**Hibernate5 第三天**

**【学习目标】**

通过本内容的学习，学生：

## 1，能够说出表与表之间的关系及建表原则

1. 能够说出表与表之间的关系。
2. 能够说出关系型数据库的建表原则。

## 2，能够配置hibernate一对多关系的映射

1. 独立创建客户表。
2. 独立创建联系人表及外键。
3. 独立创建客户的实体类。
4. 独立创建联系人的实体类。
5. 独立配置客户的映射文件。
6. 独立配置联系人的映射文件。

## 3，能够完成一对多映射关系的相应操作

1. 独立编写保存客户和联系人的代码。
2. 独立完成客户和联系人的级联保存操作。
3. 独立完成客户和联系人的级联删除操作。
4. 解释inverse的作用。

## 4，能够配置hibernate多对多关系的映射

1. 独立创建用户表。
2. 独立创建角色表。
3. 独立创建用户和角色的中间表。
4. 独立完成用户实体的创建。
5. 独立完成角色实体的创建。
6. 独立完成用户的映射文件的配置。
7. 独立完成角色的映射文件的配置。

## 5，能够完成多对多映射关系的相应操作

1. 独立编写保存用户和角色的代码。
2. 独立完成用户和角色的级联保存操作。
3. 独立完成用户和角色的级联删除操作。
4. 独立完成给某个用户分配新角色的操作。
5. 独立完成给某个用户删除某个角色的操作。
6. 独立完成给某个用户替换某个角色的操作。

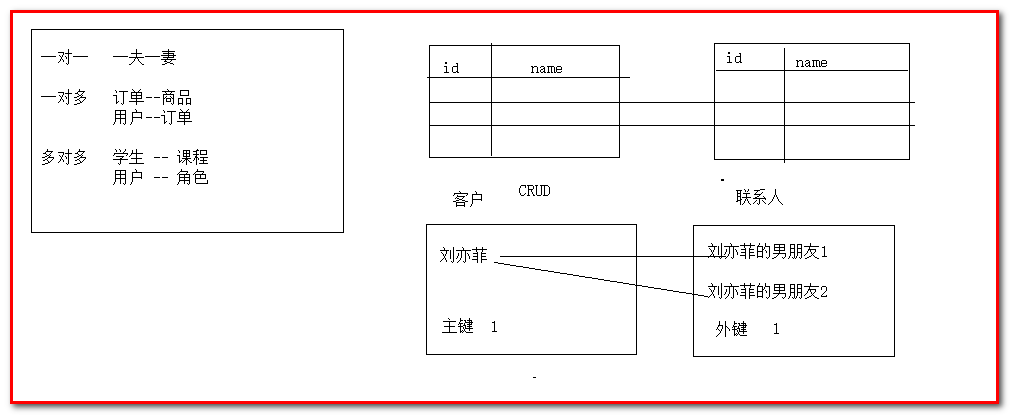
# 完成CRM的联系人的保存操作

## 需求分析

1. 因为客户和联系人是一对多的关系，在有客户的情况下，完成联系人的添加保存操作

## 技术分析之Hibernate的关联关系映射之一对多映射（重点）

### JavaWEB中一对多的设计及其建表原则



### 先导入SQL的建表语句

        \* 创建今天的数据库：create database hibernate\_day03;  
        \* 在资料中找到客户和联系人的SQL脚本

### 编写客户和联系人的JavaBean程序（注意一对多的编写规则）

      \* 客户的JavaBean如下

|  |
| --- |
| **package** com.igeek.demo1;  **import** java.util.HashSet;  **import** java.util.Set;  **public** **class** Customer {  **private** Long cust\_id ; //客户编号(主键)  **private** String cust\_name; // 客户名称(公司名称)  **private** String cust\_user\_id; // 负责人id  **private** String cust\_create\_id ;// 创建人id  **private** String cust\_source ; //客户信息来源  **private** String cust\_industry ;// 客户所属行业  **private** String cust\_level ; // 客户级别  **private** String cust\_linkman ; // 联系人  **private** String cust\_phone ; // 固定电话  **private** String cust\_mobile ; // 移动电话  //设置与多方关联的属性  **private** Set<Linkman> linkmans = **new** HashSet<>();    //省略getter和setter方法  } |

   \* 联系人的JavaBean如下

|  |
| --- |
| **package** com.igeek.demo1;//  **public** **class** Linkman {  **private** Long lkm\_id;//联系人编号(主键)  **private** String lkm\_name ;// 联系人姓名  //hibernate使用对象来做关联，不用设置外键id  //private String lkm\_cust\_id ;// 客户id  **private** String lkm\_gender ;// 联系人性别  **private** String lkm\_phone ;// 联系人办公电话  **private** String lkm\_mobile ;// 联系人手机  **private** String lkm\_email ;// 联系人邮箱  **private** String lkm\_qq;// 联系人qq  **private** String lkm\_position;// 联系人职位  **private** String lkm\_memo ;// 联系人备注  //与一方关联的属性  **private** Customer customer;    //省略getter和setter方法  } |

