# 1. Hibernate的关联映射

要想充分使用Hibernate的强大功能，必须先设计好数据库，分析实体关系。然后根据数据库表的关系写好Hibernate的映射文件。

数据库中实体之间的常用关系就有多对一、多对多和一对一，那么在Hibernate映射中也要体现实体间的联系，这样使用Hibernate效果最好。下面通过例子来讲解使用关联映射。

## 1.1 多对一/一对多映射

例子：部门与员工之间的关系就是一对多的关系，即一个部门下有多个员工。我们需要在映射文件中体现这一关系。

我们可以不自己建立数据库表，因为Hibernate有自动建表功能。我们按照逻辑写好映射文件后，直接使用自动建表，也正好可以通过查看建好的表来验证映射的正确性。

下面是开发的步骤。

（1）先设计好实体类。基于部门与员工的关系，那么部门实体类中应该有维护所有员工的属性，员工实体类中也应该有维护一个部门的属性。例如：

部门实体类：

|  |
| --- |
| **package** com.zhang.entity;  **public class** Dept {  **private** String **deptId**; *// 编号* **private** String **name**; *// 部门名称* **private** Set<Employee> **employeeSet** = **new** LinkedHashSet<>(); *// 维护的员工。  // 用set集合表示，便于关联。用LinkedHashSet保证元素有序*  *// 自行写setter/setter/toString()等方法* } |

员工实体类：

|  |
| --- |
| **package** com.zhang.entity;  **public class** Employee {  **private** String **eid**; *// 编号* **private** String **name**; *// 姓名* **private** Dept **dept**; *// 维护的部门。即员工所在的部门。*  *// 自行写getter/setter/toString()等方法* } |

注意实体类都应该写上无参数的构造方法，否则主键查询会出错。

（2）配置映射文件

Dept.hbm.xml：

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* **<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"*>*** <**hibernate-mapping package="com.zhang.entity"**>  <**class name="Dept" table="dept"**>  *<!-- 主键 -->* <**id name="id" column="id"**>  *<!-- 这里配置主键生成策略为自己指定 -->* <**generator class="assigned"** />  </**id**>  *<!-- 部门名称 -->* <**property name="name" column="name" length="20"** />   *<!-- 一对多关系配置 -->  <!--set中，指定employeeSet属性所关联的表是employee表-->* <**set name="employeeSet" table="employee"**>  *<!-- key标签中，用column属性指定本表的deptId是外键 -->* <**key column="deptId"** />  *<!--one-to-many表明一对多映射，多的一方类型是Employee类型-->* <**one-to-many class="Employee"** />  </**set**>  </**class**> </**hibernate-mapping**> |

Employee.hbm.xml：

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* **<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"*>*** <**hibernate-mapping package="com.zhang.entity"**>  <**class name="Employee" table="employee"**>  *<!-- 主键 -->* <**id name="eid" column="eid"**>  <**generator class="assigned"** />  </**id**>  *<!-- 员工姓名 -->* <**property name="name" column="name" length="20"** />    *<!--多对一配置。使用many-to-one -->  <!--  name：表示本对象对应的属性  class：表示对象该属性对应的类型  column：指明外键字段  -->* <**many-to-one name="dept" class="Dept" column="deptId"**></**many-to-one**>  </**class**> </**hibernate-mapping**> |

也就说，一的一方使用set标签，多的一方使用many-to-one。最后生成表时，会在多的一方生成外键字段。

（3）配置Hibernate主配置文件。注意加上自动建表的配置，并且引入上述两个映射文件，即：

|  |
| --- |
| <**property name="hbm2ddl.auto"**>update</**property**> <**mapping resource="com/zhang/entity/Dept.hbm.xml"** /> <**mapping resource="com/zhang/entity/Employee.hbm.xml"** /> |

（4）编写代码，测试程序。现在表是空的，想要添加一个“开发部”，并且“开发部”下有“张三”和“李四”两个人。则代码为：

|  |
| --- |
| **public class** Demo {  **public static void** main(String[] args) {  Session session = **new** Configuration().configure().buildSessionFactory().openSession();  session.beginTransaction();  *// 1.创建“开发部”* Dept dept = **new** Dept(**"BM001"**, **"开发部"**);  *// 2.创建人员* Employee zs = **new** Employee(**"YG001"**, **"张三"**);  Employee ls = **new** Employee(**"YG002"**, **"李四"**);  *// 3.直接将人员添加到部门中。即直接通过getEmployeeSet().add()方法添加，很方便。* dept.getEmployeeSet().add(zs);  dept.getEmployeeSet().add(ls);  *// 4. 使用session保存这些部门和人员* session.save(zs);  session.save(ls);  session.save(dept);  session.getTransaction().commit();  session.close();  session.getSessionFactory().close();  } } |

