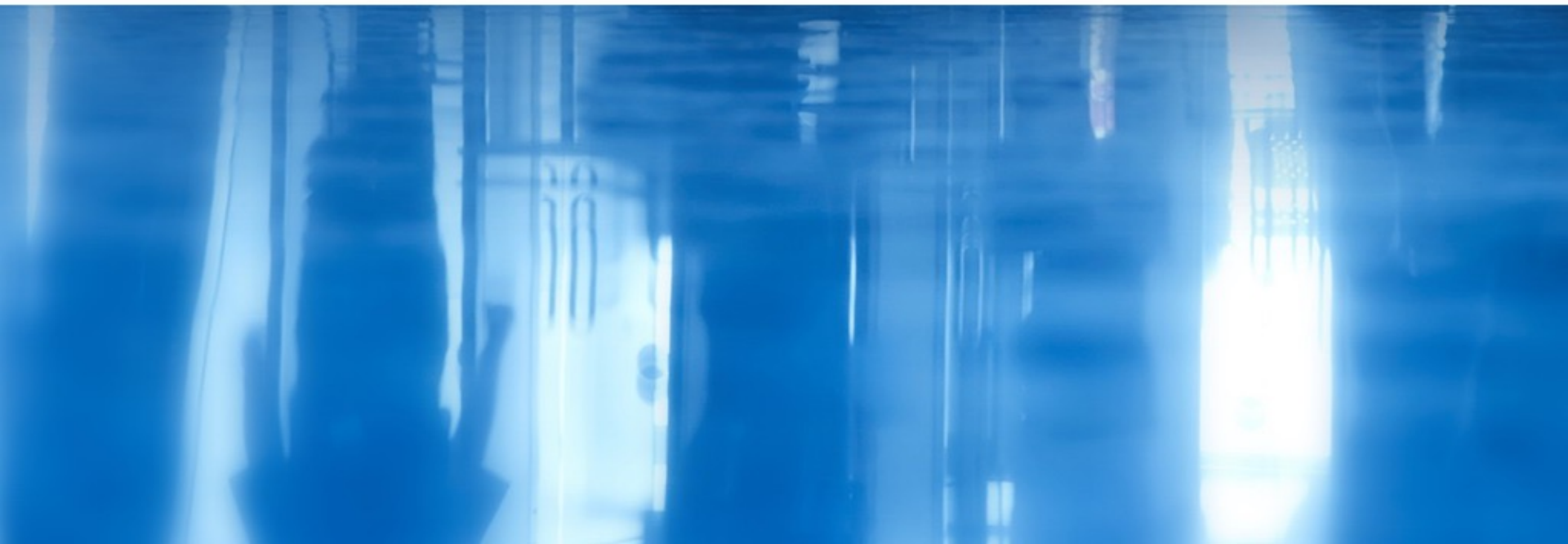
第11章

Hibernate框架基础



本章内容

- 11.1 ORM与Hibernate
- 11.2 第一个Hibernate程序
- 11.3 Hibernate框架结构
- 11.4 Hibernate核心API
- 11.5 映射文件详解
- 11.6 配置文件详解
- 11.7 关联映射

本章内容

- 11.8　组件属性映射
- 11.9　继承映射
- 11.10　Hibernate数据查询
- 11.11　其他查询技术

11.1 ORM与Hibernate

- ORM ,Object/Relation Mapping ,对象/关系映射。
- Hibernate是一个对象/关系映射的框架，它用来实现应用程序的持久化功能。

11.1.1 数据持久化与ORM

- Java程序是通过对象表示数据。而当今的数据库都是关系型数据库，通过表结构存储数据。
- 这样在程序设计语言中的对象和关系数据库的数据表之间就存在不匹配的情况。
- 如何将程序中的对象存储到数据表中，如何从数据表中取出数据构成程序中的对象，这就是对象与关系之间的映射问题，通常称为对象/关系映射。

11.1.1 数据持久化与ORM

- 使用Java的JDBC当然可以将程序中的对象数据取出，然后写入数据库，也可以将数据库中数据取出，然后构建程序中使用的对象。但这种方法要求开发人员对JDBC的底层非常熟悉，并可以依据不同的需求来完成不同的功能。

11.1.1 数据持久化与ORM

- Hibernate是一个开放源代码的对象/关系映射框架，它对JDBC进行了非常轻量级的封装，使得Java程序员可以用对象编程思维来操纵数据库。简单地说，就是将Java对象与对象关系映射到关系型数据库的数据表与数据表之间的关系，Hibernate提供了这个功能。Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用。

11.1.2 Hibernate软件包简介

- Hibernate官方网站的网址为http://www.hibernate.org/，从这个网站可以获得Hibernate所有发行包和关于Hibernate的详细信息。
- Hibernate软件包包括Hibernate Core、Hibernate Shards、Hibernate Search、Hibernate Tools、Hibernate Metamodel Generator等，其中Hibernate Core软件包包含了Hibernate的所有核心功能。

1. Hibernate常用软件包

- 最新的Hibernate 核心包文件名为hibernate-release-4.2.0.CR1.zip，将该文件解压到一个临时目录，其中包括3个目录，documentation、lib和project。
- documentation目录中包含Hibernate的开发指南、DOC文档等。

1. Hibernate常用软件包

- lib目录中包含Hibernate应用编译和运行时所依赖的类库。该目录还包含几个子目录，其中required目录中包含了开发Hibernate应用的必须的库文件。其中hibernate-core-4.2.0.CR1.jar文件是开发Hibernate应用的基础框架和核心API。其他子目录中包含了可选的库文件，如实现数据库连接池的c3p0-0.9.1.jar包就存放在optional/c3p0目录中。
- project目录中存放的是Hibernate项目的源文件。

2．在Eclipse中添加Hibernate支持

- 如果只需要Hibernate的基本支持，应将Hibernate软件包解压目录的lib/requried目录中的JAR文件复制到WEB-INF/lib目录中。
- 运行Hibernate应用程序可能还需要其他库文件，如数据库驱动程序库，应将这些库也添加到WEB-INF/lib目录中。

11.2 第一个Hibernate程序

- 本节将介绍如何使用Hibernate操作数据库，用一个对Student对象保存和读取的例子说明Hibernate的基本配置和使用。

12.2.1 准备数据库表

- 使用下面的SQL语句创建一个student表。

```
CREATE TABLE student(
    id bigserial PRIMARY KEY,
    student_no  bigint,
    student_name  character
  varying(50),
    sage   integer,
    major  character varying(15)
  DEFAULT NULL
);
```

11.2.2 定义持久化类

- 持久化类也叫实体类，是用来存储要与数据库交互的数据。
- 持久化类的实例称为持久化对象（Persistent Object，PO）。PO的作用是完成对象持久化操作。
- 通过PO可以用面向对象的方式操作数据库，实现对数据执行增、删、改的操作。下面的Student类就是一个持久化类。
- 程序11.1 Student.java

11.2.3 定义映射文件

- Hibernate的映射文件定义持久化类与数据表之间的映射关系，如数据表的主键生成策略、字段的类型、实体关联关系等。
- 在Hibernate中，映射文件是XML文件，其命名规范是*.hbm.xml。

- 例如，为持久化类Student定义的映射文件名应为Student.hbm.xml，保存在与Student.java相同的目录，内容如下
- 程序11.2 Student.hbm.xml

11.2.4 编写配置文件

- 映射文件定义了持久化类和数据表之间的对应关系以及持久化类的属性和数据表字段之间的对应关系，但还不知道连接哪个数据库，以及用户名和密码等信息。这些信息对于所有的持久化类都是通用的，称这些信息为Hibernate配置信息，使用配置文件指定。

11.2.4 编写配置文件

- Hibernate配置文件可以使用XML格式，文件名为hibernate.cfg.xml。下面的配置文件连接PostgreSQL数据库。
- 程序11.3 hibernate.cfg.xml

11.2.5 编写测试程序

- 使用Hibernate可以开发独立的Java应用程序。为了简单，本例只编写一个应用程序，在其main()方法中完成启动Hibernate，创建各种对象以及持久化操作。
- 程序11.4 Main.java

11.2.6 Hibernate的自动建表技术

- 在Hibernate中，可以根据映射文件和配置文件自动创建数据表，具体实现方式有两种。

1. 通过配置文件自动建表

- 使用Hibernate配置文件进行自动建表，只需在配置文件中配置hibernate.hbm2ddl.auto属性即可，此方法简单实用。

```
<property name="hibernate.hbm2ddl.auto">
        create</property>
```

1. 通过配置文件自动建表

- hibernate.hbm2ddl.auto属性的取值可以有以下四种：
  - **create**：每次创建SessionFactory时都重新创建数据表。如果数据表已经存在，则先将其删除。使用该选项要慎重！
  - **update**：如果表不存在，则创建数据表；如果表存在，则检查表是否与映射文件匹配，当不匹配时，更新表信息。
  - **create-drop**：先删除存在的表，然后再创建。当会话结束后将表再删除。
  - **validate**：进行有效性检查，但不会创建或更新数据表。

