JAVA 技术丛书

JDBC 数据库编程与 J2EE

JDBC Database Programming with J2EE

[美] Art Taylor 著
李东升 等译

PRENTICE HALL PTR
PHEI
電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

JDBC

Working with J2EE

JDBC Database Programming with J2EE

Java 技术丛书

JDBC 数据库编程与 J2EE

JDBC Database Programming with J2EE

[美] Art Taylor 著

李东升 等译

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

JDBC提供了Java应用访问数据库的能力。本书从JDBC和关系数据库的基本概念出发，全面介绍了JDBC数据库编程所涉及的各个方面。书中首先根据JDBC访问数据库的过程，对JDBC API中的各个类及其方法进行了详尽的分析和介绍；接着深入讨论了JDBC和J2EE使用中的一些高级主题，如JDBC设计模式等；最后详细阐述了在Web应用中使用JDBC所涉及到的相关技术，如JSP，servlet，XML，EJB等。全书提供了很多典型的程序范例，向读者清楚地演示了JDBC各种特性的使用。

本书内容丰富全面，既是一本权威的JDBC教程，又是一本实用的JDBC API参考手册。此书面向各个层次的程序开发人员，可作为他们的JDBC编程指南，也可供对Java、J2EE和数据库感兴趣的技术人员和研究人员借鉴参考。

版权贸易合同登记号　图字：01-2002-5712

图书在版编目（CIP）数据

JDBC数据库编程与J2EE/（美）泰勒（Taylor, A.）著；李东升等译. -北京：电子工业出版社，2004.4
（Java技术丛书）
书名原文：JDBC Database Programming with J2EE
ISBN 7-5053-9820-2

Ⅰ. J...　Ⅱ. ①泰 ... ②李 ...　Ⅲ. Java语言-程序设计　Ⅳ. TP312

中国版本图书馆CIP数据核字（2004）第030079号

责任编辑：陶淑毅
印　　刷：北京兴华印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编：100036
经　　销：各地新华书店
开　　本：787 × 1092　1/16　　印张：32.25　字数：826千字
印　　次：2004年4月第1次印刷
定　　价：49.00元

译 者 序

Java 语言以其高稳定性、高安全性、平台独立性以及强大的功能，给网络上的应用开发、数据共享提供了一种新的途径。本书正是要介绍 JDBC 这种重要的数据共享手段。

本书的作者 Art Taylor 是一位资深的 JDBC 专家。他对 JDBC 的规范和应用有很丰富的经验。作者把 JDBC 3.0 规范的介绍和应用教程有机地结合起来。全书共分两部分。第一部分通过丰富的示例详细介绍了 JDBC 规范的具体细节。第二部分则讲解了如何使用 JDBC 开发应用程序。此外，还讲述了 JSP，JavaBean，JNDI，EJB，Swing 和设计模式等内容。

该书内容深入全面，技术实用先进，叙述深入浅出，是一本难得的高层次的技术书籍。现将这本书译成中文，介绍给国内广大读者。

本书由李东升主持翻译，李铭和郑静在本书的翻译和审校过程中也做了大量工作。参加翻译工作的还有杨侃、陈娟、董勇、唐遇星、肖勇、谭玉波、胡光明、刘锋等。

限于译者水平有限，译文中难免有疏漏和错误，敬请读者批评指正。

前　言

编写本书主要有两个目的：详细介绍 JDBC 规范和提供一系列范例来演示如何在实际中通过 J2EE 使用 JDBC。

对 JDBC API 的介绍相对简单，其中介绍了类，描述了方法并演示了如何使用类中的方法。而 Java 数据库应用的各种示例却是举不胜举。数据库应用是基于文本的应用程序，从文件中读入信息并更新数据库。它可以是用 AWT 编写的 applet，可以是用 Swing 编写的 GUI 客户端，也可以是运行在 Web 服务器中的 servlet 或 JSP 页面，还可以是 EJB 形式的分布式组件。

因此，对各种可能的 Java 数据库应用程序都进行演示是不可能的。书中选择的例子中包含了用 J2EE 编写数据库应用的一些常见和有趣的方法。这些应用程序包括基于文本的应用程序、RMI、Java 文件 I/O、Swing GUI、JSP、Java servlet、JavaBean 和企业级 JavaBean。

