TURING 图灵程序设计丛书 Java系列

Apress®

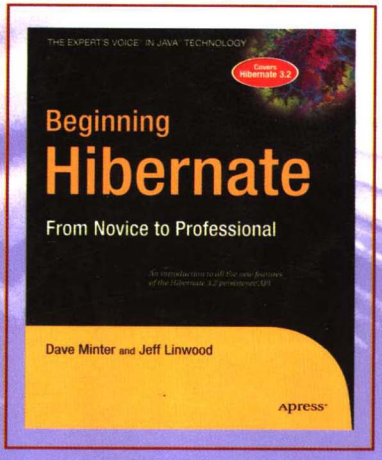
Beginning Hibernate

Hibernate 基础教程

[英] Dave Minter
[美] Jeff Linwood 著

陈剑瓯 等译

- Amazon Hibernate 最佳入门书
- 化繁为简，生动透彻
- 涵盖最新版本 3.2.x



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 Java系列

Hibernate基础教程

Beginning Hibernate

[英] Dave Minter
[美] Jeff Linwood 著

陈剑颐 等译

人民邮电出版社
北 京

图书在版编目（CIP）数据

Hibernate 基础教程 /（英）明特（Minter, D.），（美）林伍德（Linwood, J.）著；陈剑瓯等译. —北京：人民邮电出版社，2008.2
（图灵程序设计丛书）
ISBN 978-7-115-17165-8

I. H… II. ①明…②林…③陈… III. JAVA 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2007）第 174305 号

内 容 提 要

本书深入浅出地介绍了轻量级开源框架 Hibernate 3 的功能，叙述清晰，简洁明了。主要内容包括 Hibernate 的基本体系结构、如何使用 Java 5 注解和基于 XML 的映射文件来表示映射信息，并讨论了 Hibernate Session 对象和 HQL 的使用。本书结合大量实际代码，力图使读者能够更好地学习并掌握 Hibernate 的使用。

本书适合 Java 开发人员阅读。

图灵程序设计丛书

Hibernate 基础教程

◆ 著 [英] Dava Minter [美] Jeff Linwood
译 陈剑瓯 等
责任编辑 傅志红 左艾鑫

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京顺义振华印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：800×1000 1/16
印张：19
字数：454 千字 2008 年 2 月第 1 版
印数：1 – 5 000 册 2008 年 2 月北京第 1 次印刷
著作权合同登记号 图字：01-2007-3147 号

ISBN 978-7-115-17165-8/TP

定价：45.00 元

读者服务热线：(010)88593802 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

前　言

Hibernate 是一款神奇的软件。只需少许经验，再利用 Java 5 注解的强大功能，就可以非常轻松地构建出以数据库作为支撑的复杂系统。你一旦使用了 Hibernate 构建系统，就不愿再采用传统的方式了。

尽管 Hibernate 极其强大，但对于初次接触它的人来说，掌握它也有相当的难度。学习难度大实际上是好事情，因为一旦你征服了 Hibernate，就能充分了解它。当然这需要坚韧不拔的毅力。

本书的目标是向你提供开发 Hibernate 应用程序所应有的基本知识，解释这些基本知识，并且带领你根据这些知识构建一个示例应用程序，以此帮助你顺利地学习 Hibernate。充分理解了基础知识之后，我们将提供其他一些资料。在这个过程中，我们会一直通过示例加以说明，而不是依靠纯粹的理论讲解。

我们希望，当你成为 Hibernate 专家之后，本书作为参考资料仍然能够对你有所帮助。

读者对象

本书假设读者具备良好的 Java 基础知识，并且比较熟悉用 JDBC（Java Database Connectivity）API 进行数据库编程。我们不要求你了解关于 Hibernate 的任何情况，但是既然你买了这本书，那么你应该了解构建基于大型数据库的系统是多么困难，这就是你学习 Hibernate 的动力。

我们的所有示例使用的均为开源软件（主要是 Hibernate API 本身），所以不需要为了开始进行 Hibernate 开发购买任何软件。

本书不是理论教材。我们力求提供丰富的示例，并以实用的方式介绍示例涉及的技术。

对于刚接触 Hibernate API 的新手来说，我们建议至少先依次阅读前 3 章，然后再深入学习后面章节中感兴趣的主题。非常有经验的开发人员，或者熟悉与 Hibernate 相似工具的开发人员，可以直接阅读后面的内容，寻找自己感兴趣的章节。熟悉 Hibernate 的读者可以在附录中找到更多对有趣主题的讨论。

