摘 要

随着互联网应用和计算机科学技术的快速发展，数据库逐渐成为信息系统的核心部分并广泛应用于企业、金融机构、政府及国防等各个领域。然而现在传统的关系型数据库在面对越来越庞大的数据时显示出了很多问题，不能对大量的数据进行高效的存储和处理。传统的关系型数据库系统已经不能满足现状移动互联网应用程序对存储的新的需求。为了应对新的数据存储需求，很多新的非关系型数据库系统出现了，OrientDB数据库就是其中非常出色的一种。目前现有的非关系数据库对大数据量数据的处理能力很强但是在在很多关键领域，传统关系数据库并不能被新的非关系型数据库代替。如何结合关系型数据库和非关系型数据库的优点成为当前数据库领域的关键问题。

在学习有关非关系型数据库和关系型数据库的知识以后，本文基于Java语言实现了一个分布式的关系型数据库JSQL，JSQL是一个将关系型数据库技术和分布式非关系型数据库相结合的新型数据库系统。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。分布式数据库集群节点主要包含五大模块，网络模块接收前端客服端的连接请求，对客服端进行认证和授权，并对连接进行管理，并实现基于Mysql的通信协议； Sql的解析和执行模块接收客户端的SQL请求，然后解析和执行数据库存储的调用，返回执行结果；审计模块是存储和分析所有对数据库的更改情况，向分布式管理节点提供审计数据；数据库引擎模块利用了非关系型Orientdb数据库引擎，实现可靠的数据存储；分布式模块利用hazlcast实现了数据库集群，为了解决分布式数据库集群数据操作不一致的问题，本论文提出了分布式多版本并发控制的方法。ＪＳＱＬ具有高可用性和可扩展性的特性，并实现了分布式动态负载均衡算法，同时从数据库底层考虑了数据库安全审计需求，加入了数据审计图形化界面显示审计结果。

论文对这个分布式数据库系统进行了功能和性能测试。通过系统的功能测试表明，分布式数据库JSQL的功能达到了最初的要求，审计系统的功能也达到本论文的要求。通过性能测试结果的分析，论文任务本系统达到了基本性能要求。但是本系统对复杂SQL语句和分布式事务方面的支持还不是很完善，最后提出了改进的方案。

关键词：分布式数据库，关系数据库，安全审计，Mysql，非关系型数据库

ABSTRACT

With the rapid development of Internet application and computer technology, the database has gradually become the core of information system Points and widely used in enterprises, financial institutions, government and national defense and other fields. However, the traditional relational database is facing increasing When exposed to a large number of data a lot of problems, can not be efficient and real-time processing. The new needs of the system Seeking, the traditional relational database can not be solved very well. Non-relational database is to make up for the lack of relational database, the OrientDB database Is a non-relational database. At present, the existing non-relational database has great ability to handle large amounts of data But in many key areas, relational databases are non-relational databases that can not be replaced. How to combine the advantages of relational databases and non-relational databases has become the key issue in the current database area.

Through the study of distributed theory, relational database theory and non-relational database technology, This article based on the Java language to achieve a distributed relational database JSQL, JSQL is a relational database technology, The combination of non-relational database technology and distributed technology. JSQL common client-server - server architecture, divided into customer service and server-side, Server-side is divided into distributed management nodes and distributed database nodes. Distributed management nodes to achieve a database server load balancing and monitoring and management functions. Distributed database node contains five major modules, The network module accepts the connection request of the front-end application, authenticates and authorizes the client-end, and manages the connection, And realize the communication protocol based on Mysql, so easy to migrate to Mysql users to the system; Sql parsing and execution module to accept the client's SQL request, and then parse and execute the database stored call, Return the execution result The audit module is to store and analyze all changes to the database, the distributed management node to provide audit data; The database engine module makes use of the non-relational OrientDB database engine for reliable data storage; Distributed modules use hazlcast to implement a database cluster. Based on the above functional modules, JSQL has high availability, scalability, load balancing and other characteristics, while taking into account the database security audit

needs from the bottom of the database, Joined the data audit graphical interface shows the audit results.

The thesis has carried on the function and the performance test to the system. The result of function test shows that the system conforms to the basic requirements of distributed database functionally, and the function of auditing system also meets the requirements of this dissertation. Papers through the performance test results analysis, that the performance of the system basically meet the requirements of this paper. However, the system of complex SQL statement support is not perfect, and finally proposed an improved program.

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

目 录

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看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。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。根据IDC的陈述显示，到2020年全球数据总量将超过40ZB，这个数据量将是2011年的22倍，在未来，全球数据总量还会以超过百分之五十的速度继续增长。

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。现代应用程序的数据一般都存储在数据库中，方便对数据进行操作，方便对数据进行共享。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。随着智能手机用户的快速增长，利用传统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的问题，出现了各种各样的非关系型数据库系统。在非关系型数据库中，数据库不再满足强一致性和事务要求，更加重视互联网集群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。在刚过去的几年，非关系型数据库得到了前所无有的发展，出现了各种各样的非关系型数据库系统。非关系型数据库是为了解决现在大数据时代非结构化数据存储的问题而产生的。

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，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，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，所以在分布式环境下，更加容易扩展，更加适应大数据的存储和处理。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。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；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(以下简称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，一旦JSQL开发和部署成功以后，将会给系统和软件平台带来以下好处：

1.结合关系数据库和NoSQL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Nosql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。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，提高数据库开发效率。

2.提高数据库系统资源使用效率

JSQL分布式数据库系统具有负载均衡功能，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。系统为客服端选择最近的节点，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无主服务器设计，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。而当数据库的存储需求降低时，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.内置数据库安全审计功能

数据的重要性越来越重要，如何保证数据库的安全性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，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。

1.2 数据库历史与现状

数据库技术在20世纪60年代开始出现。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，这就使得数据库的出现。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，对象数据模型和关系数据模型。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。从20世纪70年代到90年代开始，关系数据库的理论逐渐成熟并得到广泛应用，关系数据库发展的里程碑发送在1974年，IBM的研究员开发了SEQUEL结结构化查询语言，这样语言后来在1980年更名为SQL。SQL是关系数据库中的标准数据查询语言，在1986年，由美国国家标准学会规范SQL，就这样成为了数据库系统的标准语言。 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，CAP理论认为，在分布式数据库系统中，不可能同时满足CAP3个条件，这三个条件分别是一致性，可用性和分区容忍性。基于CAP理论，分布式数据库系统不可能同时满足一致性，可用性和分区容忍性。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，随着移动互联网的告诉发展对数据存储的需求提高，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。要满足大数据量的存储，必须要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，但是根据CAP理论，满足了可用性和分区容忍性，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 是一个可扩展的、全球分布式的数据库，是在谷歌公司设计、开发和部署的。

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是可以扩展到几百万个机器节点的全球分布式数据库，其跨越成百上千个数据中心，可以存储几万亿数据。

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。作为一个全球分布式关系数据库，Spanner 提供了很多有趣的特性。应用程序可以对存储在分布式数据库中的数据副本进行详细的配置。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，同时能设计，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是一个阿里巴巴开发的支持海量数据存储的高性能分布式数据库系统，能存储了数千亿条记录，支持数据上的跨行跨表事务，由阿里巴巴的淘宝各部门联合开发。在设计和实现OceanBase的时候阿里巴巴暂时抛弃了数据库系统中不紧急的功能，例如临时表和视图，阿里巴巴研发团队把有限的资源集中到

关键功能的开发商，当前 OceanBase主要解决数据更新一致性和高性能的跨表读事务等功能上。

OceanBase已经实现了商用，其应用于淘宝收藏夹应用程序中，用于存储淘宝用户的收藏条目和具体的商品和店铺信息，每天支持成千上万的更新操作。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 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提供了一个功能强大且为人熟知的存储结构，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，当用SQL Azure进行数据的增删改的时候， SQL Azure会自动地将数据备份到若干节点，以保证数据的高可用性;SQL Azure后台内置的集群机制完美保证了对自动故障转移的支持，而且无需人工监守。

1.3 论文的主要工作

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的关系型数据库JSQL ，JSQL结合了非关系型数据库OrientDB和关系型数据库的优点，同时实现了Mysql的通信协议，是一个兼容Mysqk通信协议的分布式数据库。在系统的实现过程中做的主要工作包括：

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.利用分布式多版本并发控制机制解决了分布式数据一致性问题。

8.数据库系统做完以后，对系统进行了各项软件测试，包括功能测试，性能测试等，以验证系统是否满足需求，达到了系统设计的目标。

1.4 本论文的结构安排

第一章是论文的绪论，在这一章中，介绍了论文的选题背景和本论文进行研究的意义。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说明，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库分布式和数据复制技术进行了介绍。最后介绍了一种新的分布式动态负载均衡算法和分布式MVCC技术，这些算法和技术将会在系统的实现了作为重要的理论基础。

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。从分布式数据库的使用者角度出发，对分布式数据库JSQL进行详细和切合实际的需求分析，在对系统进行需求分析以后，对分布式数据库的整体架构等进行模块划分。根据每个模块的作用，对每个模块的功能进行再划分。然后本章对重要的模块进行了分析，包括对各种技术和框架的选择，详细说明了实现分布式数据库系统需要用到的各种技术。

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。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分析。

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，重点对分布式管理节点和分布式数据库节点功能的实现进行阐述，对论文中使用到的关键算法和技术，用流程图对其进行说明。

第六章，作为系统的测试部分，本章对论文中的jsq分布式数据库进行了测试，测试包括功能测试和性能测试。

第七章，论文总结了论文所做的工作，对本系统的优点和不足进行了说明。并对未来的工作进行了展望。

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数据库有关的理论知识，以及本系统所需要用到的关键算法和技术。

2.1 数据库系统

2.1.1 数据库系统概念

数据库管理系统是一种管理软件，数据库管理员通过管理数据库管理对数据库进行具体的操作和维护操作。在数据库管理系统的统一管理和控制访问下，才能保证数据库中数据的一致性和完整性。数据库的用户只有通过数据库管理系统才能访问其中的数据，通过数据库管理系统，数据库管理员还能对其进行权限管理等安全维护操作。通过数据库管理系统，不同的应用程序和用户，可以安全的同时对其中存储的数据进行安全的访问，而且能保证数据的安全性和一致性。几乎所有数据库管理系统都提供对数据的追加、删除等基本操作。数据库管理系统是管理和操作数据库的核心软件，位于操作系统之上。数据库管理系统提供了一个抽象，使得上层应用程序和用户能逻辑的操作存储在计算机上的物理数据。有了数据库管理系统，数据库用户和管理员就只需要对其进行操作，对其发送命令就能处理具体的数据，而不能担心数据在计算机中具体的存储格式，也不能担心数据库在计算机系统的安全问题。

2.1.2 关系数据库系统

2.1.2.1 关系模型

关系模型是1969年被提出的用于数据库存储的数据模型，关系模型基于严格的数学理论，有严格的理论基础作为指导。在关系模型中，所有的数据都被表示成数学上的关系，关系模型采用表的结构化存储来表示和存储数据和数据之间的关系。可以通过关系模型的规范化对关系模型中的数据进行简化，对建立一致性的数据存储有重要的作用。

在关系模型中，每种数据属于一种数据类型，也可以说是一种域。元组是属性的的结合，具体的理解中，可以理解为一条记录。而属性属于元组的一个元素，是域和值的有序对。域和名字的有序对的集合就是关系变量，关系变量是关系的表头。关系是元组的集合，也就是多条记录组成关系。尽管这些概念被严格的在

数学上定义，但是在关系数据库中，可以用更加容易理解的可视化的方式来理解。在这种可视化的理解下，表就是关系，关系就是表。而关系理论中的元组类似于关系数据库中行的概念。信息原理是关系模型的基本原理，在关系模型中，所有信息都表示为关系中的数据值。关系模型本身是没有关系的，然而，设计者通过在多个关系变量中使用相同的域名来模拟关系。也就是说，如果一个属性依赖于另一个属性，则在关系模型中通过参照完整性来强制这种依赖性。

关系数据库系统是建立在关系数据模型之上的软件系统，其通过数据中的代数等知识对数据进行操作，在关系数据库中，关系表现为一个表，该形式的表格作用的实质是存储具体的关系实体。这些表格通过属性能产生不同的关系类型，通过属性之间的关系，来模拟每个关系的变化。在关系数据库系统中，每个表格代表不同的数据类型，存储不同的数据。每行数据包含一个唯一的数据实体，代表一个表的实例，每个数据行则有不同的属性，这些属性通过列来表示。当定义一个关系表的时候，你能定义关系表上有哪些数据元素以及每种数据元素的类型。

在关系数据库中，通过关系模型对其数据进行严格的定义和操作，用户通过关系型数据库的操作需要，而不需要了解关系数据的具体存储格式对其进行操作。关系型数据库不但能根据关系模型存储结构化的关系数据，而且能保证存储数据的一致性，能支持数据的事务操作。

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性。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，不能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，每个步骤要不全部完成，要不全部失败，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

在计算机和数据库领域中，特别是在计算机编程，操作系统，关系数据库领域，并发控制确保并发操作有正确的执行结果。在关系型数据库系统中，管理软件由不同的组件组成。每个组件被设计为正确地操作，即符合某些一致性规则。在数据库操作中，一个组件可能影响另一个组件。一个事务可能影响另一个事务，从而出现不一致性甚至错误的结果。并发控制技术能保证数据的一致性，但是，并发控制引入了需要大量额外的复杂性，并发算法的执行需要开销。并发控制失败可能导致数据损坏，从而导致关系型数据库存储数据的失败。

在关系型数据库中，并发控制是正确性的必须要素。并发控制是任何系统中的正确性的基本要素，在关系型数据库中，不同的事务执行过程中，可以访问相同的数据，从而产生不正确的结果。在关系型数据库系统中，出现了很多并发控制技术来预防这样的不正确性。下面是并发控制的主要方法：

1.锁定

通过锁定一个事务操作的对象，防止其他事务的访问。就能保证不同事务之间的正确执行而不能相互干扰，从而防止出现不正确的并发错误。

2.串行化

检查计划图中的周期并通过中止来破坏它们。

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。

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。

5.多分支并发控制

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。

6.索引并发控制

将访问操作同步到索引，而不是用户数据。专业方法提供了显着的性能提升。

每个事务都为其访问的数据维护一个私有工作空间，只有在事务提交之后，其更改的数据才会在事务外部变得可见。这种模式在许多情况下提供了不同的并发控制行为，并带来好处。

2.1.4 非关系型数据库系统

随着移动互联网网站的快速增长，传统的关系数据库在应付移动互联网网站，特别是超大规模和高并发的动态网站就显得力不从心，出现了很多难以克服的问题，所以就出现了很多非关系型数据系统。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。非关系型数据库的产生就主要是为了解决大规模数据的存储问题，特别是非结构化数据的存储问题。大数据量的存储需求给非关系型数据库带了新的发展机遇。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。虽然早起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，但是他们都普遍存在下面一些共同特征：

1.不需要预定义模式：在非关系型数据库系统中，不需要事先定义数据模式，也不需要预定义表结构。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：非关系型数据库并不是将数据存放于同一个节点，非关系型数据库需要将数据进行分片分区，将记录分散在季芹中的多个节点上面。并且通常分区的同时还要做数据复制。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。因为这样，数据可以尽快地写入一个节点，不会被网络传输引起迟延。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，这样的方式在出现季芹故障的时候，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 OrientDB介绍

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

分布式数据库是一个数据库，其中存储设备没有全部连接到一个共同的处理器。它可以存储在位于相同物理位置的多台计算机中; 或者可以分散在互连计算机的网络上。与其中处理器紧密耦合并构成单个数据库系统的并行系统不同，分布式数据库系统由分布式在不同的分布式集群计算机组成，没有共享共同的物理组件。

2.2.2 分布式数据库的特点

1.数据独立性

在分布式数据库系统中，其中的每个分布式系统节点都具有数据独立性。分布式数据库独立性包括数据的逻辑独立性以及物理独立性，逻辑独立性是指数据库用户的应用程序与分布式数据库的逻辑结构是彼此相互独立的，也就是说，当数据的逻辑结构发送改变时，数据库用户程序可以保持不变；分布式数据库的物

理独立性是指数据库用户的应用程序和存储在磁盘上的数据库中数据是相互独立的。也就是说，数据在磁盘上怎样存储由分布式数据库系统来管理，用户程序就不需要了解怎么样存储，这样当数据的物理存储改变的时候，应用程序就不用改变，保证了应用程序的迁移性和可用性。

2.分布透明性

分布式数据库的分布透明性是指，分布式数据库的用户不需要关心数据是怎么分布的，不需要关心数据存储在什么地方，逻辑上，不需要知道数据库的逻辑数据模型，在物理上，不需要知道分布式系统的物理架构。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是，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价格。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

在分布式数据库系统中，每个分布式集群中的单个数据库功能和整个分布式数据库功能是一样的，用户不需要知道其连接的是一个单个数据库还是分布式集群。分布式数据库的管理可以用统一的方式对分布式数据库节点进行管理。每个节点都像单独的数据库一样，能够自己管理其本地的数据存储

4.复制透明性

分布式数据库系统的用户不用关心数据库在集群中各个节点的复制情况。分布式数据库中的数据复制，一般更新都由分布式数据库系统自动完成，在分布式数据库管理系统中，可以透明的把一个节点的数据复制到其他节点存放，，应用程序不需要关心这些变化，应用程序可以使用复制到本地的数据节点在本地完成分布式数据库有关的操作，避免通过网络传输数据，这样就能提高分布式数据库系统的整体效率和性能。对于分布式数据库数据的更新操作，其要在所有的分布式数据库节点传播，这同样对分布式数据库的用户透明。