2. 手动导出数据表

- 手动导出数据表用到org.hibernate.tool.hbm2ddl.SchemaExport类，其create()用于导出数据表。

```
public class ExportTables{
  public static void main(String[] args) {
    Configuration cfg = new
   Configuration().configure();
    SchemaExport export = new
   SchemaExport(cfg);
    export.create(true,true);
  }}
```

11.2.7 HibernateUtil辅助类

- 辅助类用来完成Hibernate的启动和会话工厂以及会话对象的创建。
- 下面的HibernateUtil类就是一个辅助类。
- 程序11.5 HibernateUtil.java

11.2.8 测试类的开发

- 下面的程序通过HibernateUtil类获得Session对象，并通过beginTransaction()方法开始一个数据库事务。
- 下面程序实现向Student表中插入一行记录。
- 程序11.6 StudentDemo.java

本章内容

- 11.1 ORM与Hibernate
- 11.2 第一个Hibernate程序
- 11.3 Hibernate框架结构
- 11.4 Hibernate核心API
- 11.5 映射文件详解
- 11.6 配置文件详解
- 11.7 关联映射

11.3 Hibernate框架结构

- 11.3.1 Hibernate的体系结构
- 11.3.2 理解持久化对象
- 11.3.3 Hibernate的核心组件
- 11.3.4 持久化对象的状态

11.3.1 Hibernate的体系结构

Persistent Object是持久化对象，是要写入持久化存储设备的对象

Hibernate映射文件*Xxx*.hbm.xml用来把持久化类和数据表、PO的属性与表的字段一一映射起来，它是Hibernate的核心文件。

11.3.2 理解持久化对象

- 持久化类是一种轻量级的持久化对象，它通常与关系数据库中的表对应，每个持久化对象与表中的一行对应。
- Hibernate中的PO完全采用普通Java对象（Plain Old Java Object，POJO）来作为持久化对象使用，PO的属性与数据表中的字段相匹配。

11.3.2 理解持久化对象

- Hibernate对持久化类没有太多的要求，应遵循下面规则。
  - 提供一个默认的构造方法。
  - 提供一个标识属性。
  - 为持久化类的每个属性提供setter和getter方法。
  - 覆盖equals()方法和hashCode()方法。

11.3.3 Hibernate的核心组件

Hibernate还包括下面的组件：

- Configuration类：用来读取Hibernate配置文件，并生成SessionFactory对象。
- SessionFactory接口：创建Session实例的工厂。
- Session接口：用来操作PO，它通过get()方法、load()方法、save()方法、update()方法和delete()方法实现对PO的加载、保存、更新和删除等操作。它是Hibernate的核心接口。

11.3.3 Hibernate的核心组件

Hibernate还包括下面的组件：

- Query接口：用来对PO进行查询操作，它从Session的createQuery()方法生成。
- Transaction接口：用来管理Hibernate的事务，它从Session的beginTransaction()方法生成，它的主要方法有commit()和rollback()。

11.3.4 持久化对象的状态

- Hibernate的持久化对象分为三种状态：

脱管态（detached）：当Session关闭后持久对象变成脱管状态，其特征是在数据库中有与之匹配的数据，但并不处于Session的管理之下。

存的管理之内。当持久对象有任何的改变时，Hibernate在更新缓存时对其进行更新。如果实例从持久态变成了临时态，Hibernate同样会对其进行删除操作，不需要手动检查脏数据。

本章内容

- 11.1 ORM与Hibernate
- 11.2 第一个Hibernate程序
- 11.3 Hibernate框架结构
- 11.4 Hibernate核心API
- 11.5 映射文件详解
- 11.6 配置文件详解
- 11.7 关联映射

11.4 Hibernate核心API

- 11.4.1 Configuration类
- 11.4.2 SessionFactory接口
- 11.4.3 Session接口
- 11.4.4 Transaction接口
- 11.4.5 Query接口

11.4.1 Configuration类

- Configuration类负责管理Hibernate的配置信息。调用Configuration类的configure()方法将加载配置文件，代码如下：

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

- 执行该语句时，Hibernate会自动在WEB-INF/classes目录中搜寻hibernate.cfg.xml。

11.4.1 Configuration类

- Configuration类的configure()方法还支持带参数的访问形式，可以指定配置文件的位置，例如：

```
File file = new File(
"D:\\cfg\\hibernateCfg.xml");
Configuration config = new
Configuration().configure(file);
```

11.4.2 SessionFactory接口

- SessionFactory是会话工厂对象，它负责创建Session实例。SessionFactory对象需要通过Configuration创建：

11.4.2 SessionFactory接口

```
private static SessionFactory factory;
private static ServiceRegistry serviceRegistry;
static{
  try{
     Configuration configuration =
              new Configuration().configure();
     serviceRegistry = new ServiceRegistryBuilder()
              .applySettings(configuration.getProperties())
              .buildServiceRegistry();
     factory = configuration.buildSessionFactory(serviceRegistry);
   }catch(HibernateException e){
              e.printStackTrace();
 }
}
```

11.4.3 Session接口

- Session对象是应用程序与数据库之间的一个会话，它是Hibernate的核心对象，相当于JDBC中的Connection对象，它是持久层操作的基础。持久化对象的生命周期、数据库的存取和事务的管理都与Session息息相关。
- 使用SessionFactory对象的openSession()方法创建Session对象。

```
Session session = factory.openSession();
```

11.4.3 Session接口

- Session接口定义了save()、load()、update()、delete()等方法分别实现持久化对象的保存、加载、修改和删除等操作。这种持久化操作是受Session控制的，即通过Session对象来完成这些操作。

1. save()方法

- save()方法用来将临时对象持久化到数据库中，对象将从临时状态变为持久状态。格式如下：

Serializable save(Object object) throws HibernateException

- 该方法将一个PO的属性取出放入PreparedStatement语句中，然后向数据库中插入一条记录（或多条记录，如果有级联）。

1. save()方法

- 例如，下面的代码把一个新建的Student对象持久化到数据库中：

```
Student stud = new Student();
stud.setStudentNo("20120101");
…
session.save(stud);
```

1. save()方法

- 如果在调用save()方法后又修改了stud的属性，则Hibernate将发出一条INSERT语句和一条UPDATE语句来完成持久化操作，如下代码所示:

```
Student stud = new Student();
stud.setStudentNo("20120101");
stud.setStudentName("王小明");
session.save(stud);
stud.setName("张大海");
// 事务提交，关闭Session
```

2. get()方法

- get()方法用来返回一个持久化类的实例，格式如下：

```
public Object get(Class clazz,
                  Serializable id)
```

- clazz是持久化类型，id是对象的主键值。以下代码取得主键id值为22的一个Student对象。

```
Student stud =(Student)session.get(
     Student.class,new Integer(22));
```

2. get()方法

- get()的执行顺序如下：
  - 首先通过id值在Session一级缓存中查找对象，如果存在此id主键值的对象，直接将其返回。
  - 否则，在SessionFactory二级缓存中查找，找到后将其返回。
  - 如果在一级缓存和二级缓存中都不能找到指定的对象，则从数据库加载拥有此id的对象。
- 因此，get()并不总是向数据库发送SQL语句，只有缓存中无此对象时，才向数据库发送SQL语句以取得数据。

3. load()方法

- load()方法也是通过标识符得到指定类的持久化对象实例，其一般格式为：

Object load(Class clazz, Serializable id)
throws HibernateException

- 返回给定的实体类和标识符的持久化实例。该方法与get()方法具有相同的格式，但二者有区别。

3. load()方法

- 当记录不存在时，get()方法返回null，load()方法抛出HibernateException异常。
- load()方法可以返回实体的代理实例。而get()方法永远都直接返回实体类。
- load()方法可以充分利用Hibernate的内部缓存和二级缓存中现有数据，而get()方法仅在Hibernate内部缓存中进行数据查找，如果在内部缓存中没有找到对应的数据，那么将直接执行SQL语句进行数据查询，并生成相应的实体对象。

4. update()方法

- update()方法用来更新托管对象，格式如下：

```
void update(Object object) throws HibernateException
```

- 这里，object为脱管实例，调用该方法将其更新为持久实例。

4. update()方法

```
stud =
   (Student)seesion.get(Student.class,ne
   w Integer(20120101));
stud.setStudentName("李明月");
session.update(stud);
```

5. saveOrUpdate()方法

- 对临时对象使用update()方法是不对的，对脱管对象使用save()方法也是不对的。这时可以使用saveOrUpdate()方法，格式如下：

```
void saveOrUpdate(Object object)
throws HibernateException
```

- saveOrUpdate()方法兼具save()和update()方法的功能，对于传入的对象，saveOrUpdate()方法首先判断该对象是临时对象还是托管对象，然后调用合适的方法。

6. delete()方法

- delete()方法用于从数据库中删除一个持久实例，格式如下：

```
void delete(Object object) throws
                HibernateException
```