### 编写客户和联系人的映射配置文件（注意一对多的配置编写）

\* 客户的映射配置文件如下

|  |
| --- |
| <?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=*"com.igeek.demo1.Customer"* table=*"cst\_customer"*>  <id name=*"cust\_id"* column=*"cust\_id"*>  <generator class=*"native"*></generator>  </id>  <property name=*"cust\_name"* column=*"cust\_name"*></property>  <property name=*"cust\_user\_id"* column=*"cust\_user\_id"*></property>  <property name=*"cust\_source"* column=*"cust\_source"*></property>  <property name=*"cust\_industry"* column=*"cust\_industry"*></property>  <property name=*"cust\_level"* column=*"cust\_level"*></property>  <property name=*"cust\_linkman"* column=*"cust\_linkman"*></property>  <property name=*"cust\_phone"* column=*"cust\_phone"*></property>  <property name=*"cust\_mobile"* column=*"cust\_mobile"*></property>  <!--设置与多方的关系 -->  <!--  name:javabean中set集合的名称  key:column:外键名称  one-to-many class:set集合中类的全路径  -->  <set name=*"linkmans"*>  <key column=*"lkm\_cust\_id"*></key>  <one-to-many class=*"com.igeek.demo1.Linkman"*/>  </set>  </class>  </hibernate-mapping> |

\* 联系人的映射配置文件如下

|  |
| --- |
| <?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=*"com.igeek.demo1.Linkman"* table=*"cst\_linkman"*>  <id name=*"lkm\_id"* column=*"lkm\_id"*>  <generator class=*"native"*></generator>  </id>  <property name=*"lkm\_name"* column=*"lkm\_name"*></property>  <!--  外键字段使用many-to-one节点设置，这里不用写  <property name="lkm\_cust\_id" column="lkm\_cust\_id"></property> -->  <property name=*"lkm\_gender"* column=*"lkm\_gender"*></property>  <property name=*"lkm\_phone"* column=*"lkm\_phone"*></property>  <property name=*"lkm\_mobile"* column=*"lkm\_mobile"*></property>  <property name=*"lkm\_email"* column=*"lkm\_email"*></property>  <property name=*"lkm\_qq"* column=*"lkm\_qq"*></property>  <property name=*"lkm\_position"* column=*"lkm\_position"*></property>  <property name=*"lkm\_memo"* column=*"lkm\_memo"*></property>  <!-- 设置多对一关系 -->  <!-- name：javabean的属性名称  class:属性的类的全路径  colum:外键名称 -->  <many-to-one name=*"customer"* class=*"com.igeek.demo1.Customer"* column=*"lkm\_cust\_id"*></many-to-one>  </class>  </hibernate-mapping> |

### 配置文件和具体的说明

        \* 一方  
            <!-- JavaBean对象中有Set集合，必须要配置Set集合 ，配置了  
                    Set        name属性：Set集合的名称  
                    key        column：外键的名称  
                    one-to-many  class：Set集合中类的全路径  
            -->  
            <set name="linkmans">  
                <key column="lkm\_cust\_id"/>  
                <one-to-many class="com.igeek.demo1.Linkman"/>  
            </set>  
          
        \* 多方  
            <!-- 配置的是多方 ，使用的是many-to-one标签  
                name        JavaBean的属性的名称  
                class        属性的类的全路径  
                column        外键的名称  
            -->  
    <many-to-one name="customer" class="com.igeek.demo1.Customer" column="lkm\_cust\_id"/>

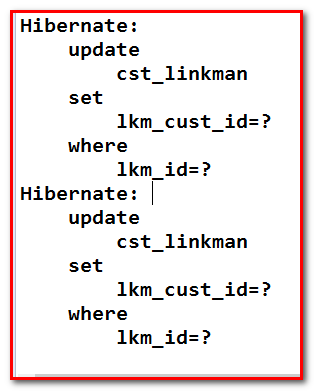
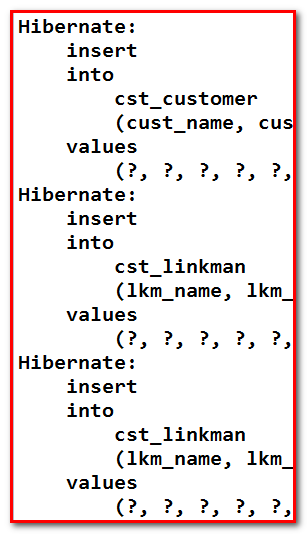
### hibernate.cfg.xml文件配置信息

