第 4 章　使用注解和 JPA

本书第 2 章的 2.3.3 节（运用 ORM 的 JPA 模式）已经介绍了 JPA（Java Persistence API）的概念，它属于 JavaEE API 的内容，为 Java 对象持久化提供了统一标准的接口。如图 4-1 所示，Hibernate 实现了 JPA，这使得应用程序既可以通过 Hibernate API 访问数据库，也可以通过 JPA API 访问数据库。

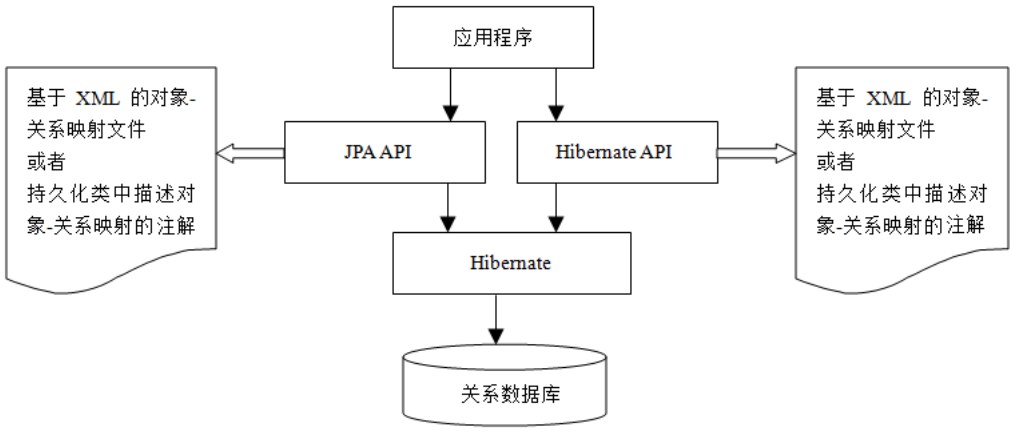

图 4-1　应用程序通过 JPA API 或 Hibernate API 访问数据库

Hibernate API 的 JavaDoc 文档位于 Hibernate 下载软件包的展开目录下：

```
hibernate-release-X.Final/documentation/javadocs/index.html
```

此外，在以下网址也可以查阅 Hibernate API 的 JavaDoc 文档：

```
http://docs.jboss.org/hibernate/orm/5.3/javadocs/
```

JPA API 的 JavaDoc 文档的官方网址为：

```
https://docs.oracle.com/javaee/7/api/
```

根据图 4-1，应用程序访问数据库有好几种方式，参见表 4-1。

表 4-1　应用程序访问数据库的方式

方式	如何描述对象-关系映射信息	访问数据库的 API	说明
方式一	在持久化类中采用注解	Hibernate API	参见本章 4.2 节。4.2.3 节的 BusinessService1 类就采用这种方式访问数据库。
方式二	在持久化类中采用注解	JPA API	参见本章 4.3 节。4.3.2 节的 BusinessService2 类就采用这种方式访问数据库。
方式三	采用基于 XML 格式的对象-关系映射文件	JPA API	参见本章 4.4 节。本章 4.4 节的例程 4-5 的 orm.xml 是 JPA 的对象-关系映射文件。
方式四	采用基于 XML 格式的对象-关系映射文件	Hibernate API	对于早期的 Hibernate 版本，这是使用最广泛的方式，现在已经逐渐被其他方式取代。本书第 3 章的范例就采用这种方式。

方式一和方式二是目前很流行的使用方式，它们的共同特点是采用基于注解的对象-关系映射元数据。本书大部分范例使用的是方式二。这些注解（也称为标注）有两个来源：

- 大部分来自 JPA API，具有很好的通用性和可移植性。
- 小部分来自 Hibernate API。Hibernate API 的注解仅起到补充作用，弥补 JPA API 中尚未提供的功能。

本章将对第 3 章的范例进行改写，主要介绍如何通过方式一、方式二和方式三来访问数据库。本章源代码位于本书配套光盘的 sourcecode/chapter4 目录下，version1、version2 和 version3 子目录分别是这三种方式的源代码。在每个子目录下有一个 ANT 工具的工程管理文件 build.xmll，它定义了名为“run”的 Target，分别用于运行 BusinessService1、BusinessService2 或 BusinessService3 类。

在 DOS 控制台转到 version1、version2 或 version3 子目录下，运行命令“ant run”，就会分别运行 BusinessService1、BusinessService2 或 BusinessService3 类。