- 参数对象可以是与事务相关的持久实例，也可以是临时实例。如果关联设置了cascade="delete"，该方法将级联删除相关的对象。

7． 其他方法

- Session接口还定义了
  - 创建Query对象方法
  - 生成Transaction对象方法
  - 管理Session的方法等。

11.4.4 Transaction接口

- Transaction对象表示数据库事务，它的运行与Session接口有关，可调用Session的beginTransaction()方法生成一个Transaction实例，如下代码所示。

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

11.4.4 Transaction接口

- Transaction接口的常用方法如下：
- public void begin()：开始事务。
- public void commit()：提交事务。
- public void rollback()：回滚事务。
- public boolean wasCommited()：返回事务是否已提交。
- public boolean wasRolledBack()：返回事务是否已回滚。

11.4.5 Query接口

- Query接口主要用来创建HQL查询对象。HQL是Hibernate提供的一种功能强大的查询语言。通过Session的createQuery()方法获得Query实例，格式如下：

Query createQuery(String queryString)

- 参数queryString是一个HQL字符串，可以是SELECT查询语句，也可以DELETE等更新语句。

11.4.5 Query接口

- 该方法返回一个Query对象，使用该查询对象可以查询数据库。

```
Query query =
session.createQuery("from Student");
// 生成一个Query实例
```

- 创建了Query对象后，就可以调用Query接口的list()、iterate()或executeUpdate()方法执行查询或更新操作。

1. list()方法和iterate()方法

- list()方法返回一个List对象，如果结果集是多个，则返回一个Object[]对象数组。
- iterate()方法返回一个Iterator对象，如果结果集是多个，则返回一个Object[]对象数组。

1. list()方法和iterate()方法

```
Query query =
    session.createQuery(
     "from Student s where s.sage > ?");
query.setInteger(0,20);      //  设 置 参 数 值
  List<Student> list = query.list();
for(int i = 0; i < list.size(); i++){
    Student stud = (Student)list.get(i);
    System.out.println(
        stud.getStudentName());
}
```

2. executeUpdate()方法

- Query的executeUpdate()方法用于执行HQL的更新和删除语句，它常用于批量更新和批量删除，格式如下。

```
int executeUpdate()
        throws HibernateException
```

- 返回值为更新或删除的行数。

```
Query query = session.
  createQuery("delete from Student");
query.executeUpdate();
```

3. setFirstResult()和setMaxResults()方法

- Query接口还提供了setFirstResult()和setMaxResults()两个方法，它们分别用来设置返回结果的第一行和最大行数。

Query setFirstResult(int firstResult)：设置要返回的第一行。如果没有设置，将从结果集的第0行开始。

Query setMaxResults(int maxResults)：设置返回的最大行数。如果没有设置，返回的结果数没有限制。

4. uniqueResult()方法

- uniqueResult()方法返回该查询对象的一个实例，如果查询无结果返回null，格式如下。

Object uniqueResult() throws HibernateException

本章内容

- 11.1 ORM与Hibernate
- 11.2 第一个Hibernate程序
- 11.3 Hibernate框架结构
- 11.4 Hibernate核心API
- 11.5 映射文件详解
- 11.6 配置文件详解
- 11.7 关联映射

11.5 映射文件详解

- Hibernate的映射文件把一个PO与一个数据表映射起来。
- 每个持久化类都应该有一个映射文件。下面是Student.hbm.xml映射文件的部分内容。
- 代码12.5.htm

11.5 映射文件详解

（1）<hibernate-mapping>元素

- 该元素是映射文件的根元素，其他元素嵌入在<hibernate-mapping>元素内，其常用属性主要有package属性，用于指定包名。

11.5 映射文件详解

（2）<class>元素

- <class>元素用于指定持久化类和数据表的映射。name属性指定持久化类名，table属性指定表名。如果缺省该属性，使用类名作为表名。

11.5 映射文件详解

（3）<id>元素

- <id>元素声明了一个标识符属性，例如在上述映射文件中的<id>元素如下：

```
<id name="id" column="id"><generator class="identity" /></id>
```

- name="id"表示使用Student类的id属性作为对象标识符，它与student表的id字段对应。同时告诉Hibernate使用Student类的getId()和setId()访问这个属性。

11.5 映射文件详解

（4）<generator>元素

- <generator>元素是<id>元素的一个子元素，它用来指定标识符的生成策略（即如何产生标识符值）。它有一个class属性，用来指定一个Java类的名字，该类用来为该持久化类的实例生成唯一的标识，所以也叫生成器（generator）。Hibernate提供了多种内置的生成器，表11-1给出了生成器的名称。

11.5 映射文件详解

（5）<property>元素

- <property>元素用来映射实体类的普通属性，通过该元素能够详细地对数据表的字段进行描述。<property>元素的常用配置属性及说明如表11-2所示。

11.5 映射文件详解

- 从映射文件可以看出，它在持久化类与数据库之间起着桥梁的作用，映射文件的建立描述了持久化类与数据表之间的映射关系，同样也告诉了Hibernate数据表的结构等信息。

本章内容

- 11.5 映射文件详解
- 11.6 配置文件详解
- 11.7 关联映射
- 11.8 组件属性映射
- 11.9 继承映射
- 11.10 Hibernate数据查询
- 11.11 其他查询技术

11.6 配置文件详解

- Hibernate配置文件用来配置Hibernate运行的各种信息，在Hibernate应用开始运行时要读取配置文件信息。
- 配置文件可以使用
  - 属性文件的格式，文件名为 **hibernate.properies**
  - 也可以使用XML文件格式，文件名为 **hibernate.cfg.xml，**
- 在Hibernate系统中使用后者比较方便一些。

11.6 配置文件详解

- 11.6.1 hibernate.properties
- 11.6.2 hibernate.cfg.xml
-

11.6.1 hibernate.properties

- 在Hibernate的project\etc目录中有一个hibernate.properties样例文件，该文件是属性文件，其中定义了各种配置参数，但每个配置参数前面使用了“#”注释符号。
- 当我们需要使用hibernate.properties文件时，修改该样例文件即可，把该文件复制到应用的类路径下（CLASSPATH），然后将需要的配置项前面的“#”注释符去掉即可。

11.6.1 hibernate.properties

- 下面是该文件中的一个片段，用来定义数据库连接参数。

```
## PostgreSQL
#hibernate.dialect
   org.hibernate.dialect.PostgreSQLDialect
#hibernate.connection.driver_class
   org.postgresql.Driver
#hibernate.connection.url
   jdbc:postgresql:template1
#hibernate.connection.username pg
#hibernate.connection.password pg
```

11.6.2 hibernate.cfg.xml

- 在Hibernate解压目录的project\etc目录中也有一个hibernate.cfg.xml文件，它可作为配置文件模板，内容如下。

11.6.2 hibernate.cfg.xml

```
<!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>
        <property name="show_sql">true</property>
        <mapping  resource="org/hibernate
                /Simple.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

1. 数据库连接配置

- Hibernate支持两种数据库连接方式：JDBC和JNDI方式。
- 使用基本JDBC连接数据库，需要指定数据库驱动程序、URL、用户名和密码等属性值。

1. 数据库连接配置

name属性值	说　明
connection.driver_class	设置数据库驱动程序类名
connection.url	设置数据库连接的URL
connection.username	设置连接数据库使用的用户名
connection.password	设置连接数据库使用的密码
dialect	指定连接数据库使用的Hibernate方言

2. 数据库方言配置

- Hibernate底层仍然使用SQL语句执行数据库操作，虽然所有关系型数据库都支持标准的SQL，但不同数据库的SQL还是有一些语法差异，因此Hibernate使用数据库方言来识别这些差异。
- 一旦为Hibernate设置了合适的数据库方言，Hibernate就可以自动处理数据库访问所存在的差异。

2. 数据库方言配置

表11.4 常用数据库的方言

数据库名	方　言
DB2	org.hibernate.dialect.DB2Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
MySQL5	org.hibernate.dialect.MySQL5Dialect
Oracle 11g	org.hibernate.dialect.Oracle10gDialect
Sybase	org.hibernate.dialect.SybaseASE15Dialect
Microsoft SQL Server 2008	org.hibernate.dialect.SQLServer2008Dialect
Pointbase	org.hibernate.dialect.PointbaseDialect

3. 连接池配置

- 使用数据库连接池技术可以明显提高数据库应用的效率。
- Hibernate提供了JDBC连接池功能，它通过hibernate.connection.pool_size属性指定的，这是Hibernate自带的连接池的配置参数。

3. 连接池配置

- 在Hibernate开发中经常使用第三方提供的数据库连接池技术，如c3p0连接池。要使用c3p0连接池，需要将Hibernate解压目录lib\optional\c3p0中的两个jar文件添加到WEB-INF\lib目录中。
- 在配置文件中使用下面代码配置c3p0连接池。

3. 连接池配置