|  |
| --- |
| <?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=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"hibernate.connection.url"*>jdbc:mysql:///hibernate\_day03</property>  <property name=*"hibernate.connection.username"*>root</property>  <property name=*"hibernate.connection.password"*>123456</property>  <property name=*"hibernate.dialect org.hibernate.dialect.MySQLDialect"*></property>  <!-- 打印sql语句的配置 -->  <property name=*"hibernate.show\_sql"*>true</property>  <property name=*"hibernate.format\_sql"*>true</property>  <property name=*"hibernate.hbm2ddl.auto"*>update</property>    <!-- 把session放在线程中 -->  <property name=*"hibernate.current\_session\_context\_class"*>thread</property>    <!-- 关联xml配置文件 -->  <mapping resource=*"com/igeek/demo1/Customer.hbm.xml"*/>  <mapping resource=*"com/igeek/demo1/Linkman.hbm.xml"*/>      </session-factory>  </hibernate-configuration> |

### 进行双向关联进行数据的保存

|  |
| --- |
| **package** com.igeek.demo1;  **import** org.hibernate.Session;  **import** org.hibernate.Transaction;  **import** org.junit.Test;  **import** com.igeek.util.HibernateUtils;  **public** **class** TestOne2Many {  //双向关联  @Test  **public** **void** run1(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();    //新建一个客户  Customer c = **new** Customer();  c.setCust\_name("小凤");    //新建2个联系人  Linkman m1 = **new** Linkman();  m1.setLkm\_name("小凤的男朋友1");    Linkman m2 = **new** Linkman();  m2.setLkm\_name("小凤的男朋友2");    //双向关联  c.getLinkmans().add(m1);  c.getLinkmans().add(m2);    m1.setCustomer(c);  m2.setCustomer(c);    //开始保存  session.save(c);  session.save(m1);  session.save(m2);    tr.commit();  }  } |

### 测试



这种保存要求:必须双方都建立关系,而且都要执行保存操作.

## 技术分析之级联保存

1. 测试：如果现在代码只插入其中的一方的数据  
       \* 如果只保存其中的一方的数据，那么程序会抛出异常。  
       \* 如果想完成只保存一方的数据，并且把相关联的数据都保存到数据库中，那么需要配置级联！！  
             
       \* 级联保存是方向性  
         
     2. 级联保存效果  
       \* 级联保存：保存一方同时可以把关联的对象也保存到数据库中！！  
       \* 使用cascade="save-update"

### 级联保存示例

1. 保存客户数据，在TestOne2Many中添加方法测试

|  |
| --- |
| //代码只插入其中一方的数据  @Test  **public** **void** run2(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();    //新建一个客户  Customer c = **new** Customer();  c.setCust\_name("小凤");    //新建2个联系人  Linkman m1 = **new** Linkman();  m1.setLkm\_name("小凤的男朋友1");    Linkman m2 = **new** Linkman();  m2.setLkm\_name("小凤的男朋友2");    //客户中添加联系人  c.getLinkmans().add(m1);  c.getLinkmans().add(m2);    //开始保存  session.save(c);    tr.commit();  } |

会报错：



原因结论： 在hibernate代码中，在session.flush前，不允许 持久态对象 关联 瞬时态对象

持久态对象只能关联持久态对象！

解决：**采用级联 ，cascade ：cascade="save-update"**

它的作用：

**可以使持久态对象”关联”瞬时态对象， 自动会隐式执行save操作，变为持久态，**

**可以使持久态对象”关联”脱管对象，自动会隐式执行update操作，变为持久态**

如果通过操作customer来级联保存 linkman，需要在Customer.hbm.xml(谁是持久的) 配置级联.

2.修改Customer.hbm.xml:

|  |
| --- |
| <!-- 配置级联保存:cascade:save-update -->  <set name=*"linkmans"* cascade=*"save-update"*> |

1. 测试

使用级联之后,Hibernate会对瞬时态的这个对象，会自动执行save操作.

