Hibernate第三天

第1章 多表设计

1.1 多表设计的总则

问题：我们为什么要学习多表映射？

答：

在实际开发中，我们数据库的表难免会有相互的关联关系，在操作表的时候就有可能会涉及到多张表的操作。试想一下，如果把我们 web 阶段的在线商城案例的持久层改为 hibernate 的实现，我们现在根本无法实现功能。究其原因是我们在线商城中表之间都是有关联关系的。

例如：商品和分类，用户和订单，订单和商品等等。

而通过第一天的 Hibernate 框架学习，我们知道 hibernate 实现了 ORM 的思想，可以让我们通过操作实体类就实现对数据库表的操作。

所以今天我们的学习重点是：掌握配置实体之间的关联关系。

要想实现多表映射，我们现阶段需要遵循的步骤：

第一步：首先确定两张表之间的关系。

如果关系确定错了，后面做的所有操作就都不可能正确。

第二步：在数据库中实现两张表的关系

第三步：在实体类中描述出两个实体的关系

第四步：配置出实体类和数据库表的关系映射

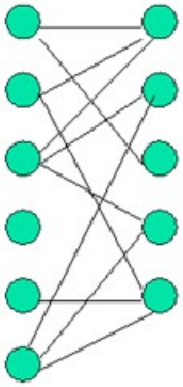
配置的方式支持注解和 XML，我们以注解为重点。

思考：表之间的关系到底有几种呢？

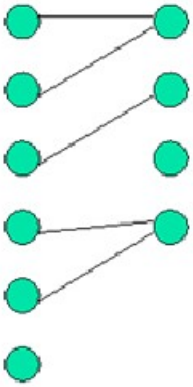
1.2 表之间的关系划分

Hibernate 框架实现了 ORM 的思想，将关系数据库中表的数据映射成对象，使开发人员把对数据库的操作转化为对对象的操作，Hibernate 的关联关系映射主要包括多表的映射配置、数据的增加、删除等。

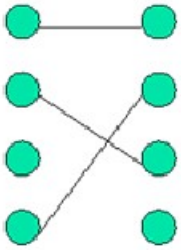
数据库中多表之间存在着三种关系，也就是系统设计中的三种实体关系。如图所示。



多对多
M：N



一对多
1：N



一对一
1：1

从图可以看出，系统设计的三种实体关系分别为：多对多、一对多和一对一关系。

注意：

一对多关系可以看为两种： 即一对多，多对一。所以说四种更精确。

明确：

我们只涉及实际开发中常用的关联关系，一对多和多对多。而一对一的情况，在实际开发中几乎不用。

1.3 数据库表和实体类的一对多关系

1.3.1 示例分析

我们采用的示例为 CRM 中的客户和联系人。
客户：通常情况下客户指的是一家公司。
联系人：一般都是指客户的员工。
在不考虑兼职的情况下，客户和联系人的关系即为一对多。

1.3.2 表关系建立

在一对多关系中，我们习惯把一的一方称之为主表，把多的一方称之为从表。在数据库中建立一对多的关系，需要使用数据库的外键约束。
什么是外键？
指的是从表中有一列，取值参照主表的主键，这一列就是外键。
一对多数据库关系的建立，如下图所示：

外键的定义：从表中有一列，该列的取值只能来源于主表的主键，我们把这一列称之为外键。外键默认情况下可以为null

客户表是主表

cst_customer

ID (PK)	Name	Industry
1	TBD云集中心	商业办公
2	修正大厦	医药制造
3	黑马程序员	IT教育
4	传智播客	IT教育
5	金燕龙办公楼	商业办公
6	中关村软件园	商业办公

一个客户可以包含多个联系人，多个联系人对应一个客户。客观和联系人之间的关系是：

一对多

一对多：靠数据库的外键约束实现

联系人表是从表

cst_linkman

ID(PK)	Name	Gender	lkm_cust_id
1	TBD联系人	male	1
2	修正联系人	male	2
3	黑马联系人	male	3
4	黑马联系人	male	3
5	金燕龙联系	male	5
6	软件园联系	male	6