```
<!-- 配置最大连接数-->
<property name="hibernate.c3p0.max_size">100</property>
<!-- 配置最小连接数-->
<property name="hibernate.c3p0.min_size">5</property>
<!-- 配置连接的超时时间，如果超过这个时间会抛出异常，单位毫秒-->
<property name="hibernate.c3p0.timeout">5000</property>
<!-- 配置最大的PreparedStatement的数量-->
<property name="hibernate.c3p0.max_statement">100</property>
<!-- 配置每隔多少秒检查连接池里的空闲连接，单位秒-->
<property name="hibernate.c3p0.idle_test">120</property>
<!-- 配置当连接池中连接用完后，C3P0一次分配的新的连接数-->
<property name="hibernate.c3p0.acquire_increment">2</property>
<!-- 配置是否每次都验证连接是否可用-->
<property name="hibernate.c3p0.validate">false</property>
```

4. 其他常用属性配置

- 在配置文件中还可以配置许多其他属性，如JNDI数据源的连接属性、Hibernate事务属性、二级缓存相关属性以及外连接抓取属性等。表11.5给出了其他一些常用属性。

4. 其他常用属性配置

属性名	说　明
hibernate.show_sql	是否在控制台显示Hibernate生成的SQL语句，值为true或false
hibernate.format_sql	是否将SQL语句转换成格式良好的SQL，值为true或false
hibernate.use_sql_comments	是否在Hibernate生成的SQL语句中添加有助于调试的注释
hibernate.jdbc.fetch_size	指定JDBC抓取数量的大小，它接受一个整数值，其实质是调用Statement.setFetchSize()方法
hibernate.jdbc.batch_size	指定Hibernate使用JDBC的批量更新大小，它接受一个整数值，建议取5到30之间的值
hibernate.connection.autocommit	设置是否自动提交。通常不建议打开自动提交
hibernate.bhm2ddl.auto	设置当创建SessionFactory时，是否根据映射文件自动建立数据库表。该属性取值可以为create、update和create-drop等

本章内容

- 11.5 映射文件详解
- 11.6 配置文件详解
- 11.7 关联映射
- 11.8 组件属性映射
- 11.9 继承映射
- 11.10 Hibernate数据查询
- 11.11 其他查询技术

11.7 关联映射

- 在一个应用系统中，数据库可能有多个数据表，这些表之间具有一定的引用关系，反映到实体中就是实体之间的关联。

11.7.1 实体关联类型

- 在关系数据库中，实体与实体之间的联系有一对一、一对多、多对一和多对多4种类型。在Hibernate中实体类之间也存在这4种关联类型。

11.7.1 实体关联类型

- 一对一：一个实体实例与其他实体的单个实例相关联。例如，一个人（Person）只有一个身份证（IDCard），人和身份证之间就是一对一关系。

11.7.1 实体关联类型

- 一对多：一个实体实例与其他实体的多个实例相关联。例如，在订单系统中，一个订单（Order）和订单项（OrderItem）具有一对多的关系。

11.7.1 实体关联类型

- 多对一：一个实体的多个实例与其他实体的单个实例相关联。这种情况和一对多的情况相反。在人力资源管理系统中，员工（Employee）和部门（Department）之间就是多对一的关系。

11.7.1 实体关联类型

- 多对多：实体A的一个实例与实体B的多个实例相关联，反之，实体B的一个实例与实体A的多个实例相关联。例如，在大学里，一门课程（Course）有多个学生（Student）选修，一名学生可以选修多门课程。因此，学生和课程之间具有多对多的关系。

11.7.2 单向关联和双向关联

- 实体关联的方向可以是单向的（unidirectional）或双向的（bidirectional）。
- 在单向关联中，只有一个实体具有引用相关联实体的字段。例如，OrderItem具有一个标识Product的字段，但是Product则没有引用OrderItem的字段。换句话说，通过OrderItem可以知道Product，但是通过Product并不能知道是哪个OrderItem实例引用它。

11.7.2 单向关联和双向关联

- 在双向关联中，每个实体都具有一个引用相关联实体的字段。通过关联字段，实体类的代码可以访问与它相关的对象。
- 例如，如果Order知道它具有什么OrderItem实例，而且如果OrderItem知道它属于哪个Order，则它们具有一种双向关联。

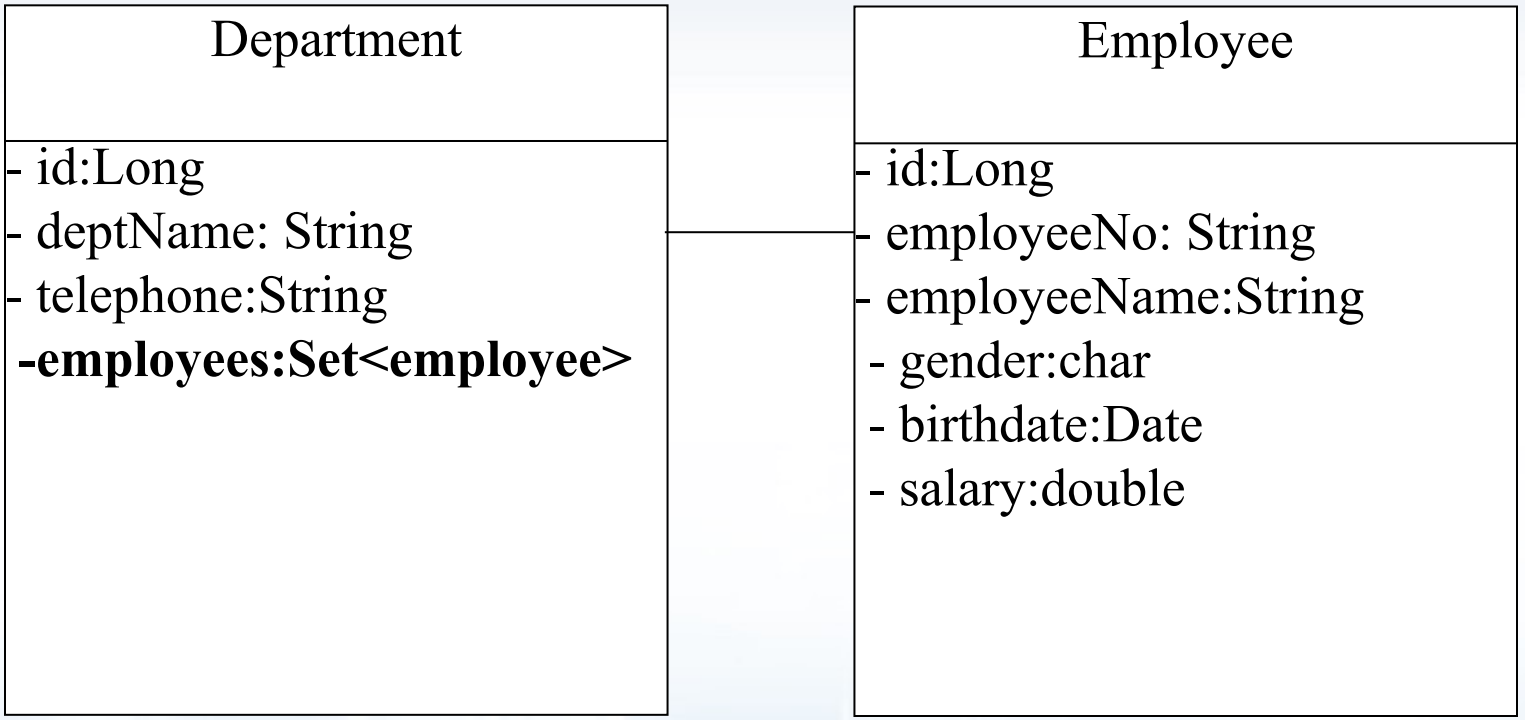
11.7.3 关联方向与查询

- HQL查询语言的查询通常会跨关系进行导航。关联的方向决定了查询能否从某个实体导航到另外的实体。
- 例如，如果从Department实体到Employee实体具有单向关联，则可以从Department导航到Employee，反之不能。但如果这两个实体具有双向关联，则也可以从Employee实体导航到Department实体。

11.7.4　一对多关联映射

- 一对多关联最常见，例如一个部门（Department）有多个员工（Employee）就是典型的一对多联系，如图11.3所示。在实际编写程序时，一对多关联有两种实现方式：单向关联和双向关联。单向一对多关联只需在一方配置映射，而双向一对多关联需要在关联的双方进行映射。

11.7.4 一对多关联映射



1. 单向关联

- 为了让两个持久类支持一对多的关联，需要在“一”方的实体类中增加一个属性，该属性引用“多”方关联的实体。
- 具体来说，就是在Department类中增加一个Set<Employee>类型的属性，并且为该属性定义setter和getter方法。

- 下面是Employee类的定义：
- 下面是Department类的定义：

1. 单向关联

- 对于单向的一对多关联只需在“一”方实体类的映射文件中使用<one-to-many>元素进行配置，即只需配置Department的映射文件Department.bhm.xml，如下所示。

1. 单向关联

•<class name="Department" table="department" lazy="true"><id name="id" column="id">
<generator class="identity" /> </id> <property name="deptName" type="string" column="dept_name" /><property name="telephone" type="string" column="telephone" /> <set name="employees" table="employee" lazy="false" inverse="false" cascade="all" sort="unsorted"> <key column="dept_id"/> <!—关联表(多方)的外键名--> <one-to-many class="com.hibernate.Employee" /> </set>
</class>

修改配置文件

- 在hibernate.cfg.xml文件中加入下面配置映射文件的代码：