问题:如果保存顺序反过来呢？即先保存联系人，同时保存客户呢？  
分析:先操作linkman，级联保存 customer ，需要在 Linkman.hbm.xml 配置级联

|  |
| --- |
| <many-to-one cascade=*"save-update"* name=*"Customer"* class=*"com.igeek.demo1.Customer"* column=*"lkm\_cust\_id"*></many-to-one> |

在TestOne2Many中添加方法测试

|  |
| --- |
| //级联保存  @Test  **public** **void** run3(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();    //新建一个客户  Customer c = **new** Customer();  c.setCust\_name("小凤");    //新建2个联系人  Linkman m1 = **new** Linkman();  m1.setLkm\_name("小凤的男朋友1");    Linkman m2 = **new** Linkman();  m2.setLkm\_name("小凤的男朋友2");    //联系人中添加客户  m1.setCustomer(c);  m2.setCustomer(c);    //开始保存  session.save(m1);  session.save(m2);    tr.commit();  } |

级联对于大量的保存或更新操作非常有用。

## 技术分析之级联删除

1.先来给大家在数据库中演示含有外键的删除客户功能，那么SQL语句是会报出错误的  
    \* 例如：delete from cst\_customer where cust\_id = 1;  
      
  2. 如果使用Hibernate框架直接删除客户的时候，测试发现是可以删除的  
      
  3. 上述的删除是普通的删除，那么也可以使用级联删除，注意：级联删除也是有方向性的！！  
    \* <many-to-one cascade="delete" />

### 级联删除示例1

1.修改Custom.hbm.xml

|  |
| --- |
| <!-- 配置级联删除:cascade:delete -->  <set name=*"linkmans"* cascade=*"delete"*> |

2.在TestOne2Many类中添加测试方法

|  |
| --- |
| //级联删除--删除客户  @Test  **public** **void** run4(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();    //获取一个客户  Customer c = session.get(Customer.**class**, 1L);    //删除客户  session.delete(c);    tr.commit();  } |

1. 测试发现hibernate发了5条sql语句，

A.找出id对应的客户

B.找出id对应的联系人

C.把找出的联系人对应的客户id设置为null

D.删除联系人

E.删除客户

### 级联删除示例2

1. 修改Linkman.hbm.xml和Customer.hbm.xml

Linkman.hbm.xml

|  |
| --- |
| <many-to-one cascade=*"delete"* name=*"Customer"* class=*"com.igeek.demo1.Customer"* column=*"lkm\_cust\_id"*></many-to-one> |

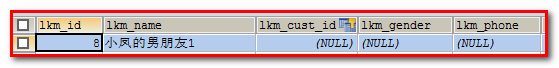
Customer.hbm.xml

|  |
| --- |
| <set name=*"linkmans"* >  <key column=*"lkm\_cust\_id"*></key>  <one-to-many class=*"com.igeek.demo1.Linkman"*/>  </set> |

1. 在TestOne2Many类中添加测试方法

|  |
| --- |
| //级联删除--删除联系人  @Test  **public** **void** run5(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();    //获取一个联系人  Linkman m = session.get(Linkman.**class**, 7L);    //删除联系人  session.delete(m);    tr.commit();  } |

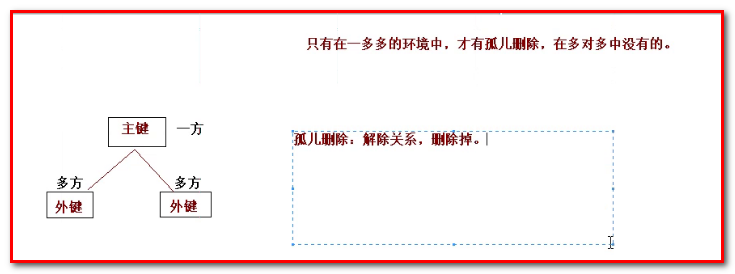
1. 测试发现剩下一条联系人数据没有关联到客户，这条数据已无用，说明这种方式不合理。



## 技术分析之级联的取值（cascade的取值）和孤儿删除（在一对多的环境下才有）

需要大家掌握的取值如下  
    \* none              -- 不使用级联  
    \* save-update          -- 级联保存或更新  
    \* delete             -- 级联删除  
    \* delete-orphan         -- 孤儿删除.(注意：只能应用在一对多关系)  
    \* all              -- 除了delete-orphan的所有情况.（包含save-update delete）  
    \* all-delete-orphan       -- 包含了delete-orphan的所有情况.（包含save-update delete delete-orphan）  
      
  2. 孤儿删除（孤子删除），只有在一对多的环境下才有孤儿删除  
    \* 在一对多的关系中,可以将一的一方认为是父方.将多的一方认为是子方.孤儿删除:在解除了父子关系的时候.将子方记录就直接删除。  
    \* <set name="linkmans" cascade="delete-orphan">

### 孤儿删除示例



1.修改Linkman.hbm.xml和Customer.hbm.xml

Linkman.hbm.xml

|  |
| --- |
| <many-to-one name=*"Customer"* class=*"com.igeek.demo1.Customer"* column=*"lkm\_cust\_id"*></many-to-one> |

Customer.hbm.xml

|  |
| --- |
| <set name=*"linkmans"* cascade=*"delete-orphan"*>  <key column=*"lkm\_cust\_id"*></key>  <one-to-many class=*"com.igeek.demo1.Linkman"*/>  </set> |

