申请上海交通大学工程硕士学位论文

基于 JAVA 的多数据库中间件的设计与实现

学校代码：	10248
作者姓名：	石明辉
学　　号：	1060379113
第一导师：	肖双九
第二导师：	杨景涛
学科专业：	软件工程
答辩日期：	2011 年 06 月 28 日

上海交通大学软件学院

2011 年　3　月

A Dissertation Submitted to Shanghai Jiao Tong University
for Master Degree of Engineering

THE DESIGN AND IMPLEMENTATION OF JAVA-BASED MULTI-DATABASE MIDDLEWARE

University Code：	10248
Author：	MingHui Shi
Student ID:	1060379113
Mentor 1：	ShuangJiu Xiao
Mentor 2:	JingTao Yang
Field：	Software Engineering
Date of Oral Defense：	

School of Software
Shanghai Jiaotong University

March., 2011

上海交通大学

学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：2011 年 7 月 5 日

上海交通大学

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□，在_____年解密后适用本授权书。

本学位论文属于

不保密☑。

（请在以上方框内打“√”）

学位论文作者签名：[signature]　　指导教师签名：[signature]

日期：2011年7月5日　　日期：2011年7月6日

上海交通大学硕士学位论文答辩决议书

姓　　名	石明辉	学号	1060379113	所在学科	软件工程
指导教师	肖双九	答辩日期	2011-06-28	答辩地点	徐汇校区新建楼2027房间
论文题目	基于JAVA的多数据库中间件的设计与实现				

投票表决结果：5 / 5 / 5 （同意票数/实到委员数/应到委员数）　答辩结论：☑通过　☐未通过

评语和决议：

论文提出了一种基于Java的多数据库中间件的解决方案，通过采用Java相关技术、可扩展标记语言XML技术以及数据库相关技术在前端客户端与后端各数据库之间引入一个中间件，通过该中间件来屏蔽这些数据库系统之间的差异，并最终形成一个统一的逻辑数据库。不但保持了这些数据库原有的自治性，而且使得用户像使用一个统一的关系数据库一样来使用多数据库。从多个角度对该中间件的运行状况进行了详细分析。并将其成功地应用于实际生产环境。

论文结构合理，论述清晰，反映作者已较好地掌握了计算机科学的基础理论和相关的专业技术知识。具有一定的解决工程实际问题的能力。

答辩时表达清楚，回答问题正确，经答辩委员会讨论（无记名投票表决），一致同意 石明辉 同学通过工程硕士学位论文答辩，并建议授予工程硕士学位。

2011年 6月 28日

答辩委员会成员签名	职务	姓名	职称	单位	签名
	主席	胡飞	教授	软件学院	
	委员	窦延平	副教授	〃	
	委员	姜丽红	〃	〃	
	委员	蔡鸿明	〃	〃	
	委员	唐新怀	副研	〃	
	秘书	李力	助研	〃	

基于 JAVA 的多数据库中间件的设计与实现

摘 要

随着网络技术和数据库技术的不断发展以及企业信息化建设的不断深入，企业中多种异构数据库并存变得越来越常见，如一个公司中不同的部门可能采用不同的数据库系统，而这些系统很有可能分布在不同的地点，并具有独立和自治等特性。同时，伴随企业应用需求的不断增加，用户希望能够透明的访问和处理这些来自多个数据库中的数据。然而，这些异构数据库的运行环境、数据存储方式、数据模型以及访问控制策略等的不同，导致了用户不能同时查询和处理各数据库上的数据，以至于使各数据库变成了一个个信息孤岛，大大阻碍了企业的内部数据流通、信息共享，继而可能引发企业高层的决策失误，对企业造成重大损失。

由于系统的复杂性、数据结构的多样性，企业也不可能将这些已有的异构数据库全部抛弃重建。因此，如何在不改变原有这些异构数据库的基础上，为用户提供一个透明的、集成的数据访问环境，使用户能以统一的模式和查询处理语言访问这些数据库，成为亟待解决的研究课题。

针对上述多数据库系统集成问题本文提出了一种基于 Java 的多数据库中间件的解决方案。该方案通过采用 Java 相关技术、可扩展标记语言(Extensible Markup Language)(简称 XML)技术以及数据库相关技术在前端客户端与后端各数据库之间引入一个中间件，通过该中间件来屏蔽这些数据库系统之间的差异，并最终形成一个统一的逻辑数据库。这样不但保持了这些数据库原有的自治性，而且使得用户像使用一个统一的关系数据库一样来使用这些多数据库。

本论文的主要研究工作和成果如下：

(1) 设计了多数据库中间件的整体架构，将其划分为连接管理模块、模式处理模块、查询处理模块、事务处理模块、数据处理模块以及安全控制模块等六大功能模块，并分别对每个功能模块进行了详细的需求分析、设计与实现；

(2) 通过采用实现一套符合 java 数据库连接(Java Data Base Connectivity)(简称 JDBC)规范的驱动程序、非阻塞式的 Java 多线程技术、安全散列算法(Secure Hash Algorithm)(简称 SHA)、网协(Internet Protocol)(简称 IP)控制技术以及虚拟数据库连接池的方法解决了客户端与服务端的连接处理、安全访问和数据处理等问题；

(3) 给出了采用 Java 语法生成器(Java Compiler Compiler)(简称 JavaCC)编译技术解决全局结构化查询语言(Structured Query Language)(简称 SQL)语句分解的方法，并通过

使用模式映射表、规则中间表以及两阶段提交查询技术分别解决了模式冲突、存储过程分发以及跨多库分组、汇总和排序等难题；

(4) 通过采用线程监控、心跳技术实现了后端数据库的动态加载以及负载均衡；

(5) 通过采用两阶段提交协议并实现 Java 事务应用编程接口(Java Transaction Application Programming Interface)(简称 JTA)解决了多数据库中间件的分布式事务处理问题。

本文最后从多个角度对该中间件的运行状况进行了详细分析，并将其成功的应用于了实际生产环境中。通过多数据库中间件的使用运行，可使企业已存的数据库系统和应用系统得以复用，信息资源得到相互整合与共享。同时，该文的系统设计与实现经验将为其它需要解决此类多数据库系统集成问题的项目组起到一个很好的借鉴作用。

关键词 多数据库中间件，JAVA，模式处理，查询处理，事务处理，安全控制

THE DESIGN AND IMPLEMENTATION OF JAVA-BASED MULTI-DATABASE MIDDLEWARE

ABSTRACT

With the network technology, database technology and enterprise information construction continues to evolve, the coexistence of multiple heterogeneous databases become more common. For example, a company in different departments may use different database systems, which most likely located in different locations and have the characteristics of independence and autonomy. With the increasing demand for enterprise applications, users want to transparently access and process data in multiple databases. However, in the operating environment, data storage, data model and access control strategies differ between heterogeneous databases, resulting in not sharing data and interoperability between the databases,and ultimately making each database into an information island,greatly hindered the flow of data , information sharing in the enterprise, in turn leading to senior corporate decision-making mistakes, causing significant losses of enterprises.

Because the system's complexity, the diversity of data structures, enterprises can not abandon all existing heterogeneous databases and rebuild mang new databases.Therefore, without changing the basis of these heterogeneous databases to provide users with a transparent, integrated data access environment, and enable users to a unified model and query processing language to access these databases, become an urgent research topic .

To address the intergration problem for multi-database system we propose a solution that is based on the Java multi-database middleware.The program through the use of Java related technologies, XML (Extensible Markup Language) and database related technologies and insert a middleware which is located in front of the client and back-end databases, and then through the middleware to mask the differences between these database systems, and eventually form a single logical database.This will not only maintain the autonomy of these databases, but also allow users like to use as a single relational database to use multiple databases.

The main research work and the results are as follows:

(1) This paper designs the overall structure of multi-database middleware, which is divided into six functional modules, namely, connection management module, model processing module, query processing module, the transaction processing module, data processing module, and security control module, and the requirements, design and implementation of each of these functional modules are analyzed in detail;

(2) By using to achieve a consistent standard JDBC (Java Data Base Connectivity) driver, Non-blocking Java multi-threading, SHA(Secure Hash Algorithm) for asymmetric cryptography, IP (Internet Protocol)technology, and method of virtual database connection pool to solve client and server connection handling, security access and data processing;

(3) Shows the method of using javaCC (Java Compiler Compiler) technology to solve Global SQL (Structured Query Language) statement decomposition and uses the pattern mapping tables, rule intermediate tables, and two-phase commit query technology to solve the model conflict, respectively, stored procedures distributed and acrossing multiple databases grouping, aggregation and sorting problems;

(4) Through the use of thread monitoring heartbeat technology to achieve dynamic load back-end database, and load balancing;

(5) By using two-phase commit protocol and implementing the JTA (Java Transaction Application Programming Interface) specification interface to solve the distributed multi-database middleware transaction processing issues.

(6) Finally, the operational status of the middleware is analyzed in detail from various angles, and the middleware is successfully used in a real production environment.

(7) By using multi-database middleware, all existing heterogeneous databases and application systems in the enterprises are reused, and information resources are integrated and shared with each other. At the same time, this system design and implementation experience will give other project teams which need to be resolved problem of multi-database integration to play a very good reference.

Keywords Multi-database Middleware, JAVA, Mode Processing, Query Processing, Transaction Processing, Security Control

目　录

1 绪 论

1.1 课题背景及来源

随着信息技术的不断发展，以及企业信息化建设的不断推进，企业按照业务发展的需要逐渐搭建了许多数据库系统。由于历史原因，使用的这些数据库可能分布在不同的网络区域，各数据库采用的数据存储方式、访问控制策略、数据库语言及事务并发控制技术等都可能不同。当初企业使用这些数据库的原因可能就是为了某一个部门或者某一个应用系统的需要，即一个应用系统只与某一个数据库系统进行连接使用。但随着企业不断发展壮大，多个部门、多个子公司之间或多个应用系统之间需要共享彼此数据。因此，如何集成企业中多个数据库使得应用系统或者客户端可同时与这些数据库进行连接访问通信，共享数据库数据资源成为了急需解决的重要问题。

由于目前各数据库厂商采用的数据库存储方式、运行环境、访问方式等存在差异，导致现有多数据库集成解决方案异常复杂，直至今日也还没有形成统一的多数据库集成标准，国内外研究者对此正进行着积极的探索研究。就目前来说，现有的多数据库集成解决方案主要有以下几类：

(1) 应用系统调用数据库驱动接口访问各数据库数据

应用系统通过在程序中硬编码调用各种数据库访问的驱动标准接口，如：JDBC，ODBC，OLE DB 等来获取和处理各数据库上的数据。这种方式给我们带来的弊端是应用程序中夹杂了大量的与数据库处理相关的源码，将系统业务的逻辑处理与数据处理混合在了一起。一旦数据库的数据逻辑结构或物理结构发生改变，应用程序也要做大量的调整和改变。这就为系统的稳定运行和系统的升级带来了诸多不便，也提高了系统的维护代价。

(2) 设计一种全局 SQL 数据库处理语言

在各个局部数据库的基础上建立一种全局的 SQL 处理语言，该全局 SQL 语句经某个中间服务代理处理器翻译成各个局部数据库本身能够识别的局部 SQL 语句，然后在各自数据库上执行处理，执行处理后的结果再经过此中间服务代理处理器返回给用户。使用此种方法的弊端是开发人员必须要重新学习一门新的语言，提高了系统的开发成本。

(3) Web Service 方式访问各数据库数据

用户使用 SOAP 和 HTTP 通过 Web 调用的方式来调用各数据库的访问对象，也就是说将对各数据库的访问，包装为 Web Services。由于 Web Services 是基于 XML 的，因此需要消耗大量的 CPU 和内存资源。因为 XML 数据需要经过多步处理才能被系统所使用。如，首先要校验，检查格式是否符合 XML 规范，应用程序定义(DTD 或 Schema)

是否符合语义规范；然后，还要进行解析；最终，还要转换为应用程序所需要的二进制表达式。因此，其对于多数据库中大数据量的处理来说是不合适的。

(4) 建立数据仓库或将数据集成抽取为平面文件存储

该方案采用将多数据库中的数据定时清理转换抽取至另一个大的数据库中存储或者转化为某种标准格式的数据（如：XML，HTML，文本，EXCEL 等）使其存放于某种可被外部用户访问的服务器上。这种解决方案的一个最大弱点是数据的实时性，可动态编辑更改及实时查询都较差。这种方式只适合那种数据变化频率不大的系统应用。

(5) 建立多数据库中间件

在客户端与多个局部数据之间建立一个多数据库中间件，由该多数据库中间件来负责与各个局部数据库之间的数据通信。客户端先发送一条数据查询处理命令给多数据库中间件，然后由该中间件来判断将到哪些数据库上去执行处理这些命令，最终将所运行的结果通过某种方式返回给客户端。使用此种方法进行多数据库集成的好处主要是避免了应用程序与数据库的高度耦合，解决了某些企业由于多种因素而无法进行数据库改造，数据库之间不能共享彼此数据所引起的信息孤岛问题。同时，也降低了企业多数据库集成的建设成本。这种方式也是本课题采用的多数据库集成解决方案。

本课题来源于上海期货交易所 NGESIII 期（新一代交易所系统第三期）统计信息系统项目中的一子课题，主要解决多个异构数据库的集成问题。近几年上海期货交易所交易数据逐年成倍增长以及数据库建设的历史原因，数据被分别存储于不同的异构数据库上。在此情况下，就需要开发出一种能够向用户提供一个统一的数据访问接口，屏蔽各局部数据库在物理和逻辑上的差异，实现对各局部数据库透明访问的多数据库中间件。

1.2 国内外研究现状

目前，国外对多数据库中间件的研究主要有：

由德国 GMD_IPSI 机构研发的 VIEWSYSTEM，它是一个面向对象的中间件，有自己的全局查询语言 VML，其公共数据类型被称为 VODAK[1]。VODAK 由 4 个部分组成，分别为对象，类型，类和方法。VML 查询语言支持可视化的视图操作，对于用户来说简单的查询比较直观且易于使用，但其有一个最大的缺点就是无法实现嵌套多链接的子查询，对于有些 SQL 语法不支持。

由加拿大阿尔伯特大学在 DIOM 项目中开发了一个基于协调器(DTC)的互操作管理体系结构的中间件，其很好的解决了查询处理功能，该查询处理模块由查询路由，动态查询执行计划器，查询结果聚合器以及查询接口管理器等构成[2,3]。该中间件最大的缺点是没有解决局部数据源的模式冲突问题。

由 IBM Almaden 研究中心研发的中间件 Carlic[4,5]，它能在不改变数据源结构的前提下，为多种异构数据源提供集成视图。其最大的特点是通过建立适配器的方法来集成这些数据并提供查询功能[6]。它最大的缺点是可扩展性不强，不能动态加载数据源。

由意大利的 Universit a di Roma Tre 数据库研究小组研制的 ARANEUS，该系统主要是集成 WEB 半结构化数据，其结构都为 XML 格式的，对各数据库的集成不支持[7]。

另外还有 HP 公司研制的 Pegasus[8,9]，斯坦福大学研制开发的 TSIMMIS[10,11,12]异构信息源集成系统，土耳其中东技术大学(MENU)研究开发的基于 CORBA 的多数据库系统[13]，美国微电子计算机技术公司(MCC)研究开发的 Carnot 系统，其主要解决企业分布应用异构信息源的集成问题[14,15]。

国内对多数据库中间件的研究主要有：

华中科技大学研制的 Panorama 系统，采用了扩展的结构化查询语言作为其全局查询语言，集成了多种数据库系统，文件系统以及 WEB 数据源[16]，但其并没有充分考虑对查询的优化，并引入了新的查询语言。

东北大学提出的基于 CORBA 的 SCOPE/CIMS，采用了 ODMG-93 协议中规定的对象数据模型和对象查询语言作为公共数据模型和查询语言[17]，由于其是基于特定系统的，因此其通用性扩展性受到了很大限制。

东南大学研制的 Galaxy 是一个基于 CORBA 的分布式异构数据源集成系统，使用对象集成模型(OIM)作为数据集成的公共模型，对象集成查询语言(OIQL)作为其查询语言，对查询优化考虑不多[18,19]。

另外，北京航空航天大学，国防科技大学，等都对多数据库系统集成的各个方面的问题进行了初步探讨。清华大学集中讨论了多数据库管理系统的体系结构问题，提出了一种可伸缩的结构方案，并给出这一方案在中国教育科研网上的一个实现 DBSHARE；中山大学软件所的联邦数据库系统 LNFDBS 以动态方式实现对 ORACLE、FOXPRO 等数据库管理系统的集成，仅在查询算法方面进行研究[20]等。

从以上国内外研究发现它们中的有些不支持 SQL 语句嵌套查询，有些没有解决局部数据库的模式冲突问题或者没有解决事务处理问题，有些则是对查询优化考虑不多，有些则是数据源适配器的扩展性不好，有些则是自定义了第三方查询语言，对开发人员来说要求很高。目前，多数据库中间件的研究还处于生长期，对于其研究一直是国内外数据库领域的一个研究热点，有很多技术问题还没有很好解决，市场上还未出现主流的产品，因此是开展该技术研究，推出自己产品的好时机[21]。

1.3 课题研究目的及意义

目前大多数企业应用都需要访问各种异构数据库，但是这些数据库却很可能分布在网络上各个不同的地方。为了满足这种需求，企业内必须要有一种能够支持同时访问这些异构数据库数据的系统平台。如何集成和访问这些数据的一个关键问题是要提供给用户一个统一的数据视图，从而屏蔽这些数据库的运行环境、网络环境及数据结构等各个方面的异构性，使得用户不必需要了解各数据库系统的知识，不必自己进行数据转换和汇总，只需要通过简便的全局查询就可以从企业内各数据库中获取各自所需的信息。对

于企业应用来说，能够同时访问各种异构数据库不仅是企业内本身发展的需要，也是适应外部环境的需要。通过建立多数据库中间件可以把原有孤立的多数据库系统中的数据进行整合，为各用户提供一个完整的统一的数据视图，从而达到充分利用现有各种数据库资源的目的。

本课题通过不改变上海期货交易所原有数据库系统数据结构的情况下，在用户与各异构数据库系统之间建立一个统一的数据访问平台，从而屏蔽掉与各异构数据库之间的差异，使得用户感觉就像在使用一个数据库一样。用户通过多数据库中间件可以对各局部数据库进行增删改查，能够处理全局事务。但除此之外，并不影响各局部自治数据库的单独运行，各局部事务的处理还是由各局部数据库来完成。通过多数据库中间件的建立，不仅避免了上海期货交易所内信息孤岛的出现，同时也避免了废弃原有数据库系统，重新建立新数据库系统的高昂代价和改造风险。数据的充分共享和利用，也使得交易所相关会员能够更加全面及时的了解掌握各品种、各合约的交易情况，为其做出防范和化解市场风险举措提供参考依据。同时也有助于形成公开公正的价格信号，保证期货市场健康稳定的发展，为政府宏观调控提供可靠的参考数据。最终有利于稳定国民经济，有利于市场经济体系的建立与完善。

同样，伴随着各企业内数据量的迅猛增长、信息共享范围的不断扩大以及对信息安全访问的迫切要求，多数据库系统之间的数据共享和集成策略已成为企业充分利用信息资源、牢牢把握竞争优势的关键因素。安全、可靠、稳定的多数据库中间件必将广泛应用于金融、电信、医疗、政府机关、教育、农业、军事、制造等领域。

因此，在当今全球化的网络环境下，研究如何建立高效安全稳定的多数据库中间件，不仅具有重要的理论意义，而且有着非常广泛的应用前景。

1.4 课题的研究内容与成果

本课题通过设计开发一个基于 JAVA 的安全、可靠、易扩展、易使用的多数据库中间件为企业内各数据库数据共享以及各应用系统同时访问修改各数据库中的数据提供一个统一的数据查询处理平台。在此设计与开发过程中主要解决了以下几个关键性问题：

(1) 如何创建、监听、释放客户端连接以及回收无效的客户端连接；
(2) 如何消除各局部数据库的语义、句法等差异；
(3) 如何将客户端发送过来的查询指令转发给后端各局部数据库系统执行，同时将后端反馈回来的指令转发给前端；
(4) 如何对全局 SQL，存储过程进行解析；
(5) 如何实现分布式事务处理；
(6) 如何实现多数据库中间件的安全访问及负载均衡；
(7) 如何实现数据的转换与网络传输。

通过集成使用各种 JAVA 和数据库等技术来解决实现本课题提出的多数据库中间件开发。该多数据库中间件的优势主要体现在以下几方面：

(1) 平台无关性

由于 JAVA 的平台无关性、面向对象、安全、高性能、分布式、多线程等特点，使得通过采用 JAVA 技术开发出来的多数据库中间件可以运行在不同的操作系统上。

(2) 可进行多数据库负载均衡

对于多数据库负载均衡功能的实现主要通过在后台建立一个监听线程对各数据库连接池进行心跳检测，并根据相应的均衡算法分配数据库连接给各客户端。

(3) 自定义 JDBC 驱动实现多数据库中间件通信

通过设计与实现一个遵守 JDBC3 协议的数据驱动器，使得用户可以通过客户端与中间件进行数据及命令的通信传递与处理。

(4) 支持解析遵守 ANSI SQL 标准的所有 SQL 语句

通过采用 javaCC 语法分析生成器，对用户传递给中间件的 SQL 语句进行分解，然后与已存的 SQL 分发规则进行判断比较，以此决定该 SQL 语句将会被分发至哪些数据库上运行。

(5) 可动态调整各异构数据库数据源

多数据库中间件在运行过程中支持动态的增加或者删除与各异构数据库的连接，通过在多数据库中间件的后台建立一个对各数据库连接的监视器，来动态维护该中间件与各数据库的连接，从而做到了无需重启中间件即可以实现到各数据库连接的调整，为多数据库的稳定运行提供了技术保障。

(6) 支持运行符合 JDBC 规范的存储过程

客户端发送调用某个存储过程的命令至中间件，中间件然后根据配置规则，转发至某几个数据库上执行，并将执行结果返回给客户端。

(7) 支持分布式事务处理

该中间件通过调用实现了 JTA 规范的程序以及 JDBC 驱动，使用两阶段提交协议来实现跨数据库的事务处理。

(8) 支持安全的访问控制策略

通过采用 SHA 安全散列非对称加密算法，实现口令的加密传输，提高多数据库中间件的安全访问控制。同时该中间件的用户名和密码与原有各数据库的用户名和密码完全独立，互不干涉，安全的控制了用户与中间件，中间件与各数据库的连接通信。

对于存在有多数据库并存的企业通过引入多数据库中间件，不仅使得已存的数据库系统和应用系统得以复用，信息资源相互整合与共享，而且可以使各数据库中的数据能够被更加高效地分析与使用，从而大大地节约企业在信息化建设中的人力和财力投入。

1.5 章节安排

全文共由 6 章组成：

第 1 章 绪论。阐述了课题研究的背景、来源、国内外研究的现状、目的、意义、内容和成果。

第 2 章 相关技术综述。首先分别对多数据库系统、中间件的定义、特征、分类等进行了详细的描述；然后阐明了多数据库中间件与前两者的关系以及介绍了多数据库中间件需要解决的几个主要关键技术问题；最后对在本课题研究中需要使用到的相关 JAVA 及数据库技术进行描述，以明确说明使用这些技术的意义和价值。

第 3 章 多数据库中间件需求分析。本章详细描述了多数据库中间件的总体需求、各主要功能模块的功能性需求以及其非功能性需求。

第 4 章 多数据库中间件的设计与实现。本章为本文的研究重点，设计并实现了多数据库中间件。其中，详细介绍了该多数据库中间件的设计原则，体系架构以及查询处理、事务处理、模式处理、数据处理、安全控制和连接管理等模块的设计开发原理及实现方案。

第 5 章 系统运行分析。首先阐述了系统的运行平台、物理部署方案；然后对实现后的多数据库中间件进行系统运行测试；最后详细分析了系统运行性能。

第 6 章 总结与展望。对论文的研究工作进行了总结，分析了尚待改进的部分，并对下一步的研究工作进行了展望。

2 相关技术综述

多数据库中间件，属于多数据库系统的一部分，负责协调客户端与多数据库系统中各个数据源之间的数据通信，其功能相当于传统数据库系统中的数据库管理系统，由于其又是一个位于逻辑业务层与数据访问层之间的独立的服务程序，因此它又属于一个中间件。本章以下部分将重点介绍多数据库系统、中间件的概念特征和分类以及在设计与实现多数据库中间件时使用到的相关技术。

2.1 多数据库系统

2.1.1 多数据库系统定义

多数据库系统(Multi-database System，MDBS)是对分布、异构、自治、已存在的数据源组成一个相互协作的系统，在不影响局部和自治性的基础上，构造一个相互协调的分布式软件系统，以支持在物理上分布的多个数据库的互操作和全局透明访问[17]。构成多数据库系统的局部数据库称为(Local-database Sytem LDBS) [17]。LDBS 分布在不同的网络节点上，在 MDBS 之上构建的中间件为用户实现了透明访问，这样就屏蔽了不同数据库在物理和逻辑上的差异，并且保证了 LDBS 拥有完全的自治性。其体系结构[22]如图 2-1 所示。

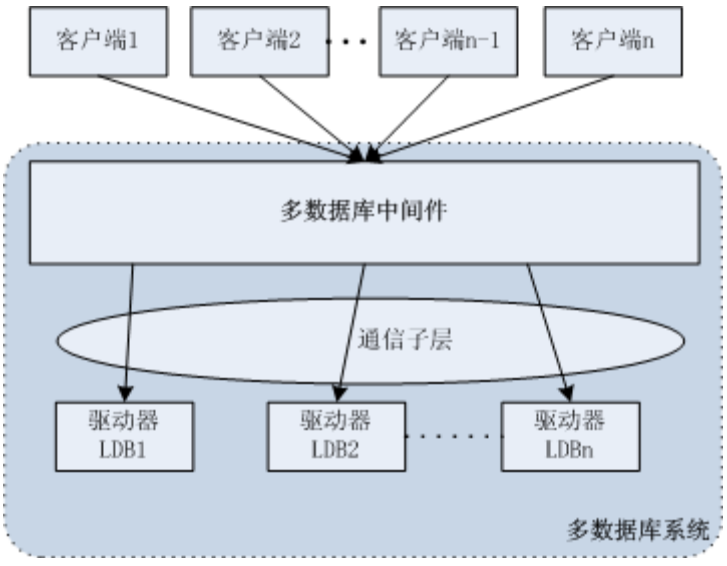

图 2-1 MDBS 体系结构

Fig. 2-1 MDBS Architecture

从图 2-1 可见多数据库系统是一种客户/服务器(Client/Server)结构。客户端通过多数据库中间件对多个局部数据库进行存取查询操作。多数据库系统通过中间件管理所有全局模式、全局查询的处理以及全局事务的提交和控制等。每个局部数据库通过一个自身

