第 3 章

Hibernate 初体验

技能目标

- 理解类和表的映射关系
- 理解持久化对象的状态及其转换
- 掌握单表的增删改
- 掌握按主键查询

本章任务

学习本章，完成以下 5 个工作任务。记录学习过程中遇到的问题，可以通过自己的努力或访问 kgc.cn 解决。

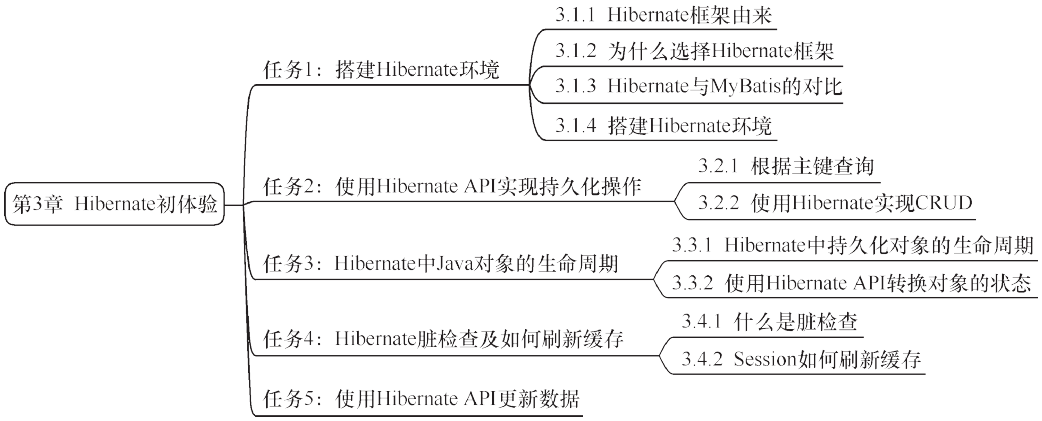
任务 1：搭建 Hibernate 环境

任务 2：使用 Hibernate API 完成持久化操作

任务 3：Hibernate 中 Java 对象的生命周期

任务 4：Hibernate 脏检查及如何刷新缓存

任务 5：使用 Hibernate API 更新数据



任务 1 搭建 Hibernate 环境

关键步骤如下。

- 下载需要的 jar 文件。
- 部署 jar 文件。
- 创建 Hibernate 核心配置文件 hibernate.cfg.xml。
- 创建持久化类和映射文件。

3.1.1 Hibernate 框架由来

1. Hibernate 框架

Hibernate 是数据持久化工具，也是一个开放源代码的 ORM 解决方案。Hibernate 内部封装了通过 JDBC 访问数据库的操作，向上层应用提供面向对象的数据访问 API。

Gavin King 是 Hibernate 的创始人、EJB 3.0 专家委员会成员、JBoss 核心成员之一，也是《*Hibernate in Action*》一书的作者。

2001 年，Gavin King 使用 EJB 的 Entity bean 1.1 时，觉得开发效率太低，于是，便开始试图寻找更好的方案。经过两年多的努力，在 2003 年，Gavin King 和他的开发团队推出了 Hibernate。

Gavin King 成为全世界 Java EE 数据库解决方案的领导者，Hibernate 也成为全世界最流行的开源 ORM 解决方案。

2. Hibernate 是 ORM 解决方案

基于 ORM，Hibernate 在对象模型和关系数据库的表之间建立了一座桥梁。通过 Hibernate，程序员就不需要再使用 SQL 语句操作数据库中的表，而是使用 API 直接操作 JavaBean 对象就可以实现数据的存储、查询、更改和删除等操作，显著降低了由于对象与关系数据库在数据表现方面的范例不匹配而导致的开发成本。

3.1.2　为什么选择 Hibernate 框架

回顾一下 DAO 层代码，以查找所有用户为例，直接使用 JDBC 查询用户的代码如下。

```
List users = new ArrayList();
User user = null;
try {
   Connection conn = DBUtil.getConnection();
   Statement statement = conn.createStatement();
   ResultSet resultSet =
         statement.executeQuery("select * from users");
   while (resultSet.next()) {
         user = new User();
         user.setId(resultSet.getInt(1));
         user.setUserName(resultSet.getString(2));
         user.setPassword(resultSet.getString(3));
         user.setTelephone(resultSet.getString(4));
         user.setRegisterDate(resultSet.getDate(5));
         user.setSex(resultSet.getInt(6));
         users.add(User);
   }
} catch (Exception e) {
   // 省略异常处理代码
} finally {
   DBUtil.close(resultSet, statement, conn);
}
```

用 JDBC 查询返回的是 ResultSet 对象，ResultSet 往往不能直接使用，还需要转换成 List，并且通过 JDBC 查询不能直接得到具体的业务对象。这样在整个查询的过程中，就需要做很多重复性的转换工作。

使用 Hibernate 完成持久化操作，只需要编写如下代码。

```
Session session = HibernateUtil.currentSession();
Query query = session.createQuery("from User");
List<User> users = (List<User>) query.list();
```

HibernateUtil 是一个自定义的工具类，用于获取 Hibernate 的 Session 对象，Session 是 Hibernate 执行持久化操作的核心 API。Hibernate 处理数据库查询时，编写的代码非常简洁。作为查询结果，可以直接获得一个存储着 User 实例的 List 集合实例，能够直接使用，从而避免了烦琐的重复性的数据转换过程。

1．Hibernate 框架的优点

（1）Hibernate 功能强大，是 Java 应用与关系数据库之间的桥梁，较之 JDBC 方式操作数据库，代码量大大减少，提高了持久化代码的开发速度，降低了维护成本。

（2）Hibernate 支持许多面向对象的特性，如组合、继承、多态等，使得开发人员

不必在面向业务领域的对象模型和面向数据库的关系数据模型之间来回切换，方便开发人员进行领域驱动的面向对象的设计与开发。

（3）可移植性好。系统不会绑定在某个特定的关系型数据库上，对于系统更换数据库，通常只需要修改 Hibernate 配置文件即可正常运行。

（4）Hibernate 框架开源免费，可以在需要时研究源代码，改写源代码，进行功能的定制，具有可扩展性。

2. Hibernate 框架的缺点

（1）不适合以数据为中心大量使用存储过程的应用。

（2）大规模的批量插入、修改和删除不适合使用 Hibernate。

3.1.3 Hibernate 与 MyBatis 的对比

Hibernate 与 MyBatis 都属于 ORM 框架，为数据层提供持久化操作的支持。下面从几个方面对 Hibernate 和 MyBatis 做一下比较，也为选择框架提供一些参考依据。

（1）相对于 MyBatis 的“SQL-Mapping”的 ORM 实现，Hibernate 的 ORM 实现更加完善，提供了对象状态管理的功能。Hibernate 对数据操作，针对的是 Java 对象，即使使用 Hibernate 的查询语言（HQL 语句），其书写规则也是面向对象的。