2.在TestOne2Many类中添加测试方法

|  |
| --- |
| //孤儿删除  @Test  **public** **void** run6(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();  //获取一个客户,这个客户下有2个联系人，ID 为11，和12  Customer c = session.get(Customer.**class**, 8L);    //获取一个联系人  Linkman m = session.get(Linkman.**class**, 11L);    //联系与客户解除关系  c.getLinkmans().remove(m);    tr.commit();  } |

1. 测试

Hibernate级联开发配置， cascade常用的取值：

* save-update：对关联瞬时对象执行save操作，对关联托管对象执行update
* delete：对关联对象进行删除

总结：

一般在业务开发中，不要两端都配置级联，（多方尽量不要配置级联,尽量在一方配置级联）。

## 技术分析之让某一方放弃外键的维护，为多对多做准备

1. 先测试双方都维护外键的时候，会产生多余的SQL语句。  
       \* 想修改客户和联系人的关系，进行双向关联，双方都会维护外键，会产生多余的SQL语句。  
       \* 产生的原因：session的一级缓存中的快照机制，会让双方都更新数据库，产生了多余的SQL语句。  
         
     2. 如果不想产生多余的SQL语句，那么需要一方来放弃外键的维护！  
       \* 在<set>标签上配置一个inverse=”true”.true:放弃.false:不放弃.默认值是false  
       \* <inverse="true">

### inverse示例

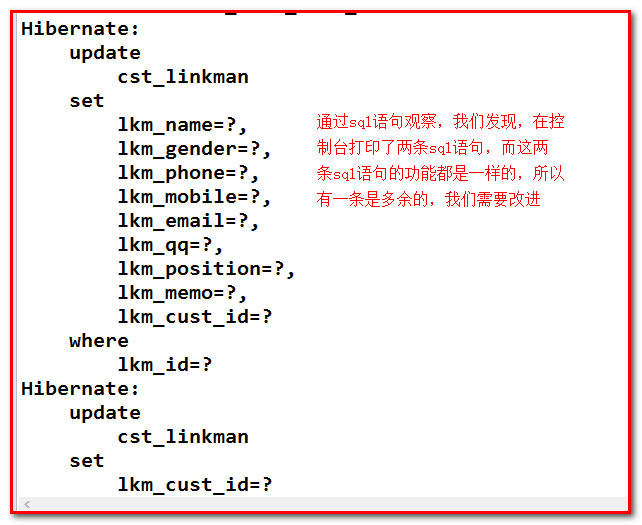
问题:多余sql的问题

将没有关系的一个客户和联系人建立关系。(双方)

1.在TestOne2Many类中添加测试代码

|  |
| --- |
| //inverse放弃外键维护  @Test  **public** **void** run7(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();  //获取一个客户  Customer c = session.get(Customer.**class**, 7L);  //获取一个联系人  Linkman m = session.get(Linkman.**class**, 8L);  //联系人与客户做双向关联  c.getLinkmans().add(m);  m.setCustomer(c);    tr.commit();  } |

产生的sql:多余sql



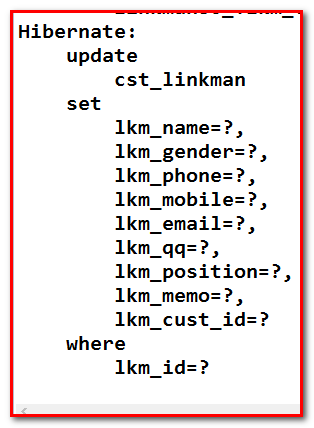
解决方法：

采用inverse属性来配置。

1. 修改Customer.hbm.xml

|  |
| --- |
| <set name=*"linkmans"* inverse=*"true"* >  <key column=*"lkm\_cust\_id"*></key>  <one-to-many class=*"com.igeek.demo1.Linkman"*/>  </set> |