基于以上考虑，这本书适合于初级程序员、中级程序员甚至高级程序员阅读。此书假定读者具有一定的 Java 和关系数据库的知识，在本书开头还提供了一系列小例子和几个代码片段。最初的几个例子不是要介绍如何在完整的应用程序中使用 JDBC，而是要演示 JDBC API 的特性及方法。本书的前面几章使用了一些简短的例子(大部分工作都在 main 程序块中完成)，重点在于介绍 JDBC API，而不是着重于复杂的应用程序。

本书后面章节中的例子会越来越复杂。显然这是基于这样的假设：读者已越来越熟悉 JDBC 和用 Java 进行数据库编程，并已准备好在更复杂、更真实的例子中学习 JDBC。如前所述，后面章节中的例子使用了 Swing，RMI，JSP，JavaBean，servlet 和 EJB，并应用了各种 Java 设计模式来给代码提供一些附加结构。

本书涵盖了 JDBC 3.0 规范的主要部分。书中的例子是在 JDK 1.4 环境下编译和测试的，例子中使用的 JDBC 驱动程序基于 JDBC 2.1 规范。

本书的支持网站 http://www.phptr.com/taylor/jdbc 上提供了全书的代码示例，同时还提供了其他范例和资料。请读者务必查看该 Web 站点以获得最新的勘误信息、更新内容和补充内容。

本书的读者对象

本书的读者应该熟悉 Java 语言，但不必是 Java 专家。读者只要知道 Java 语言的一些基本知识，熟悉 Java 语法，理解面向对象编程、java 异常处理和类文件的布局就可以了。

本书的读者还应对关系数据库有所了解。因为这里假定读者担任过一个 JDBC 项目的 Java程序员(或曾经参加过一个这样的项目)，所以本书在第 2 章中将会对关系数据库做一些介绍。书中还会介绍 SQL 语言，包括数据操作语言(DML)的基本语法和一些高级 SQL 特性，如联合和复杂连接等。

第 1 章　今天的 JDBC

1.1　简介

自从 1995 年开始投入使用以来，Java 语言已越来越流行。起初 Java 是作为嵌入式系统的一种语言，但现在 Java 语言已经远远不止于此了。今天有成百万的开发者在使用着 Java 语言，在从分布式组件（如企业级 JavaBean）到使用 Swing 和 AWT 的客户端 GUI 开发等各个方面进行着大量的开发工作。Java 和 JSP（Java Server Pages）以及 Servlet 技术的结合可以用来创建 Web 网页，Java 和 Java 插件小程序以及 Java Webstart 技术的结合可以用来开发 Web 应用。

这些应用的一个共同点是它们都需要数据。正如 Internet 时代的市场信息不断提醒我们的那样，信息驱动着企业的发展。应用使用这些信息，而 Java 数据库连接（JDBC）API 正是 Java 应用访问数据的一套工具。

1.2　JDBC 设计

正如 Java 被设计成与软硬件平台无关那样，JDBC 也被设计成向开发者提供某种程度的数据库无关性。JDBC 被设计成提供数据库无关的 API，可以访问不同厂商的数据库。和 Java 应用无需感知它所运行的操作系统平台一样，JDBC 的设计使得数据库应用可以不管底层是什么数据库产品，都使用同样的方法来访问数据。

JDBC 的开发目标是要与最常用的一类数据库——关系数据库一起工作。这并不是说，JDBC 不能够和其他类型的数据库一起使用。实际上，JDBC 驱动程序使得 JDBC API 可以用来连接高端大型机的数据库，这些数据库并不是关系数据库；也可以将文件、表格等当做数据库（显然不是关系数据库）来访问。但实际上 JDBC 通常还是和关系数据库一起使用。

1.3　关系数据库

关系数据库的定义是，将数据以相关实体集合的方式来存储的数据库。这些实体由多个描述实体的属性组成，每个实体是多个行的集合。从另外一个角度看，关系数据库存储了现实世界中的对象信息，对象信息都包含在对象的属性中。

由于现实世界中的对象之间存在着一些关系，在数据库中必须有表示对象之间关系的方法。数据库中数据对象之间的关系是用查询语言来描述的。最常用的查询语言是结构化查询语言（SQL）（第 2 章将对关系数据库和 SQL 进行更详细的介绍）。