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。如果服分布式数据库系统软件支持透明的扩展，那么就可以在不影响数据库用户的情况下，动态的增加多个服务器来进一步扩展分布式系统的性能，同时也能根据当前系统的情况。减少系统节点，减少系统的资源浪费。

2.2.3 数据分布和负载均衡

分布式系统如何拆解输入数据，将数据分发到不同的机器中。下面将介绍几种不同的数据分布方式。

2.2.3.1 哈希分布

哈希分布方式可能是最分布式数据库系统中最常见的数据分布方式,其方法就是是按照数据的某一元素特征计算计算机哈希值, 然后将哈希值与分布式机器中的机器机器建立一种映射关系,从而将具有不同哈希值的数据分布存储到不同的机器上。只要哈希函数选择的好，分布式数据库库的数据就能均匀到分发到集群中不同机器节点中。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。针对哈希分布方式扩展性差的问题,一种思路是不再简单的将哈希值与分布式集群中机器数做除法取模映射,而是将哈希和集群集群的对应关系作为元数据由专门的元数据服务器存储管理。访问数据时,首先计算哈希值并查询元数据服务器,获得该哈希值对应的分布式机器节点。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

在分布式系统中，按数据范围分布是另一个常见的数据分布方式, 将数据按关键字的范围划分成不同的区间,使得分布式集群中每台服务器节点处理不同区间的部分数据。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。在实际分布式系统中,我们一般也不按照某一维度划分所有数据范围,而是使用全部数据划分范围, 这样就能避免数据倾斜的问题。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。另外,当集群需要扩容时,也可以随意添加机器,不需要迁移数据，而且不限为倍增的方式, 只需将原机器上的部分数据分区迁移到新加入的机器上就可以完成集群扩容。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。经常系统的运行，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

复制是分布式数据库系统中保证数据安全的唯一手段。在分布式数据库系统中，复制可以分为同步复制和异步复制，两者的区别在于前者需要等待所有分布式节点中副本返回写入确认，而后者只需要一个返回确认即可。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，不同你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。

第一种保证副本数据一致性的方式叫主从复制。主从之间可以是异步复制，也可以是同步复制。例如MySQL系统中，在默认情况下采用异步复制，但是异步复制容易引起数据丢失的问题，比如主从结构中，当主节点的写入请求还没有复制到从节点的时候，主节点就就挂掉了，这样当从节点被选为新的主节点之后，主节点在这之前还没有同步的数据就会被丢失。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。

除了主从复制，另一种保证数据一致性的复制方式口水引入分区一致性算法Paxos，又叫复制状态机。复制状态机在分布式数据库开发的很多例子都可以遇到，比如， Google Spanner在单分区内就采用了这样的设计。复制状态机主要用于满足复制的两点需求，第一，客户端在面对任何一个副本时都需要具备完全一致的访问行为；第二，每个副本在执行请求时都需要按照完全一致的顺序来进行。

2.2.4.2 一致性和可用性

在分布式数据库理论中CAP定理，又被称作布鲁尔定理，有特别重要的作用，它指出对于一个分布式数据库系统来说，不可能同时满足以下三点：

1.一致性

2.可用性

3.容忍网络分区

根据定理，分布式系统最多只能满足三项中的两项而不能满足全部三项。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。

2.3 算法和技术

2.3.1 分布式负载均衡算法

负载均衡是按照一种的负载均衡算法，分发网络和计算机处理资源。提供一种均衡利用分布式系统资源的方式。提高系统的数据处理能力。避免单点故障，避免资源的浪费。

负载均衡算法包括静态负载平衡以及动态负载平衡。只是根据一定的静态信息来实现的算法就是静态负载均衡算法；考虑每个分布式节点的处理能力和每个节点当前的负载信息的算法，是一种动态的负载均衡算法。动态算法能够根据系统当前的状态，选择更加合适的分布式节点，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；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。

论文提出一种新的分布式负载均衡算法，其为后面系统负载均衡功能的实现提供了详细的理论和架构指导。下面对分布式负载均衡算法的步骤进行详细的说明。

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。

$$P = L_c * L_r \tag{2-1}$$

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。

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

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。

第五步，遍历数据库的元数据库，依次检查其中的每个服务器节点，检查其当前的负载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。

图2-1示出了负载均衡算法的一个例子。首先，我们给出4个分布式服务器节点$VM1$到VM4。根据每个节点的CPU和内存情况，计算出其权值，然后排序出一个圆。第一次，我们选择权值最大的节点VM1，检查其当前负载，并不是1，所以返回服务器VM1的地址给客户端，算法结束。第二次算法开始的时候，从节点VM2开始检查，返回VM2的地址，算法结束。依次类推。直到找到一个服务器地址为止。该算法考虑了基于CPU，RAM的每个服务器的基本负载。时间复杂度

为线性的，复杂度较低。采用的分布式的负载均衡算法，同时也避免了单点故障问题，为后面的负载均衡功能实现提供了算法理论基础。

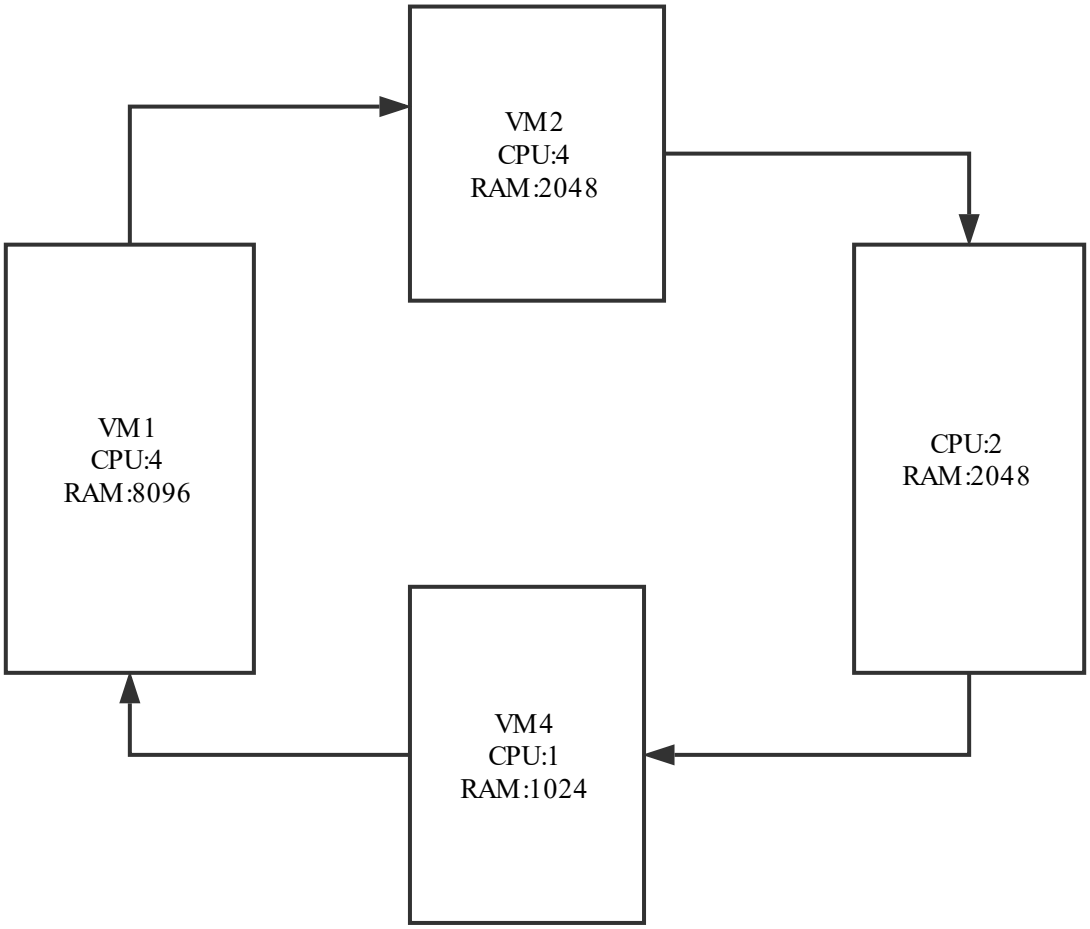


图 2-1 负载均衡示意图

2.3.2 副本和分布式MVCC技术

2.3.2.1 副本的概念

副本指在分布式数据库系统中为数据提供的冗余副本。对于数据副本指在不同的节点上持久化同一份数据，当出现某一个节点的存储的数据丢失时，可以从其他副本上读到数据。数据副本是分布式系统解决数据丢失异常的唯一手段。

2.3.2.2 副本一致性

分布式数据库系统通过副本控制协议，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。副本一致性是针对分布式数据库系统而言的，不是针对某一个副本而言。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。

2.3.2.3 分布式 MVCC

分布式 MVCC技术不但能解决分布式副本一致性问题，还能实现分布式事务。假设在一个分布式数据库系统中，数据更新操作以单个事务进行，每个事务包括若干个对不同节点的更新操作。更新事务必须具有原子性，即事务中的所有更新操作要么同时在各个节点生效，要么都不生效。假设不存在并发的事务，即上一个事务成功提交后才进行下一个事务。

基于 MVCC 的分布式事务一致性解决的方法是：为每个分布式事务分配一个递增的事务编号，这个编号同时也代表了数据的版本号。当事务在各个节点上执行时，各个节点还需记录更新操作及事务编号，当事务在各个节点都完成后，在全局元信息中记录最近事务的编号。在读取数据时，先读取元信息中已成功的最大事务编号，只读取更新操作编号小于等于最后最大已成功提交事务编号的操作，并将这些操作应用到基础数据形成读取结果。在删除数据的时候，并不真的删除数据，而只是做一个标记，记录当前删除事务的事务编号。读取数据的时候，当检查下删除编辑，就跳过这个记录。

2.4 本章小结

本节介绍数据库有关的理论知识。包括硬件知识和数据库事务相关理论，以及分布式系统系统有关的算法和协议。另外，本章最后讨论了目前几种成熟的分布式的系统，作为实例学习和学习研究。

第三章 系统分析

系统需求分析是软件开发过程的开始阶段，是软件生命周期中的一个重要环节，对于整个软件开发过程以及软件产品的质量是至关重要的，因为构建软件系统最艰难的一个部分就是准确的决定要构建什么，包括所有对人的界面、对机器的接口、以及对其他软件系统的接口的需求都需要详细的记录下来。如果做错了会对最终系统造成相当大的损害，并且后期难以调整。而后面的设计只是是将问题转换为解决方案的的创造性过程，系统设计是在系统分析的基础上研究系统如何实现需求中所描述的功能，给出满足需求的解决方案，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进行需求分析，需求分析主要包括整个系统的需求分析与功能分析，同时也包括了对各个功能模块的实现分析。

3.1 系统需求分析

需求阶段的目标是理解客户的问题和需要，需求分析是在综合分析用户对系统提出的需求的基础上，构造一个能表达用户需求的需求说明，并以“软件需求规格说明书”的形式记录下本阶段工作的结果，为下一阶段的软件设计提供设计基础。

3.1.1 系统功能需求

本论文所研究的分布式数据库系统，主要对大数据量进行高效处理，需要系统具备大容量存储和高速率运算。传统的关系数据库在移动互联网大数据量存储的要求下，出现了很多的问题，而现在的各种NOSQL数据虽然支持大量数据的存储，但是对关系的操作又很少。因此，本文结合关系型技术与分布式NOSQL技术，来设计能够对大量数据进行高效处理的分布式数据库系统。作为项目的前期工作，本文所设计的分布式数据库系统的主要功能如下：

1.提供负载均衡功能

本文所述的分布式系统，为了实现分布式系统中每个数据库节点的资源的合理分配，需要系统具有良好的负载均衡功能，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，其为客服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，为客服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

站在客户的角度，需要本系统能够提供增加、删除、修改、查询等数据库基本功能操作。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。

因此，基于对整个分布式数据库系统的功能需求的分析，系统可分为两部分，提供负载均衡功能和管理监控功能的分布式管理节点，提供增加、删除、修改、查询等数据库基本功能操作的数据库节点。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。

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。

3.数据库的运行管理：分布式数据库的运行管理功能是系统的运行控制和管理功能，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：分布式数据库系统要分类组织数据，包括数据字典等，需确定以何种文件结构和存取方式在存储级上组织这些数据。数据组织和存储的基本目标是提高存储空间的利用率，选择合适的存取方法提高存取效率。

5.数据库的保护：数据库中的数据是信息社会的战略资源，所以数据的保护至关重要。系统对数据库的保护通过4个方面来实现：数据库的恢复、数据库的并发控制、数据库的完整性控制、数据库安全性控制。

6.数据库的维护：这一部分包括数据库的重组合重构以及性能监控等功能，这些功能分别由各个使用程序来完成。

3.1.2 系统功能用例

用例图用于描述一组用例、参与者及它们之间的连接关系。一个用例描述了一组动作序列，每一个序列表示系统的外部设施（系统的参与者）与系统本身的交互。本系统的用例图是从参与者使用系统的角度来描述系统中的信息，即站在系统外部查看系统应该具有何种的功能，而不是描述功能在系统内是如何实现的。本文设计的分布式数据库系统的主要功能为新增数据、查询数据、修改数据、删除数据。系统总体的用例图如图3-1所示。

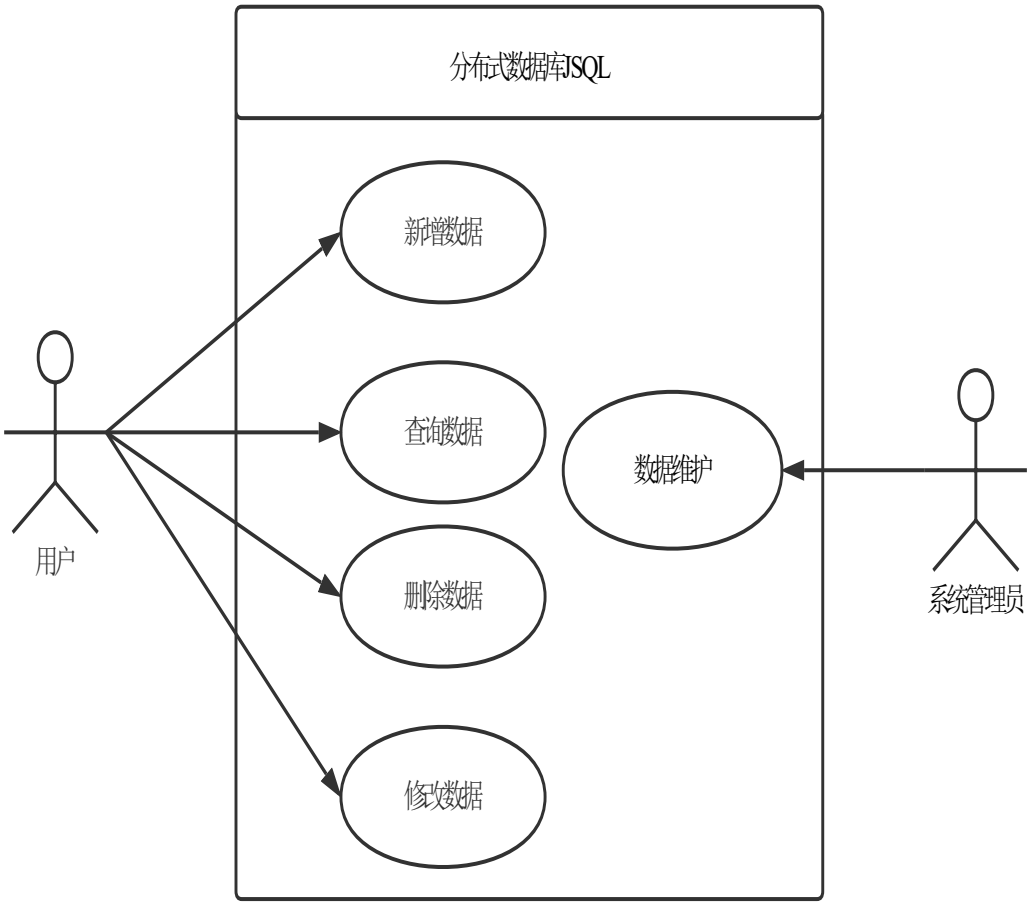


图 3-1 系统用例图

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，具体用例见表3-1。

2.数据查询是指利用用户在客户端上输入指定数据标识作为关键字，查询该条数据的基本信息，具体用例见表3-2。

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

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.删除数据是指当数据无效时，客户端发送删除信息请求分布式数据库系统删除该条数据，具体用例见表3-3。

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。

4.修改数据是指当数据需要修正时，管理员通过客户端发送修改消息请求，通知分布式数据库系统存入新数据，刷新数据变化部分，具体用例见表3-4。

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，可用实现所有主流平台的数据库部署。

2.JAVA是企业开发的首选语言，其安全性比本地语言更高。

3.非关系型数据库引擎OrienDB也是用JAVA开发的，为了方便调用。本系统也应该使用同一种语言。

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

所有的服务软件都需要实现网络模块，这样才能连接客服端的请求。JSQL用java语言开发，主要用到的是java的网络开发模块。

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[1]。

在JDK1.4出来之前，我们建立网络连接的时候采用同步阻塞的模式，首先需要先在服务端启动一个ServerSocket，然后在客户端启动Socket来连接服务端进行通信，默认情况下服务端需要对每个请求建立一个单独线程等待客户端请求，而客户端发送请求后，先咨询服务端是否能接受请求，如果不能的话则会一直等待或者遭到连接拒绝请求，如果可以的话，客户端才会连接服务器。