1. 修改中数据，再次测试，hibernate只发一条sql语句



一般，我们都让1方放弃外键维护权！

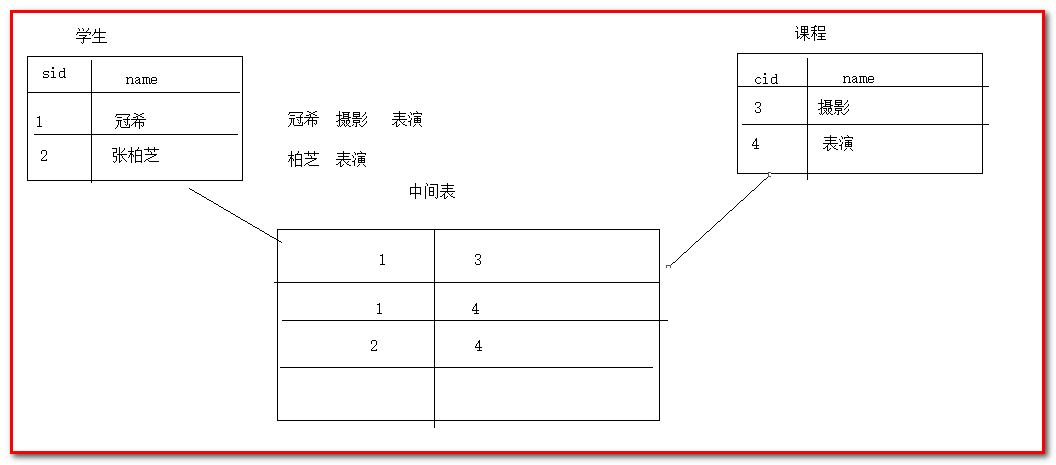
## 技术分析之cascade和inverse的区别

1. cascade用来级联操作（保存、修改和删除） 在一方设置  
     2. inverse用来维护外键的 在一方设置

# Hibernate的关联关系映射之多对多映射

## 技术分析之多对多的建表原则

1. JavaWEB的多对多



## 技术分析之多对多JavaBean的编写

<set name="roles" table="sys\_user\_role">

<!-- "user\_id":user表在中间表的外键名称 -->

<key column="user\_id"></key>

<!-- "role\_id":role表在中间表的外键名称 -->

<many-to-many class="com.igeek.demo2.Role" column="role\_id"></many-to-many>

</set>

<!--

name:角色类中关联用户类的属性

table:中间表

key:角色表在中间表的外键名称

class:角色类关联的用户类全路径

column:用户表在中间表的外键名称

-->

<set name="users" table="sys\_user\_role">

<key column="role\_id"></key>

<many-to-many class="com.igeek.demo2.User" column="user\_id"></many-to-many>

</set>

\* 用户的JavaBean代码如下

|  |
| --- |
| **package** com.igeek.demo2;  **import** java.util.HashSet;  **import** java.util.Set;  **public** **class** User {  **private** Long user\_id; //用户id  **private** String user\_code; //用户账号  **private** String user\_name; //用户名称  **private** String user\_password; //用户密码  **private** String user\_state; //1:正常,0:暂停  //使用set集合包含多个role  **private** Set<Role> roles = **new** HashSet<>();  //省略getter和setter方法  } |

\* 角色的JavaBean代码如下

|  |
| --- |
| **package** com.igeek.demo2;  **import** java.util.HashSet;  **import** java.util.Set;  **public** **class** Role {    **private** Long role\_id;//角色ID  **private** String role\_name;//角色名称  **private** String role\_memo;//备注  //包含多个user  **private** Set<User> users = **new** HashSet<>();  //省略getter和setter方法  } |

