清華
电脑学堂

- 总结了作者多年数据库开发经验和教学心得
- 系统讲解了数据库开发的要点和难点
- 实例众多、图例丰富、实用性强
- 提供丰富的课堂练习和课后习题
- 附赠大容量、高品质素材和案例

数据库开发技术标准教程

刘畅 彭涛 编著

从零开始，循序渐进
本书介绍了数据库的基础知识和操作技巧，内容由浅入深，循序渐进，适合零基础读者快速入门。

延伸学习，深入掌握
本书赠送30个办公软件模板和数据库开发PPT。

系统全面，易学易用
本书构筑了面向实际应用的知识体系，体现了理论的适度性、实践的指导性和应用的典型性，对难点和重点做了详细讲解和特别提示。

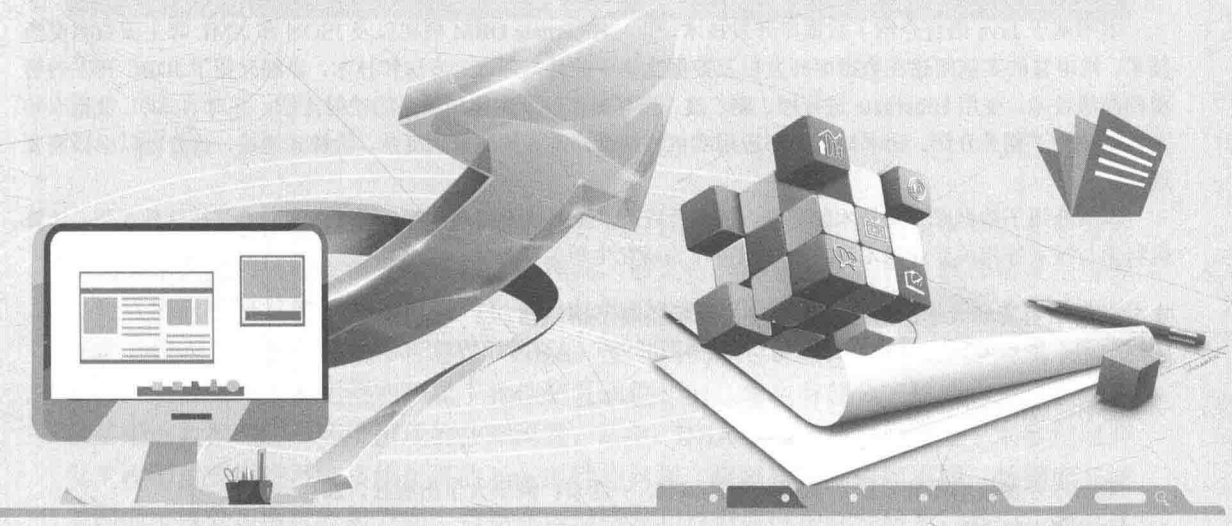
全程图解，快速上手
本书采用全程注解方式，代码做了大量注解，插图做了标注处理，信息丰富，阅读体验轻松，上手容易。

紧贴实际，案例导航
每章根据所讲内容配备精彩案例和课后练习，读者可边学边练，既可全面了解数据库开发的各种方案，又可快速掌握基于实际应用的项目和任务。

海量资源，轻松获取
本书附赠海量资源，已经上传到“益阅读”空间，读者只需扫描封底二维码，便可轻松获取丰富的学习资源。

清華大学出版社

清华电脑学堂



数据库开发技术标准教程

刘畅　彭涛　编著

清华大学出版社
北京

内 容 简 介

本书基于 Java 语言介绍了数据库开发技术，引入 Hibernate ORM 框架以及 JSON 和 XML 等主流数据交换技术，用丰富的案例阐述在数据库开发以及数据交换中的基本原理、方法和技术，详细介绍了 JDBC 开发的初级和高级技术，使用 Hibernate 进行增、删、改、查等操作以及实体和联系的映射方法，并对 NoSQL 数据库和大数据进行了相关介绍。全书知识点与应用实例相结合，内容从简单到复杂，阶梯式递进，读者可以根据需要选读。

