中文摘要

本文首先叙述了移动计算环境的形成过程，阐明了有关数据同步的核心概念，并从中提炼出了广义的数据同步定义。针对数据同步技术的两大要点难点：异构数据的映射问题和冲突处理机制的实现，在简介其基本概念和常用技术后，拟定出基于移动计算环境下异构数据库数据同步实现的解决思路。

本文分析讨论了当今业界各主流数据访问技术在异构数据的映射问题和冲突处理机制的实现这两个问题上做出的成绩和存在的不足。总结出要在移动计算环境下实现异构数据库的数据同步，必须获得更高层次的专用数据同步协议支持的结论。

针对这一结论，本文介绍了一个旨在同步任意移动设备和任意网络数据的协议—SyncML 协议。它通过对两大主题：XML 级应用和 Agent 技术的支持，利用同步引擎技术能够较好的解决异构数据映射问题和实现冲突检测机制，遗憾的是，它没有对数据同步的另一难点—冲突处理机制的解决策略提供强有力支持。因此，我们通过增加同步引擎对开放式数据库接口的支持，以两极冲突处理机制来弥补这一不足。综合上述机制提出基于移动环境下的异构数据库数据同步模型架构。

论文针对移动计算环境下的普遍应用情况，提出“在低端加强移动性，在高端提供高性能”的创新思想，结合移动数据库容量小，断接频繁；主存数据库响应时间短，存取数据快的特点，遵循 SyncML 协议，在移动环境下的异构数据库数据同步模型架构之上，搭建起一个移动客户端使用移动数据库技术，服务器端采用定制主存数据库技术的异构数据快速同步模型。实验表明，这是对数据同步技术的明显改善。拟将之运用在民航移动调度子系统中。

关键字：移动数据库；主存数据库；复制；同步；SynCml 协议；Sync4j

Abstract

In this thesis, we first depicted the source of mobile computing environment, elucidated some core concepts of data synchronization, and abstracted the broad definition of data synchronization. Aimed at the two nodus: the mapping of heterogeneous data; and the realization of conflict disposed mechanism, we introduced basic conceptions; general technology, study out a method to realize the synchronization of data in those heterogeneous database based on mobile computing environment.

The thesis analysed achievement and disadvantage that nowadays many main data accessible technologies had upon the two nodus. Drew a conclusion that in computing environment, the data synchronization of heterogeneous database couldn't be realized without support of specially advanced data synchronization protocol.

So, the paper introduced the SyncML protocol which applied for the purpose of synchronizing each mobile device and network data. It solved the problem that mapping of heterogeneous data by synchronized engine technology, the application of XML and agent technology. It also realized conflict detected mechanism. But regretfully, it provided little policy to another nodus: conflict resolved mechanism. So we strengthened the synchronized engine's support to open database interface, improved it by two-floor conflict disposed mechanism. Based on these method, we conceived the frame of data synchronization of heterogeneous database found on mobile computing environment.

Considered usual condition in mobile computing enviorenment, the thesis put forward a creative thought--"more mobility on client, high capability on server", combined that mobile database has a small capacity, connected and disconnected frequently; memory database has short latency time, fast data accessibility; therefore, according to SyncML protocol; based on the frame of data synchronization of heterogeneous database, built a model of rapid synchronization of heterogeneous data, using mobile database technology on client and customized memory database technology on server. Through experiments, it has been proved to be a improvement of data synchronization technology, and has been applied to the Mobile Dispatch system of the China Airdrome.

Keyword: mobile database　memory database　replica　synchronize　SyncML protocol　Sync4j

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：罗谦　　日期：2023年5月19日

关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将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、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：罗谦　　导师签名：

日期：2023年5月19日

第一章 引言

1．1 论文的研究背景

随着计算机技术、通讯技术和网络互联技术的迅速发展，信息管理已发展到以数据库为处理中心，移动计算、WEB 计算以及数据挖掘和开采等为技术手段的阶段。大量 PDA、掌上电脑和笔记本电脑等移动设备的普及应用，激发了个人通讯网（PRN）和网络计算机（NC）等概念的产生。移动计算网络环境正逐步形成[17,18,26]。基于移动计算的网络环境以其鲜明的特点[1,2]：移动性、断接性、带宽多样性、可伸缩性、弱可靠性、网络通讯的非对称性、电源能力局限性等等对相关的计算机技术提出了新的要求。移动办公、移动调度、移动计算等实际应用促动了网络技术、通讯技术、数据库管理技术向移动领域的延伸。适合于移动计算环境的数据库体系结构也由传统的 C/S、B/S 架构，先演变为 Client/Agent/Server[4]架构，再变至 Client/Intercept/Server[3]架构。文献[4,5]中指出：Client/Agent/Server 架构和 Client/Intercept/Server 架构通过糅合 Agent 技术很好地解决了客户机与数据服务器之间因为移动计算网络的低带宽、高延迟和易中断等特点所带来的网络连接问题，提高了网络的利用率；但它在移动客户机的移动性管理、跨区性操作以及移动客户机与数据服务器之间数据同步控制等方面都未加以考虑。因此，本文将从研究异构数据库间数据同步控制技术入手，提出移动数据库与主存数据库间的异构数据快速同步模型，该模型具有（1）客户机的移动性强（2）数据服务器的效率高（3）快速收敛异构数据等三大特点，从一定程度上填补了 Client/Agent/Server 和 Client/Intercept/Server 架构的不足，充分展现了移动计算领域广阔前景。

1．2 论文的核心概念

数据同步有狭义和广义之分。为使概念清晰易懂我们先介绍数据同步技术的分类。

1．2．1 数据同步技术的分类[19]

1．2．1．1 在线（Online）

Online 是直接对所有相关节点的数据进行修改，并以传统分布式系统的两阶段提交（Two Phase Commit Protocol）方式解决数据的一致性问题。典型应用如：民航售票系统、ATM 等。

Online 的特征为（1）低级别的本地自治（2）数据的紧密一致性（3）通过

DTS（分布式事务管理服务器）确保事务在所有节点上的提交和回滚（4）典型的对等拓扑（Peer to Peer）（5）较难支持异构环境

1．2．1．2 狭义的同步概念（Synchronization）

Synchronization 是将当前状态的最终结果数据回传至相关节点，并更新对应的数据以维护数据的一致性，忽略事务的执行过程细节。执行该操作时可以根据需求选择更新区域的大小。

Synchronization 的特征为（1）高级别的本地自治（2）数据的松散一致性（3）不保存事务的状态（4）需冲突检测和解决机制（5）支持异构环境

1．2．1．3 复制（Replication）

Replication 是将更新事务集传递到相关节点上运行，通常利用消息机制和存储转发机制实现。

Replication 的特征为（1）高级别的本地自治（2）吞吐量与数据库大小无关（3）保存事务状态（4）支持异构环境（5）低延迟、准实时

当在移动环境中讨论数据的一致性问题时，基于弱一致性[7]的前提下，如果存在大量节点、传输的数据量较小且无需保存事务现场则以狭义概念的同步技术为佳；如果有中等数量的节点但有大量的数据需要更新且必须保留事务现场则应首选复制技术。目前国内外对数据一致性问题展开研究并已成型的有：美国 CMU 研制的 CODA 系统（复制技术）、Sybase 公司的 MobiLink（狭义概念的同步技术）和 Replication Server（复制技术）等；国内则是以人大金仓公司的 KingBase Lite2.0 为代表。

1．2．2 广义的数据同步概念

业界对数据同步并没有严格意义上的定义。文献[4,5]中指出：“数据同步是一种允许在不同计算机上的多个数据库间保持数据一致性的技术手段”；在 SyncML 协议的白皮书上是：“数据同步是一个使得两个集合中的数据看起来一致的处理过程”[6]。结合上述概念并放在特定的移动计算环境中，我们对数据同步下一定义，为区别于狭义概念的同步技术故称之为广义概念上的数据同步。

定义 1：广义的数据同步是指能在不同设备、不同平台上的异构数据库间保持多个副本一致性的技术手段。

很显然从定义 1 中可以看出，狭义概念的数据同步只是广义概念数据同步的技术手段之一。在本文论述中如不加以明确说明，数据同步均指广义概念上的数据同步定义。

1．2．3 数据同步的对象

对数据同步而言，它处理的对象往往可分为：

1．2．3．1PIM 的同步

PIM（Personal Information Management）即个人信息管理是对个人产生的相关信息的存储、操作和应用等（如：电子名片、E_MAIL 等）。这些内容是数据同步初期的处理的主要对象，通常采用 GroupWare（群件技术）来实现（如 Microsoft Exchange 等）。但由于其数据类型简单、操作直接且不涉及系统管理而不是数据同步技术的研究热点。

1．2．3．2 关系数据库的同步

主要实现移动设备上的商业数据与一个企业应用程序或后台数据库数据同步，并维持数据间的复杂关系不变。这一问题在现代企业中有着广泛的需求，但数据类型多、映射复杂、需冲突处理机制等难题期待着数据同步技术的进一步突破。本文将重点讨论这个问题。

1．2．3．3 无结构数据对象的同步

移动设备和中心系统间除了有关系数据库的同步需求以外，还有一些无结构数据的同步需求，如对平面文档（WORD、PowerPoint 文档）的同步。这类型的同步已超出了一般意义的同步技术范围，但却是数据同步将来仍要面对的难题。

在移动计算网络环境下实现数据同步较固定网络环境下难度更大，而其中又以如何解决移动环境下的异构数据映射问题和实现冲突处理机制[21]为研究的重中之重。

1．2．4 异构数据间的相互映射

由于要在不同设备（如：移动电话、掌上电脑、便携式电脑以及 PC）、不同平台上（如：Linux、Windows）的不同数据库（如：SQL Server for CE、DB2、SQL Server、PointBase）之间实现数据的同步，其首要问题就是解决异构数据间的映射问题（即一个字符集如何转换为另一个字符集）。本文将沿着数据访问技术的研究、XML 语言的出现、SyncML 协议的提出至 Sync4j 的具体实现这一主要线索说明该问题求解的演变历程及其工程技术的应用。

1．2．5 冲突处理机制

当对同一对象的不同数据副本在同一时刻执行了不同的更新操作时，将出现更新冲突（Update Conflict）[8]。这种更新冲突如果不能检测出来并加以处理的话，就会破坏数据的一致性状态，形成脏数据，造成事务的无效执行，最终获得

错误结果甚至可能导致系统崩溃。研究其实质是由于移动事务的局部提交而使数据副本中的对象处于一种暂时的不一致状态。如果能及时通过冲突检测机制发现它，再由冲突解决机制将这种不一致状态转变为一致性状态（即数据的重新收敛），则系统能继续正确有效的运行；反之亦然。类似还有软删除冲突（Soft Delete Conflict）[8]等。

1．2．5．1 冲突检测机制

由于移动计算网络环境的特殊性，要求数据的紧密一致性往往是不现实的，因此通常选用松散一致性来管理事务运行，即允许移动事务先局部提交，产生数据不一致暂态；而后全局冲突检测，重新收敛数据。这种情况下常见的冲突检测方法有：

- 基于版本号（时间戳）的检测方法[22]——该方法先求出两个事务读集和写集的交集，再比较交集中的数据版本号。如果发现有读项的时间戳则必有冲突存在；如果发现更新项的原始版本号与这个数据副本的版本号不匹配则也必有冲突存在。
- 读集和写集的比较检测方法——在一个数据副本上运行事务集时，如果读集或写集的初始化值与该数据副本的值不匹配，则该事务集与该数据副本上的局部事务集冲突。
- 语义冲突检测方法——该方法不同于普通的检测方法，它是利用特殊代码来检测数据的不一致性状态。

1．2．5．2 冲突解决机制

冲突检测机制发现更新冲突后，冲突解决机制根据具体情况利用事先预定原则和方法来处理。常见的原则有：

- 最近（或最早）事务优先
- 高优先级的事务优先（需要优先级评定系统的支持）
- 管理者（或某个特定的节点）传送的事务优先
- 特殊用户（或程序）执行的事务优先
- 最大（或最小）价值的事务优先

本文将在以下的章节中结合数据访问技术、SyncML 协议和 Sync4j，并联系具体的工程应用——民航移动调度子系统对上述的概念和重点问题加以讨论。

1．3 论文的组织结构

论文首先介绍了移动计算环境的形成过程，并对数据同步的相关概念加以阐述，重点突出了狭义的数据同步概念和广义的数据同步概念的区别；然后简介了

数据同步技术的两大难点：异构数据的映射问题和冲突检测机制的实现。

第二章则以数据访问技术为主要内容讨论了对异构数据库的数据操作实现。分别介绍了 Microsoft、Borland 公司的数据库访问产品和基于 Java 应用的 JDBC 技术，评价了它们各自的优势和不足之处，重点分析了在 XML 基础上的异构数据映射解决方案和在冲突处理上的先天缺陷。为孕育出一个更为优化数据库间异构数同步方案提供了思路。

第三章在分析现有商业数据库管理系统所采用的数据同步控制技术的基础上，引出了数据同步的准工业界标准——SyncML 协议，并着重介绍了 SyncML 协议的两大核心内容：XML 级应用和 Agent 技术；然后通过对一个实例的剖析提出了基于 SyncML 协议的异构数据库间数据同步架构（HDBS）；最后在肯定了 SyncML 协议能较好解决异构数据映射问题的同时，也指出了它在冲突处理机制上的薄弱之处。

第四章我们在分别介绍了移动数据库技术和主存数据库技术，将上一章中异构数据库间的数据同步架构具体到移动数据库和主存数据库上，提出了移动数据库与主存数据库间的异构数据同步快速模型（MMQS）。其优势在于充分体现了“在低端加强移动性，在高端提供高性能”的特点。我们利用 SyncML 协议的具体产品——Sync4j，将研究重点放在异构数据映射和冲突处理机制的建立上，同时还讨论了主存数据库引擎的研制方案。

第五章分析民航移动调度的用户需求，拟采用的技术和工作环境，将 MMQS 模型分为同构实现和异构实现两个阶段在该子系统中得到了具体工程应用。

第六章总结论文，并从移动 Agent 技术的应用、异构数据同步中间件和异构移动数据库间的数据同步技术等方面讨论了下一步的研究方向和内容。

第二章 基于数据访问技术的异构数据库间数据操作实现

异构数据库系统是相关的多个数据库系统的集合，可以实现数据的共享和透明访问，每个数据库系统在加入异构数据库系统之前本身就已经存在，拥有自己的 DMBS。异构数据库的各个组成部分具有自身的自治性，实现数据共享的同时，每个数据库系统仍保有自己的应用特性、完整性控制和安全性控制。

异构数据库系统的异构性主要体现：计算机体系结构的异构、基础操作系统的异构和 DMBS 本身的异构。我们重点解决的是 DMBS 本身的异构即可以是同为关系型数据库系统的 Oracle、 SQL Server 等，也可以是不同数据模型的数据库，如关系、模式、层次、网络、面向对象，函数型数据库共同组成一个异构数据库系统。其两大难题：数据库转换和数据透明访问均由数据访问技术来解决。

数据访问技术（DAT）是一种用于访问数据库的统一界面标准，是应用程序和数据库系统之间的中间件[10]。它通过使用与所需数据库对应的驱动程序和应用程序的交互来实现对数据库的访问，避免了在应用程序中直接调用与具体数据库相关的操作，从而提供了数据库的独立性。它的出现解决了程序员以往必须花费很多时间和成本撰写程序代码来访问不同数据库这一难题。随着数据库系统（DBS）的发展；对应的数据类型不断增加，数据访问技术也日趋复杂，并逐渐成为 C/S、B/S 架构（参见图 1）中的核心技术之一。我们将通过对几个常见数据访问技术[24]的介绍来分析异构数据库间的数据操作实现。

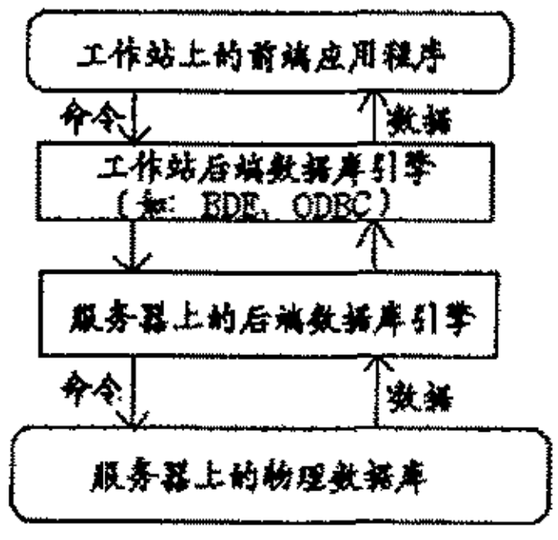

图 1 数据访问技术的常见架构

2．1Microsoft 的数据访问技术

Microsoft 针对 WINDOWS 平台提供了多种数据访问技术。从最早的 ODBC，到 DAO，到 RDO，再到 OLE DB 和 ADO，最后是 ADO.NET。这些技术都有着自身的历史特点和优势，分析其演变过程有助于我们对异构数据同步技术的认识。

2．1．1 ODBC 技术

ODBC 是 Open Database Connect 即开放数据库互连的简称，它是由 Microsoft 公司于 1991 年提出，并于 1992 年和 SYBASE、DIGITAL 共同制定的一个用于访问数据库的统一界面标准。由于获得了绝大多数据库厂商和第三方厂商的支持而成为了一个实际的标准数据存取技术，即使到现在仍广泛使用。

ODBC 主要由驱动程序和驱动程序管理器组成。驱动程序是一个用以支持 ODBC 函数调用的模块，每个驱动程序对应于相应的数据库，当应用程序从基于一个数据库系统移植到另一个时，只需更改应用程序中由 ODBC 管理程序设定的与相应数据库系统对应的别名即可。驱动程序管理器可链接到所有 ODBC 应用程序中，它负责管理应用程序中 ODBC 函数与 DLL 中函数的绑定。