```
<mapping resource="com/hibernate/Employee.hbm.xml"/>
<mapping resource="com/hibernate/Department.hbm.xml"/>
```

程序运行

- 下面代码创建了一个Department对象depart和两个Employee对象，并将它们持久化到数据库表中。

程序运行

```
Session session = HibernateUtil.getSession();
        Transaction tx = session.beginTransaction();
    Employee emp1 = new Employee("901","王小明",'M',
new GregorianCalendar(1972,11,20),3500.00),
                emp2 = new Employee("902","张大海",'F',
new GregorianCalendar(1989,5,14),4800.00);
    Set<Employee> employees = new HashSet<Employee>();
    employees.add(emp1);
    employees.add(emp2);
    Department depart = new Department("软件开发部
","3400222",employees);
    session.save(depart);
    tx.commit();
```

程序运行

- 上述代码执行后在department表中插入一条记录，在employee表中插入两条记录。对于单向的一对多关联，查询时只能从一方导航到多方，如下所示。

```
String query_str = "from Department d inner join d.employees e";
Query query = session.createQuery(query_str);
        List list =query.list();
        for(int i = 0; i < list.size(); i ++){
            Object obj[] =(Object[])list.get(i);
            Department dept = (Department)obj[0]; // dept是数组中第一个对象
            Employee emp = (Employee)obj[1];     // emp是数组中第二个对象
            System.out.println(dept.getDeptName()+ ":" +emp.getEmployeeName());
        }
```

2． 双向关联

- 如果要设置一对多双向关联，需在“多”方的类（如Employee）中添加访问“一”方对象的属性和setter及getter方法。
- 例如，如果要设置Department和Employee的双向关联，需在Employee类中添加下面代码：

2． 双向关联

```
private Department department;
public Department getDepartment(){
    return this.department;
}
public void setDepartment(Department
department){
    this.department = department;
}
```

2． 双向关联

- 在“多”方的映射文件Employee.hbm.xml中使用<many-to-one>元素定义多对一关联。代码如下：

```
<many-to-one  name="department"
  class="com.hibernate.Department"
cascade="all" outer-join="auto"
  column="dept_id"/>
```

2． 双向关联

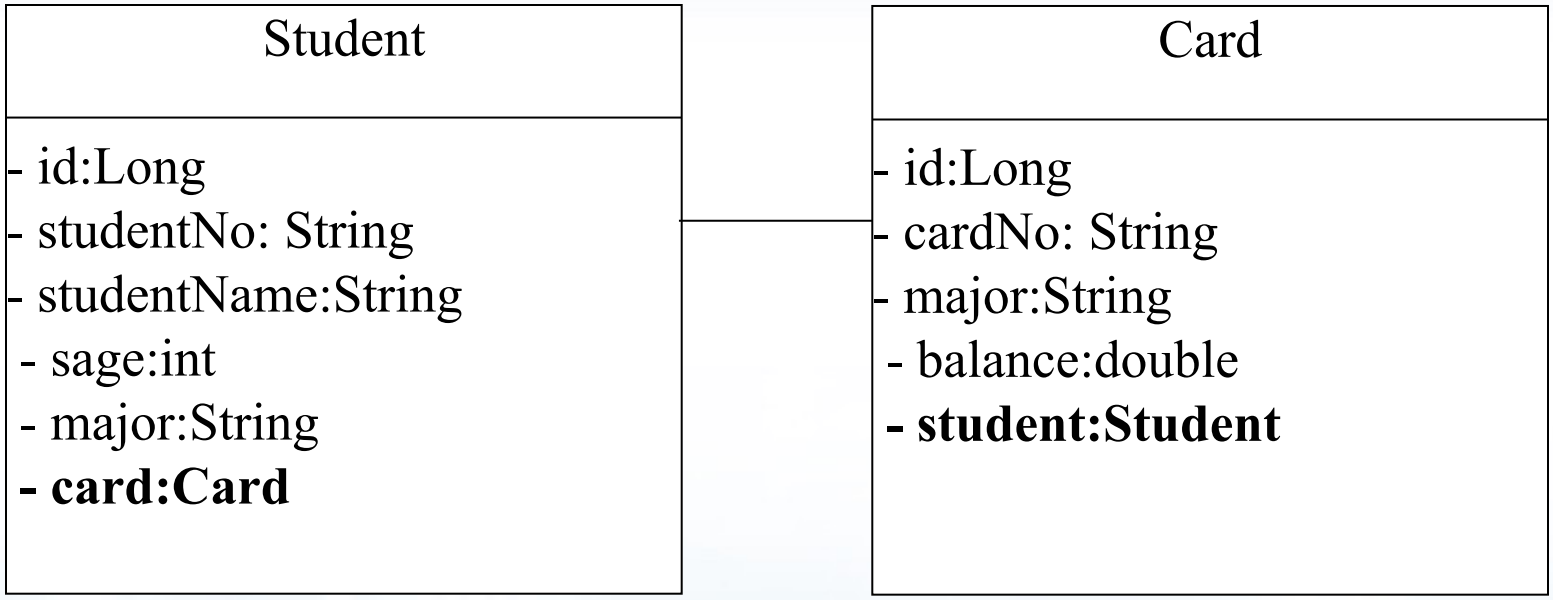
- 此外，还需要把Department.bhm.xml中的<set>元素的inverse属性值设置为true，如下所示。

```
<set name="employees"
   table="employee"     lazy="false"
   inverse="true"     cascade="all"
   sort="unsorted">     <key
   column="dept_id"/>     <one-to-many
   class="com.hibernate.Employee" />
   </set>
```

11.7.5 一对一关联映射

- 一对一关联在实际应用中也比较常见，例如学生（Student）与学生的校园卡（Card）之间就具有一对一的关联关系，如图11.4所示。

11.7.5 一对一关联映射



11.7.5 一对一关联映射

- 一对一关联也分为单向的和双向的，它需要在映射文件中使用<one-to-one>元素映射。另外，一对一关联关系在Hibernate中的实现有两种方式：主键关联和外键关联。

1． 主键关联

- 主键关联是指关联的两个实体共享一个主键值，即主键值相同。例如，Student和Card是一对一关系，它们在数据库中对应的表分别是student和card。两个关联的实体在表中具有相同的主键值，这个主键值可由student表生成，也可由card表生成。在另一个表中要引用已经生成的主键值需要通过映射文件中使用主键的foreign生成机制。

1． 主键关联

- 为了建立Student和Card之间的双向一对一关联，首先在Student类和Card类中添加引用对方对象的属性及setter和getter方法。

1． 主键关联

- 在Student类中添加下面代码。

```
private Card card;  // 一个Card类型的属性
public Card getCard(){
   return this.card;
}
public void setCard(Card card){
   this.card = card;
}
```

1． 主键关联

- 在Card类中添加下面代码。

```
private Student student;  // 一个Student类
  型的属性
public Student getStudent (){
  return this.student;
}
public void setStudent (Student student){
  this. student = student;
}
```

1． 主键关联

- 在Student类的映射文件Student.hbm.xml的<class>元素中添加<one-to-one>元素，如下所示。

```
<one-to-one name="card"
   class="com.hibernate.Card"
cascade="all"  fetch="join"/>
```

- 这里，<one-to-one>元素的cascade属性值all表示当保存、更新当前对象时，级联保存、更新所关联的对象。

1． 主键关联

- 为了实现双向关联，在Card类的映射文件Card.hbm.xml的<class>元素中也需要添加<one-to-one>元素，如下所示。

1． 主键关联

```
<hibernate-mapping package="com.hibernate">
        <class name="Card" table="card" lazy="true">
        <id name="id" column="id">
                <generator class="foreign">
                 <param name="property">student</param>
                </generator>
        </id>
        <property name="cardNo" type="string" column="cardNo" />
        <property name="major" type="string" column="major" />
        <property name="balance" type="double" column="balance"
/>
        <one-to-one name="student" class="com.hibernate.Student"
                constrained="true" />
        </class>
</hibernate-mapping>
```

1． 主键关联

- 在hibernate.cfg.xml文件中加入下面配置映射文件的代码：

```
<mapping
  resource="com/hibernate/Student.hbm.xml"/>
<mapping
  resource="com/hibernate/Card.hbm.xml"/>
```

1． 主键关联

```
Session session = HibernateUtil.getSession();
Transaction tx = session.beginTransaction();
Student student = new Student(
                              20120101,"Akbar Housein",20,"电子商务
");
Card card = new Card("110101","电子商务",1500.00);
student.setCard(card);
card.setStudent(student);
session.save(student);     // 持久化学生对象
tx.commit();
```

2． 外键关联

- 一对一的外键关联是指两个实体各自有自己的主键，但其中一个实体用外键引用另一个实体。例如，Student实体对应表的主键是id，Card实体对应表的主键是id，设在card表中还有一个studentId属性，它引用student表的id列，在card表中studentId就是外键。
- 一对一关联实际是多对一关联的特例，因此在外键所在的实体的映射文件中使用<many-to-one>元素来建立关联。

2． 外键关联

•若仍建立双向关联，则Student.hbm.xml无需修改，修改后的Card.hbm.xml如下：