的驱动器与多数据库中间件连接，这个驱动器与相应的局部数据库系统在同一个站点上。多数据库中间件与驱动器之间的通信构成一个通信子层(CSS：Communication Sub System)[23]。多数据库系统是多数据库中间件与各局部数据库系统的有机结合，其并没有对各局部数据库系统做任何改动，保证了各局部数据库系统的自治性，同时也屏蔽了各局部数据库系统的分布性和异构性。

2.1.2 多数据库系统特点

多数据库系统的基本特征是异构性、分布性、 已存性以及自治性[24]。

异构性是指多数据库系统的各数据源在操作系统平台，网络通信协议及数据管理方式等方面存在差异性。从具体数据管理角度上看主要有数据模式，数据类型，数据存储，事务处理，访问控制方式等实现方式的差异。

分布性是指多数据库系统的各数据源不只是存在于某个单一场地的单个存储设备上，而是可能分散地存放于多个能够相互通信的局域网或广域网中。

已存性是指多数据库系统中的各数据源在构建之前就已经存在，已存性要求多数据库系统必须具备有动态加载多数据源的能力。

自治性是指各局部数据库系统是相互独立运行的，其自身有独立处理本身系统作业任务的能力，各自的操作并不受多数据库系统的影响，各自的查询处理及查询优化并不受多数据库系统全局查询执行的影响，其它局部数据库的加入或离开多数据库系统并不影响各局部数据库系统本身的正常运行。

多数据库系统是在数据库技术和网络技术发展到一定阶段的产物，与传统数据库系统不同的是它集成了分布在各个地理位置上的数据库，使得应用程序同时访问这些已存数据库成为可能。同时，原有的这些数据库仍然可独立运行而不受干扰。

2.1.3 多数据库系统分类

目前，对 MDBS 的分类并没有形成一个统一的标准，但研究比较多的主要有两类，联邦数据库系统和全局模式的多数据库系统。

联邦数据库系统(Federated Database System，FDBS)[25]是虚拟视图法的一种，也可称为没有全局模式的多数据库系统，目的是实现数据库系统间部分数据的共享。联邦中的每个数据库的操作是独立于其它数据库和联邦的，各联邦站点的用户使用本地查询语言可以访问其它站点的数据，FDBS 中实现互操作最常用的方法是将每个数据库模式分别和其它所有数据库模式进行映射，如果参与联邦的局部数据库很多的话显然不是一个可取的方法 ，这将加大系统映射的任务。如果联邦数据库中有其他的数据库，但是不与用户所在的接入节点相连，那么，用户将无权访问。

全局模式的多数据库系统提供一种多数据库全局查询语言[26]，全局用户使用全局查询语言对多数据库提出查询请求，多数据库向用户返回全局查询结果，因此，用户不必

关心这些数据是从哪几个局部数据库中获取得到的。多数据库系统将所有局部数据库系统的模式信息进行集成，消除各局部数据库系统存在的模式冲突。同时，局部用户只能访问自己数据库的数据，全局用户可通过多数据库系统访问所有各局部数据库系统。但是，不可以直接进入各局部数据库系统。

2.1.4 与分布式数据库系统的区别

多数据库系统与传统的分布式数据库系统还是有一定区别的，传统分布式数据库系统将组成的各个数据源看成是整个逻辑数据库的一部分且结构相关联，由一个数据库管理系统来管理，它对各个数据源上的数据操作有绝对的控制权，各数据源有统一的全局名字空间、数据模式和数据访问语言。因此，各个数据源在很大程度上已经不具有自治性了。而组成多数据库系统的各个数据源则有自己的名字空间、数据模式、访问语言，全局的数据库管理系统是无权控制各个数据源的数据库管理系统的。因此，组成它的各个数据源是完全自治的。由于结构的不同导致他们的数据处理方式也不一样，传统分布式数据库系统可以通过其统一的数据库管理系统对各个数据节点上的数据进行查询或存取，而多数据库系统在查询数据时，需要先将各个局部数据库中的数据抽取出来进行数据融合再转发给客户端，在操作数据时，需要解决全局事务与局部事务之间的冲突[27]等关系。

2.2 中间件

2.2.1 中间件定义

中间件是近年来才提出的概念，它是一种位于客户软件和操作系统之间的软件，属于可复用软件范畴[28]。其为应用程序的运行提供领域化的服务或功能，总的作用是为处于自己上层的应用软件提供运行与开发的环境，帮助用户灵活、高效地开发和集成复杂的应用软件。也许很难给中间件一个严格的定义，目前比较普遍被接受的是 IDC 定义的中间件概念，即：中间件是一种独立的系统软件或服务程序，分布式应用借助这种软件在不同的技术之间共享资源，中间件建立在客户机服务器的操作系统之上，管理计算资源和网络通信[29]。

中间件起到了衔接上层应用与底层平台互通的桥梁纽带作用，如：中间件有可能应用于应用程序与操作系统之间，WEB 应用与数据库系统之间等。因此，从逻辑角度看，中间件是位于两个有依赖关系的服务之间的软件体[30]；从功能角度来看，其主要目的是简化应用程序的使用和开发，并提供可靠的增值服务，屏蔽下层服务的复杂性，简化上层应用服务的开发和管理的软件体[31]；从复杂性角度来看，中间件是上层服务和下层服务之间通用的软件体[32]。总之，中间件的最终目的就是为了消除信息孤岛，促进信息的流动，支撑多变、动态、开放的互联网复杂应用系统，实现对分布于互联网之上的各种信息资源的集成、协同和综合利用。

2.2.2 中间件特点

一般来讲，中间件应具有以下特点[33]:

(1) 满足大量应用的需要:用户通过调用中间件的应用程序接口，实现异构环境的通讯，从而屏蔽异构系统中复杂的操作系统和网络协议。

(2) 平台无关性:中间件能够屏蔽各类通信协议之间的差异以及实现通信协议间的互通，并向外提供统一的接口，该接口的使用与具体的操作系统平台无关。

(3) 支持分布式计算，提供跨网络、硬件和操作系统平台的透明性的应用或服务。

(4) 支持标准的协议和接口:通过采用标准的协议和接口，当计算机硬件或系统软件改变时，与之相关联的应用软件无需任何修改。这样避免了重复的软件开发与维护，为企业节约了信息化建设成本。

(5) 良好的可靠性:中间件的运行必须是可靠和稳定的。

(6) 较高的效率:中间件主要是负责接收客户端的请求，查找数据源或服务，因此必须要有较高的运行效率。

中间件的产生归结于分布式应用的需要，它实现了集成应用系统之间的互通、互操作、协同处理、资源信息的共享等。它属于一类独立的系统软件或服务程序。中间件屏蔽了上层系统与底层各种异构和分布式的系统通信时所需要实现的各种复杂的技术细节。从软件复用的角度看，它是一种建立在操作系统、网络和数据库系统以及应用软件之间的可复用的结构或构建群。

2.2.3 中间件分类

由于中间件的应用范围十分广泛，针对不同的应用需求涌现出了多种各具特色的中间件产品，因此，在不同的角度或不同的层次上，对中间件的分类也会有所不同[34]。按照 IDC 的分类方法，其可分为终端仿真/屏幕转换中间件、数据访问中间件、远程过程调用中间件、消息中间件、交易中间件以及对象中间件[35]等六类中间件。文献[34]对其分类进行了详细分析与介绍。

除以上分类方法之外，还有另一种主流的分类方法，即将中间件划分为水平中间件和垂直中间件两大类。

水平中间件：充当分布式计算支撑环境。这类中间件有交易中间件、消息中间件、分布对象中间件等。

垂直中间件：面向具体领域，一般在水平中间件支撑下工作。例如，安全保密中间件、多媒体数据管理中间件、工商/银行/税务中间件等。

目前，水平中间件的发展较快，成熟的产品较多，但垂直中间件的发展才刚刚起步，需要研究、开发、完善的地方很多。

2.3 多数据库中间件

2.3.1 多数据库中间件概念

当客户端需要同时与多个局部数据库系统进行数据通信时，这时就需要有一种能够进行指令收发及数据整合的通信软件体来协助完成以上任务。该软件体还必须能够完成与多个数据库系统的安全访问控制、事务处理及全局查询，在功能上等同于单个数据库里的数据库管理系统。我们根据前述章节所讲述的多数据库系统概念可知该软件体应属于多数据库系统的一个有机组成部分。同时由于其运行的独立性以及在软件分层模型中位于连接客户端应用程序与后端数据库系统的中间层，因此它又属于一个中间件。通过以上两者概念的结合我们称此软件体为多数据库中间件。

多数据库中间件是因为网络技术、数据库技术的不断发展以及企业信息系统建设的需要而产生的。因此，其具有多数据源的异地分布性、各数据源的系统自治性、多库操作的整合性以及通信连接的独立性等特征。

按照中间件的 IDC 分类标准，多数据库中间件属于数据访问中间件，按照水平和垂直分类标准则属于垂直中间件，即属于解决数据库领域问题的中间件，提供全局的查询、标准的数据库操作方法、事务处理以及获取各个数据源中的数据并将其提交给业务逻辑层或更高层。

2.3.2 多数据库中间件的关键技术

在进行多数据库中间件的设计与实现过程中，必须要解决以下几个主要关键技术：

(1) 模式集成

模式是指数据的特征描述和逻辑结构，即数据源的逻辑组织形式。模式中不存放真正的数据，它只是用户所见的数据组织形式描述。由于局部模式间存在差异[36]，我们需要把不同的局部数据源模式即局部模式统一到一个一致的全局模式，该过程称为模式集成[37]，也称为异构模式消解。

组成多数据库系统中的各数据源存在对同一数据有不同的定义，如 A 库上的某一张表 t_product 中有一个字段 hight 其单位为厘米，而在 B 库上所对应的字段存储单位却为毫米，在此种情况下就出现了数据精度的的冲突。因此多数据库中间件必须能够解决同一字段名存在于不同数据库中的数据类型、数据格式以及数据精度等数据异构冲突。

另外，多数据库中间件必须还有解决语义冲突的能力，即能够识别处理在多个数据库中相同的字段名称代表不同的概念以及不同的字段名称代表同一概念的能力。例如，在 A 库上有张表的字段叫做 volume，其代表日成交量，而在 B 库上则代表月成交量，即同名不同义。而假如在 A 库中的某表字段 volume 和在 B 库中的某表字段 turnover 都表示同一成交量涵义时则为异名同义。

因此，如何处理多数据库之间的数据实体、属性、命名及结构等冲突，将局部数据库模式统一集成转换为全局数据库模式为用户提供一个统一的全局数据视图成为了本

文后续章节需要探讨解决的内容。

(2) 查询处理

当用户通过多数据库中间件查询各个局部数据库上的数据时，多数据库中间件必须要处理以下几个问题。首先要解决的是如何识别出用户的此次查询操作将会牵涉到后端的哪几个数据库；然后解决如何将查询语句发送至需要参与查询的各数据库上，如果使用的是某种全局查询语言还必须将全局查询语言转换为各局部数据库所能识别的本地查询语言；最后需要解决的是如何获取从各个局部数据库上返回的数据，以及考虑是否需要对这些数据进行汇总或者转换等操作。

同时，在多数据库中间件查询处理中，如何提高查询性能也是我们必须要考虑而不能回避的问题。影响多数据库中间件的查询性能的因素主要有查询语句解析开销、各局部数据库查询开销、数据网络传输开销、数据存储的 I/O 开销以及获取数据后的后续处理开销等几方面。

查询处理主要包含三个部分：查询分解、查询转换以及查询优化[38,39]。本文第四章将对其进行详细的设计与实现。

(3) 事务处理

事务指由一组与数据库相关操作的语句序列组成的一个独立的逻辑工作单元，即完成一个特定的应用[40]。事务具有 ACID（原子性、一致性、隔离性、永久性）[41]特性。

在多数据库系统中，多数据库中间件对各局部数据库中的数据进行访问修改操作时，也是通过事务处理来协调完成的，即对各数据库数据的操作要么全部成功处理，要么失败全部回滚，以保证各数据库中数据状态的一致性。

然而由于组成多数据库系统的各个局部数据库系统本身存在不同的事务处理的机制以及它们所在的网络环境可能异常复杂因此要保证多数据库事务严格的 ACID 特性是极其困难的。再加上各局部数据库的完全自治性，使得多数据库中间件无法知道是否还有除其发起之外的事务（局部事务）运行在这些局部数据库上。局部事务的出现有可能会导致多数据库系统产生全局事务间接冲突[42,43]。

根据以上的分析我们可将多数据库系统中的事务分为三类：全局事务、全局子事务以及局部事务。其中全局事务是指由多数据库中间件进行控制的事务，其真正执行时需要分解为多个子事务，即全局子事务；这些分解后的全局子事务只能在各局部数据库上执行，每一个全局子事务只负责与其局部数据库相关的数据存取工作。这些全局子事务的执行由与其相关的各局部数据库系统来控制。因此在各局部数据库系统中有两类事务：一类是多数据库中间件提交给其执行的全局子事务；另一类是不为多数据库中间件所知的，由其它应用系统提交给其执行的局部事务。

以上三类事务的出现以及相互关系将可能导致全局死锁、事务直接或间接冲突等问题发生，使得多数据库系统中的事务管理比传统数据库系统的事务管理更加复杂。因此如何保证多数据库系统中事务的 ACID 特性，如何进行事务提交、事务回滚、事务日志、

并发控制等一直是多数据库领域的研究热点。文献[44,45,46,47]对多数据库事务所面临的这些问题以及解决方法等进行了比较详细的研究与分析。本文也将在后续章节对多数据库事务处理进行详细设计与实现。

(4) 安全访问控制

访问控制是指通过某种途径显式地准许或限制主体对客体访问能力及范围的一种方法。目的是防止非法用户对系统的入侵以及给合法用户正确的系统授权操作。

由于多数据库系统集成了各种局部数据库，大量的数据需要在网络上进行传输与交换。因此，如何保护存储在各个局部数据库中的各类数据，防止非法用户对这些数据的非法访问和修改，并保证被授权的用户能在所授权的范围内对各 LDBS 中的数据进行访问和操作成为了多数据库中间件需要解决的一个关键技术问题。

在多数据库系统中需要解决的安全访问控制问题首先是如何授权控制用户登录多数据库中间件；然后是登录中间件后如何授权与各局部数据库连接，如有的数据库只允许读取操作，有的数据库则读写都可以，这些权限如何去分配控制等；最后是如何对多数据库中间件的用户密码进行加密，以防止密码在网络传输过程中被窃取。当然还有限制 IP 访问、访问控制审计等更多系统安全加固技术需要加以解决。

2.4 使用到的相关技术

为了完成多数据库中间件，并实现其运行平台的无关性，本文决定采用 Java 相关技术、XML 技术以及数据库相关技术来进行系统的开发和实现，以下将分别对这些使用到的技术进行简要介绍。

2.4.1 Java 相关技术

本文所采用的 Java 相关技术主要有以下几个：

(1) JDBC 驱动技术

JDBC 是一种可用于执行 SQL 语句的 java API 接口。由于 JDBC 具有对硬件平台、操作系统异构性的支持，因此本文采用 JDBC 可很容易地使用 SQL 语句实现异构多数据库的访问，为异构多数据库之间的互操作奠定了基础。文献[48,49,50,51]对 JDBC 驱动技术进行了详细的解释和阐述。

(2) Java 多线程技术

多线程是指在单个程序中同时运行多个线程完成不同的工作。文献[52,53]对其运行原理进行了详细分析。利用 Java 多线程编程接口及语言和运行支持系统提供的复杂的同步机制，开发人员可以方便地写出支持多线程的应用程序，有效地减少了多线程并行并发程序设计的困难，提高程序执行效率[54]。

在本课题研究中，通过采用 Java 多线程技术可以解决客户端与多数据库中间件以及多数据库中间件与后端局部数据库之间的并发连接问题。

(3) JavaCC 技术

JavaCC(Java Compiler Compiler)是一个用 JAVA 开发的语法分析生成器。它主要用于读取上下文无关且有着特殊意义的语法并把它转换成可以识别且匹配该语法的 JAVA 程序，文献[55]详细阐述了语法分析全过程。

本课题通过采用 JavaCC 语法分析生成器，来解析 SQL 语法，以此基础完成多数据库中间件的 SQL 语句路由功能。

(4) Java Socket 通信技术

Java Socket 用于客户/服务器模型的信息通信,客户端和服务器端通过一个双向的通讯连接来实现数据的交换。文献[56]对其通信原理和过程进行了详细分析。使用 Java Socket 通信不仅传输效率高，而且支持 client 与 server 端之间传输大数据量。

本课题通过采用 Java Socket 技术来实现客户端与多数据库中间件的连接通信。

(5) JTA 分布式事务处理技术

JTA 是一种高层的，与实现无关的、与协议无关的 API，应用程序和应用服务器可以使用 JTA 来访问事务[57]。

在本课题研究中，通过实现一个 Xid 类用来标识事务，采用两阶段提交协议[58]，来实现跨数据库的事务处理。

2.4.2 XML 技术

XML 包含三部分:DTD(Document Type Definition，文档类型定义)、XSL(Extensible Style Sheet Language，可扩展样式语言)和 XLink(Extensible Link Language，可扩展链接语言)，文献[59]对其进行了详细描述。XML 独立于机器平台、供应商以及编程语言，具有开放的国际化标准、高效可扩充性、良好的移植性能、良好的自描述性等特点，主要用于设定与特定领域有关的标记语言、自描述数据、应用间交换数据、表示结构化和集成的数据。将其与跨平台的开发语言 JAVA 进行结合[60]，这将为多数据库中间件的跨平台性提供可靠保证和开发基础。

2.4.3 数据库连接池技术

该技术是指创建和管理一个数据库连接的缓冲池技术，它将数据库连接作为一个对象保存在一个矢量数组中。当一个数据库连接被建立后，各不同的访问请求就能够共享使用这些数据库连接。文献[61,62]详细描述了其基本原理和优点。在本文中当多数据库中间件与后端各数据源进行连接时，通过采用数据库连接池技术减少了数据库连接频繁创建与关闭的开销，大大提高多数据库中间件的使用性能。

2.4.4 传输加密技术

在进行多数据库中间件开发时，需要对客户端连接该中间件时进行用户安全检查和授权，如何验证用户从客户端传过来的密码，以及密码以什么形式进行传输成为了数据库传输加密技术必须要解决的问题。本文采用非对称的安全散列加密算法 SHA[63,64]来进行口令的验证和传输，通过使用 SHA 加密算法用户可以随意发布公钥，完成数字签名，并能够使用很小的密钥组和数量对信息进行安全加密和网络传输。

在多数据库中间件中引入传输加密技术可以提高系统安全，为多个数据库的系统集成提供安全保障。

2.5 本章小结

本章首先介绍了多数据库系统、中间件的定义、特征及分类，并对多数据库系统与分布式系统的不同进行了简要描述，接着在以上分析的基础上介绍了多数据库中间件的概念，并对多数据库中间件需要解决的关键技术问题进行了简单概述，最后对设计与实现多数据库中间件需要使用到的相关技术进行了简单介绍，以明确说明使用这些技术的意义和价值。

3 多数据库中间件需求分析

软件需求分析是软件生命周期中极其重要的一步，主要是确定系统能够做什么，谁使用这个系统，我们分别称其为用例与角色。软件需求分析的过程具体可分为对问题的识别、分析与综合、制定规格说明书和评审。需求分析的结果是整个软件系统开发的基础，关系到工程的成败和软件产品的质量，是软件成败的决定因素之一，因此必须做到需求的完整性、正确性、可行性、无二义性以及可验证性[65]。

本章以下部分将从多数据库中间件的总体需求、功能性需求及非功能性需求等几个角度进行详细分析。

3.1 总体需求描述

本课题设计与实现的多数据库中间件是建立在上海期货交易所 NGESIII 期统计信息系统项目的基础上，是为了解决上海期货交易所数据量不断增大引发的数据分库存储后的数据共享处理问题，即解决一个应用程序如何与多个异构数据库同时进行通信及数据处理的问题。

多数据库中间件在该项目中主要起到一个在客户端与后端各异构数据库之间进行信息沟通的桥梁作用，其具体要实现的功能主要有：

(1) 创建一个客户端到中间件，中间件到后端数据库的安全连接；

(2) 解析从客户端发过来的标准 SQL 语句或存储过程；

(3) 将解析后的 SQL 语句或存储过程路由转发到相应的后端数据库上去执行；

(4) 将执行后的结果，通过中间件返回给客户端；

(5) 可实现跨多库的分布式事务处理；

(6) 可处理多数据库之间的语义冲突；

(7) 能够防止未经授权的用户非法访问中间件；

通过以上具体功能的实现，多数据库中间件为用户同时访问多个异构数据库上的数据提供了一个统一透明的处理视图，使其感觉就像是在使用一个数据库。对外部用户来说后端的这些多个数据库就变成为了一个独立的逻辑数据库。因此，用户不必再用关心数据的物理存储。

多数据库中间件的总体用例模型，如图 3-1 所示：

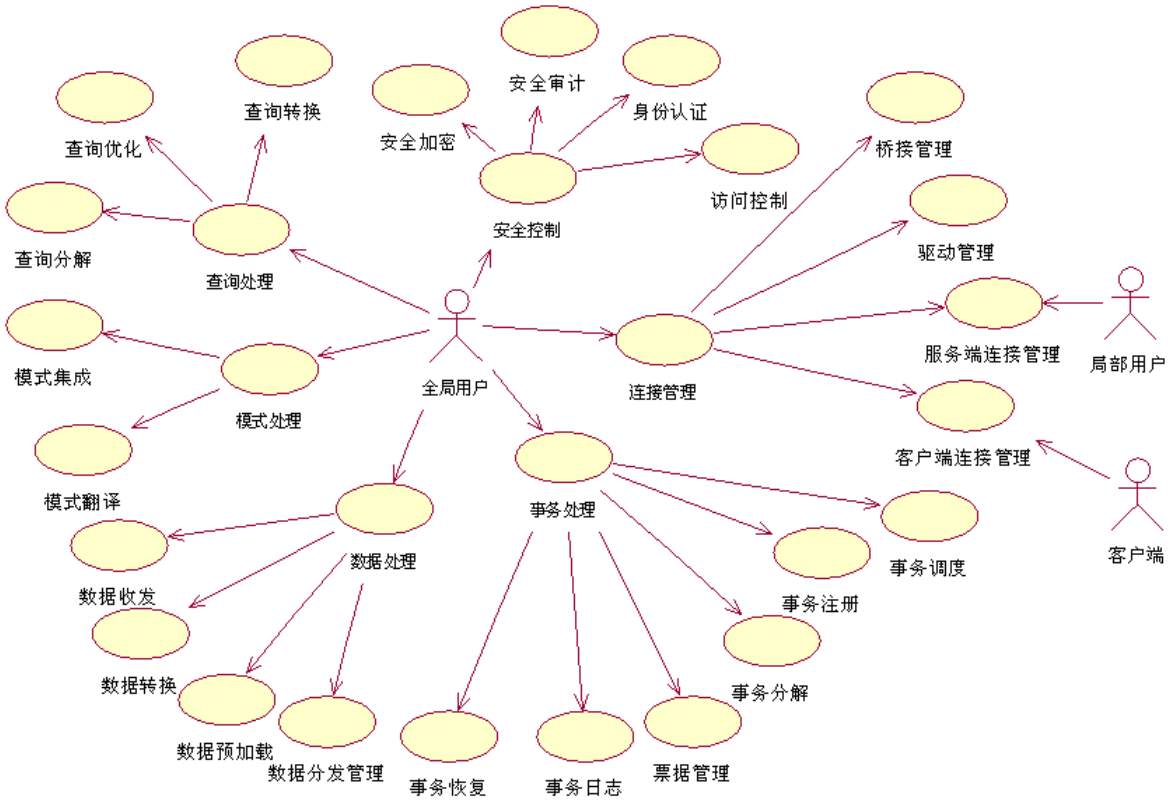

图 3-1 总体用例模型

Fig. 3-1 Overall use case model

3.2 各主要模块功能性需求

根据以上总体需求描述及总体用例模型图可知，多数据库中间件主要由连接管理模块、模式处理模块、查询处理模块、数据处理模块、事务处理模块以及安全控制模块等六大功能模块组成。本节以下部分内容将分别对这六大模块进行详细的需求分析，以便为多数据库中间件的后续设计与实现提供正确的指导思想和参考信息。

3.2.1 连接管理模块需求

连接管理主要负责客户端与多数据库中间件，多数据库中间件与后端各局部异构数据库的连接建立、监听、使用、关闭、回收或者销毁。各种查询处理命令及数据传输都建立在这些连接之上。因此连接管理在整个多数据库中间件中起到了一个信息载体的作用，为各种信息的流动提供了一个安全可靠的通道。

连接管理可分为四个子模块，分别为驱动管理、客户端连接管理、服务端连接管理以及桥接管理。它们分别主要实现以下功能：

(1) 驱动管理负责客户端命令的发起、数据的接收以及与多数据库中间件的命令和数据互动；

(2) 客户端连接管理负责监听客户端发送过来的连接请求并建立与客户端的连接，

处理客户端发送过来的各种查询处理命令，负责客户端连接的关闭、回收以及销毁各种无效的连接；

(3) 服务端连接管理负责多数据库中间件与后端各数据库的连接建立、关闭以及回收，同时负责向后端各数据库发起查询处理命令以及获取从各数据库返回的数据；

(4) 桥接管理主要负责实现客户端连接启动各种服务端连接，以此建立客户端连接与服务端连接的匹配关系，并能够识别出一个客户端连接正与哪几个服务端连接相关联，以便为命令的分发和数据的合并提供正确可靠的保障。

3.2.2 模式处理模块需求

模式处理模块主要完成将不同的局部数据库数据模式统一成一个全局数据模式，以此消除各种模式冲突，其主要模式冲突有：

(1) 字段类型冲突：同样的字段名称在不同的局部数据库中有不同的计量单位或数据类型；

(2) 同义异构冲突：相同的含义在不同的局部数据库中采用不同的数据字段结构表示，如：同样是日期，在 A 库上可能是用一个字段表示，而在 B 库上则用年，月，日等三个字段组合而成；

(3) 异义同构冲突：虽然在不同的局部数据库中的表字段名称和类型相同但其表达的含义不一样，如：在 A 库上 student 字段表示所有已毕业的学生，而在 B 库上则表示刚入学的新生等；

(4) 表结构冲突：一个数据库上的某张表存储的数据在另一个数据库上可能由多张表组合完成；

(5) 值对构冲突：相同的信息在 A 库上用字段值表示，而在 B 数据库上则可能用一个字段名称或者某张表的名称来表示；

(6) 构构冲突：相同的信息在 A 库上用一个字段名称来表示，而在 B 库上则可能用一个表的名称来表示。

在处理模式冲突过程中，数据模式结构被分为四层，分别为：局部模式、输出模式、全局模式以及外模式。其中，局部模式是用来描述 MDBS 中各局部数据源的信息；输出模式主要是将局部数据模式翻译成公共数据模型；全局模式是多个输出模式的集成；外模式主要是给用户提供某种特定视图。