本书介绍了数据库开发技术的原理、技术及应用，注重理论联系实际，既可作为高等院校软件工程、计算机科学与技术等相关专业的本科教材，也可作为研究生相关课程的参考资料。

图书在版编目(CIP)数据

数据库开发技术标准教程 / 刘畅，彭涛编著. —北京：清华大学出版社，2019
（清华电脑学堂）
ISBN 978-7-302-51736-8

Ⅰ. ①数… Ⅱ. ①刘… ②彭… Ⅲ. ①数据库系统－系统开发－教材 Ⅳ. ①TP311.13

中国版本图书馆 CIP 数据核字（2018）第 271361 号

责任编辑：秦　健　薛　阳
封面设计：杨玉兰
责任校对：徐俊伟
责任印制：李红英
出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，　http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者：三河市金元印装有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　张：**15.5　　**字　数：**375 千字
版　　次：2019 年 2 月第 1 版　　**印　次：**2019 年 2 月第 1 次印刷
定　　价：49.80 元

产品编号：063127-01

前 言

背景

数据库开发工程师是从事数据库管理系统（DBMS）和数据库应用软件设计研发的相关工作人员的统称，属于软件研发工程师，但又有一部分运维工作。软件研发工程师主要从事软件研发的工作，但同时也要参与数据库生产环境的问题优化和解决。数据库开发工程师与传统的数据库管理员（DBA）是不同的职位。传统的 DBA 主要属于运维职位，而数据库开发工程师则属于软件研发职位。但二者也有部分工作内容重合，比如都要跟进数据库生产环境出现的故障问题，其中，DBA 主要负责故障处理，而数据库开发工程师主要跟进系统模块出现的 bug 或性能问题。根据研发的内容不同，数据库开发工程师可以分为两大发展方向：数据库内核研发和数据库应用软件研发。其中，数据库应用软件研发主要负责设计和研发数据库管理系统衍生的各种应用软件产品，重点关注的是数据库外部应用软件产品架构的设计和实现，比如分布式数据库、数据库中间件等。

在计算机科学与技术、软件工程等本科专业的课程体系中，程序设计类课程和数据库类课程都是非常重要的课程群。程序设计类课程主要包括程序设计基础、面向对象程序设计等课程，而数据库类课程主要包括数据库系统等课程，而在专业综合实践和毕业设计等教学环节中，基于数据库系统的软件开发是学生需要具备的非常重要的核心技能之一。这也是编者编写本教材的初衷。通过本教材和相关课程的学习，读者将理解程序和数据之间的生产者/消费者关系，为相关的实践教学环节和就业奠定坚实的技术基础。

本书特色

本书不仅结合实例详细讲解了 Java 数据库开发的基础知识，同时还就 Java 数据库开发的主要应用进行了实例讲解。全书共 8 章，详细介绍了 JDBC 开发的初级和高级技术，使用 Hibernate 进行 CRUD 操作以及实体和联系的映射方法，并对 NoSQL 数据库和大数据进行了相关介绍。

全书知识点与应用实例相结合，介绍了数据库开发技术的原理、技术及应用，注重理论联系实际。本书内容从简单到复杂，阶梯式递进，读者可以根据需要选读。

读者对象

本书可作为高等院校软件工程、计算机科学与技术等相关本科生专业教材，也可作为相关学科的研究生参考资料，还可作为学习 Java 开发、数据库开发、Java EE 开发的职业技能培训教材。

本书作者

本书受到北京联合大学“十三五”规划教材建设项目、北京市教育委员会科技发展计划面上项目（SQKM201411417013、KM201211417002）资助，由北京联合大学软件工程优秀教学团队完成。参加本书编写的有北京联合大学机器人学院的刘畅、彭涛。其中，第1～3章、第7章和第8章由刘畅编写，第4～6章由彭涛编写。全书由刘畅、彭涛统稿。在编写过程中得到了孙连英教授、刘小安等研究生的指导和帮助，在此表示感谢。

