第3章　数据集成

3.1　数据集成概述

3.1.1　数据集成的必要性

随着信息技术的发展，数据库在各个行业中都得到了广泛的应用。银行、电信、医院、政府、学校等的日常管理与决策，都采用了大型的关系数据库来管理其业务数据。很多企业在发展过程中都开发或引进了许多独立运行的应用系统，每一个应用系统都有自己的运行环境和数据存储方式，随着企业的不断发展，积累了大量的历史数据，这些数据驱动着企业的各项活动，企业为了存储和管理这些数据不断地投资。

然而，由于种种历史原因，企业用于管理它本身数据的工具往往不一致，以至于即使是同一个企业，采用的数据管理系统可能包括简单的文件数据库系统、层次数据库系统、网络数据库系统，以及当前主流关系数据库系统，它们构成了企业的异构数据源。由于这些信息系统是在不同时期、由不同的公司、利用不同的工具、在不同的平台上开发出来的，并且运行在不同的操作系统和不同的数据库平台之上，因而各个应用系统彼此封闭，数据不能交换和共享，数据源之间数据格式和代码不统一，数据大量冗余，从而形成了大大小小的“信息孤岛”。“信息孤岛”造成系统中存在大量冗余数据、垃圾数据，无法保证数据的一致性，从而降低了信息的利用效率和利用率。

随着企业之间竞争的加剧和企业信息化进程的加快，同一企业内部的各个部门越来越需要沟通，以便协调工作、提高工作效率从而提高企业竞争力。沟通的手段是共享信息，沟通的桥梁则是应用系统，于是一个应用系统访问其他应用系统的数据就变得越来越重要、越来越频繁了。而要使这种访问得以实现并变得快捷，就需要将企业的各种异构数据源集成起来。

从外部环境来看，随着信息技术与经济的紧密结合，电子商务已经引起了企业界和商业界的广泛关注，它是一种能使企业在新的经济形势下保持竞争优势、尽快抓住商机的重要营销方式。电子商务将买方、卖方、金融部门、中间商、物流管理部门连为一个整体，企业逐渐从网络上的一个孤立的节点发展成为不断与网络上其他节点交换信息和进行商务活动的实体，企业数据集成也从企业内部集成走向了企业间集成。现在的企业比以往任何时候都需要将内部数据进行发布和交换，这必然导致越来越多的企业应用系统需要访问各种异构数据源，并且这些数据源可能分布在网络上的任何地方。同样地，为了使这种访问得以实现，需要将企业内部的各种异构数据源以及商务链上合作伙伴的异构数据源集成起来。为了解决以上问题，人们开始关注数据集成研究。

3.1.2　数据集成的概念

数据集成是对各种异构数据提供统一的表示、存储和管理，以实现逻辑或物理上有机

的集中。数据集成的核心任务是将互相关联的分布式异构数据源集成到一起，使用户能够以透明的方式访问这些数据源。集成是指维护数据源整体上的数据一致性、提高信息共享利用的效率；透明的方式是指用户不必再考虑底层数据模型不同、位置不同等问题，能够通过一个统一的查询界面实现对网络上异构数据源的灵活访问。实现数据集成的系统称做数据集成系统，它为用户提供统一的数据源访问接口，执行用户对数据源的访问请求。

数据集成的关键技术是如何以一种统一的数据模式描述各数据源中的数据，屏蔽它们的平台、数据结构等异构性，实现数据的无缝集成。数据集成的特征主要表现为分布性、自治性、异构性。

分布性是指集成的各个数据源是异地分布的，依赖网络传输数据，集成时需考虑网络传输的性能和安全性等问题。

自治性是指集成系统不能影响各局部应用系统，被集成的每个局部数据源之上都可能运行着自己原来的应用程序，局部数据源在被集成之后仍然保持一定程度的独立性不受集成系统的影响，并且可以在不通知集成系统的前提下改变自身的结构和数据，因此要求数据集成系统具有一定的鲁棒性。

由于被集成的数据源通常是独立开发的，因此被集成的数据源往往在运行环境、数据模型和数据语义方面存在异构性。环境的异构主要表现在不同的数据源有各自独立的运行环境，包括不同的硬件设备、操作系统和网络协议等；数据模型的不同是指数据源的数据模型存在差异，如结构化数据（如数据库）、半结构化数据（如 HTML，XML）和非结构化数据（如文本、图片）等；数据语义的异构是指各个数据源对相同语义数据的不同表达形式。