1.3.3 实体类关系建立

在实体类中，由于客户是少的一方，它应该包含多个联系人，所以实体类要体现出客户中有多个联系人的信息，代码如下：

```
/**
 * 客户的实体类
 */
public class Customer implements Serializable {

    private Long custId;
    private String custName;
    private String custSource;
    private String custIndustry;
    private String custLevel;
    private String custAddress;
    private String custPhone;

    //一对多关系映射：一个客户可以对应多个联系人
    private Set<LinkMan> linkmans = new HashSet<LinkMan>();

    public Long getCustId() {
        return custId;
    }
    public void setCustId(Long custId) {
        this.custId = custId;
    }
    public String getCustName() {
        return custName;
    }
    public void setCustName(String custName) {
        this.custName = custName;
    }
```

```
    public String getCustSource() {
        return custSource;
    }
    public void setCustSource(String custSource) {
        this.custSource = custSource;
    }
    public String getCustIndustry() {
        return custIndustry;
    }
    public void setCustIndustry(String custIndustry) {
        this.custIndustry = custIndustry;
    }
    public String getCustLevel() {
        return custLevel;
    }
    public void setCustLevel(String custLevel) {
        this.custLevel = custLevel;
    }
    public String getCustAddress() {
        return custAddress;
    }
    public void setCustAddress(String custAddress) {
        this.custAddress = custAddress;
    }
    public String getCustPhone() {
        return custPhone;
    }
    public void setCustPhone(String custPhone) {
        this.custPhone = custPhone;
    }
    public Set<LinkMan> getLinkmans() {
        return linkmans;
    }
    public void setLinkmans(Set<LinkMan> linkmans) {
        this.linkmans = linkmans;
    }
    @Override
    public String toString() {
        return "Customer [custId=" + custId + ", custName=" + custName + ",
custSource=" + custSource
                + ", custIndustry=" + custIndustry + ", custLevel=" +
custLevel + ", custAddress=" + custAddress
                + ", custPhone=" + custPhone + "]";
    }
```

```
    }
```

由于联系人是多的一方，在实体类中要体现出，每个联系人只能对应一个客户，代码如下：

```
    /**
     * 联系人的实体类（数据模型）
     */
    public class LinkMan implements Serializable {

        private Long lkmId;
        private String lkmName;
        private String lkmGender;
        private String lkmPhone;
        private String lkmMobile;
        private String lkmEmail;
        private String lkmPosition;
        private String lkmMemo;

        //多对一关系映射：多个联系人对应客户
        private Customer customer;//用它的主键，对应联系人表中的外键

        public Long getLkmId() {
            return lkmId;
        }
        public void setLkmId(Long lkmId) {
            this.lkmId = lkmId;
        }
        public String getLkmName() {
            return lkmName;
        }
        public void setLkmName(String lkmName) {
            this.lkmName = lkmName;
        }
        public String getLkmGender() {
            return lkmGender;
        }
        public void setLkmGender(String lkmGender) {
            this.lkmGender = lkmGender;
        }
        public String getLkmPhone() {
            return lkmPhone;
        }
        public void setLkmPhone(String lkmPhone) {
            this.lkmPhone = lkmPhone;
        }
        public String getLkmMobile() {
```

```
            return lkmMobile;
        }
        public void setLkmMobile(String lkmMobile) {
            this.lkmMobile = lkmMobile;
        }
        public String getLkmEmail() {
            return lkmEmail;
        }
        public void setLkmEmail(String lkmEmail) {
            this.lkmEmail = lkmEmail;
        }
        public String getLkmPosition() {
            return lkmPosition;
        }
        public void setLkmPosition(String lkmPosition) {
            this.lkmPosition = lkmPosition;
        }
        public String getLkmMemo() {
            return lkmMemo;
        }
        public void setLkmMemo(String lkmMemo) {
            this.lkmMemo = lkmMemo;
        }
        public Customer getCustomer() {
            return customer;
        }
        public void setCustomer(Customer customer) {
            this.customer = customer;
        }
        @Override
        public String toString() {
            return "LinkMan [lkmId=" + lkmId + ", lkmName=" + lkmName + ",
lkmGender=" + lkmGender + ", lkmPhone="
                    + lkmPhone + ", lkmMobile=" + lkmMobile + ", lkmEmail=" +
lkmEmail + ", lkmPosition=" + lkmPosition
                    + ", lkmMemo=" + lkmMemo + "]";
        }
    }
```