NIO本身是基于事件驱动的思想来完成的，其主要想解决同步阻塞的问题：在使用同步阻塞的网络应用中，如果要同时处理多个客户端请求，就必须使用多线程来处理。也就是说，将每一个客户端请求分配给一个单独线程来单独处理。这样做虽然可以达到我们的要求，但这同时又会带来另外一个问题。由于每创建一个线程，就要为这个线程分配一定的内存空间，而且操作系统本身也对线程的总数有一定的限制。如果客户端的连接请求过多，服务端程序可能会因为不堪重负而拒绝客户端的请求，甚至服务器可能会因此而崩溃。

Netty的主要目的是建立基于NIO的高性能协议服务器，分离和松散耦合网络和业务逻辑组件。它可能会实现一个广为人知的协议，如HTTP或您自己的特定协议。 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相比，这导致高吞吐量。 Netty使用事件驱动的应用程序范例，因此数据处理的流水线是一系列事件处理程序。

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，它支持高并发的客服端访问。

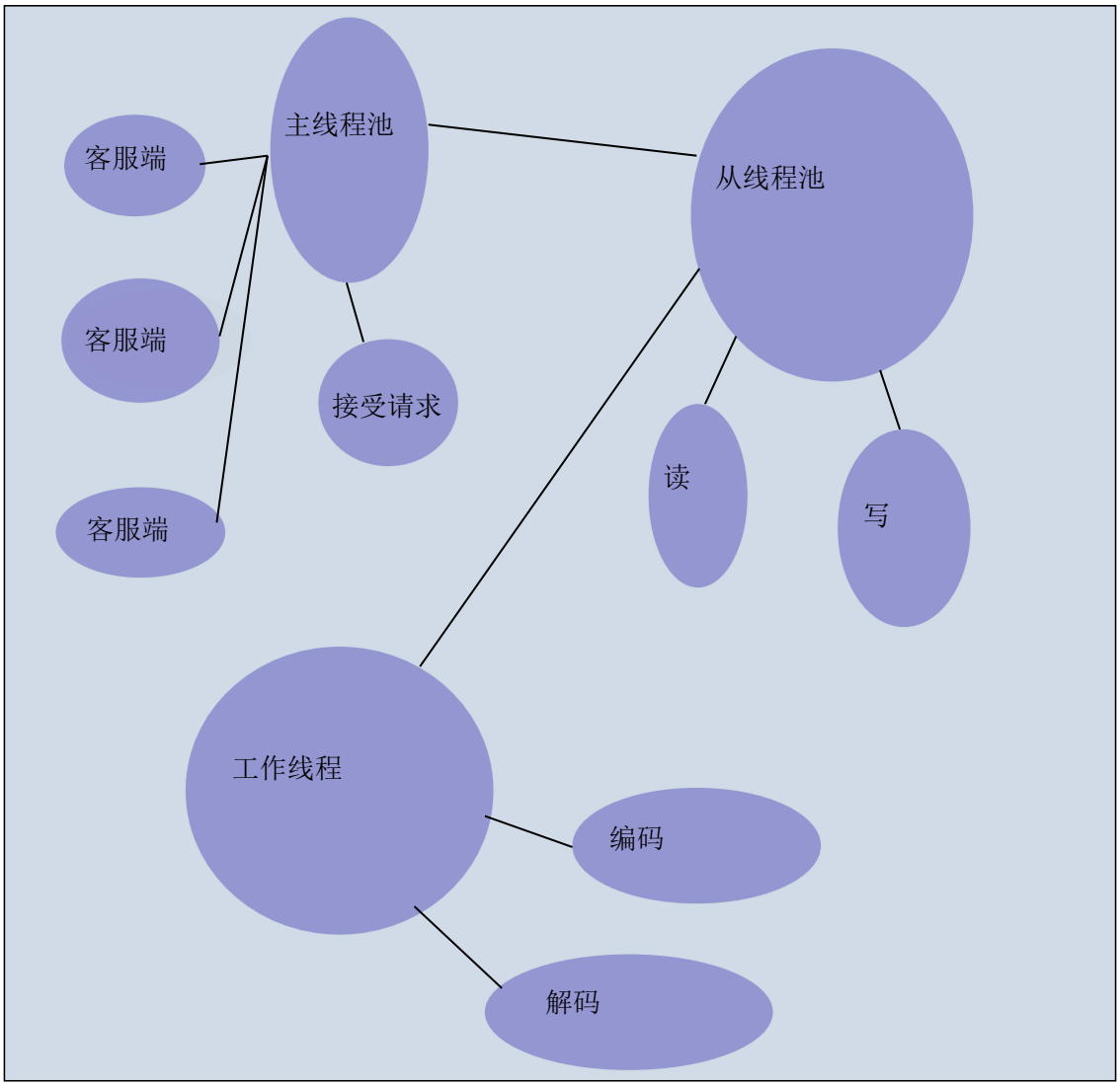


图 3-2 Netty

3.2.3 通信协议分析

每个服务器和客服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。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。当一个客户端的连接请求到达，就会执行握手和权限验证阶段。如果验证成功，会话开始。然后，客户端发送消息，服务器会以一个适合该发送命令的数据类型的数据包或者一条消息进行回复。当客户端发送完成后，会发送一个特殊的命令包，告诉服务器已发送，然后会话结束。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

握手认证阶段在客户端与服务器建立连接后立即进行，交互过程如下：

(a) 服务器发送给客户端握手初始化报文。

(b) 客服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

客户端和服务器认证成功后，就会进入命令执行阶段，交互过程如下：

(a) 客户端发送服务器执行命令报文。

(b) 服务器发送给客服端命令执行结果。

MySQL客户端和服务器之间完整的交互过程如图4-8所示。

3.2.4 SQL实现分析

结构化查询语言是关系型数据库操作的标准语言。不过现在各种关系型数据库系统都对SQL规范作了某些编改和扩充。所以，实际上不同关系型数据库系统之间的SQL语言不能完全相互通用，甚至不同版本间之间的语言也可能无法互通。

SQL是一种高级数据库操作和编码语言，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，只需要说明操作结果，不需要说明具体操作的执行过程。数据库服务器会选择合适的执行过程。这样每个数据库服务器就能实现不同的操作过程，实现不同你的性能。

Mysql支持的sql语句和标准的sql语句不全一样，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

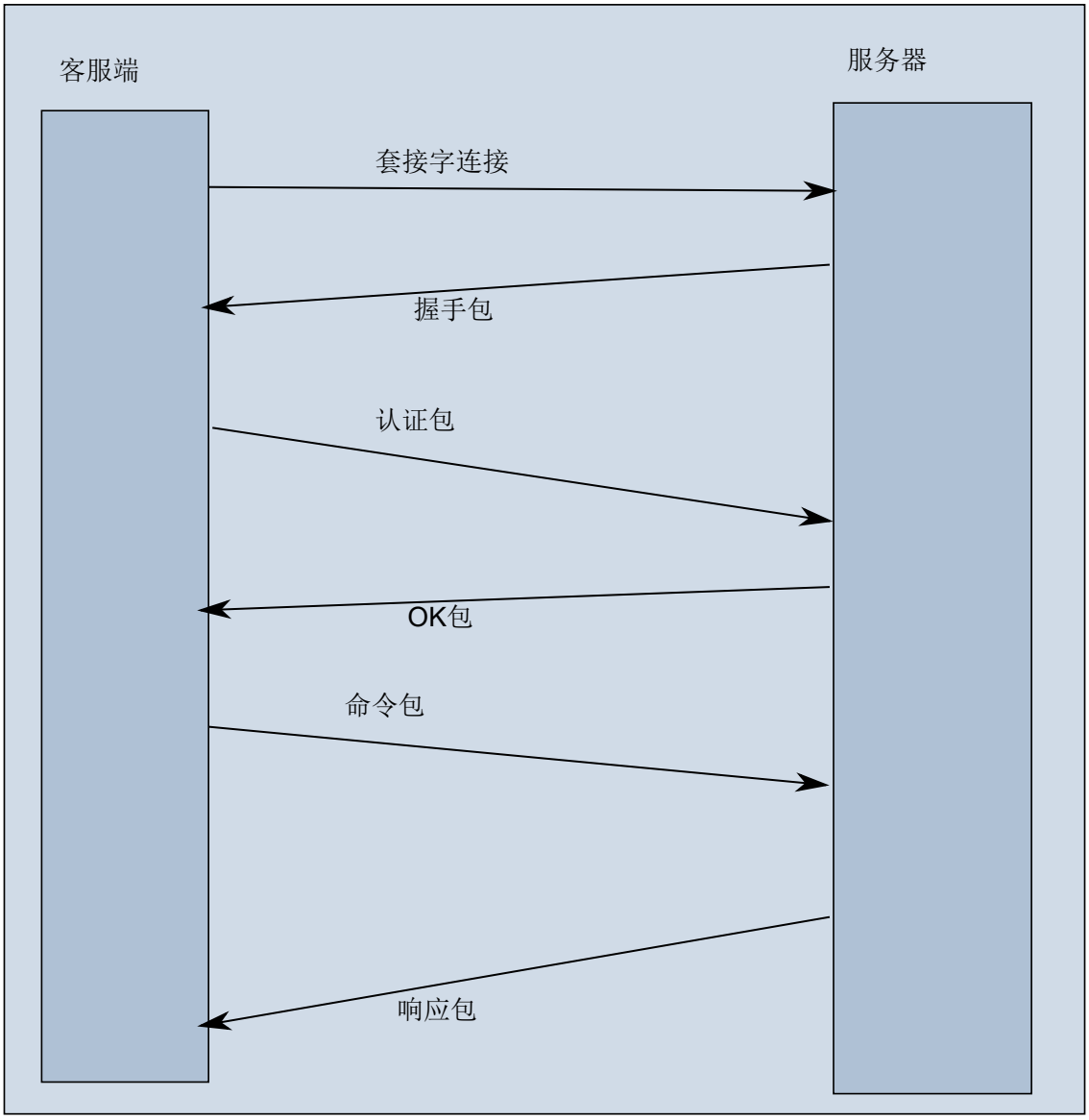


图 3-3 协议交互流程图

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。

3.2.5 存储引擎分析

存储引擎是分布式存储系统的发动机，直接影响分布式数据库系统的性能和功能。存储系统的基本功能包括增加删除数据等，其中，在数据库引擎中，数据读取操作又分为随机读取和顺序扫描。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。

3.2.6 分布式实现分析

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自己开发分布式功能。

Hazelcast是一个开源的内存数据网格，提供了一些引人注目的功能，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

该功能使Hazelcast成为应用程序中的多用途工具。它可以用于简单的消息传递，缓存，键/值存储，任务协调服务。与之前描述的其他一些服务不同，Hazelcast除了分布式配置和协调之外，还可以用来解决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，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。Hazelcast任务协调解决方案的一般概念是将信号/锁定义存储在内存中，并根据节点名称，区域名称和可用许可证数量等标准，协调信标许可证的分配。为了处理从群集中退出的节点，信号量允许基于支持时间的到期，除了明确的释放。所有的集群，一致性和可用性要求将被授予Hazelcast内部，使得Hazelcast解决方案尽可能简单。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 监控功能实现分析

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

本章对分布式数据库系统进行需求分析。首先，站在用户的角度对系统的需求进行具体的分析，描述系统的功能需求，并结合系统用例图给出功能用例。然后本章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

在软件工程中，需求分析之后就是设计阶段。首先，开发者需要对软件系统进行概要设计，即总体和框架设计。概要设计需要对软件系统的总体功能进行设计，包括系统的基本处理流程、系统的组织结构和功能模块划分。，为软件的详细设计提供基础。在概要设计的基础上，然后需要进行软件系统的详细设计。在系统的详细设计中，描述实现具体模块所涉及到的主要算法、数据结构和功能流程，需要说明软件系统各个层次中的每一个程序的设计考虑，方便以后的编码和测试工作。应当保证软件的需求完全分配给整个软件。详细设计应当足够详细，应该能够根据详细设计进行具体的编码。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，原型化过程是迭代的过程：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是作为实验室数据库，并没有做实际上线的考虑，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。重点针对分布式数据库系统的大容量可扩展性进行设计，主要完成数据库的增加、查询、修改、删除等功能，所以对系统的架构作了简化调整。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.客服端节点

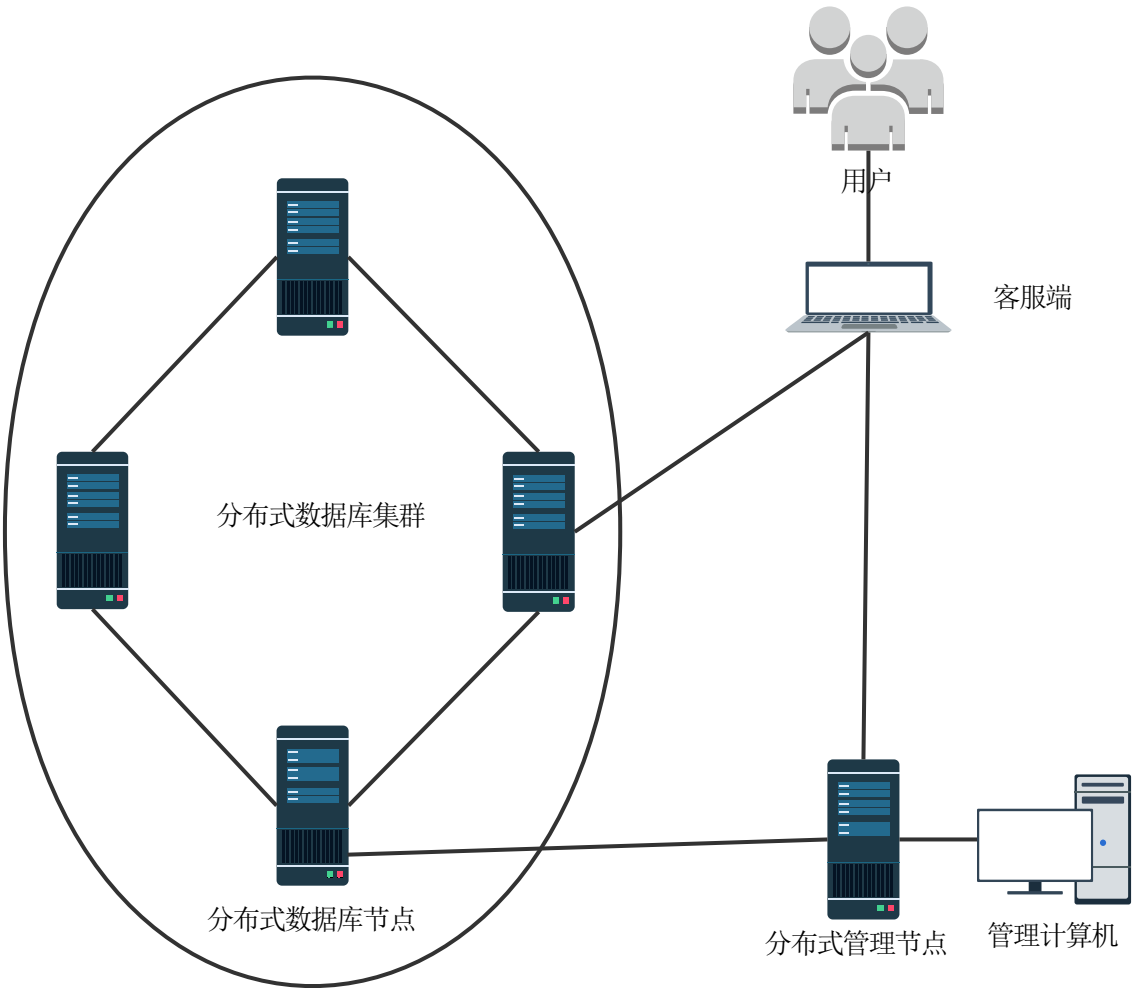


图 4-1 系统网络拓扑结构图

客服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，客服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客服端。如果是用Mysql客服端，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客服端的时候，客服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。这样当客服端请求到来的适合，分布式管理节点就可以返回适合的分布式数据库节点，从而达到负载均衡的效果。

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，响应客服端的增删改查的命令请求。当客服端连接分布式数据库节点以后，会检查客服端的权限，只有有权限的用户才能执行命令。

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和负载均衡算法。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，客户端模拟负责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。得到地址以后，客服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参数，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。根据原型化模型和系统的网络拓扑结构图，本文所涉及的分布式数据库系统的整体结构是：有两种计算机，一台作为客户端使用，另一台作为服务器使用，根据需求设定服务器端的数据库节点数个数N。这N存数据库之间相互独立、存储各自的数据信息。它们的数据结构、存储方式都是相同的，由服务器端的分布式管理节点负责把客户端发来的数据分发到对应的数据库节点，因此不同的数据库节点中存放着不同的数据。每一个数据库节点构成了自己的数据库单元，每个数据库单元具有独立处理的能力，它可以执行局部应用。同时，如果这 N 个数据库物理上应用于多台计算机时，每个数据库节点也能通过网络通信子系统执行全局应用。本系统的架构图如图4-2所示。 本系统中，客户端模拟真实终端设