ODBC 使用层次的方法来管理数据库，在数据库通信结构的每一层，对可能出现依赖数据库产品自身特性的地方，ODBC 都引入一个公共接口以解决潜在的不一致性，从而很好地解决了基于数据库系统应用程序的相对独立性，这也是 ODBC 一经推出就获得巨大成功的重要原因之一。

从结构上分，ODBC 分为单束式和多束式两类[23]。

- 单束式驱动程序

单束式驱动程序介于应用程序和数据库之间，像中介驱动程序一样数据提供一个统一的数据访问方式。当用户进行数据库操作时，应用程序传递一个 ODBC 函数调用给 ODBC 驱动程序管理器，由 ODBC API 判断该调用是由它直接处理并将结果返回还是送交驱动程序执行并将结果返回。由上可见，单束式驱动程序本身是一个数据库引擎，由它直接可完成对数据库的操作，尽管该数据库可能位于网络的任何地方。

- 多束式驱动程序

多束式驱动程序负责在数据库引擎和客户应用程序之间传送命令和数据，它本身并不执行数据处理操作而用于远程操作的网络通信协议的一个界面。前端应用程序提出对数据库处理的请求，该请求转给 ODBC 驱动程序管理器，驱动程序管理器依据请求的情况，就地完成或传给多束驱动程序，多束式驱动程序将请求翻译为特定厂家的数据库通信接口（如 Oracle 的 SQLNet）所能理解的形式并交于接口去处理，接口把请求经网络传送给服务器上的数据引擎，服务器处理完后把结果发回给数据库通信接口，数据库接口将结果传给多束式 ODBC 驱动程序，再由驱动程序将结果传给应用程序。

ODBC 在提供统一、方便访问关系数据库界面的同时也存在着不少的缺陷：

- 只支持关系数据库
- 只提供支持 C/C++的 API 函数，且非常复杂，不易掌握

2．1．2 OLE DB 技术

OLE DB 是一套通过 COM 接口访问数据的 ActiveX 接口。它的存在为用户提供了一种统一的方法来访问所有不同种类的数据源，这是针对 ODBC 仅支持关系数据库的调整；同时 OLE DB 可以在不同的数据源中进行转换。利用 OLE DB，客户端的开发人员在进行数据访问时只需把精力集中在很少的一些细节上，而不必弄懂大量不同数据库的访问协议。其架构图如下所示：

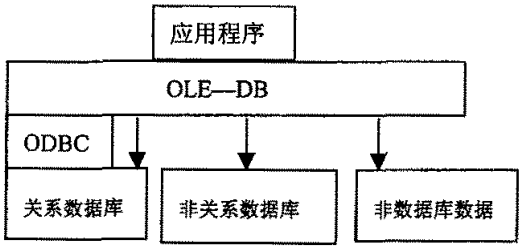

图 2OLE DB 架构图

OLE DB 仍有不少缺陷：

- 同 COM 的多数结构一样，OLE DB 的开发人员需要实现很多的接口
- 过于底层化，难于应用实现
- 不能被 VB、VBScript 等高级编程工具访问

2．1．3 ADO 技术

ADO（ActiveX Data Objects）即 ACTIVEX 数据对象是基于组件的数据库编程接口，它是一个和编程语言无关的 COM 组件系统。它是 Microsoft 以 COM 技术封装 OLE—DB 为 ADO 对象，其主要目的是在保持其支持异构数据互操作的前提下大幅度提高可用性。其架构调整如下所示：

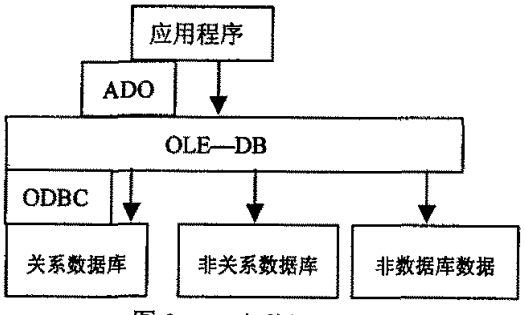

图 3 ADO 架构图

2．1．4 ADO.NET

ADO.NET 是.NET 应用程序的数据访问模型。它能用于访问关系型数据库系

统，如 SQL Server 2000，及很多其它已经配备了 OLE DB 供应器的数据源。在某种程度上，ADO.NET 代表了最新版本的 ADO 技术。然而，ADO.NET 引入了一些重大变化和革新，它们专门用于结构松散的、本质非链接的 Web 应用程序。

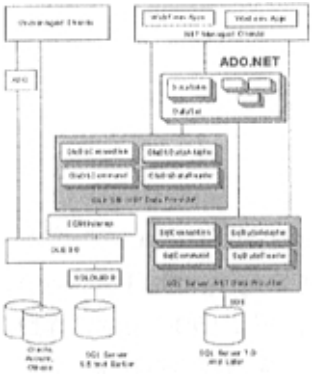

图 4ADO.NET 架构图

2．2Borland 的数据访问技术

2．2．IBDE 技术

BDE（Borland Data Engine）一直是 Borland 自行研制的存取数据引擎。1995 年推出时，其功能和执行速度都较当时 Microsoft 的 ODBC 好。但随着 Microsoft 凭借其在操作系统上的绝对优势不断地改善数据存取技术和推出新的数据存取标准。BDE 已逐渐没落，风光不再。

BDE 架构的不足：

- 需独立于操作系统的数据通道来实现对数据源的访问
- 驱动程序设置冗余且要大量手工配置

2．2．2 dbExpress

Borland 的 dbExpress 是为很多数据库提供公共 API 的一种新的处理方法。dbExpress 的设计目标为：最小化系统资源的占用，最大化运行速度，提供平台独立性，提供易用的开发环境，使驱动程序的开发简易。

dbExpress 的驱动程序既小又快，因为它们只提供了非常有限的功能。一个 dbExpress 驱动程序实现五个接口，支持获取元数据（metadata），执行 SQL 语句和存储过程和返回只读的单向游标结果集等等。不管怎样，当被使用于 DataSetProvider 和 ClientDataSet 或 SQLClientDataSet 以实现 Borland 的提供/处理

数据访问策略时，dbExpress 能提供全性能，高可靠性，高并发性的系统，很方便地处理 SQL 数据库中的数据。

dbExpress 作为一个新的技术优势明显：

- 缩短事务周期
- 瞬时排序和搜索
- 自动统计信息
- 多并发数据视图

2．3JDBC 技术

JDBC（Java Database Connectivity Java 数据库连接）是一种可用于执行 SQL 语句的 JavaAPI（Application Programming Interface 应用程序设计接口）。它由一些 Java 语言编写的类和界面组成。JDBC 为数据库应用开发人员、数据库前台工具开发人员提供了一种标准的应用程序设计接口，使开发人员可以用纯 Java 语言编写完整的数据库应用程序。

通过使用 JDBC，开发人员可以很方便地将 SQL 语句传送给几乎任何一种数据库。也就是说，开发人员可以不必写一个程序访问 Sybase，写另一个程序访问 Oracle，再写一个程序访问 Microsoft 的 SQLServer。用 JDBC 写的程序能够自动地将 SQL 语句传送给相应的数据库管理系统（DBMS）。不但如此，使用 Java 编写的应用程序可以在任何支持 Java 的平台上运行，不必在不同的平台上编写不同的应用。Java 和 JDBC 的结合可以让开发人员在开发数据库应用时真正实现“WriteOnce，RunEverywhere！”

Java 具有健壮、安全、易用等特性，而且支持自动网上下载，本质上是一种很好的数据库应用的编程语言。它所需要的是 Java 应用如何同各种各样的数据库连接，JDBC 正是实现这种连接的关键。

JDBC 扩展了 Java 的能力，如使用 Java 和 JDBCAPI 就可以公布一个 Web 页，页中带有能访问远端数据库的 Applet。或者企业可以通过 JDBC 让全部的职工（他们可以使用不同的操作系统，如 Windwos，Machintosh 和 UNIX）在 Intranet 上连接到几个全球数据库上，而这几个全球数据库可以是不相同的。随着越来越多的程序开发人员使用 Java 语言，对 Java 访问数据库易操作性的需求越来越强烈。而 JDBC 这种跨平台特性和连接异构数据库的特点正是我们讨论异构数据同步技术的基础和起点。

到目前为止，微软的 ODBC 可能是用得最广泛的访问关系数据库的 API。它提供了连接几乎任何一种平台、任何一种数据库的能力。但 JDBC 的出现有着 ODBC 难以比拟的优势：

- ODBC 并不适合在 Java 中直接使用。ODBC 是一个 C 语言实现的 API，从 Java 程序调用本地的 C 程序会带来一系列类似安全性、完整性、健壮性的缺点。
- 其次，完全精确地实现从 C 代码 ODBC 到 JavaAPI 写的 ODBC 的翻译也并不令人满意。比如，Java 没有指针，而 ODBC 中大量地使用了指针，包括极易出错的空指针“void＊”。因此，对 Java 程序员来说，把 JDBC 设想成将 ODBC 转换成面向对象的 API 是很自然的
- ODBC 并不容易学习，它将简单特性和复杂特性混杂在一起，甚至对非常简单的查询都有复杂的选项。而 JDBC 刚好相反，它保持了简单事物的简单性，但又允许复杂的特性。
- JDBC 这样的 JavaAPI 对于纯 Java 方案来说是必须的。当使用 ODBC 时，人们必须在每一台客户机上安装 ODBC 驱动器和驱动管理器。如果 JDBC 驱动器是完全用 Java 语言实现的话，那么 JDBC 的代码就可以自动的下载和安装，并保证其安全性，而且，这将适应任何 Java 平台，从网络计算机 NC 到大型主机 Mainframe。

总而言之，JDBCAPI 是能体现 SQL 最基本抽象概念的、最直接的 Java 接口。它建构在 ODBC 的基础上，因此，熟悉 ODBC 的程序员将发现学习 JDBC 非常容易。JDBC 保持了 ODBC 的基本设计特征。实际上，这两种接口都是基于 X / OPENSQL 的调用级接口（CLI）。它们的最大的不同是 JDBC 是基于 Java 的风格和优点，并强化了 Java 的风格和优点。

当然，微软除了 ODBC 以外还有其它 API，如 RDO，ADO 和 OLEDB。这些 API 事实上在很多方面上同 JDBC 一样朝着相同的方向努力，也就是努力成为一个面向对象的，基于 ODBC 的类接口。然而，这些接口目前并不能代替 ODBC，尤其在 ODBC 驱动器已经在市场完全形成的时候，更重要的是它们只是 ODBC 的“漂亮的包装”。

2．4 基于数据访问技术术的异构数据库间数据互操作模型

2．4．1 模型简介

通过对上述数据访问技术的简介和分析，我们能提炼出基于数据访问技术的异构数据库间数据互操作的模型框架（如图 5 所示），并以 ADO 为例实现对该模型的模拟。

该模型与异构连接引擎[25]的工作原理类似，以数据访问技术为中间层分别联系应用程序和对应 DBMS 的驱动程序。相对应用程序而言利用统一的 SQL（Structured Query Language）来操纵数据对象（如元数据），再交由数据访问

层将事务语句或结果集转化为驱动程序所能识别的格式，从而完成异构数据库间的数据互操作。接下来，我们将从异构数据的映射和冲突处理的实现两方面来详细讨论该模型。

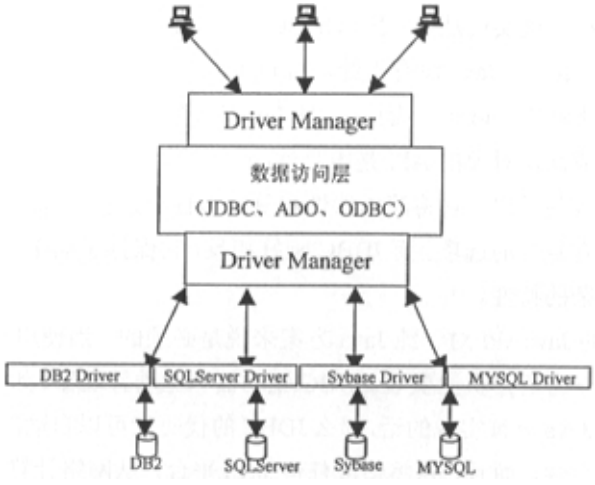

图 5 基于数据访问技术的异构数据库间数据互操作的模型框架

2．4．2 基于数据访问技术的异构数据映射

利用数据访问技术能一定程度上解决异构数据间的映射问题，但要有前提条件。如需要双方数据源的驱动程序；或双方在映射之前定义共同的 SQL 数据类型或 XML 文档作为中介等等。这些前提条件使得基于数据访问技术的异构数据映射问题的解决方案复杂化、底层化、低效率且不具有通用性，但针对这些方案的研究和讨论对我们提出更优的解决方法有很大的帮助。

2．4．2．1 转换 DLL

由于应用程序和结果集常常以不同的字符集存储数据，因此 ODBC 提供一种普通的手段允许驱动程序在不同的字符集之间转换数据。转换的内容包括所有的 SQL 语句、字符参数、字符元数据以及错误信息。这种方法由一个执行转换函数（SQLDataSourceToDriver 和 SQLDriverToDataSource）的 DLL、数据源以及转换目标的驱动程序共同组成。

应用程序通过 SQLSetConnectAttr 请求转换，同时在 SQLSetConnectAttr 中指定与 ConnectionHandle 相关的驱动程序，调用指定的 DLL，完成从数据源到驱动程序的所用数据流转换。

很显然，转换 DLL 这种方法有着诸多的不足：直接与驱动程序打交道使应用过于底层化；必须有第三方驱动程序的支持；无法同时映射多个转换目标，难于实现一次转换多方受益的要求。

2．4．2．2SQL 数据类型

SQL 数据类型是按照 SQL-92 标准由每个 DBMS 定义的。这就为异构数据间映射问题提供了解决途径—从数据源返回数据之前，驱动程序将 SQL 数据类型转换为指定的数据类型；在发送数据到数据源之前，驱动程序又将特定的数据类型转换为 SQL 数据类型。

可以看到利用 SQL 数据类型作为中介完成异构数据的映射是可行的，但仍有不足之处：离不开驱动程序的支持；必须要求驱动程序符合 SQL-92 级；难于操作（这个问题已在 ADO 和 ADO.NET 利用组件技术解决了）；局限于关系数据库的范畴，还不是真正意义上的异构数据映射解决方案。但针对这些问题的思考和探索，为 XML 的出现提供了契机。

2．4．2．3XML 文档

XML 是 World Wide Web Consortium 制定的作为 Internet 上数据交换和表示的标准语言，是一种允许用户定义自己的标记语言的元语言。它是从 SGML（Standard Generalized Markup Language）发展而来的，保留了 SGML 中大约 80%的功能，但是大大减少了 SGML 的复杂性。使用它可以描述非常复杂的 Web 页面，如复杂的数学公式、化学分子式等。

XML 语言的主要特征体现在 4 个方面，即结构化、自描述性、可扩展性和浏览器自适应性。

- 结构化：XML 是一个极端标准化的语言规范，它一般利用 DTD（Document Type Definition）规范来定义 XML 文件的语法、句法和数据结构。DTD 用来定义在文档中声明的元素是必需的、可选的还是有条件的，元素的属性值是否有限制，是否有缺省值或允许有空标记等。这种严格结构化的好处在于用户可以很容易将文档中的数据映射到数据结构或分级对象数据结构中，使得在应用系统和数据库之间的传输数据非常可靠，也让用户可以使用结构化的 XML 文件在不同种类的数据库之间进行数据传输。
- 自描述性：在 XML 文档中，自描述性是可选的，但使用自描述性可以增强 Web 的检索功能。自描述的数据称为元数据，用来描述有关整个文档的信息，如阅读范围、文件内容、文件写作语言、作者以及这个文件的其他任何信息。自描述数据的存在可以增强 Web 的检索和导航功能。
- 可扩展性：XML 的一个中心特性就是体现在它的可扩展性中。在 XML 中，标签(Tag) 是 DTD 定义的，标签定义了文档中的数据属性。用户可以自己定义标签，表示自己定义的数据和属性。
- 浏览器自适应性。一个结构化的文档能够适应不同的浏览模式。XML 在 Web 浏览器上得到了广泛的支持。微软在 Internet Explorer 5.0 中已经包含了对

XML 标准的支持。Netscape 公司在 1999 年 7 月发行的 Netscape Navigator 5.0 版的核心引擎 Gecko 中，也全面支持 XML 文档。

利用 XML 的这些特性，现有的 ADO.NET 和部分 JDBC 等数据访问技术均实现了平台互用性和可伸缩的数据访问。于是通常数据访问方法改变为：先从发送者端提取一个记录集，把它保存为 XML 格式，然后传输结果数据流，让接收者端从这个 XML 数据流重新构造出记录集供以后使用。这种方法是现阶段处理异构数据映射问题的理想方案。

2．4．3 基于数据访问技术的冲突处理

从图 5 可以看出，数据访问技术层位于 DBMS 之上。由于它的主要目的是实现访问数据库的统一界面，因此将冲突处理放在具体的 DBMS 中去实现，自身在这一问题上并无建树。关于具体的 DBMS 冲突处理机制（如：两阶段分锁、多锁系统、分布式加锁等加锁机制、基于时间戳的并发控制、基于有效性确认的并发控制、两阶段提交协议等等）在此就不一一介绍了。