4.1 创建包含注解的持久化类

第 3 章的 3.2 节（创建持久化类）已经创建了一个 Customer 持久化类。以下例程 4-1 的 Customer 类中加入了用于描述对象-关系映射信息的注解。

例程 4-1 Customer.java

```
package mypack;
import java.io.Serializable;
import java.sql.Date;
import java.sql.Timestamp;
import javax.persistence.*;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Type;

@Entity
@Table(name = "CUSTOMERS")
public class Customer implements Serializable {
  @Id
  @GeneratedValue(generator="increment")
  @GenericGenerator(name="increment", strategy = "increment")
  @Column(name="ID")
  private Long id;

  @Basic(optional=false)
  @Column(name="NAME",length=15)
  @Type(type="string")
  private String name;

  @Basic(optional=false)
  @Column(name="EMAIL",length=128)
  @Type(type="string")
  private String email;

  @Basic(optional=false)
  @Column(name="PASSWORD",length=8)
  @Type(type="string")
  private String password;
```

```
        @Basic(optional=true)
        @Column(name="PHONE")
        @Type(type="int")
        private int phone;

        @Basic(optional=true)
        @Column(name="ADDRESS",length=255)
        @Type(type="string")
        private String address;

        @Basic(optional=true)
        @Column(name="SEX")
        @Type(type="character")
        private char sex;

        @Basic(optional=true)
        @Column(name="IS_MARRIED")
        @Type(type="boolean")
        private boolean married;

        @Basic(optional=true)
        @Column(name="DESCRIPTION")
        @Type(type="text")
        private String description;

        @Basic(optional=true)
        @Column(name="IMAGE",columnDefinition="BLOB")
        @Type(type="materialized_blob")
        private byte[] image;

        @Basic(optional=true)
        @Column(name="BIRTHDAY")
        @Type(type="date")
        private Date birthday;

        @Basic(optional=true)
        @Column(name="REGISTERED_TIME",columnDefinition="TIMESTAMP")
        @Type(type="timestamp")
        private Timestamp registeredTime;

        public Customer(){}

        public Long getId(){
          return id;
        }

        private void setId(Long id){
          this.id = id;
        }

        public String getName(){
          return name;
        }
```

```
    public void setName(String name){
      this.name=name;
    }

    //此处省略 email、password 和 phone 等属性的 getXXX()和 setXXX()方法
    ……
}
```

在以上 Customer 类中，大部分注解，例如@Entity、@Table、@Column、@Id，来自于 JPA API 中的 javax.persistence 包。个别注解，例如@GenericGenerator 和@Type，来自于 Hibernate API 中的 org.hibernate.annotations 包。

下面分别介绍 Customer 类中的各个注解的用法。

- @Entity 注解：用于声明 Customer 类是一个实体类（即持久化类），实体类的实例可以被持久化到关系数据库中。
- @Table 注解：通过 name 属性指定与 Customer 类对应的表是 CUSTOMERS。name 属性的默认值为持久化类的名字，本范例中默认值为“Customer”，这意味着默认情况下，与 Customer 类对应的表是 Customer 表。
- @Id、@ GeneratedValue 和@GenericGenerator 注解：@Id 和@ GeneratedValue 来自于 JPA API，@GenericGenerator 来自于 Hibernate API。@Id 注解声明 Customer 类的 id 属性是对象标识符。@GeneratedValue 和@GenericGenerator 注解联合使用，用来指定对象标识符的生成策略。本书第 6 章（映射对象标识符）进一步介绍了对象标识符的各种生成策略。
- @Basic 注解：用来声明 Customer 类的 name 和 description 等属性都是需要被持久化的基本属性。@Basic 注解的 optional 属性指定持久化类的特性属性是否允许为 null。optional 属性的默认值是 true。例如，以下代码表明 Customer 类的 name 属性不允许为 null，而 description 属性可以为 null：

```
@Basic(optional=false)
@Column(name="NAME" length=15)
@Type(type="string")
private String name;

@Basic(optional=true)
@Column(name="DESCRIPTION")
@Type(type="text")
private String description;
```

当@Basic 注解的各个属性都取默认值时，@Basic 注解可以省略。例如修饰 Customer 类的 description 属性的@Basic 注解其实可以省略：

```
//@Basic(optional=true)
@Column(name="DESCRIPTION")
@Type(type="text")
private String description;
```

