# Hibernate ORM

# 最佳实践

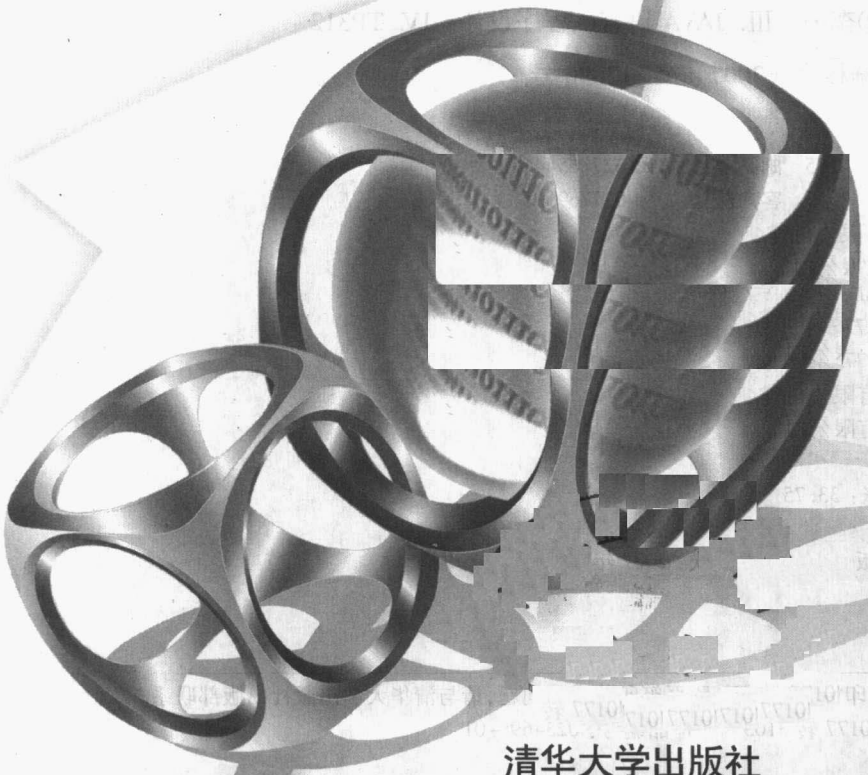
陶 勇 李晓军 编著

附光盘

清华大学出版社

# Hibernate ORM 最佳实践

陶 勇 李晓军 编著



附光盘

清华大学出版社

北京

## 内 容 简 介

本书站在客观评价 Hibernate 这门持久化技术的角度，来分析 Hibernate 的基本构成、知识点及实现原理，汇总业界及作者在 Hibernate 应用方面的实践经验，分清 Hibernate 的优势和劣势，及可代替的解决方案，展示 Hibernate 对象关系映射技术的真谛，总结 Hibernate 在项目开发中的最佳实践经验，使得读者在入门领悟 Hibernate 理论知识的同时，能了解 Hibernate 的优势和劣势，做到扬长避短，而不是盲目膜拜。

本书以 Hibernate 3 为基础，由浅入深介绍 Hibernate OR 映射的基本理论知识及最佳实践经验。内容主要分三部分，第一部分是对象关系映射技术的基本理论及 Hibernate 对象映射技术的基本知识点，第二部分讲解 Hibernate 技术的查询和事务、缓存等高级性能，第三部分是 Hibernate 业界应用的最佳实践经验，包括如何使用 Spring 集成 Hibernate，使用 DAO 模式透明化持久层设计，Hibernate 在 Web 应用中的最佳实践和 Hibernate 性能优化和如何使用 iBATIS、JdbcTemplate 来替代 Hibernate。

本书特别适合使用 Hibernate 进行企业开发的开发人员作为参考书籍，可以将书中的 Hibernate 最佳实践经验应用于项目中，也适合 Hibernate 的入门读者作为入门书籍。



**图书在版编目（CIP）数据**

Hibernate ORM 最佳实践 / 陶勇，李晓军编著. —北京：清华大学出版社，2007.9
ISBN 978-7-302-15757-1

Ⅰ. H…　Ⅱ. ①陶… ②李…　Ⅲ. JAVA 语言－程序设计　Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2007）第 112890 号