2．4．4 利用 ADO 实现异构数据的互导

本实验是参照基于数据访问技术术的异构数据库间数据互操作模型，利用 ADO 来实现异构数据库间数据的互导。我们首先用 ADO 组件提取出源数据（SoureDB）的数据项集，然后再打开目标数据源（TargetDB），依次读入源数据的每一个数据项值并写入到目标数据源中。通过两个 ADO 对象（分别对应不同的数据源）在组件级的互操作实现异构数据的互导。核心代码为：

```
Set TargetDB = New ADODB.Recordset
Set TargetDB.ActiveConnection = objconn
TargetDB.LockType = adLockOptimistic
TargetDB.CursorType = adOpenKeyset
TargetDB.Source = "tablename"注："tablename" 数据源中的表名。
TargetDB.Open

Set SoureDB = New ADODB.Connection
cjconn.Open "servername", "username", "userpassword"
注："servername"，" username"，" userpassword" 含义同上，只不过数据源是其他数据
库。
Set SoureDB = New ADODB.Recordset
Set SoureDB.ActiveConnection = cjconn
SoureDB.Source = "tablename"
```

注：“tablename” 含义同上，该表中的字段类型应与上面的表中的字段类型一致

```
SoureDB.Open
i = 0
Do While Not SoureDB.EOF
    i = i + 1
    TargetDB.AddNew
    {…读入源数据的数据项}
    TargetDB.Update
    SoureDB.MoveNext
    Debug.Print I
    注：“I”用于显示已导入了多少条记录。
Loop
End Sub
```

2．4．5 利用 DataSet 和 XML 实现异构数据的映射

在 ADO.NET 组件中 DataSet 是其核心组件之一，它提供了独立于数据源的数据访问，为了实现这种平台互用性和可伸缩的数据访问，ADO.NET 采用了基于 XML 数据的传输格式，XML 在这里充当了至关重要的中介脚色。当数据传输时，ADO.NET 是将 DataSet 表述为 XML，然后以 XML 格式传递给其他组件，当然接收数据的组件不一定就是 ADO.NET 组件，它可以是任何可以处理 XML 数据的组件。这样利用自定义的 XML 数据流就实现了异构数据的映射。

图 6 阐述了.NET 数据提供程序、DataSet、XML 之间的关系：

由 DataSet 数据转换为 XML 数据：要解决这一问题，需要导入 Data、Xml、Data SqlClient 、IO 名称空间。我们先创建 DataSet 组件对象，通过创建 CreateDataSet（）方法返回类型为 DataSet，这样就可以在别的方法中直接调用该方法，取得 DataSet。

.NET Framework 提供了操作 XML 的强大支持，在.NET Framework 中，DataSet 和 XML 文档是相同数据的不同视图。ADO.NET 对 XML 的支持为异构数据的映射提供了解决方案，改变了典型分布式系统的面貌。

2．4．6 利用 JDBC 和 XML 实现异构数据的映射

把 JDBC 返回的结果集(ResultSet)转化为 XML 格式的文本，并存放在字符串（String）中作为返回结果传送到其他的应用（系统）程序。这种方法的好处在于它与涉及到的数据库结构无关，能较好地在异构平台上解决异构数据的映射问题。其相关代码为：

```
import java.sql.*;
import java.io.*;
public class XMLWriter {
/**
* @param ResultSet rs 输入的结果集
* @return String 返回 XML 串
* @exception SQLException
*/
  public static String generateXML(final ResultSet rs) throws SQLException {
    final StringBuffer buffer = new StringBuffer(1024 * 4);
    if (rs == null) return "";
    //if (!rs.next()) return "";
    buffer.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n"); //XML 的头部
信息
    buffer.append("<ResultSet>\n");
    ResultSetMetaData rsmd = rs.getMetaData(); //得到结果集的定义结构
    int colCount = rsmd.getColumnCount(); //得到列的总数
    for (int id = 0; rs.next(); id++) { // 对放回的全部数据逐一处理
    //格式为 row id , col name、col context
      buffer.append("\t<row id=\"").append(id).append("\">\n");
    for (int i = 1; i <= colCount; i++) {
      int type = rsmd.getColumnType(i); //获取字段类型
      buffer.append("\t\t<col name=\"" + rsmd.getColumnName(i) + "\">");
      buffer.append(getValue(rs, i, type));
      buffer.append("</col>\n");
}
```

```
      buffer.append("\t</row>\n");
    }
    buffer.append("</ResultSet>");
    rs.close();
    return buffer.toString();
  }
/**
* This method gets the value of the specified column
* 通用的读取结果集某一列的值并转化为 String 表达
* @param ResultSet rs 输入的纪录集
* @param int colNum 第几列
* @param int type 数据类型
*/
  private static String getValue(final ResultSet rs, int colNum, int type) throws
SQLException {
    switch (type) {
      case Types.ARRAY:
      case Types.BLOB:
      case Types.CLOB:
      case Types.DISTINCT:
      case Types.LONGVARBINARY:
      case Types.VARBINARY:
      case Types.BINARY:
      case Types.REF:
      case Types.STRUCT:
      return "undefined";
      default: {
        Object value = rs.getObject(colNum);
        if (rs.wasNull() || (value == null))
          return ("null");
        else
          return (value.toString());
      }
    }
```

```
}
//测试例程
public static void main (String args []) throws SQLException, IOException{
  String user="test";
  String password="test";
  String url = "jdbc:mysql://localhost:3306/test";
  Connection conn=null;
    Statement stmt=null;
try{
  // Load the Mysql JDBC driver
  Class.forName("org.gjt.mm.mysql.Driver");
  conn = DriverManager.getConnection (url,user,password);

  System.out.println ("database connected successful.");

  // Create a statement
  stmt = conn.createStatement ();
  dma =conn.getMetaData();//数据库元数据 dma
}
catch (ClassNotFoundException ex) {
        System.err.println("Cannot find the database driver classes.");
        System.err.println(ex);
}
catch (SQLException ex) {
   System.err.println("Cannot connect to this database.");
   System.err.println(ex);
}
// Do the SQL "Hello World" thing
System.out.println("SQL Command: select * from food");
System.out.println("here is the food_table rows view");
//得到表 food 的所有列并转换成元数据
ResultSet rset = dma.getColumns(null,null,"food",null);
ResultSetMetaData rsmd = rset.getMetaData();
int numCols = rsmd.getColumnCount();
```

```
//打印 food 表的所有列名及列值
while (rset.next()){
  System.out.print(rset.getString("COLUMN_NAME")+" ");
}
System.out.println();
rset.close();
rset = stmt.executeQuery ("select * from food");
while (rset.next()) {
  System.out.print ("   "+rset.getString (1)+"    "); //W?username
  System.out.println (rset.getString (2)); //W?password
};
// call toxml function
System.out.println("here is the xml output");
rset.close();
rset = stmt.executeQuery ("select * from food");
String xmlstring =generateXML(rset);
System.out.println(xmlstring);
rset.close();
    stmt.close();
    conn.close();

  }
  private static DatabaseMetaData dma; }
```

2．5 小结

通过对模型的分析，可以看到受数据访问技术的限制，模型存在着诸多不足之处：

- 缺乏一个用于数据同步的统一标准中介媒体格式。
- 能实现互操作的异构数据类型有限，不能广泛地支持现有的数据类型。
- 以单向操作为主，不能很好地支持双向同步操作。
- 不能满足快速、频繁的同步需求。
- 没有更高层次上即脱离具体 DBMS 的限制（如冲突处理机制）。
- 难于适应移动计算环境下的应用。

这些不足都将在我们第三章和第四章的同步模型中得到解决。

第三章 基于 SyncML 协议的异构数据库间数据同步模型

随着移动计算领域的不断延伸，基于商业应用的各种设备、技术也在不断推陈出新。虽然每一种设备或技术的出现仅局限于特定的应用范围，但都将移动计算领域的研究推进了一步，数据同步技术也是如此。研究现有的商业数据同步技术，对我们提出具有通用性的异构数据库间同步架构有重要作用。

3．1 商业数据同步技术的现状

广泛应用在商业中的数据同步技术主要分为两大类型：同步中间件和分布式数据库采用的数据同步方法。

3．1．1 同步中间件（Sync Middleware）[9]

Synchronization middleware 是一种与移动设备交换数据的软件工具，它以中间件的方式联接了服务端和移动端（如图 7 所示），并提供同步化平台来保证移动或无线设备获得最新的商业数据、软件、Email 和文档。具体实现包括以下几大部分：

- 移动节点上的数据结构
- 移动数据的定义工具
- 移动数据的访问方法
- 保持一致性的方法
- 移动数据的恢复方法

同步中间件的出现，为移动计算环境下 C/S 程序设计提供了统一的数据同步接口，降低了开发难度；通过数据更新的可调整策略，动态地控制数据同步的信息量，以适应脆弱多变的网络环境；同时加强了网络数据的安全性。从表一中可以看出，尽管同步中间件力图实现工作在多种网络环境下，同步不同的数据类型，但仅靠一两家公司的产品是无法完成的。它需要一个能支持绝大多数网络协议，统一网络数据流格式的标准。只有构建与这一标准之上的同步中间件才能真正在不同的设备之间以及异构的网络环境下实现数据的同步。

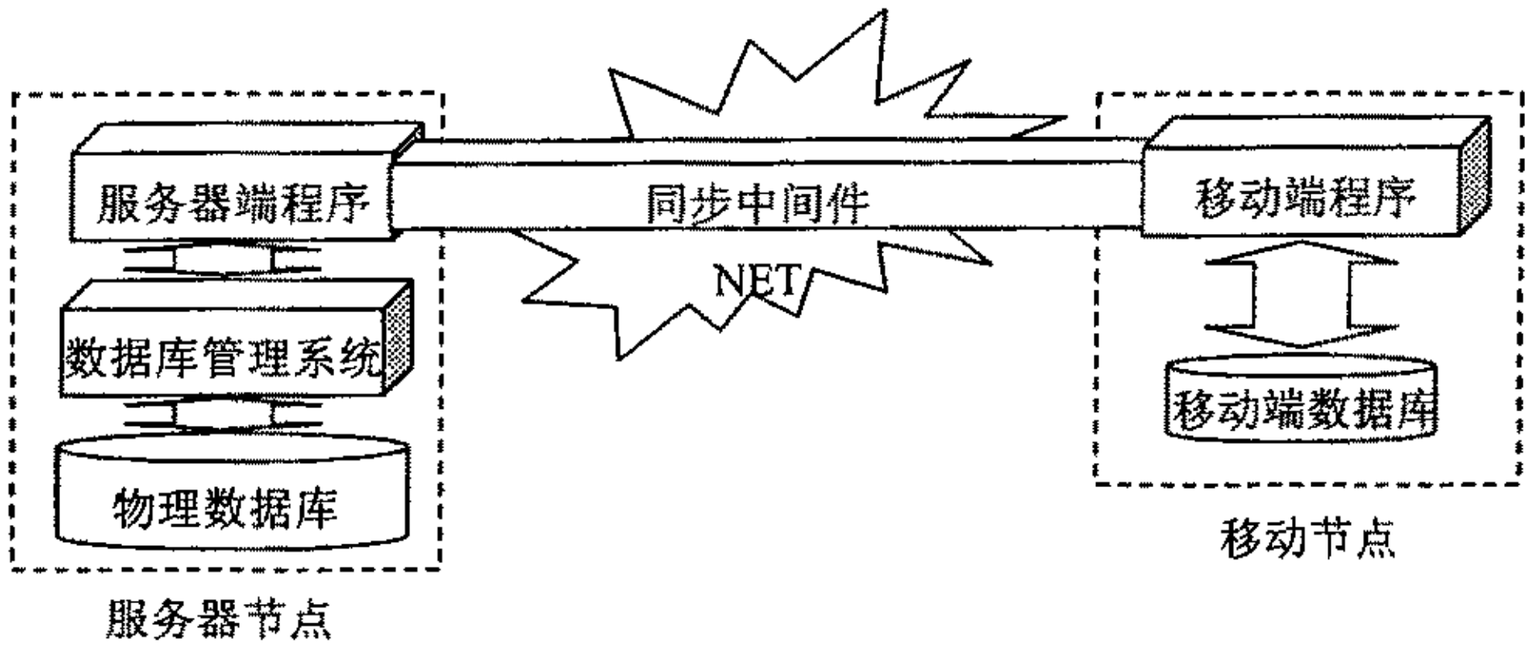

图 7 同步中间件

表 1 商业的同步中间件产品

厂商	产品	支持的网络环境	应用类型
Pumatech	Sync-it intellisync	-direct -LAN,IP -mobile	-PIM
Syclo	SIM	-LAN -mobile	-PIM -business
fusionOne	internetSync	-LAN,IP -mobile	-PIM
Aether	ScoutSync	-LAN,IP -mobile	-PIM -business

3．1．2 商业 DBMS 采用的数据同步技术

目前，主流的商业 DBMS（Oracle、Sybase、DB2、SQLServer 等）在移动计算环境下，针对分布式系统的实际需求：

- 并不是所有系统都要求所有节点均可访问数据全集
- 并不是所有系统都要求每时每刻保持数据在各节点的一致性

提出了自己的移动数据同步方案。从实现的技术手段上看有：同步技术、复制技术和刷新技术；所适用的网络类型：高速局域/广域网、较低速率的 Dial-up 、无线、间接连接(email, ftp)和 Internet (HTTP)；采用的系统拓扑结构：对等（Peer-to-peer）和级联/树状（Hierarchical）。对应的产品更是种类繁多。我们选取主流 DBMS 的产品来加以分析：

3．1．2．1Oracle

表 2　Oracle 9i and Oracle9i Lite

对称性	Asymmetric:Master(Oracle9i) and snapshot(Lite)
方向性	(1) one-way:read-only sapshot (2) two-way:updateble snapshot
架构	Master site(Oracle9i) and snapshot site(Oracle9i Lite)
更新模式	(1) Lazy master(read-only snapshot) (2) Lazy replica(updatable snapshot limited to one table)
刷新模式	(1) Snapshot pull,differential and full (2) In updatable snapshot: push to master
一致性	(1) Serializability in master (2) Snapshot consistency in snapshots
冲突检测	Comparing row values
冲突调解	Done in Master:10 pre-defined methods,PL/SQL proc
控制工具	Master:with SQL;snapshot:with OLE functions

3．1．2．2Sybase

表 3　Sybase Adaptive Server Aynwhere（ASA）and UltraLite

对称性	Asymmetric:master(ASAi)and replica(UltraLite)
方向性	(1) one-way:read-only snapshot (2) two-way:updateable replica
架构	Master site(ASA), replica site(UltraLite) and MobiLink Server
更新模式	(3) (1) Lazy master(read-only snapshot) (4) Lazy replica(updatable snapshot limited to one table)
刷新模式	(1) Replica pull,differential and full (2) In updatable snapshot: push to master
一致性	(3) (1) Serializability in master (2)Snapshot consistency in replica
冲突检测	Comparing row values
冲突调解	Done in Master
控制工具	With extended SQL and C functions

3．1．2．3IBM

表 4　IBM DB2 and DB2Everyplace

对称性	Asymmetric;master(DB2)and replica(DB2)
方向性	(3) one-way:read-only snapshot(publication) (4) two-way:updateable replica (publication)
架构	DB2,DB2 Everyplace and DB2 Everyplace Sync Server
更新模式	(5) (1) Lazy master(read-only snapshot) (6) Lazy replica(updatable snapshot limited to one table)
刷新模式	(3) Replica pull,differential and full (4) In updateable replica: push to master
一致性	(4) (1) Serializability in master (2)Snapshot consistency in snapshots
冲突检测	Version-based
冲突调解	Done in Master:predefined methods
控制工具	SQL ,GUI tools

3．1．2．4Microsoft

表 5　Microsoft SQL Server and SQL Server for CE

对称性	Asymmetric:master(SQL Server)and replica(SQL S.for CE)
方向性	(5) one-way:read-only snapshot (6) two-way:merge replication
架构	Master site(SQL Server) and replica site(SQL S.for CE)
更新模式	(7) (1) Lazy master(read-only snapshot) (8) Lazy replica(updatable snapshot limited to one table) (9) Updateable result sets(with RDA)
刷新模式	(5) Replica pull,differential and full (6) Upload to master
一致性	(5) (1) Serializability in master (2)Snapshot consistency in snapshots
冲突检测	Comparing row values
冲突调解	Done in Master:programmable with procedures
控制工具	SQL and C functions

3．1．3 普遍存在的问题

- 出于市场竞争的需要，各大厂商不可能公开自己的数据格式。这样必然出现数据同步时同一厂商的产品易于同步，不同厂商的产品难以同步的现象即异构数据的映射问题仍未有效解决。但随着厂商对数据格式向 XML 格式转换的支持力度的不断加大，这一问题有望得到解决。
- 由于不同的产品采用了不同的同步手段，处理冲突的策略也不一样，势必造成异构数据同步时的混乱。这说明缺乏一种有效的、更高层次的专用协议支持。

3．2 基于 SyncML 协议的异构数据同步方案

虽然目前已经有了一些数据同步协议，而且各种新的协议还在不断产生，但它们都有很大的局限性，其中大多数只能支持有限种类的设备、系统及数据类型。这些互不兼容的协议增加了各方面工作的复杂度，而且这类协议的增加会限制移动设备的使用。正是基于此，当今世界的一些行业先锋（IBM、Lotus、Motorola、Nokia、Palm Inc、Psion、Starfish Software）才联合起来，致力于开发一种全行业通用的开放性的移动数据同步协议，并将其命名为 SyncML。

3．2．1SyncML 协议

3．2．1．1 协议简介

SyncML 是 SyncML Initiative 开发的同步标志语言（synchronization markup language）。作为通用同步协议它的目标是普遍适用，同步两端可以是任何一种类型、在任一种网络上面，这样会：

- 同步网络数据与任一种移动设备
- 同步移动设备与任一种网络数据[6]