接下来的问题就是：

如何通过配置的方式把客户实体的 **Set** 集合和联系人实体的中 **Customer** 对象与数据库建立起来关系，这就是我们今天学习的重点内容。

1.4 数据库表和实体类的多对多关系

1.4.1 示例分析

我们采用的示例为用户和角色。

用户：指的是咱们班的每一个同学。

角色：指的是咱们班同学的身份信息。

比如 A 同学，它是我的学生，其中有个身份就是学生，还是家里的孩子，那么他还有个身份是子女。

同时 B 同学，它也具有学生和子女的身份。

那么任何一个同学都可能具有多个身份。同时学生这个身份可以被多个同学所具有。

所以我们说，用户和角色之间的关系是多对多。

1.4.2 表关系建立

多对多的表关系建立靠的是中间表，其中用户表和中间表的关系是一对多，角色表和中间表的关系也是一对多，如下图所示：

用户表

sys_user		
ID (PK)	Namc	gcndcrr
1	aaa1	male
2	aaa2	male
3	aaa3	fenale
4	aaa4	male
5	aaa5	fenale
6	aaa6	male

一个用户可以具备多个角色，一个角色可以赋予多个用户。用户和角色之间的关系是多对多。多对多的实现方式是靠第三张表，也叫中间表。里面包含了两个字段，分别引用各自的主键。同时这两个字段还是联合主键。用户和角色任何一张表和中间表的关系都是一对多

角色表

sys_role		
ID (PK)	Namc	Mcmo
1	实习生	
2	工程师	
3	学员	
4	秘书	
5	助理	
6	项目经理	

用户角色关联关系表

userid (PK)	roleid
1	1
2	1
1	2
3	3
3	1
4	1
5	5

1.4.3 实体类关系建立

一个用户可以具有多个角色，所以在用户实体类中应该包含多个角色的信息，代码如下：

```
/**
 * 用户的数据模型
 */
```

```
public class SysUser implements Serializable {

    private Long userId;
    private String userCode;
    private String userName;
    private String userPassword;
    private String userState;

    //多对多关系映射
    private Set<SysRole> roles = new HashSet<SysRole>(0);

    public Long getUserId() {
        return userId;
    }
    public void setUserId(Long userId) {
        this.userId = userId;
    }
    public String getUserCode() {
        return userCode;
    }
    public void setUserCode(String userCode) {
        this.userCode = userCode;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getUserPassword() {
        return userPassword;
    }
    public void setUserPassword(String userPassword) {
        this.userPassword = userPassword;
    }
    public String getUserState() {
        return userState;
    }
    public void setUserState(String userState) {
        this.userState = userState;
    }
    public Set<SysRole> getRoles() {
        return roles;
    }
```

```
    public void setRoles(Set<SysRole> roles) {
        this.roles = roles;
    }
    @Override
    public String toString() {
        return "SysUser [userId=" + userId + ", userCode=" + userCode + ",
userName=" + userName + ", userPassword="
                + userPassword + ", userState=" + userState + "]";
    }
}
```

一个角色可以赋予多个用户，所以在角色实体类中应该包含多个用户的信息，代码如下：

```
/**
 * 角色的数据模型
 */
public class SysRole implements Serializable {

    private Long roleId;
    private String roleName;
    private String roleMemo;

    //多对多关系映射
    private Set<SysUser> users = new HashSet<SysUser>(0);

    public Long getRoleId() {
        return roleId;
    }
    public void setRoleId(Long roleId) {
        this.roleId = roleId;
    }
    public String getRoleName() {
        return roleName;
    }
    public void setRoleName(String roleName) {
        this.roleName = roleName;
    }
    public String getRoleMemo() {
        return roleMemo;
    }
    public void setRoleMemo(String roleMemo) {
        this.roleMemo = roleMemo;
    }
    public Set<SysUser> getUsers() {
        return users;
    }
```

```
        public void setUsers(Set<SysUser> users) {
            this.users = users;
        }
        @Override
        public String toString() {
            return "SysRole [roleId=" + roleId + ", roleName=" + roleName + ",
roleMemo=" + roleMemo + "]";
        }
    }
```