模式处理模块分为模式翻译及模式集成两个子模块，首先通过模式翻译将各个数据库的局部模式翻译成一个用公共数据模型表示的输出模式，然后将各个输出模式通过模式集成统一为一个全局模式。一个典型的模式处理过程，如图 3-4 所示：

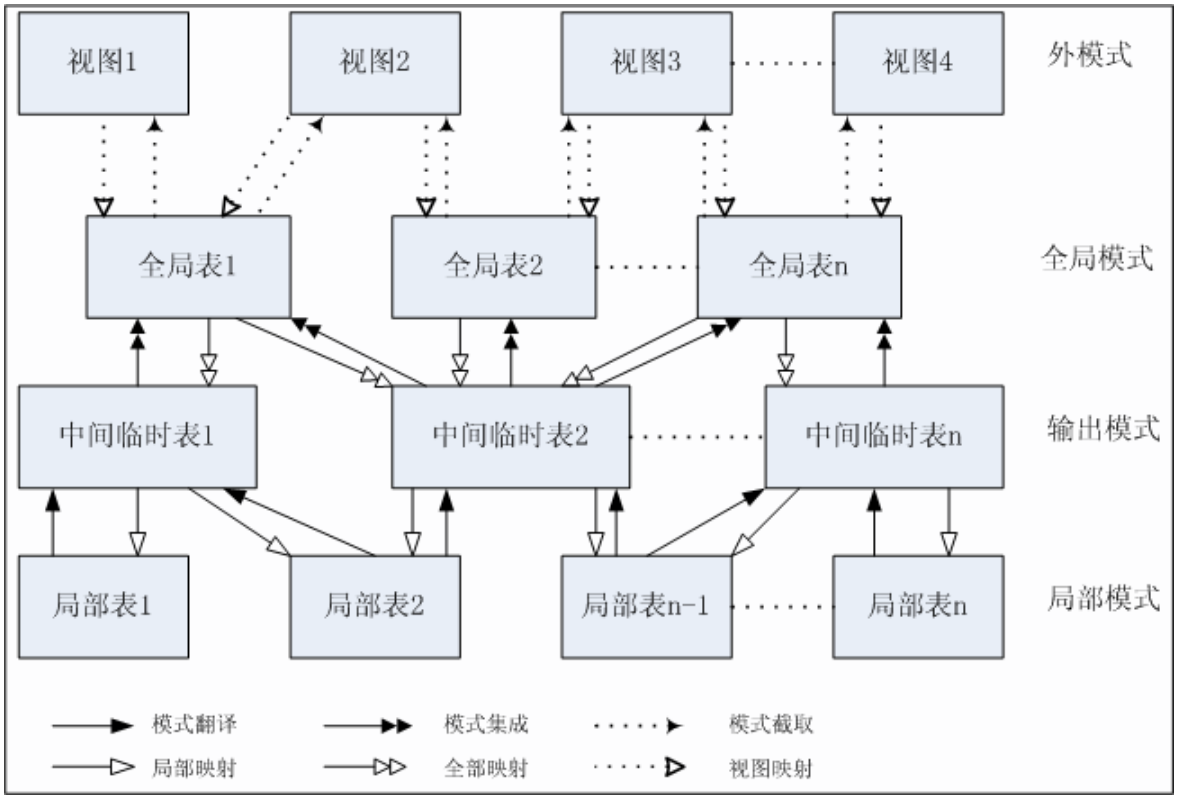

图 3-2 模式处理过程

Fig. 3-2 Model processing

模式处理的好坏关系到用户能否从各局部数据库上获取到正确的数据，是保持多数据库系统中数据一致性的一个基本环节。

3.2.3 查询处理模块需求

查询处理模块主要负责将查询命令分发至多数据库系统中的各个异构数据源上进行数据查询，并获取返回结果。其分为查询分解、查询转换及查询优化三个子模块。具体查询分为以下 4 个步骤：

(1) 全局查询根据表与数据库的模式匹配情况分解为多个子查询，每个子查询主要是为了获取对应各自的一个局部异构数据库中的数据；

(2) 把每一个全局子查询转换为各局部数据库能够识别的本地查询语言并执行；

(3) 判断子查询是否存在有查询嵌套调用，如有嵌套调用需要进行嵌套执行；

(4) 把每一个子查询得到的结果数据进行相关逻辑及合并处理返回给前端客户。

查询处理的过程如图 3-3 所示：

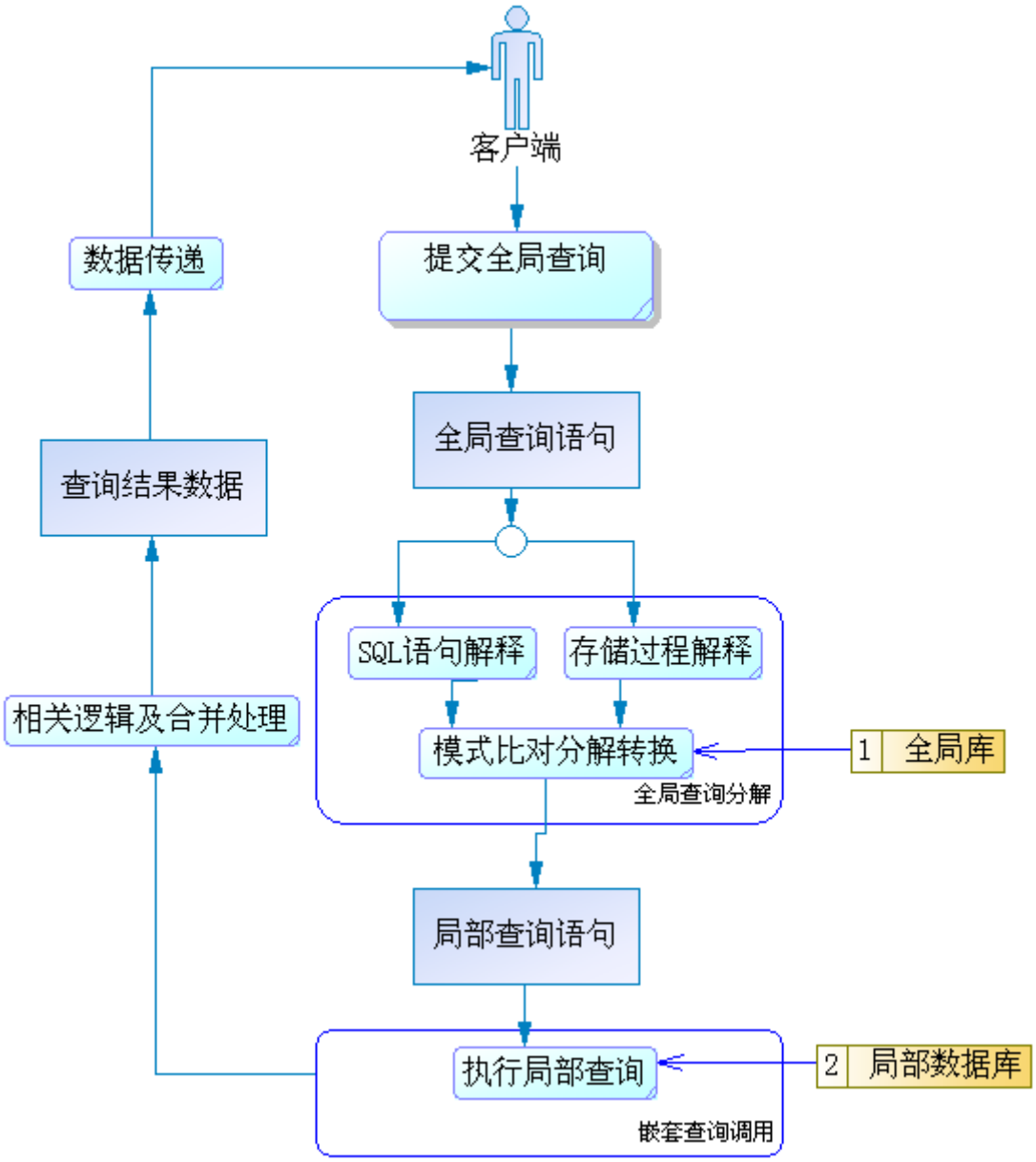

图 3-3 查询处理过程

Fig. 3-3 Query processing

同时，当出现跨多个数据库的数据排序、分组或者表连接查询时，多数据库中间件必须要能够给予支持和实现。查询处理模块是多数据库中间件需要实现的最基本功能。

3.2.4 数据处理模块需求

数据处理模块主要负责处理从参与查询的多个异构数据库中返回的数据，查询前的数据预加载以及处理客户端与中间件的数据互通。该模块由数据预加载、数据分发管理、数据转换以及数据收发等四个子模块构成。其分别主要实现以下功能：

(1) 数据预加载模块主要负责将一些初步汇总、分组的查询结果加载入会话级临时表中，为最后一步的总查询提供相关数据。

(2) 数据分发管理模块主要负责决定从局部数据库返回来的数据是直接返回给客户端还是等待所有数据查询完毕后再一次性返回。

(3) 数据转换模块主要负责将从各局部数据库返回来的数据转换为自定义 JDBC 驱动协议支持的 BYTE 流，以便在网络上进行传输。

(4) 数据收发模块主要负责将客户端发送过来的数据转发给中间件，或者将中间件中的数据转发给前端。

数据处理功能模块为全局用户获取到所需格式的正确数据提供了一个高效稳定的数据处理平台。

3.2.5 事务处理模块需求

事务处理模块主要负责多数据库系统中事务的分布式管理，以保证多数据库系统中数据的一致性，即多数据库中间件发往各局部异构数据库的子事务，要么全部成功提交完成，要么失败全部回滚。事务处理功能模块主要由事务注册、事务分解、事务调度、事务恢复、事务日志、票据管理等六大子模块组成。一个事务处理的全过程大致如下：

(1) 客户端发送一条事务处理命令给多数据库中间件，多数据库中间件通过事务注册、票据管理两子功能模块完成全局事务注册并分配全局事务 ID 号，即从事务管理容器中获取到一个事务处理对象以及事务 ID 号，以便进行事务的处理；

(2) 事务处理对象调用事务分解模块将此全局事务分解为多个全局子事务，其最终分解为多少个全局子事务，由事务处理语句所关联到的数据库个数来决定；

(3) 分解后通过事务调度模块调度各子事务至相关数据库执行；

(4) 在事务调度执行过程中将分两步完成各子事务的运行，第一步向各相关数据库发出事务执行请求，然后各数据库执行与自身相关的子事务，执行完后返回执行状态给事务调度模块；第二步，事务调度模块根据各数据库返回的事务状态判断出是要提交还是回滚，如果各子事务返回的都是准备提交的命令，那么将向各相关数据库发出提交的命令，如果有一个子事务返回错误状态，则调用事务回退命令对各相关数据库进行事务回滚；

(5) 如果正准备提交事务时系统出现宕机，则根据多数据库中间件自身所记录的事务日志进行事务恢复。

(6) 最后进行事务注销处理，将已结束事务的全局 ID 号以及产生的一些对象信息进行清空处理；

多数据库中间件的事务处理过程如图 3-4 所示：

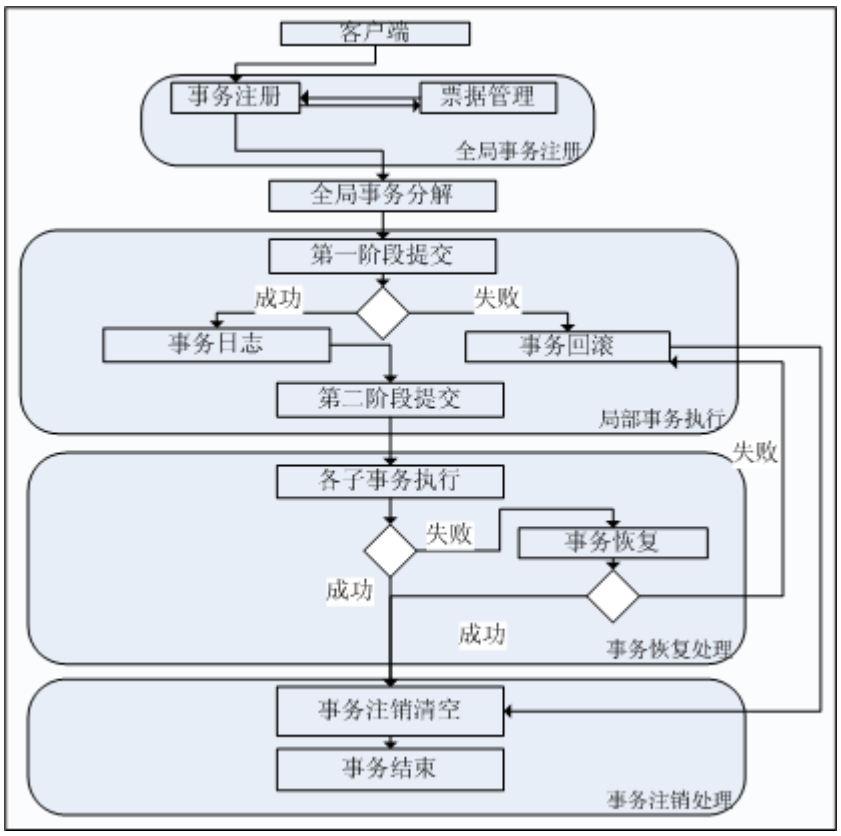

图 3-4 事务处理过程

Fig. 3-4 Transaction processing

3.2.6 安全控制模块需求

安全控制模块主要负责用户对多数据库系统的访问控制，在多数据库系统中，用户分为全局用户和局部用户，其中全局用户可以访问定义在全局模型中的全局表，而局部用户只能访问与此相关的局部数据库系统中的数据表。当全局用户需要访问局部数据库系统中的数据时，只能间接的通过局部用户进行访问。这样既保证了多数据库系统的安全也保证了局部数据库系统的自治性。

当全局用户需要登陆多数据库系统时，需要验证其真实身份，确定该用户是否是此 MDBS 的用户，同时确保该用户的 IP 为 MDBS 所允许连接的 IP。在局部用户进行真正数据库访问操作前，局部用户与各局部数据库系统之间要进行双向的身份验证，防止假冒攻击与侵入系统窃取数据；用户登录系统成功后，各局部数据库与多数据库系统之间进行数据传输之前要进行相互的身份确认；互相确认身份之后，通过建立加密后的数据传输通道，以保证数据在传输过程中不被窃听。

安全控制模块主要由身份认证、访问控制、安全加密以及安全审计等四个子模块组成。其分别主要实现以下功能：

(1) 身份认证主要负责对全局用户登陆多数据库系统进行认证，查看其传过来的用户标志、密码是否与多数据库中间件中保存的用户信息一致。如果不一致则拒绝该用户

登陆。

(2) 访问控制主要是对用户的操作权限进行控制，如对某个数据库的增删改权限的控制。还有就是授予或者拒绝只有哪些 IP 可以访问该多数据库中间件。

(3) 安全加密主要负责对用户信息以及重要的数据资料进行加密，以防止高度敏感的数据被盗取和篡改。

(4) 安全审计主要是负责把所有用户的登陆和数据操作记录进行跟踪保存，为以后的安全分析提供保障。

安全控制模块的体系结构，如图 3-5 所示：

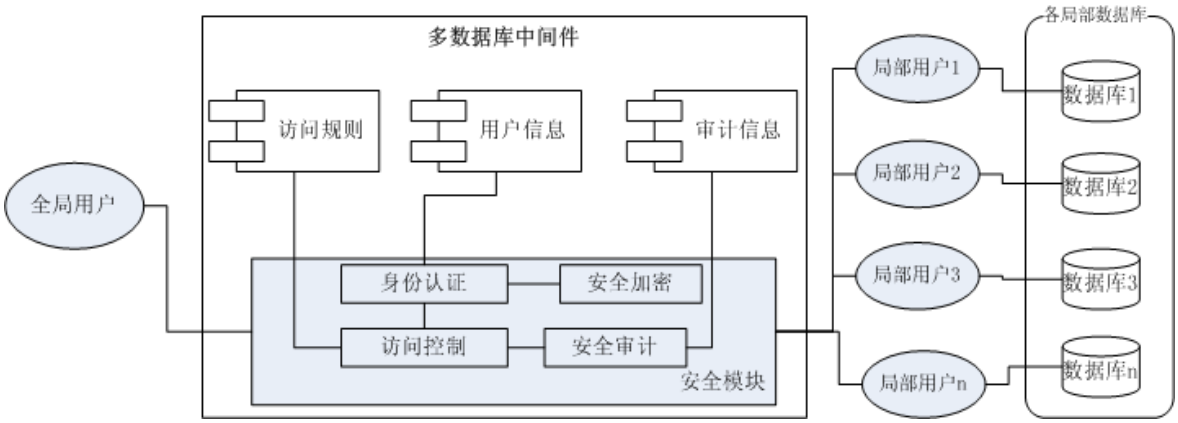

图 3-5 安全控制模块体系结构

Fig. 3-5 Security control module architecture

3.3 非功能性需求

非功能性需求是指为了满足用户需要而提出的除了功能性需求以外的需求。其主要包括有性能需求、可靠性需求以及其它需求，如：系统的可扩展性、可维护性、可移植性以及可操作性等需求。本节以下部分将就多数据库中间件的相关非功能性需求进行详细研究与分析。

3.3.1 性能需求

系统性能的高低直接影响到用户对系统的满意度，是用户在使用系统时最能直观感受出来的指标。多数据库中间件的性能需求主要体现在以下几个方面：

(1) 连接性能

如果后端有多个完全一样的数据库，多数据库中间件必须能够心跳检测，根据后端各数据库负载的大小动态调整发往这些数据库的连接，使得各数据库负载趋上均衡。

对于多数据库中间件与各异构数据库的连接必须建立连接池，通过连接的不断回收利用，减少重复创建连接的开销，大大提高多数据库系统的处理性能。

(2) 查询性能

在多数据库中间件与后端各局部异构数据库进行数据通信时必须能够支持查询处

理命令的并行运行，以缩短用户操作等待的时间。

同时，支持一定数量的 SQL 缓存，有些解析过的 SQL 可以进行一段时间的保存，避免相同的 SQL 语句再需要进行一次语法解析，以此提高系统的查询效率。

(3) 网络传输性能

数据在网络上传输时，需要考虑网络传输开销，因此在获取各数据库上的数据时必须考虑是否有必要在获取之前对各数据库上的数据进行一次粗略的分组汇总，以减少数据在网络上的传输量。

同时，在接收和发送数据时系统必须支持可配置网络缓冲区大小，通过合理的设置可减少客户端与中间件以及中间件与后端各异构数据库的网络通信开销和读写存储内存的 I/O 开销，进而可提高多数据库中间件的数据吞吐量。

3.3.2 可靠性需求

在系统运行过程中，必须要保证其运行的可靠性。当多数据库中间件发生重大异常而宕机时，必须能够保证可自动切换至另一台可备用的系统之上。同样，当后端某一数据库出现故障时，能够自动的连接到另一台备份数据库上。

同时，多数据库中间件必须要考虑大数据量的处理情况，当出现大数据量的合并或者分组排序时所消耗的内存数量必须要在系统可控范围内，如果超出了可控范围应该及时处理；对于超长时间的查询，多数据库中间件必须能够合理的处置而不会影响到其它查询的完成；对于非法的查询，系统必须能够及时规避，以免对系统造成致命影响而使系统瘫痪。如：用户误输入了跨 10 年以上数据的查询，该数据量超过上千万条记录，如果不及时处理，系统内存很有可能被快速消耗殆尽而对其它查询处理产生影响。

在系统开发过程中必须保证各迭代开发阶段实施的正确性，尽量减少以下各方面的人为错误。

(1) 设计错误：如算法错误，没有考虑到各领域模型中的特殊边界情况；

(2) 编码错误：如变量初始值设置错误，嵌套循环错误，未做垃圾回收处理导致内存溢出错误等；

(3) 测试错误：如测试用例编写错误，测试数据错误等；

(4) 相关文档错误：如发布版本错误，文档缺乏完整性，文档内容不一致等。

(5) 因此，为了保证以上各迭代开发阶段的正确性，必须要实施阶段进度管理，产生阶段质量评估报告，通过各阶段质量的检验确保差错及早排除，从而提高系统的可靠性。

3.3.3 其它需求

多数据库中间件在非功能性需求方面除了需要满足以上性能需求、可靠性需求以为还必须要满足可扩展性、可维护性、可移植性以及可操作性等其它需求。

可扩展性是指系统各功能模块能够很容易的被扩展，在本文中则指多数据库中间件能够支持对后端各数据库的动态扩展功能，即一个数据库节点的增加与删除，不需要重启系统，不会影响现有多数据库系统的正常运行。

可维护性是指在修正一次系统缺陷时的难易程度，本课题研究实现的多数据库中间件是按照功能模块进行设计开发的，某个模块缺陷的修改基本上不会影响到其它的模块。因此，具有良好的可维护性。

可移植性是指系统从一种运行环境转移到另一种运行环境时所花费工作量的大小，本课题研究开发的多数据库中间件必须要实现平台运行的无关性，即该中间件不管被迁移到何种操作环境都能正常运行，实现真正意义上的一次开发多平台运行。

可操作性是指准备输入、操作和理解系统输出的难易程度，本文可操作性主要体现在用户只要通过很少的设置就可以实现数据库的读写分离，数据的切分，SQL 语句的自动路由以及查询过滤等功能。用户使用该多数据库中间件的时候不必关心后端各异构数据库的分布位置及存储方式。对于用户来说多数据库中间件就像一个虚拟的数据库。因此，其应该具有很好的可操作性。

3.4 本章小结

本章首先对多数据库中间件的总体需求进行了描述，然后对连接管理、模式处理、查询处理、数据处理、事务处理以及安全控制等六大主要模块的功能性需求进行了详细分析，最后对性能、可靠性、可扩展性、可维护性、可移植性以及可操作性等非功能性需求进行了详细研究与探讨。目的是通过以上各种需求分析为多数据库中间件的设计与实现提供可靠依据和保障。

4 多数据库中间件的设计与实现

根据前述章节介绍的 JAVA 相关技术及数据库技术并在以上需求分析的基础上，本章节将对多数据库中间件的具体设计与实现进行详细描述与分析。具体主要包括系统设计原则、体系架构以及主要功能模块的设计与实现等内容。

4.1 设计原则

系统设计的好坏决定了一个软件系统本身的优劣，差的系统设计必然会导致产生一个差的软件系统。因此，本课题研究开发的多数据库中间件必须遵守以下设计原则：

(1) 合适性：设计的体系结构须符合上述章节所描述的功能性需求和非功能性需求；

(2) 简单性：系统设计尽量简单，以减少系统内数据处理的通信开销、I/O 开销、CPU 开销以及使用内存等开销，同时能为终端用户和开发人员提供便捷简单的使用环境；

(3) 稳定性：体系架构一旦设计完成后，不能因为体系结构的经常变动而对后续的详细设计、编程和测试产生巨大影响而导致整个项目发生混乱；

(4) 一致性：系统设计和开发应采用统一的规范标准，便于以后系统的维护；

(5) 扩展性：系统必须能够方便迅速的进行扩展以此满足日益变化的业务需求，模块与模块之间仅仅通过彼此的 API 相互通信，而不需理会模块内部的工作细节；

(6) 复用性：首先系统中的代码、算法以及数据结构必须要有较高的复用性，然后使用面向对象的设计方法，采用抽象模块的设计理念，最后还必须要遵守“开-闭”原则、迪米特法则、接口隔离原则、里氏代换原则、依赖倒转原则以及组合/聚合原则等设计原则[66]以提高系统的复用性。

(7) 安全性：确保只有授予过本系统操作权限的用户才可以使用该系统；同时系统能够防止各种网络攻击和入侵的活动。

4.2 体系架构

体系架构设计分为逻辑架构设计和物理架构设计，本小节以下部分内容将从这两个方面进行阐述。

4.2.1 逻辑架构

本课题研究的多数据库中间件共分为 6 大功能模块，其总体逻辑结构如图 4-1 所示：

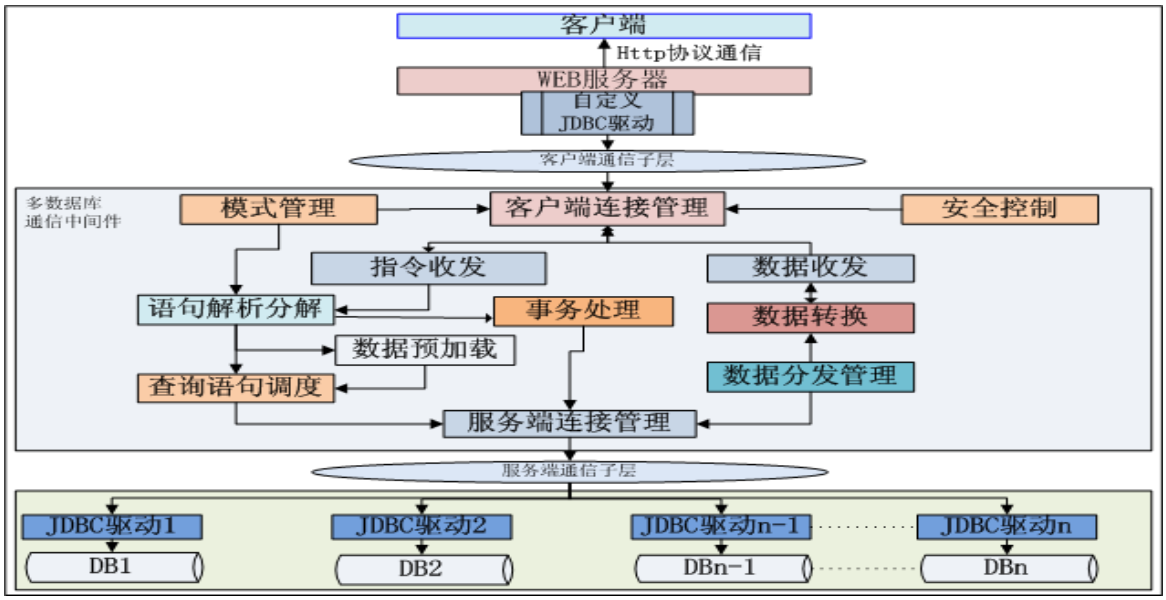

图 4-1 系统总体逻辑结构

Fig. 4-1 Overall system logical structure

4.2.2 物理架构

由于本课题是上海期货交易所 NGESIII 期统计信息系统项目中的一子课题，因此该多数据库中间件的物理架构是以该项目为前提背景而进行设计的。具体物理架构如图 4-2 所示：