要实现这个目标，协议需要下列特性：

- 可以有效的在有线和无线网络上运作
- 支持多种传输协议
- 支持任意的网络数据
- 可由多种应用程序访问
- 考虑到移动设备的资源限制
- 构造于现有的全球互联网和 Web 技术之上
- 协议最小功能就是赋予所有设备最常用的同步能力

SyncML 的典型应用是移动设备和网络服务之间的数据同步。除此之外，SyncML 还可用于对等的数据同步，如两台 PC 之间。SyncML 的同步过程是由

同步双方相互发送一系列消息来完成，消息的格式都在 SyncML 表示协议中做了具体定义。SyncML 的消息都是以 XML 格式的文档定义的，而 XML 是文本文档标记的工业标准，这必将极大地促进 SyncML 的推广和普及。我们将在 3.2.2 节对 SyncML 协议利用 XML 来解决异构数据的映射问题加以详细讨论。

利用 SyncML，网络数据可以通过多种设备同步化，用户可以使用不同的设备（包括掌上电脑、移动电话、汽车计算机、台式机等）访问和操作同一网络数据。此外，用户的个人信息（如电子邮件、通讯录等）能够在用户的不同设备上同时得到更新，并保持一致。比如，用户在掌上电脑上阅读了一封新收到的邮件，那么在他的台式机中，也会自动地将这封邮件显示为已读邮件。有了 SyncML，用户还可以在他的移动设备上更多地使用应用软件和信息，如果对这些软件和信息有所更新，可以随时将这些更新信息同步到办公室

3．2．1．2 SyncML 协议的体系结构和总体框架

SyncML 协议的体系结构如图 8、总体框架如图 9 所示。

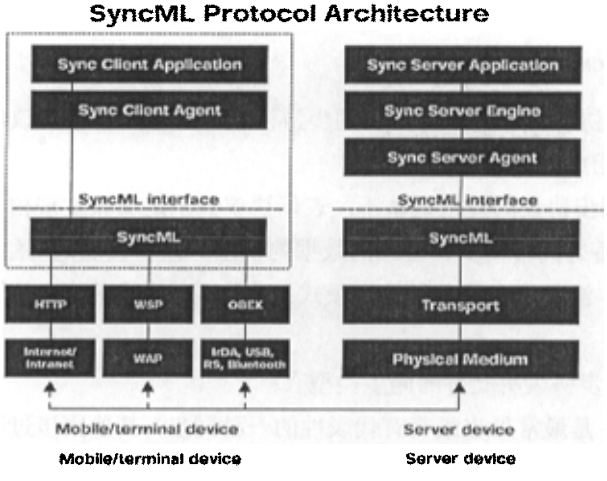

图 8 SyncML 体系结构

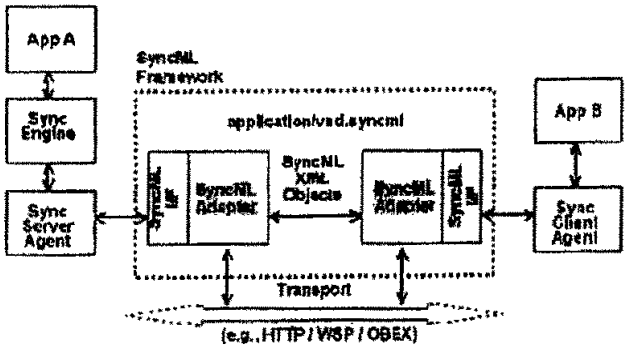

图 9 SyncML 协议框架

协议框架的要素：

- App A：是位于服务器端的应用，负责为其他应用提供数据同步服务。
- App B:是位于客户端的应用，请求并接受 App A 所提供的数据同步服务。
- Sync Engine：位于服务器端，负责在同步过程中分析数据集中所做的改动，检测和解决冲突。客户端也可以有部分类似的功能。
- Sync Server Agent 和 Sync Client Agent：调用在 SyncML 同步协议中提供的接口（SyncML I/F）与对方通信。
- SyncML I/F：即 SyncML Interface，是 SyncML 同步协议中提供的接口，它建立在各种底层通信协议之上，与应用无关，由 SyncML 行动组织定义和发行。

SyncML 客户机通常是移动电话、PDA、PC 等，而 SyncML 服务器可以是专门的服务器或 PC。一般情况下是由客户机首先把自己的更改信息发送给服务器，并等待服务器的应答，而服务器在接收到更改信息后要进行同步分析以检测和解决冲突，然后把处理结果和更改要求返回给客户机。但也有些情况是由服务器首先开始同步过程。

3．2．1．3SyncML 的协议内容

SyncML 的协议内容由 SyncML 同步协议和 SyncML 表示协议组成。二者一静一动，相辅相成。

SyncML 同步协议的提出是基于：表示协议并不足够来实现信息的交互性；需要在多种设备之间传输多种格式的数据的要求。该协议满足网络延迟的需要；满足各种设备；满足现有的各种储存模式；满足多种安全需求；能适应多种使用模式。

SyncML 同步协议所支持的同步类型有：

- 双向同步　是最常见也最具有代表性的一种同步，其他同步过程都和它有很多相似之处。双向同步总是由客户机首先发起，然后客户端和服务器互换更新信息。同步过程如下：同步初始化;客户端准备要发送的数据;客户端发送自己的更新信息；服务器接收并进行分析处理；服务器返回处理的状态信息和自己的更新信息；客户端接收更新信息并对自己的数据库进行更新；客户端把更新的状态信息发送给服务器，其中包含插入记录的 ID 映射信息；服务器对客户端发送的映射信息做应答；客户端收到服务器的应答后，把同步结果通知用户。
- 慢同步　是双向同步的一种。在一般的双向同步中，客户机只是把自己在上次同步过程中所做的修改发送给服务器，而在慢同步中，客户机则把自己所有的数据都发给服务器，服务器逐一比较接收到的数据和自己的数据，以确

定客户端哪些数据需要更新，然后把这些更新信息发送给客户端。需要慢同步的情况一般有：设备之间第一次同步；修改日志丢失；同步双方的同步标志不匹配。由于产生慢同步的原因很多，所以客户机和服务器双方都有可能发起慢同步。

- 客户端单向同步　只由客户机向服务器发送上次同步之后自己更新的信息，而服务器不把自己的更新信息发送给客户端。
- 客户端刷新同步　属于客户端单向同步的一种。客户机把自己所有的数据都发给服务器，并刷新服务器上的数据。
- 服务器单向同步　客户机从服务器那里得到上次同步之后服务器所有的更新信息，但不向服务器发送自己的更新信息。
- 服务器刷新同步　属于服务器单向同步的一种。服务器把所有的数据都发给客户端，并刷新客户端的数据。
- 服务器发起的同步　前面六种同步方式都是由客户机主动发起的同步过程，而这种同步方式是由服务器首先通知客户端，让其开始某种类型的同步。

SyncML 表示协议没有指定数据同步协议、同步引擎，而是指定了一些同步框架、格式使之适应不同的数据同步模型，同时还指定了很多同步操作的结果。其主要作用有：定义了一种格式；注册为 MIME（多用途网际邮件扩充协议）媒体格式；定义了独立的同步传输协议；可以适合于多种同步传输方式的需要；与后台数据存储无关；与同步对象类型分离。

SyncML 表示协议由以下三大部分组成：

- 一组事先定义好的消息格式，这些格式可以基于 XML 文档和 MIME（多用途网际邮件扩充协议）来制定，并为同步设备所共享。
- 提供同步的请求命令集和应答命令集（Add, alert, atomic, copy, delete, exec, get, map, put, replace, search, sequence, sync 和 Status,Results 等）。
- 同步操作后的状态值

3．1．2．4SyncML 同步传输实例

下面是一个双向传输例子，描述了同步传输的大体过程。同步传输有七种方式，下面的例子是最为复杂的一种。首先进行初始的准备工作，然后进行传输，再进行确认。具体代码请参见文献[8]。

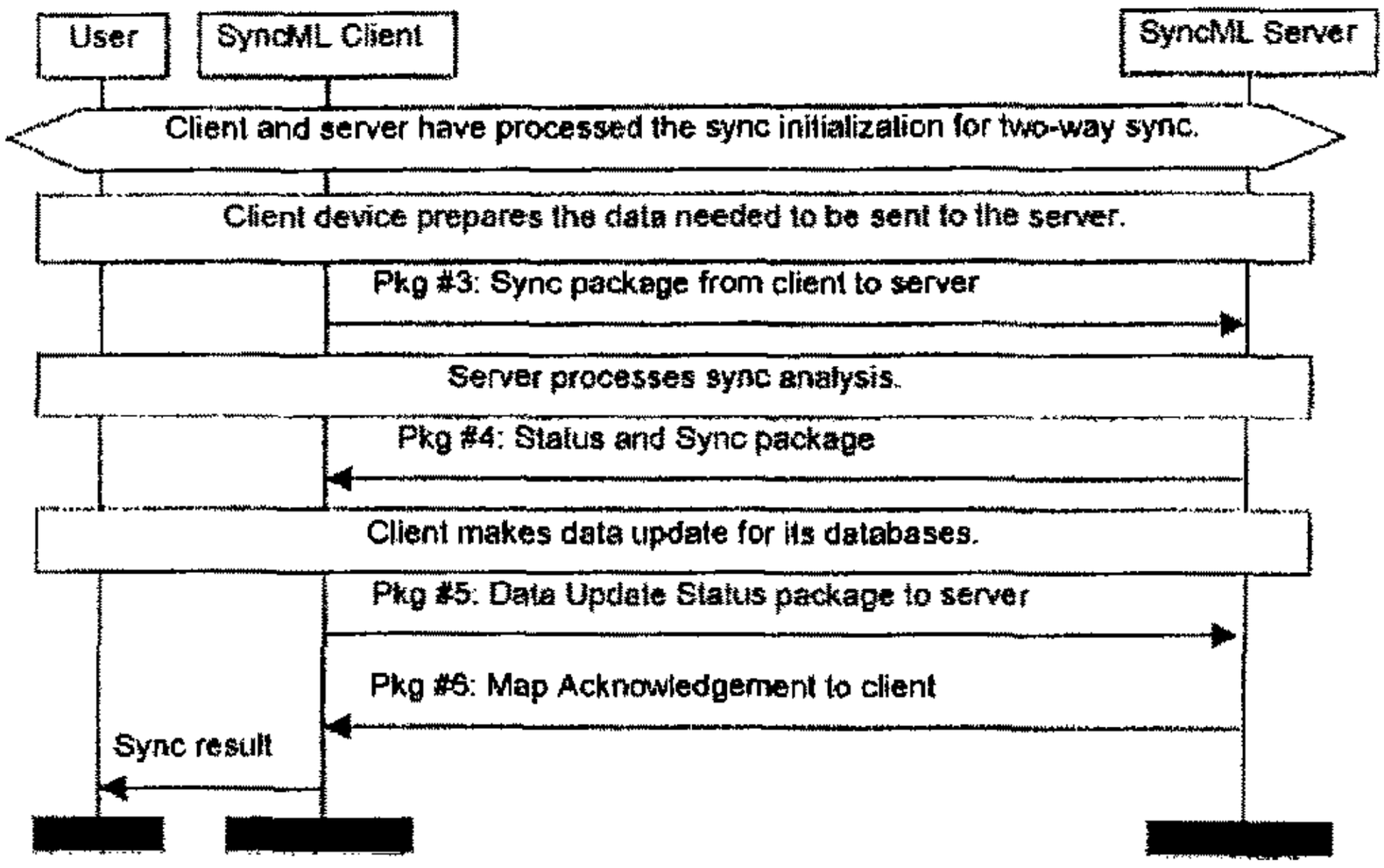

图 10 双向同步流程图

3．2．2SyncML 协议对传统数据同步技术作出的改进

SyncML 协议的提出将数据同步技术的发展向前推进了一大步。从图 8 图 9 中可以看出它支持众多的网络协议，可以同步任意的网络数据，能实现多网络数据库的并发同步。针对现有数据同步技术在异构数据映射和冲突处理机制上的不足，SyncML 协议从更高的层次来加以实现。它所依靠的 XML 技术、Agent 技术和同步引擎技术，就是解决上述问题的突破口。

3．2．2．1XML 应用

SynML 表示协议实质就是一个 XML 的应用。它定义了 SyncML 消息文档（SyncML DTD）、设备信息文档（Device info DTD）和元数据信息文档（Meta info DTD）。SyncML 大量的使用了 XML 的名空间，名空间必须在第一个元素类型声明。在 SyncML DTD 中的元素类型被定义在一个 URI 为“http://www.syncml.org/docs/syncml_represent_v10_20001207.dtd 或者 URN 为 syncml:syncml 。SyncML DTD 也可以被 ISO 9070 标准公用标示符 -//SYNCML//DTD SyncML 1.0//EN 识别。任何 XML 的标准属性可以被用在 SyncML 文档中。

下面这个例子就是一个手持设备与一个基于 HTTP 协议服务器之间传递 SyncML 消息实现数据同步。target 和 source 元素分别代表了数据来源和目标；LocURI 包含一个普通的 URL，这个 URL 是执行同步的设备的唯一标志：URL 用于和 Internet 相连的设备，如服务器；IMEI（International Mobile Equipment

Identity）—国际移动设备标志则用 15 个阿拉伯数字代码来鉴别移动设备。在<SyncBody>...</SyncBody>填入 SyncML 命令（如：Add 及增加的数据）完成一次数据同步。

```
<SyncML xmlns='SYNCML:SYNCML1.1'>
<SyncHdr>
<VerDTD>1.1</VerDTD>
<VerProto>SyncML/1.1</VerProto>
<SessionID>1</SessionID>
<MsgID>1</MsgID>
    <Target>
    <LocURI>http://www.syncml.org/sync-server</LocURI>
    </Target>
    <Source>
    <LocURI>IMEI:493005100592800</LocURI>
    <LocName>Bruce1</LocName> <!-- userid -->
    </Source>
    <Cred>
    <Meta><Type xmlns='syncml:metinf'>
syncml:auth-md5</Type></Meta>
    <Data>Zz6EivR3yeaaENcRN6lpAQ==</Data>
    <!— Base64 coded MD5 digest, for user "Bruce1", password "OhBehave", nonce
"Nonce" " -->
</Cred>
</SyncHdr>
<SyncBody>
    <Sync>
        <CmdID>1</CmdID>
    <Add>
        <CmdID>2</CmdID>
<Item>
    <Source>
        <LocURI>./adr</LocURI>
    </Source>
    <Data>begin:vcard
```

```
fn:Carl Smith
n:Smith;Carl
email;type=internet:carl@carlsmithsdomain.com
tel;type=work:+46 8 112 23456789
end:vcard
                </Data>
            </Item>
        </Add>
        </Sync>
    </SyncBody>
    </SyncML>
```

设备性能文档（Device info DTD）：描述设备的能力为客户端和服务端设备所共享。

```
<DevInf>
   …
<SwV>0.99</SwV>
<HwV>3.14</HwV>
…
<DevTyp>pda</DevTyp>
…
 </DevInf>
```

元数据信息文档（Meta info DTD）：描述同步 Session 的参数。

```
<MetInf>
<Format>…</Format>
<Type>…</Type>
…
<MaxMsgSize>586
</MaxMsgSize>
…
</MetInf>
```

这样通过统一定制 XML 文档，并在移动设备和固定设备、异构网络和异构平台间共享，有了共同的通讯语言也就解决了异构环境下的数据映射问题。

3．2．2．2Agent 技术

从最终用户的角度看，软件 Agent 是一种程序，它代表用户，是用户实现其意图的软件助手。它因用户向它指派工作而起作用。

从系统的角度看，软件 Agent 是生存于一个执行环境中的软件对象。为了实现设计目的，它能在那种环境下灵活地、自主地活动。它往往具有以下基本特性：

- 反应能力（Reactivity）：对环境变化的感知能力和应变能力；无论 Agent 生存在现实的世界中还是虚拟的世界中，它们都应该可以感知其所处的环境，并通过行为改变环境。一个不能对环境做出响应的物体不能被称为 Agent。
- 自主性（Autonomy）：又叫自治性是指执行动作的自我控制; Agent 运行时不直接由人或其它东西控制，它对自己的行为和内部状态有一定的控制权。
- 目标驱动：又可称为自发行为（Pro－activeness），传统的应用程序是被动地由用户来运行的而且机械地完成用户的指令。而 Agent 的行为应该是主动的，自发的。Agent 感知周围环境的变化，并做出基于目标的行为。。
- 连续性：运行动作的持续性。

除此以外，还可能拥有以下性质：

- 通信能力（Communicability）：或称为社会能力（Social Ability）指 Agent 能够通过某种通信语言与其它 Agent 进行信息交换。
- 移动能力：在主机站点之间转移。
- 学习能力：根据经验改进自身。
- 可靠性：对最终用户而言是可信任的。

将 Agent 技术应用在 SyncML 协议架构中，能充分利用 **Agent** 的自治性、通信能力等特性来适应移动计算环境的低带宽、易断接等特殊需求。从理论上分析图 7、图 8 所示的 SyncML 协议架构是基于 Client/Agents/Server 结构。该结构明确划分出客户端 Agent（Client Agent）和服务器端 Agent（Server Agent）。客户端 Agent（Client Agent）解释客户机的请求并与服务器端 Agent（Server Agent）一起执行优化处理，以减少在移动网络上的数据传输，改善数据的可用性，并保证移动客户机的操作不间断。此时的服务器端 Agent（Server Agent）如同在服务器端的本地客户机代理；客户端 Agent（Client Agent）相当于一个驻留在本地客户机上的局部服务器代理。这种机制对数据同步的两端都是透明的，在提供统一接口的同时又可为多个应用程序所共同使用。