第2章 多表映射

2.1 一对多 XML 关系映射

2.1.1 客户配置文件：

```
    <?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.itheima.domain">
        <class name="Customer" table="cst_customer">
            <id name="custId" column="cust_id">
                <generator class="native"></generator>
            </id>
            <property name="custName" column="cust_name"></property>
            <property name="custLevel" column="cust_level"></property>
            <property name="custSource" column="cust_source"></property>
            <property name="custIndustry" column="cust_industry"></property>
            <property name="custAddress" column="cust_address"></property>
            <property name="custPhone" column="cust_phone"></property>
            <!-- 一对多关系映射
                涉及的标签
                    set：用于映射 set 集合属性
                        属性：
                            name：指定集合属性的名称
                            table:在一对多的时候写不写都可以。
                                    它指定的是集合元素所对应的表
                    one-to-many:用于指定当前映射配置文件所对应的实体和集合元素所对应的
实体是一对多关系。
```

```
                        属性：
                            class：指定集合元素所对应的实体类名称。
                    key：用于映射外键字段的。
                        属性：
                            column：指定从表中的外键字段名称
            -->
            <set name="linkmans" table="cst_linkman">
                <key column="lkm_cust_id"></key>
                <one-to-many class="LinkMan"/>
            </set>
        </class>
    </hibernate-mapping>
```

2.1.2 联系人配置文件：

```
    <?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.itheima.domain">
        <class name="LinkMan" table="cst_linkman">
            <id name="lkmId" column="lkm_id">
                <generator class="native"></generator>
            </id>
            <property name="lkmName" column="lkm_name"></property>
            <property name="lkmGender" column="lkm_gender"></property>
            <property name="lkmPhone" column="lkm_phone"></property>
            <property name="lkmMobile" column="lkm_mobile"></property>
            <property name="lkmEmail" column="lkm_email"></property>
            <property name="lkmPosition" column="lkm_position"></property>
            <property name="lkmMemo" column="lkm_memo"></property>
            <!-- 多对一关系映射
                涉及的标签：
                    many-to-one:用于建立多对一的关系映射配置
                        属性：
                            name：指定的实体类中属性的名称
                            class：该属性所对应的实体类名称。如果在 hibernate-mapping
上没有导包，则需要写全限定类名
                            column：指定从表中的外键字段名称
            -->
            <many-to-one name="customer" class="Customer" column="lkm_cust_id"
/>  </class>
```

```
    </hibernate-mapping>
```

2.2 多对多关系映射

2.2.1 用户配置文件：

```
    <?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.itheima.domain">
        <class name="SysUser" table="sys_user">
            <id name="userId" column="user_id">
                <generator class="native"></generator>
            </id>
            <property name="userCode" column="user_code"></property>
            <property name="userName" column="user_name"></property>
            <property name="userPassword" column="user_password"></property>
            <property name="userState" column="user_state"></property>
            <!-- 多对多关系映射
                涉及的标签：
                    set:用于映射集合属性
                        属性：
                          name：指定集合属性的名称
                          table:指定的是中间表的名称，在多对多的配置时，必须写。
                     key：指定外键字段
                        属性：
                           column：指定的是当前映射文件所对应的实体在中间表的外键字段名
称
                     many-to-many：指定当前映射文件所对应的实体和集合元素所对应的实体是
多对多的关系
                        属性：
                            class：指定集合元素所对应的实体类
                            column：指定的是集合元素所对应的实体在中间表的外键字段名称


            -->
            <set name="roles" table="user_role_rel">
                <key column="user_id"></key>
                <many-to-many class="SysRole" column="role_id"></many-to-many>
            </set>
        </class>
    </hibernate-mapping>
```