（2）Hibernate 与具体数据库的关联只需要在 XML 中配置即可，Hibernate 开发者不需要关注 SQL 的生成与结果的映射，所有的 HQL 语句与具体使用的数据库无关，便于修改，可移植性好。而 MyBatis 直接使用 SQL 语句，不同数据库之间可能会有差异，修改工作量大，可移植性差。

（3）由于直接使用 SQL 语句，因此 MyBatis 的使用灵活性更高，而 Hibernate 对于关系模型设计不合理、不规范的系统则不适用。在不考虑缓存的情况下，MyBatis 的执行效率也比 Hibernate 高一些。

3.1.4 搭建 Hibernate 环境

在 MyEclipse 中新建工程后，使用 Hibernate，需做以下准备工作，如图 3.1 所示。

图 3.1 Hibernate 环境准备步骤

1. 下载需要的 jar 文件

Hibernate 的官方网站是 http://hibernate.org，在该网站可以下载到较新版本的 Hibernate，其他版本可以通过其托管网站 https://sourceforge.net/projects/hibernate/files/ 下载。推荐下载 hibernate3 目录中的 hibernate-distribution-3.6.10.Final-dist.zip，解压后的目录结构如图 3.2 所示。

注意查看根目录（hibernate-distribution-3.6.10.Final）和 lib\required 目录。在根目录

下存放着 hibernate3.jar，如图 3.3 所示， Hibernate 的接口和类就在这个文件中。

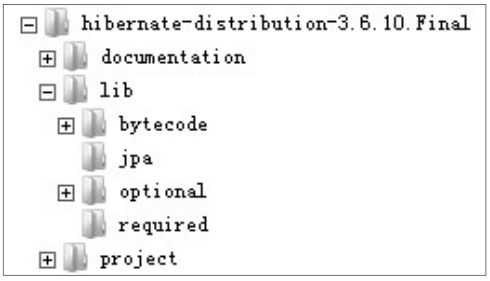


图 3.2　解压后的目录结构

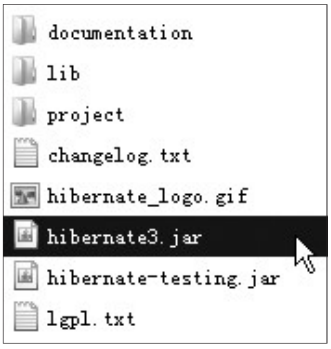


图 3.3　根目录包含的文件和文件夹

Hibernate 会使用到一些第三方类库，这些类库放在了 lib\required 及 lib\jpa 目录下，如图 3.4 所示。这些 jar 文件的作用如表 3-1 所示。

图 3.4　Hibernate 运行时所需要的 jar 文件

表3-1　Hibernate所需jar文件说明

名　　称	说　　明
antlr-2.7.6.jar	语法分析器
commons-collections-3.1.jar	各种集合类和集合工具类的封装
dom4j-1.6.1.jar	XML的读写
javassist-3.12.0.GA.jar	分析、编辑和创建Java字节码的类库
jta-1.1.jar	Java事务API
slf4j-api-1.6.1.jar	日志输出
hibernate-jpa-2.0-api-1.0.1.Final.jar	提供对JPA（Java持久化API）规范的支持

2. 部署 jar 文件

在项目中引用下载好的 hibernate3.jar 文件、lib\required、lib\jpa 目录下的 jar 文件及 Oracle 数据库驱动 jar 文件。

3. 创建 Hibernate 配置文件 hibernate.cfg.xml

Hibernate 配置文件主要用于配置数据库连接和 Hibernate 运行时所需的各种特性。

在工程的 src 目录下添加 Hibernate 配置文件（可在 project\etc 目录下找到示例文件），默认文件名为“hibernate.cfg.xml”。该文件需要配置数据库连接信息和 Hibernate 的参数，如示例 1 所示。

说明

Hibernate 的教学示例使用的是 Oracle 中 scott 用户的部门表、员工表。上机练习使用的是租房系统中的数据表，租房系统将在随后进行介绍。

若没有特别说明，则教学示例和上机练习都在测试类中运行，运行结果在控制台输出。

示例 1

```
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- 数据库 URL-->
        <property name="connection.url">
            jdbc:oracle:thin:@10.0.0.176:1521:orcl
        </property>
        <!-- 数据库用户 -->
        <property name="connection.username">scott</property>
        <!-- 数据库用户密码 -->
        <property name="connection.password">tiger</property>
        <!-- 数据库 JDBC 驱动 -->
        <property name="connection.driver_class">
            oracle.jdbc.driver.OracleDriver
        </property>
        <!-- 每个数据库都有其对应的方言（Dialect）以匹配其平台特性 -->
        <property name="dialect">
            org.hibernate.dialect.Oracle10gDialect
        </property>
        <!-- 指定当前 session 范围和上下文 -->
        <property name="current_session_context_class">thread</property>
        <!-- 是否将运行期生成的 SQL 输出到日志以供调试 -->
        <property name="show_sql”>true</property>
        <!-- 是否格式化 SQL-->
        <property name="format_sql”>true</property>
    </session-factory>
</hibernate-configuration>
```

其中几个常用参数的作用如下。

（1）connection.url：表示数据库 URL。jdbc:oracle:thin:@10.0.0.176:1521:orcl 是 Oracle 数据库的URL。其中jdbc:oracle:thin:@是固定写法，10.0.0.176是IP地址，1521是端口号，orcl 是数据库实例名。

（2）connection.username：表示数据库用户名。

（3）connection.password：表示数据库用户密码。

（4）connection.driver_class：表示数据库驱动。oracle.jdbc.driver.OracleDriver 是 Oracle 数据库的驱动类。

（5）dialect：用于配置 Hibernate 使用的数据库类型。Hibernate 支持几乎所有的主流数据库，包括 Oracle、DB2、MS SQL Server 和 MySQL 等。org.hibernate.dialect.Oracle10gDialect 指定当前数据库类型是 Oracle 10g 及以上版本。

（6）current_session_context_class：指定 org.hibernate.context.CurrentSessionContext.currentSession() 方法得到的 Session 由谁来跟踪管理。thread 指定 Session 由当前执行的线程来跟踪管理。

（7）show_sql：如果设置为 true，则程序运行时在控制台输出 SQL 语句。

（8）format_sql：如果设置为 true，则程序运行时在控制台输出格式化后的 SQL 语句。

提示

因为 Hibernate 的配置属性较多，可在 hibernate-distribution-3.6.10.Final 的 documentation\manual\zh-CN\pdf 目录中查看 hibernate_reference.pdf 的第 3 章 3.4 节，以了解可选的配置属性。