```
<hibernate-mapping package="com.hibernate">
        <class name="Card" table="card" lazy="true">
        <id name="id" column="id">
                <generator class="identity"> !-- 这里不再是foreign了 -->
                  <param name="property">student</param>
                </generator>
        </id>
        <property name="cardNo" type="long" column="cardNo" />
        <property name="major" type="string" column="major" />
        <property name="balance" type="double" column="balance" />
```

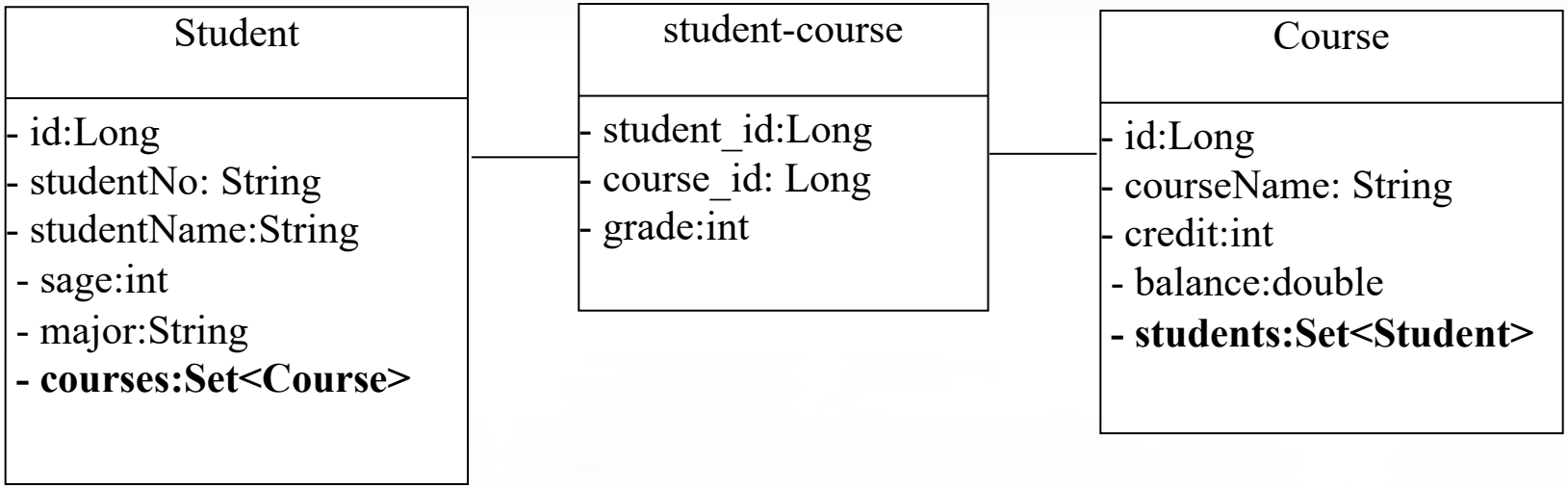
2． 外键关联

```
<many-to-one name="student" class="com.hibernate.Student"
                column="studentId" unique="true" />
        </class>
</hibernate-mapping>
```

11.7.6　多对多关联映射

- 学生（Student）实体和课程（Course）实体是最典型的多对多关联。可以设置单向的多对多关联，也可以设置双向的多对多关联。
- 本节主要讲解设置双向的多对多关联。在映射多对多关联时，需要另外使用一个连接表，如图11.5所示。

11.7.6 多对多关联映射



11.7.6 多对多关联映射

- 下面是连接表的定义：

```
CREATE TABLE student_course (
    student_id bigint NOT NULL,
    course_id bigint NOT NULL,
    grade integer DEFAULT 0,
    CONSTRAINT sc_pkey PRIMARY KEY (student_id,
    course_id)
)
```

11.7.6 多对多关联映射

- 对于双向的多对多关联，要求关联的双方实体类都使用Set集合属性，两端都增加集合属性的setter和getter方法。

11.7.6 多对多关联映射

- 在Student类中增加的代码如下。

```
private Set<Course> courses= new
   HashSet<Course>();
public void setCourses(Set<Course> courses){
  this.courses = courses;
}
public Set<Course> getCourses(){
  return courses;
}
```

11.7.6 多对多关联映射

- 下面是课程类Course的定义。

```
public class Course{
  private Long id;
  private String courseName;
  private double ccredit;
  private Set<Student> students = new
  HashSet<Student>();
```

11.7.6 多对多关联映射

- 对于双向多对多关联，需要在两端实体类的映射文件中都使用<set>元素定义集合属性并在其中使用<many-to-many>元素进行多对多映射。
- 在Student.hbm.xml文件中添加下面代码。

```
<set name="courses" table="student_course"
    cascade="all">
     <key column="student_id" />
     <many-to-many column="course_id"
    class="Course" />
</set>
```

11.7.6 多对多关联映射

- 为Course类创建一个映射文件Course.hbm.xml，如下所示。

- 将映射文件添加到配置文件hibernate.cfg.xml中。

```
<mapping
  resource="com/hibernate/Course.hbm.xml"/>
```

11.7.6 多对多关联映射

- 下面代码创建了三门课程对象，然后创建两个学生对象并将它们持久化到数据库中。

```
Session session = HibernateUtil.getSession();
Transaction tx = session.beginTransaction();
Student student1 = new Student(20120101,"王小明
    ",18,"计算机科学"),
student2 = new Student(20120102,"李大海",20,"电
    子商务");
```

1． 添加关联关系

- 现在要求为一名学生增加选修一门课程“数据库原理”，可以先得到学生对象，然后得到该学生选课集合，最后在该集合中增加一门课程，代码如下。

```
Student student = (Student)session.createQuery(
"from Student s where s.studentName='王小明
    '").uniqueResult();
Course course = new Course("数据库原理",3.5);
student.getCourses().add(course);
session.save(student);
```

1． 添加关联关系

- 如果要增加的课程已在数据表中存在，可以使用下列代码得到课程对象：

```
Course course =
    (Course)session.createQuery(
"from Course c where
    c.courseName='Database Principle'")
.uniqueResult();
```

2． 删除关联关系

- 删除关联关系比较简单，直接调用对象集合的remove()方法删除不要的对象即可。例如，要删除学生“王小明”选修的“数据结构”和“操作系统”两门课程，代码如下：

2． 删除关联关系

```
Student student = (Student)session.createQuery(
"from Student s where s.studentName='王小明
'").uniqueResult();
Course course1 = (Course)session.createQuery(
"from Course c where c.courseName='数据结构
'").uniqueResult();
Course course2 = (Course)session.createQuery(
"from Course c where c.courseName='操作系统
'").uniqueResult();
student.getCourses().remove(course1);
student.getCourses().remove(course2);
session.save(student);
```

运行上述代码，将从数据表student_course中删除两条记录，但student表和course表并没有任何变化。

本章内容

- 11.5 映射文件详解
- 11.6 配置文件详解
- 11.7 关联映射
- 11.8 组件属性映射
- 11.9 继承映射
- 11.10 Hibernate数据查询
- 11.11 其他查询技术

11.8 组件属性映射

- 如果持久化类的属性不是基本数据类型，也不是字符串、日期等标量类型的变量，而是一个复合类型的对象，这样的属性称为组件属性。
- 例如，对Person类可能有个address属性，它的类型是Address。

11.8 组件属性映射

Address类的主要代码如下。

```
public class Address {
   private String city;
   private String street;
   private String zipcode;
   public Address(){}  // 默认构造方法
   public Address(String city, String street, String zipcode) {
            this.city = city;
            this.street = street;
            this.zipcode = zipcode;
   }
   // 这里省略了属性的setter和getter方法
}
```

11.8 组件属性映射

Person类的主要代码如下。

```
public class Person {
   private Long id;
private String name;
   private int age;
   private Address address;
   public Person(){}
public Person(Long id, String name, int age,Address address) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.address = address;
```

11.8 组件属性映射

- Person类的address属性是组件属性。显然，在数据表中不能使用一个普通的列存储Address对象，因此不能直接使用<property>元素映射。
- 为了映射组件属性，Hibernate提供了<component>元素，用name属性指定要映射的组件属性名，用class属性指定组件类名。
- Person..hbm.xml映射文件的主要代码如下。

11.8 组件属性映射

```
<component name="address" class="Address">
        <property name="city" />
        <property name="street" />
        <property name="zipcode" />
</component>
```

11.8 组件属性映射

- 如果组件类型的属性是基本数据类型、String类型、日期类型时，使用<property>元素进行映射。
- 如果组件类的属性又是另一个组件类，或者是List、Set、Map等类型的集合属性，则需要在<component>元素中再次使用<component>子元素，或<set>、<list>、<map>等子元素进行映射

11.8 组件属性映射

- 将映射文件Person..hbm.xml添加到配置文件hibernate.cfg.xml中。

<mapping resource="com/hibernate/Person.hbm.xml"/>

- 执行下列代码将在person表中插入一行记录，该记录包括Person类中的属性和Address类中的属性。

11.8 组件属性映射