本书结构

本书大致划分为 3 个部分。第 1 章～第 8 章介绍 Hibernate 的基础知识，包括配置、映射文件的创建和基本 API。

第 9 章～第 11 章描述如何使用查询、条件和过滤器以较高级的方式访问持久化信息。

最后，附录讨论不太常用的特性，也就是 Hibernate 核心功能之外的特性。下面简要介绍各章的内容。

- 第1章介绍持久化工具的用途，并且用一个简单示例应用程序片段演示如何应用Hibernate。这章还要介绍关键的术语和概念。
- 第2章讨论配置Hibernate应用程序的基本方法，介绍Hibernate的基本体系结构，讨论如何将Hibernate集成进应用程序。
- 第3章完整地给出第1章的示例应用程序，并介绍创建和运行这个应用程序的完整过程。然后，研究一个略微复杂一点儿的示例，介绍如何从映射文件直接生成数据库模式。
- 第4章深入讨论Hibernate生命周期。它通过介绍核心接口中的方法来讨论生命周期，还介绍关键的术语，并且讨论为什么需要进行级联和延迟加载。
- 第5章解释Hibernate为什么必须维护映射信息，并且讲解关系数据库可以表示的各种关联类型。它还简要讨论可以放在Hibernate映射中的其他信息。
- 第6章介绍在使用Hibernate时如何使用Java 5注解特性来表示映射信息。它为最重要的注解提供了详细的示例，并且讨论标准EJB 3注解和专用的Hibernate 3注解之间的差异。
- 第7章讲解在Hibernate中如何使用基于XML的映射文件表示映射信息。它为所有最常用的映射类型提供了示例，并且解释比较晦涩的映射类型。
- 第8章详细讨论Hibernate Session对象，解释Session对象提供的各个方法。这章还讨论事务、锁和缓存的使用方法，以及如何在多线程环境中使用Hibernate。
- 第9章讨论如何使用Hibernate内置的HQL（Hibernate Query Language）对底层关系数据库进行复杂的查询。
- 第10章介绍Criteria API，这是与第9章讨论的查询语言相似的一种通过编程来查询的方法。
- 第11章讨论如何用过滤器API限制第9章和第10章介绍的查询的结果。
- 附录A介绍了很多非核心特征，在针对初学者的教程中一般不会提到这些特征。这里简要讨论Hibernate EntityManager和EJB 3的使用方法，对版本控制和乐观锁的支持，Dom4J文档模型的直接持久化和获取，映射信息的持久化和获取，以及Hibernate一些微妙的限制和解决这些限制的各种方法，并且给出示例。该附录还讨论了事件和拦截器的使用方法。
- 附录B讨论如何使用Hibernate Tools工具集改进Eclipse开发环境和Ant构建工具，并解释如何定制Ant代码生成任务。
- 附录C讨论如何将Hibernate集成进Spring API。因为将Hibernate集成为Spring应用程序的持久化层比较复杂，所以我们提供一个可以运行的示例，包括完整的bean定义文件，并讨论在Spring MVC环境中管理会话的适当方式，以及在使用Hibernate时Spring如何实施正确的事务边界。
- 附录D讨论与处理现有的Hibernate 2代码相关的一些主题。我们提供让Hibernate 2代码和Hibernate 3代码共存，以及将Hibernate 2代码迁移到Hibernate 3 API的各种方法。

下载代码

读者可以从www.apress.com的Source Code/Download部分下载本书的源代码[①]。请访问Apress网站并下载所有代码。

联系作者

我们欢迎来自读者的反馈。如果你对本书有任何疑问或建议，或者有关于Hibernate的技术问题，或者只是想与我们建立联系，那么可以分别通过dave@paperstack.com和jlinwood@gmail.com给Dave Minter和Jeff Linwood发电子邮件。

致谢

作者Dave和Jeff感谢Apress出版社的工作人员，感谢他们在时间紧迫的压力下仍然能够保持乐观。尤其要感谢我们的项目经理Kylie Johnston帮助本书顺利出版，感谢Damon Larson纠正我们的拼写和语法错误，感谢我们的产品编辑Laura Cheu。还要感谢Steve Anglin邀请我们为Apress出版社写书，感谢Sumit Pal作为技术审阅人所做的贡献。最后，要感谢Hibernate团队开发出这款出色的软件。