由于作者水平有限，书中疏漏之处在所难免，敬请读者批评指正。

编者

目 录

第1章

数据库开发技术概述

大多数读者已经学习了有关数据库系统的知识，对数据库、数据库管理系统、数据库系统有了基本的认识，也掌握了关系数据库的原理、概念，能使用 SQL 对关系数据库中的数据进行操纵。SQL 的使用方式包括交互式和嵌入式。在数据库系统课程中，主要以交互式的方式来使用 SQL，而在真正的企业级应用中，SQL 更多的是嵌入在应用程序中，以嵌入式的方式来运行。本章主要介绍应用程序与数据库之间的关系，主要说明数据库系统与应用程序之间的关系。之后在介绍多层软件架构的基础上，对数据访问层（也称持久化层）进行了阐述。在数据访问层中，有很多数据库访问接口技术，选择其中部分技术进行了简介。除了上述传统的数据库访问接口之外，近年来，对象-关系映射技术也得到了非常广泛的应用。本章介绍了对象-关系映射技术及 Hibernate 框架，同时对 XML 等非关系型数据存储技术进行了说明。

1.1 应用程序与数据库的关系

数据库系统（DataBase System，DBS）的结构如图 1.1 所示，其中，数据库（DataBase，DB）提供数据的存储功能。数据库管理系统（DataBase Management System，DBMS）提供数组的组织、存取、管理和维护等基础功能。数据库应用系统根据应用需求使用数据库，数据库管理员（DataBase Administrator，DBA）负责全面管理数据库系统。图 1.2 是引入了数据库之后的计算机系统的层次结构。

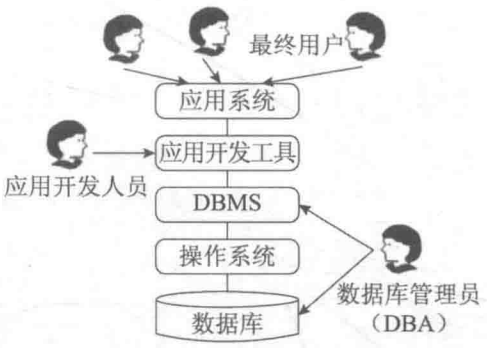


图 1.1 数据库系统的结构

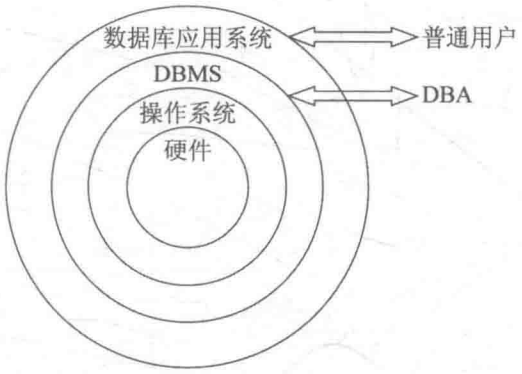


图 1.2 引入数据库后计算机系统的层次结构

从图 1.1 中可以看出，在数据库系统中，除了数据库之外，其余均为软件，尤其是用户直接使用的各类应用系统。从图 1.2 中也可以看出，普通用户直接与各类数据库应用系统进行交互，完成各类业务活动。因此，在计算机系统中普遍存在的一个关系就是应用程序与数据库二者之间的关系。

1.1.1 应用程序与数据库的关系概述

实际上，在数据库的不同历史发展阶段中，应用程序与数据之间的关系也在发生变化。在人工管理阶段（20 世纪 50 年代中期以前），应用程序与数据之间是一一对应的，其关系如图 1.3 所示。人工管理数据主要具有如下特点。