3.1.3 数据集成的分类

数据集成的方法可分为三类：①数据转换方法，这种方法实现的是企业数据的松散集成，通过转换工具实现应用系统之间的数据转换和交换，从而达到集成的目的，是一种较低层次的集成；②数据聚合方法，在各种异构数据源的基础上，借助于中间件系统构造一个虚拟的全局数据模式，是一种集中式管理、分布式存储的较高层次的集成模式；③析取、转换和装载（Extract、Transform and Load，ETL）方法，通过对异构数据源中的数据进行分析、转换和装载，建立一个数据仓库，是一种面向企业决策的数据集成方法。

1. 数据转换方法

数据转换方法是一种传统的数据集成方法，相对于其他方法来说，技术上较为简单，该方法比较容易实现，目前在很多领域仍然是一种主要的数据集成方法。数据转换方法通过转换工具在数据库之间进行模式映射，将一个数据库中的数据复制、转换为另一个数据库中的数据，从而实现数据库之间的集成。

目前能实现数据转换的工具很多，大致可以分为三类：①各种数据库管理系统（DBMS）自带的转换、迁移工具；②在应用系统内部集成的转换工具；③通用的、集成的数据转换工具。

目前绝大多数 DBMS 都带有数据转换、迁移工具，以实现本系统与其他 DBMS 甚至非结构化数据库系统的数据交换。如 Oracle 的 Migration Workbench，Microsoft SQL Server 的数据转移服务（DTS），VFP 的导入导出工具和升迁工具等。这些数据转换工具大都能完成常见种类的异构数据库之间的转换，但是也有一定的局限性，即它们都属于特定的数据

库系统，通用性不强。

应用系统内部集成的数据转换工具是本系统与其他应用系统之间的数据接口。从规范性的角度来看，这类数据接口分为两种情况：第一种是在相关的应用系统之间进行数据转换的接口，接口参数完全由设计人员自行规定，主要用于企业内部各应用系统之间的数据转换，而且要求交换数据的双方应用系统由相同的设计人员设计，如果出自不同设计人员之手，则应用系统所有者双方应取得共识，就数据转换达成一致的意见；第二种是遵循某种国家标准或国际标准的转换接口，应用系统通过转换接口将数据转换为标准格式，并传递给目标应用系统，目标应用系统中遵循同样标准的转换接口再将其转换成内部数据格式。第一种类型的转换接口局限性较大，只适合在指定的范围内转换数据，没有通用性；第二种类型由于遵循一定的标准，使用范围较广，典型代表是目前仍在使用的各种电子数据交换(EDI)软件，主要用于商业、零售业、外贸部门、进出口业、工业、化工、石油、汽车等众多行业的数据转换和交换。

集成转换工具是脱离具体的 DBMS 和应用系统，通用性很强的独立软件，可以在任意两个常见的数据源之间进行数据转换。目前成熟的集成数据转换工具比较多，一般都采用向导驱动方式和 GUI 图形用户界面，可以进行 FoxPro、Access 等桌面数据库与 Oracle，SQL Server 等大型数据库之间的数据存储、转换和调用。一般来说，这类工具都允许把一个数据库中的数据(一个或多个表中的部分或全部行)转入至另一个数据库的一个表中(这个表可以存在或不存在)。

2. 数据聚合方法

数据聚合方法是将多个数据库集成为一个统一的数据库视图的方法。可以认为数据聚合工具产生的数据聚合体是一种虚拟的企业数据库，它包括了多个实体的物理数据库。

数据聚合方法利用中间件集成异构数据源，该方法并不需要改变原始数据的存储和管理方式。负责数据集成的中间件系统位于异构数据源(数据层)和应用程序(应用层)之间，向下协调各数据库系统，向上为访问集成数据的应用系统提供统一的全局数据模式和数据访问的通用接口。各数据库的应用仍然完成它们原来的任务，中间件系统则主要为各种异构数据源提供一个高层次检索服务。任何对其他应用系统数据源的访问，都将通过中间件系统，并由中间件系统负责数据的模式映射和转换工作。基于中间件系统的数据聚合模式是实现异构数据集成较理想的解决方案。随着中间件技术的不断成熟和推广，数据聚合将会成为企业数据集成的一种重要方法。

