Broadview
www.broadview.com.cn

iBATIS——目前主流的 ORM框架
Java 软件设计师、架构师案头必备参考用书

iBATIS
框架源码剖析

任钢 著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

iBATIS
框架源码剖析

任钢 著

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

内 容 简 介

iBATIS 是一种比较流行的 ORM 框架，本书全面介绍其结构体系并分析其源程序代码，该框架的核心包括两个组件，一个是 iBATIS DAO，另一个是 iBATIS SQL Map。

本书分为三个部分，第一部分介绍 iBATIS 的一些基础知识；第二部分介绍 iBATIS DAO 的框架结构及其实现；第三部分针对 iBATIS 的底层平台 iBATIS SQL Map 进行分析。其中第三部分是主要内容：首先剖析了 SQL Map 是如何读取配置信息的；其次说明了 SQL Map 引擎的实现，勾画出 iBATIS SQL Map 的框架结构，描述其核心实现机制和主要实现步骤；再次说明 SQL Map 如何用来实现数据库处理，包括事务管理、数据库连接池，以及 SQL Map 中 Mapping 的实现，这也是 iBATIS 不同于其他 ORM 框架的独创性实现；最后是一些常用的实现，如 TypeHandler 类型转化和 iBATIS 常用工具的实现。

在源码剖析过程中，本书采用了代码注释、UML 分析和设计、GoF 设计模式抽象和归类、代码跟踪和案例的讲解和说明。目的是让读者全方位地了解 iBATIS 的实现框架和实现手段。一方面让读者理解开发者的思路，另一方面也是帮助读者在实际工作中能应用这些策略、方法和编程技巧。

本书适用于软件设计师、架构师和一些有较好 Java 基础的开发人员，既可以作为 iBATIS 的学习指南，也可以给软件架构师在设计方面进行参考。

图书在版编目（CIP）数据

iBATIS 框架源码剖析 / 任钢著. —北京：电子工业出版社，2010.6
ISBN 978-7-121-10872-3

Ⅰ. ①i… Ⅱ. ①任… Ⅲ. ①关系数据库－数据库管理系统－软件工具 Ⅳ. ①TP311.138

中国版本图书馆 CIP 数据核字（2010）第 087570 号

责任编辑：高洪霞
特约编辑：顾慧芳
印　　刷：北京东光印刷厂
装　　订：三河市皇庄路通装订厂
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
开　　本：787×1092　1/16　印张：32.75　字数：786 千字
印　　次：2010 年 6 月第 1 次印刷
印　　数：3500 册　定价：79.00 元（含光盘 1 张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

搞 IT 技术已经有十多年的历史了，接触 Java 语言也有一定的时间了。为什么到现在才要写这本书呢？几年前我开发过一个 ORM 模型框架，当时的思路居然与 iBATIS 框架有一些类似（可见英雄所见略同）。于是，为了更好地实现这个 ORM 框架，我仔细阅读了 iBATIS 框架的源码。在阅读过程中，由于 iBATIS 框架代码层层叠叠、峰回路转、跌宕起伏，为了理清框架的主要思路和核心实现方式、加快理解速度和加深理解深度，我用 PowerDesigner 画了一些 UML 图，并做了一些阅读笔记和备忘录。一个月下来，基本上从总体架构上了解了 iBATIS 框架的实现。这时候阅读笔记和备忘录已大约积累了好几万字。我想，如果能把这些笔记和备忘录进行系统化、简单化、章节化的整理，就可以给更多 iBATIS 爱好者使用。同样，这些学习心得对软件架构师、软件开发工程师等都非常有价值，所谓它山之石，可以攻玉。于是，我决定写一本关于 iBATIS 框架源码剖析的书籍。而在实际操作中，我觉得在讲述 iBATIS 源码的同时，已经涉及很多关于 ORM 的内容，也有一些 Java 的基础处理和编程技巧，甚至还包括一些经典的设计模式。