2.2.2 角色配置文件：

```
<?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.itheima.domain">
    <class name="SysRole" table="sys_role">
        <id name="roleId" column="role_id">
            <generator class="native"></generator>
        </id>
        <property name="roleName" column="role_name"></property>
        <property name="roleMemo" column="role_memo"></property>
        <!-- 多对多关系映射 -->
        <set name="users" table="user_role_rel">
            <key column="role_id"></key>
            <many-to-many class="SysUser" column="user_id"></many-to-many>
        </set>
    </class>
</hibernate-mapping>
```

第3章 多表增删改操作

3.1 一对多关系的操作

3.1.1 保存操作

保存原则：先保存主表，再保存从表。

```
/**
 * 保存操作
 * 需求：
 *  保存一个客户和一个联系人
 * 要求：
 *  创建一个客户对象和一个联系人对象
 *  建立客户和联系人之间关联关系（双向一对多的关联关系）
 *  先保存客户，再保存联系人
 * 问题：
 *  在使用 xml 配置的情况下：
```

```
                当我们建立了双向的关联关系之后，先保存主表，再保存从表时：
                会产生 2 条 insert 和 1 条 update.
         *      而实际开发中我们只需要 2 条 insert。
         *
         */
        @Test
        public void test1(){
            //创建客户和联系人对象
            Customer c = new Customer();//瞬时态
            c.setCustName("TBD 云集中心");
            c.setCustLevel("VIP 客户");
            c.setCustSource("网络");
            c.setCustIndustry("商业办公");
            c.setCustAddress("昌平区北七家镇");
            c.setCustPhone("010-84389340");

            LinkMan l = new LinkMan();//瞬时态
            l.setLkmName("TBD 联系人");
            l.setLkmGender("male");
            l.setLkmMobile("13811111111");
            l.setLkmPhone("010-34785348");
            l.setLkmEmail("98354834@qq.com");
            l.setLkmPosition("老师");
            l.setLkmMemo("还行吧");
            //建立他们的双向一对多关联关系
            //l.setCustomer(c);
            c.getLinkmans().add(l);

            Session s = HibernateUtil.getCurrentSession();
            Transaction tx = s.beginTransaction();
            //按照要求：先保存客户，再保存联系人(此时符合保存原则：先保存主表，再保存从表)
            s.save(c);//如果在把客户对象转成持久态时，不考虑联系人的信息。就不会有联系人
的快照产生
            s.save(l);

            tx.commit();//默认此时会执行快照机制，当发现一级缓存和快照不一致了，使用一级
缓存更新数据库。
        }
```

3.1.1.1 保存时遇到的问题

我们已经分析过了，因为双向维护了关系，而且持久态对象可以自动更新数据库，更新客户的时候会修改一次外键，更新联系人的时候同样也会修改一次外键。这样就会产生了多

余的 SQL，那么问题产生了，我们又该如何解决呢？

其实解决的办法很简单，只需要将一方放弃外键维护权即可。也就是说关系不是双方维护的，只需要交给某一方去维护就可以了。通常我们都是交给多的一方去维护的。为什么呢？因为多的一方才是维护关系的最好的地方，举个例子，一个老师对应多个学生，一个学生对应一个老师，这是典型的一对多。那么一个老师如果要记住所有学生的名字很难的，但如果让每个学生记住老师的名字应该不难。其实就是这个道理。

所以在一对多中，一的一方都会放弃外键的维护权（关系的维护）。这个时候如果想让一的一方放弃外键的维护权，只需要进行如下的配置即可。

```
<!-- 配置关联关系映射： -->
<!-- 配置一个set标签：代表一个集合 name属性：集合的属性名称 -->
<set name="linkMans" cascade="delete,save-update" inverse="true">
    <!-- 配置一个key标签 column属性：多的一方的外键的名称 -->
    <key column="lkm_cust_id"/>
    <!-- 配置one-to-many class对用多的一方的类的全路径-->
    <one-to-many class="cn.itcast.hibernate.demo1.LinkMan"/>
</set>
```

inverse 的默认值是 false，代表不放弃外键维护权，配置值为 true，代表放弃了外键的维护权。代码如下：

```
解决多一条 update 语句的问题：
<set name="linkmans" table="cst_linkman" inverse="true">
        <key column="lkm_cust_id"></key>
        <one-to-many class="LinkMan"/>
</set>
```