3. ETL 方法

ETL 方法是一种实现异构数据源的集中式管理、集中式存储的方法。ETL 工具从多个数据源中抽取数据，然后进行数据转换和加载，最终得到统一的、完备的数据仓库。原来分散的应用系统仍然独立运作，原来存在的异构数据源仍然为各自的应用系统提供数据服务。这种集成方法的特点是：不会破坏企业原有的应用架构，比较适合于大量数据的迁移，可以提供复杂的数据转换功能，可以集成多种数据源和复杂的商业规则，能容忍数据在时间上的延迟等。

ETL 工具与数据仓库技术密切相关。数据仓库是一种面向主题的、集成的、稳定的、包含历史数据的数据集合，它能够将分布在企业网络中的不同站点的商业数据集成到一起，为决策者提供各种类型的、有效的数据分析，起到决策支持的作用。数据仓库在各种异构数

据源(包括结构化数据源和非结构化数据源)的基础上建立统一的全局模式,用户可以通过数据仓库提供的统一的数据接口进行决策支持方面的查询。

专门提供 ETL 工具的厂商包括 Ascential,Acta,Infor2mation,SAS,Iway 等公司,它们的 ETL 产品一般都提供多种数据析取、转换适配器,同时允许大批量的数据转换和实时的数据操作。与前两类数据集成工具不同,ETL 是基于数据库级的集成工具,用它进行数据集成不涉及应用的集成。

3.1.4 XML 在数据集成中的作用

不同的应用系统(尤其是不同企业的应用系统)开发语言不同、部署平台不同、通信协议也可能不同,对外交换的数据格式更是可能有着巨大的差异。这些不同和差异给系统集成带来了巨大的困难。XML 技术及其相关技术为解决这些困难提供了合理的方案。XML 格式具备描述各种类型数据的能力,DOM/SAX 针对 XML 文档封装了一套有效的处理方法,开发人员可以使用 DOM/SAX 对 XML 文档进行处理,不必自行开发文档格式处理模块,XML 解析器在各种平台上都被开发人员使用,使用 XML 在不同的异构系统之间交换数据是一件非常方便的事情。XML 解决了在不同平台/系统之间的数据结构模式的差异,使得数据层在 XML 技术的支持下统一起来。

随着因特网的发展和 XML 技术的引入,基于因特网的 XML/EDI 方式正逐渐取代传统的 EDI 方式。其工作流程是:用户从自己的数据库中提取出所需的数据并将其转化为标准的 XML 文档(该 XML 文档需遵循行业内共同遵守的一套 XML Schema),应用系统通过 HTTP 将 XML 文档传送到目标系统的电子邮箱或指定的 FTP 目录中,目标系统从相应的位置接收 XML 文档,按照约定的 XML Schema 对传来的数据进行校验,通过 XML 解析器取出 XML 文档中的数据并保存到自己的应用系统之中。

由于 XML 具有极强的适应性并得到多方支持,使其可以实现对资源的快速包装和集成发布。XML 技术与全局数据模式相结合可以使异构数据源集成中间件系统能更好地适应于开放、发展环境(如企业的动态联盟环境)中的数据集成。

3.1.5 数据集成的关键问题

异构数据源集成是数据库领域的经典问题,并随着 XML 技术和中间件技术的兴起,再次成为该领域的一个研究热点。尽管目前已经有不少比较成熟的数据集成方法和相应的工具投入到实际的应用中,然而由于企业应用系统的复杂性、异构数据源的多样性等诸多因素的制约,使得企业数据集成过程变得相当复杂。为了使企业数据集成工作能顺利进行并能取得比较满意的效果,应考虑以下问题。

(1) 集成范围问题。企业应用系统涉及的数据源可能包括结构化数据库、文本文件、HTML 文件、多媒体文件等多种形式,企业数据集成并不是将所有数据源中的所有数据全部集成,那么集成哪些数据源以及数据源中的哪些数据,则是首先要考虑的问题。

(2) 数据资源所有权问题。不同的数据资源可能属于不同的独立经济核算部门,如何在访问异构数据源数据基础上保障原有数据资源的权限不被侵犯,实现对原有数据资源访问权限的隔离和控制,也是企业数据集成过程中应该解决的问题。

