建立多表关系

一、表与表之间关系回顾

- 一对多

※案例：一个人可以拥有多辆汽车，要求查询出某人所拥有的所有汽车。

方案一(差的设计)：注意要求，如果是查每种车型的销量，那么一对多的一方又不一样

编号	姓名	性别	年龄	汽车编号	车型	排量	价格
P001	Jack	男	25	C001	BMW	12L	80w
P001	Jack	男	25	C002	Benz	12L	100w
P001	Jack	男	25	C003	Benz	12L	100w
P002	Tom	男	25	C004	BMW	12L	80w
P002	Tom	男	25	C005	Benz	12L	100w
P003	Rose	女	25	C006	Benz	12L	100w

方案二(好的设计)：

1）把一方单独建个表　要分清楚一方和多方，这里的是一方是人，我们的案例

编号	姓名	性别	年龄
P001	Jack	男	25
P002	Tom	男	25
P003	Rose	女	25

是要求查每个人买了几辆车　而不是查每种车型被多少个人购买

所以一方是人的信息表

2）把多方也建个表(依赖一方,通过外键--补一个字段)

外键:位于依赖一方，它是被依赖一方是主键

汽车编号	车型	排量	价格	车主
C001	BMW	12L	80w	P001
C002	Benz	12L	100w	P001
C003	Benz	12L	120w	P001
C004	BMW	12L	80w	P002
C005	Benz	12L	100w	P002
C006	Benz	12L	100w	P003

把一方的主键作为多方的外键即可

- 多对多

数据库设计分析

※案例：一个人可以选择多门课程，一门课程又可以被很多人选择。

方案一(差的设计)：

1)学生表

编号	姓名	性别	年龄	电话 ...
P001	Jack	男	25	
P002	Tom	男	25	
P003	Rose	女	25	

---

2)课程表

编号	名称	教材	学分...	学生
S001	Java	...	.....	P001
S001	Java	...	.....	P002
S001	Java	...	.....	...
S002	数据库	...	.....	P001
S002	数据库	...	.....	P002

方案二(好的设计 两个实体表+一个关系表)：

1)学生表(独立)---实体

编号	姓名	性别	年龄	电话 ...
P001	Jack	男	25	
P002	Tom	男	25	
P003	Rose	女	25	

2)课程表(独立)---实体

编号	名称	教材	学分...
S001	Java	...	.....
S002	数据库	...	.....
S003	XML	...	.....

3)选课表(专为体现多对多的关系而新增的表)--关系

课程编号	学生编号
S001	P001
S001	P002
...	
S002	P002
S002	P003
...	
S003	P001
...	

二、一对多操作

2.1 一对多关系配置

2.1.1 实体类中的体现

"一"的一方

```java
//在客户实体类中表示多个联系人，一个客户有多个联系人
  //用set集合表示
  private Set<Contacter> setContacter=new HashSet<Contacter>();
一的一方建立set集合存储多的一方的实体类对象，并生成set和get方法
    public Set<Contacter> getSetContacter() {
        return setContacter;
    }
    public void setSetContacter(Set<Contacter> setContacter) {
        this.setContacter = setContacter;
    }
```

"多"的一方

```java
    private Customer customer;

       生成一的一方的一个成员变量并生成get和set方法
    public Customer getCustomer() {
        return customer;
    }
    public void setCustomer(Customer customer) {
        this.customer = customer;
    }
```

2.1.2 映射文件中的体现

"一"的一方

set标签；外键名称key标签；one-to-many;

```xml
        <!-- 在客户映射文件中表示所有联系人
          使用set标签表示所有联系人
          -->          1.存储多的一方的那个set集合的变量名
          <set name="setContacter" cascade="save-update,delete" inverse="true">
             <!-- 一对多建表，有外键
             双向维护外键，在一和多的一方都要配置外键
             column:外键名称-->     2. 外键名称
             <key column="ccid"></key>
             <one-to-many class="cn.scct.entity.Contacter"/>3. 多的一方的类全路径名
          </set>
```

"多"的一方

many-to-one;

```xml
        <!-- 表示联系人所属的客户
        name属性：因为在联系人实体类使用Customer对象表示，所以写该变量的名称
        class写customer的全路径-->
        <many-to-one name="customer" class="cn.scct.entity.Customer" column="ccid"></many-to-one>
```

2.1.3 核心配置文件中的体现

```
    <!-- 第三步：
        路径书写： 填写src下的路径引入的映射文件
     -->
        <mapping resource="cn/scct/entity/Customer.hbm.xml" />
        <mapping resource="cn/scct/entity/Contacter.hbm.xml" />
```

执行工具类就能生成以上两个表

2.2 一对多级联操作

2.2.1 一对多级联保存

级联是有方向性的，所谓的方向性指的是，在保存一的一方级联多的一方和保存多的一方级联一的一方。

2.2.1.1 复杂朴素版

```
            //添加一个客户，为该客户添加一个联系人
            //创建客户和联系人的对象
            Customer customer=new Customer();
            customer.setCustName("华南理工");
            customer.setCustLevel("vip");
            customer.setCustSource("皮炎");
            customer.setCustPhone("123456");
            customer.setCustMobile("13289899999");

            Contacter contacter=new Contacter();
            contacter.setCon_name("张某");
            contacter.setCon_gender("男");
            contacter.setCon_phone("12345678901");

            //2.在客户表示联系人，在联系人中表示客户
            //建立客户对象与联系人对象的关系
              //2.1 把联系人放到客户实体类的set集合中即可
            customer.getSetContacter().add(contacter);
            contacter.setCustomer(customer);

            //3.保存到数据库
            session.save(customer);
            session.save(contacter);
```