初次运行程序时，发现Hibernate会自动创建表。再查看数据库，发现Hibernate正确的维护了表之间的关系，员工表中有外键指向部门表。并且所插入的数据也正确的维护了关系，因为我们代码中就说明了这个关系，通过部门的getEmployeeSet().add()直接维护了关系，完全是面向对象的方法。可见Hibernate十分强大。

但是我们发现，一共插入了三条数据，但Hibernate会产生5条数据库操作，即：

|  |
| --- |
| Hibernate: insert into employee (name, deptId, eid) values (?, ?, ?)  Hibernate: insert into employee (name, deptId, eid) values (?, ?, ?)  Hibernate: insert into dept (name, id) values (?, ?)  Hibernate: update employee set deptId=? where eid=?  Hibernate: update employee set deptId=? where eid=? |

出现这样的原因是Hibernate一开始会直接添加好数据，并没有维护外键，比如添加员工时，所插入的deptId值是NULL。当数据添加完后，Hibernate再来维护外键关系，就是通过两条update语句改变员工的deptId为正确的部门编号。

实际上，后两条update是可以不需要的，因为本身就可以先添加部门，然后添加员工时，根据他们的关系直接维护好deptId。要实现这一点，只要通过“多”的一方（即员工方）来维护关系，并且，要先保存“一”的一方，即先保存该部门数据。即代码为：

|  |
| --- |
| **public class** Demo {  **public static void** main(String[] args) {  Session session = **new** Configuration().configure().buildSessionFactory().openSession();  session.beginTransaction();  *// 1.创建“开发部”* Dept dept = **new** Dept(**"BM001"**, **"开发部"**);  *// 2.创建人员* Employee zs = **new** Employee(**"YG001"**, **"张三"**);  Employee ls = **new** Employee(**"YG002"**, **"李四"**);  *// 3. 用员工维护关系* zs.setDept(dept);  ls.setDept(dept);  *// 4. 保存数据* session.save(dept); *// 先保存部门* session.save(zs);  session.save(ls);  session.getTransaction().commit();  session.close();  session.getSessionFactory().close();  } } |

这样做可提高效率。

## 1.2 关于使用Set集合

我们在配置一对多（包括下面的多对多）关系时，使用的是Set集合，Set集合是无序的，但是我们可以在“set”标签中配置“order-by”属性，以便对集合进行排序，这样拿到的Set集合就是“有序”的。例如：

|  |
| --- |
| <**set name="employeeSet" table="employee" order-by="eid"**>  <**key column="deptId"** />  <**one-to-many class="Employee"** /> </**set**> |

order-by中指定的是“employee”表中的字段，如果“employee”中有age字段，还能按照年龄排序，即order-by="age"，这样从Dept中取到的Set<Employee>就是年龄从小到大排序了。默认是升序排序的，也可设置为降序排序，即设置desc，比如：order-by="age desc"。实现的原理就是通过SQL的“order by”查询的。

## 1.3 关于懒加载

做好关联映射后，在Hibernate中，只要获得某方数据，那么可直接通过对象的属性得到其关联对象的数据，不需要再编写查询语句进行查询。这就是Hibernate的懒加载（Lazy Load）功能。

使用懒加载十分方便，只有在用到相关的对象时，懒加载才会查询额外数据，而并不是先把所有的关联数据都查询出来，否则效率很低。并且Hibernate是有缓存的，已经查询过的数据不会重复查询。

使用懒加载前提是当前session没有被关闭，如果session被提前关闭了，就不能使用对象的属性获取数据了，否则出现以下报错信息：

org.hibernate.LazyInitializationException: could not initialize proxy - no Session

懒加载示例代码：

|  |
| --- |
| **package** com.zhang.test;  **public class** Demo {  **public static void** main(String[] args) {  Session session = **new** Configuration().configure().buildSessionFactory().openSession();  session.beginTransaction();  *// 得到员工* Employee employee = session.get(Employee.**class**, **"YG001"**);  *// 员工姓名* System.***out***.println(employee.getName());  *// 部门名称* System.***out***.println(employee.getDept().getName());  *// 员工该部门下所有的员工* Set<Employee> employeeSet = employee.getDept().getEmployeeSet();  **for** (Employee emp : employeeSet) {  System.***out***.println(emp.getName());  }  *// 不要用system.out.println(employee)直接输出一个对象。  // 因为他们会相互调用，出现死循环而出错。* session.getTransaction().commit();  session.close();  session.getSessionFactory().close();  } } |