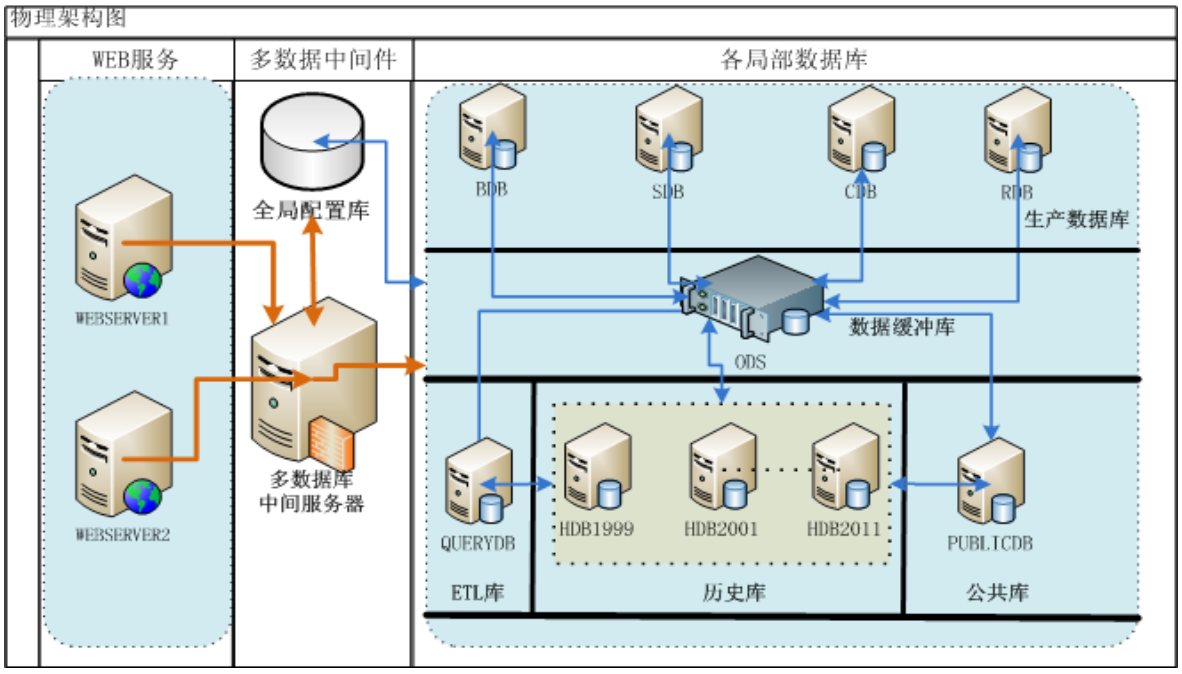

图 4-2 物理架构

Fig. 4-2 Physical architecture

图 4-2 中的全局配置库主要用于存储多数据库中间件的一些模式规则匹配表、全局表、预加载数据存储表、汇总表以及一些过渡性的中间临时表。

4.3 主要功能模块设计与实现

本节以下部分将根据前述章节的需求分析对如何设计与实现各主要功能模块进行详细介绍与分析。

4.3.1 连接管理模块设计与实现

根据前述连接管理模块需求分析可知，该模块由驱动管理、客户端连接管理、服务端连接管理以及桥接管理等四个子模块组成，以下将就这四个子模块的具体设计与实现进行详细分析。

(1) 驱动管理子模块

该功能子模块主要是实现一个自定义的 JDBC3.0 类型驱动程序，为各客户端连接到多数据库中间件时提供一个网络通信通道。具体实现的功能有与中间件连接的建立，客户端命令的发起，为从中间件中返回给客户端的数据提供传输通道。

具体主要通过创建以下 JAVA 类以实现 JDBC 规范中定义的相关接口：

Driver 类：实现 java.sql.Driver 接口中的方法，并设置相关属性，创建连接，同时向 java.sql.DriverManager 注册；

Connection 类：实现 java.sql.Connection 接口中的相关方法，该类主要负责创建与多数据库中间件的 IO channel 以及创建发送 SQL 或者存储过程到多数据库中间件的 Statement 对象；

ShfeIO 类：通过该类主要实现与多数据库中间件的通信，如与中间件建立握手通信、创建 socket、数据包的控制（校验包、发送包、读取包等）、关闭流、清空流、获取中间件的状态等；

Statement，PreparedStatement，CallableStatement 类：Statement 类主要用来执行一个静态的 SQL 声明，以及获取从中间件返回的结果； PrepareStatement 类可以预编译和存储一个动态的带参数的 SQL 声明，该对象可以对预编译过的 SQL 进行高效的多次执行；CallStatement 对象用于调用存储过程并从中间件获取返回结果；

ResultSet 类：该类通过实现 java.sql.ResultSet 接口，访问由 Statement 对象执行后产生的表数据集；

RowDataDynamic、RowDataStatic 类：RowDataDynamic 类主要是为了实现以流的形式读取数据集，比如用户需要从中间件中每加载 1000 条记录到内存后即开始读取数据，而不是一次性从中间件中加载完所有记录后才开始读取数据。RowDataStatic 类主要是为了实现一次性从内存中读取数据集，其方法名与 RowDataDynamic 一致，只是实现的方式不一样，这两个类都是实现了 RowData 接口；

ResultSetMetaData 类：该类实现了 java.sql.ResultSetMetaData 接口，主要是为了获得存储在数据库中的表字段名称、表字段的类型等。

以上 JAVA 类的类图之间关系如图 4-3 所示：

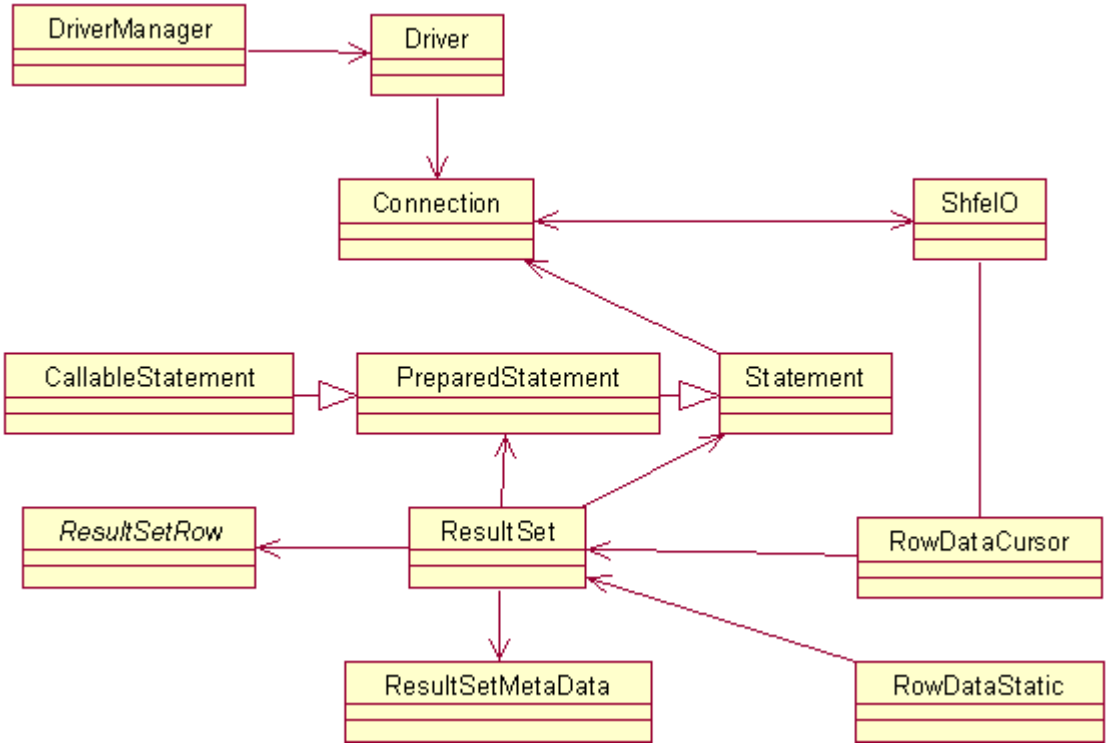

图 4-3 驱动管理类图

Fig. 4-3 Driver Manager Class Diagram

在客户端一个 SQL 查询处理的流程大致如下：

(a) 通过 Class.forName()方法加载本文实现的 JDBC 驱动，向 DriverManager 进行注册；

(b) 通过调用 DriverManager.getConnection(url,username,password)方法获取连接对象。在调用这个方法时主要完成了 socket 通道的创建、与多数据库中间件的握手通信；

(c) 通过上述创建的 Connect 对象调用 prepareStatement(sql)方法创建 Statement 对象；

(d) 创建后的 Statement 通过执行 executeQuery()方法进行 SQL 查询返回 ResultSet 结果记录集对象，通过执行 executeUpdate()方法完成增删改的 SQL 操作，通过执行 execute()方法即可以查询也可以更新数据库数据，如果执行的是查询语句则支持返回多个结果集，并通过 getResultSet()或者 getMoreResults 获取返回的当前或者下一个结果集。

(e) 获取到 ResultSet 结果集后，通过循环调用 next()、getString()等方法不断获取下一条记录直至指针移至到最后一条记录。

(f) 结果集获取后相继调用 ResultSet、Statement 以及 Connect 中的 close()方法清空缓存的数据释放内存，关闭与中间件的连接通道释放连接。

以上通过调用 JDBC 执行某个 SQL 或存储过程的时序图如图 4-4 所示：

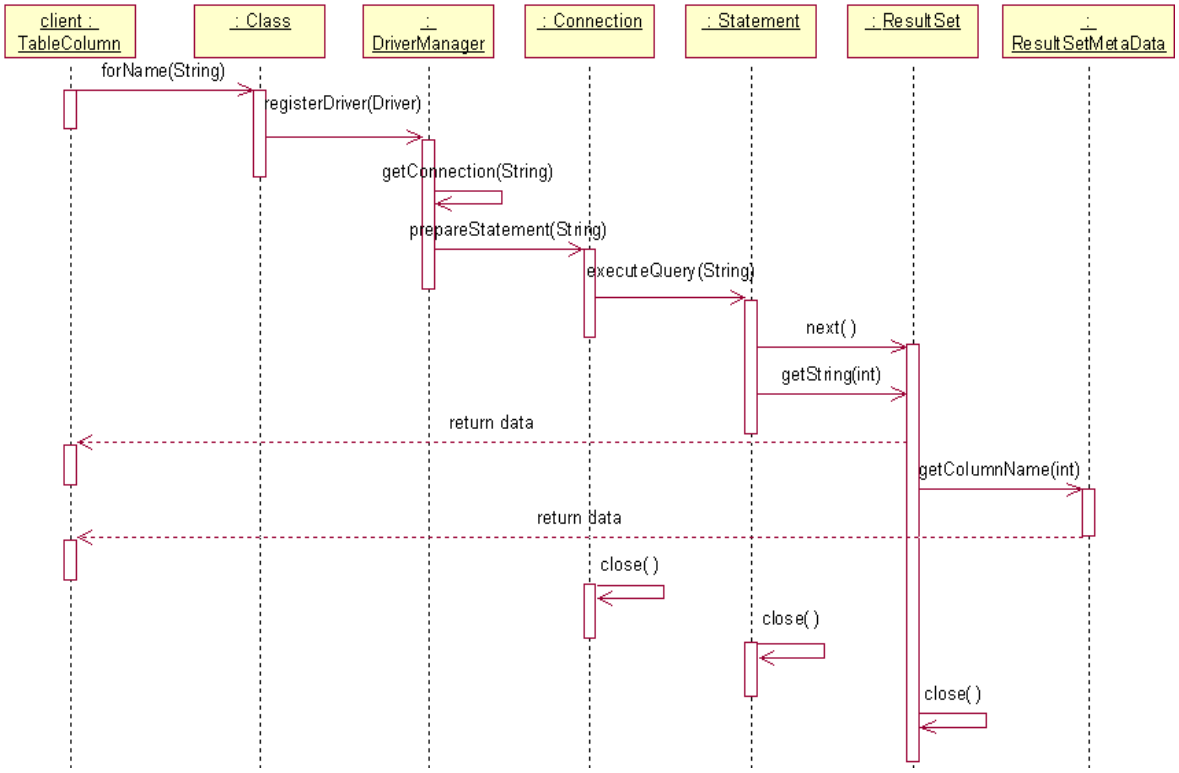

图 4-4 驱动管理时序图

Fig. 4-4 Driven management sequence diagram

(2) 客户端连接管理子模块

客户端连接管理主要完成中间件与客户端连接的建立，以及管理这些连接的运行状态，根据状态判断是否需要对这些连接进行销毁或者回收。

本文通过采用 JAVA 提供的 java.nio.channels 包中的类来实现客户端与中间件连接处理。该包中主要有两大类 Channel 和 Selector，分别称为通道和选择器。通道是指连接到一个网络套接字的开放连接，其可以被异步关闭和中断。例如，一个线程在某条通道的 I/O 操作上阻塞时，那么另一个线程可以将这条通道关闭。选择器被看成是一个事件处理器，负责将从客户端发过来的事件分派到相应的通道上进行处理。该两大类的结合使用主要是实现 NIO 通信，即非阻塞式的 I/O 通信，以降低服务器线程开销，提高服务器的性能。

具体连接处理实现步骤如下：

第 1 步：打开一个未绑定具体 IP 地址和端口号的服务端套接字通道；

第 2 步：设置此通道的阻塞模式为非阻塞模式；

第 3 步：绑定此服务套接字到某一个具体的 IP 地址和端口号上；

第 4 步：将通道向 Selector 注册，并返回选择器关键字，以便后续服务器监听使用；

通过以上 4 步的处理后服务端可以接收客户端发送过来的连接请求了。

第 5 步：调用 accept 方法接受连接，产生服务器端对应的 SocketChannel；

第 6 步：将该 channel 设置为非阻塞模式并注册到 selector；

第 7 步：根据 selkey 中的值判断是否有需要处理的命令，并进行处理。

以上实现方式示意如图 4-5 所示：

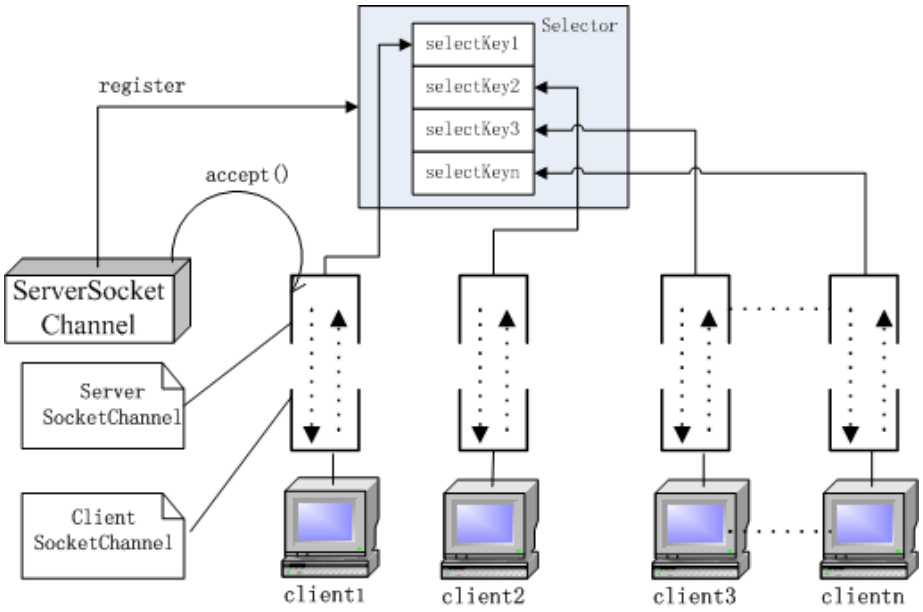

图 4-5 客户端连接管理示意图

Fig. 4-5 Schematic diagram of the client connection management

以上解决的只是中间件如何处理客户端的连接请求，下面将阐述客户端是如何与多数据库中间件建立通信的。

首先客户端在获取连接对象时创建一个输入输出对象用于处理与多数据库中间件的通信，然后创建套接字的工厂实例，该工厂创建到指定服务器 IP 地址及端口的客户端 Socket 并建立与服务器端的连接，当服务端接收客户端的连接请求时，客户端发送握手连接请求消息，最后服务端进行握手校验，发送应答消息，握手成功后即可进行下一步的 SQL 命令处理。

其具体的通信流程示意如图 4-6 所示：

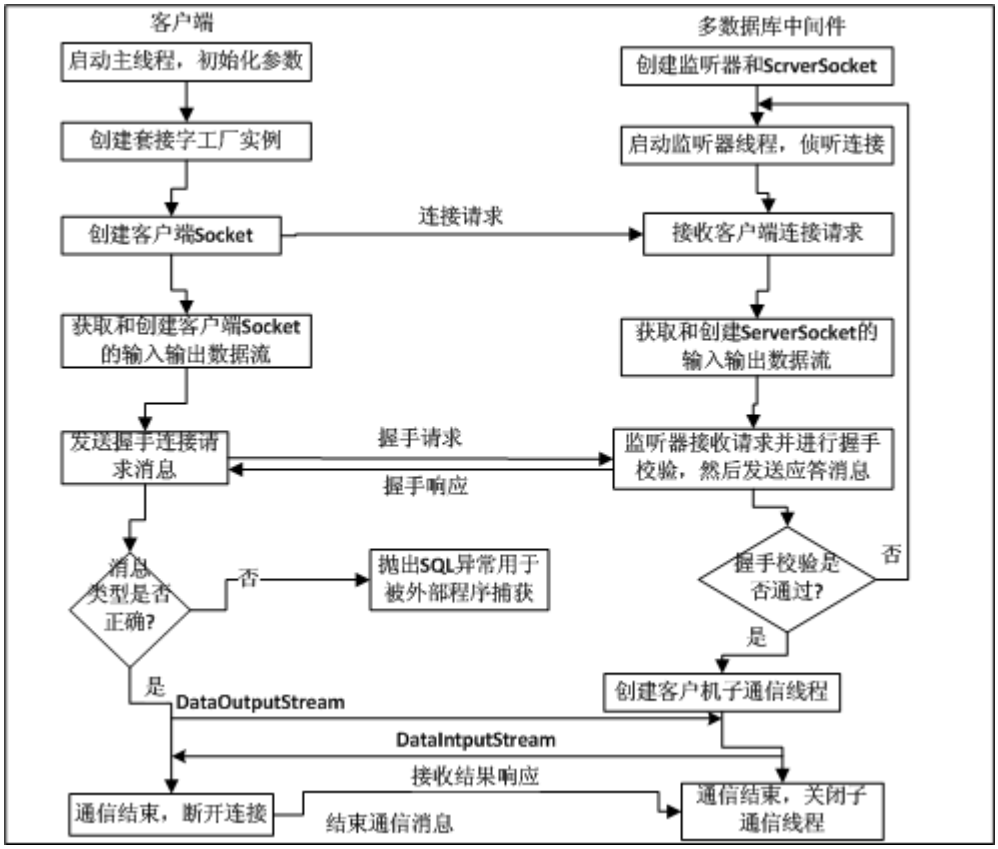

图 4-6 客户端连接通信流程示意图

Fig. 4-6 Communication flow diagram of client connections

以上产生的各客户端连接主要通过 ConnectManager 类来进行管理，具体管理的时序图如图 4-7 所示：

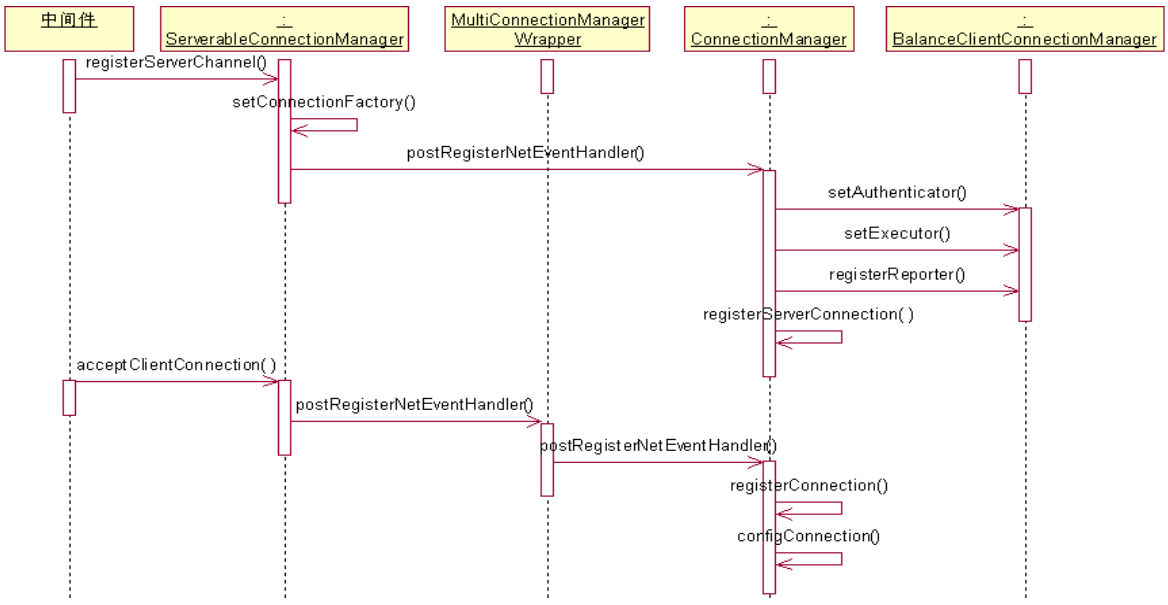

图 4-7 客户端连接管理时序图

Fig. 4-7 Sequence diagram of the client connection management

(3) 服务端连接管理子模块

服务端连接管理主要负责多数据库中间件与后端各局部异构数据库的连接，该连接管理的方式通过采用连接池的原理机制来实现，具体实现步骤如下：

(a) 当多数据库中间件进行启动时对其与各异构数据库的连接进行连接池初始化；

本文将各局部数据库的数据源连接信息配置到一个 xml 文件中。初始化工作首先是将该文件加载至 Document 对象中，接着使用该对象中的相关方法将连接信息加载入 ProxyServerConfig 对象中，然后通过调用 ProxyServerConfig.getDbServers()方法获取到文件中所表示的 dbServer。再通过调用 dbServerConfig.getPoolConfig()方法得到 poolConfig 类名称，最后采用 JAVA 的反射机制原理获取到连接池对象。

连接的对象获取后，通过调用初始化连接池的方法完成连接池的初始化工作，为了支持分布式事务这些初始化后的连接池都实现了 javax.sql.XADataSource 接口，连接池中的连接都实现了 javax.sql.XAConnection 接口。

从连接池建立到初始化完成整个过程的时序图如图 4-8 所示：

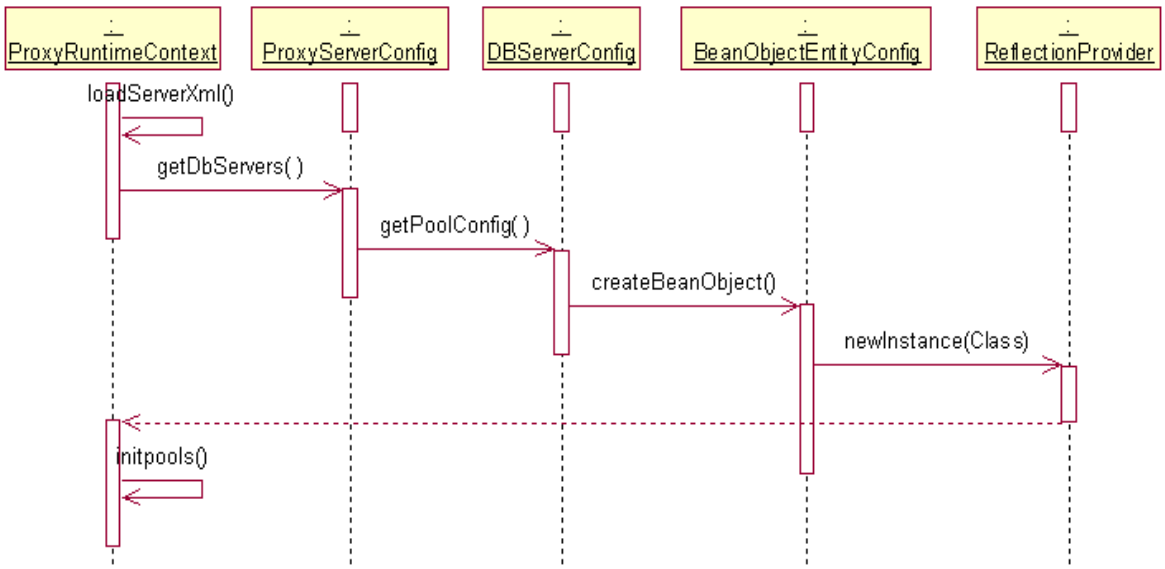

图 4-8 服务端连接管理时序图

Fig. 4-8 Sequence diagram of the service connection management

(b) 连接池初始化完后通过 borrowObject()方法获取连接或者通过 returnObject()方法释放连接；

(c) 多数据库的负载均衡处理；

多数据库负载均衡功能的实现主要通过在后台建立一个监听线程，该线程通过调用一个实现了 java.util.concurrent.Delayed 接口的 HeartbeatDelayed 类对各数据库连接池进行心跳检测，该线程的 run()方法具体实现代码参考附录 1。

当需要从各连接池中取出一个连接时，具体负载均衡算法有二种，其实现方式如下：

第一种：请求按照轮询的方式平均分配到每个连接池中

首先获取存放有各有效连接池的对象数组，然后计算出此次连接的历史排位，最后

算出历史排位除以总共连接池个数所得的余数，接着从连接池数组中取出排在余数位置上的连接池，作为本次连接的获取数据源；

第二种：按连接池的繁忙程度对请求的连接进行分配，其算法为将所有连接池中处于活跃状态的连接数量做一个排序，存在最小活跃数量的 pool 将被优先分配连接请求

其心跳检测示意状态图如图 4-9 所示：

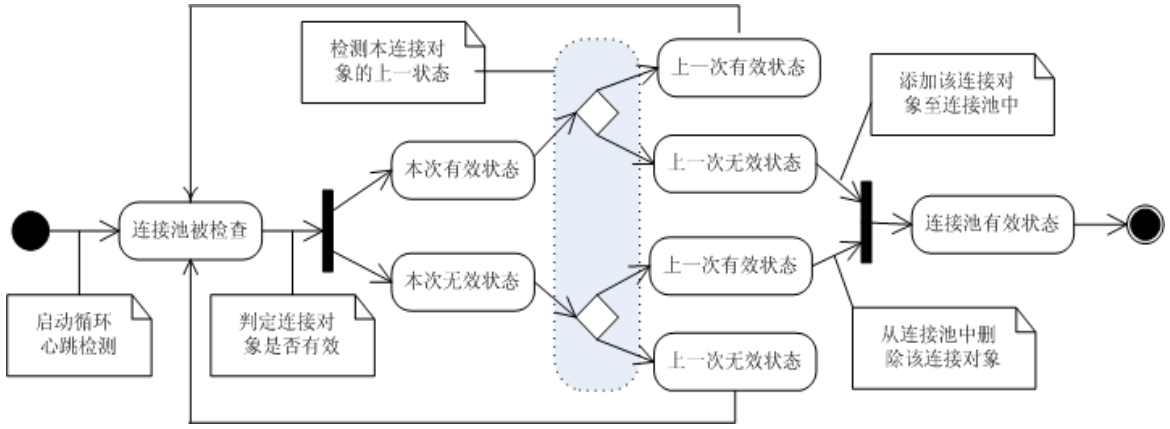

图 4-9 心跳检测示意状态图

Fig. 4-9 Heartbeat detection signal state diagram

(d) 动态添加数据源处理

对于动态挂载多数据库系统中的数据源功能，本文通过启动一个线程在后台监控数据库配置文件是否改变来实现，如果发生了改变即重新生成一个各数据库连接池的数组对象，并将该对象中包含的内容与原有数组对象进行比较，如果不一样则进行相应的修改。其解决途径是通过调用运行一个继承至 Thread 类的对象来完成，具体实现步骤如下：