在国内介绍和讲述开源软件的书可谓是琳琅满目，不胜枚举。但这些书基本上都归纳为应用型或工具型，更趋向于软件的使用说明或使用指南之类。而且，在全球这么多开源框架代码中，我国做出的贡献还是非常少的。分析原因，主要是我们热衷于拿来主义，直接就用，能解决问题就行。而对于源码，也许只有在使用过程中遇到了障碍，为了解决问题才做一些源码阅读和分析，这也是国内许多人很少去分析开源框架源码的原因，而且介绍开源软件实现的书籍也是凤毛麟角。我写这本书的目的，就是希望在这方面能与大家多分享一些学习心得和体会。

对于开源代码，能读懂并搞明白是一回事。但是理解了源代码，把这些东西用文字表述出来，让别人也能理解却是另一回事。我觉得后者的难度远远大于前者。当然，如果仅仅是简单地介绍 iBATIS 框架，我相信只要有几句话就能说清楚。但是要把一个实现框架说得条理清晰、层次鲜明，这不仅仅要求有一定的技术背景，还要有文字语言的表达和掌控能力。我用一个月时间就基本上搞明白了 iBATIS 框架的内容，但是要把它写出来，的确非常头疼。有的时候要非常细致地去推敲，因为很多琐碎的细节决定了整个框架的核心，这需要有一定的耐心和抽象能力。同时阅读和理解程序源码是一种实践性非常强的工作，所谓读书破万卷，下笔如有神，这是一个真理。但若阅读并理解了几十万行程序代码后，再来编写程序代码，那基本上就可以非常有章法并有一定的深度了。事实上，我国大学目

前计算机教育的水平是基础性质的，对于如何去阅读源码没有相关的课程来进行讲授。对于源码的分析有什么办法、手段和策略，即如何把那些复杂的程序代码用简单的语言表达出来，让别人能迅速地理解并掌握，我国的教育还是做得不够的，也没有专人或学者来搞这方面的研究。但笔者觉得这项工作是非常重要，而且也是很有意义的。事实上，每个开发人员和设计人员在实践中都有自己的一套读取和分析源码方法。当然这也是仁者见仁、智者见智的事。我的方法和手段只是其中之一而已。所以说我在编写这本书时对源码的剖析只是做了一些非常粗浅但是有意义的尝试，以试图弥补国内软件行业在这方面的缺陷。

当然，阅读本书也要具备一定的基础知识，否则，有些术语和解释还是比较难以理解的。

我在本书的编写过程中花费了大量的时间。而且，这些工作都是在业余时间内完成的，每天都要照常上班，只有到了晚上或者节假日，才有闲暇写这些东西。一般技术人员都喜欢新鲜事物，如果说去阅读和理解开源代码，也许还有一些挑战性，大家可能都有兴趣去做这件事情，但是要把这些东西用文字表述出来并给别人讲述，则许多人就不太愿意干了，原因是这是一项非常枯燥的任务，写作的艰辛旁人是很难理解的。很高兴的是我还是坚持下来了，度过了无数个寂寞和孤单的晚上终于把这本书奉献给了读者。

在这期间，我要感谢我的家人对我的理解和支持。我要感谢我的妻子在这期间承担了全部的家务，同时也要感谢我女儿在我撰写书稿的时候没给我添太多的麻烦，她一直没有搞明白爸爸总是待在电脑前做什么。我要感谢我的父母，他们肯定不知道我写的内容，但是无论我做什么，有没有成绩，他们都是一如既往地给予鼓励。我把这里的一切都献给他们。

我还要感谢本书的编辑高洪霞和顾慧芳。没有她们耐心的指导和完善，这本书也许只是一个读书笔记，我也要感谢本书的策划编辑袁金敏，是她给我信心让我继续下去，否则这些资料只能是束之高阁。我还要感谢那些在编写本书作出贡献的所有人，他们都是默默无闻的后台工作者。