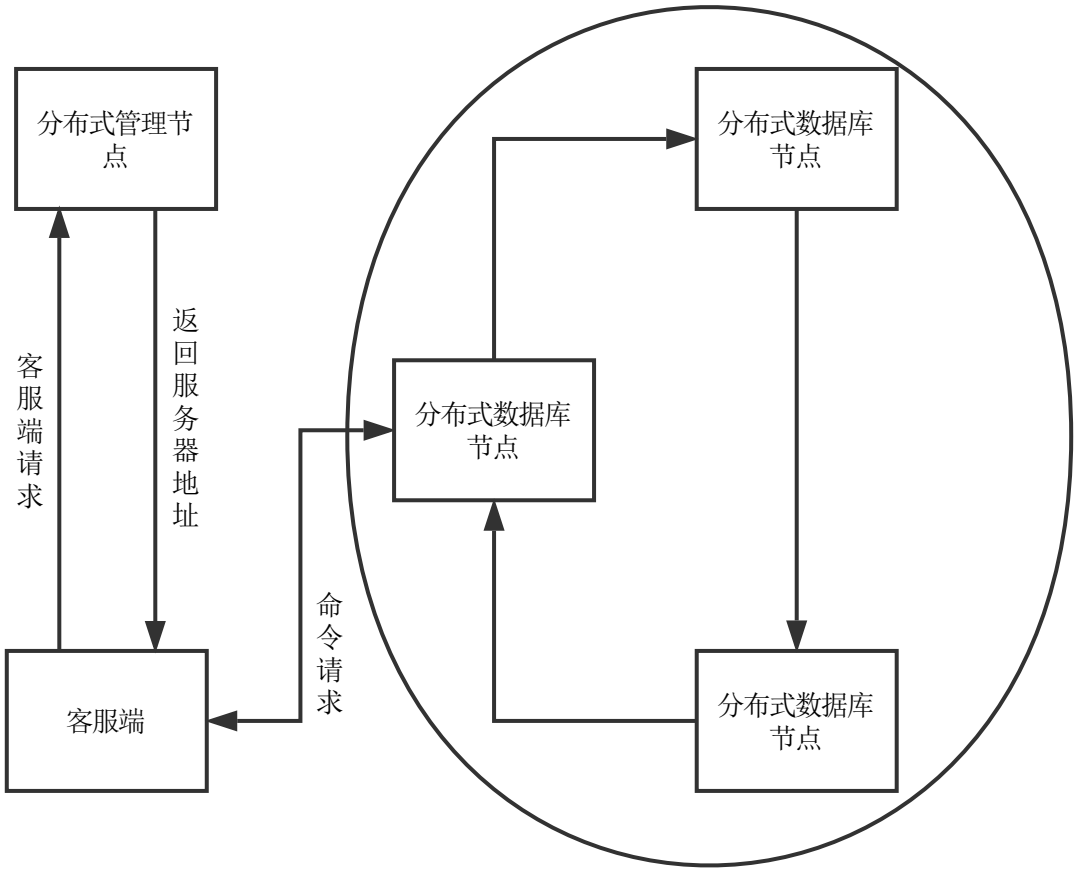

分布式管理节点
客服端请求
返回服务器地址
客服端
命令请求
分布式数据库节点
分布式数据库节点
分布式数据库节点
分布式数据库节点

图 4-2 系统总体架构图

备。分布式管理节点收到客服端请求以后，返回给客服端数据库节点的地址，然后客服端再连接数据库节点，数据库节点负责对请求消息进行处理，包括增加、删除、修改、查询等操作，并将响应消息返回给客户端。分布式管理节点与各个数据库模块在同一台机器中运行，也可以分布到不同的计算机上面。在系统可靠性保护方面，当出现某个数据库模块发生故障时，分布式管理节点会实时的更新管理信息，同时其他数据库节点也会同步这台计算机的信息。数据因为在不同的计算机上面都有备份，所以不存储丢失的危险。

4.1.2 系统功能设计

根据系统的需求分析，采用模块化分解结构划分本系统，使系统能够进行增加、删除、查询、修改等数据库基本操作。系统采用传统的C/S架构，分布式管理节点和数据库节点共同组成了服务器端。分布式数据库系统的功能模块分解图如图4-3所示。

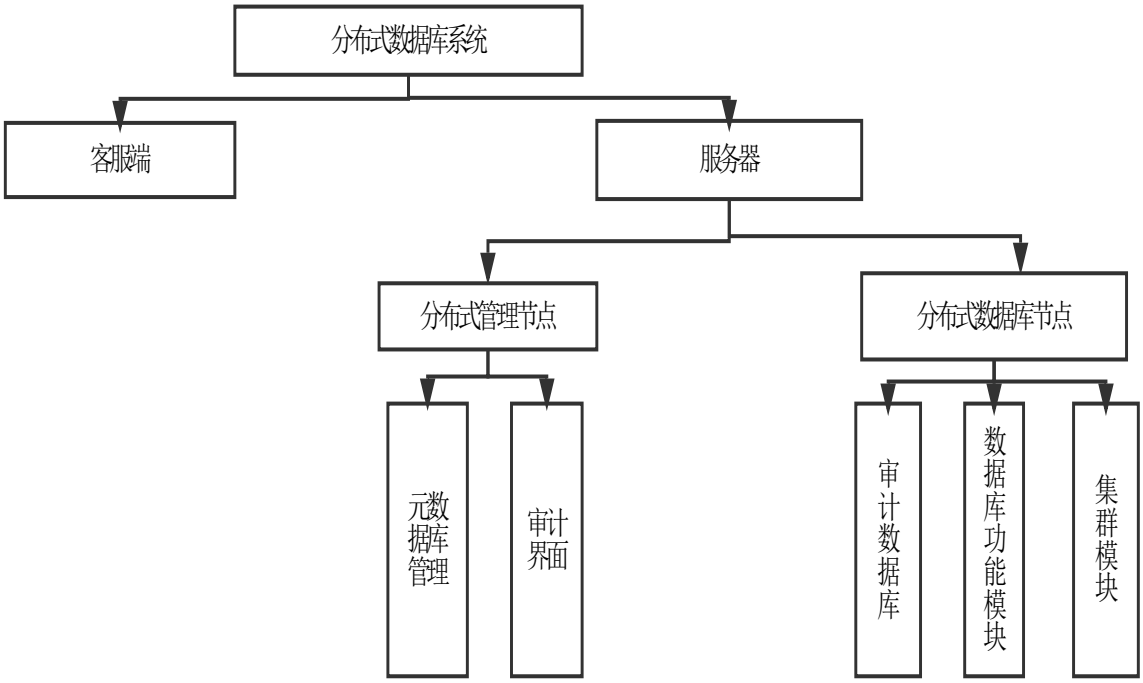


图 4-3 系统总体架构图

以下对图4-3中服务器端各个模块的功能进行简要说明。分布式管理节点按逻辑功能划分为两个子模块，元数据管理模块、审计界面模块。下面分别对两个子模块的功能作介绍。

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。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 客服端模块设计

分布式系统需要解决的一个重要问题便是决定数据在集群中的分布策略，好的分布策略应该能将数据均衡地分布到所有节点上，并且还应该能适应集群节点的变化，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。对于分布式系统，系统中每个节点的负载均衡是非常重要的，分布式算法必须具有良好的分散性，使消息能够均匀的转发到各个数据库节点。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。客服端连接的流程图如图4-4所示。 每个流程解释如下：

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

2.然后客服端连接到正确的数据库节点。

3.最后数据库节点返回给客服端具体的数据。

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的客服端，减少用户的使用成本。使用Mysql客服端，可以执行几乎所有的Mysql命令。在用户看来，分布式数据库节

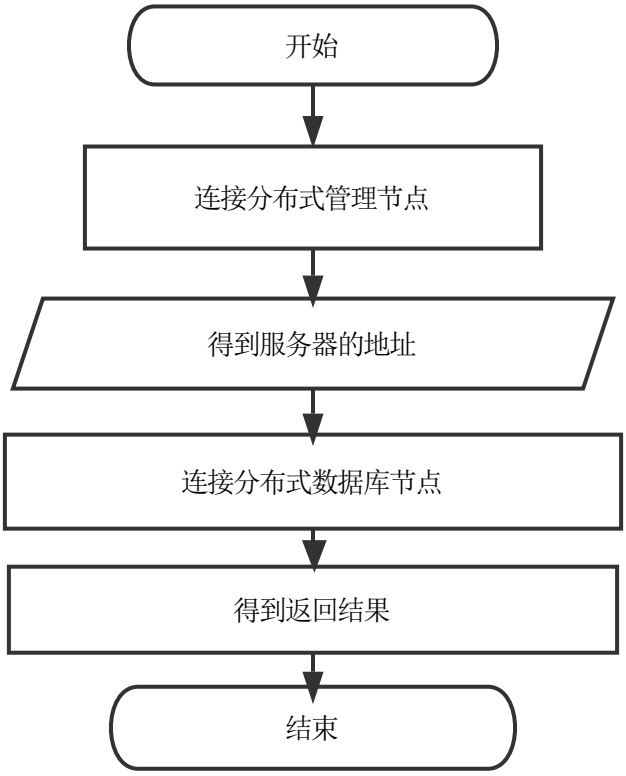


图 4-4 客服端连接流程图

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，只实现客服端是不够的。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

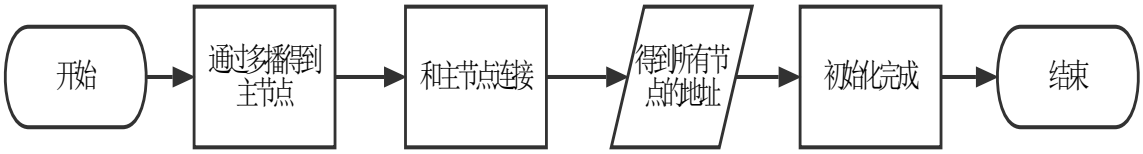


图 4-5 分布式管理节点初始化过程

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。分布式管理节点初始化成功以后，客服端就可以通过自己开发的基于JDBC的客服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

数据库功能模块主要负责根据命令消息类型来完成对应的业务逻辑处理。本模块利用资源管理模块提供的接口，完成数据的增加、删除、修改、查询等业务功能，并将响应消息返回给客户端。下面设计出每个操作业务的流程。

4.4.1 数据库功能操作流程

4.4.1.1 新增数据过程

第一步：客户端发送新增数据请求消息，消息包到达服务器端的分布式管理节点，由分布式管理节点返回客服端的连接会话信息，客服端得到服务器的地址。第二步：客服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：客户端发送查询和修改数据请求消息，消息包到达服务器端的分布式管理节点，由分布式管理节点返回客服端的连接会话信息，客服端得到服务器的地址。第二步：客服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。

4.4.1.3 数据删除过程

第一步：客户端发送删除数据请求消息，消息包到达服务器端的分布式管理节点，由分布式管理节点返回客服端的连接会话信息，客服端得到服务器的地址。第二步：客服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。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。

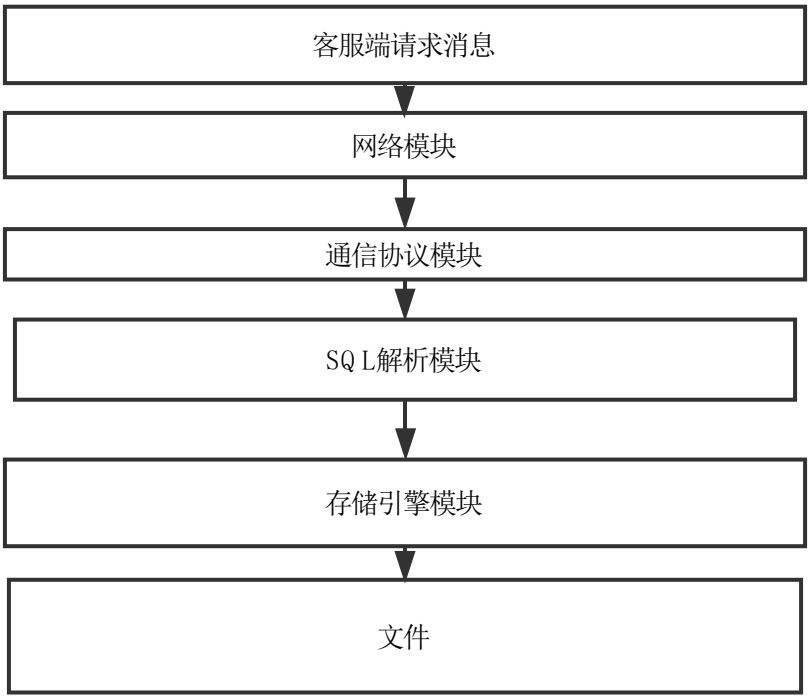


图 4-6 数据库功能模块图

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，这样才能连接客服端的请求。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-

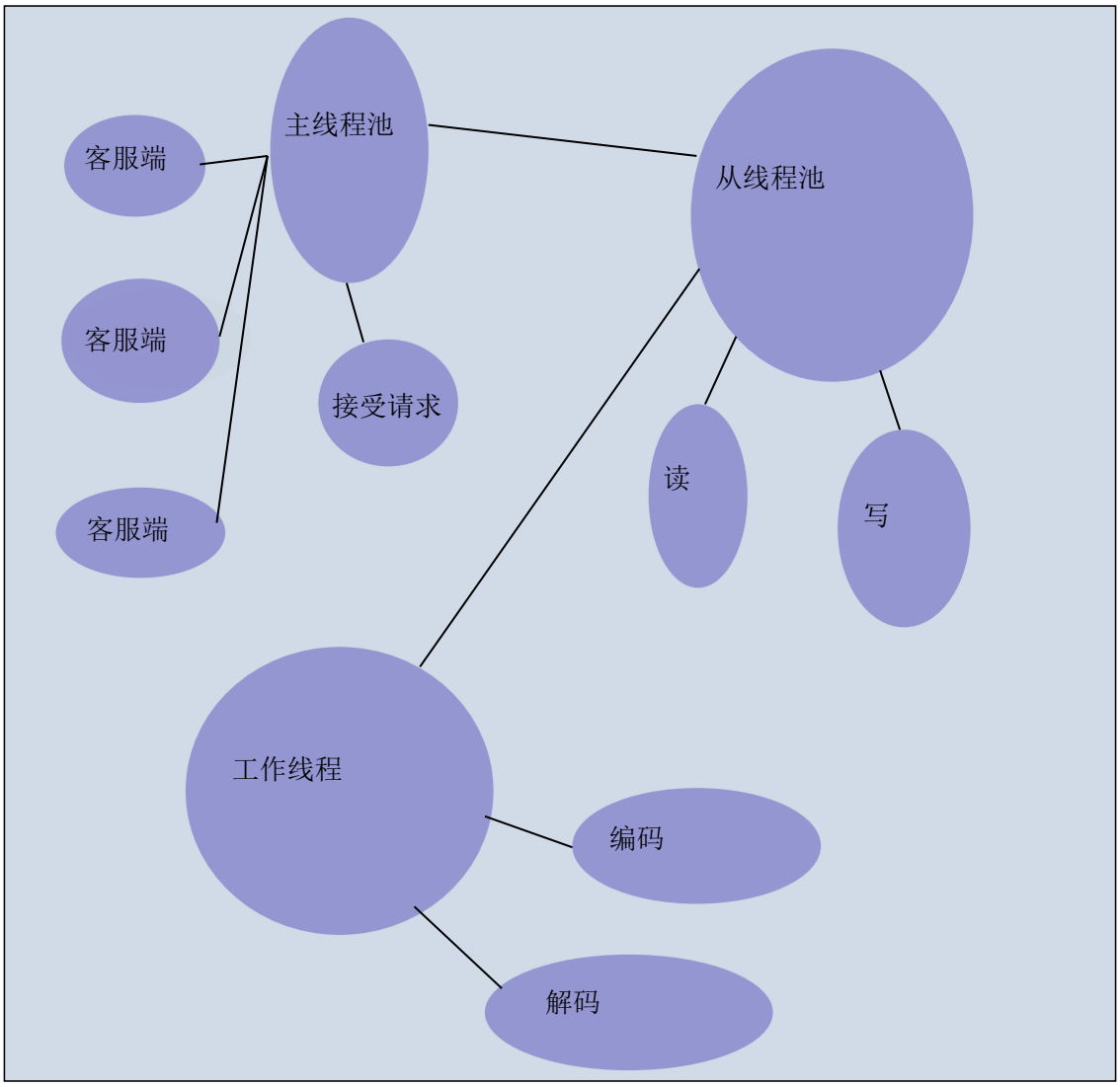


图 4-7 Netty线程模型

7所示，netty主要有三种线程：

1.主线程池主要接受客服端的网络连接请求，当主线程接受到客服端的请求以后，就可以把客服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服端的连接请求。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服端的连接以后，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，这样客服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服端连接。

4.4.3 通信协议设计

每个服务器和客服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。这样就不用自己开发协议了。在MySQL数据库通信过程开始时，服务器会使用TCP监听一个本地socket 端口或本地socket链接。当一个客户端的连接请求到达，就会执行握手和权限验证。如果验证成功，会话开始。客户端发送消息，服务器会以一个适合该发送命令的数据类型的数据集或一条消息进行回复。当客户端发送完成后，会发送一个特殊的命令，告诉服务器己发送，然后会话结束。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主要分为两个阶段：握手认证阶段和命令执行阶段。