```
Session session = HibernateUtil.getSession();
Transaction tx = session.beginTransaction();
Person person = new Person();
Address address = new Address("北京","前门外大街15号","110011");
person.setName("王小明");
person.setAge(20);
person.setAddress(address);
session.save(person);
tx.commit();
```

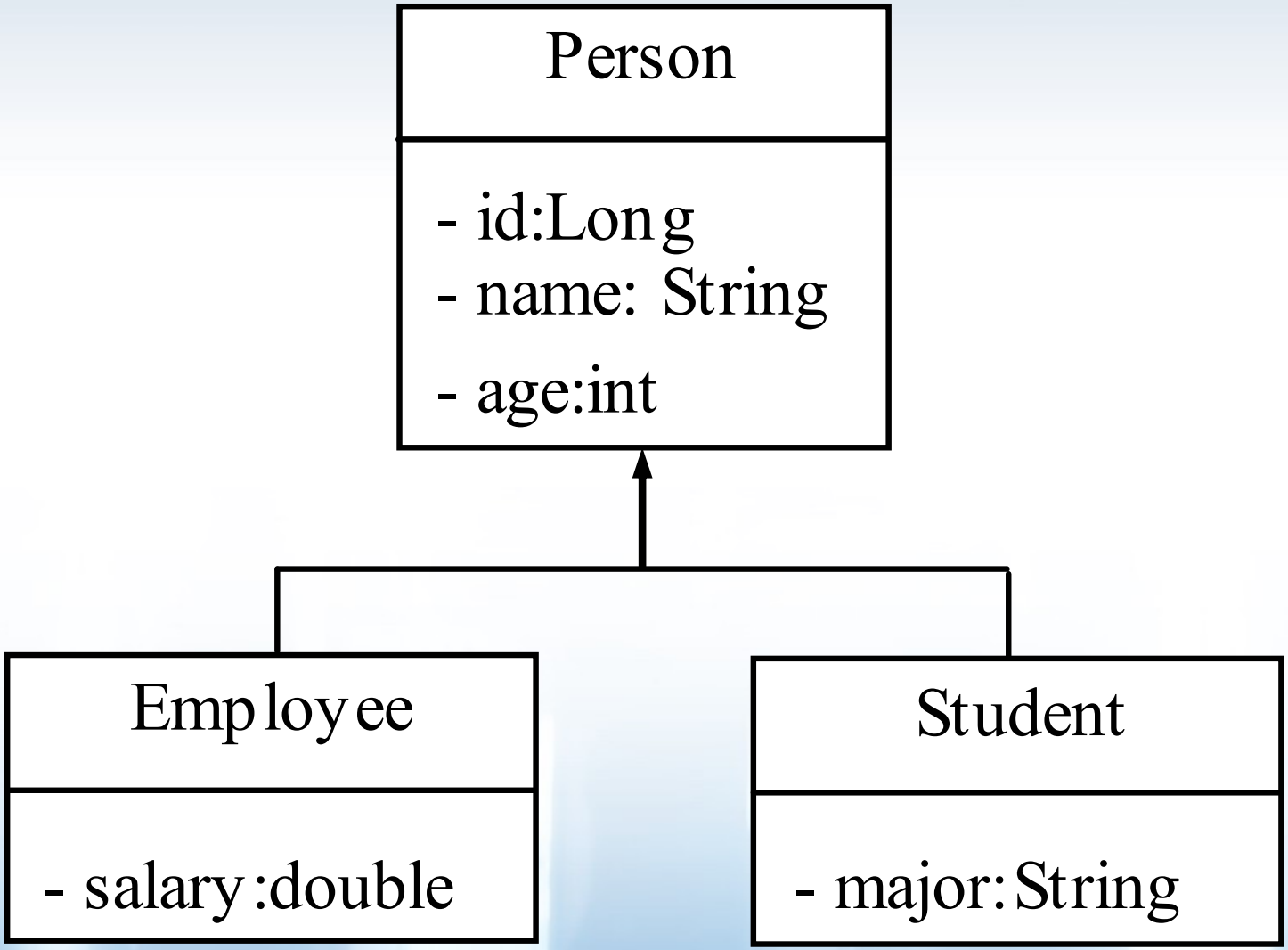
本章内容

- 11.5 映射文件详解
- 11.6 配置文件详解
- 11.7 关联映射
- 11.8 组件属性映射
- 11.9 继承映射
- 11.10 Hibernate数据查询
- 11.11 其他查询技术

11.9 继承映射

- Hibernate为具有继承关系的类也提供了映射的方法。例如，Person类、Employee类和Student类之间就具有继承关系，如图11.7所示。

11.9 继承映射


Person
- id:Long
- name: String
- age:int
Employee
- salary:double
Student
- major:String

11.9 继承映射

- Hibernate实现对象的继承映射主要有三种方式：
  - 所有类映射成一张表，
  - 每个子类映射成一张表，
  - 每个具体子类映射成一张表，
- 在实际应用中可根据需要进行选择。

11.9.1 所有类映射成一张表

- 这种映射策略是将整个继承树的所有实例都保存在一个数据库表中。对上述的继承关系就是将Person实例、Employee实例和Student实例都保存在同一个表中。
- 为了在一个表内区分哪一行是Employee、哪一行是Student，就需要为表增加一列，该列称为判别者（discriminator）列。

11.9.1 所有类映射成一张表

- 在这种映射策略下，使用<discriminator >元素指定判别者列，使用<subclass>元素指定子类及其属性。

- 下面是映射文件Person.bhm.xml的主要内容。

11.9.1 所有类映射成一张表

- 在主类中我们使用下面代码创建3个对象，然后把它们持久化到数据表中。

```
Person p = new Person(new Long(101),"王小明",25);
Student stud = new Student(new Long(102),"李大海",23,"计算机科学");
Employee emp = new Employee(new Long(301),"刘明",24,3800);
session.save(p);
session.save(stud);
session.save(emp);
```

11.9.2 每个子类映射成一张表

- 采用每个子类映射一张表的策略使用<joined-subclass>标记，父类实例保存在父类表中，子类实例则由父类表和子类表共同存储。
- 对于子类中属于父类的属性存储在父类表中。使用这种映射策略不需要使用判别者列，但需要为每个子类使用key元素映射共有主键，该主键的列表必须与父类标识属性的列名相同。

11.9.2 每个子类映射成一张表

- 使用<joined-subclass>映射策略的映射文件Person.bhm.xml的具体内容。

```
<joined-subclass name="Employee “
        table="employee">
    <key column="person_id" />
    <property name="salary" type="double">
      <column name="salary" />
    </property>
</joined-subclass>
```

11.9.3 每个具体类映射成一张表

- 采用每个具体类映射一张表使用<union-subclass>标记。采用这种映射策略，父类实例的数据保存在父表中，子类实例的数据仅保存在子表中。由于子类具有的属性比父类多，所以子类表的字段要比父类表的字段多。
- 在这种映射策略下，既不需要使用判别者列，也不需要<key>元素来映射共有主键。如果单从数据库来看，几乎难以看出它们存在继承关系。

11.9.3 每个具体类映射成一张表

- 采用<union-subclass>标记的继承映射文件代码如下。

```
<union-subclass name="Employee “
                table="employee">
    <property name="salary" type="double">
        <column name="salary" />
    </property>
</union-subclass>
```

本章内容

- 11.5 映射文件详解
- 11.6 配置文件详解
- 11.7 关联映射
- 11.8 组件属性映射
- 11.9 继承映射
- 11.10 Hibernate数据查询
- 11.11 其他查询技术

11.10 Hibernate数据查询

- 数据查询是Hibernate的最常见操作。Hibernate提供了多种查询方法：
  - HQL
  - 条件查询
  - 本地SQL查询
  - 命名查询

11.10.1 HQL查询概述

- HQL（Hibernate Query Language）称为Hibernate查询语言，它是Hibernate提供的一种功能强大的查询语言。
- HQL与SQL类似，用来执行对数据库的查询。当在程序中使用HQL时，它将自动产生SQL语句并对底层数据库查询。
- HQL使用类和属性代替表和字段。HQL功能非常强大，它支持多态、关联，并且比SQL简洁。

11.10.1 HQL查询概述

- 一个HQL查询语句可能包含下面元素：子句、聚集函数和子查询。
- 子句包括from子句、select子句、where子句、order by子句和group by子句等。
- 聚集函数包括avg()、sum()、min()、max()和count()等。
- 子查询是嵌套在另一个查询中的查询，如果底层数据库支持子查询，则Hibernate将支持子查询。

11.10.1 HQL查询概述

- HQL查询结果是Query实例，每个Query实例对应一个查询对象。使用HQL查询的一般步骤如下：
  - （1）获取Session对象。
  - （2）编写HQL语句。
  - （3）以HQL语句作为参数，调用Session对象的createQuery()方法创建Query对象。
  - （4）如果HQL语句包含动态参数，则调用Query的setXxx()方法设置参数值。
  - （5）调用Query对象的list()方法或iterate()方法返回查询结果列表（持久化实体集）。

11.10.2 查询结果处理

- 调用Session对象的createQuery()方法返回一个Query对象，在该对象上迭代可以返回结果对象。
- 有两种方法处理查询结果：
  - 在Query实例上调用list()方法返回List对象
  - 调用iterate()方法返回Iterator对象

11.10.2 查询结果处理

- 下面代码说明如何使用list()方法返回List对象，然后通过其get()方法检索每个Student持久类实例。