- @Column 注解：通过 name 属性来指定 Customer 类的 name 和 address 等属性分别和 CUSTOMERS 表中的 NAME 字段和 ADDRESS 字段对应。@Column 注解的 columnDefinition 属性以及 length 属性还可以分别指定数据库中相应字段的 SQL 类型和长度，例如：

```
@Basic(optional=true)
```

```
@Column(name="ADDRESS",length=255)
@Type(type="string")
private String address;

@Basic(optional=true)
@Column(name="REGISTERED_TIME",columnDefinition="TIMESTAMP")
@Type(type="timestamp")
private Timestamp registeredTime;
```

@Column 注解的 columnDefinition 属性以及 length 属性主要和 Hibernate 的 hbm2ddl 工具联合使用。当使用 hbm2ddl 工具来自动生成数据库表时，hbm2ddl 工具将依据它们精确地定义表的字段。关于 hbm2ddl 工具的用法参见本章 4.2.2 节。如果没有设定@Column 注解的 columnDefinition 属性以及 length 属性，hbm2ddl 工具会采用这些属性的默认值。

@Column 注解还有一个 nullable 属性，指定当前属性是否允许为空，默认值为 true。因此，@Basic(optional=false)和@Column(nullbable=false)的作用是等价的，都表示当前属性不允许为空。

- @Type 注解：通过 type 属性设定 Customer 类的各个属性的 Hibernate 映射类型，例如以下代码表明 Customer 类的 sex 属性的 Hibernate 映射类型为“character”：

```
@Basic(optional=true)
@Column(name="SEX")
@Type(type="character")
private char sex;
```

第 3 章的 3.4 节（ 创建对象-关系映射文件）介绍了 Hibernate 的 XML 格式的映射文件 Customer.hbm.xm。表 4-2 对 XML 格式的映射方式以及基于注解的映射方式做了比较，两者各有优缺点。开发人员应该根据应用需求来权衡到底采用哪种方式。

表 4-2 比较 XML 格式的映射方式以及基于注解的映射方式

比较内容	XML 格式的映射方式	基于注解的映射方式
持久化类的独立性	优点：在持久化类中不涉及到关系数据库的信息，持久化类具有更好的独立性。	缺点：在持久化类中涉及到关系数据库的信息，持久化类依赖于关系数据库的结构。
可维护性	优点：映射信息集中放在一些映射文件中，便于管理和维护。	缺点：注解分散在各个持久化类中，不利于统一管理和维护。
代码的复杂性	缺点：XML 配置代码比较冗长。	优点：注解比较简洁。
错误检测时机	缺点：在编辑和编译阶段无法发现语法错误，只有到程序运行时才能发现映射文件中的语法错误。	优点：在编译阶段就能发现注解的语法错误。

4.2 方式一：注解和 Hibernate API

本节介绍如何联合使用基于注解的对象-关系映射元数据和 Hibernate API，来完成数据访问操作。

4.2.1 创建 Hibernate 配置文件

为了确保 Hibernate 能够顺利读取持久化类中描述对象-关系映射的注解，需要在 Hibernate 的配置文件中通过<mapping>元素设定范例包含的实体类，本范例包含 Customer 实体类。以下例程 4-2 的 hibernate.cfg.xml 是本范例的 Hibernate 配置文件。

例程 4-2　hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory >
    <property name="dialect">
      org.hibernate.dialect.MySQL57Dialect
    </property>
    <property name="connection.driver_class">
      com.mysql.jdbc.Driver
    </property>
    <property name="connection.url">
      jdbc:mysql://localhost:3306/sampledb?useSSL=false
    </property>
    <property name="connection.username">
      root
    </property>
    <property name="connection.password">
      1234
    </property>

    <property name="show_sql">true</property>

    <!-- Drop and re-create the database schema on startup -->
    <property name="hbm2ddl.auto">create</property>