当然，由于笔者水平有限，书中的错误和缺点在所难免，希望读者能给予批评和指正。笔者的联系方式：rengang66@sina.com。

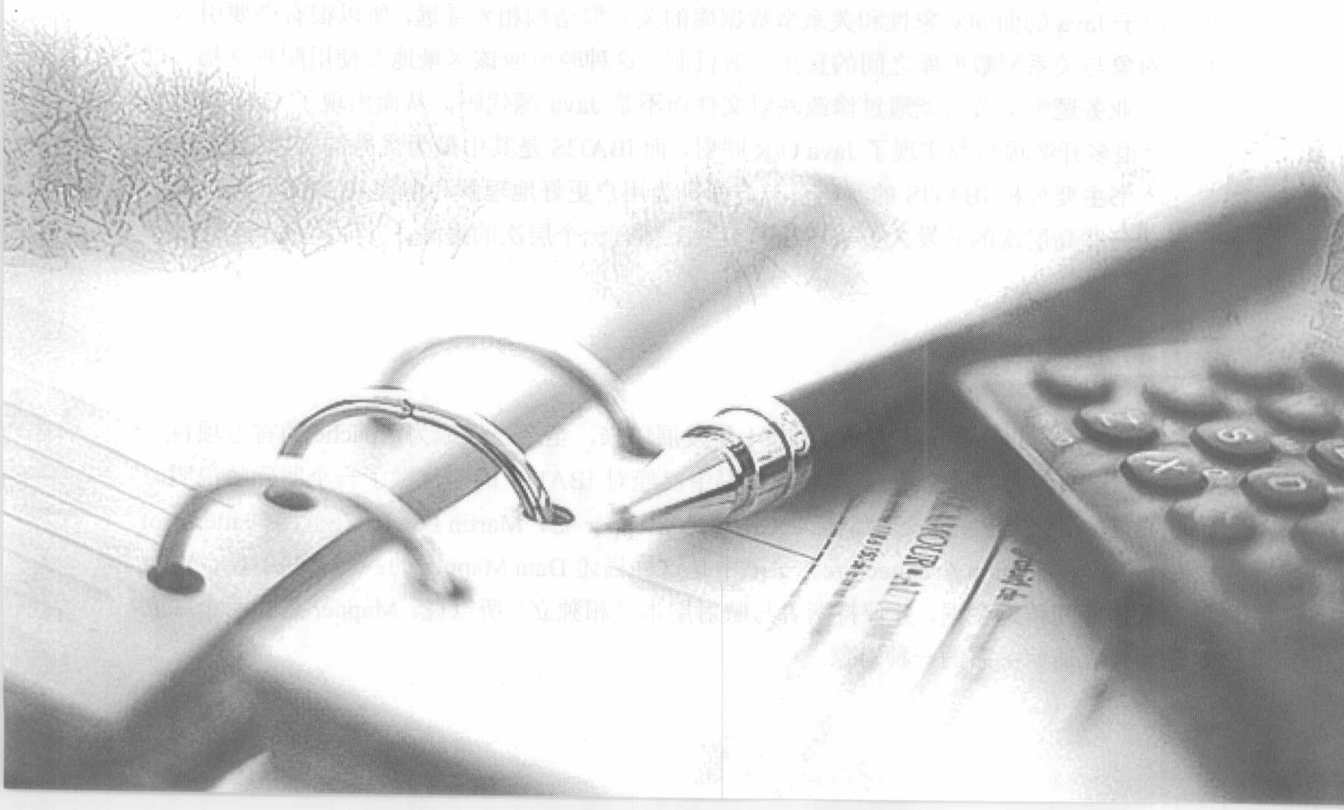
编　者

2009 年 12 月于深圳

目 录

PART 一

第一部分　iBATIS 的基础知识



第 1 章
iBATIS 概述

本章内容：

1．首先对 iBATIS 进行总体性的介绍。

2．对 ORM 模式和持久层进行了分析说明。

3．说明 iBATIS 主要的组件和实现方式。

在信息系统的开发过程中，由于绝大多数业务模型都涉及关系数据库，在采用 Java 作为系统的开发语言时，传统应用开发方法是直接用 JDBC 与数据库交互。但在这种模式下开发和维护工作量都很大并且维护调试也非常不方便，并且一旦业务逻辑稍微有一些变更，就需要大量地更改这些 JDBC 中的 SQL 语句，因此不管是开发还是维护系统都很不方便。由于 Java 的面向对象性和关系型数据库的关系型结构相差甚远，所以很有必要引入一种在对象与关系型数据库之间的直接映射机制。这种映射应该尽量地多使用配置文档，以便今后业务逻辑更改后能通过修改映射文件而不是 Java 源代码，从而出现了 O/R 映射模式。有很多开源项目都实现了 Java O/R 映射，而 iBATIS 是其中最为优秀的实现架构之一。

本书主要分析 iBATIS 的源码，一方面是为用户更好地理解和掌握 iBATIS，另一方面也是让一些高层次的开发人员从理论和实践上都有一个层次的提高。

1.1 iBATIS 概论

iBATIS Database Layer 架构自 2001 年发展以来，至今已经成为 Apache 的官方项目。在 iBATIS 创始人的《iBATIS 实战》一书中已经对 iBATIS 的定位做了一个明确的说明。iBATIS 是一种 Data Mapper，那什么又是 Data Mapper 呢？Martin Fowler 在他的《Patterns of Enterprise Application Architecture》一书中是这样描述 Data Mapper 的：一个映射层，在对象和数据库间传递数据，并保持两者与映射层本身相独立。所以说，Mapper 是在两个独立对象间建立通信关系的一种对象。