Dave感谢Kalani Seymour对他这个坏脾气作者的耐心和关心。他还要感谢他的父母，请他们原谅他太痴迷计算机。他要特别感谢文字编辑Damon Larson，还要感谢Sun Java技术论坛（http://forum.Java.sun.com）上的朋友们提出许多有意思的问题，这些问题给了他很多启发。

Jeff感谢他的家人、朋友和他在Gossamer Group的同事。

① 本书源代码也可从图灵网站本书配套网页下载。——编者注

注：参与本书翻译工作的还有：陈国华、陶灿、马朝晖、马成科、郁桂臣、张斌、何美英。

目　录

第1章 Hibernate 3简介

大多数有一定规模的开发项目都涉及关系数据库。多数商业应用程序的核心是大型的有序信息存储库，比如编目、顾客列表、合同详细信息、出版的文章和建筑设计方案。

由于万维网的出现，对数据库的需求增长了。在线书店和在线报纸的顾客都要使用数据库，尽管他们可能不会意识到这一点。而实际上，在应用程序内部会查询数据库并提供响应。

尽管对这种应用程序的需求增长了，但是创建它们的过程并没有显著地简化。业界已经制订了一些标准，其中最成功的是J2EE（Java 2 Enterprise Edition）的EJB（Enterprise JavaBeans）标准，它针对的是实体bean类的容器管理持久化（CMP）和bean管理持久化（BMP）。然而，关系模型和面向对象模型之间的不匹配，或多或少损害了EJB和其他持久化模型。简单地说，数据库持久化仍然是困难的。

EJB适合某些解决方案，像Hibernate这样的对象-关系映射（Object-Relational Mapping，ORM）适合一部分解决方案，而对于其他情况，通过JDBC（Java Database Connectivity）API直接访问数据的传统方式可能更合适。我们认为Hibernate是很好的首选方式，因为使用它并不妨碍同时使用其他方式。

为了说明Hibernate的一些优势，我们在本章中将给出一个使用Hibernate的简短示例，并且与传统的JDBC方式进行对比。

1.1 POJO

在理想环境中，获得任何Java对象并将它持久化到数据库中都应该是很轻松的。不需要为此编写特殊的代码，也不会有性能损失，而且结果是完全可移植的。

在这样的理想环境中，我们可能会按照代码清单1-1所示的方式执行持久化操作。

代码清单1-1　对象持久化的理想情况

```
POJO pojo = new POJO();
ORMSolution magic = ORMSolution.getInstance();
magic.save(pojo);
```

1

没有特殊情况发生，不需要为类与数据库表的关联做任何额外的工作，也没有性能问题。

Hibernate已经非常接近这个理想目标了，至少与其他替代方式相比已经很方便了，但还是需要创建配置文件，而且要考虑微妙的性能问题。但无论如何，Hibernate实现了它的基本目标——使我们能够在数据库中存储POJO（Plain Old Java Object，普通Java对象）。图1-1说明了Hibernate如何在客户机代码和数据库之间起到桥梁的作用。

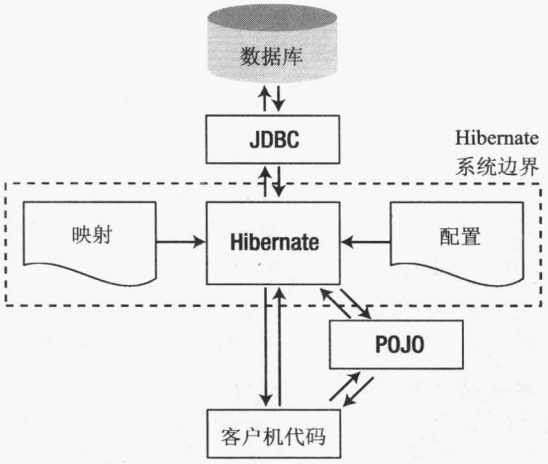


图1-1　Hibernate在Java应用程序中的角色

表示传统Java对象的直接持久化的常用术语是*对象-关系映射*——也就是说，将Java中的对象映射到数据库中的关系实体。

实体bean必须遵守一大套麻烦的命名约定，但是POJO实际上可以是任何Java对象。Hibernate使我们能够对POJO进行持久化，而受到的限制却非常少。代码清单1-2是一个简单的POJO示例，它代表一个消息。