    <!-- Names the annotated entity class -->
    <mapping class="mypack.Customer"/>

  </session-factory>
</hibernate-configuration>
```

以上配置文件通过<mapping>元素设定本范例包含 Customer 实体类，Hibernate 在初始化时，会根据这个配置信息去读取 Customer 类的描述对象-关系映射信息的注解，把这些元数据加载到内存中。

4.2.2　自动创建数据库表

Hibernate 自带了一个 hbm2ddl 工具，它能根据对象-关系映射元数据在数据库中自动创建相应的表。可以通过 Hibernate 配置文件中的“hbm2ddl.auto”属性来使用 hbm2ddl 工具。“hbm2ddl.auto”属性有以下可选值：

- none：什么也不做。这是该属性的默认值。在这种情况下，需要手工创建数据库中的表。
- create：每次初始化 Hibernate 时都会删除已经存在的表，然后根据对象-关系映射元数据来重新生成新表。
- create-drop ：每次初始化 Hibernate 时都会根据对象-关系映射元数据来重新生成新表，但是当关闭 SessionFactory 时，会自动删除所创建的表。
- update：很常用的属性。初始化 Hibernate 时，根据对象-关系映射元数据的变化来更新数据库中表的结构，使表的结构和对象-关系映射元数据保持一致。如果表尚不存在，就创建相应的表。
- validate ：每次初始化 hibernate 时，验证数据库中表的结构，判断是否与对象-关系映射元数据保持一致。

本章 4.2.1 节的例程 4-2 的 hibernate.cfg.xml 配置文件把 hbm2ddl.auto 属性设为 create，因此每次运行范例程序时，hbm2ddl 工具都会删除已经创建的 CUSTOMERS 表，然后再根据 Customer 类的对象-关系映射元数据重新创建新的 CUSTOMERS 表。这会导致以前在 CUSTOMERS 表中插入的数据全部丢失。把 hbm2ddl.auto 属性设为 create，在程序调试阶段会很方便。但是当程序作为产品正式运行时，把 hbm2ddl.auto 属性设为 update 更符合实际需求。

另外要注意的是，当通过 hbm2ddl 工具来自动生成数据库表时，需要在 Hibernate 的配置文件中准确地设置“dialect”属性，它表示与数据库软件版本所匹配的 SQL 方言。否则，可能会导致 Hibernate 生成的 DDL（Data Definition Language）语句与数据库软件版本不匹配，因此数据库无法执行这些 DDL 语句。

例如对于 MySQL，当使用 MySQL5.5 或者以下版本的 SQL 方言时， hbm2ddl 工具自动创建 CUSTOMERS 表所使用的 DDL 语句为：

```
create table CUSTOMERS (ID bigint not null, ADDRESS varchar(255),
BIRTHDAY date, DESCRIPTION longtext, EMAIL varchar(128) not null,
……primary key (ID)) type=MyISAM
```

以上 DDL 语句在 MySQL5.7 中是不合法的，因为 MySQL5.7 不支持“type=MyISAM”语句。本书配套光盘提供的 MySQL 安装软件为 MySQL5.7，所以在 Hibernate 配置文件中需使用与该版本匹配的 SQL 方言：

```
<property name="dialect">
  org.hibernate.dialect.MySQL57Dialect
</property>
```

针对以上 SQL 方言，hbm2ddl 工具自动创建 CUSTOMERS 表时所使用的 DDL 语句为：

```
create table CUSTOMERS (ID bigint not null, ADDRESS varchar(255),
BIRTHDAY date, DESCRIPTION longtext, EMAIL varchar(128) not null,
……primary key (ID)) engine=InnoDB
```

以上 DDL 语句在 MySQL5.7 中是合法的，可以成功地创建 CUSTOMERS 表。

如果不清楚与数据库版本所匹配的具体 Dialect 类，可以查阅 Hibernate 的 JavaDoc 文档。

4.2.3 使用 Hibernate API

对象-关系映射信息无论是存放在采用 XML 格式的对象-关系映射文件中，还是用持久化类中的注解来表示，这对通过 Hibernate API 访问数据库的代码不会带来什么变动。

第 3 章的 3.5 节的范例 3-5 的 BusinessService 类会读取 Customer.hbm.xml 文件中的映射信息，而本书配套光盘中的 sourcecode/chapter4/version1 目录下的 BusinessService1 类则会读取 Customer 持久化类中的基于注解的映射信息。

BusinessService1 类的源代码和第 3 章的 BusinessService 类非常相似。区别仅仅在于 BusinessService1 类去除了把 Customer 对象包含的信息打印到网页上的代码。本节对 BusinessService1 类的源代码不再做赘述。

4.3 方式二：注解和 JPA API

根据本章开头的介绍，当在 Customer 类中存放了基于注解的对象-关系映射元数据时，Java 程序既可以通过 Hibernate API，也可以通过 JPA API 来对 Customer 对象进行持久化操作。本节介绍通过 JPA API 来访问数据库的方式。

4.3.1 创建 JPA 的配置文件

JPA 的配置文件名为 persistence.xml，位于 classpath 根目录的 META-INF 目录下，它的作用和 Hibernate 的配置文件 hibernate.cfg.xml 很相似。以下例程 4-3 的 persistence.xml 是本范例的 JPA 配置文件。

例程 4-3　persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
      http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
      version="2.0">
  <persistence-unit name="myunit">

    <provider>
      org.hibernate.jpa.HibernatePersistenceProvider
    </provider>

    <class>mypack.Customer</class>

    <properties>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.MySQL57Dialect" />

      <property name="javax.persistence.jdbc.driver"
        value="com.mysql.jdbc.Driver" />

      <property name="javax.persistence.jdbc.url"
        value="jdbc:mysql://localhost:3306/sampledb?useSSL=false" />

      <property name="javax.persistence.jdbc.user" value="root" />
      <property name="javax.persistence.jdbc.password" value="1234" />

      <property name="hibernate.show_sql" value="true" />

      <!-- Drop and re-create the database schema on startup -->
      <property name="hibernate.hbm2ddl.auto" value="create" />

    </properties>
  </persistence-unit>
</persistence>
```