**责任编辑**：刘　霞
**责任校对**：张　剑
**责任印制**：何　芊
**出版发行**：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
c－service@tup.tsinghua.edu.cn
**社 总 机**：010-62770175　**邮购热线**：010-62786544
**投稿咨询**：010-62772015　**客户服务**：010-62776969
**印 刷 者**：北京鑫丰华彩印有限公司
**装 订 者**：三河市新茂装订有限公司
**经　　销**：全国新华书店
**开　　本**：185×260　**印　　张**：33.75　**字　　数**：799 千字
附光盘 1 张
**版　　次**：2007 年 9 月第 1 版　　**印　　次**：2007 年 9 月第1 次印刷
**印　　数**：1～4000
**定　　价**：59.00 元

### 为什么要写作本书

OR Mapping 对象关系映射技术是 Java 领域和.NET 领域中面向对象编程中的一个重大技术进步，在 Java 领域中已经得到广泛的应用。Hibernate 是目前 Java 领域中最受欢迎的 OR 映射开源框架，Hibernate 技术的出现使得程序员可以摆脱 JDBC 编程编写繁琐 SQL 语句的麻烦，可以使得程序员能将更多的精力应用于业务逻辑的设计。

而 Hibernate 使用的对象关系映射技术所带来的性能问题受到业界的普遍怀疑，其使用的查询机制、持久化、对象关联等技术使得 Hibernate 的使用更加方便，但是同时由于使用不当也会带来严重的性能瓶颈问题。在 Hibernate 刚出现时，其便利性和优越性得到很多项目架构人员的青睐，但也正是由于其性能和不可预知问题而使项目架构人员望而却步。

本书正是站在客观评价这门持久化技术的角度，来分析 Hibernate 的基本构成，知识点及实现原理，汇总业界及作者在 Hibernate 应用方面的实践经验，分清 Hibernate 的优势和劣势，及可代替的解决方案，展示 Hibernate 对象关系映射技术的实质，总结 Hibernate 在项目开发中的最佳实践经验。使得读者能掌握 Hibernate 的基本理论知识，带领读者进入 Hibernate 对象关系映射技术视野的同时，更进一步掌握 Hibernate 的特性，学会在项目中如何运用，做到有能力驾驭运用 Hibernate 这门出色的对象关系映射技术，能了解 Hibernate 的优势和劣势，做到扬长避短，而不是顶礼膜拜。

### 本书内容组织

本书基于 Hibernate 3，以 Hibernate 的知识点为主线，由浅入深，从对象-关系映射基础知识，到 Hibernate 的上手；从对象映射配置、对象关系映射、集合映射、继承映射、对象操作、对象查询、事务、缓存、到 Spring 继承 Hibernate，DAO 透明持久化及 Hibernate 在 Web 应用中的实践和 Hibernate 的性能优化，知识点逐步深入。

在知识点讲解过程中使用恰当的实例进行举证，从实例的运行结果中分析验证理论，这种表达方式将非常有利于读者的阅读理解。另外，在讲解过程中，结合实践经验给出这些知识点的运用技巧和注意事项，融入各知识点的最佳实践经验，深入每一个知识点的应用，对读者可能会产生疑惑的内容做精辟讲解，并纳入业界最佳解决方案，给出在不同场合下如何使用的建议。

书中融入大量 Hibernate 业界最佳实践经验，给出二层架构及 Web 三层架构中的持久化实现方案以及 Hibernate 的性能调优，使得读者可以轻松驾驭 Hibernate 应用于自己的项目实践中。本书的目的就是使读者在了解 Hibernate 的基本知识点的同时，能合适地运用 Hibernate 技术于不同场景的项目，更好地应用这门数据库持久化技术。

### 本书主要特色

现在市面上已经有一些 Hibernate 相关的书籍，但是本书和它们相比，有如下不同之处：

（1）本书以 Hibernate 3 最新版本为基础讲解，兼顾 Hibernate 3 的最新知识点，及 JDK5 的特性，并在讲解过程中不断回顾 Hibernate 2 的知识点。本书既适用于基于 Hibernate 3 开发，也适用于基于 Hibernate 2 开发的读者使用。

（2）本书内容以最佳实践为线索指导组织内容，不是一路高歌 Hibernate 的好处，而是站在客观的角度来评价这门技术，对 Hibernate 各方面知识点的优势和劣势做出客观的评价，并汇集业界运用 Hibernate 的最佳实践经验，融入书中内容，给出不同场合下该如何使用的建议。