3.1.2 修改操作

```
/**
 * 更新操作
 * 需求：
 *  更新客户
 * 要求：
 *  创建一个新的联系人对象
 *  查询 id 为 1 的客户
 *  建立联系人和客户的双向一对多关联关系
 *  更新客户
 * 问题：
 *  当更新一个持久态对象时，它关联了一个瞬时态的对象。执行更新
 *      如果是注解配置：什么都不会做
 *      如果是 XML 配置：会报错
 * 解决办法：
 *  配置级联保存更新
 */
@Test
public void test2(){
```

```
        //创建联系人对象
        LinkMan l = new LinkMan();//瞬时态
        l.setLkmName("TBD 联系人 test");
        l.setLkmGender("male");
        l.setLkmMobile("13811111111");
        l.setLkmPhone("010-34785348");
        l.setLkmEmail("98354834@qq.com");
        l.setLkmPosition("老师");
        l.setLkmMemo("还行吧");


        Session s = HibernateUtil.getCurrentSession();
        Transaction tx = s.beginTransaction();
        //查询 id 为 1 的客户
        Customer c1 = s.get(Customer.class, 1L);
        //建立双向关联关系
        c1.getLinkmans().add(l);
        //l.setCustomer(c1);
        //更新客户
        s.update(c1);
        tx.commit();
    }
```

3.1.2.1 修改中遇到的问题：

什么是级联操作：

级联操作是指当主控方执行保存、更新或者删除操作时，其关联对象（被控方）也执行相同的操作。在映射文件中通过对 cascade 属性的设置来控制是否对关联对象采用级联操作，级联操作对各种关联关系都是有效的。

级联操作的方向性：

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在保存多的一方级联一的一方。

【保存客户级联联系人】

首先要确定我们要保存的主控方是那一方，我们要保存客户，所以客户是主控方，那么需要在客户的映射文件中进行如下的配置。

代码如下：

解决报错的问题，配置级联保存更新：

```
<set      name="linkmans"      table="cst_linkman"      cascade="save-update"
inverse="true">
        <key column="lkm_cust_id"></key>
        <one-to-many class="LinkMan"/>
    </set>
```

3.1.3 删除操作

```
        /**
         * 删除操作
         *   删除从表数据：可以随时任意删除。
         *   删除主表数据：
         *       有从表数据引用
         *           1、在默认情况下，它会把外键字段置为null，然后删除主表数据。
         *               如果在数据库的表结构上，外键字段有非空约束，默认情况就会报错了。
         *           2、如果配置了放弃维护关联关系的权利，则不能删除（与外键字段是否允许为null，没有关系）
         *               因为在删除时，它根本不会去更新从表的外键字段了。
         *           3、如果还想删除，使用级联删除
         *       没有从表数据引用：随便删
         *
         * 在实际开发中，级联删除请慎用！(在一对多的情况下)
         */
        @Test
        public void test3(){
            Session s = HibernateUtil.getCurrentSession();
            Transaction tx = s.beginTransaction();
            //查询 id 为 1 的客户
            Customer c1 = s.get(Customer.class, 2L);
            //删除 id 为 1 的客户
            s.delete(c1);
            tx.commit();
        }
```

3.1.3.1 删除中遇到的问题

我们之前学习过级联保存或更新，那么再来看级联删除也就不难理解了，级联删除也是有方向性的，删除客户同时级联删除联系人，也可以删除联系人同时级联删除客户（这种需求很少）。

原来 JDBC 中删除客户和联系人的时候，如果有外键的关系是不可以删除的，但是现在我们使用了 Hibernate，其实 Hibernate 可以实现这样的功能，但是不会删除客户同时删除联系人，默认的情况下如果客户下面还有联系人，Hibernate 会将联系人的外键置为 null，然后去删除客户。那么其实有的时候我们需要删除客户的时候，同时将客户关联的联系人一并删除。这个时候我们就需要使用 Hibernate 的级联删除操作了。