以上配置文件的<persistence-unit>元素设定了一个持久化单元包，它的 name 属性设定持久化单元包的名字。

<provider>元素指定 JPA API 由哪个 ORM 软件来实现，本范例由 HibernatePersistenceProvider 来实现。

<class>元素指定包含描述对象-关系映射信息的注解的持久化类。JPA 在初始化时会从这些持久化类中读取对象-关系映射元数据。

在以上 persistence.xml 配置文件中，有些<property>元素的 name 属性以“javax.persistence”开头，表明这是属于 JPA 的属性；有些<property>元素的 name 属性以“hibernate”开头，表明这是属于 Hibernate 的属性。

4.3.2 使用 JPA API

以下图 4-2 显示了 Hibernate API 和 JPA API 中主要接口之间的对应关系。从图中可以看出，SessionFactory 接口在 JPA API 中的对等接口是 javax.persistence.EntityManagerFactory；Session 接口在 JPA API 中的对等接口是 javax.persistence.EntityManager；Transaction 接口在 JPA API 中的对等接口是 javax.persistence.EntityTransaction。Query 接口在 JPA API 中的对等接口是 javax.persistence.Query。

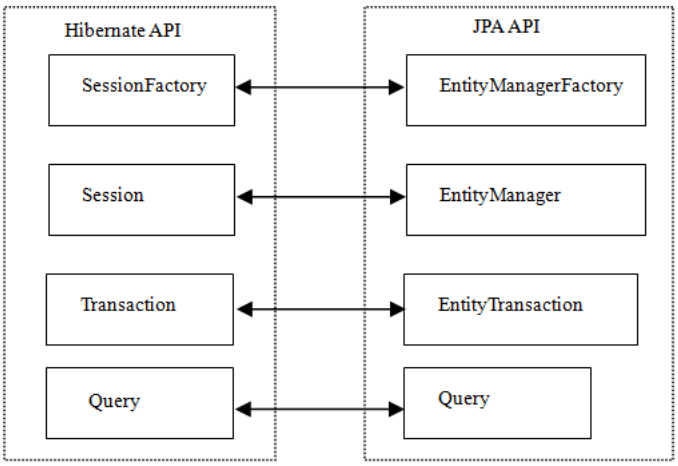

图 4-2 Hibernate API 和 JPA API 的对应关系

EntityManager 接口提供了操纵数据库的各种方法，如：

- persist()方法：把 Java 对象保存到数据库中。等价于 Session 接口的 persist()方法。
- merge()方法：保存或更新数据库中的 Java 对象。等价于 Session 接口的 merge()方法。
- remove()方法：把特定的 Java 对象从数据库中删除。类似于 Session 接口的 delete()方法。EntityManager 接口的 remove()方法只能删除持久化状态的对象，而 Session 接口的 delete()方法可以删除持久化状态或游离状态的对象。关于持久化状态和游离状态的概念，参见本书第 8 章的 8.3 节（Java 对象在持久化层的状态）。
- find()方法：从数据库中加载 Java 对象。等价于 Session 接口的 get()方法。