关系数据库是今天使用得最多的主流数据库。其他数据库类型有层次式数据库、网络数据库、文件数据库以及面向对象的数据库。尽管层次式数据库仍然是很多大型机系统中通常使用的数据库类型，但它在其他平台上很少使用。

1.4 Java 与关系数据库

因为 Java 是面向对象的语言，所以它和关系数据库在管理数据的方式上不同。在 Java 应用设计中数据以对象的方式进行建模。这些数据对象包含属性（也称做成员），由属性来表示对象的细节。从对象的设计角度看，对象是不被存储的——它是持久存在的。对象的生命周期在应用的多次激活中一直延续。可以用一个语法上类似 C 语言的过程化语言来操作这些对象。

这一切都和关系数据库不同。关系数据库用表和列来表示数据，使用非过程化的语言 SQL 来对数据进行操作。因此需要面对 Java 的面向对象模型和关系数据库的关系模型之间不相匹配这一障碍。我们最终必须通过类的设计来调和它们之间的这些不同。这也就是所谓的“对象-关系”（Object-Relational，OR）映射过程。“对象-关系”的映射过程可以通过手工地应用本书后面的某些设计模式，也可以通过使用各种 OR 映射工具（如 TopLink——www.objectpeople.com，CocoBase——www.thoughtinc.com 等）来完成。

1.5 面向对象数据库

从纯面向对象的角度来看，面向对象数据库很适合面向对象的开发。Java 使用面向对象数据库提供的 API 和查询语言可以访问面向对象数据库。虽然面向对象数据库的这些工具为 Java 中的持久对象提供了便利的工具，但它们一般都不能提供标准的查询语言，并且当数据集的大小增长时会存在性能问题，同时数据查询也会变得更复杂。

面向对象数据库管理系统（ODBMS）在信息技术（IT）的某些领域（如金融和研究领域）比较流行，但现在还没有关系数据库那么流行。

1.6 对象-关系映射工具与 JDO

替代 JDBC 和面向对象数据库这两者的一个有趣方法是 Java 数据对象（Java Data Object，JDO）API。JDO 为持久 Java 数据对象提供了厂商独立的工具。和面向对象数据库一样，JDO 提供一种自然的、面向对象的方法来和 Java 应用中的数据一起工作。像对事务的支持和查询语言能力等问题都已在 JDO 规范中提供了。因为 JDO 规范和任何厂商都不相关，开发者可以使用 JDO 在 SQL Server 数据库上开发 Java 应用；然后可以将该应用移植到 Oracle 或 DB2 上，而无需修改任何代码。

JDO 并不是对 JDBC 的一种替代，相反它是和 JDBC 互补的一种方法。JDO 能够提供 Java 的对象定义与关系数据库的实体及属性之间的 OR 映射，而 JDBC 可提供对数据库的底层访问。JDO 和 JDBC 可以在应用中一起使用，JDO 用来管理大量的持久性对象，而 JDBC 用来提供对那些复杂的传统关系数据库的访问，这些传统的关系数据库到对象的映射已被证明过于困难并且代价高昂难以实现。

1.6.1　OR 映射与 JDO 的限制

从表面上看，JDO 和 OR 映射提供了很有吸引力的方法，但是它们也有很多潜在的问题。即使对于相对简单的应用，数据查询也可能会非常复杂。SQL 自然的非过程化特性使得复杂的查询能够被较为简单地表达出来。JDO 的查询语言是否能够提供这种简练的表达还需进一步观察。

在为应用开发者提供性能和可用性上的益处方面，关系数据库在过去 20 多年中一直积累着重要的技术和经验。而且，Java 应用必须访问的大量现存数据都是存储在关系数据库中的。

1.7　关系数据库与 SQL

关系数据库的一个主要优点是它们都采用标准的 SQL 语言作为查询语言。起初人们希望通过 SQL，针对某个厂商的数据库开发的应用能够很容易地移植到另一个厂商的数据库上。但后来的现实情况不是这样，各个数据库厂商为了和其他厂商相区别，对 SQL 语言在很多方面都进行了扩展。