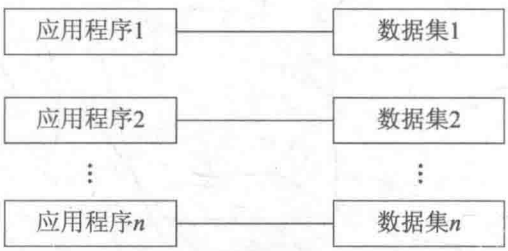


图 1.3 人工管理阶段应用程序与数据之间的一一对应关系

- 数据不保存；
- 应用程序管理数据；
- 数据不共享；
- 数据不具有独立性。

在文件系统阶段（20 世纪 50 年代后期到 60 年代中期），应用程序与数据之间的对应关系如图 1.4 所示。用文件系统管理数据具有如下特点。

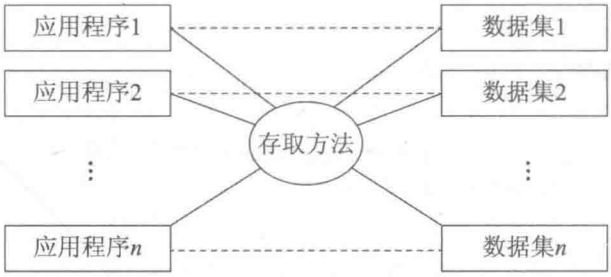


图 1.4 文件系统阶段应用程序与数据之间的对应关系

- 数据可以长期保存;
- 由文件系统管理数据。

文件系统存在的缺点如下。

- 数据共享性差、冗余度大;
- 数据独立性差。

从 20 世纪 60 年代后期以来，主要采用数据库系统来管理数据。该阶段应用程序与数据之间的对应关系如图 1.5 所示。

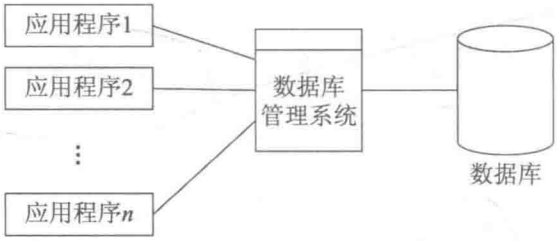


图 1.5 数据库系统阶段应用程序与数据之间的对应关系

与人工管理和文件系统相比，数据库系统的特点主要有以下几个方面。

- 数据结构化：数据库系统实现整体数据的结构化，这是数据库的主要特征之一，也是数据库系统与文件系统的本质区别。
- 数据的共享性高、冗余度低且易扩充。
- 数据独立性高。
- 数据由数据库管理系统统一管理和控制。

数据库是长期存储在计算机内有组织、大量、共享的数据集合。它可以供各种用户共享，具有最小冗余度和较高的数据独立性。数据库管理系统在数据库建立、运行和维护时对数据库进行统一控制，以保证数据的完整性和安全性，并在多用户同时使用数据库时进行并发控制，在发生故障后对数据库进行恢复。

数据库系统的出现使软件系统从以加工数据的程序为中心转向围绕共享的数据库为中心的新阶段。这样既便于数据的集中管理，又能简化应用程序的研制和维护，提高了数据的利用率和相容性，提高了决策的可靠性。目前，数据库已经成为现代软件系统的重要组成部分。具有数百 GB、数百 TB、甚至数百 PB（1PB=2^{50}B）、百 EB（1EB=2^{60}B）的数据库已经普遍存在于科学技术、工业、农业、商业、服务业和政府部门的软件系统中。

1.1.2 数据库系统的结构

考察数据库系统的结构可以有多种不同的层次或不同的角度。从数据库应用开发人员角度看，数据库系统通常采用三级模式结构，这是数据库系统内部的系统结构。从数据库最终用户角度看，数据库系统的结构分为单用户结构、主从式结构、分布式结构、客户机/服务器、浏览器/应用服务器/数据库服务器多层结构等。这是数据库系统外部的体系结构。目前，许多主流的数据库系统采用了客户机/服务器（Client/Server）体系结构。

数据库系统的三级模式结构是指数据库系统是由外模式、模式和内模式三级构成，如图 1.6 所示。数据库系统的三级模式是数据的三个抽象级别，它把数据的具体组织留给数据库管理系统管理，使用户能逻辑地、抽象地处理数据，而不必关心数据在计算机中的具体表示方式与存储方式。为了能够在系统内部实现这三个抽象层次的联系和转换，数据库管理系统在这三级模式之间提供了两层映像：外模式/模式映像和模式/内模式映像。这两层映像保证了数据库系统中的数据能够具有较高的逻辑独立性和物理独立性。其中，外模式/模式映像主要解决的就是应用程序与数据之间的独立性问题。