以下例程 4-4 的 BusinessService2 类通过 JPA API 来访问数据库。

例程 4-4 BusinessService2.java

```
package mypack;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.EntityTransaction;
import javax.persistence.Query;
import java.io.*;
import java.sql.Date;
import java.sql.Timestamp;
import java.util.*;

public class BusinessService2{
```

```
public static EntityManagerFactory entityManagerFactory;

/** 初始化 JPA，创建 EntityManagerFactory 实例 */
static{
  try{
    entityManagerFactory=
        Persistence.createEntityManagerFactory( "myunit" );
  }catch(Exception e){
    e.printStackTrace();
    throw e;
  }
}

/** 查询所有的 Customer 对象，
   然后调用 printCustomer()方法打印 Customer 对象信息 */
public void findAllCustomers(PrintWriter out)throws Exception{
  EntityManager entityManager =
        entityManagerFactory.createEntityManager();
  EntityTransaction tx = null;
  try {
    tx = entityManager.getTransaction();
    tx.begin(); //开始一个事务
    Query query=entityManager.createQuery(
        "from Customer as c order by c.name asc");
    List customers=query.getResultList();
    for (Iterator it = customers.iterator(); it.hasNext();) {
      printCustomer(out,(Customer) it.next());
    }

    tx.commit(); //提交事务

  }catch (RuntimeException e) {
    if (tx != null) {
      tx.rollback();
    }
    throw e;
  } finally {
     entityManager.close();
  }
}

/** 持久化一个 Customer 对象 */
public void saveCustomer(Customer customer){
  EntityManager entityManager =
         entityManagerFactory.createEntityManager();

  EntityTransaction tx = null;
  try {
    tx = entityManager.getTransaction();
    tx.begin();
    entityManager.persist(customer);
    tx.commit();
```

```
  }catch (RuntimeException e) {
    if (tx != null) {
      tx.rollback();
    }
    throw e;
  } finally {
    entityManager.close();
  }
}

/** 按照 OID 加载一个 Customer 对象，然后修改它的属性 */
public void loadAndUpdateCustomer(Long customer_id,String address){
  EntityManager entityManager =
              entityManagerFactory.createEntityManager();
  EntityTransaction tx = null;
  try {
    tx = entityManager.getTransaction();
    tx.begin();
    Customer c=entityManager
                .find(Customer.class,customer_id);
    c.setAddress(address);
    tx.commit();

  }catch (RuntimeException e) {
    if (tx != null) {
      tx.rollback();
    }
    throw e;
  } finally {
    entityManager.close();
  }
}

/**删除 Customer 对象 */
public void deleteCustomer(Customer customer){
  EntityManager entityManager =
         entityManagerFactory.createEntityManager();
  EntityTransaction tx = null;
  try {
    tx = entityManager.getTransaction();
    tx.begin();
    //获得持久化状态的 Customer 对象
    Customer c=entityManager
        .find(Customer.class,customer.getId());
    entityManager.remove(c);
    tx.commit();

  }catch (RuntimeException e) {
    if (tx != null) {
      tx.rollback();
    }
    throw e;
```

```
    } finally {
      entityManager.close();
    }
  }

  /** 把 Customer 对象的信息输出到控制台，如 DOS 控制台*/
  private void printCustomer(PrintWriter out,Customer customer)
                                  throws Exception{……}

  public void test(PrintWriter out) throws Exception{……}

  public static void main(String args[]) throws Exception{
    new BusinessService2().test(new PrintWriter(System.out,true));
    entityManagerFactory.close();
  }
}
```

对 JPA 的初始化非常简单，只要通过 javax.persistence.Persistence 的静态方法 createEntityManagerFactory()来创建 EntityManagerFactory 对象：

```
  entityManagerFactory=
    Persistence.createEntityManagerFactory( "myunit" );
```

以上 Persistence.createEntityManagerFactory("myunit")方法中的参数“myunit”指定持久化单元包的名字。JPA 会到 persistence.xml 配置文件中读取相应的持久化单元包中的配置信息。

所有访问数据库的操作都使用以下流程：

```
EntityManager entityManager =
    entityManagerFactory.createEntityManager();
EntityTransaction tx = null;
try {
  tx = entityManager.getTransaction();
  tx.begin();  //声明开始事务
  //执行查询、保存、更新和删除等各种数据访问操作
  ……
  tx.commit();  //提交事务
}catch (RuntimeException e) {
  if (tx != null)
    tx.rollback();
  throw e;
} finally {
  entityManager.close();
}
```