SyncML 协议架构中的 Agent 是理论阐述中的 Agent：纯粹、单一。在应用实现时，往往不会完全遵照该架构来设计。在后面介绍的 Sync4j 中， SyncML (Server) Agent 就是由 SyncBean、SyncEngine 和 SyncStrategy 三个部分共同组成；SyncML (Client) Agent 则交由具体的客户端开发工具来实现（J2ME 等）。

3.2.2.3 同步引擎技术（Sychronization Engine）

同步引擎实际上是同步服务的逻辑实现，要求具有以下功能：

- 识别出同步数据的来源和目标
- 识别对数据应做的操作。如：增加、删除、更新等等。
- 决定如何实现对应的操作。
- 检测冲突。
- 解决冲突。

同步引擎同步服务的核心内容。但在强调功能完善的的同时，还应具有开放性，允许用户或开发人员加入自己的同步引擎实现。这样开发人员就可以根据实际情况定制出最优的同步策略，从而提高数据同步的执行效率。

同步引擎中的冲突处理机制是实现的重点内容。当一个数据项同时被客户端和服务器端修改时就产生一个更新冲突即同一个数据有两个不同的版本[5]。一个同步协议要求必须有处理冲突的策略。在 SyncML 中规定冲突处理机制是同步引擎的主要功能之一。通常情况下，由 SyncML 的服务端同步引擎处理版本号冲突；有时也可由 SyncML 的客户端引擎提供特定的处理机制。

SyncML 的表示协议提供通报 SyncML 的客户端遭遇冲突的功能和通常处理处理该冲突的状态码。如果通过一个服务端的引擎来解决冲突，它就会利用状态码和通报函数告知客户端冲突的出现及其处理策略。以下是 SyncML 中提供的冲突状态码[7]：

- 207：指通过数据合并解决了冲突。这一状态码表示执行请求命令产生了一个冲突，但冲突通过客户端和服务端数据的合并得到了解决。一同返回的还有数据源和目标的 URL。
- 208：指以客户端命令优先的方式解决冲突。该状态码表示有更新冲突的出现，但通过客户端优先解决了冲突。
- 209：指通过数据副本解决了冲突。该状态码表示请求命令产生了一个更新冲突，通过客户端的数据副本重写服务器端来解决。一同返回还有目标 URI。另外如果是双向同步的话，一条 Add 命令将携带数据副本定义一同返回。
- 409：表示有冲突出现。因为客户端和服务端的数据版本号有更新冲突而造成请求失败。该冲突的处理策略不属于 SyncML 的职能范围，但可以辨别该冲突类型，如有可能应在 SyncML 的范围内加以处理。
- 419：指利用服务端数据解决冲突。该状态码表示客户端请求产生了一个冲突，通过服务端命令优先的策略解决了冲突。
- 423：代表软删除冲突。该状态码表示请求命令失败，因为服务端的“软删除”数据项先于“硬删除”数据项（同一数据项）存在。

以下为当冲突发生时，一个SyncML服务端发往客户端的消息事例：

```
<Status>
<CmdID>1</CmdID>
<MsgRef>1</MsgRef>
<CmdRef>2</CmdRef>
<Cmd>Replace</Cmd>
<SourceRef>1212</SourceRef>
<Data>208</Data> <!—有冲突，客户端数据优先-->
</Status>
```

3．2．3 基于 SyncML 协议的异构数据库间数据同步模型

3．2．3．1 同步模型架构

通过对 SyncML 协议的理解和研究，结合异构数据库数据同步的需求，我们提出了基于 SyncML 协议的异构数据库间数据同步模型（HDBS）。

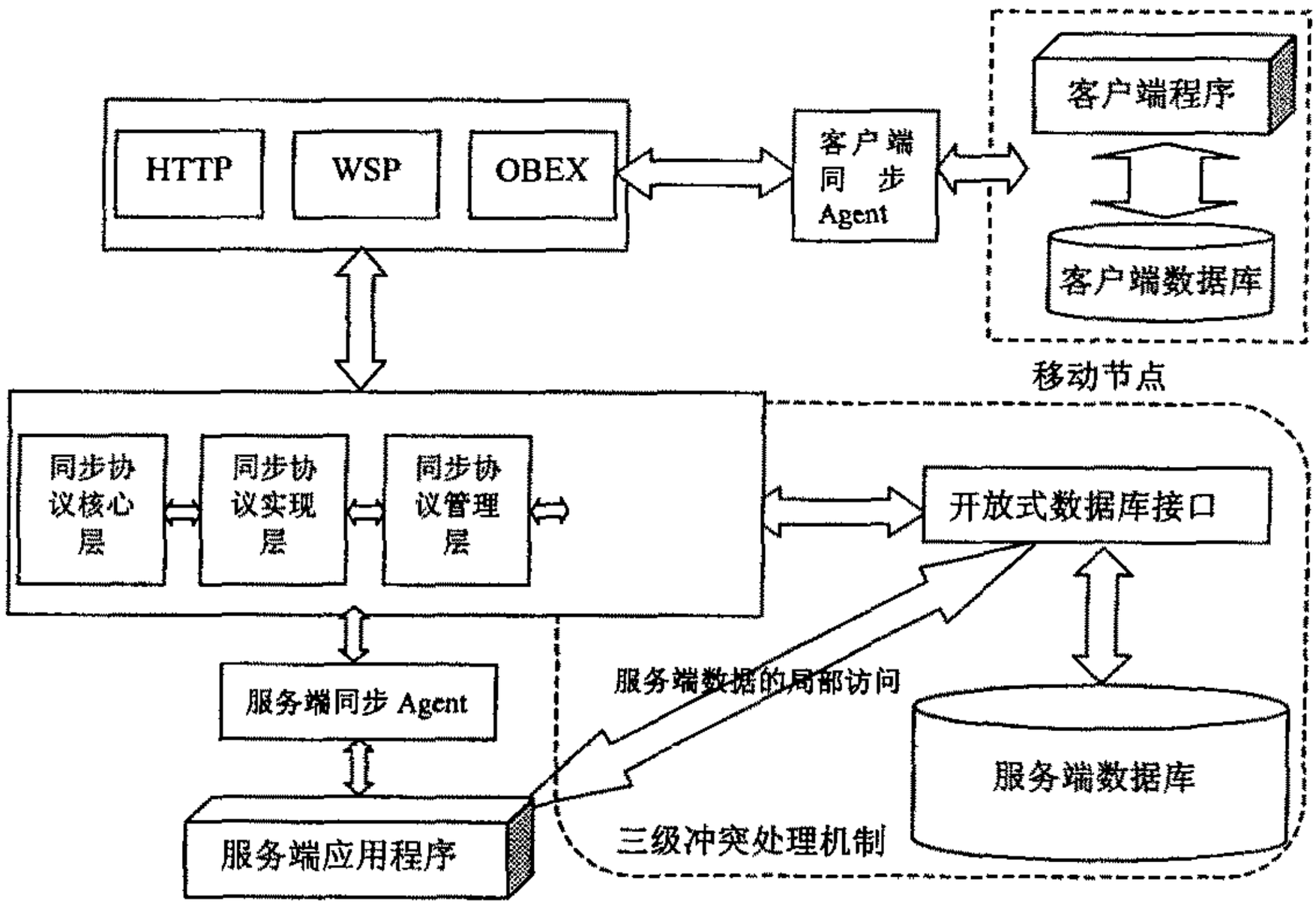

图 11 基于 SyncML 协议的异构数据库间数据同步模型架构

模型中的同步引擎实现同步逻辑（如：冲突处理、设备管理和数据源管理）；同步协议的管理层负责协议工作流的实现，创建和管理协议下层提供的同步对象（如：确保同步双方都完成初始化工作后才进入下一步操作）；同步协议的实现

层完成协议的底层细节工作，如定义同步协议的消息动作等；同步协议的核心层则对应 SyncML 的主要内容，如定义 SyncML 消息文档（SyncML DTD）、设备性能文档（Device info DTD）和元数据信息文档（Meta info DTD）等等，它提供了一个基于 SyncML 协议的最小应用框架；同步 Agent 处理同步两端的相关消息，服务端同步 Agent 负责调用同步引擎的特定函数去实现同步操作，同步 Agent 的设计应具有平台无关性。开放式数据库接口通过扩充数据访问技术，在服务端应用程序、同步引擎和 DBMS 之间提供了统一的访问界面，它既是服务端局部数据的缓冲区又是同步数据的转换场所。

3．2．3．2 三级冲突处理机制

我们在前面提到，同步引擎中应实现冲突处理策略。但对 SyncML 协议的分析发现，作为一个通用的数据同步化协议，它只能提供有限的、常用的、有共性的冲突处理策略，并侧重于冲突的检测而将冲突的解决留给了预定义的冲突处理策略（开发人员自定义的策略和具体 DBMS 预定义的策略）去完成。这样针对一个完整的冲突处理策略就分为三个部分：SyncML 协议的同步引擎发现并部分解决冲突；开发人员自定义的处理策略进一步解决余下的冲突；仍不行就交由具体的 DBMS 去做最后的尝试。对前两级的实现我们在 4.2.2 节和 4.2.3 节中详细阐述，而第三级的实现随具体的 DBMS 而各不相同。如：Oracle9i 是在服务端预定义 10 个方法和可扩展的 PL/SQL 过程来实现；DB2 则是仅在服务端预定方法；其余的类似。

当同步引擎检测到冲突的发生但无法处理该冲突时，就通过开放式数据库接口将它传递到具体 DBMS 上，利用 DBMS 的冲突策略来判断处理并返回结果。具体流程如图 12 所示。这样通过 SyncML 协议同步引擎、开发人员自定义策略和具体 DBMS 冲突处理机制的相互衔接，共同构成一个灵活、开放的三级冲突处理机制（Three Phose Resolved Conflict Mechanism），为移动计算环境下的异构数据库间数据同步实现提供了有力的保障。但在冲突处理策略的效率问题、具体实现的兼容性和复杂性上仍有很长的路要走。

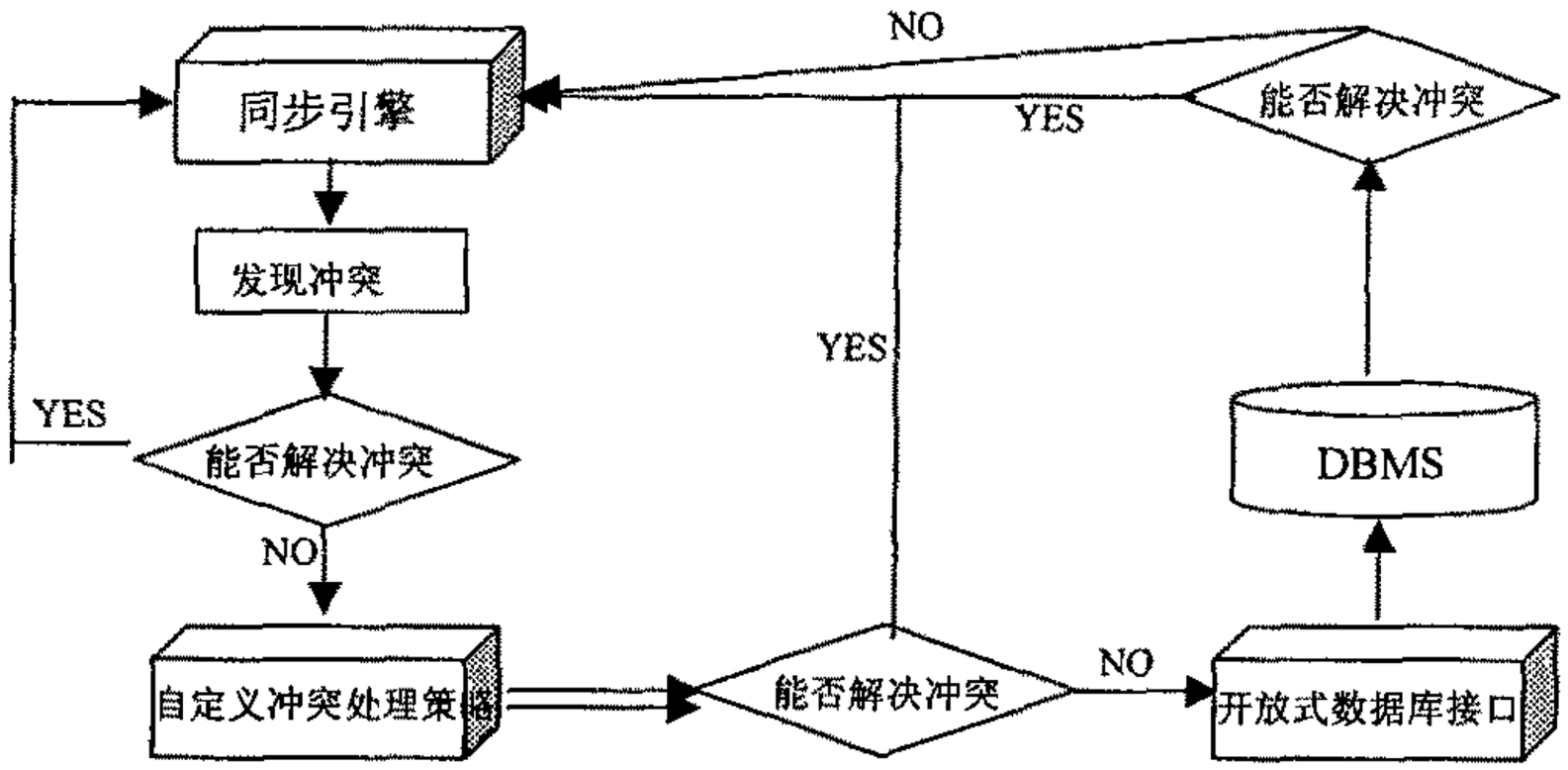

图 12 三级冲突处理机制

3．3 小结

通过对 SyncML 协议架构的扩展和调整，增加同步引擎对开放式数据库接口的支持，利用三级冲突处理机制，使之能满足基于移动环境下的异构数据库数据同步的需求，为 SyncML 协议在数据库领域的应用提供了一条新的思路。我们将在下一章中对这一模型具体化令其更有实际意义。

第四章 移动数据库与主存数据库间的异构数据快速同步模型

图 11 所示的 HDBS 模型是既能在固定网络中实现又能在移动计算网络中运行的通用性模型。为进一步突出移动计算环境的应用特点，满足“在低端移动性强，在高端执行高性能”的要求。我们在 HDBS 模型的基础上再次细化，提出了移动数据库与主存数据库间异构数据快速同步模型（MMQS）。

4．1 MMQS 模型架构

MMQS 模型以 SyncML 协议的 JAVA 实现—Sync4j 为核心，在移动端将移动数据库中的数据格式转换为 SyncML 消息格式并传递给 Sync4j（SyncML Server）；在服务器端则通过 Sync4j.Engine 和 JDBC 联系具体的 DBMS 即 Sync4j.Engine 接受 SyncML 消息再与 JDBC 合作转换为具体的 DBMS 的数据格式，从而实现异构数据的同步。但我们认为这还不符合“在高端提供高性能”的需求。因此，在二者之间再插入一层定制的主存数据库，将它作为 DBMS 活跃数据的内存副本，利用主存的高速访问特性快速实现数据同步。

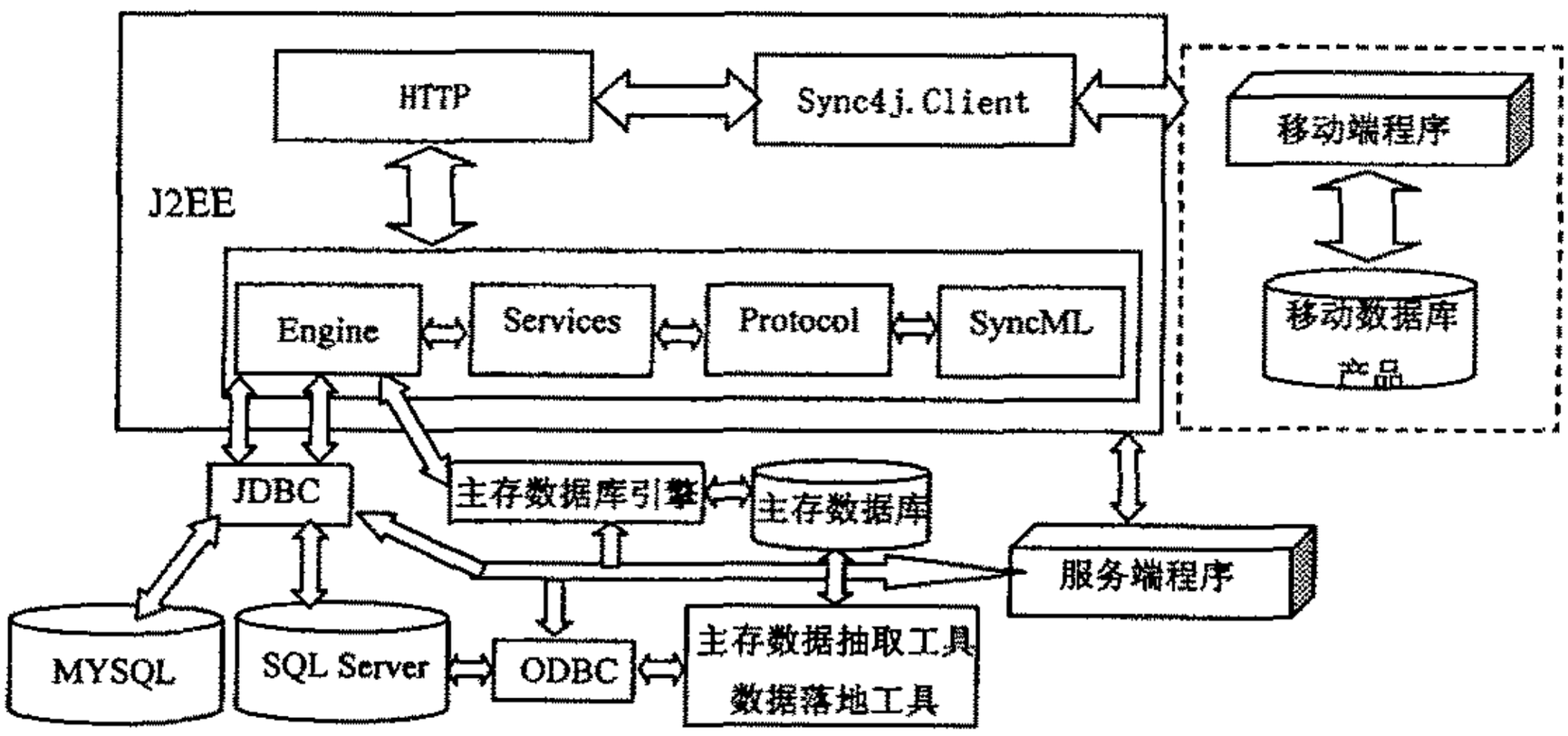

图 13 移动数据库与主存数据库间的异构数据快速同步模型架构

从 MMQS 模型架构可以看出，它在继承了 HDBS 模型良好的弹性架构之上，突出了访问数据的层次划分，满足了快速同步的要求。我们接下来对 MMQS 模型架构三大组成部分：Sync4j、移动数据库和定制的主存数据库及其相互之间的接口设计详细分析。

