一对多映射配置

以学生，老师关系为例子， 老师为一，学生为多。

第一步：创建俩个实体类： 老师 和学生。

第二步：让俩个实体类之间相互表示

1、在老师实体类中表示包含多个学生

一个老师能包含多个学生（注：在 mybatis 中是使用 List 来构造这种关系，但是 hibernate 中 要求是使用Set来关联一对多的关系）

```
private Set<Student> students = new HashSet<>();

    public Set<Student> getStudents() {
        return students;
    }
    public void setStudents(Set<Student> students) {
        this.students = students;
    }
```

2、在学生实体类中表示所属老师

一个学生只能属于一个老师（在多的那一个实体类，添加一个从属于的实体类的属性，来指定从属于的实体类）

```
private Teacher teacher = new Teacher();;

    public Teacher getTeacher() {
        return teacher;
    }
    public void setTeacher(Teacher teacher) {
        this.teacher = teacher;
    }
```

第三步： 配置映射关系

1、一般一个实体类对应一个映射文件

2、把映射最基本配置完（参考如下配置）

```
<?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>
    <class name="daiwei.learning.hibernate.pojo.Teacher" table="tb_teacher">
        <id name="TeaId" column="id">
            <generator class="native"></generator>
        </id>
        <property name="name" column="name"></property>
        <property name="gender" column="gender"></property>
    </class>
</hibernate-mapping>
```

3、在映射文件中，配置一对多的关系

- 在老师映射文件中，表示所有学生

id：外键，在 hibernate 的机制当中，要双向维护外键，即在一的那一方和多的那一方都要写外键属性

```
<set name="students" >
            <key column="tid"></key>
            <one-to-many class="daiwei.learning.hibernate.pojo.Student"/>
        </set>
```

- 在学生映射文件中，表示所属老师

```
<many-to-one name="teacher" class="daiwei.learning.hibernate.pojo.Teacher"
column="tid"></many-to-one>
```

（注：调试到此处遇到的问题：

1、在映射文件中不能有空的 <property> 标签

2、在 pojo 实体类中属性的 set get方法的方法名一定要规范，如果方法名改了，建议重新创建 set 和 get方法

3、看报错能帮助，快速定位出错位置

）；

一对多操作 级联保存

方法一 （较复杂）：

基本思路与步骤：

创建 A 类，补全信息 ---> 创建 B 类， 补全信息 ---> A、B 确认关系，相互设置属性 ---> 调用 session.save() 保存数据。

代码演示：

```
@Test
public void cascadeAdd() {
    Session session = null;
    Transaction tx = null;

    try {
        session = HibernateUtils.getSessionObj();
        tx = session.beginTransaction();
        tx.begin();

        //创建老师并补全老师信息
        Teacher teacher = new Teacher();
        teacher.setName("wk");
        teacher.setGender("man");
        teacher.setTeachedCourse("javaWeb");

        //创建学生对象，并补全学生信息
        Student student = new Student();
        student.setName("dw");
        student.setGender("man");

        //学生老师互相确认关系
        student.setTeacher(teacher);
        teacher.getStudents().add(student);

        //保存
        session.save(student);
        session.save(teacher);
        tx.commit();
    }catch (Exception e) {
        e.printStackTrace();
    } finally {

    }
```

```
    }
```

方法二（简便的写法）：

基本思路和步骤：

在双方的表示关系的配置中（从属类即techer 类中 的 set ）加入 cascade 配置，这样就可以做级联添加的（参考配置如下）；

```
<set name="students" cascade="save-update">
            <key column="tid"></key>
            <one-to-many class="daiwei.learning.hibernate.pojo.Student"/>
        </set>
```

--->创建 A类 并补全信息，创建 B 类，补全信息 ---> 俩个类相互确定关系 ---> 调用 session.save() 保存从属类（在本例中是teacher类） 即可。

详细测试代码：

```
    @Test
    public void cascadeAdd2() {
        Session session = null;
        SessionFactory factory = null;
        Transaction tx = null;

        try{
            factory = HibernateUtils.getSessionFactory();
            session = factory.openSession();
            tx = session.beginTransaction();
            tx.begin();

            Teacher tea = new Teacher();
            tea.setName("wk2");
            tea.setGender("man");
            tea.setTeachedCourse("javaBasic");

            Student stu = new Student();
            stu.setName("cjx");
            stu.setGender("man");

            tea.getStudents().add(stu);
            stu.setTeacher(tea);
```

```
24
25              session.save(tea);
26
27              tx.commit();
28          } catch (Exception e) {
29              e.printStackTrace();
30              tx.rollback();
31          }finally{
32              if(session != null) {
33                  session.close();
34                  factory.close();
35              }
36          }
37
38      }
```

一对多操作 级联删除

操作方法及基本步骤：

在从属方（在本例子中是teacher）的 set 的标签中的 cascade 添加 delete（这个测试时，不用再从属于方及 student 中的 cascade 中添加 delete）， 参考配置如下：

```
1  <set name="students" cascade="save-update,delete">
2              <key column="tid"></key>
3              <one-to-many class="daiwei.learning.hibernate.pojo.Student"/>
4          </set>
```

---> session.get() 方法获得要删除的对象（要是从属方对象即Teacher） ---> 调用session.delete()方法进行删除操作。

测试代码：

```
1      @Test
2      public void cascadeDelete() {
3          Session session = null;
4          Transaction tx = null;
```

```
5
6           try {
7               session = HibernateUtils.getSessionObj();
8               tx = session.beginTransaction();
9               tx.begin();
10
11              Teacher teacher = session.get(Teacher.class, 2);
12  //          Set<Student> students = teacher.getStudents();
13  //          Iterator<Student> iterator = students.iterator();
14  //          while(iterator.hasNext()) {
15  //              System.out.println(iterator.next());
16  //          }
17
18              session.delete(teacher);
19              tx.commit();
20          } catch (Exception e) {
21              e.printStackTrace();
22              tx.rollback();
23          } finally{
24
25          }
26      }
27
```

一对多操作 更新（inverse 属性)

操作方法和基本步骤：

在从属方（本例子中的teacher）的映射文件中的 set 标签添加 `inverse` 属性（默认 false） 设置为 true（不是必需的，但是俩个类都会维护外键，一方这一边主动放弃维护节约资源）；

属性 inverse ： 设置是否放弃对外键的维护（hibernate外键双维护的特性）

----> session.get() 获得要更新的对象和要更新的从属方 ---> 在双方互相确认关系 ---> 调用session.saveOrUpdate() 更新数据（这一步不是必须的，因为持久态会自动更新数据库）

测试代码：

```
1       @Test
2       public void updateTest() {
3           Session session = null;
```

```
        Transaction tx = null;
        try {
            session = HibernateUtils.getSessionObj();
            tx = session.beginTransaction();
            tx.begin();

            Teacher teacher = session.get(Teacher.class, 2);
            Student stu = session.get(Student.class, 5);
            teacher.getStudents().add(stu);
            stu.setTeacher(teacher);
            session.saveOrUpdate(teacher);

            tx.commit();
        } catch (Exception e) {
            tx.rollback();
            e.printStackTrace();
        } finally {

        }
    }
```