4.3.3 从 JPA API 中获得 Hibernate API

当 JPA API 不能满足所有的应用需求，还可以利用 Hibernate API 来辅助完成个别功能。JPA API 的 EntityManager 接口和 EntityManagerFactory 接口都有一个 unwrap()方法，它们分别返回相应的 Session 以及 SessionFactory 对象：

```
//获得 Hibernate API 中的 SessionFactory
SessionFactory sessionFactory =
        entityManagerFactor.unwrap( SessionFactory.class );
```

```
//获得 Hibernate API 中的 Session
Session session = entityManager.unwrap( Session.class );
```

得到了 SessionFactory 对象和 Session 对象后，就可以通过它们来操纵数据库了。

4.4 方式三：对象-关系映射文件和 JPA API

在本章开头的表 4-1 的方式三中，JPA 从对象-关系映射文件中获取映射信息。当 JPA 通过底层 Hibernate 来访问数据库时，JPA 支持以下两种类型的对象-关系映射文件：

（1）JPA 的对象关系-映射文件，采用 JPA 所规定的语法。默认的文件为 META-INF/orm.xml。

（2）Hibernate 的对象-关系映射文件，采用 Hibernate 所规定的语法。第 3 章的 Customer.hbm.xml 文件就是这样的映射文件。第 12 章的 12.5.2 节（用对象-关系映射文件来映射）介绍了通过 JPA API 来访问这种类型映射文件的范例。

JPA 的对象-关系映射文件与 Hibernate 的对象-关系映射文件虽然作用相同，但语法不相同。本节将简单介绍 JPA 的对象-关系映射文件。以下例程 4-5 为本范例的 orm.xml，它设置了 Customer 类和 CUSTOMERS 表的映射关系。

例程 4-5　orm.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<entity-mappings
  version="2.1"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence/orm
  http://xmlns.jcp.org/xml/ns/persistence/orm_2_1.xsd">

  <persistence-unit-metadata>
    <xml-mapping-metadata-complete/>
    <persistence-unit-defaults>
      <delimited-identifiers/>
    </persistence-unit-defaults>
  </persistence-unit-metadata>

  <entity class="mypack.Customer" access="FIELD">
    <table name="CUSTOMERS" />
    <attributes>
      <id name="id">
        <generated-value strategy="AUTO"/>
        <column name="ID"/>
      </id>
      <basic name="name" optional="false" >
        <column name="NAME" length="15" />
      </basic>
      <basic name="email" optional="false" >
        <column name="EMAIL" length="128" />
      </basic>
      <basic name="password" optional="false" >
        <column name="PASSWORD" length="8" />
      </basic>
```

```
            <basic name="phone" >
              <column name="PHONE"/>
            </basic>
            <basic name="address" >
              <column name="ADDRESS" length="255" />
            </basic>
            <basic name="sex" >
              <column name="SEX"/>
            </basic>
            <basic name="married" >
              <column name="IS_MARRIED"/>
            </basic>
            <basic name="description" columnDefinition="TEXT" >
              <column name="DESCRIPTION"/>
            </basic>
            <basic name="image" >
              <lob/>
              <column name="IMAGE" columnDefinition="BLOB"/>
            </basic>
            <basic name="birthday" >
              <column name="BIRTHDAY"/>
            </basic>
            <basic name="registeredTime">
              <column name="REGISTERED_TIME" columnDefinition="TIMESTAMP"/>
            </basic>
          </attributes>
        </entity>
      </entity-mappings>
```

关于 JPA 的对象-关系映射文件，有以下使用规则：

（1）如果对象-关系映射文件不采用默认的文件名 orm.xml，而是采用自定义的文件名字，则需要在 JPA 的配置文件 META-INF/persistence.xml 中通过<mapping-file>元素来显式设定它们，例如：

```
      <persistence-unit name="myunit">
        ……
        <mapping-file>mypack/Customer.xml</mapping-file>
        <mapping-file>mypack/Order.xml</mapping-file>
        ……
      </persistence-unit>
```

（2）JPA 可以同时从对象-关系映射文件或者持久化类的 JPA 注解中获取映射信息，当两者提供的映射信息不一致时，JPA 注解会覆盖对象-关系映射文件中的映射信息。如果仅仅希望从对象-关系映射文件中获取映射信息，那么可以在对象-关系映射文件中加入<<xml-mapping-metadata-complete>元素：