完成了 Hibernate 的配置文件 hibernate.cfg.xml，接下来就要准备持久化类和映射文件了。

4. 创建持久化类和映射文件

持久化类是指其实例状态需要被 Hibernate 持久化到数据库中的类。在应用的设计中，持久化类通常对应需求中的业务实体。Hibernate 对持久化类的要求很少，它鼓励采用 POJO 编程模型来实现持久化类，与 POJO 类配合完成持久化工作是 Hibernate 最期望的工作模式。Hibernate 要求持久化类必须具有一个无参数的构造方法。

下面首先以 Oracle 中 scott 用户的部门表为例，定义部门持久化类，添加一个无参数的构造方法。注意该类实现了 java.io.Serializable 接口，这并不是 Hibernate 所要求的，为了在将持久化类用于数据传输等用途时能够对其实例正确执行序列化操作，建议实现该接口。

部门持久化类 Dept.java 的代码如示例 2 所示。

示例 2

```
public class Dept implements Serializable {
    /* 字段 */
    private Byte deptNo;
```

```
    private String deptName;
    private String location;
    public Dept() {
    }
    // 省略 getter/setter 方法
}
```

Dept 持久化类有一个 deptNo 属性，用来唯一标识 Dept 类的每个实例。deptNo 属性又称为 id 属性。在 Hibernate 中，这个 id 属性被称为对象标识符（Object Identifier，OID），一个 Dept 实例和 DEPT 表中的一条记录对应。

创建持久化类后，还需要“告诉”Hibernate，持久化类 Dept 映射到数据库的哪个表，以及哪个属性对应到数据库表的哪个字段，这些都要在 Dept 类的映射文件 Dept.hbm.xml 中配置。Dept.hbm.xml 的代码如示例 3 所示。

注意

在 Hibernate 中，映射文件通常与对应的持久化类同名，并以“.hbm.xml”作为后缀。

示例 3

```
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="cn.hibernatedemo.entity.Dept" table="'DEPT'">
        <id name="deptNo" type="java.lang.Byte" column="'DEPTNO'" >
            <generator class="assigned"/>
        </id>
        <property name="deptName" type="java.lang.String" column= "'DNAME'"/>
        <property name="location" type="java.lang.String">
            <column name="'LOC'"></column>
        </property>
    </class>
</hibernate-mapping>
```

经验

如果表名或字段名是数据库关键字，或包含空格等特殊字符，可以使用反单引号（'）进行约束。为了避免不必要的错误，建议使用反单引号（'）对数据库表名和字段名统一进行约束。

示例 3 中 Dept.hbm.xml 定义了 Dept 类到数据库表 DEPT 的映射，其中各元素的含义如下。

- class：定义一个持久化类的映射信息。常用属性如下。
  - name 表示持久化类的全限定名。
  - table 表示持久化类对应的数据库表名。
- id：表示持久化类的 OID 和表的主键的映射。常用属性如下。
  - name 表示持久化类属性的名称，和属性的访问器相匹配。
  - type 表示持久化类属性的类型。
  - column表示持久化类属性对应的数据库表字段的名称，也可在子元素column中指定。

注意

这里所指的持久化类属性的名称，是指符合 JavaBean 命名规范的属性名称，即通过 getter 和 setter 访问器得到的默认属性名称。如无特别说明，书中涉及的属性名称均属此类。

- generator：id 元素的子元素，用于指定主键的生成策略。常用属性及子元素如下：
  - class 属性用来指定具体主键生成策略。
  - param 元素用来传递参数。示例 3 使用的主键生成策略是 assigned，不需要配置 param 元素。

常用的主键生成策略如下。

（1）assigned：主键由应用程序负责生成，无须 Hibernate 参与。这是没有指定 <generator> 元素时的默认生成策略。

（2）increment：对类型为 long、short 或 int 的主键，以自动增长的方式生成主键的值。主键按数值顺序递增，增量为 1。

（3）identity：对 SQL Server、DB2、MySQL 等支持标识列的数据库，可使用该主键生成策略生成自动增长主键，但要在数据库中将相应的主键字段设置为标识列。

（4）sequence：对 Oracle、DB2 等支持序列的数据库，可使用该主键生成策略生成自动增长主键，通过子元素 param 可传入数据库中序列的名称，语法如下。

```
<generator class="sequence">
    <param name=”sequence”> 序列名 </param>
</generator>
```

（5）native：由 Hibernate 根据底层数据库自行判断采用何种主键生成策略，即由使用的数据库生成主键的值。

- property：定义持久化类中的属性和数据库表中的字段的对应关系。常用属性如下。
  - name 表示持久化类属性的名称，和属性的访问器相匹配。
  - type 表示持久化类属性的类型。
  - column表示持久化类属性对应的数据库表字段的名称，也可在子元素column中指定。
- column 元素：用于指定其父元素代表的持久化类属性所对应的数据库表中的字