代码清单1-2　本章示例中使用的POJO

```
public class Message {
   private Message() {
   }

   public Message(String messageText) {
      this.messageText = messageText;
   }

   public String getMessageText() {
      return messageText;
   }
   public void setMessageText(String messageText) {
      this.messageText = messageText;
```

2

```
    }

    private String message;
}
```

在这里，Hibernate需要的唯一东西是一个私有的默认构造器。Hibernate要求被存储的所有POJO都要提供一个默认构造器。但是，即使第三方类无法满足这个小小的要求，也是有其他解决方法的（我们将在附录A中进行说明）。

1.2 Hibernate 和对象-关系映射的起源

如果Hibernate是一种解决方案，那么它所要解决的问题是什么呢？其一便是希望简化对象持久化的编程。这是因为在使用JDBC时需要编写相当多的代码，并且要仔细考虑各种规则（比如进行连接管理的那些规则），从而确保应用程序不会造成资源泄漏。即使在知道正确的消息标识符的情况下，用数据库中的数据填充下面的Motd对象也需要大量代码（见代码清单1-3）。

代码清单1-3 获取POJO的JDBC方式

```
public static List getMessages(int messageId) throws MessageException {
    Connection c = null;
    PreparedStatement p = null;
    List list = new ArrayList();

    try {

        Class.forName("org.postgresql.Driver");
        c = DriverManager.getConnection(
                "jdbc:hsqldb:testdb;shutdown=true",
                "hibernate",
                "hibernate");
        p = c.prepareStatement(
                "select message from motd");

        ResultSet rs = p.executeQuery();

        while(rs.next()) {
            String text = rs.getString(1);
            list.add(new Message(text));
        }
        return list;
    } catch (Exception e) {
        log.log(Level.SEVERE, "Could not acquire message", e);
        throw new MotdException(
                "Failed to retrieve message from the database.", e);
    } finally {
        if (p != null) {
            try {
```

```
                p.close();
            } catch (SQLException e) {
                log.log(Level.WARNING,
                        "Could not close ostensibly open statement.", e);
            }
        }

        if (c != null) {
            try {
                c.close();
            } catch (SQLException e) {
                log.log(Level.WARNING,
                        "Could not close ostensibly open connection.", e);
            }
        }
    }
}
```

其中一部分代码可以删减掉，还可以利用各种技术减少那些处理打开连接和日志记录问题的模板代码。但是，随着对象本身变得复杂，从ResultSet中提取对象实例的基本逻辑也会变得非常复杂。一旦对象包含对其他对象的引用（甚至包含对其他对象集合的引用），那么这些“手工”技术的缺陷就更明显了。

1.2.1 EJB 作为持久化解决方案

那么，为什么不单使用EJB来获取数据呢？毕竟，设计实体bean的目的就是表示、存储和获取数据库中的数据。

严格地说，实体bean允许在EJB服务器中进行两种类型的持久化：BMP（Bean-Managed Persistence，bean管理持久化）和CMP（Container-Managed Persistence，容器管理持久化）。在BMP中，bean本身负责执行与存储和获取数据相关的所有SQL——换句话说，它要求bean的作者创建适当的JDBC逻辑，以及代码清单1-3那样的模板代码。而CMP要求容器负责存储和获取bean的数据。那么，CMP为什么不能解决持久化问题呢？下面是几个原因：

- CMP实体bean需要对数据库表的一对一映射。
- 它们不直接支持继承关系。
- 它们的速度慢（至少舆论是这么认为的）。
- 必须有人来决定哪个bean字段映射到哪个表列。
- 它们需要特殊的方法名。如果没有正确地遵守这些命名约定，它们就会失败。
- 实体bean必须驻留在J2EE应用服务器环境中——它们是一种重型的解决方案。
- 无法轻松地把它们抽象成“通用”组件，供其他应用程序使用。
- 它们是不可序列化的。
- 它们很少作为可移植组件出现，很难在其他应用程序中重用——一般来说，必须开发自己的EJB解决方案。