1.握手认证阶段

握手认证阶段为客户端与服务器建立连接后进行，交互过程如下：

(a) 服务器发送给客户端握手初始化报文。

(b) 客服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

客户端认证成功后，会进入命令执行阶段，交互过程如下：

(a) 客户端发送服务器执行命令报文。

(b) 服务器发送给客服端命令执行结果。

MySQL客户端与服务器之间的完整交互过程如图4-8所示。

每个报文分为报文头和报文数据两部分，其中报文头占用固定的4个字节，报文数据长度由报文头中的长度字段决定。报文头后面三个字节用于标记当前请求报文的实际数据长度值，以字节为单位，最大值为0xFFFFFF，即接近16 MB大小(比16MB少1个字节)。除了报文长度还有报文序列号，在一次完整的请求和响应交互过程中，用于保证报文顺序的正确，每次客户端发起请求时，序号值都会从0开始计算。每个报文后面的字节就是报文数据，报文数据用于存放请求的内容及响应的数据，长度由报文头中的长度值决定。

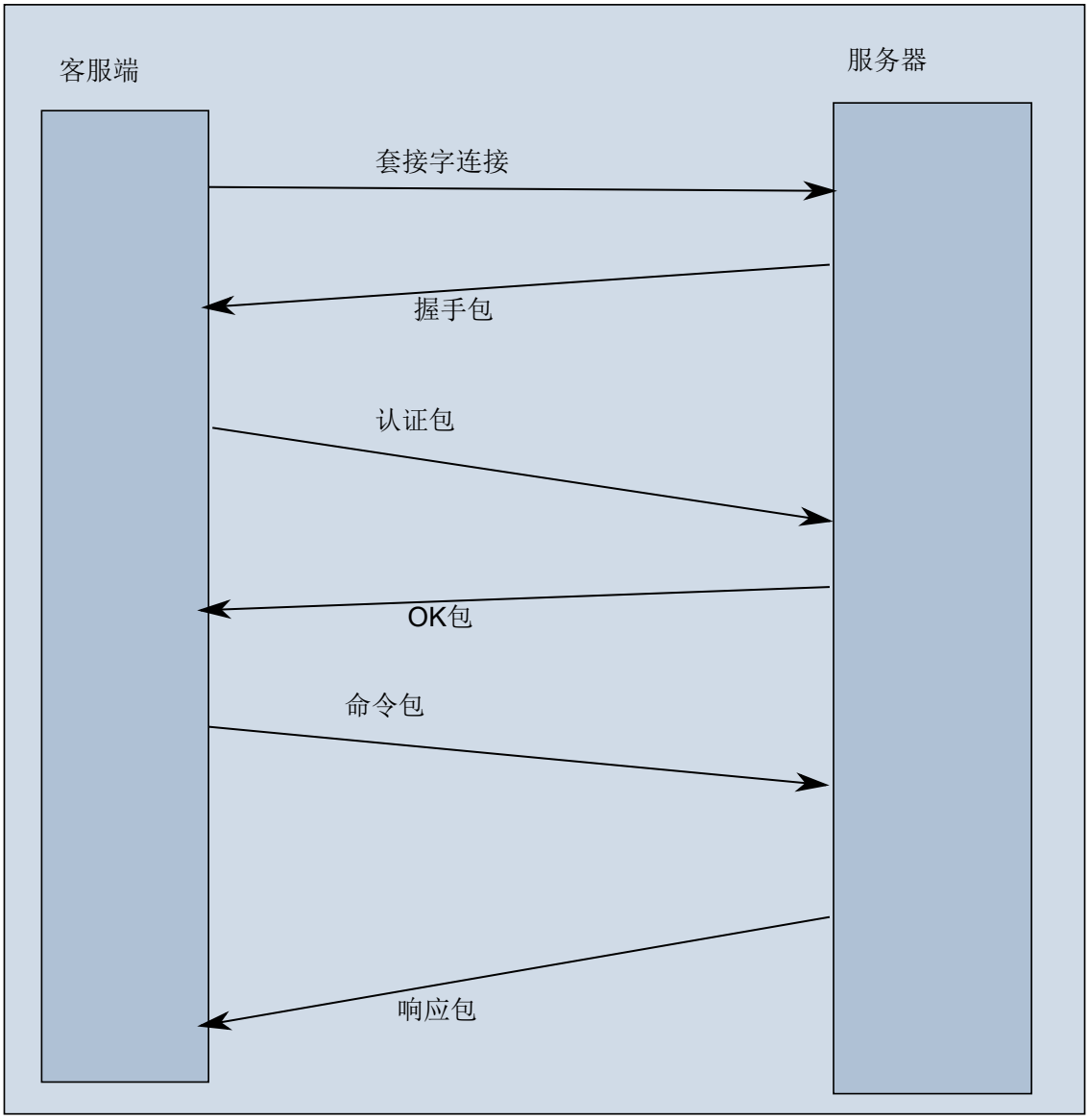


图 4-8 协议交互流程图

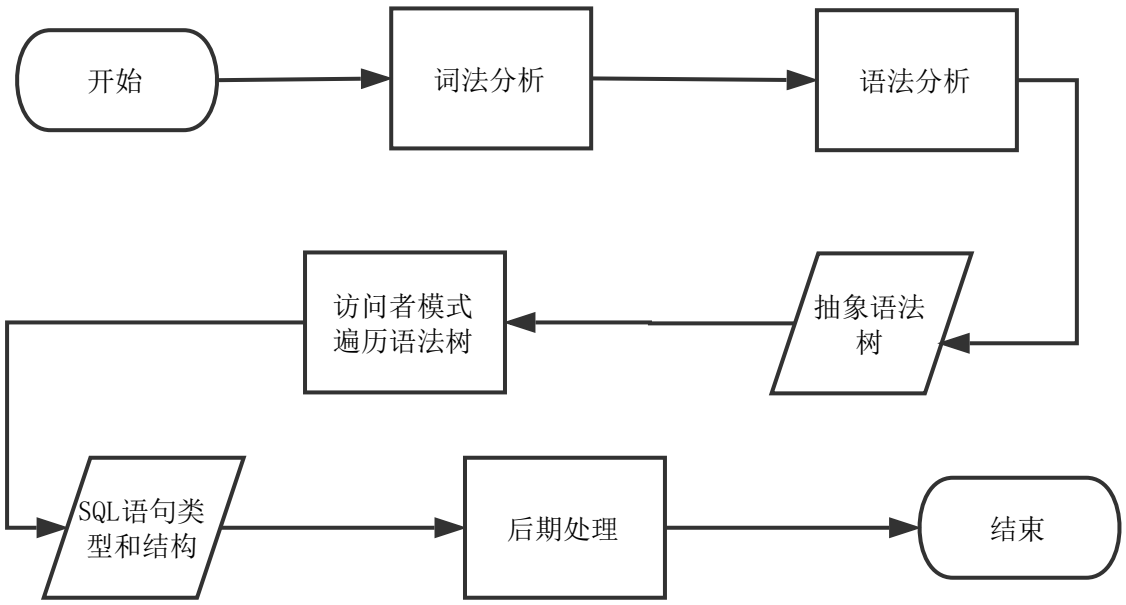


图 4-9 SQL解析流程图

4.4.4 SQL引擎设计

本文基于成熟的开源框架Druid 来实现自己的解析模块，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。

4.4.5 存储引擎设计

存储引擎是存储系统的发动机，直接决定了存储系统能够提供的性能和功能。存储系统的基本功能包括：增、删、读、改，其中，读取操作又分为随机读取和顺序扫描。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，这样可以结合关系数据库和非关系型数据库的优点。在OrientDB这个存储引擎基础之上，封装成关系型的操作。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

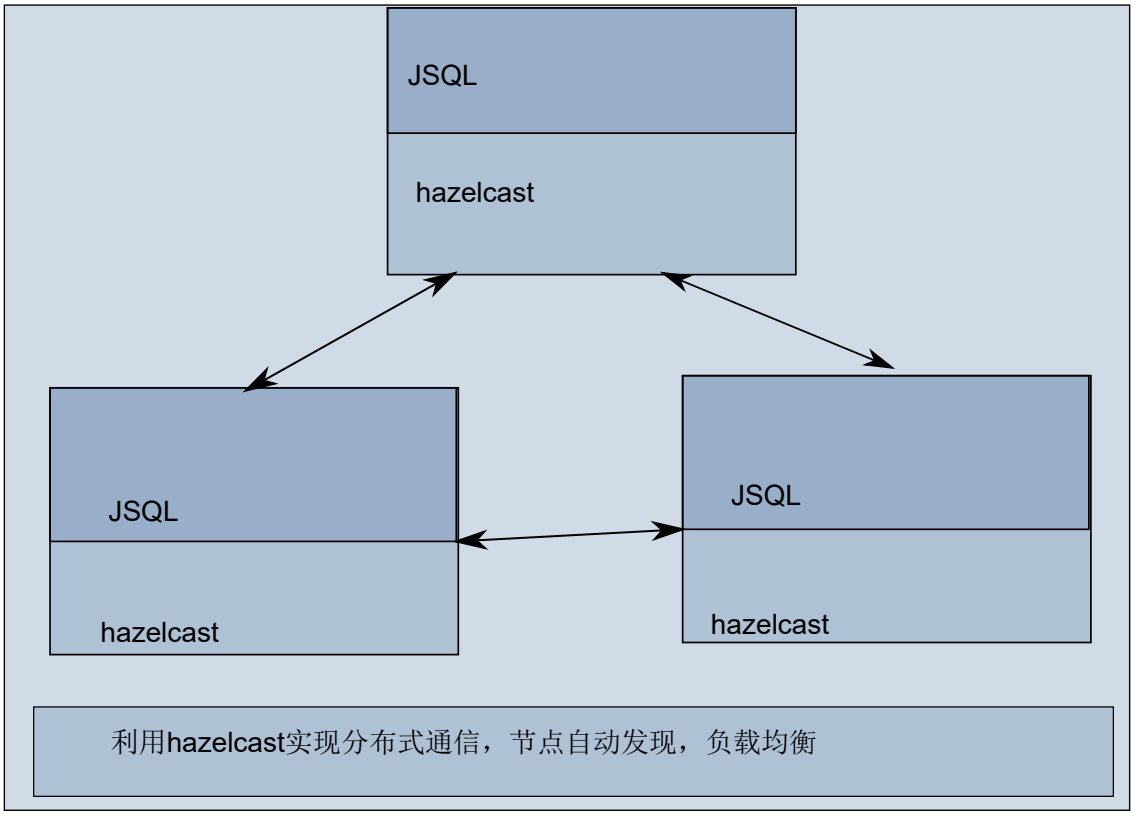


图 4-10 集群架构图

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。 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。

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

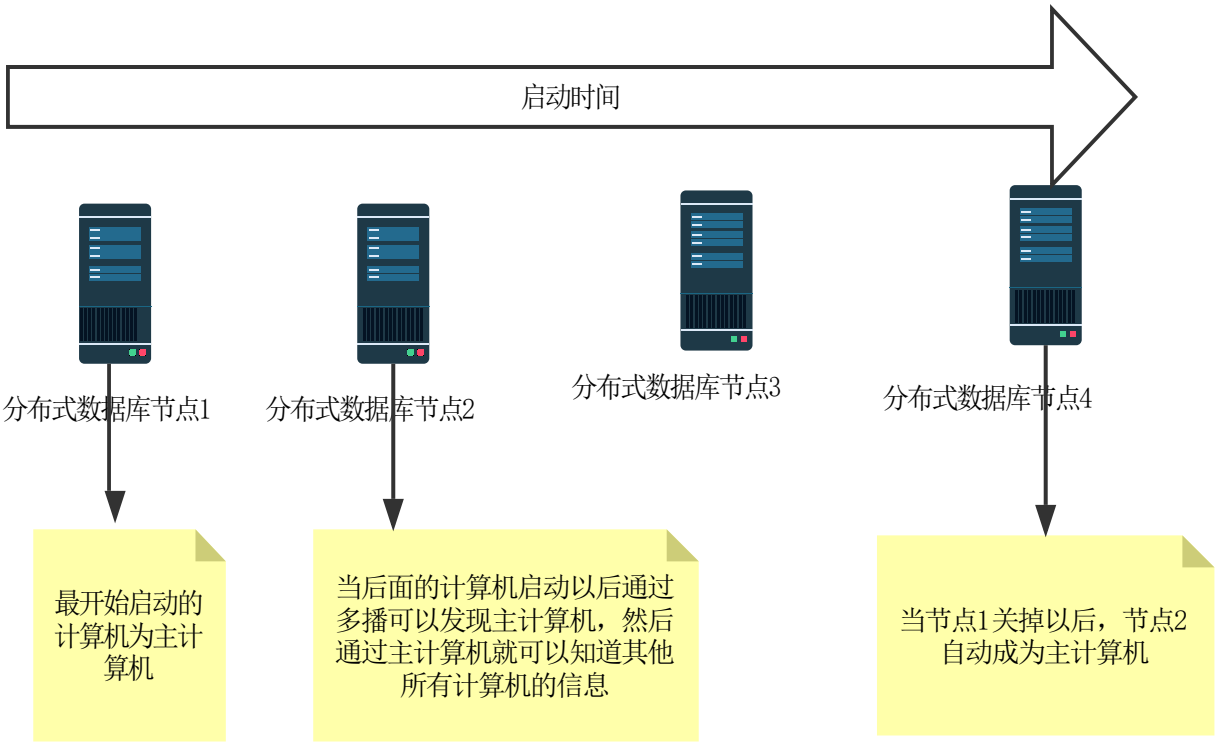


图 4-11 集群架构图

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结构图。从上大下一共可以分为下面4个模块：

1.grafana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。

4.7 本章小结

任何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。本章从总体上设计了系统的部署模式和模块图，对每个模块进行了详细了的模块设计，便于后面的具体实现。

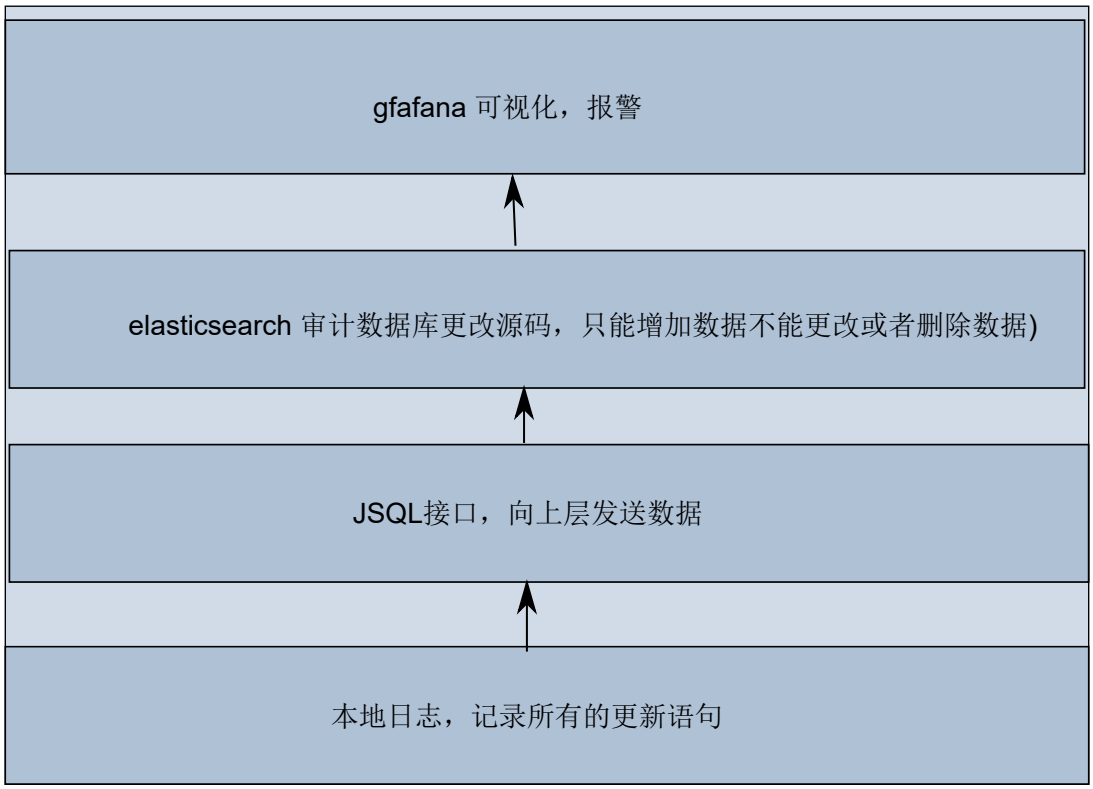

gfafana 可视化，报警
elasticsearch 审计数据库更改源码，只能增加数据不能更改或者删除数据)
JSQL接口，向上层发送数据
本地日志，记录所有的更新语句

图 4-12 监控模块

第五章 系统实现

在对系统进行详细设计以后，就是编码实现阶段。本章按照前一章的设计，给出每个功能模块的具体实现。详细分析关键功能实现流程图，也包括对关键代码的分析。

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。 在本章的后面会给出其他其他功能模块的详细实现。

5.2 客服端功能实现

客服端主要实现负载均衡功能，利于JDBC连接后端的分布式数据库，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