（3）本书在知识点讲述过程中，充分使用实例进行举证，书中内容均给出实例代码，并给出实例的运行结果，对结果进行充分的分析。通过这些恰当的举例，读者很容易理解这些知识点。

（4）本书内容覆盖范围广，本书内容囊括 Hibernate，XDoclet，DAO，Spring，iBATIS 等知识点。众所周知，Hibernate 不是单独使用的，需要整合到其他项目中去，对于如何整合到两层应用模式或 Web 应用的三层模式中，本书给出了最佳实践的方案。



（5）Spring 出现后，其提供的对 Hibernate 的操作支持使得 Hibernate 的使用更方便，在实际项目中已经大量使用 Spring 和 Hibernate 来结合使用，本书对于如何集成 Spring 和 Hibernate 做了充分的讲述

（6）本书知识点包含大量的业界使用 Hibernate 的最佳实践经验、Hibernate 性能优化、周边应用集成等知识，这些知识点，能充分指导读者将 Hibernate 更好地应用到项目开发中。

### 本书内容结构

本书内容主要分三部分，第一部分是对象关系映射技术的基本理论及 Hibernate 对象映射技术的基本知识点，第二部分讲解 Hibernate 技术的查询和事务、缓存等高级性能，第三部分是 Hibernate 业界应用的最佳实践经验，包括如何使用 Spring 集成 Hibernate，使用 DAO 模式透明化持久层设计，Hibernate 在 Web 应用中的最佳实践和 Hibernate 性能优化和如何使用 iBATIS 等技术来替代 Hibernate。主要内容介绍如下。

第 1 章　主要讲述对象-关系映射的基本理论知识，包括软件分层设计和软件设计模型，对象关系映射的概念和工具。

第 2 章　以一个实例介绍如何开发 Hibernate，讲述开发环境的搭建，Hibernate 的基本构成和操作，开发 Hibernate 的工具，Hibernate Tools 插件的使用。

第 3 章　讲述映射文件的定义，映射文件的结构和生成，以及如何使用 XDoclet 生成映射文件。

第 4 章　讲述对象间的关联关系映射，一对一关联，一对多关联，关联关系的检索策略，组件映射和传播性持久化的操作，以及关联关系映射和操作的最佳实践建议。

第 5 章　讲述集合类映射，Java 中的集合和 Hibernate 的集合关系，映射 Map、Set、

List、Bag 的添加、删除、更新、查询操作和集合排序等集合映射操作的最佳实践建议。

第 6 章　讲述继承映射，Hibernate 下的继承映射关系，类和子类的继承映射和表存储的关系，并给出继承映射的最佳实践建议。

第 7 章　讲述 Hibernate 如何操纵持久化对象，以及对象的生命周期、瞬时对象、持久对象、脱管对象间的关系，给出 Session 清理和 Session 的 save、get、update、delete、saveOrUpdate 的操作方法和实践建议。

第 8 章　讲述 Hibernate 查询，包括条件查询、使用 NativeSQL 查询和 Hibernate 查询的优化、迭代查询、多对象查询、统计查询、绑定参数查询、Hibernate 过滤器、Hibernate 分页等重要知识点。

第 9 章　讲述 Hibernate 的 HQL 查询语言，及 HQL 的语言特点、语法和查询构成，包括 JOIN、查询分组、子查询、批量操作等。

第 10 章　讲述 Hibernate 事务和并发，包括事务边界的概念、JDBC 中的事务、JTA 的事务和 Hibernate 的事务、Hibernate 的并发控制、悲观锁定和乐观锁定、怎样划分事务等最佳实践。

第 11 章　讲述 Hibernate 的缓存，包括缓存的原理，Hibernate 的一级缓存、二级缓存，给出最佳实践，二级缓存和查询缓存的应用。缓存是一个比较难以操作的配置，本章给出精确控制缓存失效的策略，和建议用户大数据量处理时一定要及时清理缓存，并给出了使用第三方 OSCache 进行缓存的使用。

第 12 章　讲述通过 Spring 访问 Hibernate，讲述了使用 Hibernate 的难度，进行会话管理的难度，并给出了 Spring 对 Hibernate 的支持，Spring 事务管理策略，Spring 编程式事务管理和 Spring 声明式事务管理。