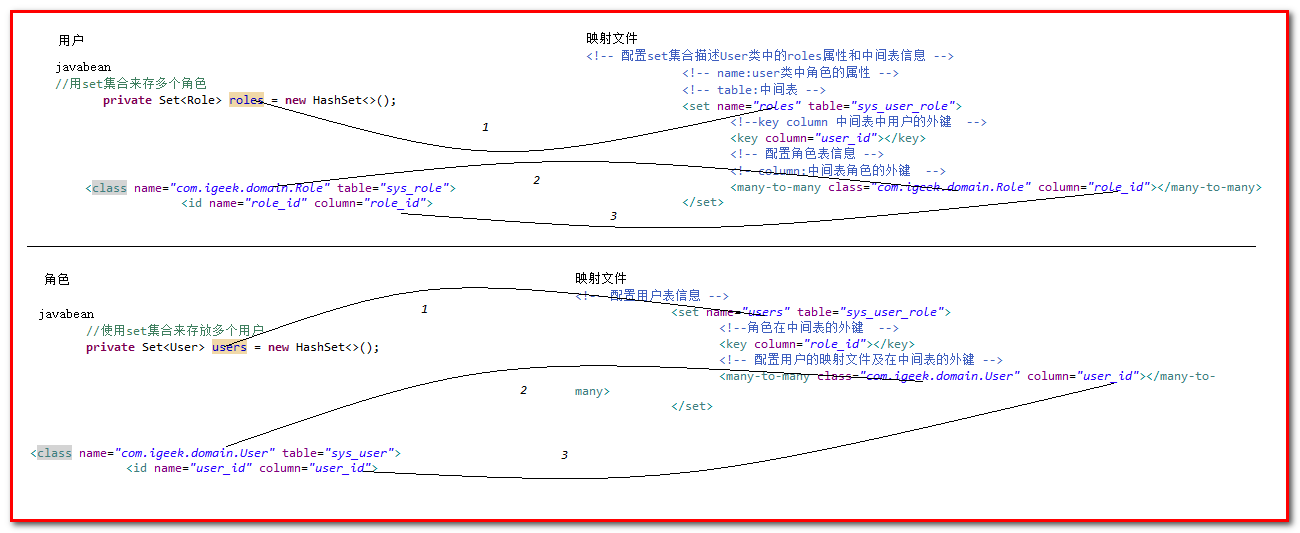
## 用户和角色的映射配置文件如下

\* 用户的映射配置文件如下

|  |
| --- |
| <?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=*"com.igeek.demo2.User"* table=*"sys\_user"*>  <id name=*"user\_id"* column=*"user\_id"*>  <generator class=*"native"*></generator>  </id>    <property name=*"user\_code"* column=*"user\_code"*></property>  <property name=*"user\_name"* column=*"user\_name"*></property>  <property name=*"user\_password"* column=*"user\_password"*></property>  <property name=*"user\_state"* column=*"user\_state"*></property>  <!-- 配置多对多 -->  <!-- name:User类的roles属性  table:中间表名称 -->  <set name=*"roles"* table=*"sys\_user\_role"*>  <!-- "user\_id":user表在中间表的外键名称 -->  <key column=*"user\_id"*></key>  <many-to-many class=*"com.igeek.demo2.Role"* column=*"role\_id"*></many-to-many>  </set>  </class>  </hibernate-mapping> |

\* 角色的映射配置文件如下

|  |
| --- |
| <?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=*"com.igeek.demo2.Role"* table=*"sys\_role"*>  <id name=*"role\_id"* column=*"role\_id"*>  <generator class=*"native"*></generator>  </id>  <property name=*"role\_name"* column=*"role\_name"*></property>  <property name=*"role\_memo"* column=*"role\_memo"*></property>  <!-- 配置多对多 -->  <!--  name:角色类中关联用户类的属性  table:中间表  key:角色表在中间表的外键名称  class:角色类关联的用户类全路径  column:用户表在中间表的外键名称  -->  <set name=*"users"* table=*"sys\_user\_role"*>  <key column=*"role\_id"*></key>  <many-to-many class=*"com.igeek.demo2.User"* column=*"user\_id"*></many-to-many>  </set>  </class>  </hibernate-mapping> |

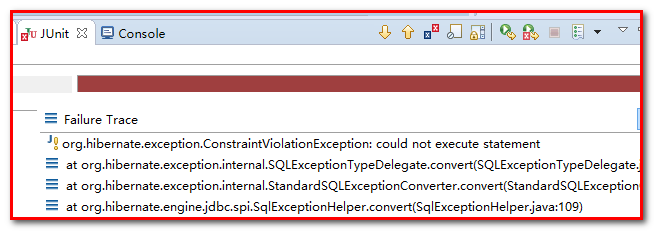
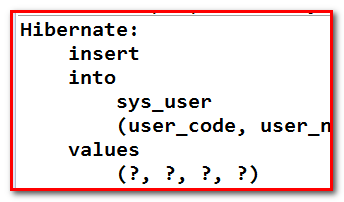
## hibernate配置文件引入映射配置

|  |
| --- |
| <mapping resource=*"com/igeek/demo2/User.hbm.xml"*/>  <mapping resource=*"com/igeek/demo2/Role.hbm.xml"*/> |

## 技术分析之多对多的级联保存--双向

|  |
| --- |
| **package** com.igeek.demo2;  **import** org.hibernate.Session;  **import** org.hibernate.Transaction;  **import** org.junit.Test;  **import** com.igeek.util.HibernateUtils;  **public** **class** TestMany2Many {    @Test  **public** **void** run1(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();    //1.新建2个用户  User u1 = **new** User();  u1.setUser\_name("小凤");    User u2 = **new** User();  u2.setUser\_name("冠希");    //2.新建2个角色  Role r1 = **new** Role();  r1.setRole\_name("演员");    Role r2 = **new** Role();  r2.setRole\_name("歌手");    //3.双向关联  //小凤是演员和歌手  u1.getRoles().add(r1);  u1.getRoles().add(r2);  r1.getUsers().add(u1);  r2.getUsers().add(u1);    //冠希是演员  u2.getRoles().add(r1);  r1.getUsers().add(u2);    //保存  session.save(u1);  session.save(u2);  session.save(r1);  session.save(r2);    tr.commit();  }  } |