通过对程序源码的分析，iBATIS 也具备了 ORM 框架的一些基本特性，但实际上 iBATIS 更像是一个 SQL 工具。同时，iBATIS 不是直接在类与数据表或字段与列之间进行关联，而是把 SQL 语句的参数(parameter)和返回结果(result)映射至实体类(或 JavaBean)。iBATIS 是处于实体类和数据表之间的一个中间层，这使得它在类和数据表之间进行映射时更加灵活，而不需要数据库模型或对象模型（Object Model）的任何修改。我们所说的中间层实际上就是 SQL 映射层，它使得 iBATIS 能够更好地分离数据库和对象模型的设计，这样就相对减少了两者间的耦合。

相对 Hibernate 和 Apache OJB 等“一站式”ORM 对象关系映射解决方案而言，iBATIS 是一种“半自动化”的 ORM 实现。这里要说明一下“全自动化”和“半自动化”在实现 ORM 模式上的区别。

Hibernate 和 Apache OJB 都是对数据库结构提供了较为完整的封装．提供了从 POJO（Plain Old Java Objects 普通 Java 对象）到数据库表的全套映射机制。软件开发人员往往只需定义好了 POJO 到数据库表的映射关系，即可通过 Hibernate 或者 OJB 提供的方法完成持久层操作，软件开发人员甚至不需要对 SQL 的熟练掌握。Hibernate、Apache OJB 会根据制定的存储逻辑，自动生成对应的 SQL 并调用 JDBC 接口去执行。我们把这种模式称为“全自动化”模式。

“半自动化”ORM 框架是相对上述提到的 Hibernate 等提供了全面的数据库封装机制的“全自动化”ORM 实现而言，“半自动化”ORM 框架重点在于 POJO 与 SQL 之间的映射关系。也就是开发人员编写 SQL 语句，通过映射配置文件，将 SOL 所需的参数，以及返回的结果字段映射到指定的 POJO。这些过程全是手工来进行操作。iBATIS 就属于“半自动化”ORM。

“全自动”ORM 机制在大多数情况下有很大的优势。但是，也有一些特定的环境，这种一站式的解决方案却未必是最佳的选择。在现实中，这些特定的环境条件具有如下的特点：

① 数据库系统对于开发人员并不是完全控制的。处于安全考虑，也许只有部分数据或者局部权限开放给开发人员。只对开发团队提供几条 Select SQL 或存储过程以获取所需数据，具体的表结构不予公开。