4．2Sync4j

Sync4j 是基于 J2EE 平台的 SyncML 协议的 JAVA 实现。它提供了一个能创建、发送、接收和处理 SyncML 协议封装过的消息的框架；还提供了 Sync4j 的实现接

口，允许用户和开发人员自定义同步数据源以及冲突处理机制；它将整体划分为三个部分[11]：

- 客户端角色：Sync4j 在定义客户端角色时，并不试图支持所用的客户端设备。作为一个开发工具，它将如何识别和映射各种各样的客户端设备的任务交给了 SyncML 协议去完成。它只提供 Sync4j.Client 怎样获得设备性能文档（Device info DTD）信息的功能，从而简化了客户端角色的繁琐定义，确保了瘦客户端的短小、精干。
- 服务端角色：对于服务端角色，Sync4j 主要实现了 SyncML 协议的数据同步和设备控制的所用功能。它被设计为一个运行在 Java 2 Enterprise Edition (J2EE) 的应用服务器，试图利用这种架构充分获得高质量、高可靠的 QoS。但遗憾的是它仅支持 HTTP 传输协议，今后的 Sync4j 将在适应多传输协议方面作更多的努力。
- 联系二者的开放式架构：Sync4j 服务器是建立在一个模块化、开放式的框架之上。这一框架的设计目的是确保每个模块是可替换的、可插入的和可扩展的。整个框架最重要的模块有：client、core、engine、protocol 和 server。我们就不一一介绍了。

4．2．1Sync4j 对 SyncML 消息的重构

SyncML 消息虽然由 SyncML 消息文档(SyncML DTD)定义，但传递给 Sync4j 处理时要转化为 Sync4j 能识别的格式。Sync4j 通过两种方法对 SyncML 消息进行重构：

- Sync4j 对 SyncML 消息再封装：Sync4j 在 SyncML 消息的外围再定义一层，将 SyncML 消息包含在 Sync4j 的 header 和 body 内，从而转换为 Sync4j 的数据格式。例如：

```
…
SyncHeader header = new SyncHeader(...);
SyncBody body = new SyncBody(...);
SyncMLMessage msg;
msg = new SyncMLMessage(header, body);
String strXML = msg.toXML();
…
```

- 直接调用 Sync4j 的基本类来构造，例如：

```
try
    {…
        String strXML = "<SyncML> … </SyncML>";
        SyncMLMessage msg;
        try
            {…
```

```
                msg = new SyncMLMessage(strXML);
            …}
        catch (InvalidSyncMLException ex)
        {…
            //无效的 SyncML 消息
        …}
catch (XMLSyntaxException ex)
    {…
        //无效的 XML 消息
    …}
```

在不修改 SyncML 消息内容的前提下，Sync4j 实现了对 SyncML 消息的重构。

4．2．2 MMQS 的冲突检测机制实现

Sync4j 的同步化过程包括同步准备（PrepareSync）、同步执行（Sync）和同步结束（EndSync）三个部分（具体同步流程请参阅图 14）。数据同步的冲突检测机制则在同步准备阶段实施（如图 14 的虚线所示）。我们仅针对这一问题加以分析，其余部分请参阅文献[12]。

当一个同步请求传递到同步引擎时，同步引擎首先激活 SyncStrategy.prepareSync(SyncSource[])，由它去查询同步双方已提交的更新内容并整理成两张线性表分别对应同步数据源；然后调用 checkSyncOperation(SyncItem[], SyncItem[])比较表中的每一个数据项的状态（插入状态、删除状态、更新状态等），遵从表 6 所规定的原则得出每个数据项可进行的操作序列（如在数据源 A 上插入一个数据项，从数据源 B 中删除数据项 X，当然也有冲突的出现就返回冲突状态码）；最后将这一结果集传递给 execSyncOperation(SyncOperation operation)执行。

以下是表 6 的第一行数值（数据源 A 的状态为 NEW 时对应数据源 B 的不同状态的检测值）的冲突检测代码实现：

```
case syncItemA.SyncItemState.NEW:
                switch (syncItemB.getState()) {
                    case SyncItemState.NEW:
                        return new SyncConflict(syncItemA, syncItemB,);
                    case SyncItemState.UPDATED:
                    return new SyncConflict(syncItemA, syncItemB,);
                case SyncItemState.DELETED:
                    return new SyncConflict(syncItemA, syncItemB,);
                case SyncItemState.SYNCHRONIZED:
                    return new SyncConflict(syncItemA, syncItemB,);
                case SyncItemState.NOT_EXISTING:
                    return new SyncOperationImpl(principal, syncItemA);
```

}

向客户端和服务端显示发生冲突的数据项及冲突类型。

```
public SyncConflict(SyncItem SyncItemA, SyncItem SyncItemB, String type) {
    super(SyncItemA, SyncItemB, CONFLICT);
    this.type = type;
}
```

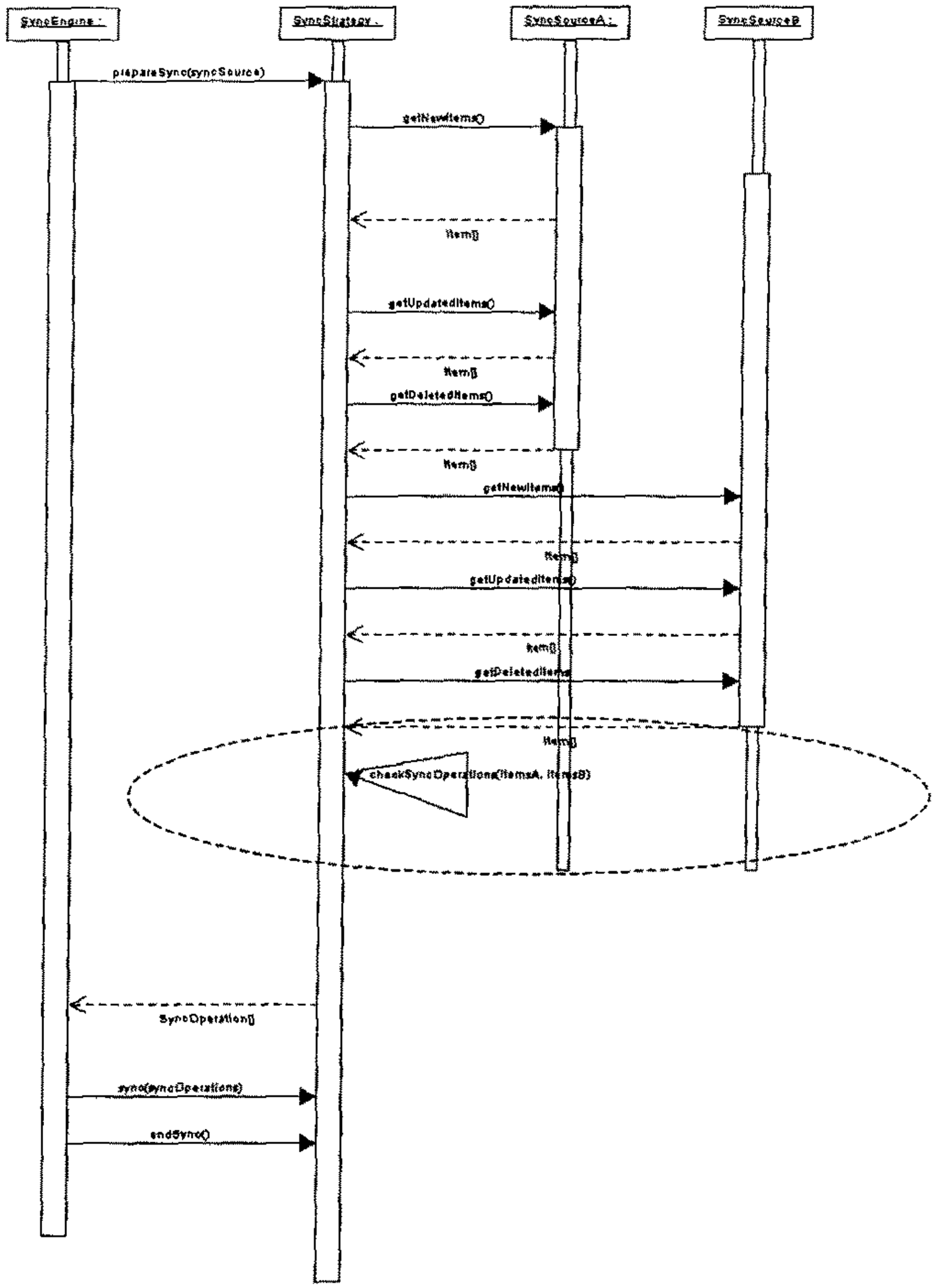

图 14 Sync4j 同步流程图

表 6 Sync4j 数据同步操作检测表

Source A → *↓ Source B*	*New*	*Deleted*	*Updated*	*Synchronized/Unchanged*	*Not Existing*
New	C	C	C	C	B
Deleted	C	X	C	X	X
Updated	C	C	C	B	B
Synchronized/Unchanged	O	X	A	=	B
Not Existing	A	X	A	A	X

X：不存在的数据项

C：有冲突

A：数据项 A 替换数据项 B

B：数据项 B 替换数据项 A

4．2．3MMQS 的冲突解决机制实现

当 checkSyncOperation(SyncItem[ ],SyncItem[ ])检测掉冲突的发生，返回一个 SyncOperation.CONFLICT 结果给 execSyncOperation(SyncOperation operation)方法。但分析其具体实现时却发现以下代码：

```
case SyncOperation.CONFLICT:
    // 忽略并中断该操作
    status = new SyncOperationStatus[0];
    break;
```

这说明 Sync4j 仅提供冲突检测机制，但没有冲突解决机制的具体实现。针对 MMQS 模型我们认为应使用管理者的同步数据优先的解决策略。这是因为：

- 主存事务的高效性要求：在 MMQS 模型中我们采用了主存数据库来代替常见的磁盘 DBMS，在其中存放了活跃数据的副本，并且随着事务的运行其内容会不断调整。因此保持主存事务的高效运行是决定 MMQS 模型效率的重要因素。如果主存更新事务提起的数据同步请求因冲突而中断或回滚事务的话，必造成更新数据的长期锁定而影响整个系统的执行效率。
- 主存更新事务的回滚实现复杂：受储存空间大小的制约，主存更新事务不可能在内存中保留完整、持久的事务日志。数据同步操作是异步执行的，故当同步请求因冲突而回滚是，主存数据库需从磁盘 DBMS 中提取事务日志恢

复，这样经过多次的磁盘读写大大降低了系统的效率。

我们自定义了 ConflictResolvedWithServerData(SyncItem syncItemA, SyncItem syncItemB)类并以数据源 A（主存数据库）为 NEW 状态，数据源 B（移动数据库）也为 NEW 状态这种更新冲突为例说明管理者的同步数据优先的解决策略的具体实现。

```
public class ConflictResolvedWithServerData(SyncItem syncItemA, SyncItem syncItemB)
{
    switch (syncItemA.getState()) {
                case SyncItemState.NEW: // 数据源 A 为 NEW 状态
                    switch (syncItemB.getState()) {
                        case SyncItemState.NEW:{ // 数据源 B 为 NEW 状态
                            syncItemB.State= "NotExisting" );
                            //修改数据源 B 的 NEW 状态为不存在状态
                            execSyncOperation(SyncOperation, NEW)
                        }// 重新调用同步执行操作，此时可执行数据源 A 的 NEW 操作
                    }
}
```

完成管理者的同步数据优先的解决策略后，向移动端和服务器端抛出提示信息：

```
public class ConflictResolvedWithServerDataException extends ServerException
{
    public ConflictResolvedWithServerDataException(final String msg){
        this(msg, null);
    }
    public ConflictResolvedWithServerDataException(final Throwable cause) {
        this("", cause);
    }
    public ConflictResolvedWithServerDataException(final String msg, final Throwable cause){
        super(StatusCode.CONFLICT_RESOLVED_WITH_SERVER_DATA, msg, cause);
    }
}
```

其余的情况（参见表 6）类似，就不一一赘述。

4．3 移动数据库系统

基于移动环境下的数据库系统与传统的基于固定网络环境下的数据库系统相比，具有用户数量变化巨大、众多异构数据过于集中、位置相关信息管理开销大等特点。因而要求移动环境下的数据库系统应具有更好的可伸缩性、模糊性和灵活性。再加上移动网络的高延迟、易断接、低带宽和弱可靠性等客观条件，迫切要求移动环境下的事务处理能在不同设备、平台和网络上正确、高效地运行，

体现出“在低端移动性强，在高端执行效率高”的特点。

4．3．1 移动数据库简介

所谓移动数据库技术是指支持移动计算环境的分布式数据库技术，它涉及数据库、分布式计算以及移动通讯等多个学科领域，已成为分布式数据库一个新的研究方向[20]。由于移动数据库系统的终端设备通常不是传统的台式计算机，而是诸如掌上电脑、PDA、车载设备、移动电话等嵌入式设备，因此，它又被称为嵌入式移动数据库系统。

移动数据库技术在移动计算平台（如 HPC、PDA）、家庭信息环境（如机顶盒和数字电视）、通讯计算平台、电子商务平台（如智能卡应用）、车载计算平台等领域得到广泛的应用。正是基于这一事实，各国研究机构纷纷展开了对嵌入式移动数据库的研究，各大数据库厂商也将开发相应主打数据库系统的嵌入式移动数据库系统作为一个重要的发展方向。国内针对这一领域的主要的产品有：

- 人民大学的《“小金灵”嵌入式数据库》
- 宝钢东软的《嵌入式数据库 OpenBASE Mini》
- 北京大学的《嵌入式数据库 ECOBASE》
- 另外还有华中理工大学的《DM2》、东北大学的《MU/FO-NET》、武汉大学的《WDDBS-32》以及东南大学的《SUNDDB》等

4．3．2 移动数据库系统结构

文献[18]中指出：移动计算数据库环境是移动的、异构的、多数据库系统环境；移动数据库是传统分布式数据库系统的扩展，可以看作客户与固定服务器节点动态连接的分布式系统。传统的分布式数据库管理系难以完全满足移动计算的需求。移动数据库系统结构如图 15 所示。

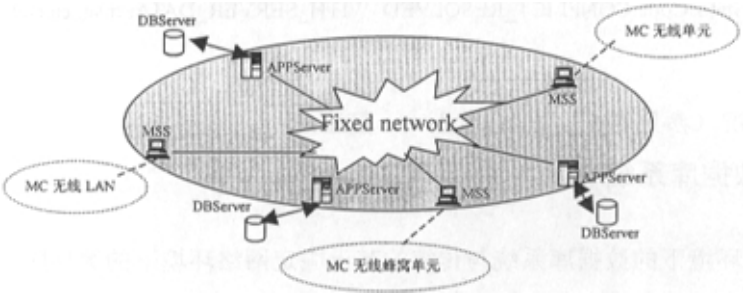

图 15 移动数据库系统结构

其中 MU：移动设备或移动 Client 端；MSS：支持移动计算的固定点，具有无线通信接口；APPServer：应用服务器；DBServer：数据库服务器。

4．3．3 移动数据库的选用

JDataStore™6 是一个纯 JAVA 实现的数据库。它拥有强大的功能：可以轻松地在移动设备上发布；只有很少的 footprint；能自动恢复事务；不需要过多的手工配置。它甚至支持 SQL-92 级操作和多处理器进程等等。这些特点使得它非常适合于商业应用。

在 MMQS 模型中选用 JDataStore™6 作为移动端数据库，主要是它与 Sync4j 能无缝衔接以及配置简单。只要将移动端应用程序代码 ClientApplication.jar 和 JDataStore 的系统类(jdsserver.jar、jdatastore.license)复制到移动设备的指定位置即可。当然也可选用其他移动数据库产品，如：SQL Server for CE、DB2Everyplace 等，但应有相应的 JDBC 驱动程序配合使用。