首先通过采用 java.io.File.lastModified()方法获取到配置文件最近一次修改的时间，然后再与前一次保存的修改时间进行比较，如果发现不一样，则对配置中的各异构数据库服务器进行一次连接池的重新生成，产生的新连接池数组与原有连接池数组进行循环比对，如果有新的连接池产生，则将其加入到原有的连接池数组中。

以上循环检测的时间定为每隔 1 分钟检测一次，具体通过 Thread.sleep(60000)方法实现。

(4) 桥接管理子模块

桥接管理主要负责将客户端连接与服务端连接建立关联关系，以便客户端发出去的查询处理命令能够正确地分发到后端各异构数据库上，并通过这种关联关系准确无误的将参与查询处理的各数据库中返回的数据传递回客户端。其实现的类图如图 4-10 所示：

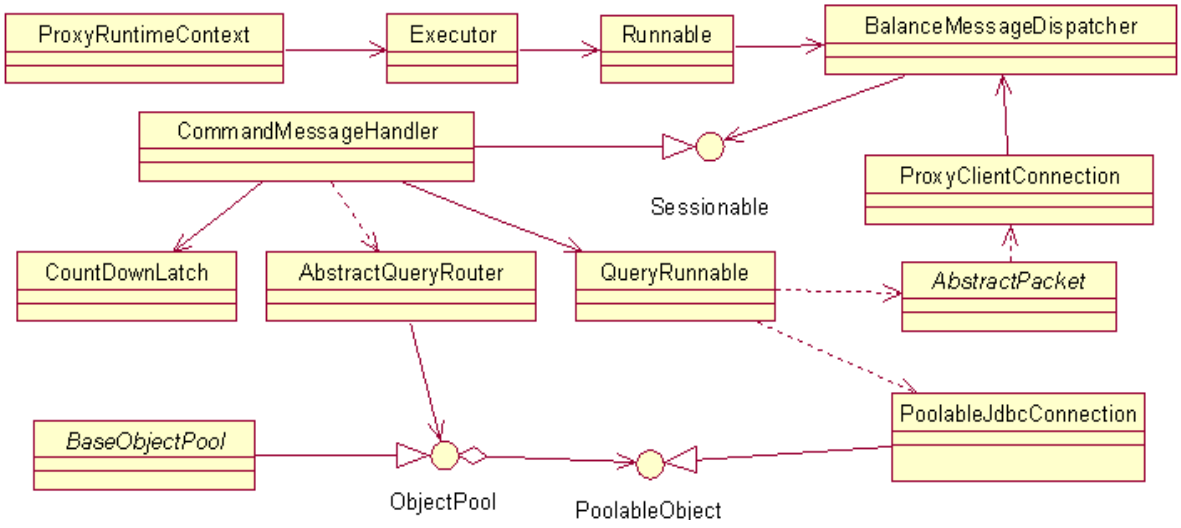

图 4-10 桥接管理类图

Fig. 4-10 Bridge management class diagram

桥接管理的具体实现步骤如下：

(a) 客户端发起连接至多数据库中间件，中间件处理连接事件；

(b) 中间件接收从客户端发送过来的消息；

(c) 消息接收完毕后，从线程池中取出一个线程用于消息的处理；

(d) 如果从客户端传递过来的消息是 SQL 语句或者是存储过程命令，则进行路由分析，以确定这些语句或者命令将与哪些后台数据库相关，并获取到对应的数据库连接池；

(e) 结合上面获取到的数据库连接池，在各相关数据库上进行数据查询处理，并将所获取到的数据传输给相关客户端；该方法的具体实现方式如下：

首先循环获取各连接池，以便从各数据库上取到连接，获取到服务端连接后，产生一个 Runnable 对象便于线程池调用，从线程池中获取线程并运行，这样客户端与各异构数据库便实现了并行连接交互，然后等待所有线程运行结束，最后返还后端数据库连接，从各数据库中获取到的数据通过客户端连接输出给用户；

4.3.2 模式处理模块设计与实现

根据前述章节的模式处理模块需求分析可知，该模块分为模式翻译以及模式集成两个子模块。本文通过引入一个全局配置数据库以及在其上建立相应的模式映射表、中间表来解决多数据库中间件中碰到的模式冲突问题，以下将从模式翻译以及模式集成两个角度进行具体的设计与实现分析。

(1) 模式翻译子模块

模式翻译主要是完成将后端各局部异构数据库中存储的数据模式转换成用统一模型表示的数据模式，以消除各字段含义以及字段结构表示的异义性和冲突性。针对模式处理模块需求分析章节中提出的各种模式冲突，本文通过建立相应规则转换函数以及全局中间表来解决，其设计与实现方式如下：

(a) 对于有冲突含义或冲突精度的字段通过建立规则转换函数来解决，如：在数据库 A 上表示的日期字段，在数据库 B 上分别由年、月、日三个字段构成。当出现这种情况时，通过在对应局部数据库 B 上建立规则转换函数，以满足字段转换的需要。用户查询该字段数据时，只需在字段映射表中找到对应的规则转换函数进行规则转换即可。

整个多数据库系统中的全局表字段与局部字段以及局部字段规则的映射关系存储于表 t_field_map 中，该表的字段结构如表 4-1 所示：

表 4-1 字段规则映射表

Table 4-1 Field rules mapping table

字段 名称	字段 类别	允许 空值	字段 描述
Gschema	char(15)	否	全局数据库模式，用于区分不同用户下的表
Gtablename	Char(20)	否	全局表名称
Gfieldname	Char(20)	否	全局字段名称
Lschema	Char(15)	否	局部数据库模式，用于区分不同用户下的表
Ltablename	Char(20)	否	局部表名称
Lfieldname	Char(20)	否	局部字段名称
lfieldRule	Char(100)	是	局部字段规则转换函数名称

(b) 对于那些在各局部异构数据库之间存有复杂冲突关系的表通过建立全局中间表或者中间表之上再建立中间表来解决，例如：在数据库 A 上的某张表 tableA，在数据库 B 上被定义成了两张表 tableB1，tableB2。当出现这种情况时，通过在全局配置库中配置一张全局中间表来解决，该表的结构与 tableA 一至，并属于会话级的临时表。用户发出查询指令时，根据相关映射表进行两阶段提交查询，第一阶段为数据准备阶段，即从各局部数据库上根据加载规则将数据载入全局中间表，第二阶段为数据整合查询处理阶段，即对获取到的数据进行相应的分组、排序或者汇总处理得到所需查询结果。其具体的查询设计与实现方法将在查询处理模块中进行详细阐述。

(2) 模式集成子模块

模式集成子模块主要负责表之间、查询处理命令之间以及查询处理命令与表之间关联关系的建立。

对于以上模式翻译转换所产生的转换规则函数、全局中间表需要进行相关表关联集成处理；对于各局部数据库上的表数据存储规则需在全局配置库中进行注册登记；对于两阶段提交查询的调用规则也需进行注册登记处理，其具体实现方式如下：

首先，由多数据库中间件管理员通过登陆规则配置后台对各种规则映射表进行维护，该配置后台由一个 web 服务应用构成，其主要的规则映射表有：字段规则映射表 t_field_map，数据存储规则映射表 t_table_rule，负责两阶段提交查询的存储过程与各局部数据库数据关系映射表 t_sp_table，父存储过程与子存储过程关系映射表 t_sp_subsp

以及子存储过程与各局部数据库数据关系映射表 t_subsp_table，这些表之间的相互关系如图 4-11 所示：

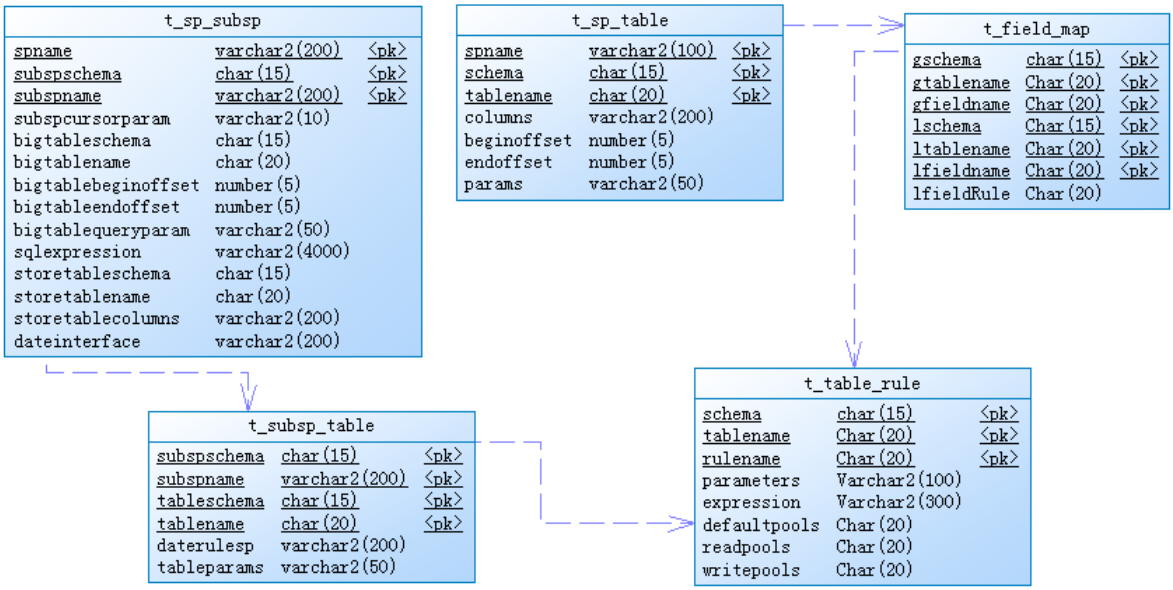

图 4-11 各表关系图

Fig. 4-11 The tables relationship diagram

其次，当多数据库中间件系统启动时，将这些映射表从全局配置数据库中加载入内存对象中保存，以缩短中间件在做查询路由处理时的时间消耗，从而提高系统查询效率；

最后，在多数据库中间件后台启动一个映射表监控线程，当其中某一个映射表发生改变时，即可将其重新加载入内存对象中，使得多数据库中间件管理员对映射表的修改可马上生效。

4.3.3 查询处理模块设计与实现

根据前述章节的需求分析可知，查询处理模块由查询分解、查询转换及查询优化等三个子模块构成，以下将分别对各子模块的设计与实现进行详细的阐述与分析。

(1) 查询分解子模块

查询分解子模块主要是解决全局查询语句如何分解成能够在各个数据库上执行的局部查询语句。由于本文所设计采用的全局查询语言是标准的 SQL 语句与存储过程，因此不存在查询语言之间转换的问题。因此，我们最关心的问题是集中在某条具体查询语句会与后端哪些数据库相关，也就是说该条语句将被发送到哪些数据库上去执行。其具体分解算法如图 4-12 所示：

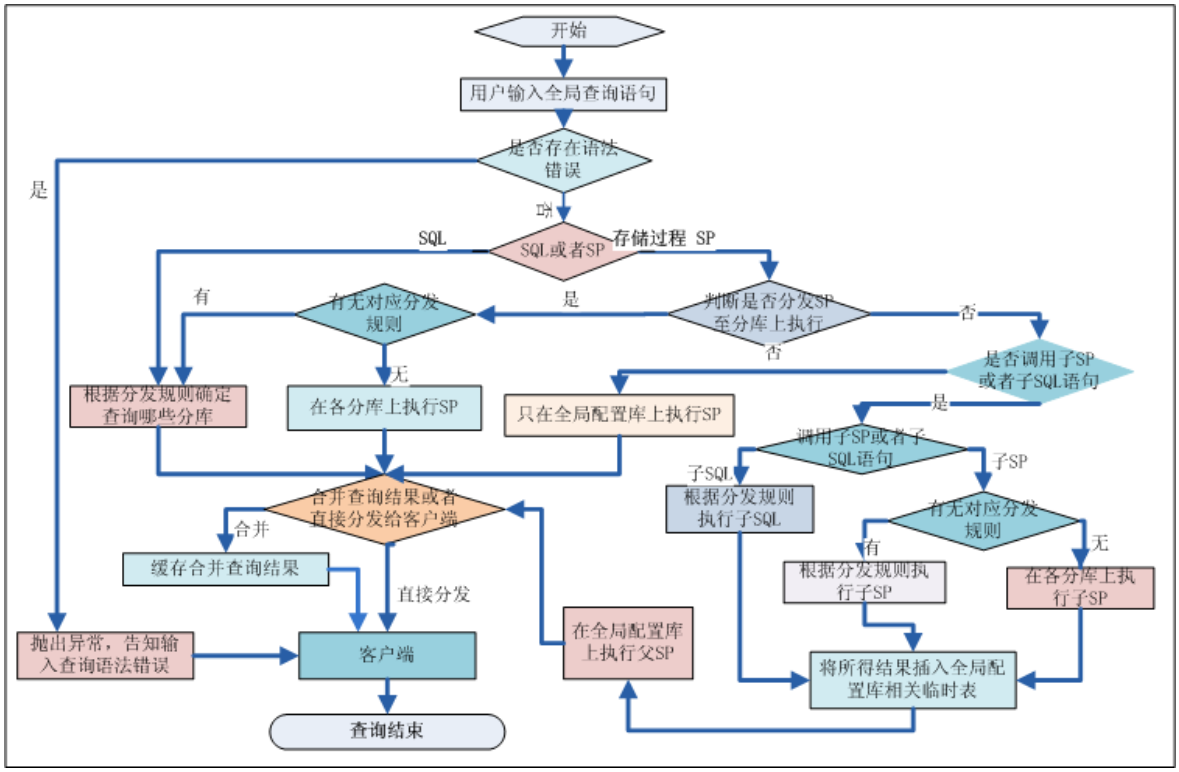

图 4-12 查询分解算法

Fig. 4-12 Query decomposition algorithm

从图 4-12 可知，全局查询语句可以是 SQL 语句或者存储过程，这些语句传递到多数据库中间件后，通过语法解释器对各语句进行语法分析，将分析后的结果与相应规则映射表进行匹配，并得出是否需要做二阶段查询处理，即子查询处理。该阶段查询处理的目的是为第一阶段的查询预备数据。第二阶段查询处理的原理与第一阶段一样，也是通过语法分析及模式匹配来得出该查询与哪些数据库相关联。

本文将采用 JavaCC 技术并结合连接管理模块中建立的多数据库连接池以及模式处理模块中建立的相关模式表来实现查询分解的功能。其具体实现步骤如下：

(a) SQL 语句解析

为了能够解释 SQL 语句，需要通过 JJTree 来完成分析树或抽象语法树的生成。jjTree 可以看成是 JavaCC 的预处理程序，它的输入文件的后缀名是 jjt，经它处理之后生成后缀名为 jj 的文件就包含了生成分析树的能力，然后通过 JavaCC 对后缀为 jj 的文件编译成 7 大 JAVA 类，分别为：

TokenManager：词法分析类；

Parser：语法分析类；

Constants：词法和语法分析常用接口类；

SimpleCharStream：把字符串传给词法分析类进行处理的适配器类；

Token ：标识符定义类；

ParseException：语法分析异常定义类，继承自 Exception 类；

TokenMgrError；词法分析异常定义类，继承自 Error 类。

根据以上生成的 JAVA 类来完成 SQL 的语法解释。本文所实现的一个 BalanceParser.jjt 文件总共分为 5 部分，具体文件内容说明参考附录 2。

通过对以上 BalanceParser.jjt 文件的实现，然后采用 JJTree 工具生成 BalanceParser.jj 文件，最后通过 JavaCC 编译生成 BalanceParser.java 类文件。根据生成的 BalanceParser 类来完成 SQL 语法解析，具体实现方式如下：

首先根据传入的 SQL 语句新建一个 BalanceParser 对象，然后调用对象中的 doParse() 方法对 SQL 语句进行解析。

SQL 解析流程示意如图 4-13 所示：

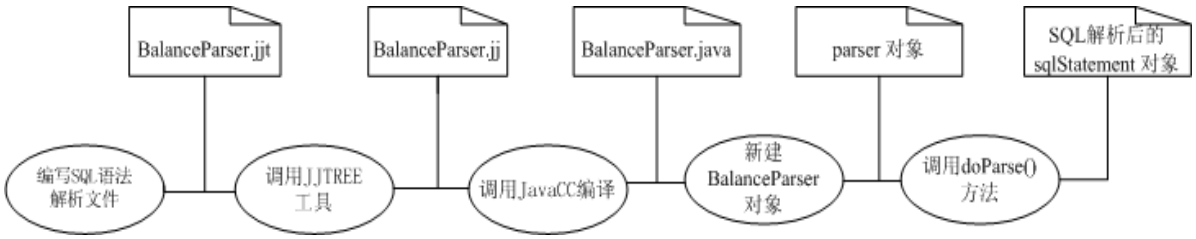

图 4-13 SQL 解析流程示意图

Fig. 4-13 SQL parsing flow diagram

(b) 解析后 SQL 语句与规则映射表比对分解

根据上述设计实现可知，从客户端传递给多数据库中间件的全局 SQL 查询语句内容经过计算转换存储于 sqlStatement 对象中，该对象包含的属性有表达式属性 Expression，用于存储 SQL 语句中 where 条件部分的内容；字段列属性 Map<String,Column> selectColumnMap，用于存储 SQL 语句中 select 部分的内容；表名与 where 条件中各列值的关系属性 Map<Table, Map<Column, Comparative>> evaluatedTableMap，该属性起到的作用是用于后续与数据存储规则映射表 t_table_rule 中的内容比较。该表的字段结构如表 4-2 所示：

表 4-2 表规则映射表

Table 4-2 Table rules mapping table

字段名称	字段类别	空值	字段描述
schema	char(15)	否	数据库模式，用于区分不同用户下的表
tablename	Char(20)	否	表名称
rulename	Char(20)	否	规则名称
parameters	Varchar2(100)	否	参数名称，多个参数以逗号分隔
expression	Varchar2(300)	否	参数的值，该值可以用各种函数表示
defaultpools	Char(20)	否	缺省连接池的名称，多个连接池以逗号分隔
readpools	Char(20)	是	只读连接池名称
writepools	Char(20)	是	可写连接池名称

当多数据库中间件初始化时该表的内容被加载进入 tableRuleMap 对象中，该对象为一个 HashMap<Table, TableRule>类型，其中 Table 类是一个包含表名称及 schema 模式的

类，TableRule 类是一个包含以上各字段内容的类。该类同时引进了一个 RowJep 类的对象变量，目的是为了将 parameters 与 expression 建立某种关联关系。

最后根据以上生成出来的 sqlStatement 对象与 tableRuleMap 对象进行相关比较计算得出客户端发送给多数据库中间件的全局查询语句最终将被发送到哪些数据库上去执行。部分实现代码参考附录 3。

其具体实现流程示意如图 4-14 所示：

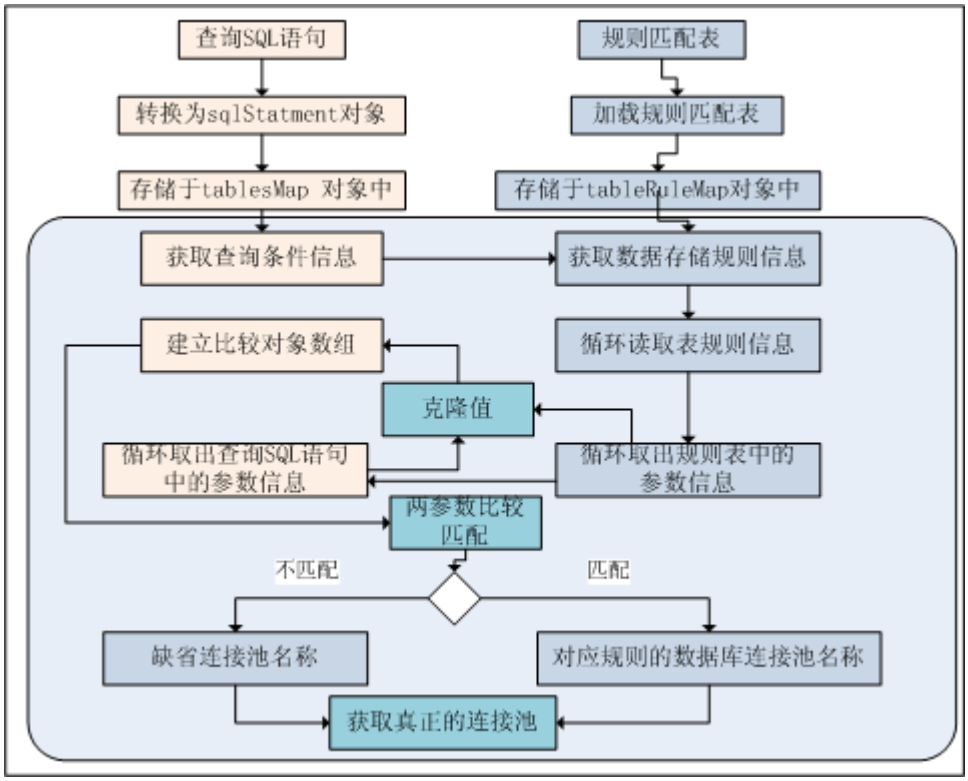

图 4-14 比对分解流程示意图

Fig. 4-14 Schematic diagram of the decomposition process

(c) 存储过程解析

对于从客户端传递过来的存储过程语句，我们将其分解为存储过程名字、输入参数位置、输入参数的值、输出参数的位置、输出参数的类型存储于 spStatement 对象中，以便后续处理，具体实现方式如下：

首先新建一个 SpStatement 对象，然后从存储过程语句中获取存储过程名字，最后将获取到的存储过程名字保存至 SpStatement 对象中。

(d) 解析后的存储过程与关系映射表比对分解

由于存储过程中使用到的数据表可能有很多张，而这些表又很有可能按照时间段或者流水号划分存储于不同的数据库上，如何正确判断某个存储过程发往哪些数据库上去执行，成为了一个需要解决的难题。本文通过采用模式处理模块中产生的三张存储过程映射表来实现存储过程与数据库关系的建立，这三张表分别为存储过程与表的关系映射表 t_sp_table，存储过程与子存储过程及子 SQL 关系映射表 t_sp_subsp 以及子存储过程

与表的关系映射表 t_subsp_table，其具体实现流程如下：

首先，当中间件接收到客户端传递过来的存储过程语句后，根据第一个参数值判断该存储过程是否需要分发至分库上去执行，如果第一个参数值等于 DISTRI，则说明需要将该存储过程进行分发，然后通过查找 t_sp_table 表中该存储过程所使用到的数据表，通过查找到的数据表以及存储过程参数并结合上述 SQL 解析的方法联合决定该存储过程将被分发到哪些数据库上去执行。如果没有找到相应数据表则将该存储过程分发到所有分库上去执行。

其次，当存储过程的第一个参数值不等于 DISTRI 时，则到 t_sp_subsp 表中去查找是否有该存储过程对应的子存储过程或者子 SQL 语句，如果有子存储过程则说明在执行此存储过程之前必须先调用子存储过程，如果有子 SQL 语句则必须先调用该子 SQL 语句，以上目的是为存储过程预加载数据。如果没有找到对应子存储过程以及子 SQL 语句则该存储过程只到查询库上执行。

最后，子存储过程根据 t_subsp_table 表中存储的该子存储过程与数据表的对应关系以及父存储过程的参数来决定具体分发到哪个库上去执行，如果没有找到与该子存储过程相关的数据表则该子存储过程将被分发至所有分库上去执行。子 SQL 语句根据 SQL 解析的方法来确定该 SQL 将被分发至哪些数据库上去执行。

(2) 查询转换子模块

查询转换子模块主要负责如何将客户端传递过来的查询命令进行查询转换，该查询转换有两步，第一步是将传递过来的查询命令转换为查询处理语句；第二步是将查询处理语句与模式匹配表进行比对做查询分解转换。

(a) 查询命令转换为查询处理语句

本文是通过采用 byte 流的方式实现客户端向多数据库中间件发送查询命令，然后中间件对获取到的 byte 流进行解析并将其转换为查询处理语句，具体实现步骤如下：

首先，在客户端通过自定义 JDBC 驱动将 SQL 语句或者存储过程转换为一个个的 packet 数据报文。每个包的内容都必须遵守本文所设计的通信协议。该通信协议规定每个 packet 由包头和包体两部分组成，每个 packet 的初始化大小为 1024 字节，最大不超过 65536 个字节，其中包头由 4 个 byte 位构成，前 3 个 byte 位表示包的长度，第 4 个 byte 位表示该包的序列号；包体由 1 个 byte 位表示查询命令的类型，其余部分表示查询命令的内容。

然后，客户端通过以下方法往多数据库中间件发送 byte 数组类型的 packet 包：

通过构造一个 socket 缓冲输出流，将 packet 写入该缓冲输出流，然后刷新缓冲输出流，并将 packet 通过 socket 通道发送至中间件，最后多数据库中间件从非阻塞式的通道中读取从客户端传递过来的数据并存入输入流 ByteBuffer 缓冲对象中，数据读取完后再将缓冲对象中的数据读入 byte 数组中。然后对其进行解析处理转换为查询语句。

(b) 查询分解转换

对于从客户端传递过来的 SQL 语句有些需要进行再一次的转换处理，如：全局表 GlobalTableA 的某一个成交量字段值 volume 是分别由某个局部数据库 db1 中的表 LocalTableA 的投机成交量字段 specVolume 和另一个局部数据库 db2 中的表 LocalTableB 的套期保值字段 hedgevolume 求和后构成的。因此，对于诸如 select volume from GlobalTrbleA 这样的查询 SQL 语句就必须再进行一次查询分解转换处理，并将转换处理后的 SQL 语句根据前一节所说的查询分解算法发送到与之相关的局部数据库上去执行。转换后的 SQL 语句如下：