② 为了提高系统的性能和实现分层开发，必须由数据库层的存储过程来实现所有的业务逻辑部分。

③ 对于一些系统，由于表与表之间主键和外键的约束关系复杂，这造成了生成标准 POJO 对象之间的关系也比较复杂，但是可以通过多种复合 SQL 可以编写出比较简洁高效的语句来实现多表关联查询。

④ 由于系统数据处理量巨大，对性能要求极为苛刻，这往往意味着我们必须通过经过高度优化的 SQL 语句或存储过程才能达到系统性能设计指标。

面对这样的需求。如果再使用 Hibernate 这种数据访问策略来实现对数据库的访问就显得很不明智了。此时“半自动化”的 iBATIS，却刚好解决了这个问题。

当然，iBATIS 的 ORM 模式也并不是没有缺陷。首先，对于开发人员是增加工作量。iBATIS 并不会为开发人员在运行期自动生成 SQL 语句执行，具体的 SQL 语句需要开发人员编写。其次，对开发人员的要求要高一点，毕竟要求开发人员对 SQL 语句要有一定的要求。最后，iBATIS 支持的 SQL 是标准规范的 SQL，可以在所有支持标准 SQL 的数据库系统中移植和运行。但是针对一些对标准 SQL 有扩展的 SQL，如 T-SQL、PL SQL 等，则缺乏对扩展部分的支持。

在 iBATIS 创始人的《iBATIS 实战》一书中，也专门提到了 iBATIS 的不适合环境。其中 iBATIS 不适合的三种环境为：① 当对数据库永远拥有完全控制权；② 当应用程序需要完全动态的 SQL 语句；③ 当数据是非关系数据库时。

1.2 ORM 模型介绍

当 Java 作为一门语言出现时，一个比较重要的工作任务就是要操作数据库。这就不可避免地要涉及数据库的接口。Sun 公司从 1996 年就把 JDBC 加入了 Java 平台的 1.1 版本之中，将其作为 RDBMS 资源管理和数据访问的标准低级抽象层。但是，直接使用 JDBC API 是相当麻烦的，并且开发的效率也比较低。这样，Java ORM 模型就横空出世了。

1.2.1 什么是 ORM

O/R Mapping 全称 Object Relation Mapping，即对象关系映射，把对数据表映射为对象类，将在数据库中直接进行的原始操作演变为对类的属性和方法的操作，而间接更改数据表的数据。

通常，实现 ORM 框架一般包括以下四部分：

① 对映射类进行 CRUD（新增、查询、修改和删除）操作的 API；

② 规定 Object 与 Relational 之间的映射规则，一般采用 metadata 进行表示；

③ 规定类和类属性相关的查询规则；

④ 实现 ORM 中对数据库操作的事务管理。

实现 ORM 框架也有很多其他的扩展功能，包括缓存处理、分布式事务管理、多种数据库之间的无缝迁徙等等。

1.2.2 ORM 的实现方式

关系数据模型则基于数学原理，特别是集合论的原理，这些原理都能在数学理论上得到完全的证明。而对象模型基于程序设计的一些原则，类同于一种解决问题的工具，并没有得到公理或数学理论的支持。面向对象设计的目标是通过把一个业务过程分解成具有状

态和行为的对象来进行建模的业务过程，而关系数据库的设计目标是规范化数据以消除数据冗余度。两种不同的理论基础导致对象模型与关系数据模型之间的阻抗不匹配。面向对象模型和关系数据模型的阻抗不匹配使得进行对象关系映射时存在着许多问题。关系数据库不支持诸如类、继承、封装和多态等面向对象的概念。对象关系映射是在对象与数据库之间进行映射的一种技术，对象间的关系、类及其属性必须以某种方式映射为关系数据库中的数据库关系、表和字段。