4．3．4 移动数据库数据向 SyncML 消息的转换

在 MMQS 模型中，移动数据库除了处理本地局部事务外，还要与服务端数据库同步更新数据。因此，移动端的应用程序首先应事先移动数据库的数据格式向 SyncML 消息的转换，并将此消息传递给 Sync4j.engine。为此我们在移动数据库的 JDBC 协助下具体实现了转换功能：

```
/**
* @param ResultSet rs 输入的结果集
* @return String 返回 XML 串
* @exception SQLException
*/
  public static String generateSyncml(final ResultSet rs) throws SQLException {
    final StringBuffer buffer = new StringBuffer(1024 * 4);
    if (rs == null) return "";
    ResultSetMetaData rsmd = rs.getMetaData(); //得到结果集的定义结构
    int colCount = rsmd.getColumnCount(); //得到列的总数
    for (int id = 0; rs.next(); id++) { // 对放回的全部数据逐一处理
    //格式为 row id , col name, col context
      int cmdno=id+2;
buffer.append("\t<Replace>\n");
buffer.append("\t\t<CmdID>").append(cmdno).append("</CmdID>\n");
buffer.append("\t\t<Item>\n");
buffer.append("\t\t\t<Source>\n");
buffer.append("\t\t\t\t<LocURI>clientdb</LocURI>\n");
```

4．4 主存数据库系统

4．4．1 主存数据库的工作原理

现有的数据库服务器虽拥有大容量的主存，但由于数据存储量大故均以磁盘阵列为主要存储载体。这样在事务执行时会因多次磁盘访问而降低系统效率。针对移动计算环境的特殊需求，我们在主存空间中开辟了规模适度的共享内存区域，并以优化的数据结构重新组织、存储和控制系统的活跃数据（同构和异构数据均可），从而避免了磁盘阵列低效的机械读取数据；开发出独有的锁池技术，能最大限度的解决了用户的并发度问题；设计出的规范的接口技术使得平台在提取、交换异构数据和数据结果的多样化呈现方面表现异常出色。在 MMQS 模型中充分体现了“在高端提供高性能”的特点。

4．4．2 主存数据库系统结构

主存数据库系统由四个子系统组成（如图 17 所示）；具体工作流程如图 18 所示。

4．4．2．1 转换子系统

- 主存数据提取工具——即数据的提取机制，采用订制或与用户交互的方式将数据库系统中重要或活跃的数据送入共享内存。
- 主存数据落地工具——当主存数据库空间重整时，当主存数据库与磁盘数据库系统交换数据时，主存中更新事务所对应的数据如何写回到磁盘文件确保更新永久化的实现机制。

4．4．2．2 存储子系统

- 数据在共享内存中的存储实现机制——当数据进入共享内存，建立相应的（多）索引数据并（分别）送入对应的 AVL 树中，数据部分分配空间后链入到唯一的数据占用链中。
- 共享内存空间的动态伸缩管理机制——由于共享内存中可存放异构数据，因此空间的管理不采用定长的分配和回收技术，而是弹性的、可伸缩的实现机制。
- 共享内存空间的重新整理——和大多数系统程序一样，该数据库运行一段时间后会有一定的存储空间碎片产生。在不影响系统运行的前提下，透明的、智能型的实现空间的重整形成了该主存数据库的一大特色。

4．4．2．3 事务并发控制机制[28]

同步事务的并发控制：在系统运行中，数据同步事务与主存数据库本地事务之间的协调运行依赖与主存数据库的事务并发控制机制。故并发机制能支持的事务并发数是决定系统效率的关键因素。特有的锁池技术能在直接高速访问共享内

存数据的同时获得操作系统所能支持的最大进程并发数，满足系统高效的需求。

4．4．2．4 主存数据库引擎

有关主存数据库操作的 JAVA 接口：凡有关主存数据库的操作集均用 C++按 JAVA 调用格式实现并封装为 DLL（动态连接库）。为了 Sync4j.engine 能透明访问主存数据库中的数据，我们利用 JAVA 的 JNI（本地方法）对该 DLL 进行静态加载，提供 DLL 的每个操作的 JAVA 调用接口，使 Sync4j 能轻松、自如地操纵主存数据库。

平台的三个子系统有机的结合在一起，相互协作共同实现高效率的事务处理。

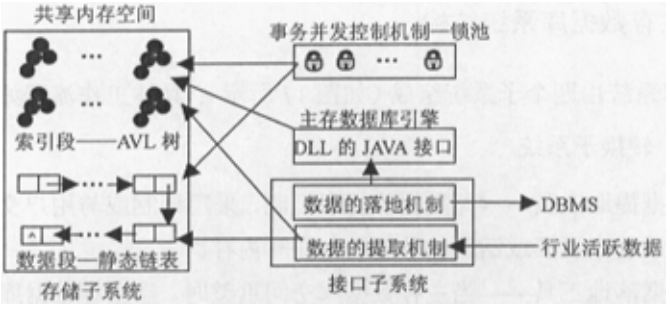

图 17 主存数据库系统结构

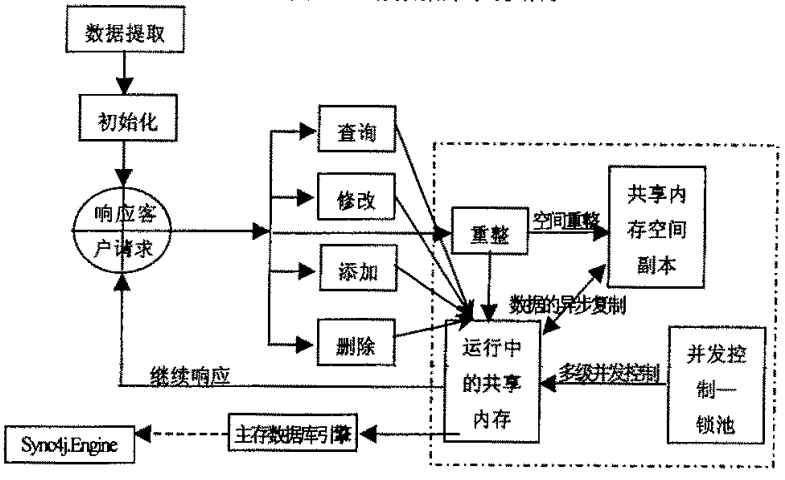

图 18 主存数据库系统的工作流程

4．4．3 实现主存数据库设计的关键技术

4．4．3．1 共享内存机制

主存数据库必须为多用户提供一种机制来解决进程间的数据共享和通信等相关问题。Win32 的 IPC[31]（InterProcess Communication）机制有：Event 事件机制、CritialSection 临界区、Semaphore 信号量、Mutex 互斥量和 FileMapping 文件映射机制。其中 FileMapping 的共享内存形式以访问速度最快、存储空间占用最小及易于并发控制而被首选。

启动主存数据库时通过 CreateFileMapping( )函数（其 HANDLE 参数设置为 0xFFFFFFFF）从数据库服务器中划出指定大小的共享内存并返回该共享内存的 ID，每当用户进程第一次访问主存数据库时，MapViewOfFile ( )函数或 MapViewOfFileEx( )将该共享内存联入到进程的地址空间（两函数的区别详见 Win32 帮助文档）。由于我们在共享内存的存储分配时统一使用相对地址，因此用户进程可以在自己的地址空间中毫无障碍的访问共享内存中的数据。这种机制使每个进程都虚拟的拥有一个共享内存区域的副本。用户进程结束数据访问后，利用 UnmapViewOfFile( )脱离与共享内存的联系，直至最后一个进程完成任务或需要关闭（重启）主存数据库时，CloseHandle( )释放共享内存。

4．4．3．2 存储空间的管理

当 DBMS 的活跃数据送入共享内存后，如何合理地组织存储结构，如何有效地分配、管理存储空间成了主存数据库设计的核心问题。我们从以下两方面加以分析。

- 存储实现

我们将共享内存分为两个段（参见图 19）。一个是索引段，用以支持并发事务访问数据时的快速定位；另一个是数据段，存放完整的 DBMS 活跃数据。针对二者的不同作用，采用了各不相同的存储结构来加以实现。

主存数据库以 AVL 树作为索引结构。针对每一个索引项建立一棵对应的索引项值 AVL 树，其树节点 P 的结构如下所示：

```
struct   AvlTree Node
{
long     Left ;          /*  左孩子偏移量*/
long     Right;          /*右孩子偏移量*/
long     Balancefactor;  /*平衡因子*/
long     DataAddr;       /*对应的数据记录起始地址*/
long     SelfOffset;     /*当前索引项自身的偏移量*/
int      IndexLen        /*整个索引项的长度*/
char     Key;            /*索引关键字*/
```

```
long      DBKey;                    /*对应数据记录的数据库关键字*/
                                    .
                                    .
                                    .
}   P;
```

由于并发事务的任何请求都必须经索引段才能定位具体数据，因此相对而言数据段压力较小，我们采用易于实现的静态链表作为其存储结构。整个数据段再分为占用链表和空闲链表，辅以最快适应算法来分配和回收空间。其数据记录 A 的结构如下所示：

```
struct    DataElement
{ int          Datalen       /*数据记录长度*/
  datatype     Data          /*活跃数据，其类型随着数据的填入而定*/
  long         SelfOffset    /*当前数据自身的偏移量*/
  long         NextOffset    /*下一个数据的偏移量*/
}    A;
```

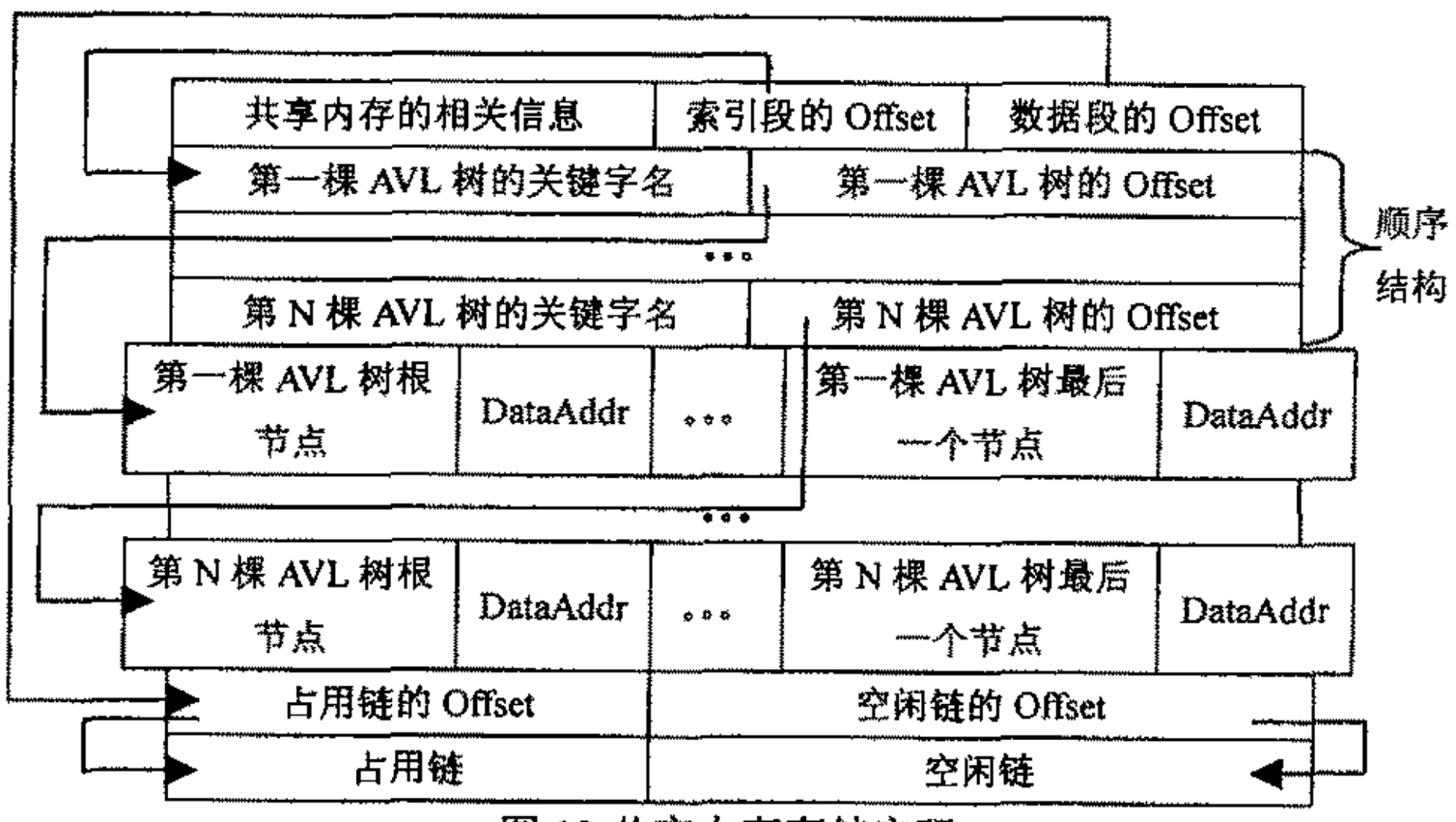

图 19 共享内存存储实现

当并发事务（携带着索引项名和相应的值）发出查询请求时，通过索引段的偏移地址找到第一棵 AVL 树的关键字名（即索引项名）并顺序得到剩下的 N-1 棵 AVL 树的关键字名；这些关键字名依次与并发事务提供的索引项名匹配找出对应的第 i 棵 AVL 树的关键字名，利用其后的偏移地址取得由该关键字名所建立的第 i 棵 AVL 树；再用 AVL 树的相关操作查出与并发事务携带的索引值相一致的索引节点 Q，取出 Q 点中的 DataAddr 就能定位到数据段中的结果数据。类似的，当并发事务发出更新操作时，利用索引段中 AVL 树来定位，利用数据段中占用链和空闲链来分配和回收空间，它们相互协作，共同完成进程请求。

4．4．3．3 存储空间的重整技术

由于涉及到存储空间的分配和回收，不可避免会产生内存碎片，因此需要有一种空间的重整技术来提高共享内存的利用率。我们提出了一种方法：在数据库服务器的主存中再开辟一块同样大小的共享内存，启动重整操作后将正在运行的共享内存中的数据依次导入新划分的共享内存中，同时加以整理。这对于新开辟的共享内存而言就如同执行初始化工作一样，方便、快捷。完成相关操作后在平台空闲时挂起一切请求，将新划分共享内存中重整后的数据一次性写入正在运行的共享内存中，从而实现空间的重整和再利用（图 18 中虚线部分）。这种技术有以下特点：

- 速度快——数据的复制全在内存中完成。
- 智能型——有操作员发布命令、定时启动和监测空闲链使用情况三种触发方式。
- 安全性——集热备份功能于一体，安全性更强。
- 透明性——用户不会感觉到平台状态的变化，即使是在重整期间也能接受查询请求，但为防止重整后脏数据的产生应挂起更新操作。

4．4．3．4 并发控制——锁池技术

为保证共享内存中的数据能在多用户环境下有效的利用，就必须对数据加以并发控制。针对整个共享内存区域，Win32 提供的 CritialSection 临界区、Semaphore 信号量、Mutex 互斥量机制都能实现存取的同步，但从加锁的粒度大小、响应的时间快慢等效率因素来看，信号量均优于其它机制[27]。为取得最大的事务并发数，加锁的粒度要求小到数据记录级，而现有的机制难以实现在共享内存中低粒度地并发控制多个（以系统资源为上限）异构数据记录[31]。为此，我们在信号量的基础上提出了锁池技术，其工作原理是抓住了数据始终是并发控制的落脚点这一本质，将唯一代表数据记录的 **DBKey** 与唯一代表信号量的 **LocksPool[i]**加以绑定，从而实现低粒度的并发。其具体的实现如下所示。

- 锁池的建立

在实现中，我们一次性从系统资源中申请了 16 个信号量并以数组的方式组织起来构成锁池。

```
//创建锁池
var lockspool:array [1..9] of THandle;
...
procedure TForm1.FormCreate(Sender: TObject);
var
    i:integer;
```

```
begin
…
for i:=1 to 16 do
        begin
          lockspool[i]:=Createsemaphore(nil,1,1,PAnsiChar(i));
        end;
…
end;
```

- 锁池的调用

如何将用户访问的数据与锁池中的某把锁发生对应关系是锁池技术的关键所在。当用户处理数据时，同时取出其在索引段中的 **DBKey** 部分，利用函数 DBKeytoI 将其映射为锁池中的一把锁。

```
function DBKeytoI(dbkey:integer):string;
begin
   DBKeytoI:=inttostr(DBKey mod 16);
end;
// DBKeytoI 的值即为 lookspool 中的下标值 i
此时，用户就可以使用求得的锁 lookspool[i]。
if waitforsingleobject(lookspool[i],INFINITE)=WAIT_OBJECT_0
   then
      begin
          …//对该数据的具体操作
      end;
…
//释放该占用锁，放入锁池供下一用户使用
releasesemaphore(lookspool[i],1,nil);
…
```

当多个并发事务访问同一个数据时，由于 **DBKey** 不变导致申请到同一把锁，从而解决了记录级的加锁机制。相应的我们很容易实现了 AVL 树、索引段（数据段）和共享内存的多级加锁机制，顺利完成了高并发度、多层次的同步控制。

4．4．4 主存数据库引擎的实现

主存数据库引擎是整个主存数据库系统的重要环节，它是服务端应用程序、Sync4j 同步引擎分别与主存数据库联系的纽带。其主要内容有服务端应用程序与主存数据库的接口实现；Sync4j 同步引擎与主存数据库引擎的接口实现。

4．4．4．1 普通主存数据库操作的实现

应用程序对主存数据库仍有普通数据库操作的需求（如：插入数据项、删除数据项以及更新数据项等）。为此，我们遵从上述的关键技术要求，利用 C++设计了数据库的普通操作集（MDBOperation.dll），在锁池技术的配合下，高效运行