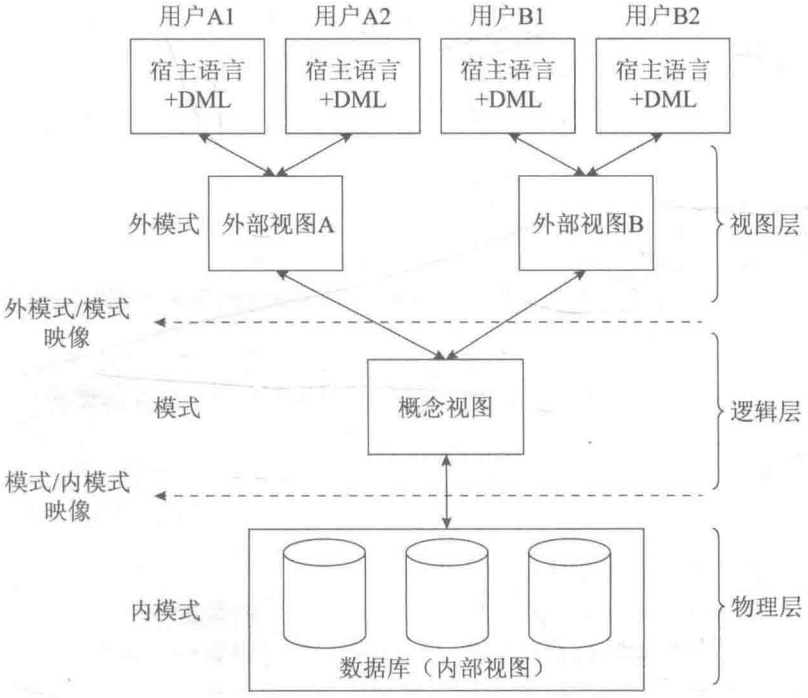


图 1.6 数据库系统的三级模式结构

模式描述的是数据的全局逻辑结构，外模式描述的是数据的局部逻辑结构。对应于同一个模式可以有任意多个外模式。对于每一个外模式，数据库系统都有一个外模式/模式映像，它定义了该外模式与模式之间的对应关系。这些映像定义通常包含在各自外模式的描述中。当模式改变时（例如，增加新的关系、新的属性、改变属性的数据类型等），由数据库管理员对各个外模式/模式映像做出相应改变，可以使外模式保持不变。

应用程序是依据数据的外模式编写的，从而应用程序不必修改，保证了数据与程序的逻辑独立性，简称数据的逻辑独立性。

特定的应用程序是在外模式描述的数据结构上编制的，它依赖于特定的外模式，与数据库的模式和存储结构独立。不同的应用程序有时可以共用同一个外模式。数据库的二级映像保证了数据库外模式的稳定性，从而从底层保证了应用程序的稳定性，除非应用程序的需求本身发生了变化，否则应用程序一般不需要修改。

数据与应用程序之间的独立性使得数据的定义和描述可以从应用程序中分离出去。另外，由于数据的存取由数据库管理系统来管理，从而简化了应用程序的开发工作，大大减少了应用程序维护和修改的成本。

数据库系统一般由数据库、数据库管理系统（及其应用开发工具）、应用程序和数据库管理员构成。各种人员的数据视图如图1.7所示。

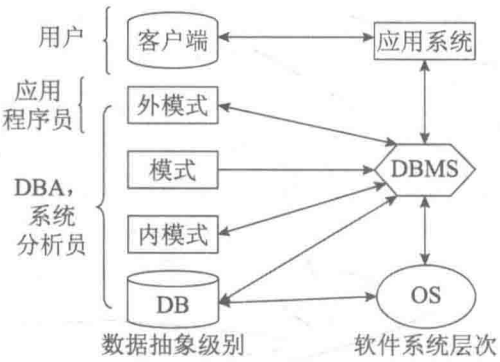


图1.7 各种人员的数据视图

其中，数据库系统的软件主要包括以下几个方面。

（1）数据库管理系统。数据库管理系统是为数据库的建立、使用和维护配置的系统软件。

（2）支持数据库管理系统运行的操作系统。

（3）具有与数据库接口的高级语言及其编译系统，便于开发应用程序。

（4）以数据库管理系统为核心的应用开发工具。应用开发工具是系统为应用开发人员和最终用户提供的高效、多功能的应用生成器、第4代语言等各种软件工具。它们为数据库系统的开发和应用提供了良好的环境。

（5）为特定应用环境开发的数据库应用系统。

本书主要讨论上述第3～5部分中涉及的相关数据库应用系统开发技术。

1.2 多层软件架构

对程序员来说很常见的一种情况是在没有合理的软件架构时就开始编程。在没有一个清晰的和定义好的架构的情况下，大多数开发者和架构师通常会使用标准式的传统多层架构模式（也被称为多层架构）。在多层架构中，通常将源码模块分割为几个不同的层。

应用程序缺乏合理的架构一般会导致程序的过度耦合、容易被破坏、难以应对变化等情况。这样的结果是，如果没有充分理解程序系统里的每个组件和模块，就很难定义这个程序的结构特征。