第 13 章　讲述使用 DAO 进行透明持久化，给出了基于 DAO 模式的 Hibernate 设计，DAO 设计策略，并建议不要盲目使用 DAO 模式，也不要让应用局限于 Hibernate。

第 14 章　讲述 Web 应用中的 Hibernate，讲述了 Hibernate 在 Web 应用中的层次，如何将 Struts 结合 Hibernate 开发，如何使用 Hibernate 页面分页，如何优化 Web 应用中的 Hibernate 性能。

第 15 章　讲述 Hibernate 的性能优化措施，讲述了 Hibernate 性能优化方向，使用连接池和缓存优化，执行查询优化，优化批量操作，结合多种持久化策略，和如何使用 iBATIS 和 JdbcTemplate 的替代方案。

**本书是否适合您**

进行数据库开发是 J2EE 开发的主要任务，是所有 J2EE 开发程序员必须掌握的技能。而使用 Hibernate 作为数据库桥梁的持久化工具，是目前在 J2EE 领域中最受欢迎，最成熟，使用人员最广的技术，掌握 Hibernate 这门技术是所有从事 J2EE 开发人员的必备功课。

如果您是刚进入 J2EE 领域的开发新手，对 Hibernate 还不太了解，阅读本书，可以使您少走很多弯路。Hibernate 是优越的，但是并不是完美的。在没有任何 Hibernate 使用经验的情况下，如何做到扬长避短，将 Hibernate 最优越的地方使用在项目中，本书的最佳实践经验会帮助您解决这个问题。本书是以 Hibernate 的知识点来组织内容的，所以

您不用担心本书无法入门，正是充分照顾了入门级的读者，做到内容由浅入深，从理论到实践。

如果您已经在开发 Hibernate 应用方面有着丰富的经验，则可以把本书作为您必备的参考书籍。本书融入了业界使用 Hibernate 的最佳实践经验，这些经验和使用的建议在书中的任意知识点均随处可见，可以成熟应用于实际项目开发中，本书正能帮助您解决令人棘手的 Hibernate 性能问题，Web 集成问题和其他的查询等性能问题。本书囊括了 Hibernate 对象关联关系的最佳实践经验，查询的优化技巧，一级缓存和二级缓存的使用，事务隔离，Spring 集成，DAO 持久，Hibernate 性能调优等内容，这些最佳实践技巧，将使您能更大胆地驾驭这门技术。

本书的所有知识点，讲解均使用了充分的实例进行举证，并对这些代码实例和运行结果给予了充分地讲解和分析，阅读这些实例代码，将更有利于读者理解这些知识和实践理论。建议读者在学习 Hibernate 技术的过程中，善于将理论与实践相结合，达到事半功倍的效果。

**本书服务与致谢**

本书光盘包括本书所有已经调试通过的范例源代码程序和所需要的 Java 包和类文件，并在书中给出了详细的代码文件路径。此外，光盘还附赠了 Eclipse 3.2 和 MySQL 5.0 的内容。



本书由创想科技组织策划，由阿里巴巴（中国）网络技术有限公司的资深工程师陶勇、李晓军主笔编写，除此之外，李振捷、温才燚、柯华坤、张琳、张湘华、黄青川、刘威、李晓军、张英男、刘春辉、徐建飞、颜志军、杨鲲鹏、李新峰、姚远、高喆、陈芳、缪勇等为本书的编写也提供了有益的帮助，在此表示衷心地感谢。对于书中内容及编排我们力求精益求精，但错误及疏忽之处在所难免，敬请广大读者批评指正。

编　者

2007 年 3 月

# 目 录

CONTENTS

# 第 1 章　对象-关系映射概述

本章从软件开发的层次结构入手，着重分析数据持久层的技术解决方案；通过对象模型与关系数据模型之间所存在的阻抗不匹配着手，引入对象-关系映射（ORM）的技术背景和原理，并对目前主流的对象-关系映射（ORM）工具做简要介绍；如果读者是初次接触 Hibernate，而希望能够对 Hibernate 有一个较快捷的认识，那么可以直接从第 2 章开始阅读。如果希望进一步地深入了解对象-关系映射（ORM），请阅读本章。