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

客服端返回正确数据库地址以后，客服端就可以通过JDBC连接数据库节点。客服端连接功能流程图如图5-1所示。

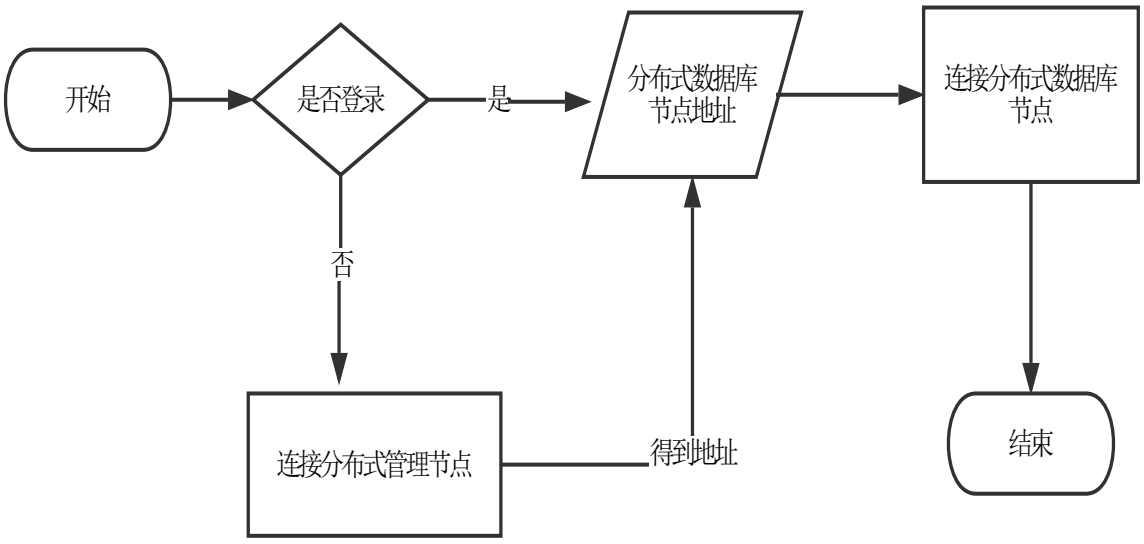


图 5-1 客服端连接功能流程图

首先，判断客服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。当客服端没有连接会话的时候，客服端就首先连接分布式管理节点，分布式管理为客服端返回合适的数据节点地址，然后客服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服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 分布式管理节点实现

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。

5.4 负载均衡功能实现

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。

负载均衡算法包括静态负载平衡和动态负载平衡。只是利用系统负载的静态信息, 而忽视系统当前节点的的负载状况和性能的方法被称为静态负载均衡; 根据系统当前的负载状况和性能来调整任务划分的方法被称为动态负载均衡。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，分布式管理节点就是这样的管理节点。

在集群启动的时候，初始化管理节点，其流程如图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。

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

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。

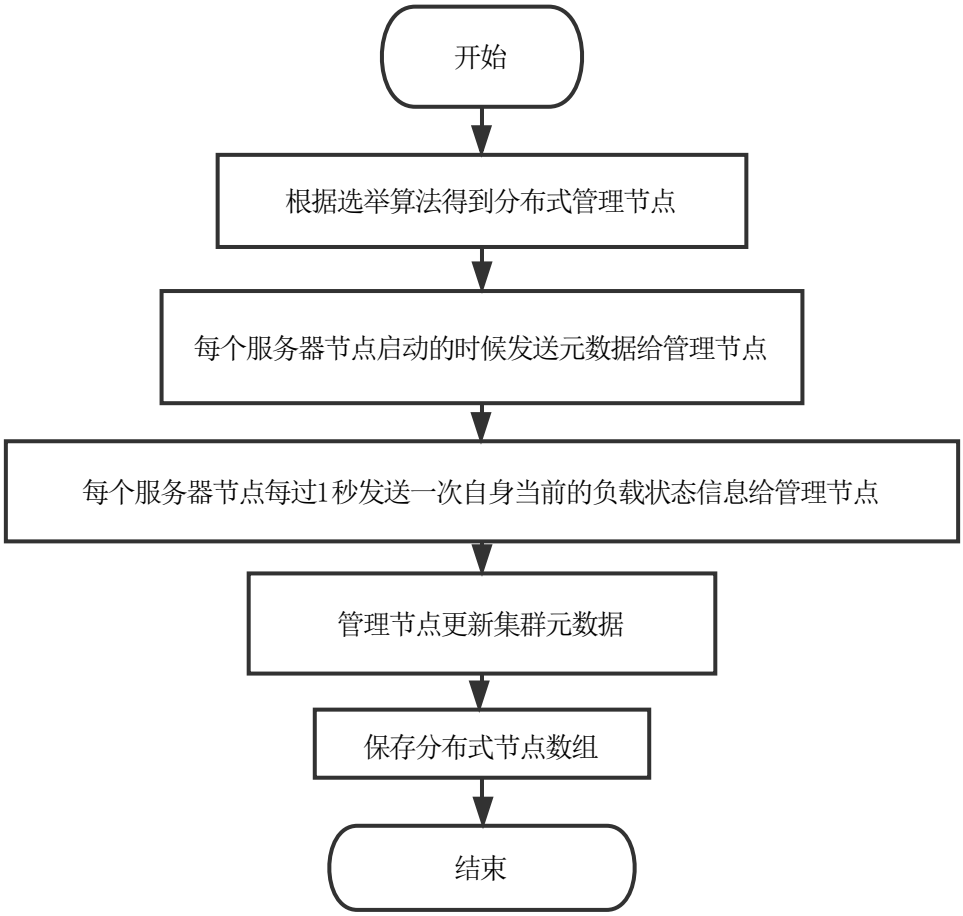


图 5-2 分布式管理节点初始化流程

在集群初始化完成以后，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

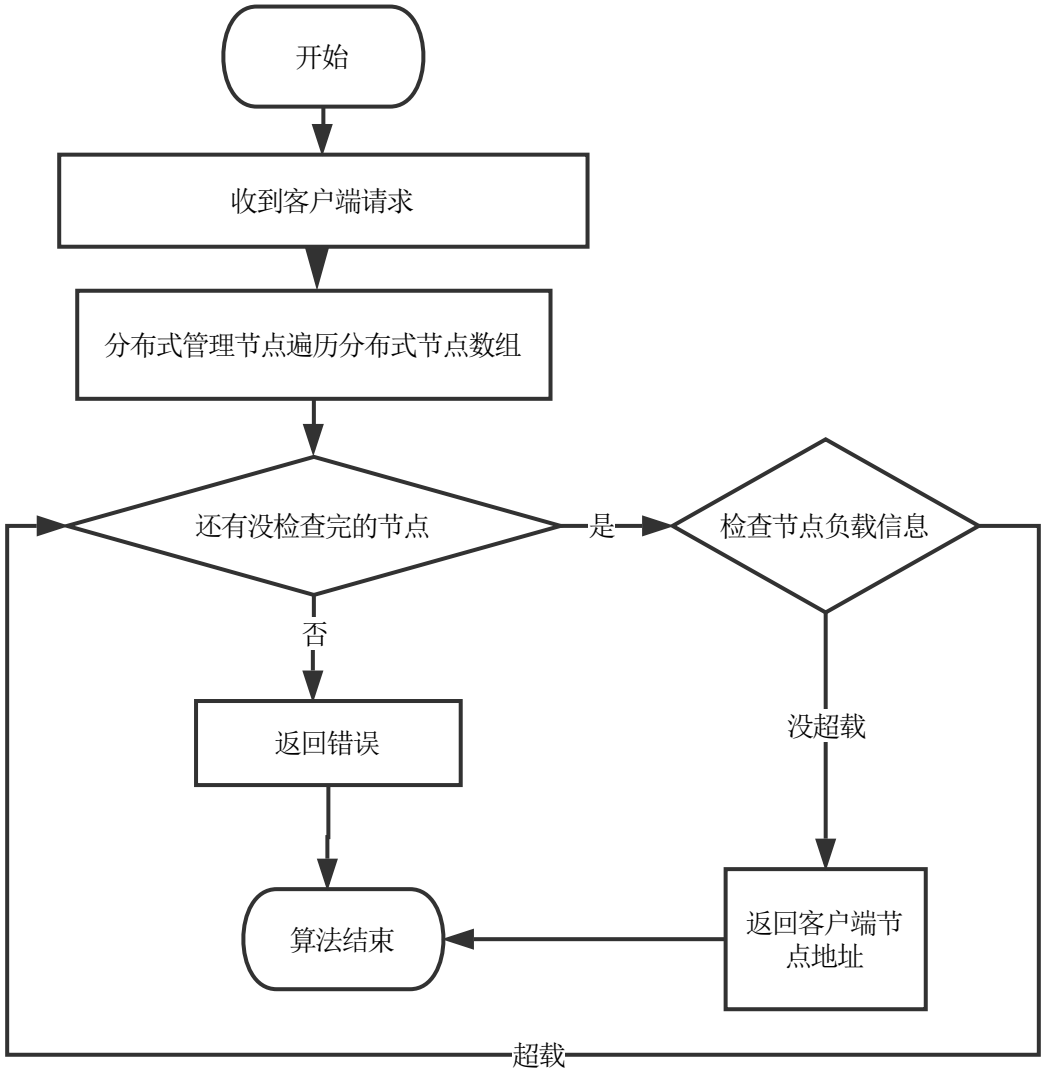


图 5-3 负载均衡流程图

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。

5.5 多版本MVCC算法实现

2.7.3.1 Megastore 中的 MVCC Megastore 利用了 Big Table 中数据的多版本特性实现分布式的更新事务。每个事务更新的都是不同版本（timestamp）的 Big Table

数据，在读取数据时利用 timestamp 过滤，从而不会读到正在进行的尚未生效的事务数据。其原理与本节中介绍完全一致，不再赘述。 2.7.3.2 Doris*中的 MVCC 在 Doris*系统中，数据按批量进行更新，每个批量的数据都可以认为是一个事务，必须同时原子性的生效。为此，Doris*将每条数据附带了一个导入的版本号，在读取数据时根据元数据中已生效的版本号与数据上的导入版本号做过滤，从而不读取正在更新的尚未生效的数据，实现了分布式事务更新。其详细流量与本节中介绍的一致。

5.6 数据库系统实现

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，这个类主要调用其他模

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

块，完成数据库系统的启动。 ShutdownMain类用来关闭数据库系统。按照客服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。

5.6.1 网络模块

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服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。 在接收到客服端套接字连接以后，客服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5-4所示。 网络模块收到客服端的套接字连接以后需要包网络

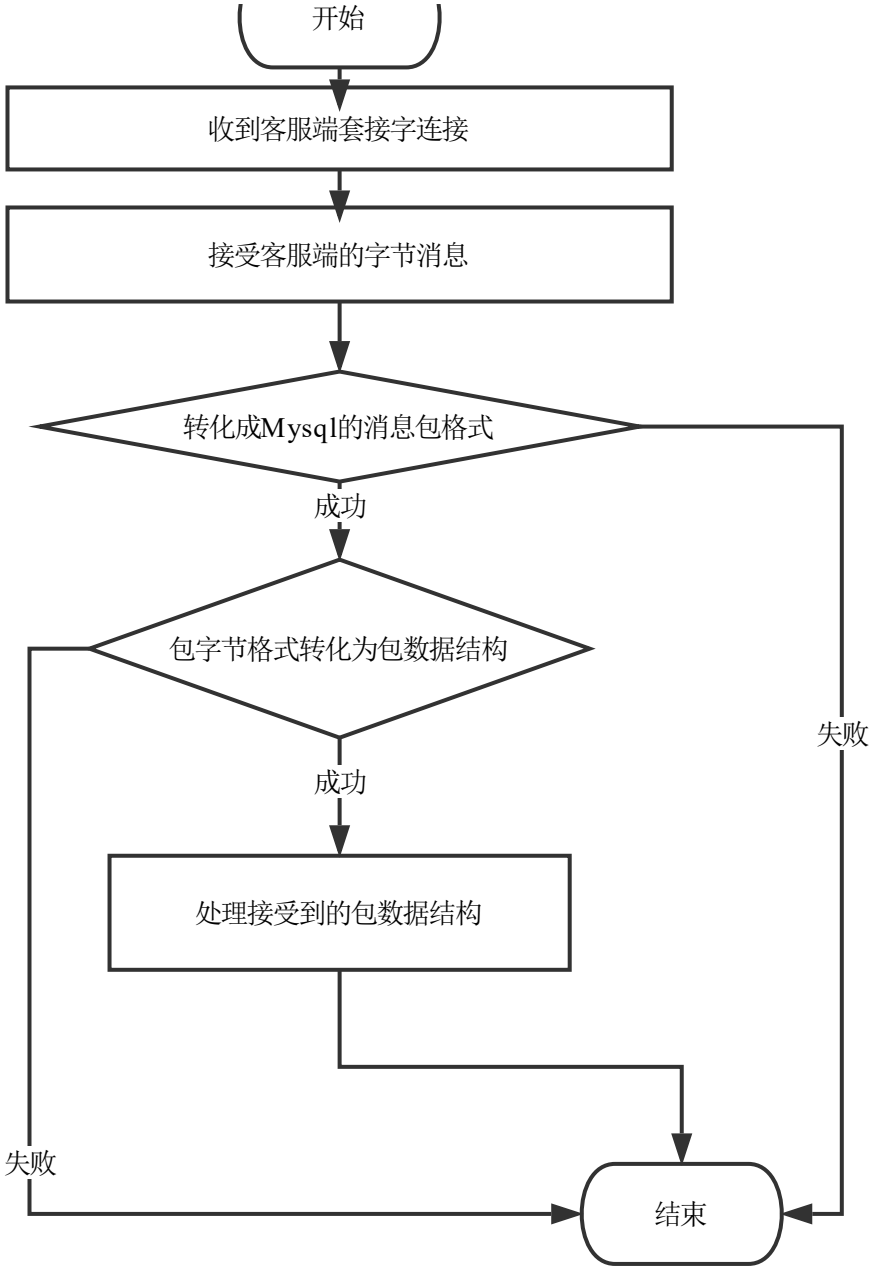


图 5-4 网络模块功能图

字节包装成Mysql的消息包格式，如果错误就说明客服端不是我们的客服端或者客服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。

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

网络服务器接受到客服端的连接以后，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，表5-6描述了所有的mysql通信协议的封装对象。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5-5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。

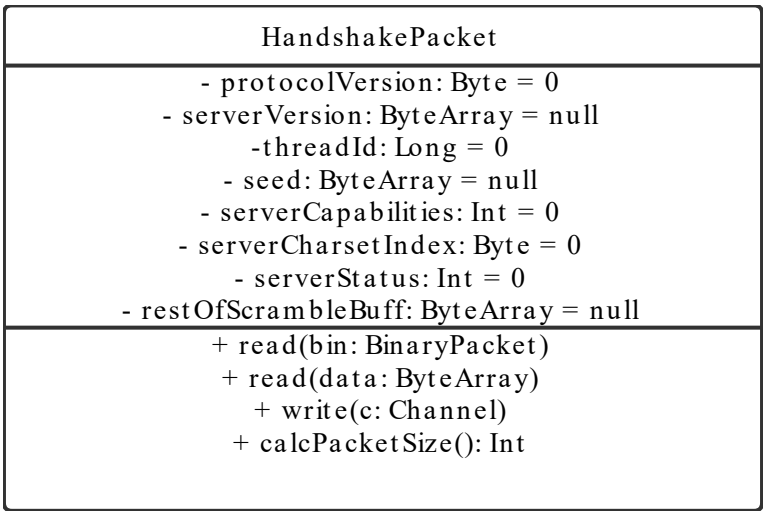


图 5-5 握手包实现类图

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服端的请求，然后检查用户的认证信息，比如用户名和密码，图5-6显示了认证流程过程。 如果认证成

表 5-6 mysql所有协议包的封装对象

文件	解释
AuthPacket	From client to server during initial handshake.
BinaryPacket	class BinaryPacket : MySQLPacket
CommandPacket	From client to server , the client wants the server to do something
EOFPacket	the EOF packet contains a warning
EmptyPacket	class EmptyPacket : MySQLPacket
ErrorPacket	From server to client in response to command, if error.
ExecutePacket	class ExecutePacket : MySQLPacket
FieldPacket	part of Result Set Packets. One for each column in the result set.
HandshakePacket	From server to client during initial handshake
HandshakeV10Packet	Connection Phase The Connection Phase performs these tasks
HeartbeatPacket	From client to server when the client do heartbeat between jsql cluster
LongDataPacket	class LongDataPacket : MySQLPacket
MySQLPacket	abstract class MySQLPacket
OkPacket	From server to client ,if no error and no result set
PingPacket	class PingPacket : MySQLPacket
PreparedOkPacket	class PreparedOkPacket : MySQLPacket
QuitPacket	class QuitPacket : MySQLPacket
Reply323Packet	class Reply323Packet : MySQLPacket
RequestFilePacket	load data local infile
ResetPacket	class ResetPacket : MySQLPacket
ResultSetHeaderPacke	The Result Set Header Packet is the first of several packets
RowDataPacket	From server to client. One packet for each row in the result set