对 SQL 的扩展既有益处，也带来了新的问题。SQL 的扩展带来的问题是 SQL 本来是一个标准，而在各个厂商进行不同的扩展之后，就减小了 SQL 作为标准的好处。但是 SQL 扩展带来的益处是有些 SQL 扩展是非常有用的，如 Oracle 的 decode 语句。

对现存的 SQL 实现的部分扩展是存储过程语言（Stored Procedure Language，SPL）。由于 SQL 是一个非过程化语言，所以它在管理一些涉及到很多层逻辑的复杂操作（如对大量报表数据应用复杂的业务逻辑）上存在困难。SPL 是类 C 或 Java 的过程化语言，可以通过提供过程化语言工具（如条件语句和控制操作流等）以及声明方法和函数来管理这些复杂的逻辑操作。

数据库引擎中的 SPL 实现是完备的编程语言。在数据库引擎中包含编程语言看起来是冗余和不必要的，因为它们是和像 Java 这样的全能编程语言一起使用的。但使用 SPL 进行数据处理的好处是，数据处理是在数据库引擎中进行的。SPL 过程使用的数据是在数据库引擎的主存空间中，因此没有必要为了进行处理而将数据通过网络传输给一个程序。尽管这种性能上的提升对 2000 行数据的小型处理影响不大，但对大数据集（如百万行数据）的处理带来的好处是非常明显的。对大块的数据，SPL 的使用可能意味着明显的区别：使用一个 SPL 过程对百万行数据进行处理只需 1 小时，而将数据从数据库中导出并由一个程序来进行处理则需要 8 个小时。

很多关系数据库也提供了数据库触发器（trigger）。这些触发器和某个数据库表相关联，当数据库上产生与该数据库表相关的操作时触发某些行为。数据库更新触发器可能是最常用的触发器，它在数据库对相关数据库表进行插入、更新或删除操作时触发。触发器是提高数据库完整性的非常好的方法，还可以用于执行业务规则、复制数据，以及通过对表更新记日志的办法提供一些审计工具等。

对关系数据库的其他重要扩展还有数据分片。数据分片是当数据表的数据分布在多个分离的逻辑设备上时产生的，数据分片可以提高对数据表的大量行进行的扫描操作的性能。同

时，由运行在不同机器上的两个不同数据库服务器保持完全同步的数据库复制也能够带来很多重要的好处。

1.8 JDBC API

在 1996 年制定了一系列规范之后，于 1997 年发布了 JDBC API。JDBC API 的设计使得对关系数据库的调用级接口（Call Level Interface，CLI）访问是厂商无关的。每个关系数据库都已经有自己的访问数据库的 CLI 版本。这些 CLI 原来是为 C 语言以及后来的 C ++ 语言创建的。为了减少不同的 CLI 实现带来的麻烦，X/Open 联盟创建了一个标准 CLI 规范。

JDBC 目前被分成了两个 Java 包：`java.sql` 和 `javax.sql`。`java.sql` 包中包含了原有的 JDBC API 核心部分以及后来对其进行的部分改进。`javax.sql` 包中包含了对 JDBC API 的一些扩展，这些扩展提供了很多有用的特点，并且原来是作为 JDBC 2.0 标准扩展的一部分加入的。`java.sql` 和 `javax.sql` 这两个包都是 J2SE 1.4 版本的一部分。

JDBC 规范提供了数据库厂商必须实现的接口集合。不同厂商在如何实现 JDBC 规范上可以具有一定的灵活性。目前有四类不同类型的实现，详见表 1.1。

表 1.1 JDBC 驱动程序类型

驱动程序	描述
类型 1	通过将 JDBC 调用映射到其他 CLI 调用来实现 JDBC，使用由其他语言所写的二进制库。要求客户机提供软件，如 JDBC-ODBC 桥驱动程序
类型 2	驱动程序一部分由 Java 代码组成，另一部分由使用其他 CLI 的本地代码组成。要求客户端二进制代码
类型 3	纯 Java 驱动程序，使用中间件（middleware）将 JDBC 调用转换成访问数据库所需的厂商相关的调用和协议
类型 4	纯 Java 驱动程序，实现了本地协议。无需中间件或任何客户端二进制代码。需要时可下载到客户端