段。其常用属性如下。

- name 表示字段的名称。
- length 表示字段的长度。
- not-null 设定是否不能为 null，设置为 true 表示不能为 null。

映射文件定义完毕，还需要在配置文件 hibernate.cfg.xml 中声明，如示例 4 所示。

示例 4

```
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- 省略其他配置 -->
        <!-- 映射文件配置，注意文件名必须包含其相对于 classpath 的全路径 -->
        <mapping resource="cn/hibernatedemo/entity/Dept.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

通过前面的学习，了解了 Hibernate 框架及如何搭建 Hibernate 环境。接下来为租房系统搭建 Hibernate 环境。

技能训练

租房系统是一个 B/S 架构的信息发布平台，包括两种角色：非注册用户和注册用户。其主要功能如下。

（1）发布房屋信息（注册用户）。

（2）浏览房屋信息（注册用户与非注册用户）。

（3）查看房屋详细信息（注册用户与非注册用户）。

（4）查询房屋信息（注册用户与非注册用户）。

（5）修改房屋信息（注册用户）。

（6）删除房屋信息（注册用户）。

系统使用 Oracle 数据库实现，请按以下描述创建数据表，如图 3.5 所示。

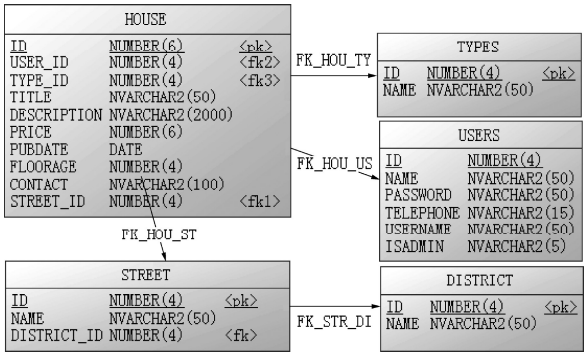


图 3.5　租房系统数据库表及关系

图 3.5 描述了租房系统中的 5 张表，以及它们之间的关系。下面通过表 3-2 至表 3-6 对这 5 张表进行说明。

表3-2　用户表结构

表名：USERS（用户表）			
字　段　名	字 段 说 明	数 据 类 型	说　　明
ID	用户编号	NUMBER(4)	主键
NAME	用户名	NVARCHAR2(50)	不允许为空
PASSWORD	密码	NVARCHAR2(50)	
TELEPHONE	电话	NVARCHAR2(15)	
USERNAME	姓名	NVARCHAR2(50)	
ISADMIN	是否是管理员	NVARCHAR2(5)	

表3-3　房屋类型表结构

表名：TYPE（房屋类型表）			
字　段　名	字 段 说 明	数 据 类 型	说　　明
ID	类型编号	NUMBER(4)	主键
NAME	类型名称	NVARCHAR2(50)	不允许为空

表3-4　区县表结构

表名：DISTRICT（区县表）			
字　段　名	字 段 说 明	数 据 类 型	说　　明
ID	区县编号	NUMBER(4)	主键
NAME	区县名称	NVARCHAR2(50)	不允许为空

表3-5　街道表结构

表名：STREET（街道表）			
字　段　名	字 段 说 明	数 据 类 型	说　　明
ID	街道编号	NUMBER(4)	主键
NAME	街道名称	NVARCHAR2(50)	不允许为空
DISTRICT_ID	所属区县编号	NUMBER(4)	外键，引用区县表主键

表3-6　房屋信息表结构

表名：HOUSE（房屋信息表）			
字　段　名	字 段 说 明	数 据 类 型	说　　明
ID	房屋信息编号	NUMBER(6)	主键
TITLE	标题	NVARCHAR2(50)	

续表

表名：HOUSE（房屋信息表）			
字 段 名	字段说明	数据类型	说 明
DESCRIPTION	描述	NVARCHAR2(2000)	
PRICE	出租价格	NUMBER(6)	
PUBDATE	发布时间	DATE	
FLOORAGE	面积	NUMBER(4)	
CONTACT	联系人	NVARCHAR2(100)	
USER_ID	用户编号	NUMBER(4)	外键，引用用户表主键
TYPE_ID	类型编号	NUMBER(4)	外键，引用房屋类型表主键
STREET_ID	街道编号	NUMBER(4)	外键，引用街道表主键

数据表中字符串类型字段被定义为 NVARCHAR2。NVARCHAR2 能根据定义的长度存储相应的汉字个数，避免了字母和汉字一起存储时数据长度的混乱。如果数据库的字符集是支持中文的字符集（如 GBK、UTF-8 等），那么字符串类型字段也可以采用 VARCHAR2 类型。例如，VARCHAR2(20 char) 代表可以保存 20 个汉字，或者 VARCHAR2(40) 代表 40 个字节，也可以保存 20 个汉字。

上机练习 1——为租房系统搭建 Hibernate 环境

➢ 需求说明

为租房系统搭建 Hibernate 环境。

提示

（1）在 MyEclipse 中创建工程，导入 Hibernate 所需的 jar 文件。

（2）创建 Hibernate 配置文件 hibernate.cfg.xml。

（3）创建用户表对应的持久化类 User 和映射文件 User.hbm.xml。

任务 2 使用 Hibernate API 实现持久化操作

关键步骤如下。

➢ 使用 get()、load() 方法查询部门表数据。

➢ 使用 save()、delete() 方法对部门表数据进行增删改。

为工程准备了 Hibernate 环境后，就可以通过 Hibernate API 操纵数据库。Hibernate 内部也是采用 JDBC 来访问数据库的。图 3.6 和图 3.7 展示了通过 JDBC API 及 Hibernate API 访问数据库的差异。

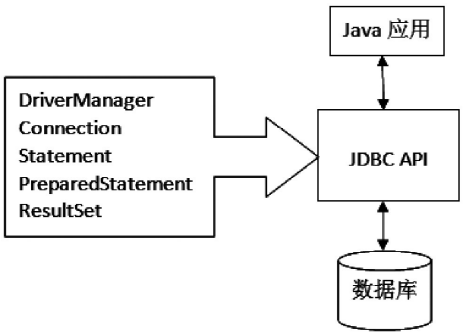


图 3.6　通过 JDBC API 访问数据库

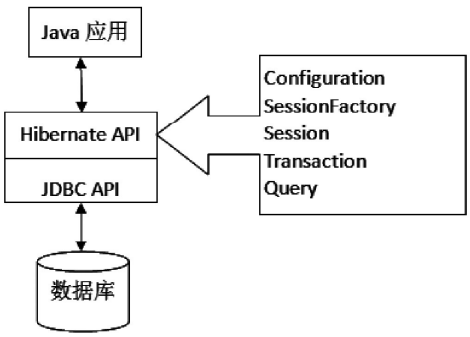


图 3.7　通过 Hibernate API 访问数据库

使用 Hibernate 操作数据库包括 7 个步骤。

（1）读取并解析配置文件及映射文件。

```
Configuration conf = new Configuration().configure();
```

根据默认位置的 Hibernate 配置文件中的信息，构建 Configuration 对象。Configuration 对象负责管理 Hibernate 的配置信息。

（2）依据配置文件和映射文件中的信息，创建 SessionFactory 对象。

```
SessionFactory sf = conf.buildSessionFactory();
```

Configuration 对象会根据当前的数据库配置信息，构造 SessionFactory 对象。SessionFactory 对象一旦构造完毕，Configuration 对象的任何变更将不会影响已经创建的 SessionFactory 对象。如果 Hibernate 配置信息有改动，那么需要基于改动后的 Configuration 对象重新构建一个 SessionFactory 对象。

（3）打开 Session。

```
Session session = sf.getCurrentSession();     // 或者使用 sf.openSession();
```

SessionFactory 对象负责创建 Session 对象。

Session 是 Hibernate 持久化操作的基础。Session 作为贯穿 Hibernate 的持久化管理器的核心，提供了众多持久化方法，如 save( )、delete( )、update( )、get( )、load( ) 等。通过这些方法，即可透明地完成对象的增删改查（CRUD）。

（4）开始一个事务。

```
Transaction tx = session.beginTransaction();
```

（5）数据库操作。

```
session.save(user);          // 保存操作
```

（6）结束事务。

```
tx.commit();                 // 提交事务
```

或

```
tx.rollback();              // 回滚事务
```

（7）如果是通过 SessionFactory 的 openSession( ) 方法获取的 Session 对象，则需关闭 session。

```
session.close();
```

如果在 Hibernate 配置文件中将参数 current_session_context_class 设置为 thread，并采用 SessionFactory 的 getCurrentSession( ) 方法获得 Session 对象，则不需要执行 session.close( ) 方法，因为通过这种方式获得的 Session 对象，会在关联的事务结束（提交或回滚）时自动关闭。

经验

在项目开发过程中，通常使用工具类来管理 SessionFactory 和 Session，参考代码如下：