**本章内容主要包括：**

- 软件的分层体系结构
- 软件设计模型
- 对象-关系映射技术背景
- 对象-关系映射工具

## 1.1 分层体系结构

在软件开发中，有两个非常重要的概念：业务逻辑和数据持久化。

可以说，企业应用是围绕着业务逻辑进行开展的，例如产品入库、购物车、货品订单等都是业务逻辑。从业务逻辑的底层实现来看，业务逻辑其实是对业务实体进行组织的过程。这一点对于面向对象的系统才成立，因为在面向对象的系统中，识别业务实体，并制订业务实体的行为是非常基础的工作，而不同的业务实体的组合就形成了业务逻辑。

另一个重要的概念就是数据持久化。持久化就是将业务状态保存到特定的存储设备中，应用软件中的大部分数据都是需要可持久化的，因此，如何对业务数据进行持久化就显得尤其重要。根据业务系统的大小和设计者的选择，数据持久化也有许多途径，如文件、数据库等。目前最为流行的持久化机制是数据库，而最为普遍的是通过关系数据库 RDBMS 进行业务状态的持久化。

为了能够更有效地对应用软件中的各种逻辑进行有效的组织，可使用应用分层技术来实现具体的软件系统。应用分层在计算机领域中有着悠久的历史，计算机的实现中就应用了分层的概念。用于分层的最常见的模型是由国际标准化组织（International Organization for Standardization，ISO）所定义的 OSI 7 层模型（OSI 7 Layer Model），如图 1-1 所示。

该模型定义了一组网络协议：每一层都针对通信的一个具体方面，并且依赖于它下面的层。该模型还使用了基于响应的分层策略，通过特殊的响应将各层关联起来。分层策略也可以基于其他系统特性，如复用等。


图 1-1 OSI 7 层模型（基于响应的分层）

分层的优势在于：

- 上层的逻辑不需要了解所有的底层逻辑，它只需要了解和它邻接的那一层的细节。TCP/IP 协议栈就是通过不同的层对数据进行层层封包的，不同层间的耦合度明显降低。通过严格的区分层次，大大降低了层间的耦合度。
- 某一层次的下级层可以有不同的实现。例如同样的编程语言可以在不同的操作系统和不同的机器中运行。
- 同一个层次可以支持不同的上级层。TCP 协议可以支持 FTP、HTTP 等应用层协议。

可以看到，应用分层可以使层次逻辑更加清晰，并且实现了一定程度的解耦，综合上面的考虑，可把软件应用分为多个层次。

### 1.1.1 层次结构

分层是软件设计者在设计复杂软件系统时最常采用的一种技术。例如，在计算机领域中的分层体系降低了编程语言和操作系统在调用硬件驱动和 CPU 指令集的复杂性。

批处理系统时代（相对计算机历史来说，那已经是比较古老的年代了）人们并不需要过多地考虑分层的概念，通常是写一个程序来操作某些格式的文件（如 ISAM，VSAM 等），很显然，那时候的系统并不需要分层实现。

在 20 世纪 90 年代，基于客户端/服务端（Client/Server）架构的系统不断增加，使得分层的概念开始流行起来，在这个时候，两层结构占有很重要的位置：客户端是一些用户界面和应用代码，而服务端则通常是一个关系数据库，如常见的使用 VB、PowerBuilder 和 Delphi 语言开发的访问数据库的应用程序都属于这种两层结构。这些工具使基于数据的应用程序开发变得非常的容易，只需要通过简单的拖曳和属性填写就可以实现界面组件和数据库之间的连接。例如在银行中应用很广的大型主机/终端方式。

两层的体系结构一直到现在还广泛存在，如果软件应用的主要功能是实现关系数据的展现和简单的更新等操作，那么客户端/服务端结构的软件系统能够非常好地满足要求。但是问题随着业务逻辑的出现接踵而至：如何进行业务规则的处理；数据校验、计

算等诸如此类的问题。这时候，通常是将这些需求实现到客户端，也就是将业务逻辑绑定到了用户界面代码中间。当业务逻辑变得越来越复杂的时候，客户端的代码变得越来越难于维护，更甚者，绑定业务逻辑的客户端代码复制得到处都是，致使简单的需求改变却不得不找遍所有的客户端代码来进行修改。