【删除客户的时候同时删除客户的联系人】

确定删除的主控方式客户，所以需要在客户端配置：

```
<!-- 配置关联关系映射: -->
<!-- 配置一个set标签:代表一个集合 name属性: 集合的属性名称 -->
<set name="linkMans" cascade="delete">
    <!-- 配置一个key标签 column属性:多的一方的外键的名称 -->
    <key column="lkm_cust_id"/>
    <!-- 配置one-to-many class对用多的一方的类的全路径-->
    <one-to-many class="cn.itcast.hibernate.demo1.LinkMan"/>
</set>
```

如果还想有之前的级联保存或更新，同时还想有级联删除，那么我们可以进行如下的配置：

```
<!-- 配置关联关系映射: -->
<!-- 配置一个set标签:代表一个集合 name属性: 集合的属性名称 -->
<set name="linkMans" cascade="delete,save-update">
    <!-- 配置一个key标签 column属性:多的一方的外键的名称 -->
    <key column="lkm_cust_id"/>
    <!-- 配置one-to-many class对用多的一方的类的全路径-->
    <one-to-many class="cn.itcast.hibernate.demo1.LinkMan"/>
</set>
```

代码如下：

```
级联删除的配置：
<set name="linkmans" table="cst_linkman" cascade="delete" inverse="true">
    <key column="lkm_cust_id"></key>
    <one-to-many class="LinkMan"/>
</set>
```

【删除联系人的时候同时删除客户.】

同样我们删除的是联系人，那么联系人是主控方，需要在联系人端配置：

```
<!-- 关联关系的映射的配置: 联系人关联客户 -->
<!-- 在多的一方配置many-to-one:代表多对一 -->
<!--
    many-to-one标签:代表多对一
        * name      :一的一方的对象的属性名称.
        * class     :一的一方的类的全路径
        * column    :在多的一方的外键的名称.
 -->
<many-to-one name="customer" cascade="delete" class="cn.itcast.hibernate.demo1.Customer" column="lkm_cust_id"/>
```

如果需要既做保存或更新有有级联删除的功能，也可以如下配置：

```
<!-- 关联关系的映射的配置: 联系人关联客户 -->
<!-- 在多的一方配置many-to-one:代表多对一 -->
<!--
    many-to-one标签:代表多对一
        * name      :一的一方的对象的属性名称.
        * class     :一的一方的类的全路径
        * column    :在多的一方的外键的名称.
 -->
<many-to-one name="customer" cascade="delete,save-update" class="cn.itcast.hibernate.demo1.Customer" column="lkm_cust_id"/>
```

3.2 多对多关系的操作

3.2.1 保存操作

```
/**
 * 需求：
 *   保存用户和角色
 * 要求：
 *   创建 2 个用户和 3 个角色
 *   让 1 号用户具有 1 号和 2 号角色(双向的)
 *   让 2 号用户具有 2 号和 3 号角色(双向的)
 *  保存用户和角色
 * 问题：
 *  在保存时，会出现主键重复的错误，因为都是要往中间表中保存数据造成的。
 * 解决办法：
 *   让任意一方放弃维护关联关系的权利
 */
@Test
public void test1(){
    //创建对象
    SysUser u1 = new SysUser();
    u1.setUserName("用户 1");
    SysUser u2 = new SysUser();
    u2.setUserName("用户 2");

    SysRole r1 = new SysRole();
    r1.setRoleName("角色 1");
    SysRole r2 = new SysRole();
    r2.setRoleName("角色 2");
    SysRole r3 = new SysRole();
    r3.setRoleName("角色 3");

    //建立关联关系
    u1.getRoles().add(r1);
    u1.getRoles().add(r2);
    r1.getUsers().add(u1);
    r2.getUsers().add(u1);

    u2.getRoles().add(r2);
    u2.getRoles().add(r3);
    r2.getUsers().add(u2);
    r3.getUsers().add(u2);
```

```
        Session s = HibernateUtil.getCurrentSession();
        Transaction tx = s.beginTransaction();
        s.save(u1);
        s.save(u2);
        s.save(r1);
        s.save(r2);
        s.save(r3);
        tx.commit();
    }
```