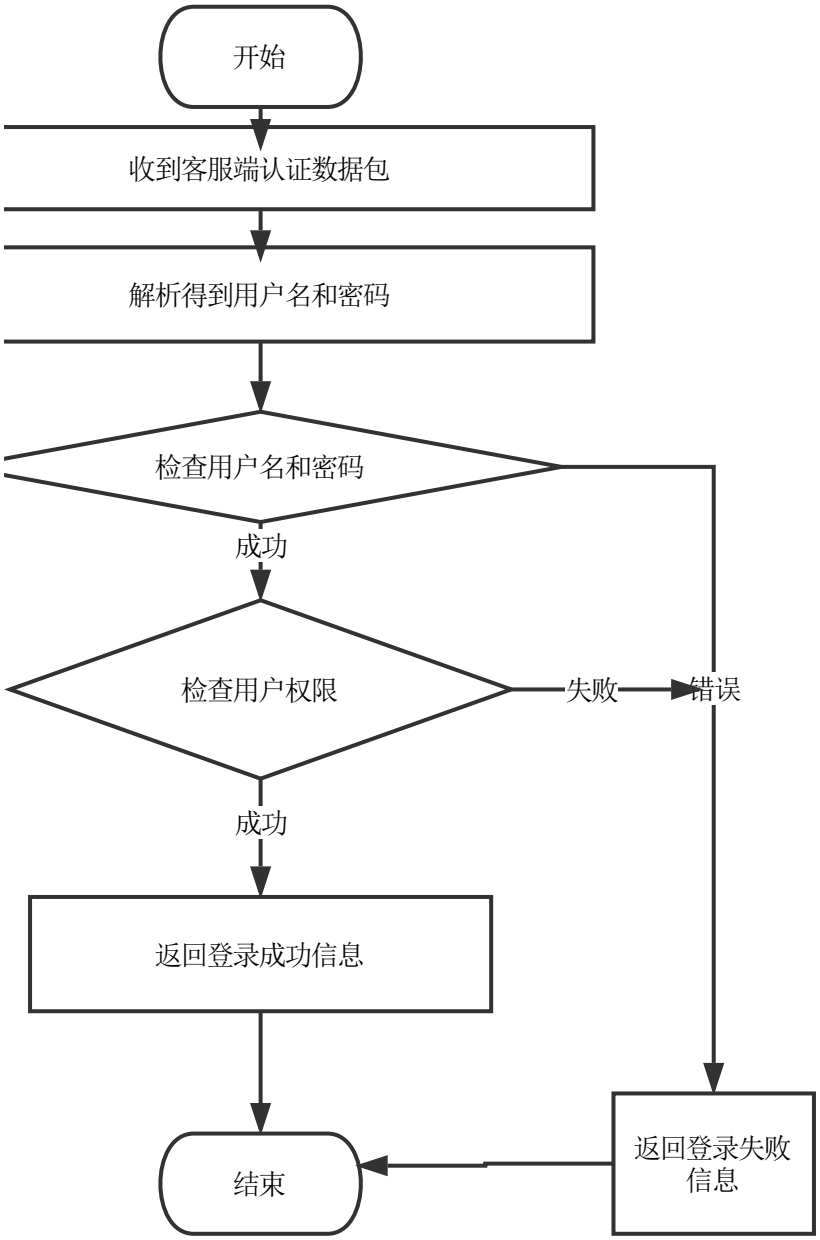

开始
收到客服端认证数据包
解析得到用户名和密码
检查用户名和密码
成功
检查用户权限
失败
错误
成功
返回登录成功信息
结束
返回登录失败信息

图 5-6 认证流程图

功以后，就建立连接会话，然后客服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服端失败原因。认证成功以后就是命令阶段，服务器接受客服端的命令，然后处理，图5-7显示了命令处理流程。 在接收到客服端的命令请求以后，网络模块就会调用下面模块的功能接口来

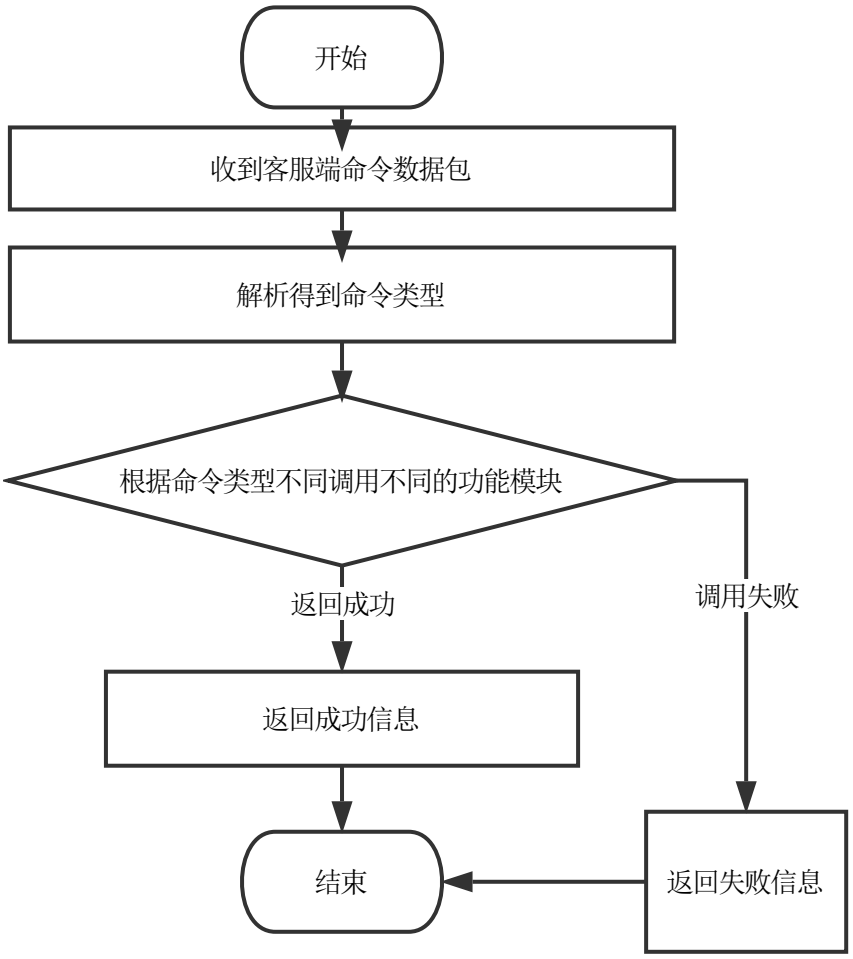


图 5-7 命令处理流程图

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果发送给客服端，如果命令处理工程中发送了异常，就返回失败原因。

5.6.2 SQL解析模块

经过网络模块的处理，我们建立了客服端的连接会话，接下来我们就要对客服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服端连接和会话的封装和管理，表5-7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 5-7 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

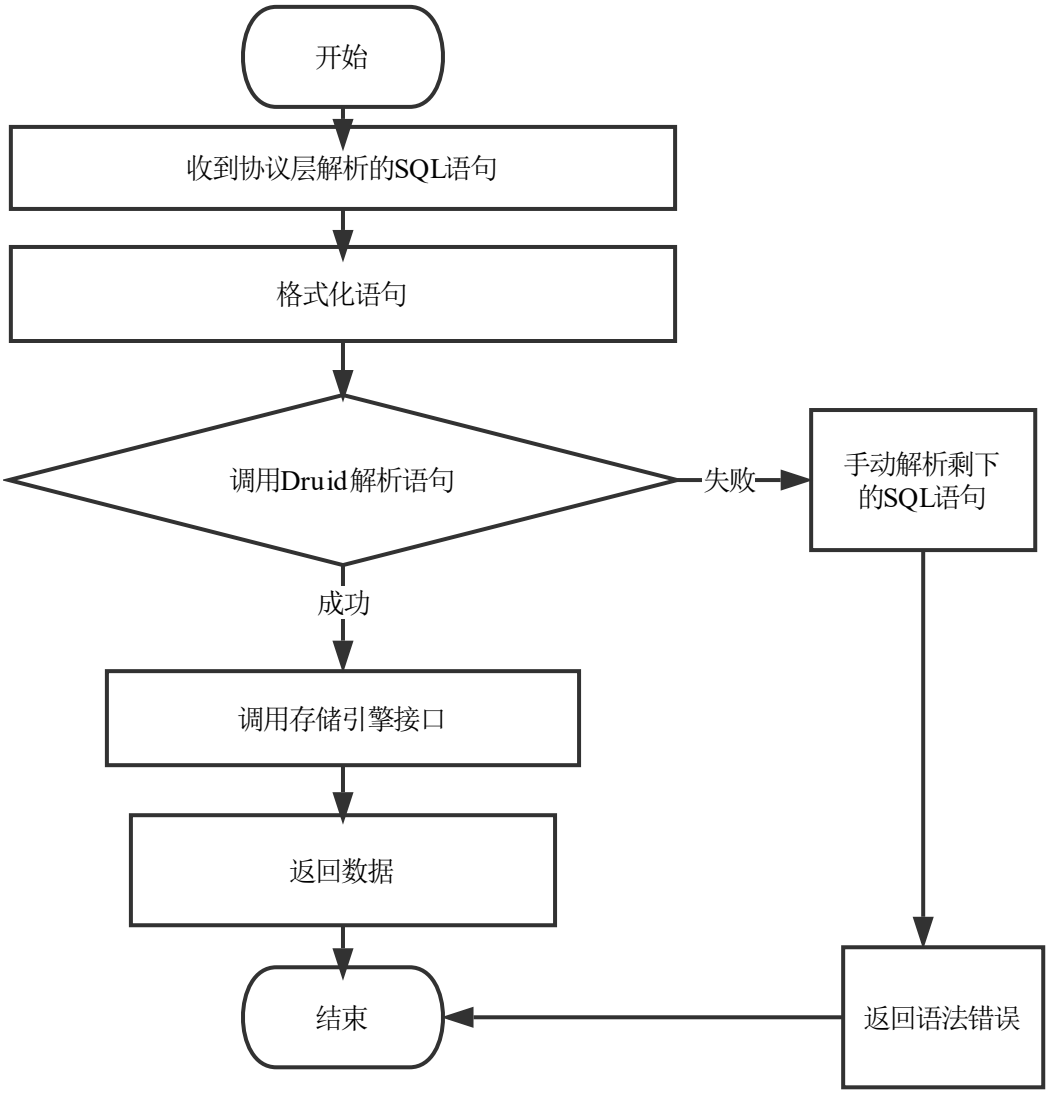


图 5-8 SQL处理流程图

要做具体的数据处理，图5-8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服端。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， 5-8描述了sql模块下实现的各种类型的语句。

表 5-8 SQL模块下实现的各种类型的语句

类	作用
io.jsql.sql.handler.adminstatement	处理管理语句
io.jsql.sql.handler.componed_statement	处理组合语句
io.jsql.sql.handler.data_define	处理数据定义语句
io.jsql.sql.handler.data_mannipulation	处理数据操作数据
io.jsql.sql.handler.preparestatement	处理准备语句
io.jsql.sql.handler.replication_statement	处理复制语句
io.jsql.sql.handler.tx_and_lock	处理事务和锁控制语句
io.jsql.sql.handler.utilstatement	处理其他工具语句

每一种类型语句下面又回有很多种具体语句的实现，表5-9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 5-9 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

5.6.3 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5-10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5-11所示。 和表有关的功能接口如表5-12所示。

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5-9b所示，有关表操作的类接口如图5-10所示。类的实现主

表 5-10 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 5-11 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 5-12 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>

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。

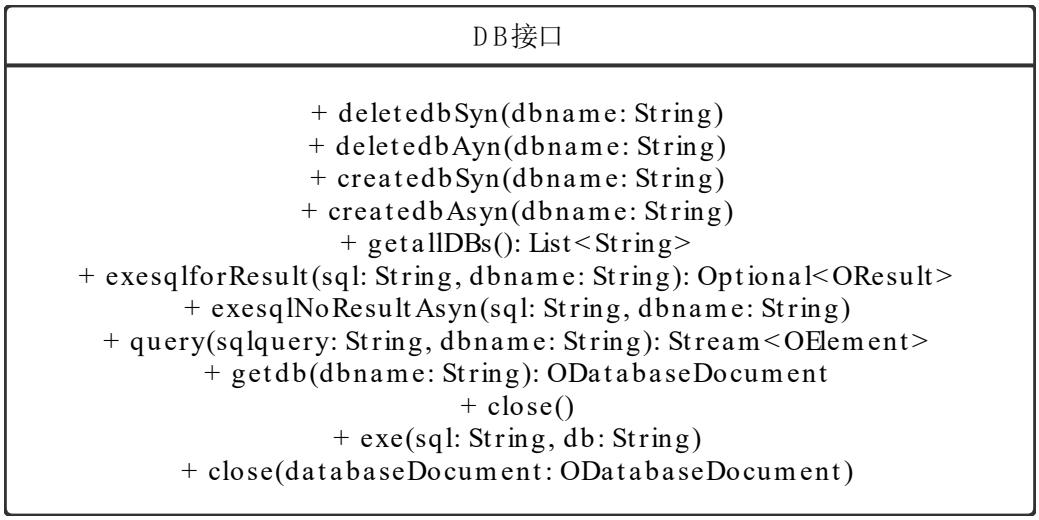


图 5-9 数据库操作接口类图

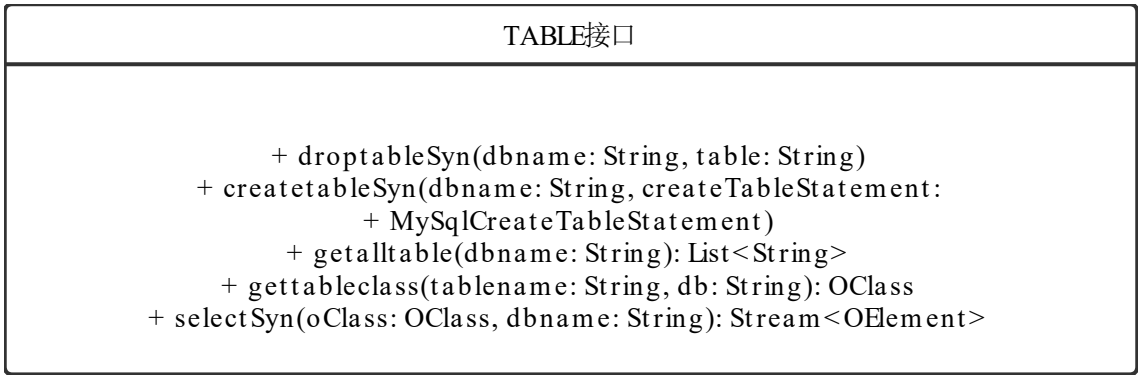


图 5-10 表操作接口类图

5.7 集群架构的实现

集群功能主要是利用了开源的hazelcast框架来实现，其中的功能全在hazelcast包下面实现，表5-13给出了该包下面每个类的具体的作用。 其中最关键的代码如下。

```
01 //sql队列 复制队列命令队列。2个锁。发布
02 @Component
03 class MyHazelcast : ItemListener<SqlUpdateLog> {
```

表 5-13 集群模块下面各个类的作用

类	作用
LogFile	本地文件系统中存储日志文件
MyHazelcast	利用Hazelcast的分布式数据结构实现集群，比如分布式队列和分布式锁
ReplicationCMD	分布式对象，在不同集群之间传输
SqlUpdateLog	当前系统LSN最大值，新的事务日志LSN将在此基础上生成

```
04      private fun exeSqlforReplication() {
05          Collections.sort(localqueneReplication)
06          var log: SqlUpdateLog? = localqueneReplication.poll()
07          var lastlsn: Long = 0
08          while (log != null) {
09              locals_maxlsn = log.LSN
10              logger.info("exeSqlforReplication exe sql " + log)
11              oNullConnection!!.schema = log.db
12              sqLhander.handle(log, oNullConnection)
13              logFile!!.write(log)
14              lastlsn = log.LSN
15              log = localqueneReplication.poll()
16          }
17          isreplicating = false
18          log = localquene.poll()
19          val remotel = iAtomic_remote_lsn!!.get()
20          while (log != null) {
21              locals_maxlsn = log.LSN
22              logger.info("exe  Sql : " + log)
23              oNullConnection!!.schema = log.db
24              sqLhander.handle(log, oNullConnection)
25              logFile!!.write(log)
26              lastlsn = log.LSN
27              if (lastlsn > remotel) {
28                  remotequene!!.offer(log)
29              }
```

```
30              log = localqueneReplication.poll()
31          }
32          if (lastlsn > remotel) {
33              iAtomic_remote_lsn!!.set(lastlsn)
34          }
35      }
36      /**
37       *  本机发出的sql语句.记录到本地logfile。
38      同时发布到其他服务器*/
39      fun exeSql(sql: String, db: String) {
40          if (isreplicating) {
41              val l = iAtomic_remote_lsn!!.get()
42              val log = SqlUpdateLog(l + 1, sql, db)
43              localquene.addLast(log)
44          } else {
45              locals_maxlsn++
46              val l = iAtomic_remote_lsn!!.addAndGet(1)
47              val log = SqlUpdateLog(l, sql, db)
48              logger.info("exe sql " + log)
49              logFile!!.write(log)
50              remotequene!!.offer(log)
51          }
52      }
53      private fun exesqlforremoteData() {
54          Collections.sort(localquene)
55          var log: SqlUpdateLog? = localquene.poll()
56          var last: Long = 0
57          while (log != null) {
58              locals_maxlsn = log.LSN
59              logger.info("exesqlforremoteData()  " + log)
60              oNullConnection!!.schema = log.db
61              sqLhander.handle(log, oNullConnection)
62              logFile!!.write(log)
```

```
63              last = log.LSN
64              log = localquene.poll()
65          }
66          val l = iAtomic_remote_lsn!!.get()
67          if (last > l) {
68              iAtomic_remote_lsn!!.set(last)
69          }
70
71      }
72
73  }
```