针对对象模型与关系数据模型间的阻抗不匹配，人们提出了一些解决方案来实现对象模型向关系数据库模型的映射，由于数据库是以二维表为基本管理单元的，所以在使用关系数据库的面向对象软件应用中对象模型最终是由二维表以及表间关系来描述的，而这其实就是一个类与数据库表的变换过程。从某些方面看来，在对象与关系数据库表的行之间有着很多共同点。对象是类的实例，它的内部数据的结构以及其他一些特征由某个类定义。类似地，关系数据库的脚本定义了数据库表的结构，例如表包含哪些字段等。类的实例与数据的行以相似的方式存储数据。通常一个类可以映射为一个或一个以上的关系数据库的表，类属性将映射成关系数据库中的字段，而对象间的关系可以通过使用关系数据库中的外键来维护。对象关系映射的解决方案一般是把每个对象映射到数据库表的单个行上，这一行通常来自一个表，也可以由一个表的连接操作产生。

1. ORM 中对象与数据库表之间的映射机制

ORM 的实现方式也就是 O/R Mapping 的映射机制，一般有以下几种情况。

（1）类属性和数据库的数据表与列建立一种随机的映射关系

也就是说，对象类和数据库表并非一一对应，同时对象类中的属性也不是与数据库表中的列并非一一对应。一个类属性可对应 1 或多个实体表的字段。同样，一个实体表可以对应 1 个或多个实体类的属性。这种实现模式主要还是根据业务逻辑来划分对象的，一方面一个业务逻辑类可以获取一个数据库表的一部分字段，同时还要获取另一个或几个数据库表的部分或全部字段。另一方面一个数据库表可以映射成为多个对象，分别应用在多个业务实体类中。这种方式的好处有两点，一是映射的对象结构简单、易于使用，二是避免每次构造对象的时候传送大量的不相关的数据。

当然，基于这种模式，存在实现对象在数据库中的唯一标识的问题。其实这有两种解决方案，第一种解决方案还是采用数据库主键来实现唯一标识。如果一个实体类映射的是一个数据库表，可以采用数据库表的主键来形成实体类的唯一标识。如果一个实体类映射的是多个数据库表，可以把多个数据库表的主键组合形成一个实体类的唯一标识。第二种解决方案是采用无业务意义的字段 OID 作为各个实体对象的主键。这样 OID 也作为类与数据库映射时的对象的唯一标识。

（2）实体类和数据库表一一映射

这是一种最简单的实现模式，即数据库表与对象一对一，即一个数据库表映射为一个 Java 对象。所有的表字段（field）映射为 Java 对象的属性（attribute）。但是不同层次的实

体类映射到数据表时，应根据数据库表的关系来进行。这样也有三种模式。

① 一个类层次对应一个数据表

这里所说的一个类层次，指的是父类及其所有子类。将父类和子类中有持久性需求的属性设置为同一数据表的字段。此方法实现起来简单，并且较好地支持多态。因为不同的子类可以用一个标志位加以区分。子类实例的转换比较容易实现，但是，类与数据库之间的耦合程度高。由于子类所特有的属性被所有类层次中的对象所共有，数据库中冗余字段较多。

② 一个实体类对应一个数据表

各个子类所特有的属性，联合从父类中继承的公共属性，构成表的结构。父类不映射为数据库中的实体表，它只作为子类公共属性的载体。这种映射模型使得类属性值的保存和对象还原实现方便。缺点同样是类结构与数据库的耦合程度高，特别在父类属性变列时，所有从此继承下来的子类都需要进行变更。此外，对多态性的支持也较差。子类的角色转换需要在子类对应的数据表之间准确地传递适当的属性值，同时，需要赋予新的 OID。这种情况下，就不如第一种映射模型，用同一个 OID，在同一张表内就可以实现不同子类之间的转换。

③ 一个类对应一个数据表

无论是父类还是子类，只要类中的属性有持久性保存的需要，就将类映射到数据表。子类的表以父类类表的 OID 为外键。在关系数据库中最大程度地实现了类的多态性。与前两者比较，对象与数据库的耦合程度是最低的，某一个类属性的变更引起的表结构变动最少。这种方法的缺点在于生成子类实例时，继承层次多的子类的属性值还原很困难。由于子类的公共属性包含在父类对应的数据表中，当需要单独获取子类实例时，需要从多个数据表中获取数据合成完整的实例。当类的层次较多时，子类的访问可能会成为系统与数据库交互的瓶颈。