(3) 全局模式问题。数据聚合方法和 ETL 方法分别需要建立统一的虚拟数据库和数

据仓库，这两种数据集成方法都需要在各个异构数据源局部模式的基础上充分做好元数据工作，从而建立全局模式。不同的数据源针对同一对象建立起来的局部数据模式往往差别较大，如何在不更改现有异构数据源结构的基础上建立一个能与它们兼容的全局数据模式，应该说是一项比较艰巨的任务。

(4) 模式映射问题。这是对前一个问题的补充。如果经过各方协调已经建立有全局数据模式，应用系统的数据库与虚拟数据库或数据仓库之间的模式映射问题就相对简单一些。如果采用数据转换方法进行数据集成，由于双方的数据模式不同、语义不同，较难进行准确的模式映射，转换后的数据往往不能完全准确地表达源数据的信息。

(5) 数据动态集成问题。企业数据集成的目标并不是建立一个大的统一的数据库并抛弃旧的数据库，而是在将各个数据源的数据集成到某个数据库(某个应用系统的数据库或者整个企业的数据仓库)的同时还保留原来的数据库，并继续为相应的应用系统提供数据服务。因此，企业数据集成应该是一个动态的过程。集成数据库需要经常从变化的数据源中集成新的数据，数据库管理员需要选定合适的周期来刷新集成数据。

3.2　主流的数据访问技术

主流的数据访问技术根据开发平台的不同主要分为微软体系和 Java 体系。微软体系里包含 ODBC(Open Database Connectivity)、OLE DB、ADO 等。Java 体系里包含 JDBC(Java Database Connectivity)和 Hibernate。以下对这些主流的数据访问技术分别做简单介绍。

3.2.1　ODBC

ODBC(Open Database Connectivity，开放数据库互连)是微软倡导的、当前被业界广泛接受的、用于数据库访问的应用程序编程接口。它以 X/Open 和 ISO/IEC 的调用级接口(CLI)规范为基础，使用结构化查询语言(SQL)作为数据库访问语言。

ODBC 总体结构有 4 个组件。

(1) 应用程序。执行处理并调用 ODBC API 函数，以及提交 SQL 语句并检索结果。

(2) 驱动程序管理器。根据应用程序需要加载/卸载驱动程序，处理 ODBC 函数调用，或把它们传送到驱动程序。

(3) 驱动程序。处理 ODBC 函数调用，提交 SQL 请求到一个指定的数据源，并把结果返回到应用程序。必要时，驱动程序还会修改应用程序的请求，以使请求与相关的 DBMS 支持的语法一致。

(4) 数据源。数据源包含了数据库位置和数据库类型等信息，实际上是一种数据连接的抽象。

ODBC 驱动程序的使用把应用程序从具体的数据库调用中隔离开来，驱动程序管理器针对特定数据库的各个驱动程序进行集中管理，并向应用程序提供统一的标准接口，这就为 ODBC 的开放性奠定了基础。ODBC 具有以下几个特性。

1. 数据库独立性

ODBC 是为最大的复用性而设计的。实现一个应用程序使用相同的源代码(不需要重新编译或重新链接)可以访问不同的数据库管理系统和不同的数据源，体现了 ODBC 的数

据库独立性。

2. 互操作能力

通过使用多个驱动程序可以同时访问多个 DBMS 系统。ODBC 提供的 DriverManager 实现所有的 ODBC 函数，多数是传递调用给驱动程序中的 ODBC 函数，并静态链接应用程序，或在应用程序运行时加载它。这样，应用程序在 DriverManager 中按名调用驱动 ODBC 函数，而不是通过每个驱动程序中的指针。当应用程序需要通过特定的驱动程序时，它首先需要一个标识驱动程序的连接句柄。DriverManager 加载驱动程序，并存储每个驱动程序中的函数地址。这样可以实现应用程序调用的 ODBC 函数。基于这样的机制，可以通过 ODBC 实现多个 DBMS 系统的数据互操作。

3.2.2 OLE DB

OLE DB 是一种技术标准，目的是提供一种统一的数据接口。这里所说的“数据”，除了标准的关系型数据库中的数据之外，还包括邮件数据、Web 上的文本或图形等非结构化数据。OLE DB 标准的核心内容就是要求各种各样的数据存储(Data Store)都提供一种相同的访问接口，使得操作数据的应用程序可以使用相同的方法访问各种数据，而不用考虑数据的具体存储地点、格式或类型。