```
public class HibernateUtil {
    private static Configuration configuration;
    private final static SessionFactory sessionFactory;

    // 初始化 Configuration 和 SessionFactory
    static {
       try {
           configuration = new Configuration().configure();
           sessionFactory = configuration.buildSessionFactory();
        } catch (HibernateException ex) {
           throw new ExceptionInInitializerError(ex);
        }
    }

    private HibernateUtil() {}

    // 获取 Session 对象
    public static Session currentSession() {
        return sessionFactory.getCurrentSession();
    }
}
```

在此工具类中，采用 SessionFactory 的 getCurrentSession( ) 方法获取 Session 对象，结合 Hibernate 配置文件中的以下设置：

```
<property name="current_session_context_class">thread</property>
```

可以在多线程的应用环境中获得线程安全的 Session 对象。多线程情况下共享 Session 是不安全的，通过以上配置，在每个执行的线程中首次调用 getCurrent-

Session( )方法时，会为该执行线程创建并保持一个Session对象。其后，该线程在执行中再次调用getCurrentSession( )方法，只会返回和该线程绑定的那个Session对象。这就保证了每个执行线程都使用自己独立的Session对象，并且保证了在任何情况下获取Session对象时都拥有统一的方法调用风格。

值得注意的是，这种方式获得的 Session 对象，在关联的事务结束（提交或回滚）时，会自动关闭并和当前执行线程解绑，无须显式调用相关代码，从而自动执行了对当前执行线程的清理工作，为使用该线程处理另一个用户请求做好准备。

如无特别说明，本书中的示例都将采用该工具类来管理 Session。

3.2.1　根据主键查询

在进行修改或删除操作时，应先加载对象，再执行修改或删除操作。Hibernate 提供了两种方法按照主键加载对象：get( ) 和 load( )。

- Object get(Class clazz, Serializable id)。
- Object load(Class clazz, Serializable id)。

虽然两个方法都能够加载对象，但它们是有区别的。下面以部门表为例，通过示例 5 和示例 6 讲解它们的部分区别。

get( ) 方法加载部门对象的代码如示例 5 所示。


按主键
查询数据

示例 5

DeptDao 中的关键代码：

```
public class DeptDao {
    public Dept get(Serializable id) {
        // 通过 Session 的 get() 方法根据 OID 加载指定对象
        return (Dept) HibernateUtil.currentSession().get(Dept.class, id);
    }
}
```

业务层的关键代码：

```
public class DeptBiz {
    private DeptDao deptDao = new DeptDao();

    public Dept findDeptById(Byte id) {
        Transaction tx = null;
        Dept result = null;
        try {
            tx = HibernateUtil.currentSession().beginTransaction(); // 开启事务
            result = deptDao.get(id); // 调用 DAO 方法，根据 OID 加载指定 Dept 对象
            tx.commit(); // 提交事务
```

```
            } catch (HibernateException e) {
                e.printStackTrace();
                if (tx != null)
                    tx.rollback(); // 回滚事务
            }
            return result;
        }
        // 省略 getter/setter 及其他方法
}
```

测试方法的关键代码：

```
// 1. 加载数据操作
Dept dept = new DeptBiz().findDeptById(new Byte("10"));
// 2. 输出数据
System.out.println(dept.getDeptName());
```

在示例 5 中，使用 Session 的 get( ) 方法查询主键为 10 的部门信息。如果数据表中没有主键为 10 的数据，get( ) 方法返回的是 null。

load( ) 方法加载数据如示例 6 所示。

示例 6

DeptDao 中的关键代码：

```
public class DeptDao {
    public Dept load(Serializable id) {
        // 通过 Session 的 load( ) 方法根据 OID 加载指定对象
        return (Dept) HibernateUtil.currentSession().load(Dept.class, id);
    }
}
```

业务层的关键代码：

```
public class DeptBiz {
    private DeptDao deptDao = new DeptDao();

    public Dept findDeptById(Byte id) {
        Transaction tx = null;
        Dept result = null;
        try {
            tx = HibernateUtil.currentSession().beginTransaction(); // 开启事务
            result = deptDao.load(id); // 调用 DAO 方法，根据 OID 加载指定 Dept 对象
            // 输出结果，与调用 get() 方法时不同，须在会话关闭前测试查询效果
            // 原因会在后续章节中有关延迟加载的内容中进行分析
            System.out.println(result.getDeptName());
            tx.commit(); // 提交事务
        } catch (HibernateException e) {
```

```
            e.printStackTrace();
            if (tx != null)
                tx.rollback(); // 回滚事务
        }
        return result;
    }
    // 省略 getter/setter 及其他方法
}
```

测试方法的关键代码：

```
// 加载数据操作
Dept dept = new DeptBiz().findDeptById(new Byte("10"));
```

在示例 6 中，使用 Session 的 load( ) 方法查询主键为 10 的部门信息。如果数据表中没有主键为 10 的数据，程序运行到 result.getDeptName( ) 时会抛出异常，如图 3.8 所示。

```
org.hibernate.ObjectNotFoundException: No row with the given identifier exists:
 [cn.hibernatedemo.entity.Dept#10]
  at org.hibernate.impl.SessionFactoryImpl$2.handleEntityNotFound(SessionFactoryImpl.java:419)
  at org.hibernate.proxy.AbstractLazyInitializer.checkTargetState(AbstractLazyInitializer.java:154)
  at org.hibernate.proxy.AbstractLazyInitializer.initialize(AbstractLazyInitializer.java:143)
  at org.hibernate.proxy.AbstractLazyInitializer.getImplementation(AbstractLazyInitializer.java:174)
  at org.hibernate.proxy.pojo.javassist.JavassistLazyInitializer.invoke(JavassistLazyInitializer.java:190)
  at cn.hibernatedemo.entity.Dept_$$_javassist_0.getDeptName(Dept_$$_javassist_0.java)
  at cn.hibernatedemo.test.T1.m3(T1.java:73)
```

图 3.8　使用 load( ) 方法抛出的异常信息

当使用 Session 的 get( ) 方法时，如果加载的数据不存在，则 get( ) 方法会返回一个 null；但是使用 load( ) 方法，若加载的数据不存在，则会抛出异常。这是 get( ) 方法和 load( ) 方法的区别之一，两个方法的其他区别会在后续的内容中介绍。

在后续编码过程中，DAO 层的代码中会频繁出现对 HibernateUtil.currentSession( ) 方法的调用，为了简化编码，不妨定义一个 DAO 的基类，对该方法调用进行封装，参考代码如下。

```
public class BaseDao {
    public Session currentSession() {
        return HibernateUtil.currentSession();
    }
}
```

在此基础上，其他 DAO 类就可以简化获取 Session 对象的编码。