（3）实体类和数据库视图映射

第 3 种映射方式是根据数据库视图来构造对象，数据库视图是一种虚拟表，它可以合并多个数据库表，然后再把这个数据库视图映射成一个实体对象。最为常见的是在数据库中体现为主从表的结构，映射成一个实体对象。

2. 类间关系映射为键值

数据库表的约束决定了实体类关系中的关联和聚合。

一对一、一对多的关联是通过在关联的某一方（一般在关联角色多重性≥1 的一方）引用对方的 OID，并根据关系的紧密程度为外键的字段加上非空以及唯一性的约束。在生成相关联的类实例时，可根据外键自动获取另一方的实例。多对多的关联需要创建关联表。关联表是统一的以 OID 作为主键，同时以关联角色双方的 OID 作为联合外键。

聚合关系映射到数据库中与通常所说的主附表结构类似，在聚合关系中的子类对应的数据表中含有指明父类的 OID 的域。特别是强制型聚合（或称之为组合），子类单独存在

是没有意义的，必须与父类同时存在、同时消亡。当然，数据库键值本身只是这一关系的体现。

数据库表之间主键外键关联映射的关系对象模型中存在着三种关系，分别是一对一、一对多、多对多。体现在数据库中分别是一个主键对应一个外键，一个主键对应多个外键，中间表（join table）。

① 一对一：这种情况可以直接映射成在对象之间保持一个引用关系，一个对象持有对另一个对象的引用。具体体现在对象中实现一个方法，该方法的返回值即它的一对一关联对象。如果这种关系是双向的，那么必须在两个对象中都实现这样一个方法，如果关系是单向的，那么由方向性决定在哪个对象中实现。

② 一对多：这种关系有两种类型，分别是关联（association）和聚合（aggregation）。对于关联，仍然体现为引用关系。不同于一对一的是，一方对象持有的是一个集合的引用，该集合中的元素是多方对象，多方对象持有的是一方对象的引用。一方对象实现的方法的返回值是一个集合，集合中的元素是多方对象；多方对象实现的方法的返回值就是一方对象。当然可根据业务需求决定关系的方向性，从而决定在哪个对象中实现对应的方法。对于聚合，需要在一方对象中增加一个集合的属性，该集合中的元素为多方对象。同时一方对象的增、改、删、查也要加入相应的操作以实现多方对象的对应操作。而多方对象的实现可以根据实际需求来决定是否需要实现独立的增、改、删、查方法。

③ 多对多：这种关系可以看作是一个双向的一对多，都把自己看作是一方，对方看作是多方。两个对象分别持有一个集合的引用，集合中的元素即为对方对象。如果将数据库的中间表映射成一个对象，那么可以将多对多关系的两个对象分别实现为对应中间表对象的一对多，即这两个对象都看作是一方，而中间表对象则看作是多方。从而利用一对多的实现方式去实现这种关系。

3. 面向对象操作映射为数据库操作

ORM 框架的一项重要工作就是将对数据库的操作封装成实体类的方法。其好处就在于封装了操作的细节，开发人员根本不用关心如何去连接数据库，如何发送 SQL 语句，如何取各个字段，他只需调用一个 CRUD（Create、Read、Update 和 Delete）方法，该方法完成一系列的底层操作，返回的是一个构造好的对象，然后他就可调用相应的 get 方法取得他所需要的字段的数值。

ORM 框架的方法也就是增、改、删、查等基本操作，而实体对象属性也是由数据库表中的字段所构成的。ORM 框架在构造过程中，实际上把 CRUD 方法，根据对象属性与数据库表的映射字段转化为数据库操作中的 insert、select、update 和 delete 等 SQL 语句。

4. 事务管理

由于 ORM 框架封装了数据库的存储操作，软件开发人员不能处理底层存储细节，没有利用数据库的事务管理机制的可能。可以通过两种办法解决这个问题，一是采用独立的