当然，另外一种方式是将业务逻辑作为存储过程绑定到数据库中间。然而，存储过程同样导致了软件系统对数据库的依赖；人们之所以采用关系数据库是因为 SQL 语言是独立于特定数据库的数据查询标准，这使得更换更合适的数据库厂商变得非常的容易。

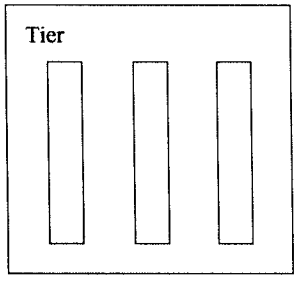
在客户端/服务端应用变得越来越流行的时候，面向对象领域在不断的发展壮大过程中。针对客户端/服务端应用的缺点，面向对象社区提出了三层体系结构：表示层负责处理界面，业务层负责业务逻辑，再加上持久层（数据库或者数据源）。但由于两层架构转变到三层架构并不是那么容易，因此三层体系理论在很长一段时间内没有得以发展。

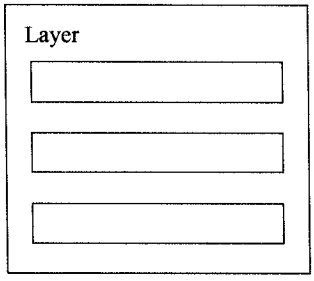
但随着互联网的发展，人们需要将客户端/服务端应用能够通过浏览器进行访问。然而，如果所有的业务逻辑都在客户端，那么客户端业务逻辑必须要重新实现一套来支持基于 WEB 的界面，而一个设计良好的三层体系结构只需要增加一个新的表示层就可以满足这个要求。


**区分 Layer 和 Tier**

Layer 和 Tier 在英文中都有层的概念。当人们在讨论分层的时候，通常混淆了 Layer 和 Tier。那么，如何来划分 Layer 和 Tier 之间的区别呢？

可以直观地通过下面的图示来感受二者的特点：




垂直和水平的划分某种程度上来说只是一个视角问题，其中揭示了物理上和逻辑上的区别。

人们习惯将 Tier 表示为物理上的划分。也就是说，客户端 / 服务端系统通常表示为 two-tier 的系统，它们是物理上的区分：客户端是一个桌面系统，而服务端是服务器。

而 Layer 是表示系统所划分的 Layer，并不需要运行在不同的机器上。业务逻辑层可以运行在桌面系统机器上也可以运行在数据库服务器上。在这种情况下，整个系统有两个节点而有三个 Layer。

除了物理与逻辑上的区分外，Layer 一般还暗示了“下面的 Layer 为上面的 Layer 提供服务”。

请注意，下面提到的层都是 Layer 的概念。

如图 1-2 所示，三层体系结构将应用划分为 3 个层次：

### 1. 表示层

表示层用来处理用户和软件之间的交互，如现在比较常见的如富客户端图形化界面

程序，基于 HTML 和网页浏览器（以下简称浏览器）的 Web 程序。它主要的责任是向用户展现信息以及处理用户请求，例如鼠标单击、内容输入、HTTP 请求等。

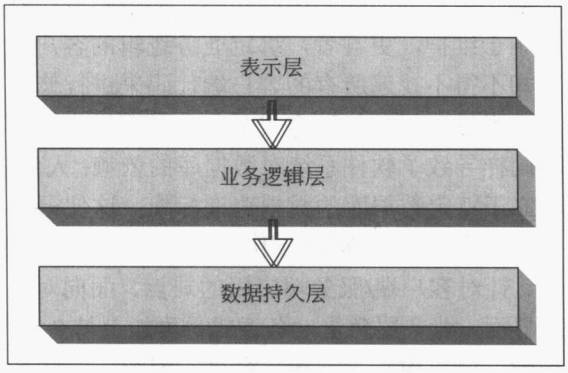



图 1-2 三层体系结构图

### 2. 业务逻辑层

业务逻辑层，又叫做领域逻辑层。其职责包括处理业务逻辑和存储业务数据，校验从表示层传过来的数据，通过表示层提交的命令来执行相应的业务逻辑。

### 3. 数据持久层

数据持久层是用来存取业务状态数据的。数据持久层通过与其他系统进行通信来完成应用的调用。其职责包括事务监控、消息系统、数据源等。在大多数的软件应用中，数据持久层最基本的功能就是存储持久化数据到数据库中，而关系数据库（RDBMS）则是目前最为流行的数据库。