发往局部数据库 db1 的子 SQL 语句：select specVolume from LocalTableA；

发往局部数据库 db2 的子 SQL 语句：select hedgeVolume from LocalTableB；

当发往两个局部数据库上的 SQL 语句执行完毕后，将其返回给多数据库中间件的结果进行求和处理即得到了所需结果。

另外，对于那些涉及到跨多个数据库的数据排序、分组汇总或者表连接的查询，也需要进行再次查询转换，本文通过采用两阶段提交查询来实现，即第一阶段为数据准备阶段，将查询涉及到的表数据先从各分库加载入查询库对应的会话级的临时表中，第二阶段为数据查询阶段，在查询库中对这些会话级的临时表数据进行排序或者分组汇总查询等。具体实现步骤如下：

第 1 步：在查询库中建立相关会话级的临时表，用于存放从各局部数据库查询返回的数据；

第 2 步：将涉及到跨多个库的排序、分组汇总等查询语句封装进存储过程；

第 3 步：建立多个子存储过程，目的在于向多个分库中查询使用到的表数据并装载进步骤 1 所建的临时表中；

第 4 步：调用执行步骤 2 所建的存储过程，得出所需结果。

以上存储过程与子存储过程的关系，以及子存储过程与各表的关系分别存储于关系映射表 t_sp_subsp 以及表 t_subsp_table 中。

(3) 查询优化子模块

查询优化子模块主要负责查询性能的优化，其可通过以下四个方面来实现解决：

(a) 使用 SQL 语句缓存

对于解析过的 SQL 语句我们将它保存至一个 LRUMap 类的对象中，该类最大的一个特点是它能够保留最近经常使用的对象，而删除最不会经常使用的对象。同时它可初始化保存对象的数目。通过 SQL 语句的保存可以避免系统重复的对 SQL 进行解析，从而提高了多数据库中间件的查询效率。具体实现步骤如下：

首先初始化 LRUMap 类对象，默认可保存 1000 个对象，然后设置对象关键字，最后将解析后的 sql 内容加入 map 对象中，当下次需要使用时只要从中查找即可。

(b) 采用查询并行处理

对于那些需要分解成在多个分库上执行的查询语句，可通过启用多线程来调度分发

执行这些分解后的子查询语句，从而达到 SQL 语句并行查询的目的，既而缩短了系统查询时间。

(c) 提高数据吞吐量

对于多数据库中间件从后台各分库读取数据以及发送数据到各客户端，都牵涉到数据从 socket 通道读取和写入的问题，必须尽量设置好网络缓冲区的大小，确保其与应用程序缓冲区的大小一致，以提高网络的数据吞吐量。具体的设置内容有：设置每次在执行通道的写数据时期望发送的字节数；设置每次在执行通道读数据时期望接收的字节数；设置 TCP 数据包延迟数。

(d) 减少网络传输开销

由于有些查询需要将数据从各分库加载到查询库中以便进行分组汇总或者其它业务逻辑处理，这个时候就必须考虑是否在加载前有必要先在各分库上进行一次数据的粗略汇总，以提高数据的颗粒度，从而减少数据在网络上的传输开销。最终缩短数据在网络上传输的时间，进而提高系统的查询效率。

4.3.4 数据处理模块设计与实现

从前述数据处理模块需求分析可知，数据处理模块主要由数据预加载、数据分发管理、数据转换以及数据收发等四个子模块构成，以下将分别对这些子模块的设计与实现进行详细的阐述与分析。

(1) 数据预加载子模块

数据预加载的主要目的是为相关查询提供基础数据。辟如某个查询语句使用到的数据库表有的在数据库 db1 上，有的则在 db2 上，而这些表在查询语句中又做了全连接。如果不做任何处理，这个查询语句是无法正常执行的。

在本文中通过采用数据预加载的方式来处理以上的难题，既将查询语句中涉及到的表数据从分库中加载入查询库的会话级的临时表中，然后再在查询库中执行 SQL 查询处理。以上实现方式利用了会话级的临时表特点：空间独立性、数据会话性、并发性。所谓空间独立性是指数据库为每一个发起会话的用户分配了独立的数据存储空间，虽然使用的表名称一样，但是一个用户的数据存储操作不会对另一个用户产生影响；数据会话性是指数据存储在表中的有效时间只与会话相关，当会话结束时存储的数据会自动被删除；并发性是指多个用户同时对同一临时表进行操作处理时互不干扰。

数据预加载的实现步骤如下：

首先，从查询库连接池中获取到一个连接，然后通过该连接将从各分库上获取的数据加载入查询库对应的会话级的临时表中，最后再通过该连接使用先前的查询语句对临时表中的数据进行查询得到所需结果。

(2) 数据分发管理子模块

数据分发管理子模块主要负责解决数据传递模式，即当后端数据库的数据通过网络

传输到达多数据库中间件时，中间件是如何处理分发这些数据的。分发方式有两种：一种是当各分库上查询所得数据都加载到中间件后，才启动数据传输引擎，将这些数据传输至客户端；另一种方式是只要有一部分查询数据加载入中间件，则马上将数据转发传递至客户端。系统到底采用哪种数据分发方式，由客户端来决定。具体实现步骤如下：

(a) 客户端向中间件发送从服务器获取记录条数的命令以及取数命令；

(b) 中间件以该命令设置的记录条数，每次轮流从后端各参与的局部数据库中抓取数据；

(c) 当每次抓取的数据到达指定的记录条数后，便立即转发至客户端；

(d) 客户端接收从中间件发过来的数据，每次数据接收完毕后，重复步骤 a 的操作；

(e) 一直到所有参与查询的数据库上数据被抓取完为止；

为了完成以上操作，首先必须保证在同一时刻多数据库中间件系统只能与一个局部数据库进行数据的传输；其次必须保证能够完整正确的获取到所有分库上传递过来的数据；最后必须协调好客户端与中间件以及中间件与数据库之间的数据传递步调。

其实现流程示意图如图 4-15 所示

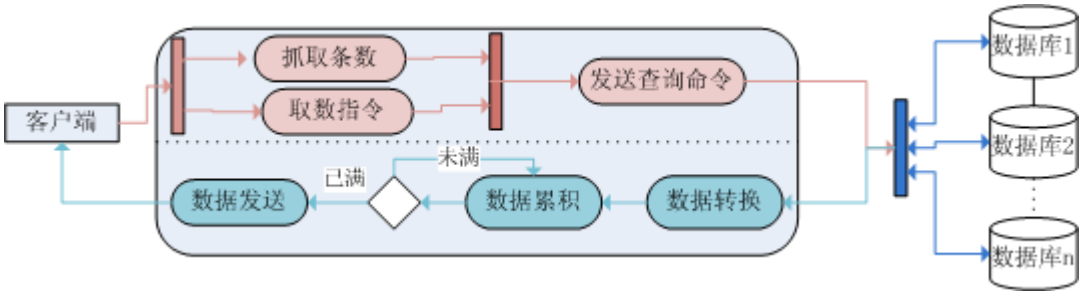

图 4-15 数据分发管理流程示意图

Fig. 4-15 Data distribution management flow diagram

(3) 数据转换子模块

数据转换子模块主要完成数据的转换工作，即将从数据库中获取到的数据转换为可在网络上传输的 packet 包。在本文中用于数据转换的 packet 包被设计为如表 4-3 所示的 6 种类型：

表 4-3 数据转换包类型

Table 4-3 Data conversion package type

包名称	包属性	包用途
RowDataPacket	packetId—序列号 Columns—每条记录的各字段值集合	转载每条记录内容的数据包

	isBinaryEncoded—二进制编码标志	
FieldPacket	packetId—序列号 Table—字段对应的表名称 Name—字段名称 Length—字段长度 Character—字段类别	记录每个字段说明信息的数据包
EOFPacket	packetId—序列号 serverStatus—服务器状态标志 warningCount—警告记录数	数据传输结束标志的数据包
OkPacket	packetId—序列号 affectedRows— 有效记录数 serverStatus—服务器状态 message—返回信息	标记所有记录已正确传输的数据包
ErrorPacket	packetId—序列号 errno—错误代码流水号 sqlstate—sql 错误码 serverErrorMessage—错误信息	标记记录传输失败的数据包
ResultSetPacket	fieldPackets—表字段数据包集合 rowList—表示记录数据包集合	表字段说明与记录集合的数据包

具体实现步骤如下：

(a) 查询返回记录内的每个字段的列名、表名、字段长度、字段类别等说明信息；

(b) 将这些信息保存至 FieldPacket 包中；

(c) 获取到每条记录的列字段值，并将这些值保存至 RowDataPacket 包中；

(d) 将产生的这些 FieldPacket 包以及 RowDataPacket 包打包进 ResultSetPacket 包中；

(e) 产生一个数据传输结束标志的 EOFPacket 包；

(f) 最后将产生的这些 ResultSetPacket 包以及 EOFPacket 包通过 socket 通道传递到客户端，如果传输正确，则发送一个 OkPacket 确认包，如果失败，则发送一个 ErrorPacket 确认包，客户端通过解析这些包获得所需数据，并可判断出是否传输出错，以及错误的原因是什么。

(4) 数据收发子模块

数据收发子模块主要是实现客户端与多数据库中间件系统之间的数据收发功能。本文通过采用非阻塞式的 socket 通道来完成数据的收发。其具体实现流程如下：

(a) 上述小节所产生的 ResultSetPacket 包以及 EOFPacket 包首先被放入一个 ByteBuffer 比特流缓冲类型的 Queue 队列中；

(b) 每一个非阻塞式的 SocketChannel 通道都被注册到了一个选择器 selector 上，同

时创建了一个 SelectionKey 选择键。通过对该键值的判断，可知通道内数据操作的准备状态是可读还是可写的。

(c) 如果选择键是可写状态，则通过调用队列的非阻塞式方法取出一条缓冲记录，接着调用 SocketChannel.write(buffer)方法将缓冲记录发送到客户端；

(d) 由于网络带宽的问题需要判断该缓冲记录是否已经全部被发送出去，如果发现结果大于零，则需要将剩余的缓冲数据再重新放进队列中进行下一次的传输处理，以防止数据被丢失。

(e) 如果选择键是可读状态，说明从客户端传递了数据过来，则多数据库中间件通过调用 SocketChannel.read(byteBuffer)方法，将数据读入 byteBuffer 比特流缓冲对象内。

其具体实现的时序图如图 4-16 所示：

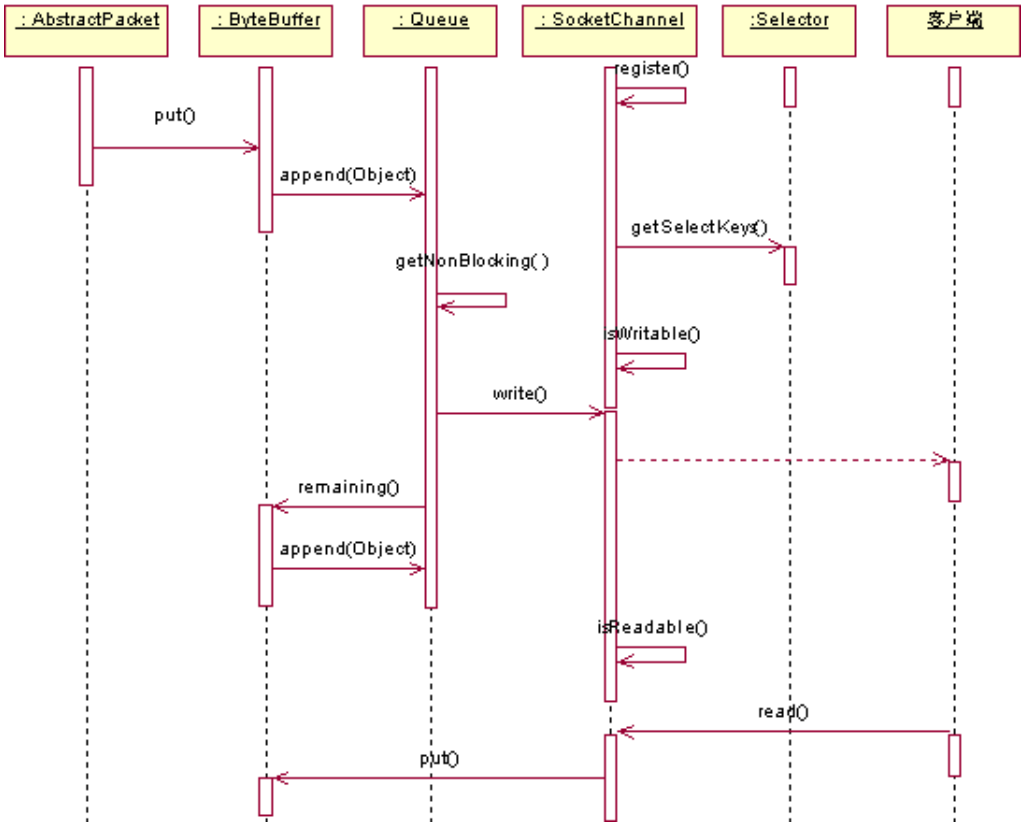

图 4-16 数据收发时序图

Fig. 4-16 Data send and receive sequence diagram

4.3.5 事务处理模块设计与实现

根据前述章节的事务处理模块需求分析可知，该模块由事务注册、票据管理、事务分解、事务调度、事务恢复、事务日志等六个子模块组成。以下将就这六个子模块的具体设计与实现进行详细分析。

(1) 事务注册子模块

事务注册子模块主要负责全局事务管理器的生成和初始化工作，以便全局事务向该

管理器进行注册，完成分布式事务处理工作。其具体设计与实现方式如下：

当多数据库中间件系统启动时，通过调用 TransactionServer 类的 init()方法对全局事务管理器进行初始化，其主要完成以下几方面的工作：

(a) 构建一个全局事务管理器的工厂，用于产生全局事务管理对象；

(b) 通过调用工厂的创建事务管理对象方法，初始化一个缺省的全局事务管理对象，用于准备执行客户端提交过来的事务命令。该事务对象实现 JTA 规范中 UserTransaction 以及 TransactionManager 的两个接口；

(c) 创建全局事务管理对象时，同时创建以下变量：

threadTran：用于保存事务对象相关状态和内容，它是 ThreadLocal 类的一个实例；

tranXids：全局事务 ID 号与事务对象之间关系的集合，属于 HashMap 类对象；

timerManager：线程超时管理对象，它是 Tread 类的一个实例；

rollbackNum:主要用于记录保存全局事务提交时有多少子事务提交失败的数量；

commitNum:记录在一个全局事务中各全局子事务提交成功的数量；

(d) 将产生的全局事务管理器工厂注册到系统的上下文中，以便每次用户提交事务命令的时候可以通过系统的上下文环境找到该工厂，并通过该工厂创造一个全局事务；

(2) 票据管理子模块

票据管理子模块主要是对每个事务命令执行时产生一个事务对象，并对该事务对象建立一个票据 ID 号 XID，并将该 ID 号与事务对象的对应关系保存至上述介绍的 tranXids 变量中，同时设置事务对象的超时时间。主要目的是用于标记和区分每个全局事务，为每个事务命令与数据库进行正确的交互提供保障；

当多数据库中间件调用全局事务对象的 begin()方法开始提交事务命令时触发该票据管理功能，具体实现方式如下：

(a) 创建一个具体事务对象，该对象是一个实现了 javax.transaction.Transaction 接口的 TransactionImpl 类实例。主要用于全局事务的提交、回滚、连接资源的管理、提交状态的获取等，其方法有以下几个：事务准备提交的方法 prepare()、实现事务提交的方法 commit()、建立连接资源列表的方法 enlistResourcedelistResource()、删除连接资源列表的方法 delistResource()以及连接资源清理方法 doAttach()等；

(b) 从上述本地线程变量 threadTran 中查找是否已存在事务对象，如果存在则与 tranXids 变量中保存的事务对象进行比较，看是否相同？如果相同则说明该事务是一个嵌套事务，即一个大事务中嵌套了小事务，在本文中未实现该功能，被看成是异常并退出；如果不相同则说明线程中有一个异常的事务未处理，需要进行事务回退操作；

(c) 通过实现 JTA 规范中的 javax.transaction.xa.Xid 接口来实现事务票据号 XID 的产生，XID 包含三个部分：格式 ID 号、全局事务 ID 号以及分支事务 ID 号。对应的有三个方法，分别为：获取格式 ID 的方法 getFormatId()、获取全局事务 ID 号的方法 getGlobalTransactionId()、获取分支事务 ID 号的方法 getBranchQualifier()。同时规对定

全局事务 ID 号与分支事务 ID 号最大长度都不能超过 64byte。在本文中格式 ID 号被设置为一个常量，全局事务 ID 号由转换为十六进制的系统时间戳，IP 地址、服务器主机名组合而成，分支事务 ID 号由各提交阶段的版本号(RU1，RU2)、子事务号、IP 地址及服务器主机名组成；

(d) 调用事务对象的 doAttach()方法，对已经放入垃圾回收站的的连接资源进行判断是否可以再次重新启用，以提高系统事务处理效率；

(e) 将票据 ID 号与生成的事务对象关系保存到关系集合对象 tranXids 中；

(f) 为事务对象设置时间管理器以及设置时间戳。

(3) 事务分解子模块

事务分解子模块主要负责全局事务的分解，即解决一个全局事务最终会分解成为几个全局子事务，这些分解后的子事务与哪些后端数据库相关。其设计与实现方式如下：

(a) 当客户端提交一个事务处理开始标志传递至多数据库中间件时，多数据库中间件通过执行工厂的 getTranManage()方法获取到一个全局事务管理对象；

(b) 接着客户端将事务处理语句传递至多数据库中间件，多数据库中间件系统根据查询处理模块设计与实现章节中所讲述的方法，通过对这些语句的分析确定出有哪些局部数据库将要参与此次事务的处理，并获取到这些数据库的连接池，同时建立连接池与事务处理语句的对应关系；

(c) 对每个连接池赋予一个步骤 a 所获取到的事务管理对象；

(d) 多数据库中间件通过调用事务管理对象中 begin()方法开始执行事务处理，完成事务对象、票据的生成以及事务对象的相关属性配置，该方法的具体实现在上述票据管理子模块中已经详细阐述，在此不再赘述；

(e) 分别从连接池与事务处理语句对应关系对象中获取连接及事务处理语句发往各数据库执行；

(f) 在执行 conn.prepareStatement()方法时首先从全局事务管理对象中获取事务对象，获取到事务对象后设置该对象事务处理方式为不自动提交；然后通过调用该连接的 getXAResource()方法获取到对应资源对象，并将该资源对象保存至事务对象的 enlistedXARes 列表变量中；最后通过调用资源对象的 start()方法发送票据号至后端数据库，探测数据库是否已经准备就绪，如果准备就绪则发送处理语句至后端数据库执行，否则异常退出；

(4) 事务调度子模块

事务调度子模块主要完成事务的调度提交，该调度子模块通过采用两阶段提交协议来完成事务的调度提交处理，其具体实现方式如下：

(a) 当所有事务处理语句在对应数据库上正确执行完毕，多数据库中间件通过全局事务管理对象的 commit()方法触发调用事务对象中的 commit()方法，以提交各具体事务。同时，我们将各具体事务的状态分为以下 10 类，详见表格 4-4 所示：

表 4-4 事务状态表

Table 4-4 Transaction status table

状态名称	状态代码	状态说明
ACTIVE	0	激活态：事务已经开始，提交阶段还没开始
MARKED_ROLLBACK	1	回退态：标志事务回退命令已经提交
PREPARED	2	已准备：事务准备已经完成，必须马上提交
COMMITTED	3	已提交：事务提交成功完成
ROLLBACK	4	已回滚：事务已回退恢复
UNKNOWN	5	异常态：事务提交发生异常
NO_TRANSACTION	6	无状态：不能从事务管理器中得到状态
PREPARING	7	正准备：事务正准备将命令提交至数据库
COMMITTING	8	正提交：事务正在发往数据库
ROLLING_BACK	9	正回滚：事务正在回退恢复

(b) 接着通过调用 one_phase_commit ()方法进入第一阶段提交，该阶段主要完成全局子事务 ID 号的生成并向参与的各数据库发送准备提交命令，并获取各数据库返回的结果状态，该结果状态分为允许提交(0)、回退(1) 以及只读(2)三个状态，根据这些返回状态值来判断是否需要进行第二阶段的事务提交或者回退事务，具体实现方式如下：

首先根据票据 ID 号生成全局子事务号，将全局子事务 ID 号保存于事务对象的列表变量中，然后从资源列表对象中获取资源，将全局子事务号发送至各数据库，准备提交命令，最后根据各数据库返回的状态值决定是否进行第二阶段提交，如果返回的状态全部为正确状态，则参与的各数据库进行第二阶段提交，否则全部回滚；

(c) 当第二阶段事务提交方法被调用时，其主要完成的工作是通过每一个资源对象执行 commit(subGlobalXid, false)方法向数据库提交第二阶段事务，如果第二阶段事务提交成功，则并将该事务状态改为已提交，同时将 commitNum 变量加 1；

(d) 当 doRollback()方法被调用时，其主要完成的工作是对第一阶段提交的所有事务进行回滚，并将该事务状态改为已回滚状态，同时将 rollbackNum 变量值加 1；

(5) 事务恢复子模块

为了防止事务在进行第二阶段提交时，多数据库中间件或者后端数据库发生不可预料的系统宕机事件，需要有相应的事务恢复处理机制。本文通过采用在事务第二阶段提交前，对事务相关信息写进事务日志，并持久化至磁盘中保存，来实现事务的恢复管理。其具体实现方式如下：

(a) 当全局事务管理对象初始化时，同时初始化事务恢复对象，该对象类继承至一个线程类 Thread，并在后台进行不断的监控。

(b) 当事务进行第二阶段提交时，将提交的有关信息按一定格式保存至磁盘中，具体日志格式将在事务日志子模块中进行详细描述；

(c) 当事务恢复对象监控到有异常的事务提交，则将该异常事务对应的资源管理器名称与该资源管理器提供的 XAResource 对象进行关联，并将这种对应关系保存至一个集合的 Map 对象中；

(d) 通过 XAResource 对象的 recover 方法从后端数据库中获取到所有需要恢复的全局事务 ID 号 Xid，当然该数据库必须要支持可恢复事务的功能。接着到 log 日志中寻找看是否存在有这些 Xid，如果存在则调用 XAResource 对象的 commit(Xid)方法提交这些事务，最终完成事务的恢复；

(e) 事务恢复后将 Map 对象中已经完全恢复过事务的资源管理器对应关系删除，以防止事务重复恢复。

(6) 事务日志子模块

事务日志子模块主要是对事务的第二阶段提交内容进行持久化的记录保存，为将可能发生的系统崩溃提供可恢复的事务日志记录。本文将事务日志分为两类：一类为事务恢复记录，另一类为资源恢复记录，其中事务恢复记录的格式设计如表 4-5 所示：

表 4-5 事务恢复记录格式

Table 4-5 Transaction Recovery record format

字段名称	字段描述	字段类型	字段长度
RR1	事务恢复记录标志	byte	3
resDateTime	恢复记录存储时间	long	8
formatID	格式 ID	int	4
gtridLength	全局事务 ID 号的长度	int	4
Gtrid	全局事务 ID 号	Byte[]	<=64
bqualLength	分支事务 ID 号的长度	int	4
Bqual	分支事务 ID 号	Byte[]	<=64
traDateTimeLength	事务恢复记录的存储时间长度	int	4
traCreateDateTime	事务创建的时间	Byte[]	8
resCount	完成该全局事务对应的资源数量	int	4

资源恢复记录的格式如表 4-6 所示：

表 4-6 资源恢复记录格式

Table 4-6 Resource Recovery record format

字段名称	字段描述	字段类型	字段长度
RR2	资源恢复记录标志	byte	3
resIndex	资源对应索引号	int	4
XaResourceLength	资源长度	int	4
XaResource	资源对象	Byte[]	40
ResClassNameLength	资源对象的类名称长度	int	4

ResClassName	资源对象的类名称	Byte[]	40
formatID	XID 的格式 ID	int	4
gtridLength	与该资源对应的全局 ID 号长度	int	4
Gtrid	与该资源对应的全局 ID 号	Byte[]	<=64
bqualLength	与该资源对应的分支事务 ID 长度	int	4
Bqual	与该资源对应的分支事务 ID 号	Byte[]	<=64
XidStatus	XID 的状态	int	4

以上两类记录最终被转换为二维 byte 数组，并通过调用输入输出缓冲流的方法输出到磁盘文件中。

以上整个事务处理的相关类图如图 4-17 所示：

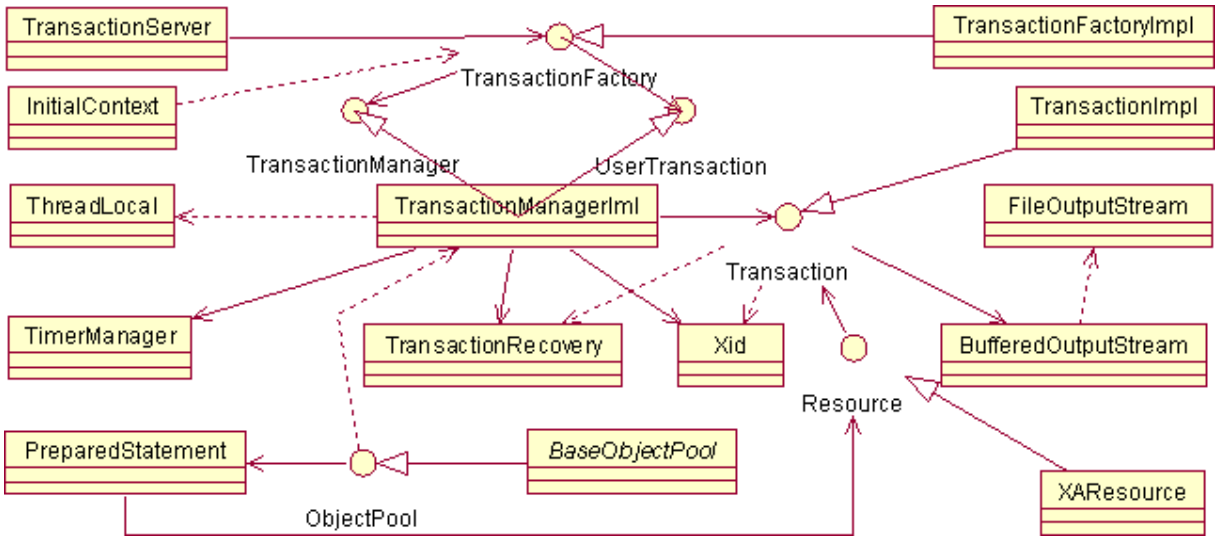

图 4-17 事务处理类图

Fig. 4-17 Transaction class diagram

4.3.6 安全控制模块设计与实现

根据前述章节的安全控制模块需求分析可知，该模块分为身份认证、访问控制、安全加密以及安全审计等四个子模块，以下将就这四个子模块的具体设计与实现进行详细分析。

(1) 身份认证子模块

在多数据库中间件中用户被分为全局用户及局部用户，其中全局用户指与中间件建立连接的用户，其用户名及密码等信息保存在一个 xml 文件中；局部用户指中间件与后端数据库建立连接的用户，该用户必须已存于局部数据库中，同时这些用户的用户名及密码等信息保存在另一个 xml 文件中。由于局部用户身份的认证已由各局部数据库进行了保证，因此在这里我们只对全局用户的身份认证进行具体实现，其具体实现流程如下：