可以在映射文件中，设置class或者set节点的lazy属性值，默认为true，即使用懒加载。也可以设置成false，即关闭懒加载，这样Hibernate查询时就会先查出所有关联的数据。

使用Hibernate时，经常会出现懒加载异常问题，原因就是session被提前关闭了，解决方法有：

（1）在session关闭之前使用一下懒加载的数据；

（2）在映射文件的package节点中，设置default-lazy值为false，这样就会关闭懒加载，一开始就会查出所有的关联数据。

## 1.4 多对多映射

例子：项目（Project）与开发人员（Developer）之间的关系就是多对多关系：

一个项目，可有多个开发人员；一个开发人员，可参与多个项目。因此，项目与开发人员之间还有一张表，即“项目人员表”。

映射数据库表时，我们使用peoject、developer和relation三张表，其中relation采用复合主键。

（1）写Project和Developer两个实体类

Project：

|  |
| --- |
| **package** com.zhang.entity;  **public class** Project {  **private** String **pid**; *// 项目主键* **private** String **pname**; *// 项目名称* **private** Set<Developer> **developerSet** = **new** LinkedHashSet<>();   *// setter/getter...* } |

Developer：

|  |
| --- |
| **package** com.zhang.entity;  **public class** Developer {  **private** String **devId**; *// 开发者编号* **private** String **devName**; *// 开发者姓名* **private** Set<Project> **projectSet** = **new** LinkedHashSet<>();  *// getter/setter...* } |

两个实体类中的Set集合分别表示项目对应的人员和人员对应的项目。

（2）映射文件

Project.hbm.xml：

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* **<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"*>*** <**hibernate-mapping package="com.zhang.entity"**>  <**class name="Project" table="project"**>  <**id name="pid"**>  <**generator class="assigned"**></**generator**>  </**id**>  <**property name="pname" length="20" column="pname"**></**property**>  *<!-- 多对多映射 -->  <!-- 还是用set标签指明属性和关系表。这时关系表叫relation-->* <**set name="developerSet" table="relation"**>  *<!--key中指明本表的外键-->* <**key column="pid"**></**key**>  *<!-- 这时应该用many-to-many指明要关联的列名和类型-->* <**many-to-many column="devId" class="Developer"**></**many-to-many**>  </**set**>  </**class**> </**hibernate-mapping**> |

Developer.hbm.xml：也是用多对多，和上述“相反”。

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* **<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"*>*** <**hibernate-mapping package="com.zhang.entity"**>  <**class name="Developer" table="developer"**>  <**id name="devId"**>  <**generator class="assigned"**></**generator**>  </**id**>  <**property name="devName" length="20" column="devName"**></**property**>  *<!-- 多对多映射 -->  <!-- 关系表还是relation-->* <**set name="projectSet" table="relation"**>  *<!--key中指明本表的外键-->* <**key column="devId"**></**key**>  *<!-- many-to-many指明要关联的列名和类型-->* <**many-to-many column="pid" class="Project"**></**many-to-many**>  </**set**>  </**class**> </**hibernate-mapping**> |

（3）编写测试类：例如：张三和李四OA系统；张三和王五开发图书馆系统。

|  |
| --- |
| **public class** Demo {  **public static void** main(String[] args) {  Session session = **new** Configuration().configure().buildSessionFactory().openSession();  session.beginTransaction();  *// 1. 项目对象* Project oa = **new** Project(**"PJ001"**, **"OA系统"**);  Project lib = **new** Project(**"PJ002"**, **"图书馆系统"**);  *// 2. 开发者对象* Developer zs = **new** Developer(**"DEV001"**, **"张三"**);  Developer ls = **new** Developer(**"DEV002"**, **"李四"**);  Developer ww = **new** Developer(**"DEV003"**, **"王五"**);  *// 3. 维护对象间关系* oa.getDeveloperSet().add(zs);  oa.getDeveloperSet().add(ls);  lib.getDeveloperSet().add(zs);  lib.getDeveloperSet().add(ww);  *// 4. 保存所有对象* session.save(oa);  session.save(lib);  session.save(zs);  session.save(ls);  session.save(ww);    session.getTransaction().commit();  session.close();  session.getSessionFactory().close();  } } |

由于是多对多关系，任意一方维护好关系即可，并且不存在“优化”SQL执行次数问题。我们查看数据库表，relation是一张关系表。大家自行测试获取数据的方法。

## 1.5 一对一映射

例如一个用户和他的身份证信息就是一对一的关系。一般是用户表只存储基本信息，然后身份证表存储身份证信息，并且身份证表有外键关联用户表。

（1）实体类：

User：

|  |
| --- |
| **package** com.zhang.entity;  **public class** User {  **private** String **userId**; *// 用户编号* **private** String **userName**; *// 用户姓名* **private** IDCard **idCard**; *// 对应的身份证* } |