OLE DB 为一种开放式的标准，并且被设计成基于 COM 组件的数据存储对象，能够提供对所有数据类型的操作，甚至能在离线的情况下存取数据。OLE DB 位于 ODBC 层与应用程序之间。

OLE DB 分成两部分：一部分由数据提供者实现，包括一些基本功能，如获取数据、修改数据、添加数据项等；另一部分功能由系统提供，包括一些高级服务，如游标功能、分布式查询等等。这样的层次结构既为数据使用者提供了多种选择方案，又为数据提供方简化了服务功能的实现手段，它只需按照 OLE DB 规范编写一个 COM 组件程序即可。这使得第三方发布数据更为简便，而在应用程序方可以得到全面的功能服务。

OLE DB 模型主要包括以下一些 COM 对象：

1. 数据源(Data Source)对象

数据源对象对应于一个数据提供者，它负责管理用户权限，建立与数据源的连接等初始操作。

2. 会话(Session)对象

在数据源连接的基础上建立会话对象，会话对象提供了事务控制机制。

3. 命令(Command)对象

数据使用者利用命令对象执行各种数据操作，如查询命令、修改命令等。

4. 行集(RowSet)对象

行集对象提供了数据的抽象表示，它可以是命令执行的结果，也可以直接由会话对象产生，它是应用程序的主要操作对象。

OLE DB 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。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。

OLE DB 是一套通过 COM 接口访问数据的 ActiveX 接口。这个 OLE DB 接口相当通用，

足以提供一种访问数据的统一手段，而不管存储数据所使用的方法如何。同时，OLE DB 还允许开发人员继续利用基础数据库技术的优点，而不必为了利用这些优点而把数据移出来。

由于直接使用 OLE DB 的对象和接口设计数据库应用程序需要书写大量的代码。为了简化程序设计，Visual C++ 提供了 ATL 模板用于设计 OLE DB 数据应用程序和数据提供程序。

利用 ATL 模板可以很容易地将 OLE DB 与 MFC 结合起来，使数据库的参数查询等复杂的编程得到简化。MFC 提供的数据库类使 OLE DB 的编程更具有面向对象的特性。Visual C++ 所提供用于 OLE DB 的 ATL 模板可分为数据提供程序的模板和数据使用程序的模板。

3.2.3　ADO

为了实现让各种编程语言都能以统一的访问接口访问数据库，微软推出 ADO(ActiveX Data Objects)数据对象。ADO 是一种与编程语言无关的面向对象的编程接口。

如图 3-1 所示，ADO 可以看作是架接在应用程序与 OLE DB 之间的一座桥梁。它封装了 OLE DB 接口与数据库之间的细节，让各种编程语言直接通过 ADO 来访问数据库。

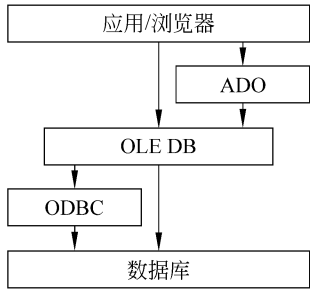

图 3-1　ADO 架构图

ADO 是应用层接口，它的应用场合非常广泛，不仅可用在 VC、VB、Delphi 等高级编程语言环境，还可应用在 Web 开发等领域。它比 DAO(Data Access Object，数据访问对象)和 RAO(Remote Data Objects，远程数据对象)都具有更好的灵活性。ADO 扩展了 DAO 和 RAO 所使用的对象模型，并且使用方便，语法简单，非常易于学习，已成为常用的实现数据访问的主要手段之一。ADO 是 COM 自动接口，几乎所有数据库工具、应用程序开发环境和脚本语言都可以访问之。

ADO 主要包括以下对象，它们的主要功能见表 3-1。

表 3-1　ADO 对象描述

ADO 对象	对　象　描　述
Command	用来定义可在数据源上执行的 SQL 命令与查询
Connection	建立一个前端应用程序到远程数据源之间的连接通道，具体内容包括服务器名、数据库名、用户名和密码等
Error	数据库提供的程序出错时的扩展信息
Field	记录集之中数据的某单个列的信息
Parameter	参数化的 Command 对象的某个参数的信息。该 Command 对象有一个包含其所有 Parameter 对象的 Parameters 集合
Property	某个 ADO 对象的提供程序定义的特征
Recordset	包含某个查询返回的记录，以及那些记录中的游标。选择创建一个 Connection 对象，就可以在同一个连接上打开多个 Recordset 对象