1.2.1 多层软件架构简介

多层架构是一种很常见的架构模式，也称为*N*层架构。这种架构是大多数企业级应

用系统的实际标准，因此很多的架构师、设计师以及程序员都很熟悉多层架构。许多传统 IT 公司的组织架构和分层模式十分相似。因此，多层架构很自然地成为大多数应用程序的架构模式。

多层架构模式的组件被分成几个平行的层次，每一层都代表了应用程序的一个功能（展示逻辑或者业务逻辑）。尽管多层架构没有规定自身要分成几层，大多数多层架构其结构都包括以下 4 个层次：展示层（也称为表现层，Presentation Layer）、业务层（也称为业务逻辑层，Business Layer）、持久层（也称为持久化层，Persistence Layer）和数据库层（Database Layer），如图 1.8 所示。有时候，业务层和持久层会合并成单独的业务层，尤其是持久层的逻辑绑定在业务层的组件当中。因此，有一些小的应用程序可能只有 3 层，而一些有着更复杂业务的应用程序可能会有 5 层或者更多的分层。

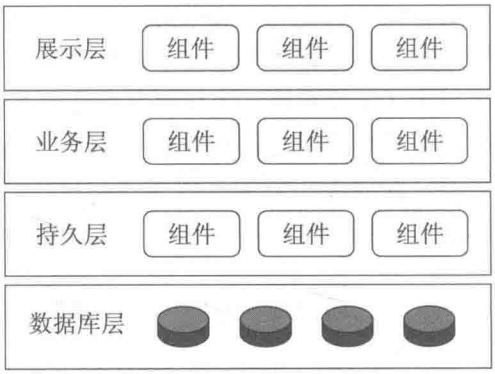


图 1.8 常见的多层软件架构

多层架构中的每一层都有着特定的角色和职能。例如，展示层负责处理所有的界面展示以及交互逻辑；业务层负责处理请求对应的业务。每一层是具体工作的高度抽象，都是为了实现某种特定的业务请求。例如，展示层不需要关心如何得到用户查询的数据，只需在屏幕上以特定的格式来展示这些数据。业务层不关心数据的展现形式，也不关心数据来自何处，只负责从持久层得到数据，执行与数据有关的相应业务逻辑，然后把这些信息传递给展示层。

多层架构的一个突出特点是组件之间的关注点分离。一个层中的组件只会处理本层的逻辑。例如，展示层的组件只会处理展示逻辑，而业务层中的组件则只会处理业务逻辑。利用组件分离的技术，能够更容易构造有效的角色和强有力的软件模型。这样极大地降低了应用程序的开发、测试、管理和维护等工作的成本。

1.2.2 多层软件架构的特点

注意图 1.9 中每一层都是封闭的。这是多层架构中非常重要的特点。这意味着用户的请求必须一层一层地传递。例如，从展示层传递来的请求首先会传递到业务层，然后传递到持久层，最后才传递到数据层。

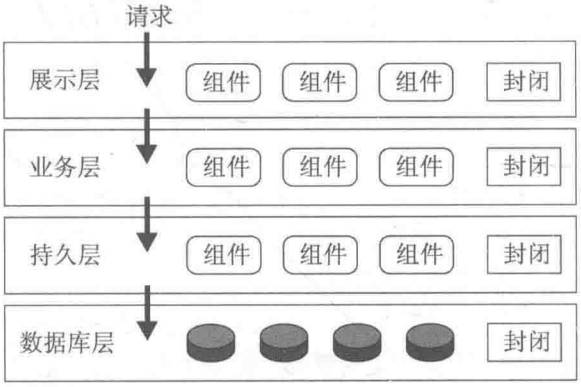


图 1.9 多层架构中每一层的封闭性

那么为什么不允许展示层直接访问数据层呢？如果只是获得以及读取数据，展示层直接访问数据层，比穿过很多层一步步得到数据快得多。之所以这么做，涉及一个概念：层隔离。层隔

