 计算机时代, 1998, 1(10):8–10

[32] P.-Å. Larson, S. Blanas, C. Diaconu, et al. High-performance concurrency control mechanisms for main-memory databases[J]. Proceedings of the VLDB Endowment, 2011, 5(4):298–309

[33] 别小凡. 分布式内存数据库的设计与实现[D]. 西安: 西安电子科技大学, 2013, 45–68

[34] R. Hitchens. Java Nio[M]. The United States: O'Reilly & Associates, Inc., 2002, págs. 452–499

[35] 王先兵, 唐旭章. 面向最终用户的原型化方法[J]. 计算机应用, 1997, 1(3):40–42

[36] N. Maurer, M. Wolfthal. Netty in Action[M]. The United States: Manning Publications, 2016, 92–108

[37] 安延文. 数据库审计系统中MySQL协议的研究与解析[D]. 北京: 华北电力大学(北京) 华北电力大学, 2016, 40–60

[38] 党永兴. 键值数据库存储引擎设计与实现[D]. 武汉: 华中科技大学, 2014, 45–68

[39] E. Betke, J. Kunkel. Real-Time I/O-Monitoring of HPC Applications with SIOX, Elasticsearch, Grafana and FUSE[J]. Springer, 2017, 1(12):174–186

[40] 石树刚. 关系数据库[M]. 北京: 清华大学出版社, 1993, 56–60