```
public class DeptDao extends BaseDao {
    public Dept get(Serializable id) {
        return (Dept) currentSession().get(Dept.class, id);
    }
}
```

3.2.2 使用 Hibernate 实现 CRUD

1. 使用 Hibernate 实现增加部门记录

示例 7

DeptDao 中的关键代码：

```
public class DeptDao extends BaseDao {
    public void save(Dept dept) {
        currentSession().save(dept); // 保存指定的 Dept 对象
    }
    // 省略其他 DAO 方法
}
```

业务层中的关键代码：

```
public class DeptBiz {
    private DeptDao deptDao = new DeptDao();

    public void addNewDept(Dept dept) {
        Transaction tx = null;
        try {
            tx = deptDao.currentSession().beginTransaction(); // 开启事务
            deptDao.save(dept); // 调用 dao 保存 Dept 对象的数据
            tx.commit(); // 提交事务
        } catch (HibernateException e) {
            e.printStackTrace();
            if (tx != null)
                tx.rollback(); // 回滚事务
        }
    }
    // 省略 getter/setter 和其他业务方法
}
```

测试方法中的关键代码：

```
// 构建测试数据
Dept dept = new Dept();
dept.setDeptNo(new Byte("11"));
dept.setDeptName(" 测试部 ");
dept.setLocation(" 东区 ");
// 保存新部门信息
new DeptBiz().addNewDept(dept);
```

2. 使用 Hibernate 实现部门的修改和删除

下面学习如何使用 Hibernate 修改和删除数据。对于 Hibernate 这种 ORM 工具，操作都是针对对象的。要修改和删除数据，首先要获得数据，然后才能修改和删除数据，

如示例 8 和示例 9 所示。

示例 8

DeptDao 中的关键代码：

```
public class DeptDao {
    // 通过 Session 的 get( ) 或 load( ) 方法加载指定对象
    public Dept load(Serializable id) {
        return (Dept) currentSession().load(Dept.class, id);
    }
    // 省略其他 DAO 方法
}
```

业务层中的关键代码：

```
public class DeptBiz {
    private DeptDao deptDao = new DeptDao();

    public void updateDept(Dept dept) {
        Transaction tx = null;
        try {
            tx = deptDao.currentSession().beginTransaction(); // 开启事务
            // 通过 get() 或 load() 方法加载要修改的部门对象
            Dept deptToUpdate = deptDao.load(dept.getDeptNo());
            // 更新部门数据
            deptToUpdate.setDeptName(dept.getDeptName());
            deptToUpdate.setLocation(dept.getLocation());
            tx.commit(); // 提交事务
        } catch (HibernateException e) {
            e.printStackTrace();
            if (tx != null)
                tx.rollback(); // 回滚事务
        }
    }
    // 省略 getter/setter 和其他业务方法
}
```

测试方法中的关键代码：

```
// 构建测试数据
Dept dept = new Dept();
dept.setDeptNo(new Byte(“11”));
dept.setDeptName(" 质管部 "); // 发生变化的属性
dept.setLocation(" 东区 ");
// 更新部门信息
new DeptBiz().updateDept(dept);
```

在使用 Hibernate 修改数据时，首先要加载对象，然后才能修改对象的属性，最后提交事务。Hibernate 会生成并执行修改的 SQL 语句，其中的原理在后续的小节中详细说明。

示例 9

DeptDao 中的关键代码：

```
public class DeptDao extends BaseDao {
    // 通过 Session 的 get( ) 或 load( ) 方法加载指定对象
    public Dept load(Serializable id) {
        return (Dept) currentSession().load(Dept.class, id);
    }

    public void delete(Dept dept) {
        currentSession().delete(dept); // 删除指定的 Dept 对象
    }
    // 省略其他 DAO 方法
}
```

业务层中的关键代码：

```
public class DeptBiz {
    private DeptDao deptDao = new DeptDao();

    public void deleteDept(Byte id) {
        Transaction tx = null;
        try {
            tx = deptDao.currentSession().beginTransaction(); // 开启事务
            // 通过 get() 或 load() 方法加载要删除的部门对象
            Dept deptToDelete = deptDao.load(id);
            deptDao.delete(deptToDelete); // 删除部门数据
            tx.commit(); // 提交事务
        } catch (HibernateException e) {
            e.printStackTrace();
            if (tx != null)
                tx.rollback(); // 回滚事务
        }
    }
    // 省略 getter/setter 和其他业务方法
}
```

测试方法中的关键代码：