解决保存失败的问题：

```
<set name="users" table="user_role_rel" inverse="true">
    <key column="role_id"></key>
    <many-to-many class="SysUser" column="user_id"></many-to-many>
</set>
```

3.2.2 删除操作

```
    /**
     * 删除操作
     *   在多对多的删除时，双向级联删除根本不能配置
     * 禁用
     *   如果配了的话，如果数据之间有相互引用关系，可能会清空所有数据
     */
    @Test
    public void test2(){
        Session s = HibernateUtil.getCurrentSession();
        Transaction tx = s.beginTransaction();
        SysUser u1 = s.get(SysUser.class,3L);
        s.delete(u1);
        tx.commit();
    }
```

在映射配置中不能出现：双向级联删除的配置

第4章 hibernate 中的多表查询

4.1 对象导航查询

4.1.1 概述

对象图导航检索方式是根据已经加载的对象，导航到他的关联对象。它利用类与类之间的关系来检索对象。

例如：我们通过 OID 查询方式查出一个客户，可以调用 Customer 类中的 getLinkMans() 方法来获取该客户的所有联系人。

对象导航查询的使用要求是：两个对象之间必须存在关联关系。

4.1.2 对象导航检索示例

4.1.2.1 查询一个客户，获取该客户下的所有联系人

```
/**
 * 需求：
 *  查询 ID 为的 1 客户有多少联系人
 */
@Test
public void test1(){
    Session s = HibernateUtil.getCurrentSession();
    Transaction tx = s.beginTransaction();
    Customer c = s.get(Customer.class, 1L);
    Set<LinkMan> linkmans = c.getLinkmans();//此处就是对象导航查询
    for(Object o : linkmans){
        System.out.println(o);
    }
    tx.commit();
}
```

4.1.2.2 查询一个联系人，获取该联系人的所有客户

```
/**
 * 需求：
 *  查询 ID 为 1 的联系人所属客户
 */
@Test
```

```
public void test3(){
    Session s = HibernateUtil.getCurrentSession();
    Transaction tx = s.beginTransaction();
    LinkMan l = s.get(LinkMan.class, 1L);
    System.out.println(l.getCustomer());
    tx.commit();
}
```

4.1.3 对象导航查询的问题分析

问题 1：我们查询客户时，要不要把联系人查询出来？

分析：

如果我们不查的话，在用的时候还要自己写代码，调用方法去查询。

如果我们查出来的，不使用时又会白白的浪费了服务器内存。

解决：

采用延迟加载的思想。通过配置的方式来设定当我们在需要使用时，发起真正的查询。

配置的方式：

在 Customer.hbm.xml 配置文件中的 set 标签上使用 lazy 属性。取值为 true（默认值）|fasle

```
<set name="linkmans" table="cst_linkman" inverse="true" lazy="true">
    <key column="lkm_cust_id"></key>
    <one-to-many class="LinkMan"/>
</set>
```

问题 2：我们查询联系人时，要不要把客户查询出来？

分析：

如果我们不查的话，在用的时候还要自己写代码，调用方法去查询。

如果我们查出来的话，一个对象不会消耗太多的内存。而且多数情况下我们都是要使用的。

例如：查询联系人详情时，肯定会看看该联系人的所属客户。

解决：

采用立即加载的思想。通过配置的方式来设定，只要查询从表实体，就把主表实体对象同时查出来。

配置的方式：

在 LinkMan.hbm.xml 配置文件中的 many-to-one 标签上使用 lazy 属性。取值为 proxy|fasle

false：立即加载

proxy：看客户的映射文件 class 标签的 lazy 属性取值，如果客户的 class 标签 lazy 属性是 true 那么 proxy 表示延迟加载，如果是 false 就表示立即加载。

```
<many-to-one name="customer" class="Customer" column="lkm_cust_id" lazy="false"/>
```