1.2.2 Hibernate作为持久化解决方案

Hibernate可以解决上述的许多问题，或者缓解某些问题，所以我们依次讨论Hibernate对这些问题的影响。

Hibernate不要求开发人员将POJO一一映射到表。可以由多个表列构造出一个POJO，也可以将几个POJO持久化到同一个表中。

Hibernate直接支持类之间的继承关系和各种其他关系。

尽管在Hibernate启动和处理配置文件阶段有一些性能开销，但总地来说，Hibernate被认为是一种很快速的工具。性能问题很难量化，而且实体bean在性能方面声名狼藉，在某种程度上是由于应用程序设计者和部署者所犯的错误，而不是由于它们本身的问题。对待所有性能问题，你都应该进行测试，而不是仅根据舆论趋势做出选择。

在Hibernate中，可以在部署时指定映射，但这不是必须的。EJB解决方案确保应用程序可以跨环境移植，但Hibernate方式致力于减少在新环境中部署应用程序的代价。

Hibernate持久化不要求使用J2EE应用服务器或任何其他特殊环境。因此，对于独立应用程序、客户端应用程序存储以及不能直接使用J2EE服务器的其他环境，它是更合适的解决方案。

Hibernate使用的POJO可以非常轻松自然地实现泛化，从而在其他应用程序中使用。对于Hibernate库没有直接的依赖性，所以POJO可以放进不需要持久化的环境中，也可以使用任何其他的“对POJO友好的”机制对它们进行持久化。

在处理可序列化POJO方面，Hibernate没有任何问题。

有许许多多现有的代码。任何能够持久化到数据库中的Java对象都适合进行Hibernate持久化。因此，Hibernate很自然地成为专用解决方案的替代品，也可以作为还没有集成数据库持久化功能的应用程序的持久化引擎。因此，通过选用Hibernate持久化，就不必亲自为应用程序中的业务对象做出任何特殊的设计决定。

1.2.3 瘦解决方案

舆论常常提到的Hibernate优点之一是，它是一个“瘦（thin）”解决方案。这种说法的问题是，“瘦”这个词非常不正式，所以实际上它并没有说明Hibernate具备的哪些属性使它被归入瘦解决方案。

Hibernate不要求使用应用服务器（而EJB有这种要求[①]）。因此，它能够应用在客户端应用程序中，而在客户端应用程序中EJB是完全不合适的。所以，从这个角度来说，它是瘦的。

另一方面，Hibernate要使用许多支持库。所以，如果考虑一个applet所需的下载时间和硬盘空间的话，Hibernate就显得有点儿臃肿了。但是，在当今这个时代，快速连接和巨大的硬盘空间已经很普及了，所以这不可能成为决定性因素。

① EJB 3中吸收了Hibernate的思想，也提供了不使用应用服务器的选项。——编者注

1.3　Hibernate Hello World 示例

代码清单1-4说明Hibernate所需的模板代码要比代码清单1-3所示的JDBC代码简短得多。

代码清单1-4　获取POJO的Hibernate方式

```
public static List getMessages(int messageId)
   throws MessageException
{
   SessionFactory sessions =
      new Configuration().configure().buildSessionFactory();
   Session session = sessions.openSession();
   Transaction tx = null;
   try {
      tx = session.beginTransaction();

      List list = session.createQuery("from Message").list();

      tx.commit();
      tx = null;
      return list;

   } catch ( HibernateException e ) {
      if ( tx != null ) tx.rollback();
      log.log(Level.SEVERE, "Could not acquire message", e);
      throw new MotdException(
            "Failed to retrieve message from the database.",e);
   } finally {
      session.close();
   }
}
```


即使对于这样的简单示例，真正部署中所需的代码量也会进一步减少——尤其是在应用服务器环境中。例如，一般会在别的地方创建SessionFactory，作为JNDI（Java Native Directory Interface）资源提供给应用程序使用。

注意，填充消息对象所需的手工代码并未消除——相反，它转移到了一个外部配置文件中，这使实现细节与主逻辑分隔开了。

在代码清单1-4给出的Hibernate 3示例中，一些额外的代码提供了JDBC示例不具备的功能（尤其是事务处理和缓存）。

1.4　映射

正如前面指出的，Hibernate需要有某个东西告诉它，哪些表与哪些对象相关（这一信息常常在一个XML映射文件中提供）。有些工具迫使它们的用户付出巨大的XML配置文件，而且缺乏良好的文档，但是Hibernate不会让用户这么累——对于希望映射到数据库中的每个POJO，只需创