三层体系结构图中箭头所示的方向，表示各个层之间的数据调用和依赖关系。这个依赖暗示了分层系统的特性规则：一层中的组件只能与同一级别中的对等实体或较低级别中的组件交互。

可以看到，业务逻辑层不会访问表示层中的内容，同样，数据层也不会访问业务逻辑中的内容。这个规则使得在同一个基础架构上实现不同的表示层成为可能，同时使得表示层的修改并不会影响到更深层次的实现。而业务逻辑层和数据层之间的关系就没有这么简单了，它依赖于数据层所使用的架构模式，在后面的章节将会详细地阐述。


通常标准的三层体系结构的目的是为了隔离层之间的依赖。例如，业务逻辑层将数据层和表示层完全隔离开来，这样的好处是显而易见的：它实现了表示层和数据层之间的松耦合。然而，更多的情况是表示层直接调用数据层的接口。虽然从理论上来说它并不是标准的三层结构，但是在实际的开发中，这种方式比标准结构能够获得更好的性能效率的提升。

通常，表示层将用户输入的命令传给数据层，数据层读取数据后交给业务逻辑层组织数据，然后再交给表示层把数据展现给用户。当然，如何控制各个层之间的关系以及如何抽象层内的基础特性是每个软件开发者在进行系统架构的时候需要仔细考虑的。

## 1.1.2 分层架构特点

前面已经提到，分层架构的目的就是为了隔离各个层之间的耦合，并划分各个层之间的职责关系。有经验的开发者在进行架构设计时适当地采用分层来搭建基础的应用环境，分层架构具有以下优点：

### 1. 松耦合

由于层与层之间的依赖关系被最大程度地剥离，并且层之间的高内聚，以及层与层之间通过接口交互而剥离了对接口实现的依赖，使得系统解决方案的维护和增强变得更容易。比如，可以将原有基于 Swing/AWT 的 Java 应用切换到 Web 页面，改变表示层的实现而不需要改变业务逻辑层和数据层的实现；又或者，将数据库从 MySQL 切换到 Oracle 时，对表示层的影响减小到最低。

### 2. 伸缩性

这个特性依赖于各个层实现的方式。当一个设计合适的分层系统为了满足业务增长的需要来提高系统的性能时，可以在压力大的层增加机器来增加其负载能力。显而易见的一个例子就是，原有的系统在进行数据库操作时，需要每一个客户端都建立一个数据库连接，而三层架构通过数据库连接池机制，可以利用少量的数据库资源来满足大量的用户请求的需要。另外，通过各个层的集群能够实现大容量的应用服务，将层分布在多个物理层上可以改善可伸缩性、容错和性能。

### 3. 重用性

重用性和松耦合是有联系的，都是为了达到同一个实现能够满足多种应用需求的目的。例如，业务逻辑层能够被 Web、WAP 等多种表示层实现调用；同样，业务逻辑层在调用数据层时并不需要了解其对数据库的依赖。

### 4. 扩展性

通过分层，可以实现新功能的增加而不会影响其他层的实现。只有在影响原有实现的接口时，才有可能影响到各个层之间的关系。同时，具有定义明确的层接口以及交换层接口的各个实现的能力提高了可测试性。

但是，分层架构也带来了一些损失，这主要是以下几个方面：

（1）性能影响。穿越各层（而不是直接调用组件）所需的额外开销会对性能造成不利的影响。例如，客户端访问数据库中的数据时，通常需要经过多个层次，非常耗费性能，如何尽量减少数据库访问是 J2EE 应用系统首要解决的问题。例如，过于严格的分层会禁止表示层与持久层无法直接交互，导致开发表示层复杂的应用程序时可能需要更长的时间。此时如果要弥补由此而带来的性能损失，可以使用松散的分层方法。通过这种方法，较高层可以直接调用较低层。

（2）复杂性。层的使用有助于控制和封装大型应用程序的复杂性，但增加了简单应用程序的复杂性。对较低级别接口的改变可能会渗透到较高级别，尤其是在使用了松散的分层方法的情况下可能性更大。多层架构实际是将以前系统中的显示功能、业务运算功能和数据库功能完全分开，杜绝彼此的耦合与影响，从而实现松耦合和良好的可维护性。