IDCard：

|  |
| --- |
| **package** com.zhang.entity; **public class** IDCard {  **private** String **cardNum**; *// 身份证号（主键）* **private** String **address**; *// 地址* **private** User **user**; *// 对应的用户* } |

（2）映射文件

User.hbm.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* **<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"*>*** <**hibernate-mapping package="com.zhang.entity"**>  <**class name="User" table="user"**>  <**id name="userId"**>  <**generator class="assigned"** />  </**id**>  <**property name="userName" length="20" column="userName"** />  *<!-- 一对一映射，这是主表，直接写类型即可。关联在外键一方配置 -->* <**one-to-one name="idCard" class="IDCard"** />  </**class**> </**hibernate-mapping**> |

IDCard.hbm.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* **<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"*>*** <**hibernate-mapping package="com.zhang.entity"**>  <**class name="IDCard" table="idCard"**>  <**id name="cardNum"**>  *<!-- 主键映射。 -->* <**generator class="assigned"** />  </**id**>  <**property name="address"** />  *<!-- 一对一映射。但是用many-to-one标签，因为此标签可维护外键信息。 -->  <!-- column指明外键的列名，由于是一对一，这里用unique为true表示唯一 -->* <**many-to-one name="user" class="User" column="user\_Id" unique="true"**></**many-to-one**>  </**class**> </**hibernate-mapping**> |

（3）测试。注意，由于外键关系是通过IDCard维护的，因此代码中，也要通过IDCard维护关联。即用IDCard对象设置其用户，而不能通过用户设置其IDCard信息。而获取也是单向获取。

|  |
| --- |
| **public class** Demo {  **public static void** main(String[] args) {  Session session = **new** Configuration().configure().buildSessionFactory().openSession();  session.beginTransaction();  User user = **new** User(**"U001"**, **"张三"**);  IDCard idCard = **new** IDCard(**"321322"**, **"江苏"**);  idCard.setUser(user);  session.save(user);  session.save(idCard);  session.getTransaction().commit();  session.close();  session.getSessionFactory().close();  } } |

# 2. Hibernate常用技术

## 2.1 添加连接池技术

Hibernate自带就有一个连接池，不过此连接池只设定了一个连接，比较简陋，不适合在生产环境中使用。

Hibernate可支持C3P0连接池，下载的Hibernate文件中也提供了C3P0相关包和整合的JAR包。我们只要把lib/optional/c3p0目录下所有JAR包引入到项目中即可。然后，需要在主配置文件hibernate.cfg.xml中配置连接池：

|  |
| --- |
| *<?***xml version='1.0' encoding='utf-8'***?>* **<!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="connection.driver\_class"**>com.mysql.jdbc.Driver</**property**>  <**property name="connection.url"**>jdbc:mysql://localhost:3306/dbtest?useSSL=false**&amp;**useUnicode=true**&amp;**characterEncoding=UTF8</**property**>  <**property name="connection.username"**>root</**property**>  <**property name="connection.password"**>123456</**property**>  <**property name="dialect"**>org.hibernate.dialect.MySQLDialect</**property**>  <**property name="show\_sql"**>true</**property**>  <**property name="hibernate.format\_sql"**>true</**property**>  <**property name="hbm2ddl.auto"**>update</**property**>  *<!-- 连接池配置 -->  <!-- 配置连接池驱动管理类 -->* <**property name="connection.provider\_class"**>org.hibernate.c3p0.internal.C3P0ConnectionProvider</**property**>  *<!-- 连接池参数信息 -->  <!-- 最小连接数 -->* <**property name="c3p0.min\_size"**>2</**property**>  *<!-- 最大连接数 -->* <**property name="c3p0.max\_size"**>4</**property**>  *<!-- 超时时间 -->* <**property name="c3p0.timeout"**>5000</**property**>  *<!-- 最多执行的命令条数 -->* <**property name="c3p0.max\_statements"**>10</**property**>  *<!-- 空闲测试时间，定期向数据库发送信息，防止连接中断 -->* <**property name="c3p0.idle\_test\_period"**>30000</**property**>  *<!-- 连接不够用时，每次增加的连接数 -->* <**property name="c3p0.acquire\_increment"**>2</**property**>  <**mapping resource="com/zhang/entity/User.hbm.xml"** />  <**mapping resource="com/zhang/entity/IDCard.hbm.xml"** />  </**session-factory**> </**hibernate-configuration**> |