```
new DeptBiz().deleteDept(new Byte("11"));
```

与修改类似，删除时也需要先加载数据。在使用 Hibernate 编写持久化代码时，业务不需要再有数据库表、字段等概念。从面向业务领域对象的角度，要删除的是某个业

务对象。以面向对象的方式编写代码是 Hibernate 持久化操作接口的一个设计理念。

需要注意的是，操作一定要在事务环境中完成。

注意

所谓删除一个持久化对象，并不是从内存中删除这个对象，而是从数据库中删除相关的记录，这个对象依然存在于内存中，只是状态变为瞬时状态。有关“瞬时状态”的概念会在下文中进行介绍。

技能训练

上机练习 2——在租房系统中实现用户表的增删改查操作

➢ 需求说明

在上机练习 1 搭建的环境中，使用 Hibernate 实现对用户的增加、修改、删除和查询操作，要求按用户编号查询指定的用户。

任务 3　Hibernate 中 Java 对象的生命周期

关键步骤如下。

➢ 使用 Hibernate API 转换对象的状态。

3.3.1　Hibernate 中持久化对象的生命周期

当应用通过调用 Hibernate API 与框架进行交互时，需要从持久化的角度关注应用对象的生命周期。持久化生命周期是Hibernate中的一个关键概念，正确地理解生命周期，可以更好地了解 Hibernate 的实现原理，掌握 Hibernate 的正确用法。Hibernate 框架通过 Session 来管理 Java 对象的状态，在持久化生命周期中，Java 对象存在以下 3 种状态。

1. **瞬时状态**（Transient）

瞬时状态又称临时状态。如果 Java 对象与数据库中的数据没有任何的关联，即此 Java 对象在数据库中没有相关联的记录，此时 Java 对象的状态为瞬时状态。Session 对于瞬时状态的 Java 对象是一无所知的，当对象不再被其他对象引用时，它的所有数据也就丢失了，对象将会被 Java 虚拟机按照垃圾回收机制处理。

2. **持久状态**（Persistent）

当对象与 Session 关联，被 Session 管理时，它就处于持久状态。处于持久状态的对象拥有数据库标识（数据库中的主键值）。那么，对象是什么时候与 Session 发生关联的呢？第一种情况，通过 Session 的查询接口、get( ) 方法或者 load( ) 方法从数据库中加载对象时，加载的对象是与数据库表中的一条记录关联的，此时对象与加载它的 Session 发生关联。第二种情况，对瞬时状态的对象调用 Session 的 save( )、saveOrUpdate( ) 等

方法时，在保存对象数据的同时，Java 对象也会与 Session 发生关联。对于处于持久状态的对象，Session 会持续跟踪和管理它们，如果对象的内部状态发生了任何变更，Hibernate 会选择合适的时机（如事务提交时）将变更同步到数据库中。

3. **游离状态**（Detached）

游离状态又称脱管状态。处于持久状态的对象，脱离与其关联的 Session 的管理后，就处于游离状态。处于游离状态的对象，Hibernate 无法保证对象所包含的数据与数据库中的记录一致，因为 Hibernate 已经无法感知对该对象的任何操作。Session 提供了 update( )、saveOrUpdate( ) 等方法，将处于游离状态的对象的数据以更新的方式同步到数据库中，并将该对象与当前的 Session 关联。这时，对象的状态就从游离状态重新转换为持久状态。

3.3.2 使用 Hibernate API 转换对象的状态

在 Hibernate 应用中，不同的持久化操作会导致对象状态的改变。图 3.9 描述了对象状态的转换。

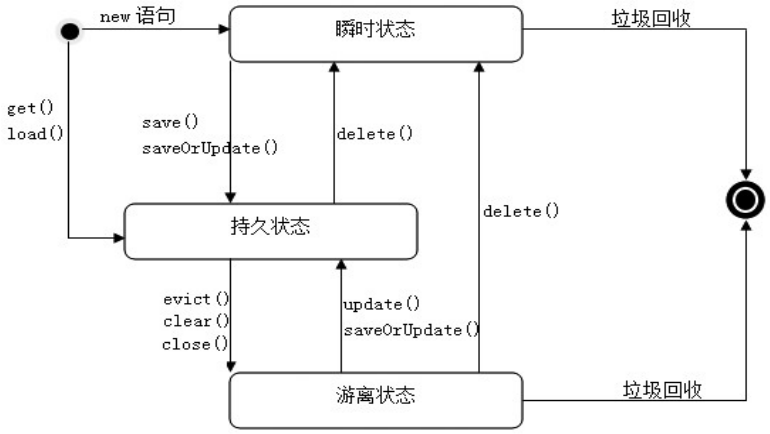


图 3.9 对象状态的转换

●表示开始 ◉表示结束

1. **瞬时状态转为持久状态**

使用 Session 的 save( )、saveOrUpdate( ) 等方法保存对象后，该对象的状态由瞬时状态转换为持久状态。

使用 Session 的 get( ) 或 load( ) 方法获取对象，该对象的状态是持久状态。

2. **持久状态转为瞬时状态**

执行 Session 的 delete( ) 方法后，对象由原来的持久状态变为瞬时状态，因为此时该对象没有与任何的数据库数据关联。

3. **持久状态转为游离状态**

执行 Session 的 evict( )、clear( ) 或 close( ) 方法，对象由原来的持久状态转为游离状态。

4. 游离状态转为持久状态

执行 Session 的 update( ) 或 saveOrUpdate( ) 方法后，对象由游离状态转为持久状态，再次与当前 Session 相关联。

5. 游离状态转为瞬时状态

执行 Session 的 delete( ) 方法，对象由游离状态转为瞬时状态。

提示

处于瞬时状态或游离状态的对象不再被其他对象引用时，会被 Java 虚拟机按照垃圾回收机制处理。

技能训练

上机练习 3——输出对象的状态

➢ 需求说明

指出以下两段代码各个执行阶段中对象状态的变化过程。(提示：可以在代码中补充输出语句，在控制台打印出执行完每一行代码时对象所处的状态。)

代码 1：

```
try {
    //  省略部分代码
    session = sessionFactory.getCurrentSession();
    //  开始一个事务
    tx = session.beginTransaction();
    //  获取用户对象
    User user = (User) session.load(User.class, new Integer("1001"));
    //  修改用户信息
    user.setUsername("rose");
    //  提交事务
    tx.commit();
} catch (HibernateException e) {
    e.printStackTrace();
    if (tx != null)
    tx.rollback(); //  回滚事务
}
```

代码 2：

```
try {
    // 省略部分代码
    // 打开 session
    session = sessionFactory.getCurrentSession();
    // 开始一个事务
```

```
        tx = session.beginTransaction();
        // 获取用户对象
        User user = (User) session.load(User.class, new Integer("1000"));
        // 持久化操作
        session.delete(user);
        // 提交事务
        tx.commit();
    } catch (HibernateException e) {
        e.printStackTrace();
        if (tx != null)
            tx.rollback(); // 回滚事务
    }
```

任务 4 Hibernate 脏检查及如何刷新缓存

Session 是 Hibernate 向应用程序提供的持久化操纵的主要接口，它提供了基本的保存、更新、删除和加载 Java 对象的方法。Session 具有一个缓存，可以管理和跟踪所有持久化对象。在某些时间点，Session 会根据缓存中对象的变化来执行相关 SQL 语句，将对象发生的变化同步到数据库中，换句话说就是将数据库同步为与 Session 缓存一致，这一过程称为刷新缓存。

3.4.1 什么是脏检查

在 Hibernate 中，数据前后发生变化的对象，称为脏对象，如以下代码所示。