5.8 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5-14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，一种是客服端的登录记录日志，一种是客服端的命令执行日志。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5-11显示了审计功能演示图。 审计功能主要包括审计数据库，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

表 5-14 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

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5-14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。

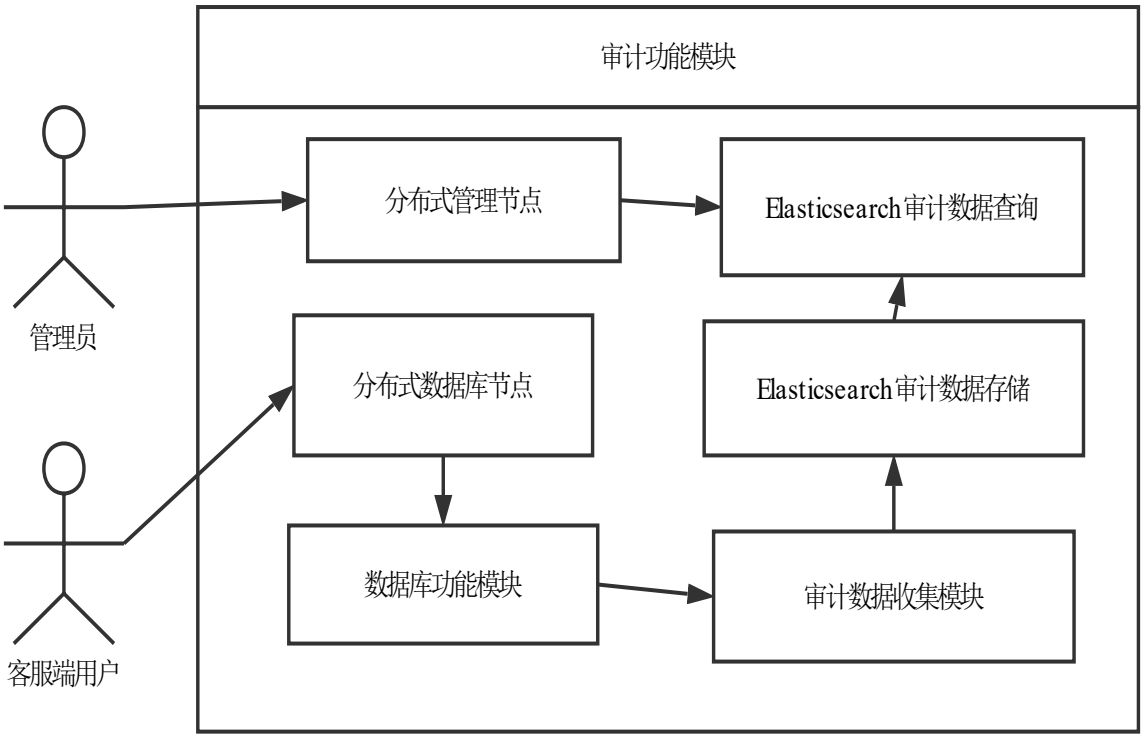


图 5-11 审计功能演示图

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。

5.9 本章小结

本文前面一章设计了本系统的架构图和各个模块的详细功能，本章给出每个功能模块的具体实现。。

第六章 系统测试

6.1 测试环境

本文的测试环境为个人的计算机。它的硬件和软件参与如下

1.硬件参数：

(a) CPU：intel(R) Xeon(R) E5620

(b) 内存：8G

(c) 磁盘：500G

(d) 网络：100Mbit/s

2.软件参数：

(a) 操作系统：windows10

(b) java版本：8.0

6.2 功能测试

6.2.1 数据库功能测试

功能测试设计的细节非常多，项目繁杂。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，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客服端连接系统

(c) 测试结果：系统能正常启动，而且mysql客服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，客服端连接上系统

(b) 测试步骤：

i. 打开命令行客服端

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，客服端连接上系统

(b) 测试步骤：

i. 打开命令行客服端

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，客服端连接上系统

(b) 测试步骤：

i. 打开命令行客服端

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，客服端连接上系统

(b) 测试步骤：

i. 打开命令行客服端

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，客服端连接上系统

(b) 测试步骤：

i. 打开命令行客服端

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，客服端连接上系统

(b) 测试步骤：

i. 打开命令行客服端
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

6.2.2 集群功能测试

集群的功能测试和单机版一样，考虑到不可能每条语句的测试。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) 打开命令行客服端，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

6.2.3 审计功能测试

审计功能的测试主要是测试可视化的显示功能。

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) 打开命令行客服端，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

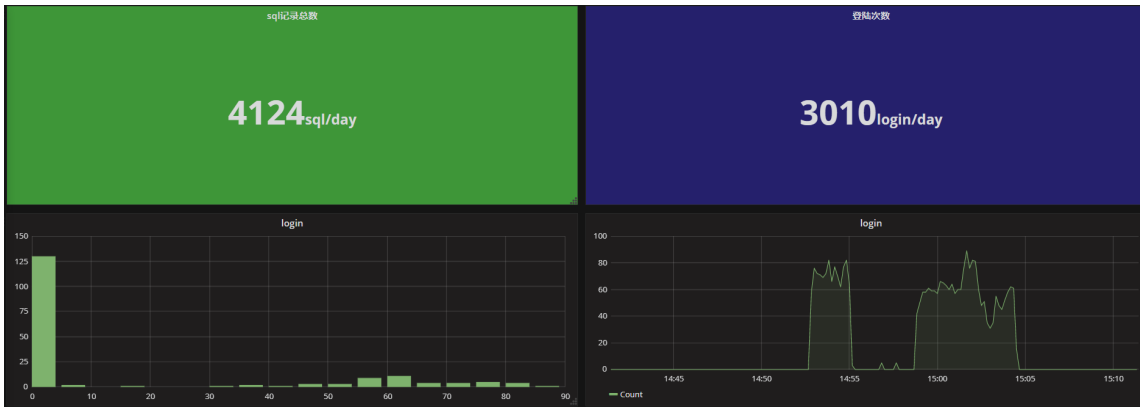


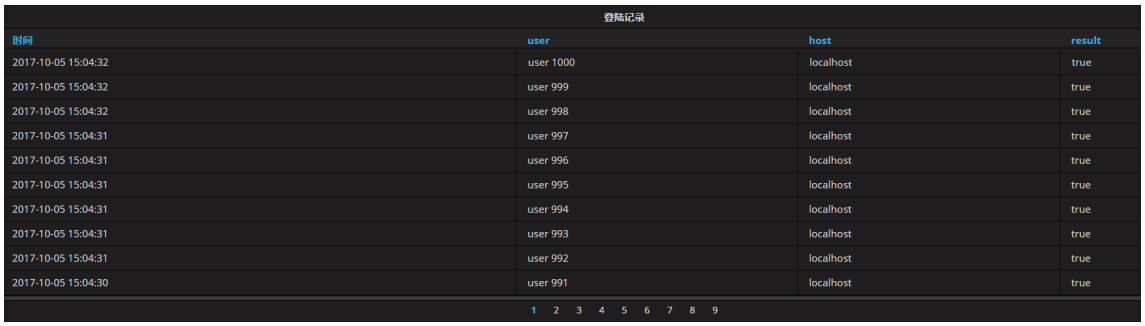
图 6-1 监控可视化结果

图 6-2 监控可视化结果

6.3 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
03        val sqlnumber = intArrayOf(10, 100, 1000, 3000, 5000, 10000)
```



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

```
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的执行时一般都响应的增加，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Apache组织开发的基于Java的压力测试工具。用于对软件做压力测试，它最初被设计用于Web应用测试，但后来扩展到其他测试领域。它可以用于测试静态和动态资源，例如静态文件、Java 小服务程序、CGI 脚本、Java 对象、数据库、FTP 服务器，等等。JMeter 可以用于对服务器、网络或对象模拟巨大的负载，来自不同压力类别下测试它们的强度和分析整体性能另外，JMeter能够对应用程序做功能/回归测试，通过创建带有断言的脚本来验证你的程序返回了你期望的结果。为了最大限度的灵活性，JMeter允许使用正则表达式创建断言。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，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。

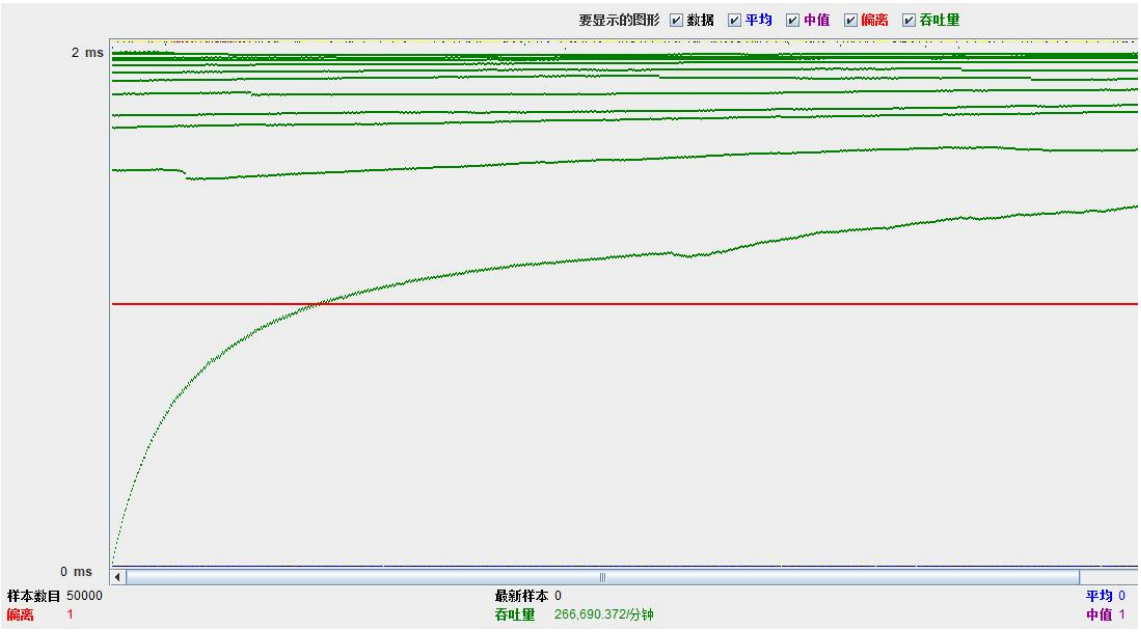


图 6-5 jsql的JMeter性能测试

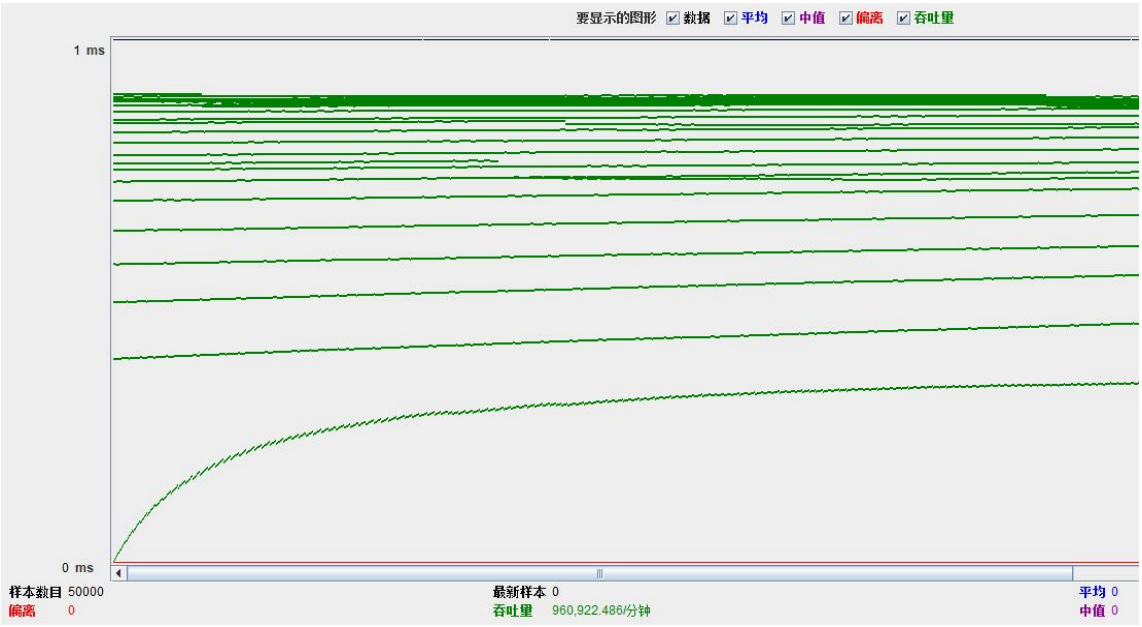


图 6-6 mysql的JMeter性能测试

6.4 本章小结

本章对JSQL数据库系统进行测试，主要分为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。性能测试分为JDBC和JMeter测试两部分。

第七章 总结与展望

现在社会的发展，离不开各种各样的数据库系统。关系型数据库在很多关键场合有不可替代的优势，对关系型数据库有关的理论研究和实践非常有意义。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，包括硬件硬件有关的知识，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先：

1.从数据库引擎到SQL模块完全采用JAVA语言编写，具有跨平台和安全的特点。

2.利用哈希树实现了高性能的存储引擎，特别适合现在的应用程序。

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。

当然，由于作者水平有限，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，后续需要对存储引擎进行进一步的开发，加入日志等功能。

2.分布式数据库中数据迁移目前还没有实现，这是值得进一步研究的课题。

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。

作者认为，虽然现在出现了很多的Nosql数据库系统，但是关系型数据库的作用是无法替代的，电子商务和各种银行业务都需要关系型数据库的支持，所以对关系型数据库的学习和实践特别有意义。另外，数据库的安全问题越发严重，时常发生管理员篡改数据的情况发生，所以作者觉得，每个数据库系统，都要从底层加入安全审计功能，这样我们的数据才能保证最基本的可靠性。关于本系统，需要做的事情还有很多，但是我相信随着进一步的开发，本系统的功能会进一步完善。

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教授，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] 刘蓬. NIO高性能框架的研究与应用[D]. 长沙: 湖南大学, 2013, 50–60

[2] 杨东, 谢菲. 分布式数据库技术的研究与实现[J]. 电子科学技术, 2015, 02(01):68–71

[3] 杨传辉. 大规模分布式存储系统[M]. 北京: 机械工业出版社, 2013, 56–60

[4] 马应龙. 分布式系统概念和设计[M]. 北京: 机械工业出版社, 2013, 56–60

[5] 安延文. 数据库审计系统中MySQL协议的研究和解析[D]. 河北: 华北电力大学, 2013, 50–60

[6] 贺杰. 分布式数据库中数据复制的研究和实现[D]. 南京: 东南大学, 2014, 50–60

[7] 邓蕾. 基于关联规则的数据库安全审计系统[D]. 长沙: 中南大学, 2011, 50–60

[8] 黄贵. OceanBase分布式存储引擎[N]. 华东师范大学学报, 2014年9月

[9] 李黎明. 安全数据库概述和前瞻[R]. 北京: 南京航天大学信息与技术学院, 2005年5月

[10] Wikipedia. Hard disk drive [EB/OL]. https://en.wikipedia.org/wiki/Hard_disk_drive

[11] Wikipedia. B+ tree [EB/OL]. https://en.wikipedia.org/wiki/B

[12] Wikipedia. Log-structured merge-tree [EB/OL]. https://en.wikipedia.org/wiki/Log-structured_merge-tree

[13] Wikipedia. Netty [EB/OL]. https://en.wikipedia.org/wiki/Netty

[14] Wikipedia. Elasticsearch [EB/OL]. https://en.wikipedia.org/wiki/Elasticsearch

[15] Wikipedia. Apache JMeter [EB/OL]. https://en.wikipedia.org/wiki/Apache_JMeter

[16] Wikipedia. Hazelcast [EB/OL]. https://en.wikipedia.org/wiki/Hazelcast

[17] 王智慧. 安全数据库审计子系统[D]. 上海: 复旦大学, 2011, 50–60

[18] 张瑞芳. 分布式数据库的查询优化方法设计与实现[D]. 成都: 电子科技大学, 2010, 50–60

[19] 王威. MySQL 数据库源代码分析及存储引擎的设计[D]. 南京: 南京邮电大学, 2012, 50–60

[20] 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, 2012, 50–60

[21] 解辉. 嵌入式数据库的设计与实现[D]. 太原: 太原科技大学, 2008, 50–60

[22] 杨磊. 数据库安全审计检测系统的设计与实现[D]. 北京: 北京交通大学, 2014, 50–60

[23] 张忠能. 分布式数据库关键技术研究与应用[D]. 上海: 上海交通大学, 2015, 50–60

[24] 王琳. 数据库监控系统的设计与实现[D]. 天津: 南开大学, 2011, 50–60

[25] 谢鹏. 分布式数据库存储子系统设计与实现[D]. 成都: 电子科技大学, 2013, 50–60

[26] 张俊民. 在线审计建模与实现[D]. 河北: 华北电力大学, 2015, 50–60