离是指架构中的某一层的改变不会影响到其他层，这些变化的影响范围仅限于当前层次。如果展示层能够直接访问持久层，展示层就会与持久层紧密相关。如果持久层中的结构变化了，展示层就会受到影响，很可能需要修改程序。这样会让应用变得紧耦合，组件之间互相依赖，这种架构的致命缺点是难以维护。从另外一个方面来说，分层隔离使得层与层之间都是相互独立的，架构中的每一层的相互了解都很少。为了说明这个概念的强大之处，可以想象一个超级重构：把展示层从 JSP 换成 JSF。假设展示层和业务层之间的联系保持一致，业务层不会受到重构的影响，它和展示层所使用的界面技术完全独立。

然而封闭的架构层次也有不便之处，有时候也需要开放某一层。例如，如图 1.10 所示，新建了一个服务层。新的服务层是处于业务逻辑层之下的，表现层不能直接访问这个服务层中的组件。如果服务层是封闭的，业务逻辑层需要通过服务层才能访问到持久层，这样使操作复杂化，不合理。应将服务层设置为开放的，所有请求可以绕过这一层，直接访问持久层，这样就更加合理、灵活了。

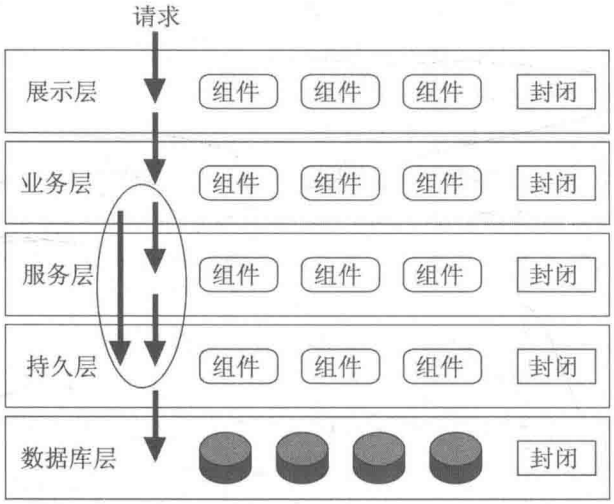


图 1.10　增加了开放的服务层的多层软件架构

开放和封闭层的概念确定了架构层和请求流之间的关系，并且给设计师和开发人员提供了必要的信息，以理解架构里各种层之间的访问限制。如果随意地开放或者封闭架构里的层，整个项目可能都是紧耦合、一团糟的，以后也难以测试、维护和部署。

多层架构是一个很可靠的架构模式，适合大多数的应用程序开发。从架构的角度考虑，选择这个模式还要考虑很多的因素，例如，整体灵活性、易于部署、可测试性、系统总体性能、可伸缩性、易开发性等方面。

1.3　数据访问层

数据访问层又称为 DAL（Data Access Layer），有时候也称持久层或持久化层（Persistence Layer），主要负责数据的访问，可以访问数据库系统、二进制文件、文本

文档或是 XML 文档。简单来说，就是实现对数据表的 Select、Insert、Update、Delete 等操作，也就是常说的增删改查操作，英文缩写为 CRUD（即增加（Create）、读取查询（Retrieve）、更新（Update）和删除（Delete））。如果要加入 ORM（Object Relation Mapping，对象关系映射，请参见 1.5 节及第 5 章）的元素，那么就会包括对象和数据表之间的映射，以及对象实体的持久化。数据访问层在多层架构中的地位如图 1.11 所示。

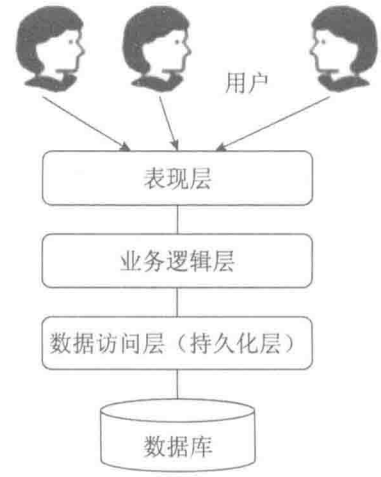


图 1.11 数据访问层在多层架构中的地位

狭义的理解，“持久化”仅指把领域对象（Domain Object，即业务对象（Business Object），属于业务逻辑层）永久保存到数据库中；广义的理解，“持久化”包括和数据库相关的各种操作，具体如下。