```
tx = session.beginTransaction();
//  获取部门对象，dept 对象处于持久状态
Dept dept = (Dept) session.load(Dept.class, new Byte("11"));
//  修改后，部门信息和之前不同，此时 dept 对象成为所谓的“脏对象”
dept.setDname(" 质管部 ");
//  提交事务
tx.commit();
```

以上代码中 dept 对象处于持久状态，当 dept 对象被加入 Session 缓存中时，Session 会为 dept 对象的值类型的属性复制一份快照。操作中，dname 属性发生改变，dept 对象即成为脏对象。在事务提交时，Hibernate 会对 Session 中持久状态的对象进行检测，即比较 dept 对象的当前属性与它的快照，以判断 dept 对象的属性是否发生了变化，这种判断称为脏检查。如果对象发生了改变，则 Session 会根据脏对象的最新属性值来执行相关的 SQL 语句，将变化更新到数据库中，以确保内存中的对象数据与数据库中的数据一致。

3.4.2　Session 如何刷新缓存

需要注意的是，当 Session 缓存中对象的属性发生变化时，Session 并不会立即执行脏检查和执行相关的 SQL 语句，而是在特定的时间点，即刷新缓存时才执行。这使得 Session 能够把多次变化合并为一条或者一批 SQL 语句，减少了访问数据库的次数，从而提高了应用程序的数据访问性能。

在默认情况下，Session 会在以下时间点刷新缓存。

（1）应用程序显式调用 Session 的 flush( ) 方法时。

Session 的 flush( ) 方法进行刷新缓存的操作，会触发脏检查，视情况执行相关的 SQL 语句。

（2）应用程序调用 Transaction 的 commit( ) 方法时。

commit( ) 方法会先调用 Session 的刷新缓存方法 flush( )，然后向数据库提交事务。

在提交事务时执行刷新缓存的动作，可以减少访问数据库的频率，尽可能缩短当前事务对数据库中相关资源的锁定时间。

任务 5　使用 Hibernate API 更新数据

关键步骤如下。

- 使用 update( )、saveOrUpdate( )、merge( ) 方法更新数据。

Hibernate 中的 Session 提供了多种更新数据的方法，如 update( )、saveOrUpdate( )、merge( ) 方法。

（1） update( ) 方法，用于将游离状态的对象恢复为持久状态，同时进行数据库更新操作。当参数对象的 OID 为 null 时会报异常。

（2） saveOrUpdate( ) 方法，同时包含了 save( ) 与 update( ) 方法的功能，如果传入参数是瞬时状态的对象，就调用 save( ) 方法；如果传入参数是游离状态的对象，则调用 update( ) 方法。

（3） merge( ) 方法，能够把作为参数传入的游离状态对象的属性复制到一个拥有相同 OID 的持久状态对象中，通过对持久状态对象的脏检查实现更新操作，并返回该持久状态对象；如果无法从 Session 缓存或数据库中加载到相应的持久状态对象，即传入的是瞬时对象，则创建其副本执行插入操作，并返回这一新的持久状态对象。无论何种情况，传入对象的状态都不受影响。例如，修改 12 号部门的信息。

DEPT 数据表中 12 号部门的原有信息如下：

DEPTNO：12　　DNAME：质管部　　LOC：西区

将该部门的信息修改为：

DEPTNO：12　　DNAME：开发部　　LOC：西区

使用 merge( ) 方法实现，如示例 10 所示。

示例 10

映射文件 Dept.hbm.xml：

```
<hibernate-mapping>
    <class name="cn.hibernatedemo.entity.Dept" table="'DEPT'"
        schema="scott" dynamic-update="true">
      <id name="deptNo" column="'DEPTNO'" type="java.lang.Short">
      <!-- 为了避免主键生成器 assigned 造成的干扰，这里把主键生成器
          替换为 increment -->
        <generator class="increment" />
     </id>
     <property name="deptName" type="java.lang.String" column="'DNAME'"/>
     <property name="location" type="java.lang.String" column="'LOC'"/>
  </class>
</hibernate-mapping>
```

DeptDao 中的关键代码：

```
public class DeptDao extends BaseDao {
    public Dept merge(Dept dept) {
        return (Dept) currentSession().merge(dept);
    }
    // 省略其他 DAO 方法
}
```

业务层中的关键代码：

```
public class DeptBiz {
    private DeptDao deptDao = new DeptDao();

    public Dept mergeDept(Dept dept) {
        Transaction tx = null;
        Dept persistentDept = null;
        try {
            tx = deptDao.currentSession().beginTransaction(); // 开启事务
            // 合并 dept 的数据或者保存 dept 的副本，返回持久状态对象
            persistentDept = deptDao.merge(dept);
            tx.commit(); // 提交事务
        } catch (HibernateException e) {
            e.printStackTrace();
            if (tx != null)
                tx.rollback(); // 回滚事务
        }
        return persistentDept;
    }
    // 省略 getter/setter 及其他业务方法
}
```

测试方法中的关键代码：

```
// 构建测试数据
Dept dept = new Dept();
dept.setDeptNo(new Short("12")); // 游离状态，去掉本行代码则为临时状态
dept.setDeptName(" 开发部 ");
dept.setLocation(" 西区 ");
// 合并游离状态 dept 的数据或者保存临时状态 dept 的副本
new DeptBiz().mergeDept(dept);
```

执行以上程序后，Hibernate 执行以下 SQL 语句：

```
select * from dept where deptno=?
update dept set dname=? where deptno=?
```

可以看出，使用 merge( ) 方法，Hibernate 会根据 OID 加载对应的 Dept 类对象。在 Dept.hbm.xml 映射文件中，为 <class> 标签配置 dynamic-update="true"，作用是只修改发生变化的属性。

综上所述，如果当前 Session 缓存中没有包含具有相同 OID 的持久化对象（如打开 Session 后的首次操作），可以使用 update( ) 或 saveOrUpdate( ) 方法；如果想随时合并对象的修改而不考虑 Session 缓存中对象的状态，可以使用 merge( ) 方法。

技能训练

上机练习 4——在租房系统中修改用户信息

➢ 需求说明

使用 Session 接口的 saveOrUpdate( )、merge( ) 方法修改用户信息，并体会两个方法的区别。

本章总结

- Hibernate 是一个基于 ORM 的持久化框架，对 JDBC 操作进行了封装，提高了持久化层的开发效率。
- Hibernate 提供了完整的 ORM 实现，针对 Java 对象实现数据库操作，支持更多面向对象的特性，可移植性好。
- 使用 Hibernate 需要创建 Hibernate 的配置文件及持久化类的映射文件。
- 本章涉及的 Hibernate 核心 API 包括 Configuration、SessionFactory、Session、Transaction。
- Session 是 Hibernate 持久化管理器的核心，提供了众多持久化方法，如 save( )、delete( )、update( )、get( )、load( ) 等。
- Hibernate 提供了对象状态管理的功能，将所操作的 Java 对象的状态分为 3 种，即 Transient（瞬时状态）、Persistent（持久状态）、Detached（游离状态）。

本章练习

1．通过与 JDBC 类比的方式简述使用 Hibernate 的几个步骤，并写出 Java 领域的相关技术。

2．说明 Hibernate 中 Java 对象的 3 种状态的特点及转换规律。

3．说明 saveOrUpdate( ) 和 merge( ) 方法的区别。

4．某电子备件管理系统的详细设计文档中有以下数据库表（见表 3-7）。

表3-7　数据库表

字　段　名	数 据 类 型	Java类型	说　　明
编号	NUMBER(4)	java.lang.Integer	主键
型号	NVARCHAR2(20)	java.lang.String	不允许为空
出厂价格	NUMBER (7, 2)	java.lang.Double	
出厂日期	DATE	java.util.Date	

（1）编写 SQL 语句创建数据表，并编写对应的持久化类和映射文件。

（2）为数据库添加以下备件信息，如表 3-8 所示。

表3-8　添加备件信息

型　　号	出厂价格（元）	出 厂 日 期
CDMA-1	650	2008-10-25

（3）更新表 3-8 的这条记录，按以下数据（见表 3-9）完成更新。

表3-9　更新数据

型　　号	出厂价格（元）	出 厂 日 期
CDMA-1	800	2009-9-9

（4）删除这条记录。