```
String query_str="from Student as s";
Query query = session.createQuery(query_str);
List<Student> list = query.list();
for(int i = 0; i < list.size(); i ++){
   Student stud =(Student)list.get(i);
   System.out.println("学号: " + stud.getStudentNo());
   System.out.println("姓名: " + stud.getStudentName());
}
```

11.10.2 查询结果处理

- 下面代码说明如何使用iterate()方法返回Iterato对象，然后在其上迭代获得每个Student持久类的实例。

```
Query query = session.createQuery(query_str);
for(Iterator<Student> it = query.iterate();it.hasNext()
   Student stud =(Student)it.next();
   System.out.println("学号: " + stud.getStudentNo());
   System.out.println("姓名
: " + stud.getStudentName());
 }
```

11.10.3 HQL的from子句

- from子句是最简单的HQL语句，也是最基本的HQL语句。from关键字后紧跟持久化类的类名，例如：

```
from Student
```

- 表示从Student持久化类中选出全部实例，实际是查询student表中所有记录。。

11.10.3 HQL的from子句

- 通常，在from中为持久化类名指定一个别名，例如：

  from Student as s

- 命名别名时，as关键字是可选的，但为了增加可读性，建议保留。
- from子句后面还可出现多个持久化类，此时将产生一个笛卡尔积或多表连接，但实际上这种用法很少使用。

11.10.4 HQL的select子句

- 有时并不需要得到对象的所有属性，这时可以使用select子句指定要查询的属性，例如：

```
select s.studentName from Student s
```

11.10.4 HQL的select子句

- 下面代码说明如何执行该语句：

```
String query_str = "select s.studentName from Student s";
Query query = session.createQuery(query_str);
List<String> list = query.list();
for(int i = 0; i < list.size(); i ++){
    String sname =(String)list.get(i);
   System.out.println("姓名: " + sname);
 }
```

11.10.4 HQL的select子句

- 如果要查询两个以上的属性，查询结果会以对象数组的方式返回，如下代码所示。

```
String query_str =
"select s.studentNo,s.studentName,s.major from
Student s";
Query query = session.createQuery(query_str);
List<Object[]> list =query.list();
for(int i = 0; i < list.size(); i ++){
    Object obj[] =(Object[])list.get(i);
    System.out.println("学号: " + obj[0]);
    System.out.println("姓名: " + obj[1]);
   System.out.println("专业: " + obj[2]);
}
```

11.10.5 HQL的聚集函数

- 可以在HQL的查询中使用聚集函数。HQL支持的聚集函数与SQL完全相同，有如下5个。
  - count()：统计查询对象的数量。
  - avg()：计算属性的平均值。
  - sum()：计算属性的总和。
  - min()：统计属性值的最小值。
  - max()：统计属性值的最大值。

11.10.5 HQL的聚集函数

- 例如，要得到Student实例的数量，可使用如下语句。

```
select count(*) from Student
```

11.10.5 HQL的聚集函数

- 要得到全体Student实例的平均年龄，可使用如下语句。

```
String hql = "select avg(s.sage) from Student as s";
Query query = session.createQuery(hql);
List list =query.list();
System.out.println("平均年龄: " + list.get(0));
```

11.10.6 HQL的where子句

- 使用where子句筛选查询结果，缩小查询范围。如果没有为持久化实例指定别名，可以直接使用属性名来引用属性，例如：

```
from Student where studentName like
  'Akaba%'
```

- 上面的HQL语句与下面的语句效果相同。

```
from Student as s where
  s.studentName like  'Akaba%'
```

11.10.6 HQL的where子句

- 在where子句中可以使用各种运算符和函数构成复杂的表达式，常用的运算符如下：
  - 数学运算符：+、-、*、/等。
  - 比较运算符：=、>=、<=、>、<、!=、like等。
  - 逻辑运算符：not、and、or等。
  - 字符串连接符：||，它实现两个字符串连接。其用法是value1||value2。
  - 集合运算符：in、not in、between、is null、is not null、is empty、is not empty、member of、not member of等。

11.10.6 HQL的where子句

- 在where子句中可以使用的函数包括:
  - 算术函数：abs()、sqrt()、sign()、sin()等。
  - SQL标量函数：substring()、trim()、lower()、upper()、length()等。
  - 时间操作函数：current_date()、current_time()、current_timestamp()、hour()、minute()、second()、day()、month()、year()等。

11.10.6 HQL的where子句

- 下面语句查询有一名员工年龄为22岁，其所在部门的信息。

```
from Department d where 22=any(select s.age from d.employees e)
```

11.10.7 HQL的order by子句

- HQL查询语句返回的集合可以根据类或组件属性的任何属性进行排序。例如：

```
from Student as s order by s.sage
```

- 可以使用asc和desc指定按升序或按降序排序，例如：

```
from Student as s order by s.studentNo asc,s.sage desc
```

- 默认为升序规则。

11.10.8 HQL的group by子句

- 与SQL语言一样，在HQL查询语句中可以使用group by子句对查询结果分组。
- 请看下面示例：

```
select s.gender, avg(s.sage) from Student as s
group by s.gender having avg(s.sage) > 20
```

- 与SQL规则相同，having子句必须与group by子句配合使用，不能单独使用。

11.10.9 带参数的查询

- 如果HQL查询语句中带有参数，则在执行查询语句之前需要设置参数。
  - 如果使用的是命名参数，应该使用setParameter()方法设置
  - 如果使用的是占位符（？），则应该使用setXxx()方法设置
- 常用方法如下。

11.10.9 带参数的查询

- Query setParameter(String name, Object val)：将指定的对象值绑定到指定名称的参数上。
- Query setParameter(int position, Object val)：将指定的对象值绑定到指定位置的参数上。
- Query setInteger(String name, int val)： 将指定的整数值val绑定到指定名称的参数上。
- Query setInteger(String name, int val)： 将指定的整数值val绑定到指定名称的参数上。

11.10.9 带参数的查询

- 大多数方法都有两种格式，一种是通过名称为指定的参数赋值，一种是JDBC风格的通过问号为指定的参数赋值。
- 使用名称指定参数，然后使用setParameter()方法为指定的参数赋值，例如：

```
Query query = session.createQuery("from Student s
where s.sage > :age");  // 用名称指定一个参数
    query.setParameter("age" , 20);
```

11.10.9 带参数的查询

- Hibernate也支持JDBC风格的查询参数，即使用问号（?）作为占位符，然后使用Query接口的set*Xxx*()方法设置参数值。例如：

```
Query query = session.createQuery("from Student s
where s.sage > ?"); // 用?指定一个参数
 query.setInteger(0, 20);
```

11.10.10 关联和连接

- 如果程序使用的数据来自多个表，可以使用SQL语句的连接查询。
- 在Hibernate中则使用关联映射来处理底层数据表之间的连接。一旦我们建立了正确的关联映射后，就可利用Hibernate的关联来进行连接。
- HQL支持两种关联连接形式：显式（explicit）连接和隐式（implicit）连接。

11.10.10 关联和连接

- 显式连接需要使用join关键字，这与SQL连接表类似。HQL所支持的连接类型借鉴了SQL99的多表连接，具体支持如下几种连接方式：
  - inner join，内连接，可简写成join。
  - left outer join，左外连接，可简写成left join。
  - right outer join，右外连接，可简写成right join。
  - full outer join，全外连接，可简写成full join。

11.10.10 关联和连接

- 下面代码查询各Department情况以及该系Student情况，代码如下。

```
String query_str = "from Department d inner join d.students s";
Query query = session.createQuery(query_str);
List<Object[]>  list =query.list();
for(int i = 0; i < list.size(); i ++){
Object obj[] =(Object[])list.get(i);
  Department dept = (Department)obj[0]; // dept是数组中第一个对象
Student stud = (Student)obj[1];  // stud是数组中第二个对象
System.out.println(stud.getStudentName()+ " 系:" +dept.getDeptName());
}
```

11.11 小　结

- Hibernate是轻量级的O/R映射框架，它用来实现应用程序的持久化功能。
- 本章介绍Hibernate的框架结构、核心组件和运行机制，接下来介绍了映射文件和配置文件，之后详细讨论了关联映射、组件映射和继承映射。
- 最后介绍Hibernate 数据查询语言HQL的使用、条件查询、本地SQL查询以及命名查询等。