对 ADO 对象的主要操作同 DAO、RDO 库的实现基本相同。主要包括 6 个方面：

(1) 连接到数据源。这是可选的，通常涉及 ADO 的 Connection 对象。

(2) 向数据源提交命令。通常涉及 ADO 的 Command 对象。

(3) 执行命令,例如一个 SELECT 脚本。

(4) 如果提交的命令有结果返回,可以通过 ADO 的 Recordset 对象对结果进行操作,数据存储在缓存中。

(5) 如果适合,可将缓存中被修改的数据更新到物理的存储上。

(6) 提供错误检测。通常涉及 ADO 的 Error 对象。

3.2.4 JDBC

JDBC(Java Database Connectivity)是实现 Java 程序与数据库系统互连的标准 API,它允许发送 SQL 语句给数据库,并处理执行结果。

Java 程序与数据库的连接方式主要有如下 4 种:

(1) 将 JDBC API 作为到另一个数据访问 API 的映射来实现,如开放式数据库连通性(Open Database Connectivity,ODBC)。这类驱动程序通常依赖本机库,这限制了其可移植性。JDBC-ODBC 驱动程序就是类型 1 驱动程序的最常见的例子。

(2) 部分用 Java 编程语言编写,部分用本机代码编写。这些驱动程序使用特定于所连接数据源的本机客户端库。同样,由于使用本机代码,所以其可移植性受到限制。

(3) 使用纯 Java 客户机,并使用独立于数据库的协议与中间件服务器通信,然后中间件服务器将客户机请求传给数据源。

(4) 纯 Java 的方式实现针对特定数据源的网络协议。客户机直接连接至数据源。目前,有关 JDBC 最新的工业规范是 JDBC 3.0,这是在 JCP 上的规范。

JDBC 相关接口/类存放于两个包中:java.sql 和 javax.sql。常用的 JDBC 接口/类主要有:

(1) Java.sql.Driver:是驱动程序必须实现的接口,它提供连接数据库的基本方法。

(2) Java.sql.DriverManager:管理 JDBC 驱动程序,提供获取连接对象的方法,建立与数据库的连接。

(3) Java.sql.Connection:用于 Java 应用程序与数据库建立通信的对象,通过它进而创建 Statement 对象,执行 SQL 语句。

(4) Java.sql.Statement:是对 SQL 语句进行封装的特定对象,通过它执行 SQL 语句进行数据库操作。

(5) Java.sql.ResultSet:用于封装 SQL 语句查询的结果,是一个包含数据库记录的特殊对象。

Java 代码与数据库服务器之间的通信连接通过两种方式:直接连接和池连接。直接连接是在 Java 代码中打开和维护数据库连接,池连接是在一开始创建一定数量的数据库连接,并用一个连接池来管理这些连接。以下介绍直接连接的完整数据库开发步骤。

1. 建立数据源

根据需要安装数据库系统并创建数据库。

2. 装载驱动程序

装载 JDBC/ODBC 桥:Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

装载 JDBC 类,如 JDBC 驱动程序类为 jdbc.JDBCDriverName:

```
Class.forName("jdbc. DriverName");
```

3. 建立连接

建立驱动程序与数据库的连接，语法如下：

```
Connection con=DriverManager.getConnection(url,Login,password);
```

对于 JDBC-ODBC 桥连接，url 可以写成 jdbc:odbc:datasourceName；Login 是登录数据库的用户名；password 是登录密码。

对于使用数据库系统自带的 JDBC 驱动程序，则需查阅其相关文档得知其驱动程序的名字，并使用特定格式的 url 字符串。

4. 建立语句对象

用于向数据库系统发送 SQL 语句：

```
Statement stmt=con.createStatement();
```

5. 执行 SQL 语句

有查询和更新两种类型，查询使用 select 语句，使用语句对象的 executeQuery()方法执行，返回结果集对象 ResultSet；更新包括 insert、update、delete 三种语句，使用语句对象的 executeUpdate()方法执行，不返回结果集，返回影响记录的个数。