(a) 多数据库中间件系统启动时，将这些用户信息从以上所说的两个 xml 文件中加载入 ProxyServerConfig 类的对象中，该对象包含的属性有该中间件的 IP 地址、端口号、

全局用户名、密码、线程数、网络缓冲字节大小、编码方式以及各局部数据库的配置信息，如局部数据库的 IP、所使用的驱动类别、URL 连接、局部用户名及密码等；

(b) 客户端向多数据库中间件发起一个连接请求，其具体的连接实现方式请参考连接管理模块设计与实现小节中的内容，在此就不再赘述；

(c) 多数据库中间件接收到连接请求后，向客户端发送通信协议版本号、用于密码加密的随机字符串、中间件服务器状态、中间件的版本号、连接线程序列号等信息；

(d) 客户端根据收到的加密随机字符串对需要发往中间件进行验证的密码加密；

(e) 客户端发送需要连接的用户名及加密过后的密码给中间件，中间件通过连接中存储的用户名和用相同随机字符串加密过后的密码与传递过来的用户名和密码进行比较判断是否一至，如果一致则认为此用户是一个合法的用户，接着可进行下一步的事务处理操作，否则为一个非法用户，强制关闭与客户端的连接。

以上实现流程示意如图 4-18 所示：

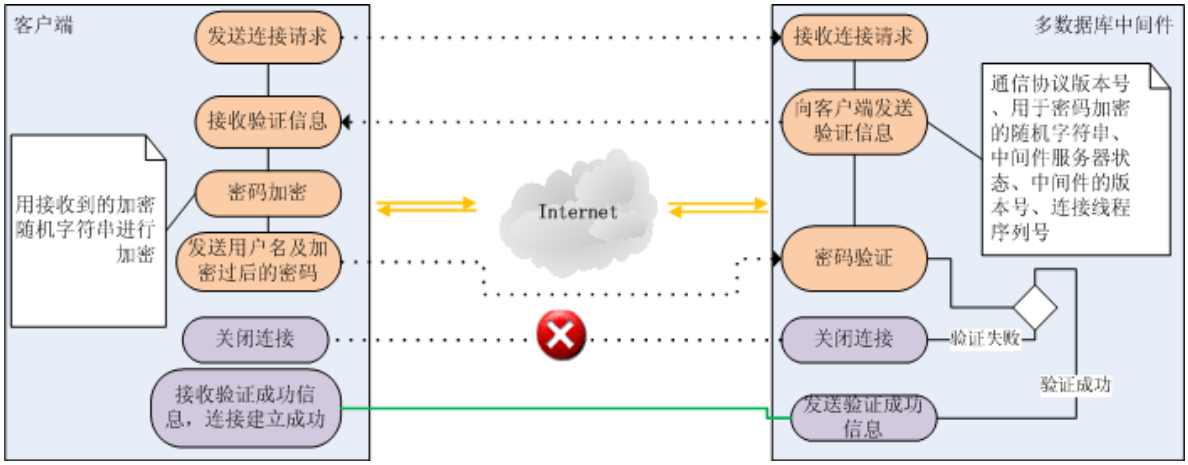

图 4-18 身份认证流程示意图

Fig. 4-18 Authentication process diagram

(2) 访问控制子模块

该子模块主要是负责限制 IP 地址的访问，以及后端数据库的可读写访问，具体实现方式如下：

(a) 限制 IP 地址访问

对于限制客户端 IP 地址的访问，中间件通过将配置属性文件 limitedIp.properties 中的 IP 限制规则与客户端的实际 IP 地址进行匹配比较，看其符合哪一条规则，以此来判断是否某个 IP 或者某段 IP 地址有访问的权限。该配置文件规则的优先级别从上往下逐渐降低，即排在前面的规则优先生效；以“#”号表示该条规则被注释掉；如果某 IP 地址在所有规则中匹配不上，默认为不允许访问该中间件。其文件结构如下：

#以下规则表示该 IP 地址段的客户端允许访问多数据库中间件

192.168.*.1-220:yes；

#以下规则表示该 IP 地址不允许访问多数据库中间件

202.96.134.133:no；

IP 访问权限判断的时机为客户端发送握手连接时，密码验证前。当客户端向多数据库中间件发送握手数据包时，中间件通过调用 SocketChannel 通道类的方法获取客户端的 IP 地址，并与上述文件中的规则进行匹配，以此决定是否允许该客户端建立连接进行访问；

(b) 数据库可读写访问

中间件在对后端异构数据库进行读写操作时，可以通过 t_table_rule 这张规则配制表以及配置文件中的 writePool、readPool 属性字段来决定其访问的权限。比如，当某条 SQL 语句符合 t_table_rule 中设定的规则时，则根据 SQL 判断其是查询语句还是事务处理语句。如果是查询语句则从对应匹配记录的 readPool 字段值中获取到连接池名称，反之则从 writePool 字段值中获取，如果在 t_table_rule 表中没有该 SQL 语句的匹配规则，则到配置文件的 writePool、readPool 属性字段中获取对应的连接池名称。最后通过所获得的连接池名称，从系统连接池库中获取到该名称对应的连接池，并从该池中获取到数据库连接，最终通过该连接在数据库上完成 SQL 操作。

(3) 安全加密子模块

安全加密主要是防止重要信息被窃取和篡改而对数据所采取的加密行为。在本文中通过采用非对称的 SHA-1 安全散列加密算法来进行密码的加密与验证。具体实现通过采用 JAVA 提供的 java.security.MessageDigest 类相关方法来处理，部分实现代码参考附录 4：

(4) 安全审计子模块

安全审计子模块主要负责对多数据库中间件的用户登录、运行状况、操作等进行日志记录，为以后的安全审计分析提供保障。本文采用 Apache 基金组织提供的一个开源的日志功能组件 Log4j 来完成系统日志记录。具体实现方式如下：

首先，配置好一个 log4j.xml 文件，并把该文件的路径设置到系统启动脚本的路径中，当多数据库中间件系统启动时，Log4j 便可自动去初始化该配置文件；

其次，为了能够监控到用户的登陆情况，所有与连接有关的 JAVA 类设计成为 Audit 类的一个子类，该类主要负责记录连接的使用情况：如连接的开始时间、结束时间、使用者 IP，用户名等信息，并初始化一个日志控制器，用于记录这些日志内容；

再次，对于中间件的运行状况的监控，通过在后台启动一个监控线程，该线程通过调用相关 JAVA 代码实现对中间件的内存使用，线程使用等情况的实时监控，主要包括有 JVM 虚拟机总内存、已使用内容以及空闲内存，当前线程所运行 CPU 的时间和 JVM 虚拟机内整个正在运行的线程数；

最后，对于用户有关的命令操作记录，通过创建日志控制器来实现完成；

以上所有日志记录信息最终按日期时间段来进行划分，分别存储于不同的文件中，实现的方式是通过 Log4j 调用 org.apache.log4j.DailyRollingFileAppender 类中的方法来完成。

4.4 本章小结

本章在前述需求分析的基础上，对多数据库中间件的设计原则、逻辑架构、物理架构进行了介绍分析并对各主要功能模块的设计与实现进行了详细阐述。

本系统主要功能模块分为连接管理、模式管理、查询处理、数据处理、事务处理以及安全控制等 6 大模块。

连接管理模块分为驱动管理、客户端连接管理、服务端连接管理以及桥接管理等四个子模块；

模式处理模块分为模式翻译与模式集成两个子模块；

查询处理模块分为查询分解、查询转换及查询优化等三个子模块；

数据处理模块分为数据预加载、数据分发管理、数据转换及数据收发等四个子模块；

事务处理模块分为事务注册、票据管理、事务分解、事务调度、事务恢复以及事务日志等六个子模块；

安全控制模块分为身份认证、访问控制、安全加密以及安全审计等四个子模块。

本章节通过对以上各子模块的具体设计与实现，最终实现完成了多数据库中间件的开发，并成功解决了 1.4 小节中提出的几个关键性问题。

5 系统运行分析

本文设计与实现的多数据库中间件已经应用于上海期货交易所 NGESIII 期统计信息系统中，本章节将从多数据库中间件系统的物理部署、运行平台、运行测试以及运行性能分析等几个角度进行系统运行分析。

5.1 系统物理部署

由于上海期货交易所系统运行环境所在地分为“两地三中心”，即上海期货大厦数据中心，上海张江数据中心以及北京数据中心，因此多数据库中间件也需三套，平时只启用某一中心的中间件作为生产运行环境，其它中间件作为故障发生时的备用系统。

上海期货交易所的历史数据分别存储于多个异构数据库中，通过使用本文所开发的多数据库中间件对各异构数据库进行逻辑集群。用户如需对这些异构数据库中的数据进行增删改查，只需与该多数据库中间件进行交互操作即可，对用户来说就像是在使用同一个数据库进行数据操作。其具体物理部署如图 5-1 所示：

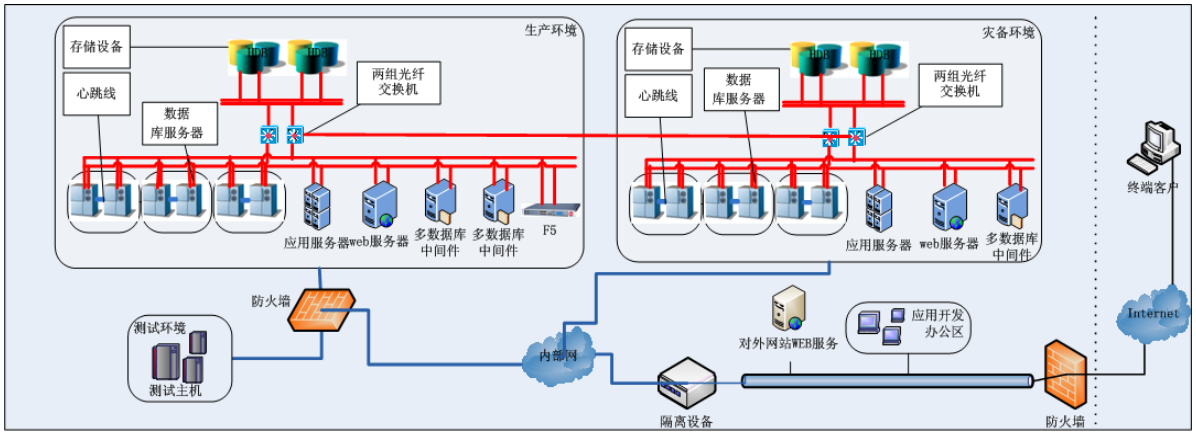

图 5-1 物理部署图

Fig. 5-1 Physical deployment diagram

从图 5-1 所知，每个数据库服务器采用了双机容错技术，即在两个数据库服务器之间增加心跳线来完成服务器的容错处理。在数据库服务器与存储设备之间使用两组光纤交换机来进行数据的通信处理。同时，采用 F5 对多数据库中间件进行多机集群负载均衡处理，防止多数据库中间件系统本身出现单点故障。

5.2 系统运行平台

系统运行平台是应用的支柱，它是能否建立一个好的应用软件系统的根本保证。平台选型是一项专业跨度大、技术难度高的工作，事关整个系统成败。本次通过遵循开放

性、成熟性、先进性、实用性以及可扩展性等选型原则并结合上海期货交易所本身系统环境的运行特点进行平台的选型。以下将从软件平台与硬件平台两个方面进行阐述。

(1) 软件平台

软件平台主要是指开发系统时所使用到的开发、建模、管理等工具以及数据库和操作系统等软体运行环境，本文所使用到的软件平台如表 5-1 所示：

表 5-1 软件平台配置表

Table 5-1 Software platform configuration table

软件类别	软件名称	软件类别	软件名称
操作系统	Redhat Enterprise Linux 5.4	数据库设计工具	Power Design 12
开发语言	Java SE 1.6，SQL	数据库系统	Oracle 11g，MySQL 5.1
开发工具	Eclipse3.4.1	项目管理工具	Microsoft Project 2007
Web 服务	Tomcat 6.0.18	测试管理工具	TestDirector 8.0
建模工具	Rational Rose 2003	配置管理工具	Visual SourceSafe 6.0

(2) 硬件平台

硬件平台主要指服务器和存储设备，按照运行环境划分可将其分成开发环境、测试环境、主生产环境、备份生产环境等 4 个部分。其中服务器按照用途又可分为 Web 服务器、数据库服务器以及应用服务器（用于运行多数据库中间件）等 3 个部分。

根据系统物理部署设计以及运行环境的不同，硬件平台的具体配置如表 5-2 所示：

表 5-2 硬件平台配置表

Table 5-2 Hardware platform configuration table

运行环境	配置种类	配置描述	配置数量
开发环境	数据库服务器	两路 4 核 CPU，内存 64G，1T 本地硬盘，4 个网路接口	3
	应用服务器	两路 4 核 CPU，内存 32G，500G 本地硬盘，4 个网路接口	2
	Web 服务器	两路 4 核 CPU，内存 16G，500G 本地硬盘，4 个网路接口	2
	存储设备	Dell EqualLogic PS6000/6100	3
测试以及主备环境	数据库服务器	两路 4 核 CPU，内存 128G，2.4T 本地硬盘，8 个网路接口	28
	应用服务器	两路 4 核 CPU，64G 内存，1.2T 本地硬盘，4 个网口	8
	Web 服务器	两路 4 核 CPU，64G 内存，500G 本地硬盘，4 个网口	8
	存储设备	HP EVA8400	12

5.3 系统运行测试

本文主要从以下几个方面对多数据库中间件进行了系统运行测试，以验证其查询数据的正确性、查询效率的高低、分布式事务处理及并发连接处理的正确性。

(1) 跨数据库查询处理测试

上海期货交易所的查询业务主要分为 2 大类：

第一类是只需要简单的查询各异构数据库上的数据，这类查询的基础数据或集中在一个数据库中，或分布在某几个异构数据库中，其结果集可在各单独的数据库中求得。对于这类业务查询，通过多数据库中间件在各个独立的异构数据库上执行查询请求，获得结果集后直接返回给前端系统，其查询示意流程如图 5-2 所示：

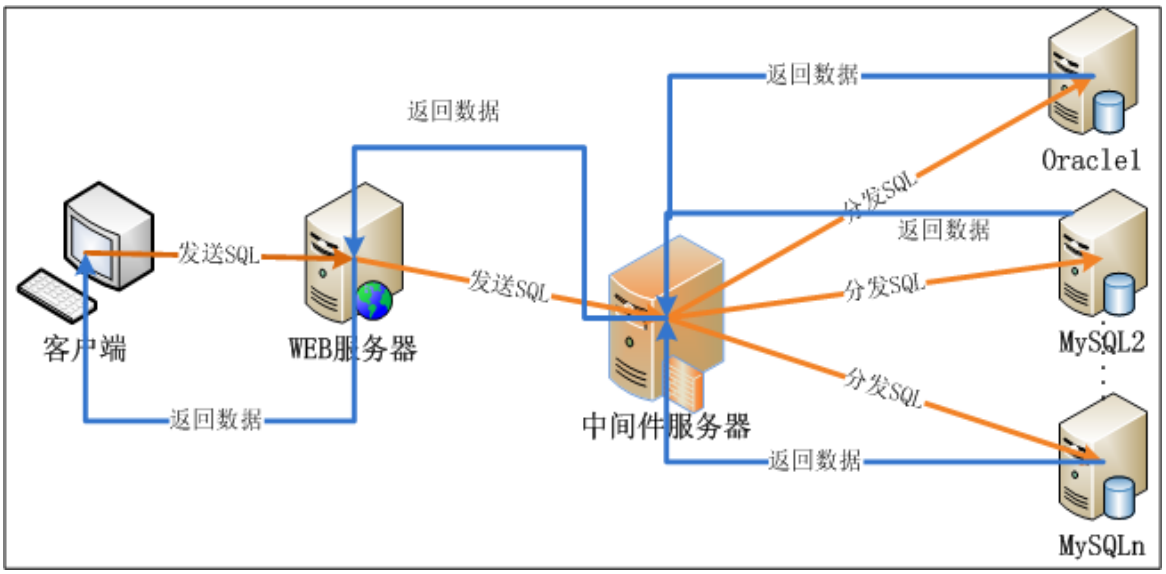

图 5-2 第一类查询流程示意图

Fig. 5-2 The first query process diagram

第二类是需要将各异构数据库上查询所得的数据进行再次排序、分组或者汇总等操作，对于这类数据的查询需要进行两阶段提交查询处理，即中间数据准备阶段（需将所得数据先保存至某几个中间临时表）和最终数据计算阶段（对这些临时表中的数据进行再次关联计算），其查询示意流程如图 5-3 所示：

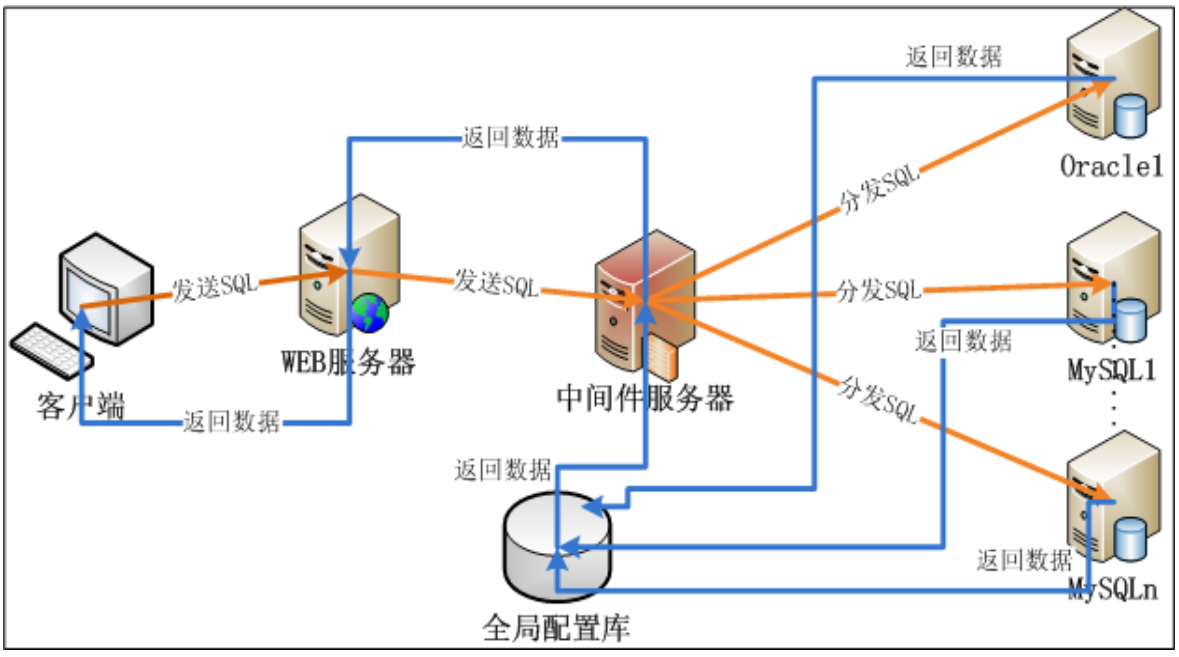

图 5-3 第二类查询流程示意图

Fig. 5-3 The second query process diagram

对于以上查询处理本文采用的测试方法是通过在客户端同时启动多个并发线程向多数据库中间件系统发出同一查询请求命令来进行测试验证。

(2) 分布式事务处理测试

通过多数据库中间件向后端各局部异构数据库系统提交事务处理，验证其事务处理的准确性，看是否保持了事务的 ACID 特性。在这里我们采用从 WEB 服务端的连接池中获取到多数据库中间件的连接，并通过该连接调用分布式事务处理命令以验证该多数据库中间件进行事务处理的准确性。

(3) 多并发连接处理测试

多并发连接处理测试主要是验证当多用户并发连接时，多数据库中间件的响应情况，连接耗时情况，查看是否会出现漏连接、异常连接、假死连接以及随着并发数的增大系统是否会发生宕机情况。

对于多并发连接处理测试本文采用通过 WEB 服务端连接池建立 10-100 个连接，并模拟 10-100 个用户同时提交查询请求，以观察多数据库中间件是否运行正常。

5.4 测试结果及分析

通过对以上多数据库中间件的运行测试，我们得出了以下系统测试结果并对这些结果进行了详细分析。

(1) 跨数据库查询测试结果及分析

对于以上第一类查询测试，我们通过多数据库中间件对分布在三个数据库上的同一张会员结算表 t_member_clear 进行查询，该表数据量为 11606847 条记录，每条记录 136 个字节。最后得出的结论是查询返回数据准确，所消耗的平均查询时间为 48 秒，该时间小于单独在每个数据库上进行查询时所需时间的总和 56 秒。

通过以上查询测试结果分析得出公式 5-1：

$$t(mdb) < \sum_{1}^{n} t(i)\ ; \qquad (5\text{-}1)$$

公式 5-1 备注：

t(mdb)：表示客户端调用多数据库中间件查询各数据库上的数据所消耗的时间；

t(i):表示客户端查询某单个数据库上的数据所消耗的时间；

对于简单的跨数据库查询，该多数据库中间件可充分发挥出其并发调度查询的功能，大大提高系统的查询性能。

对于以上第二类查询测试，我们分别启动 1-10 个并发线程通过多数据库中间件对按时间段分别存储于两个异构数据库上的客户持仓表 t_client_position 进行查询，其中在数据库 Oracle 上存储的是 2003 年的客户持仓数据，总共有 5119414 条记录，在数据库

MySQL 上存储的是 2004 年的客户持仓数据，总共有 6459344，同时将查询后的结果插入全局配置库上的某张中间临时表中，然后对该临时表做分组汇总运算得出所需结果，该类测试的运行状况记录如表 5-3 所示：

表 5-3 第二类查询测试记录表

Table 5-3 Second query test record table

	数据库 Oracle	数据库 MySQL	多数据库中间件	
	2003 年	2004 年	2003-2004 年	
并发数	查询耗时	查询耗时	总耗时	分库查询和插入临时表耗时
1	23s	32s	33s	24s
2	23s	32s	50s	41s
3	32s	44s	74s	60s
4	40s	56s	98s	79s
5	49s	67s	123s	100s
6	58s	79s	146s	118s
7	66s	91s	172s	140s
8	75s	102s	199s	163s
10	92s	122s	222s	196s

当 10 个并发调度查询的时候，多数据库中间件的内存消耗、CPU 消耗、线程数量以及被加载的对象数如图 5-4 所示：

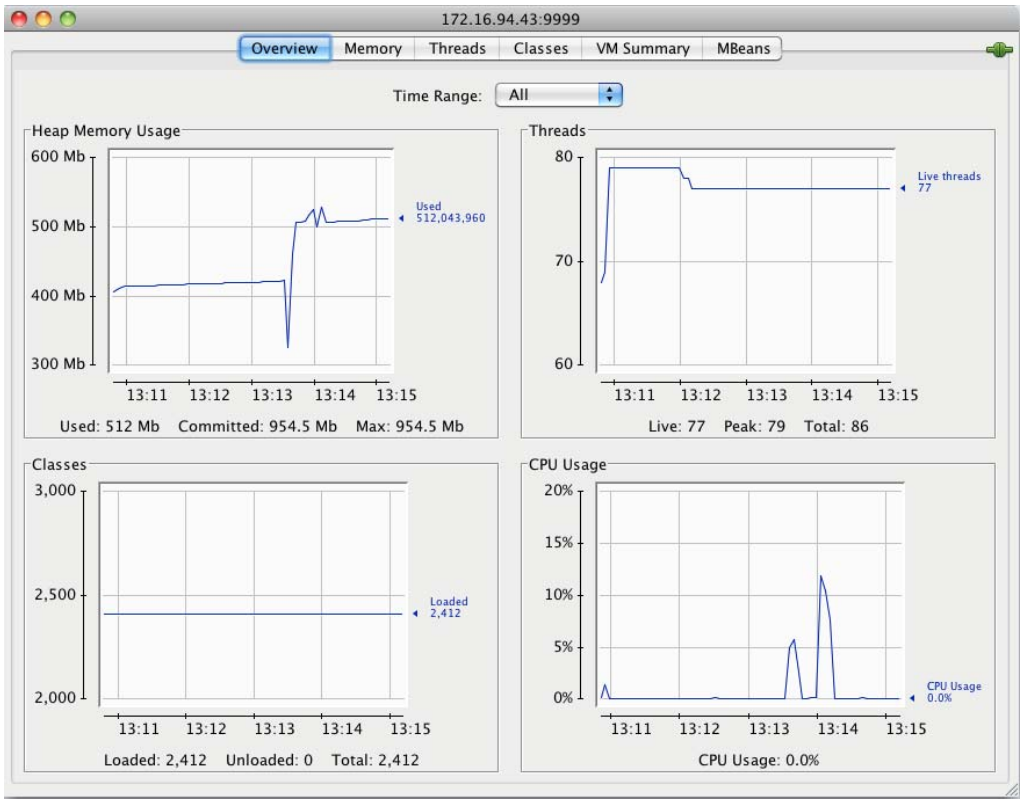

图 5-4 并发查询性能指标图

Fig. 5-4 Concurrent query performance indicators

备注

13:11－13:13 为各异构数据库数据查询阶段；

13:13－13:14 为数据插入全局配置库的临时表阶段；

13:14－13:15 为临时表中数据合并计算阶段。

从以上测试结果分析可知，随着并发数量的增加，单独使用某一数据库的查询耗时与使用多数据库中间件的查询耗时都呈线性增长，其中，各异构数据库查询和插入临时表的耗时占据了总耗时的大部分比例，因此该多数据库中间件的查询性能取决于各异构数据库对并发访问的处理能力以及对临时表并发插入的处理能力。

(2) 分布式事务处理与多并发连接测试结果及分析

通过 5.3 小节的分布式事务处理及多并发连接处理测试，测试结果及分析如下：

在进行分布式事务处理测试时，所有测试提交的分布式事务要么全部成功，要么失败全部被回滚，未出现异常的事务处理，事务处理的结果符合 ACID 特性，从测试结果可知该多数据库中间件能够正确处理各分布式的事务。