- 保存：把业务对象永久保存到数据库。
- 更新：更新数据库中业务对象的状态。
- 删除：从数据库中删除一个业务对象。
- 加载：根据特定的 OID（Object IDentifier，对象标识符，具有唯一性），把一个业务对象从数据库加载到内存。
- 查询：根据特定的查询条件，把符合查询条件的一个或多个业务对象从数据库加载在内存中。

持久化技术封装了数据访问细节，为大部分业务逻辑提供了面向对象的 API，其优点主要如下。

- 通过持久化技术可以减少访问数据库的数据次数，增加应用程序执行速度；
- 代码重用性高，能够完成大部分数据库操作；
- 松散耦合，使持久化不依赖于底层数据库和上层业务逻辑实现，更换数据库时只需修改配置文件而不用修改代码。

在数据访问层（即持久化层）中可以使用数据库访问接口直接访问数据库（请参见 1.4 节），也可以使用某种持久化框架来访问数据库。目前广泛使用的持久化框架包括 Hibernate（请参见第 5 章）、MyBatis（原来的 iBatis 已更名为 MyBatis）等。

1.4 常见数据库访问接口

在数据访问层中，主要使用的数据库访问接口包括 ODBC 数据库接口、OLE DB 数据库接口、ADO 数据库接口、ADO.NET 数据库接口、JDBC 数据库接口等。

1. ODBC 数据库接口

ODBC 即开放式数据库互连（Open Database Connectivity），是一种实现应用程序和关系数据库之间通信的接口标准。1991 年 11 月，Microsoft 宣布了 ODBC。1992 年 2 月，

Microsoft 推出了 ODBC SDK 2.0 版。符合标准的数据库就可以通过 SQL 编写的命令对数据库进行操作，但只针对关系数据库。目前所有的关系数据库都符合该标准（如 SQL Server、Oracle、Access 等）。ODBC 本质上是一组数据库访问 API（应用程序编程接口），由一组函数调用组成，核心是 SQL 语句，其结构如图 1.12 所示。

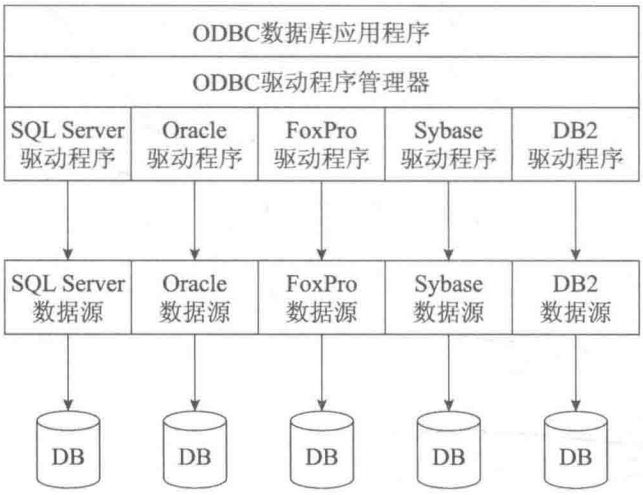


图 1.12 ODBC 数据库接口

2. OLE DB 数据库接口

OLE DB 即数据库链接和嵌入对象（Object Linking and Embedding Data Base）。OLE DB 是 Microsoft 推出的战略性的、通向不同的数据源的低级应用程序接口。OLE DB 不仅包括 Microsoft 资助的标准数据接口开放数据库连通性（ODBC）的结构化查询语言（SQL）能力，还具有面向其他非 SQL 数据类型的通路。

OLE DB 是微软提出的基于 COM 思想且面向对象的一种技术标准，目的是提供一种统一的数据访问接口访问各种数据源，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、目录服务（Directory Services），以及主机系统中的文件和地理数据以及自定义业务对象等。

OLE DB 标准的核心内容是提供一种相同的访问接口，使得数据的使用者（应用程序）可以使用同样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，其结构图如图 1.13 所示。

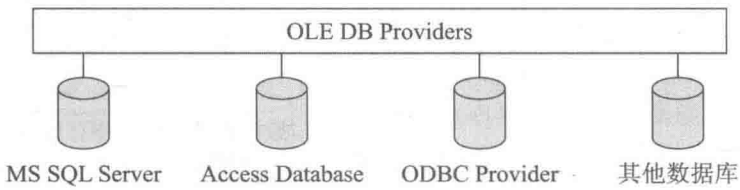


图 1.13 OLE DB 数据库接口