```
      <persistence-unit-metadata>
        <xml-mapping-metadata-complete/>
        ……
      <persistence-unit-metadata>
```

（3）对象-关系映射信息无论是存放在采用 XML 格式的对象-关系映射文件中，还是用持久化类中的注解来表示，这对通过 JPA API 访问数据库的代码不会带来什么变动。

（4）尽管在例程 4-5 的 orm.xml 文件中把 CUSTOMERS 表的 REGSITERED_TIME 字

段的 SQL 类型设为 TIMESTAMP 类型，但是 Hibernate5.7 的 hbm2ddl 工具在自动创建 CUSTOMERS 表时，会忽略这一设置，把 REGSITERED_TIME 字段定义为 datetime 类型。也许将来的 Hibernate 版本会完善这一个功能。目前解决这个问题的办法是把 REGSITERED_TIME 字段的 SQL 类型手工修改为 TIMESTAMP 类型，或者在 JPA 的 persistence.xml 文件中设置 hibernate.hbm2ddl.import_files 属性：

```
<property name="hibernate.hbm2ddl.auto" value="create" />
<property  name="hibernate.hbm2ddl.import_files"
     value="schema-generation.sql" />
```

当 Hibernate 的 hbm2ddl 工具生成了 CUSTOMERS 表后，还会执行 schema-generation.sql 脚本。schema-generation.sql 文件位于 classpath 的根目录下，它用于修改 REGSITERED_TIME 字段的 SQL 类型，内容如下：

```
alter table CUSTOMERS alter column REGISTERED_TIME timestamp;
```

本书不详细讨论 JPA 的对象-关系映射文件的语法。总的说来，该映射文件中的元素和 JPA 注解存在着对应关系。因此，掌握了 JPA 注解的用法后，会很容易地学会对象-关系映射文件的用法。

本节范例位于配套光盘的 sourcecode/chapter4/version3 目录下。Customer 类为普通的 POJO 类，里面不包含 JPA 注解或 Hibernate 注解，它的源代码参见第 3 章的 3.2 节的例程 3-3。BusinessService3 类和本章 4.3.2 节的例程 4-4 的 BusinessService2 类的代码相同。

4.5 小结

本章介绍了如何在持久化类中通过注解来描述对象-持久化映射信息，这些注解大部分来自 JPA API，小部分来自 Hibernate API。本章主要介绍了以下注解的用法：

- @Entity 注解：声明实体类（即持久化类），实体类的实例可以被持久化到关系数据库中。
- @Table 注解：设定与持久化类对应的表。
- @Id、@ GeneratedValue 和@GenericGenerator 注解：设定持久化类的对象标识符。
- @Basic 注解：声明持久化类的特定属性是需要被持久化的基本属性。
- @Column 注解：设定持久化类的特定属性在数据库表中对应的字段。
- @Type 注解：设定持久化类的特定属性的 Hibernate 映射类型。

本章还介绍了如何通过 Hibernate API 和 JPA API 来操纵数据库。JPA API 和 Hibernate API 在功能上有许多相似之处。区别在于 JPA API 是由 Oracle 公司制定的标准的对象持久化 API，但它本身没有具体的实现，依赖于第三方 ORM 软件。而 Hibernate API 是 Hibernate 软件特有的接口。

此外，JPA 和 Hibernate 都支持基于 XML 格式的对象-关系映射文件，但两者的映射文件的语法不一样。

4.6 思考题

1．以下哪些注解来自 JPA API？（多选）

（a）@Entity　（b）@GenericGenerator　（c）@Id　（d）@Table

2．在 Hibernate 配置文件中，把“hbm2ddl.auto”属性设为什么值，会导致每次初始化

Hibernate 时，都会根据对象-关系映射元数据来重新生成表，但是当关闭 SessionFactory 时，会自动删除所创建的表？（单选）

（a）none　　（b）create　　（c）create-drop　　（d）update

3．以下代码为 Customer 类的 address 属性设定了一些注解。以下哪些说法正确？（多选）

```
@Basic(optional=true)
@Column(name="ADDRESS",length=255)
@Type(type="string")
private String address;
```

（a）与 Customer 类的 address 属性对应的字段是 ADDRESS。

（b）Customer 类的 address 属性的长度必须是 255。

（c）Customer 类的 address 属性的取值不允许为 null。

（d）ADDRESS 字段的长度是 255。

4．在 JPA 的配置文件中，哪个元素用来设定持久化单元包的名字？（单选）

（a）<class>　　（b）<provider>　　（c）<property>　　（d）<persistence-unit>

5．通过 javax.persistence.EntityManager 接口把 Java 对象保存到数据库中，应该调用哪个方法？（单选）

（a）merge()　　（b）persist()　　（c）save()　　（d）find()