6. 查询结果处理

ResultSet 对象提供了多种方法用于处理查询返回的结果，以下是几个常用的方法：

Next()：将记录指针(游标)指向当前记录的下一个记录，返回值为 boolean 型，若返回 false 则表示此后已无记录。

beforeFirst()：将游标指向结果集的第一个记录的前面。

Last()：将游标指向结果集的最后一个记录。

getString()、getInt()、getDouble()、……，获取当前记录指定列的值，参数为整数时指定列号，为字符串时指定列名。

7. 获取元数据

元数据是指描述数据的数据，这里主要是记录集的结构信息。

8. 关闭对象

关闭对象使用 close()方法，并按照 ResultSet-Statement-Connection 的顺序依次关闭所使用的对象。关闭前最好先检查对象是否为 null，否则将产生异常。

9. 处理异常和警告

与数据库应用相关的异常类主要有两个：装载驱动时发生异常的处理类是 ClassNotFoundException，数据库操作时发生异常的处理类是 SQLException。

3.2.5　Hibernate

Hibernate 是一种 Java 语言下的对象关系映射解决方案。它是一种自由、开源的软件，用来把对象模型表示的对象映射到基于 SQL 的关系模型结构中去，为面向对象的领域模型到传统的关系型数据库的映射，提供了一个使用方便的框架。

Hibernate 不仅管理 Java 类到数据库表的映射(包括从 Java 数据类型到 SQL 数据类

型的映射),还提供数据查询和获取数据的方法,可以大幅度减少开发时人工使用 SQL 和 JDBC 处理数据的时间。

它的设计目标是将软件开发人员从大量相同的数据持久层相关编程工作中解放出来。无论是从设计方案开始还是从一个遗留数据库开始,开发人员都可以采用 Hibernate。

Hibernate 对 JDBC 进行了非常轻量级的对象封装,使得 Java 程序员可以很方便地使用对象编程思维来操纵数据库。Hibernate 可以应用在任何使用 JDBC 的场合,它既可以在 Java 的客户端程序使用,也可以在 Servlet/JSP 的 Web 应用中使用。Hibernate 可以在应用 EJB 的 J2EE 架构中取代 CMP,完成数据持久化的重任。

应用程序可以直接通过 Hibernate API 访问数据库。Hibernate API 中的接口可以分为以下几类:

- 提供访问数据库的操作(如保存、更新、删除和查询对象)的接口。这些接口包括:Session、Transaction 和 Query 接口。
- 用于配置 Hibernate 的接口:Configuration。
- 回调接口,使应用程序接收 Hibernate 内部发生的事件,并做出相关的回应。这些接口包括:Interceptor、Lifecycle 和 Validatable 接口。
- 用于扩展 Hibernate 的功能的接口,如 UserType、CompositeUserType 和 IdentifierGenerator 接口。如果需要的话,应用程序可以扩展这些接口。

Hibernate 内部封装了 JDBC、JTA(Java Transaction API)和 JNDI(Java Naming and Directory Interface)。JDBC 提供底层的数据访问操作,只要用户提供了相应的 JDBC 驱动程序,Hibernate 可以访问任何一个数据库系统。JNDI 和 JTA 使 Hibernate 能够和 J2EE 应用服务器集成。

所有的 Hibernate 应用中都会访问 Hibernate 的 5 个核心接口:

- Configuration 接口:配置 Hibernate,启用 Hibernate,创建 SessionFactory 对象。
- SessionFactory 接口:初始化 Hibernate,充当数据存储源的代理,创建 Session 对象。
- Session 接口:负责保存、更新、删除、加载和查询对象。
- Transaction:管理事务。
- Query 和 Criteria 接口:执行数据库查询。

图 3-2 为这 5 个核心接口的类框图。

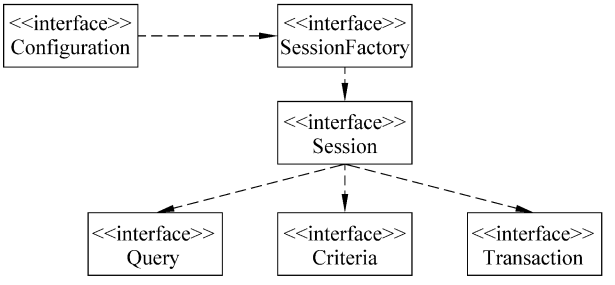

图 3-2 Hibernate 核心接口的类框图