测试



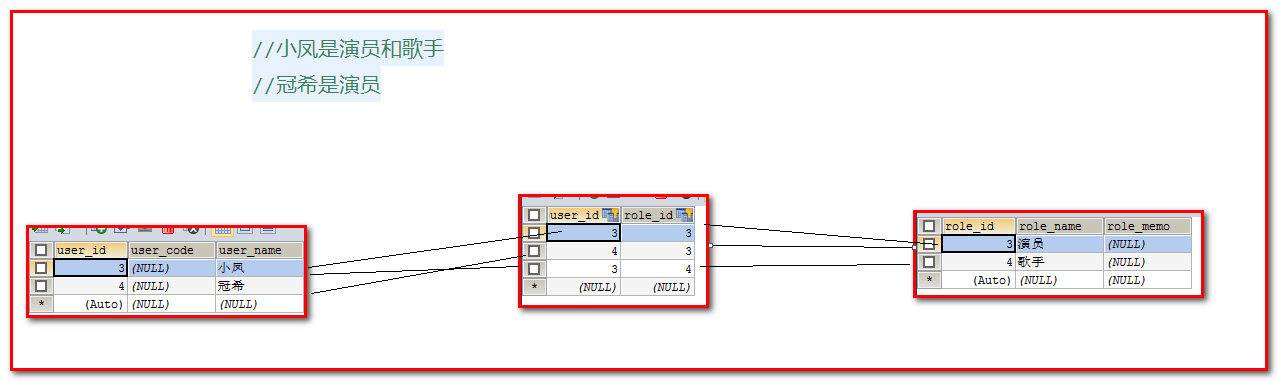
|  |
| --- |
| Caused by: com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException: Duplicate entry '1-1' for key 'PRIMARY' |

由于双方都在维护外键，导致更新中间表失败，解决办法，任选一方配置inverse=true，放弃外键维护

例如：修改Role.hbm.xml

|  |
| --- |
| <set name=*"users"* table=*"sys\_user\_role"* inverse=*"true"*> |

再次测试，可以往数据表中插入数据了。



总结：多对多进行双向关联的时候:必须有一方去放弃外键维护权

## 技术分析之多对多的级联保存--单向

级联保存  
   \* <set cascade="save-update">

1.在TestMany2Many类中添加测试方法，代码如下：

|  |
| --- |
| @Test  **public** **void** run2(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();    //1.新建2个用户  User u1 = **new** User();  u1.setUser\_name("小凤");    User u2 = **new** User();  u2.setUser\_name("冠希");    //2.新建2个角色  Role r1 = **new** Role();  r1.setRole\_name("演员");    Role r2 = **new** Role();  r2.setRole\_name("歌手");    //3.单向关联--用户关联角色  //小凤是演员和歌手  u1.getRoles().add(r1);  u1.getRoles().add(r2);    //冠希是演员  u2.getRoles().add(r1);    //保存 用户,hibernate使用快照机制保存角色  session.save(u1);  session.save(u2);    tr.commit();  } |

2.修改User.hbm.xml文件,在set节点上增加cascade=*"save-update"*,由用户级联保存角色

|  |
| --- |
| <set name=*"roles"* table=*"sys\_user\_role"* cascade=*"save-update"*> |

3.测试

## 级联删除（在多对多中是很少使用的）

1.在TestMany2Many类中添加测试方法，代码如下：

|  |
| --- |
| //小凤是演员和歌手，现在要删除小凤歌手的角色  @Test  **public** **void** run3(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();  //1.获取小凤用户  User u = session.get(User.**class**, 1L);  //2.获取 歌手角色  Role r = session.get(Role.**class**, 2L);  //3.删除小凤歌手的角色（注意在集合中操作，hibernate使用快照机制同步数据）  u.getRoles().remove(r);  tr.commit();  } |

注意：千万不要在映射文件中配置cascade=*"delete"* ,否则在删除一个用户时会把关联的角色删除，导致其他用户无法关联到角色

【示例】

1.修改User.hbm.xml

|  |
| --- |
| <set name=*"roles"* table=*"sys\_user\_role"* cascade=*"delete"*> |

2.修改Role.hbm.xml

|  |
| --- |
| <set name=*"users"* table=*"sys\_user\_role"* inverse=*"true"* cascade=*"delete"*> |

1. 在TestMany2Many类中添加测试方法，代码如下：

|  |
| --- |
| @Test  **public** **void** run4(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();  //1.获取小凤  User user = session.get(User.**class**, 11L);  session.delete(user);  tr.commit();  } |

4.测试发现用户表和角色表中的数据都没有了，原因是设置了级联删除。

作业：功能--保存联系人

1. 先跳转到添加联系人的页面  
       \* 先查询所有的客户，使用下来选择框显示客户  
         
     2. 保存联系人  
       \* 联系人需要客户的信息，先查询该客户的信息，把客户设置到联系人中  
       \* 保存联系人功能