也就是互相设置另一方的变量值而已

2.2.1.2 简化版

一般在一的一方添加多的一方，只需要在一的一方的配置文件中的set标签中设置cascade属性为save-update

```
<set name="setContacter" cascade="save-update,delete" inverse="true">
    <!-- 一对多建表，有外键
    双向维护外键，在一和多的一方都要配置外键
    column:外键名称-->
    <key column="ccid"></key>
    <one-to-many class="cn.scct.entity.Contacter"/>
</set>
```

然后代码就可以不要在多的一方设置一的一方的值了。

```
//添加一个客户，为该客户添加一个联系人
//创建客户和联系人的对象
Customer customer=new Customer();
customer.setCustName("华南理工");
customer.setCustLevel("vip");
customer.setCustSource("皮炎");
customer.setCustPhone("123456");
customer.setCustMobile("13289899999");

Contacter contacter=new Contacter();
contacter.setCon_name("张某");
contacter.setCon_gender("男");
contacter.setCon_phone("12345678901");

//2.在客户表示联系人，在联系人中表示客户
//建立客户对象与联系人对象的关系
  //2.1 把联系人放到客户实体类的set集合中即可
customer.getSetContacter().add(contacter);

//3.保存到数据库
session.save(customer);
```

只需要设置一的一方即可，而且只需保存一的一方

注意：如果也在多的一方设置cascade属性为save-update，那么保存多的一方也会保存一的一方，但是不常用。

2.2.2 一对多级联删除

原先的JDBC操作在删除一的一方时，必须在多的一方先删除以解除外键约束。
但是在hibernate中没有这个必要。在一的一方配置set标签的cascade属性上加上delete属性值即可。完成的是**删除一必删除多。**

```
<set name="setContacter" cascade="save-update,delete" inverse="true">
    <!-- 一对多建表，有外键
    双向维护外键，在一和多的一方都要配置外键
    column:外键名称-->
    <key column="ccid"></key>
    <one-to-many class="cn.scct.entity.Contacter"/>
</set>
```

删除的代码很简单，先查询获取再删除：

```
//一对多级联删除
//直接删客户就可
Customer customer=session.get(Customer.class,2);

session.delete(customer);
```

也可以配置多的一方的many-to-one标签中设置cascade属性，加上delete属性值，这样，删除多的一方会删除一的一方，但是不常用。因为一的一方是主表。

2.2.3 一对多级联修改

主表和从表默认双向都维护外键。

```
//1根据id查询联系人，根据ID查询要修改到的客户
Customer scct=session.get(Customer.class, 1);
Contacter chen=session.get(Contacter.class,4);
// 2.设置持久态对象的值，不需要调用update方法，自动更新
//把联系人放到客户中
scct.getSetContacter().add(chen);修改了一次外键
//把客户设置到联系人汇总
chen.setCustomer(scct);又修改了一次外键
```

我们一般让主表放弃外键维护，设置inverse属性为true。

```
<!-- 在客户映射文件中表示所有联系人
使用set标签表示所有联系人
-->
<set name="setContacter" cascade="save-update,delete" inverse="true">
    <!-- 一对多建表，有外键
    双向维护外键，在一和多的一方都要配置外键
    column:外键名称-->
    <key column="ccid"></key>
    <one-to-many class="cn.scct.entity.Contacter"/>
</set>
```

表示放弃外键维护

三、多对多操作

3.1 多对多建表

3.1.1 实体类创建

按照一对多的多一方设置，即**每一方都要有一个set集合维护另一方的对象**

```
private Set<Role> setRole=new HashSet<Role>();
public Set<Role> getSetRole() {
    return setRole;
}
public void setSetRole(Set<Role> setRole) {
    this.setRole = setRole;
}
```

```java
    private Set<User> setUser=new HashSet<User>();

    public Set<User> getSetUser() {
        return setUser;
    }
    public void setSetUser(Set<User> setUser) {
        this.setUser = setUser;
    }
```

3.1.2 配置文件设置

3.1.2.1 映射文件配置

```xml
        <!--
          使用set标签
          -->
          <set name="setRole" table="ur_map" cascade="save-update,delete">
             <!-- 多对多建表
             column:外键名称在第三张表的名称-->
             <key column="user_id"></key>
             <!-- class:角色实体类全路径
                  column：角色在第三张表外键的名称 -->
            <many-to-many class="cn.scct.entity.Role" column="role_id"></many-to-many>
          </set>
```

User表的配置
1.本实体类的另一个实体类变量的set集合名字
5.第三张表的名字
2.本表主键在第三张表的外键名称
4.另一张表在第三张表的外键名
3.映射表的全路径

另一张表同理

```xml
        <!--
          使用set标签
          -->
          <set name="setUser" table="ur_map" cascade="save-update">
             <!-- 多对多建表
             column:外键名称在第三张表的名称-->
             <key column="role_id"></key>
             <!-- class:角色实体类全路径
                  column：角色在第三张表外键的名称 -->
            <many-to-many class="cn.scct.entity.User" column="user_id"></many-to-many>
          </set>
```

3.1.2.2 核心配置文件配置

加上两个类的全路径名即可

```xml
        <mapping resource="cn/scct/entity/User.hbm.xml" />
        <mapping resource="cn/scct/entity/Role.hbm.xml" />

```

3.2 多对多级联操作

3.2.1 级联保存

和一对多类似，配置一个表的save属性值为save-update，只需调用一个表的save操作即可。

3.2.2 级联删除

同理，但基本不用啊，一般没有这种需求。

3.2.3 修改

也就是set集合的add和remove操作罢了。不提。