JDBC 驱动程序有四种不同的类型。这些驱动程序的不同主要在于驱动程序的组成部分、各个组成部分所处的位置以及用来开发这些组成部分的语言。每个数据库厂商都使用不同的调用和不同的网络协议来访问数据库。数据库厂商提供自己特殊的 API 和驱动程序来访问自己的数据库，任何类型的 JDBC 驱动程序都需要将 JDBC 调用映射或转换成厂商自己的协议。对第一种类型的 JDBC 驱动程序，这种映射在二进制库到本地的 CLI 间有个额外的间接层。第三种类型的驱动程序通过一个中间服务器组件来提供这种映射，由中间服务器组件与客户端驱动程序进行通信并提供映射及数据库通信。第四种类型的驱动程序通过编写的用于管理厂商相关协议的纯 Java 代码来提供映射。

第一种类型和第二种类型的驱动程序都需要客户机器上提供一些二进制代码，而第三种类型和第四种类型驱动程序是纯 Java 的解决办法，大大减少了 JDBC 驱动程序的移植性问题。

第二种类型的驱动程序需要客户机器上的一些二进制代码。JDBC 调用被转换成数据库厂商相关的协议，隐式地将这些 JDBC 调用映射到使用其他语言所写的数据库驱动程序（这些驱动程序通常由数据库厂商提供）上。

通常我们推荐第四种类型的驱动程序。第四种类型驱动程序是纯 Java 代码，提高了驱动程序的可移植性，从而使得驱动程序开发者无需为多种平台间的移植煞费苦心。第四种类型

驱动程序在性能上也有一定的优势，因为它采用了高效率的代码。在第四种类型的驱动程序中，JDBC 调用无需映射到专门的 CLI 调用（如同第二种类型驱动程序中的那样），也无需第三种类型驱动程序中所需的中间件，从而避免了额外的网络开销。

1.9　今天的编程

今天的编程已远远不止针对客户-服务器或单个集成应用这些简单需要。在 Internet 时代，应用由分布在多个机器上的很多不同部分或组件组成是很常见的。这种分布式编程要求多层或 n 层的开发方法。

多层编程也叫做分布式编程：应用由多个一起工作的组件构成。多个组件可以在一个服务器上运行，也可以在多个服务器上运行，它们都被看成是一个完整应用的一部分。

通过多层编程，单个应用由运行在不同体系结构层次上的多个组件构成。从设计角度看，这些逻辑层次以及各层要完成的工作反映了各个组成部分的“责任”。为多层应用提供一个一致的结构对应用开发是有益的。通常 n 层开发使用下面的几个层次。

- 客户层
- 表示层
- 业务层
- 资源层

客户层负责和用户进行交互。交互包括用户界面的显示，以及用户输入的初始化处理。在 Web 应用中，客户层就是 Web 浏览器。

表示层负责为客户层准备输出并与业务层有接口。表示层不执行任何业务逻辑。也就是说，表示层不涉及企业的业务逻辑，业务逻辑应由业务层来处理。在 Web 应用中，表示层通常是能够处理 JSP 或 servlet 页面的 Web 服务器。

业务层负责执行企业的业务逻辑。业务层处理来自表示层的请求（这些请求是由客户层转发给表示层的）。业务层与资源层有接口，以访问处理时所需的数据。

资源层负责管理应用的资源。对大多数应用来说，资源层代表着数据库。资源层是应用数据被持久存储和管理的地方。

1.9.1　n 层结构中的 JDBC 代码

使用 JDBC 的 Java 程序通常处于业务层。业务层上实现数据访问的代码被隔离并封装成一系列的黑盒对象，这些对象隐藏了内部细节，对外只展示一个简明接口。

多层分布式应用的体系结构和 Java 设计模式将会在第 13 章进行详细的介绍。值得注意的是，JDBC 代码会因我们所编写的组件处于体系结构的不同层次而有着不同的应用。可以使用 Java 设计模式来帮助指导代码的编写过程。

1.9.2　分布式编程中的 Java 技术

Sun 公司将众多 Java 技术组合起来形成一个包作为 Java 2 企业版（Java 2 Enterprise Edition，J2EE）来销售和发布。J2EE 包中包含了大量的 API 和技术，这些技术提供了开发分布式