通过对多数据库中间件进行为期一个星期的多并发连接压力测试，多数据库中间件没有产生宕机，也没有出现漏连接、异常连接以及假死连接的情况。图 5-5 表示了多并发连接时多数据库的响应耗时情况。

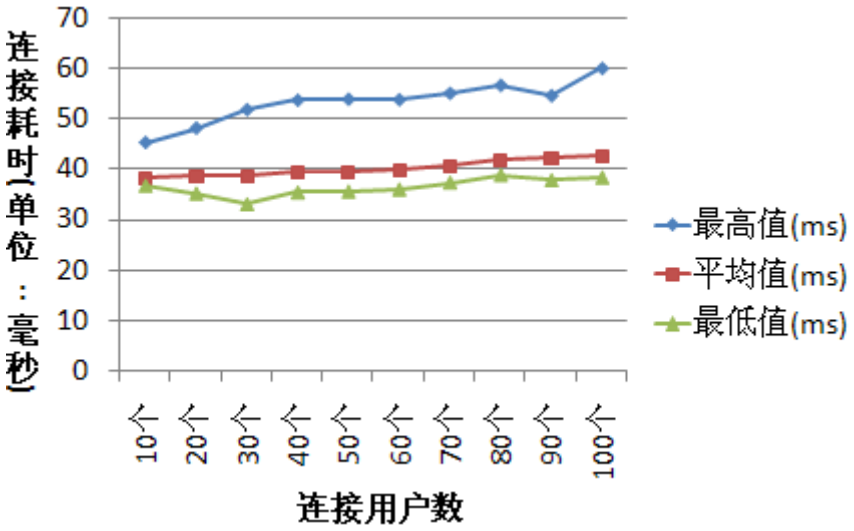

图 5-5 发连接耗时图

Fig. 5-5 Concurrent connections time-consuming

从图 5-5 可知，每次连接耗时平均在 40ms 左右，随着并发数的增加，多数据库中间件的响应耗时呈线性增加，但其表现的并不是很明显，上下相差不超过 15 毫秒。通过连接池连接多数据库中间件时，连接数量始终没有超出设置的最大值，这说明客户端从连接池获取多数据库中间件的连接时中间件能够给出正常的连接。

(3) 其它处理测试结果及分析

对于该多数据库中间件的安全控制、后端数据库动态扩展、映射表的维护等也进行了相应测试，测试结果符合系统设计要求。未经授权的用户或者 IP 地址无法访问该多数据库中间件；在多数据库中间件中新增加一个后端数据库无需启动系统即可识别和操作使用该数据库；多数据库中间件管理员可正常登陆配置管理后台对映射表进行维护。

通过上述系统运行测试结果可知，本文设计与实现的该多数据库中间件系统符合生产运行需要。目前该中间件已进入试运行阶段，运行状态良好。

5.5 本章小结

本章对多数据库中间件的物理部署、运行平台、运行测试以及测试结果进行了详细阐述与分析。其中物理部署主要从系统的实际运行环境，数据存储特点以及存储安全性等角度进行了分析；运行平台主要从软件平台和硬件平台两个角度进行了分析；运行测试主要从多库查询处理、分布式事务处理以及多并发处理等角度进行了阐述；测试结果及分析主要是对该中间件的测试结果进行具体分析。

通过对以上多数据库中间件的运行分析，不仅提高了该中间件在生产环境中运行的安全稳定性，而且为下一步的系统改进提供了数据参考依据。

6 总结与展望

本章以下部分将对上述几个章节所阐述的内容做一个总结，对本文设计与实现的多数据库中间件系统存在的不足进行分析，并在此分析的基础上对下一步的研究工作进行展望。

6.1 本文工作总结

本课题是上海期货交易所 NGESIII 期（新一代交易所系统第三期）统计信息系统项目中的一子课题。其研究目的是为了解决多个数据库系统的集成问题，即如何将各个自治的、异构的数据库系统集成为一个统一的逻辑数据库系统，使其向外界提供一致的数据库访问视图及接口。

本文通过采用设计与实现一个基于 JAVA 的多数据库中间件来解决以上提出的多数据库集成问题，研究的工作内容主要有以下几个方面：

(1) 对开发实现多数据库中间件时所用到的相关技术以及需要解决的一些关键性问题进行了详细探讨和分析；

(2) 将多数据库中间件按功能模块进行了划分，分别为连接管理模块、模式处理模块、查询处理模块、数据处理模块、事务处理模块以及安全控制模块等六大模块，并详细分析了这些各主要模块的功能性需求和非功能性需求；

(3) 在以上需求分析的基础上，对上述各主要功能模块进行了详细设计和实现，并最终完成了多数据库中间件的开发；

(4) 对本文实现的多数据库中间件从多个角度进行了系统运行分析，并将其成功的应用于了实际的生产环境中。

本文研究内容的成果主要有以下几个方面：

(1) 通过采用 JAVA 相关技术对多数据库中间件进行开发，使得其可以跨平台运行；

(2) 实现了一套符合 JDBC 规范的自定义驱动程序，各客户端只需调用该驱动接口即可实现与多数据库中间的通信，并最终通过该多数据库中间件查询处理后端各数据库上的数据。由于此驱动程序是符合 JDBC 规范的，因此该多数据库中间件的全局查询语言仍然采用标准的 SQL 语句，而无须再自定义一套别的查询处理语句，因而从某个角度来说则是降低了用户的学习曲线；

(3) 给出了采用 javaCC 技术实现所有遵守 ANSI SQL 标准的 SQL 语句语法解析方法，从而解决了全局 SQL 分解的难题；

(4) 通过采用 SHA 安全散列非对称加密算法以及 IP 访问控制技术，实现了多数据库中间件的安全访问控制。

(5) 通过实现 JTA 规范接口，采用二阶段提交协议，实现完成了多数据库中间件的分布式事务处理功能；

(6) 通过采用线程监控技术，实现了多数据库中间件对数据库的动态挂载以及负载均衡处理；

(7) 给出了两阶段提交查询方法，通过该方法解决了跨多个数据库的排序、分组以及汇总等问题；

(8) 给出了多数据库中间件分发调用标准存储过程的方法；

6.2 展望

虽然本文设计与实现的多数据库中间件已经开始在生产环境中试运行，但是还是有许多地方值得进一步研究和改善，其主要包含以下几个方面：

(1) 本文研究设计与实现的多数据库中间件到目前为止还只支持 SQL 语句的缓存处理，对于数据的缓存处理还未实现；

(2) 该多数据库中间件挂载的数据源只支持关系型数据库，对于对象型数据库以及其它形式的数据源缺乏支持；

(3) 对于模式冲突的处理还只能依靠手工进行维护解决，不能做到程序自动处理；

(4) 本文采用的分布式事务处理方法只支持那些已经实现了两阶段提交协议的数据库系统，对于那些没有实现该协议的数据库系统则不能通过本文提出的以上方法解决，同时该多数据库中间件对嵌套事务还不支持；

(5) 该多数据库中间件只支持 JDBC 驱动连接，对于其它协议的连接则无法支持；

(6) 在两阶段提交查询的方法中，目前只支持将中间数据存储于会话级的临时表中，对其它形式的存储方式还缺乏支持；

鉴于以上所列举出来的不足之处，我们将在下一步的研究工作中进行弥补。

参考文献

[1] S1lirley A.Becker，Gibson，NaIlcy L.LeiSt.A Study of a Genricc Schema for Management of Multidatabase Systems[J]，Joumal of Database Management，1996，7(4)：14-20

[2] Ioarmidis YE，Kang YC.Left-deep vs. Bushy Trees：an Analysis of strategy Spaces and its Implications for Query Optimization[C]，In：Proe.ACM SIGMOD Conf. Management of Data. DenvegColo，May 1991：168-177

[3] Qiang Zhu，P.Larson，Global Query Proeessing and Optimization in the CORDS Multidatabase System[C]，In:Proc. of 9th ISCA Int'Conf. on Parall and Distr. Comput.Syst，1996：25-27

[4] Mary Tork Roth，Manish Arya，Laura M.Haas，et.al. The Garlic Project. SIGMOD Conference，1996，49-57

[5] Michael J.Carey，Laura M.Haas，Peter M，et.al. Towards Heterogeneous Multimedia Information Systems：The Garlic Approach. Proc. RIDE-DOM'95，1995，124-131

[6] A Silberschatz，Z Kedem，Consistency in hierarchical database systems [J]，Journal of the ACM，1980，27(l)：72-80.

[7] Jakobovits R.Integrating Autonomous Heterogeneous Information Sources [J]，IEEE Computer Society Press，2003，34(2)：122-130.

[8] Witold Litwin，R.Ahmed，The Pegasus heterogenous multi-database system [J]，IEEE Computer，1991，24(12)：19-27

[9] Pegasus，http://www.openpegasus.org

[10] H.Garcia-Molina，J.Hammer，K.Ireland，et.al. Integrating and Accessing Heterogeneous Information Sources in TSIMMIS. In Proceedings of the AAAI Symposium on Information Gathering，Stanford，California，March 1995，61-64

[11] S.Chawathe，H.Garcia-Molina，J.Hammer，K.Ireland，et.al，The TSIMMIS Project：Integration of Heterogeneous Information Sources. In Proceedings of IPSJ Conference，Tokyo，Japan，October 1994，7-18

[12] SIMMIS， http://www-db.stanford.edu/tsimmis/tsimmis.html

[13] A.Dogac，C.Dengi，E.Kilic，G Ozhan，F.Ozcan et.a1. METU Interoperable Database System. ACM Sigmod Record，1995，24(3)：56-61

[14] C.Collet，M.Huhns，and W.Shen，Resource Integration Using a Large Knowledge Base in Carnot. IEEE Computer，1991，24(12)：55-62

[15] D.Woelk，P.Cannata，and C.Tomlinson，Using Carnot for Enterprise Information Integration. In：Second International Conference on Parallel and Distributed Information

Systems. London，United Kingdom. 1993：133-136
[16]石祥滨，郑怀远，面向对象的多数据库技术[J]，计算机科学，1996，23(5)：33-35
[17]李瑞轩，卢正鼎，多数据库系统原理与技术[M]，北京：电子工业出版，2005:50-320
[18]王宁，徐宏炳，王能斌，基于带根连通有向图的对象集成模型及代数[J]，软件学报，1998，9(12)：894-898
[19]王宁，陈滢，俞本权等，一个基于 CORBA 的异构数据源集成系统的设计[J]，软件学报，1998，9(5)：378-382
[20]姚卿达，何听，黄晓春等，LNFDBS 的查询优化算法及联邦条件下的考虑[J]，软件学报，1998，9(6)：453-457
[21]S.Agarwal，A.M.Keller，An Approach for Integration Data from Multiple Possibly Inconsistent Databases，Proc. IEEE Int．Conf．on Data Engineering，1995．
[22]韩伟红，贾焰，王志英，杨晓东，多数据库系统中的关键技术[J]，计算机工程与科学，1996，(6).
[23]贾焰，王志英，韩伟红，李霖，分布式数据库技术[M]，国防工业出版社，2000，126-153.
[24]Hector Garcia-Molina，数据库系统全书[M]，北京：机械工业出版社，2003，587-629
[25]张兵，张荣肖，潘玉平. 联邦数据库系统[J]，计算机系统应用，1995，(01).
[26]李瑞轩，异构信息集成中的查询处理与优化研究[D]，华中科技大学，2004，39-40.
[27]D.Georgakopolous，M.Rusinkiewcz，A.Sheth，On Serializability of Multidatabase Transactions Through Forced Local Conflicts，In Proceedings of the Seventh In-ternational Conference On Data Engineering，Kobe，Japan，1991．
[28]魏勇，张权. 中间件技术研究[J]. 电子技术应用，2004，(11):1-4
[29]王辉，施小英，中间件服务及其集成框架[J]，电脑与信息技术，1997，(05)：12-15
[30]DeTurck，F.Volckaert，B.Demeester，P.VanhaStel，Stefaan. A generic middleware_based Platform for scalable cluster computing [J]，Future Generation Computer Systems. March 2002:549-560
[31]Nahrstedt，K.Xu，Wichadakul，D.Li. QoS-aware middleware for ubiquitous and heterogeneous environments [M]，IEEE Communications Magazine. November 2001:140-148.
[32]Desmond Dsouza. Enterprise Integration [J]，Software Development，1999:20-23
[33]王艳华，基于中间件技术的分布式数据集成研究与实现[D]，武汉理工大学硕士论文，2006，16-17
[34]郑雪，徐亚娟．中间件的概念、分类与应用[J]，微型电脑应用，1999，(02)：15-17
[35]蓝雯飞，邹吕新，对象中间件技术及其发展趋势[J]，中南民族大学学报(自然科学版)，2006，(03).

[36]俞红奇，丁宝康，多数据库环境下的模式集成及查询分解[J]，计算机工程，2000，26(10).
[37]A.K.Elmagarmid，C.Pu．Special Issue on Heterogeneous Databases．ACM Computing Surveys，1990，22(3)：175-178
[38]S.Adali et a1．Query caching and optimization in distributed mediator systems．In Proc．of SIGMOD．1996，137-148
[39]W Du，et a1．Query optimization in heterogeneous DBMS．In Proc．of VLDB，1992，277-291
[40]肖卫军，李兵等一种多数据库事务模型[J]，小型微型计算机系统，2003，224-226.
[41]Abraham Silberschatz 等著，杨冬青等译，数据库系统概念(原书系 5 版)[M]，北京:机械工业出版社，2006.10， 414-443
[42]Gligor V.，and PoPescu-Zeletin R.，Transaction Management in Distributed Heterogeneous Database Management System．Information Systems，1986，287-296
[43]W.Du and A.Elmagarmid. Quasi serializability:a correctness criterion for global concurrency control in Intertrase.Poc.13th International Conference on very large Databases，Aug 1989，11-21.
[44]H.F.Korth，E.Levy，A.Silberschatz，A Formal Approach to Recovery by Compensating Transactions，Proceedings of the Sixteen International Conference on VLDB，Brisbane，PP，August 1990，95-106
[45]Mehrotra S.，Rastogi R.，and Breitbart Y.，The Coneurreney Control Problem in Multidatabase: Characteristics and Solutions. 1992，ACM，288-297
[46]Breithart Y.， SilbersehatZA.，and Thompson G.R.，Reliable transaction management in a multidatabase system. Proeeedings of ACM-SIGMOD 1990 International Conference on Management of Data，Atlantic City，New Jersey，1990，2l5-224.
[47]J.Gray，A.Reuter. Transaction Processing: Concepts and Techniques. San Francisco: Morgan Kaufmann，1993.
[48]齐永强，熊家军，龙中. JDBC 连接数据库[J]，重庆邮电学院学报，1999，(01).
[49]张作宸，JDBC 原理及操纵数据库方法[J]，科技信息，2009，(02).
[50]JDBC overview， http://www.oracle.com/technetwork/java/overview-141217.html
[51]郭广军，陈代武，胡玉平，李芝城，基于 JDBC 的数据库访问技术的研究[J]，南华大学学报(自然科学版)，2005，(02).
[52]王金海，江务学，基于 Java 多线程的并发机制的研究和实现[J]，微机发展,2004，(03).
[53]龚成清，Java 多线程的创建和线程同步的实现[J]，宁波职业技术学院学报，2007，(2).
[54]王保华，薛锦云，丁树良，Java 多线程机制及其应用[J]，计算机与现代化，2000，

(01).
[55]闫静，王联辉，基于 JavaCC 的 SQL 编译器的设计与实现[J]，微计算机信息，2010，(15).
[56]马喜春，张曾科，基于 Socket 进行通用的网络通信程序设计[J]，实验技术与管理，2005，(03).
[57]赵晓君，熊前兴，吴业福等，基于 J2EE 的分布式事务研究[J]，交通与计算机，2006，(6).
[58]齐勇，马莉，赵季中，齐向明，侯迪，分布式事务处理技术及其模型[J]，计算机工程与应用，2001，(09).
[59]瞿裕忠， 张剑锋， 陈峥， 王丛刚，XML 语言及相关技术综述[J]. 计算机工程，2000，(12).
[60]刘晟，王振兴，王智强. XML 与 Java 的结合[J]，微电脑世界，2002，(06).
[61]张英，杜炜，JDBC 数据库连接池技术简介及设计[J]，电脑知识与技术，2010，(08).
[62]王秀义，基 JDBC 的数据库连接池及实现[J]，计算机系统应用，2005，(04).
[63]Eastlake D，Jones P. US Secure Hash Algorithm1 (SHA1) [S]. RFC 3174，2001.
[64]林雅榕，侯整风，对哈希算法 SHA-1 的分析和改进[J]，计算机技术与发展，2006，(03).
[65]王继成，高珍，软件需求分析的研究[J]，计算机工程与设计，2002，(08).
汪作文，试论面向对象的设计模式[J]，科技信息，2009，(28).

附 录

附录1　处理多数据库负载均衡线程的 run()方法具体实现代码

```
public void run() {//线程运行时调用的方法；
HeartbeatDelayed delayed = null;
try {//将程序放入 try-catch 块中捕捉异常
while (true) {//循环心跳检测；
delayed =heart_beat_queue.take();//从阻塞对列中获取一个 Delayed 对象；
//心跳检测连接池中的对象是否有效；
ObjectPoolStatus status = delayed.virtualPool.statusCheck(delayed.pool)；
if (status.status == ObjectPoolStatus.STATUS.INVALID) {//如果状态为无效；
delayed.setDelayedTime(5, TimeUnit.SECONDS);//设置下一次检测的间隔时间：5 秒；
heart_beat_queue.offer(delayed);//将该 Delayed 对象重新放入阻塞队列里；
} else {//如果状态为有效；
status.lock.lock();//考虑到并发运行，对该对象进行加锁；
try {status.inChecked = false;//将状态属性设置为不需要检测；
} finally {tatus.lock.unlock();//对该对象进行解锁；}}}
} catch (InterruptedException e) {
logger.error("心跳检测失败!",e); //抛异常处理，记录日志；
}
}
```

附录2　BalanceParser.jjt 文件内容说明

BalanceParser.jjt 文件总共分为 5 部分，具体文件内容说明如下：

第 1 部分：定义运行环境选项

```
options {//定义运行 jjt 文件时的相关环境变量
//产生 JAVA 文件时所使用的 JDK 版本号
    JDK_VERSION = "1.5"；
//忽略大小写
    IGNORE_CASE=true ；
//是否调试输出日志
    DEBUG_PARSER=false ；
//编译后 jj 及 JAVA 文件的输出目录
```

```
        JJTREE_OUTPUT_DIRECTORY ="sql"；
//从节点标志符构造节点类名称的前缀
        NODE_PREFIX="Balance"；
//  节点范围钩子，确定是否需要在每个节点作用域的入口和出口处调用用户自定义方法
        NODE_SCOPE_HOOK=true；
//创建多模式解析树
        MULTI = true；
//不在使每一个非包装产生式中定义一个节点
        NODE_DEFAULT_VOID = true；
//是否根据静态解释器来生成代码
        STATIC = false；
    }
```

第 2 部分：解析语法类 BalanceParser 定义

```
PARSER_BEGIN(BalanceParser)   // BalanceParser 类定义开始标志
//各具体方法定义
         //语法解释方法，返回解释后的对象 Statement，用于后续模式匹配
        public Statement doParse() throws com.shfe.report.parser.ParseException{
            try{
                Statement statement = this.parse();
                if(statement != null){
                    statement.setParameterCount(parameterIndex);
                }
                return statement;
            }catch(Exception e){
                throw new com.shfe.report.parser.ParseException(e);
            }
    }
        //由于篇幅关系，其它具体方法实现在此省略。
PARSER_END(BalanceParser)   // BalanceParser 类定义结束标志
```

第 3 部分：定义空格、跳格、回车和换行作为分隔符处理

```
    SKIP:{   " "|    "\t"|    "\r"|    "\n"}；
```

第 4 部分：定义词法分析器

该部分主要是用于标识 SQL 语句的关键字（如：SELECT、INSERT、UPDATE、DELETE、FROM 等），数字，SQL 注释，特殊字符（如：“$”、“_”、“@”等）。

```
    TOKEN:
    {<K_SELECT: "SELECT">| <K_INSERT: "INSERT">
    | <K_UPDATE: "UPDATE">| <K_DELETE: "DELETE">|<K_FROM:”FROM”>
```

```
|<#DIGIT: ["0" - "9"]>|< INTEGER_LITERAL: (<DIGIT> )+ >}
SPECIAL_TOKEN:
{<LINE_COMMENT: "--"(~["\r","\n"])*>
| <MULTI_LINE_COMMENT: "/*"(~["*"])* "*" ("*" | (~["*","/"] (~["*"])* "*"))* "/">
| < #SPECIAL_CHARS: "$" | "_"|"@">}
```

第 5 部分：定义语法分析器，即解析规则产生式

该部分主要是对 SQL 等语句的一个语法解释，用于验证这些语句语法的正确性。并将分析后的表达式值存储于相关 Statement 对象中，以便与模式表中存储的规则进行比较，从而分析出这些 SQL 语句所关联的数据库。其部分代码实现如下：

```
Statement parse() ：
{//方法变量声明部分
    Expression expression = null；
    Statement statement = null；
}
{//语法规则解析部分
     ["EXPLAIN"] ((
     statement = DeleteQuery()|//解析删除语句
     statement = InsertQuery()|//解析修改语句
     statement= SelectQuery()|//解析查询语句
     statement = UpdateQuery()|//解析更新语句
     <EOF>//文件结束标志符))
     {return statement； }}
```

附录3　解析后 SQL 语句与规则映射表比对分解代码

```
//将 sqlStatment 对象中的内容经过计算转换生成出一个 Map 对象 tables
Map<Table, Map<Column, Comparative>> tablesMap = sqlStatment.evaluate(parameters)；
//将 tablesMap 对象和 tableRuleMap 对象进行比较得出结果
Set<Map.Entry<Table, Map<Column, Comparative>>> entrySet=tablesMap.entrySet()；
//循环比较
for (Map.Entry<Table, Map<Column, Comparative>> entry : entrySet) {
//从客户端接收到的 SQL 语句中获取每张表对应的查询条件信息
  Map<Column, Comparative> columnMap = entry.getValue()；
//从数据存储规则映射表中获取以上相同表的数据存储规则信息
TableRule tableRule = tableRuleMap.get(entry.getKey())；
//循环读取表规则信息
for (Rule rule : tableRule.ruleList) {
```

```
//建立比较对象数组，数组的大小与映射表中参数个数相同
Comparable<?> [] comparables = new Comparable[rule.parameterMap.size()];
//从规则中循环取出参数信息
for (Map.Entry<Column, Integer> parameter : rule.cloumnMap.entrySet()) {
//从查询条件信息中取出对应参数的参数条件值
Comparative condition = columnMap.get(parameter.getKey());
//将 condition 克隆值赋给先前创建的比较对象数组 comparables
comparables[parameter.getValue()] = (Comparative) condition.clone();
}
//将查询中的参数条件值与映射表的参数条件值比较匹配
Comparable<?> result = rule.rowJep.getValue(comparables);
//判断是否匹配成功
if((Boolean) ((Comparative) result).getValue()){//如果匹配成功
//获取映射表中对应规则的数据库连接池名称
  poolsName = dmlStatment.isReadStatement() ? rule.readPools : rule.writePools;
}else{//如果匹配不成功则赋予缺省连接池
  poolsName = this.defaultPools;
}
//根据得到的连接池名称从系统中获取真正的连接池
ObjectPool[] pools = new ObjectPool[poolNames.size()];
            int i = 0;
          for (String name : poolNames) {//循环获取
    ObjectPool pool = ProxyRuntimeContext.getInstance().getPoolMap().get(name);
            pools[i++] = pool; }}}
```

附录4　安全加密部分代码

```
//获取一个实现了 SHA-1 算法的 JAVA 对象
MessageDigest md = MessageDigest.getInstance("SHA-1");
//对密码进行第一次摘要计算
byte[] passwordHashStage1 = md.digest(password.getBytes());
//重置 md，为进行第 2 次摘要做准备
md.reset();
//在第一次摘要的基础上进行第 2 次摘要计算
byte[] passwordHashStage2 = md.digest(passwordHashStage1);
//重置 md，为再次更新摘要做准备
```

```
md.reset();
//将加密随机字符串转换为 byte 数组
byte[] seedAsBytes = seed.getBytes();
/ /以上面计算出的 byte 数组以及第 2 次计算出的哈希散列值为条件对摘要进行更新
md.update(seedAsBytes);
md.update(passwordHashStage2);
//获取计算出的摘要结果
byte[] toBeXord = md.digest();
//对上面所得的摘要结果与第 2 次所得的摘要结果按位异或计算，得出最终摘要值
for (int i = 0; i < numToXor; i++) {
        toBeXord[i] = (byte) (toBeXord[i] ^ passwordHashStage1[i]);
        }
```

致 谢

本论文的完成得益于很多人的帮助、关心与支持，请允许我在此向他们表示衷心的敬意和感谢。

首先要感谢的是我的导师肖双九老师，肖老师在百忙之中抽出时间对我的课题研究给予了悉心指导，在她的耐心帮助和指导下最终完成了本篇论文。肖老师和蔼可亲、平易近人的品格，精益求精的态度，广纳并蓄的研究方法，诲人不倦的教育精神都将对我以后的学习和工作产生深远的影响。借此机会，谨向恩师致以最崇高的敬意和最诚挚的感谢！

感谢孙德文老师，您严谨求实的治学风范，一丝不苟的处事精神，尽心尽责的工作态度为我以后的工作和生活提供了学习榜样。

感谢上海交通大学软件学院。软件学院为我提供了这次难得的学习机会，使我学到了很多丰富的专业知识，开拓了我的视野，为我以后更好的工作打下了坚实基础。

感谢本次项目组全体同事，特别是张宇灏和项晶，由于你们的团结协作和辛勤的工作，才使得本次项目得以胜利的完成结束，在此向你们表示衷心感谢！

感谢我的家人，感谢我的父母!感谢他们在我学习期间给予我的不懈支持、鼓励，他们无私的厚爱和殷切的期望是我在人生道路上不断前进的巨大动力。特别要感谢的是我的爱人杨静，在我攻读工程硕士学位期间，自始至终给我无微不至的照顾、关心和爱护，风雨同舟，患难与共。此时此刻，我唯有希望此文能稍稍回报你多年来对我的理解与支持!

最后感谢各位专家和评委耐心审阅我的论文并提出宝贵意见。

成功的道路上没有独行者，感谢所有给予我帮助的人！谢谢您们！我不会辜负你们的期望，将会以饱满的热情努力工作回报社会。

攻读学位期间发表的学位论文目录

［1］石明辉，基于 JAVA 的多数据库系统集成技术研究[J]，上海交通大学软件学院网站公示，2011 年 3 月