## 2.2 Hibernate的缓存

为了减少Hibernate访问数据库的次数，提高效率，Hibernate使用了一级缓存和二级缓存。

（1）一级缓存，也叫做session缓存，可以在session范围内减少访问数据库次数。Session关闭，当前一级缓存就失效。

（2）当调用session的save/saveOrUpdate/get/load/list/iterator方法时，都会把对象放入session缓存中，因为这些对象是使用较为频繁的。

## 2.3 项目中如何管理session

为了更好地管理session，应该用线程的方式创建session，这样当线程结束时，session也会自动关闭。我们需要在hibernate.cfg.xml文件中加入配置：

|  |
| --- |
| <**property name="current\_session\_context\_class"**>thread</**property**> |

然后，在代码中就能用sessionFactory.getCurrentSession()方法得到session了，而不是用之前的openSession()方法。因为用openSession()的方法，每次都会创建一个新的session对象，不利于资源利用率提高。

并且，以线程方式创建的session可以不用手动关闭，线程结束session自动关闭。总体上，这样进行管理session较好，项目中一般也会使用这样的方式。

# 3. Hibernate逆向工程

Hibernate是JBoss负责维护的，JBoss提供了一个名叫“Hibernate Tool”的Eclipse插件，该插件能根据我们在数据库中设计好的数据库表，自动生成实体类和映射文件！这种根据数据库自动生成Hibernate相关文件的方法，我们称之为Hibernate的“逆向工程”。

为何生成Hibernate相关文件是可行的呢？因为我们手动编写Hibernate配置文件也大部分是根据设计的表来进行的。对于较复杂的表结构，使用Hibernate Tool能大幅减少开发者工作，但前提是表的设计必须合理且适合使用Hibernate框架。当然，利用Hibernate Tool生成的文件也不是100%满足要求，有时我们还需根据实际情况进行微调，如懒加载配置、级联设置等。

## 3.1 环境配置

Hibernate Tools包含在JBoss Tools工具包中，因此在Eclipse中安装JBoss Tools插件中的Hibernate Tool即可。如果安装不能成功，可以直接使用“JBoss Developer Studio”开发工具，该开发工具是JBoss基于Eclipse EE开发的集成开发环境，其中已经集成了JBoss Tools全部工具。

## 3.2 使用Hibernate Tool

进行Hibernate的逆向工程，需要做如下操作：

（1）在Eclipse中新建数据源。

在“Window – Show View”中选择打开“Data Source Explorer”。在打开的选项卡中右击“Database Connections”，点击“New…”菜单项以便连接数据库。

在弹出的框中选择好使用的数据库（如MySQL），可输入名称Name和描述Description，点击Next。Drivers中选择合适的驱动。初次使用时可能没有合适的驱动，需要自己点击选择框右边的图标按钮“New Driver Definition”进行添加。添加时，我们可选择最新的5.1版本驱动，当然此时也要在“JAR List”选项卡中选择好对应的5.1版本的JDBC驱动包。

驱动加载好后，再设置MySQL数据库配置，输入数据库名、URL等，然后就可点击Finish完成了。完成后，Database Connections中就能看到连接到的数据库和表（Schemas）。

（2）配置Hibernate Configurations

在“Window – Show View”中选择打开Hibernate Configurations。在该标签中，右键“Add Configuration…”，可以给该配置起个名字，默认为“hibernate”。选择“Type”为“Core”。选择好Hibernate版和Project项目，并且设置“Database connection”为刚刚设置的MySQL数据源连接后，就可点击OK完成配置了。

（3）进行逆向工程

点击“Run – Hibernate Code Generation… - Hibernate Code Generation Configurations…”。选中其中的“Hibernate Code Generation”，再点击左上角的“新建（New launch configuration）”图标新建一个配置，可起名为“hibernate逆向工程”，在Main选项卡中，选择“Console configuration”为刚设置的“hibernate”，设置Output directory为本项目的src路径；勾选“Reverse engineer from JDBC Connection”，并设置包名（如com.entity）。除此之外，Main选项卡无需再手动配置（如果想指定某几张数据库表来生成Hibernate映射，则可设置reveng.xml）。

在“Exporters”选项卡中，勾选“Use Java 5 syntax”，表示会使用Java 5的语法，即会使用泛型，推荐使用。若勾选“Generate EJB3 annotations”，表示是使用注解方式，此时就不用生成映射文件了。

在“Reporters”中，一般会勾选“Domain code”、“Hibernate XML Mappings”和“DAO code”，分别表示生成实体类、映射文件和基本的DAO层代码。按需选择即可。完毕后，点击“Run”即可生成代码和文件，逆向工程完毕。