分布式事务。

4．4．4．2 主存数据库引擎与 Sync4j 同步引擎的接口设计

当主存数据库与移动数据库间有数据同步请求时，主存数据库必须由主存数据库引擎向 Sync4j 同步引擎提交更新数据项集；或从 Sync4j 同步引擎中接收数据项集来同步自身数据。相关操作包括：指定主存同步数据源、测试主存同步数据源的有效性、传递主存数据库的所有更新数据项等。它们共同构成主存数据库引擎与 Sync4j 同步引擎的接口部分。受篇幅限制我们仅以获得主存数据库的所有更新数据项来分析实现代码。

```
// 存储同步数据项的分类线性表
    private ArrayList unchangedSyncItems = new ArrayList();
    private ArrayList newSyncItems       = new ArrayList();
    private ArrayList updatedSyncItems   = new ArrayList();
    private ArrayList deletedSyncItems   = new ArrayList();
//获得主存数据库的所有更新数据项
public SyncItem[] getAllSyncItems(Principal principal)
    throws SyncSourceException {
        SyncItem[] unchangedItems = getUnchangedSyncItems(principal, null);
        SyncItem[] newItems = getNewSyncItems(principal, null);
        SyncItem[] deletedItems = getDeletedSyncItems(principal, null);
        SyncItem[] updatedItems = getUpdatedSyncItems(principal, null);

        SyncItem[] allItems = new SyncItem[ unchangedItems.length
                                          + newItems.length
                                          + deletedItems.length
                                          + updatedItems.length
                                          ];
int c = 0;
for(int i=0; (i<unchangedItems.length); ++i) {
    allItems[c++] = unchangedItems[i];
}
for(int i=0; (i<newItems.length); ++i) {
    allItems[c++] = newItems[i];
}
for(int i=0; (i<deletedItems.length); ++i) {
```

```
            allItems[c++] = deletedItems[i];
        }
        for(int i=0; (i<updatedItems.length); ++i) {
            allItems[c++] = updatedItems[i];
        }
        return allItems;
    }
```

4．5 小结

本章将 HDBS 模型针对移动计算环境进一步落实为 MMQS 模型。在分析了 MMQS 模型的架构之后，对三大组成部分：Sync4j 研究其冲突检测机制并补充实现管理者优先的冲突解决策略；移动数据库简介、常见系统结构及产品的选用；重点分析了移动数据库数据格式向 SyncML 消息转换的实现、主存数据库的架构与关键实现技术。从各个角度、各个方面完善了 MMQS 模型并为其下一步工程应用打下坚实的基础。

第五章　民航移动调度系统

5．1 民航移动调度系统简介

CAPAMD（China Airdrome Parking Apron Mobile Dispatch System）[29]机场移动调度系统是充分运用现代信息、网络和通信技术对民航机场生产运营保障的各个环节和岗位进行统一的组织、协调、指挥和调度的计算机信息管理系统。旨在提高机场客货运输生产的现代化水平、生产效率和服务质量。民航机场外场移动调度系统大大扩展了机场信息化的范围，从过去的机场航站楼扩展到整个机场生产营运范围：航站楼+机坪；改进原有外场工作人员工作无审计记录，提高了工作效率；实现“无处不在的移动办公”；加快了机场航班生产营运信息的流动，大幅度提高了航班正点率，为机场创造了经济效益。

外场监控中心是各个外场业务单位的具体监控系统，实现与外场移动系统数据同步和消息同步的功能，简称为 CAPAMonitor(China Parking Apron Monitor System)。

外场移动终端系统为机场机坪具体业务执行单位使用系统，简称为 CAPAMS(China Airdrome Parking Apron Mobile System)。

5．2 民航移动调度系统的运行环境

5．2．1 硬件平台

- CAPAMonitor 采用 Inter 奔腾处理器的品牌机；
- CAPAMS 支持 Pocket Pc 2002 的移动设备

5．2．2 软件平台

5．2．2．1 操作系统

- CAPAMonitor 采用 Win2000 Server Sp2 以上版本；
- CAPAMS 采用 Pocket Pc 2002。

5．2．2．2 数据库平台

- CAPAMonitor 的数据库根据各地机场不同，采用大型关系型数据库 Sybase、Oracle 等等；
- CAPAMS 选用 JDataStore™6 作为移动端数据库。

5．2．3 其它限制

- 数据同步遵循 SyncML 协议，系统架构参照 HDBS 模型和 MMQS 模型。
- CAPAMonitor 采用 Microsoft Visual C++ 6.0 和 JBuilder4.0 开发。
- CAPAMS 采用 J2ME 开发。
- 采取 XML 业界标准传递数据。
- XML 标准遵循依据采用的 XML 解析器，如 MSXML 4.0 或者其它 API；
- 机场建设的无线局域网络使用频率状况需要机场航空管制的备案；
- 网络协议遵循 IEEE 802.11b；

5．3 民航移动调度系统的设计方案

民航移动调度系统的实现分两步走：第一阶段参照 MMQS 模型实现，重点是定制消息格式，实现移动端程序、服务端程序与 Sync4j 的协同开发，实现同构数据的同步；第二阶段在此基础之上，主要解决主存数据库的设计和扩充 Sync4j 对 IEEE 802.11b 的支持，同时支持两端异构数据库的同步实现。现在正进行第一阶段的开发工作，计划在 2003 年 12 月完成该阶段的编码任务。2004 年产品投入试运行，在调试的同时，开始第二阶段的开发工作，拟定在 2004 年末完成。

5．4 民航移动调度系统的主要功能

- **机坪车辆管理**：接收机坪调度和指挥调度的信息，进行生产运作，向中心报告完成情况等，并查询指挥调度航班计划表和调度表。其中廊桥可看作为机坪的一种特种车辆。
- **客舱服务管理**：读取航班动态信息，接受机坪调度的指令、消息，并向机坪调度报告客舱服务完成状况。主要是作客舱清洁服务。
- **客货搬运管理**：提供完成所有进出港货物、邮件、行李的装卸和装卸工作的班组的排班，行李、货邮交接的记录和班组生产统计，查询显示航班动态信息，向机坪调度报告完成或异常信息等功能。
- **候机服务管理**：引导旅客到达远机位登机，与航班机组交接舱单，并汇报指挥中心和运输调度上客完毕。
- **机务维修管理**：机务工程部为机场的重要保障部门，使用本系统接收航班动态，查询航班计划，报告完成或延误信息，记录航班保障架次和资源使用情况，填写飞机维修记录，接收发送消息。
- **机务车辆管理**：接收机务调度和指挥调度的信息，并进行生产运作，向中心报告完成情况等，并可查询指挥调度航班计划表。

- **食品服务管理**：食品服务人员通过本系统接收航班动态信息，接收食品调度发送的配餐指令、发送接收消息、报告食品完成、处理异常，提高食品服务的工作效率。
- **油料服务**：加油工作人员通过本系统接收航班动态信息、发送接收消息、报告油料完成、处理异常、记录加油信息，提高油料服务工作的工作效率。

5.5 民航移动调度系统的消息表格式

民航移动调度系统除遵循 HDBS 模型和 MMQS 模型定义其系统结构外，与行业运作紧密相关的是业务消息的定义。民航移动调度系统与指挥调度系统通过 TCP/IP 协议组的 802.11b 协议相连。接口以消息表的形式存储在数据库中，消息表名为 MSG_TABLE。以下为系统消息的详细定义。

消息表中 CONTENT 字段内容格式的说明：

表名　操作类型　ROWID更改字段名 旧值　新值

有以下几种操作类型：

- U 更新操作
- I 插入记录
- D 删除记录
- A 插入多条记录
- R 删除多条记录

ROWID 为修改表的记录的行号，如果修改的表没有 ROWID 字段，则在 ROWID 的位置写入该表的主码值，例如：通航机场表，航空公司表等

旧值和新值的格式：当对表做更新操作 U 时，

（1）字符串格式不变

（2）日期格式为 2001/6/19

（3）时间格式为 2001/6/19 12:00:00

（4）数字格式为 str(number) 转换后总长为 10 的字符串

在 DI 插入删除操作时：只有 ROWID 有值，后面的内容没有

在 AR 插入删除多条记录操作时 格式如下：

R　15　表示删除了 15 条记录

A　20　表示插入了 20 条记录

接口有指挥中心内部、航显、楼宇、离港等。即 ZHZX(char[1])、HX(char[1])、LY(char[1])、LG(char[1])。如果消息内容对接口有用值为 1，如果消息内容对接口无用值为 0，消息内容已经用过值为 2。

为提高表在做多条记录同时删除或插入时，消息分发的效率，建议由原来每

条记录产生一条消息改为整个操作完成时只产生一条消息。消息的格式变化如下：删除多条记录（五条以上）的操作代码由　D 改为　R(remove)

插入多条记录（五条以上）的操作代码由　I 改为　A(append)

例如：动态表 HB_DTB 删除 10 条记录后，产生的消息为 HB_DTB　R　10

动态表 HB_DTB 插入 20 条记录后，产生的消息为 HB_DTB　A　20

5．6 小结

民航移动调度系统的实施，是 HDBS 模型和 MMQS 模型在大型商业信息系统中应用。通过前期的论证和试验结果表明，它能弥补外场人员缺乏管理的不足，有效提高民航机场生产营运指挥调度系统工作效率，充分体现了移动技术的发展前景。

第六章 总结与展望

本文从数据访问技术入手，在介绍了 SyncML 协议基础之上，提出了 HDBS 模型并针对移动计算环境的特点，进一步将模型细化为 MMQS 模型，在民航移动调度子系统中得到了应用。沿着这一主线索提供的思路，我们着重分析了异构数据库数据同步的两大难题：异构数据的映射和冲突处理机制，找到了分别用 XML 文档、Sync4j 冲突检测机制和自定义管理者同步数据优先策略作为上述问题的解决方案，取得了较好的实验效果。同时为满足“在低端移动性强，在高端执行高性能”的要求，我们一方面扩充 SyncML 协议以适应客户端设备的多样性；另一方面利用共享内存机制设计了一个保存磁盘 DBMS 活跃数据副本的主存数据库系统，通过减少磁盘读写时间提高服务端工作效率。

下一步我们把研究的重点落在了以下三个方面：

- 调整 SyncML 协议架构，加入静态同步 Agent 和移动同步 Agent 二级代理机制，结合现有的主存数据库系统，扩展移动计算机的数据服务功能[26]，提高 HDBS 模型和 MMQS 模型对移动计算环境的适应能力；
- 另外，文献[15、16]中认为发布、订阅中间件技术是实现数据同步的新技术，我们将结合 SyncML 协议进一步探讨异构数据同步中间件的研制；
- 还有 Martin Rennhackkamp 在文献[14]中指出：随着移动技术的发展，移动数据库间的数据同步将是研究的热点之一。我们认为在一个小范围内的抛弃传统的 C/S 架构，建立移动数据库间自主数据同步机制是具有现实意义的。

HDBS 模型和 MMQS 模型的提出为 SyncML 协议与数据库技术糅合，共同解决移动计算环境下的异构数据库数据同步问题提供了一条新的解决思路。相信沿着这一发展思路不断探索下去，能为移动技术的商业应用起到积极的促进作用。

参考文献

[1] Alonso R, Korth H F. Database System Issue in Nomadic Computing[C]. Proceeding of the 1993 ACM SIGMOD International Conference On Management of Data, WashingtonDC, 1993

[2]Im ielinski T, Basrinath B R. Mobile Wireless computing[J]. Communications of the ACM, 1994, 37(10):19-28

[3]E. pitoura and G. samaras. Data Management for Mobile Computing. Kluwer Academic Publishers, 1998

[4]Execute White Paper. A strategy for the Mobile Enterprise solution. http://www.verizonit.com/pdf/mdm_whitepaper.pdf.

[5] A White Paper by Joe Owen Chief Technology Officer XcelleNet, Inc. When Data Sync Breaks.

[6] SyncML Group. Building an industry-wide mobile data synchronization Protocol, http://www.syncml.org/SyncML White Paper.pdf

[7]SyncML Group. SyncML Representation Protocol version 1.1, http://www.syncml.org/docs/syncml_represent_v11_20020215.pdf

[8]SyncML Sync Protocol version 1.1.1 .Http://www.syncml.org/syncml_sync_protocol_v111_20021002.pdf

[9] Uwe Hansmann, Riku Mettala, Apratim Purakayastha and Peter Thompson. SyncML: Synchronizing and Managing Your Mobile Data[书]。Publisher of Prentice Hall PTR

[10] http://www.mobiledesigntech.com/service/development.htm

[11] http://www.sourceforge.net/Sync4j Architecture.htm

[12] http://www.softforge.net/sync4j/ Sync4j Coding Standards.htm

[13] Yolanda Villate, Evaggelia Pitoura, Arantza IIIarramend. Extending the data services of mobile computers by external data lockers, IEEE

[14]Martin Rennhackkamp. Mobile Database Replication, http://www.dbmsmag.com/DBMS - October 1997 - Server Side.htm

[15] Gianpaolo Cugola and Elisabetta Di Nitto fcugola, Dipartimento di Elettronica e Informazione Politecnico di Milano P.za Leonardo da Vinci 32 20133 Milano, Italy. using a publish subscribe middleware

[16] Dennis Heimbigner University of Colorado at Boulder Technical Report CU-CS-909-00 Department of Computer Science Campus Box 430 University of Colorado Boulder, Colorado 80309 .adapting publish subscribe middleware

[17] 冯玉才 李东 王元珍 曹忠升。一种移动数据库管理系统的体系结构[J]。计算机研究与发展，2001, 5:620-625

[18] 李东 冯玉才 王元珍。适于移动数据库的客户/服务器体系结构研究[J]。计算机应用研究，2001, 4:32-34

[19] 刘翔。DM06：在分布式环境中选择正确的数据同步及复制解决方案。www.sybase.com

[20] 李霖 周兴铭。WCSR：一个弱一致性的复制数据库系统[J]。计算机工程，1999, 4 Vol.25

[21] 数据同步的两大难题。

Http://www.wirelessdevent.com/colums/mar2000/mobdevo4.html

[22] 丁志明，孟小峰，王珊。复制的移动数据库系统事务及同步处理策略，软件学报，2002-13

（2）。
[23]Linux/Unix 下 ODBC 的 安 装 、 配 置 与 编 程 http://www-900.ibm.com/developerWorks/cn/linux/database/odbc/index.shtml
[24] 余萍，罗谦。利用ADO技术对基于BDE架构的数据库应用程序的改造。重庆师范学院学报自然版，2002-27
[25] Microsoft ODBC3.0 程序员参考及 SDK 指南。Microsoft 公司 著，希望图书创作室 译。北京希望电子出版社，1999
[26] 李东 曹忠升 冯玉才 王元珍。移动数据库技术研究综述，计算机应用研究，2000-10（4~7）
[27] W.Richard Stevens 著 尤晋元 等译。UNIX环境高级编程[M]。机械工业出版社，2000.2
[28] 夏英 彭大芹 葛君伟。动态多版本并行控制技术在主存数据库中的实现[J]。计算机应用研究，2002,19(5):105-108
[29] CAPMD-系统功能需求说明书-V0.9.doc
[30] 罗谦、吴跃“基于移动数据库的异构数据一致性技术研究” 已录用
[31] 袁婷、罗谦、周明天“一种基于 UNIX 环境的中间业务平台的研制技术”，计算机应用研究，2003 年 10 月

致谢

衷心感谢我的导师吴跃教授。在研究生阶段的三年中，他一直悉心关怀和耐心指导我的学习和工作，我的每一点进步都是与吴老师的言传身教分不开的。吴老师求实的治学态度，缜密的思维和广博的学识都将使我受益终生。

衷心感谢 2000 级研究生李显双、王珑同学和 2001 级研究生李钰同学，本文中许多思想来自于与他们几年来共同学习和讨论时所受的启发，在此谨对他们表示诚挚的谢意。

衷心感谢 2001 级研究生袁婷同学，在论文撰写和代码测试上的无私帮助。她务实的作风和扎实的专业基础令人钦佩。

衷心感谢我的母亲、姐姐、姐夫和可爱的小侄儿对我学业的支持，感谢你们在生活上给我无微不至的关怀和照顾，你们的温暖永远是我前进的动力。

感谢所有关心和帮助过我的人！

最后，衷心感谢为评阅本论文而付出辛勤劳动的各位专家和学者！

个人简历

罗谦，男。生于 1975 年 6 月 28 日。1998 年毕业于重庆师范学院数学与计算机科学系，计算机应用专业，获理学学士学位。2001 年进入电子科技大学计算机学院攻读硕士，师从吴跃教授，专业计算机软件与理论，数据库技术和应用研究方向。

发表论文

1. 罗谦、潘林森“数据库原理教学中对学生逻辑思维的训练”　《高师教育研究》 1999 年第 2 期
2. 罗谦　“多媒体系统中多种声音的同时输出控制”　《计算机科学》　2000 年第 27 期（8）增刊
3. 罗谦　“数据源动态配置方法的探讨”　《计算机科学 》 2002 年第 51 期（8）增刊
4. 罗谦　“利用 ADO 技术对基于 BDE 架构的数据库应用程序的改造”　《重庆师范学院学报》2002 年第 27 期
5. 罗谦　“高校基于 DW 技术的决策支持系统的设计构想”　《重庆师范学院学报》2002 年第 28 期
6. 袁婷、罗谦、周明天“一种基于 UNIX 环境的中间业务平台的研制技术”《计算机应用研究》2003 年 10 月
7. 罗谦、吴跃“基于移动数据库的异构数据一致性技术研究”　已录用
8. 罗谦、袁婷“动态的主从式移动数据库间数据同步更新策略”　投稿中

科研项目

1.《合川市旅游规划演示系统》 重庆师范学院院级课题 2000 年 6 月~2001 年 1 月
2.《通用演示系统的设计》重庆师范学院院级课题 2001 年 6 月~2002 年 1 月

3.《民航机场站坪移动调度系统》 民航二所开发项目 2003 年 3 月~至今