案例一：完成一对多的关联关系映射并操作

1.1 案例需求

1.1.1需求描述

一个客户对应多个联系人，单独在联系人管理模块中对联系人信息进行维护，功能包括： 添加联系人、修改联系人、删除联系人。

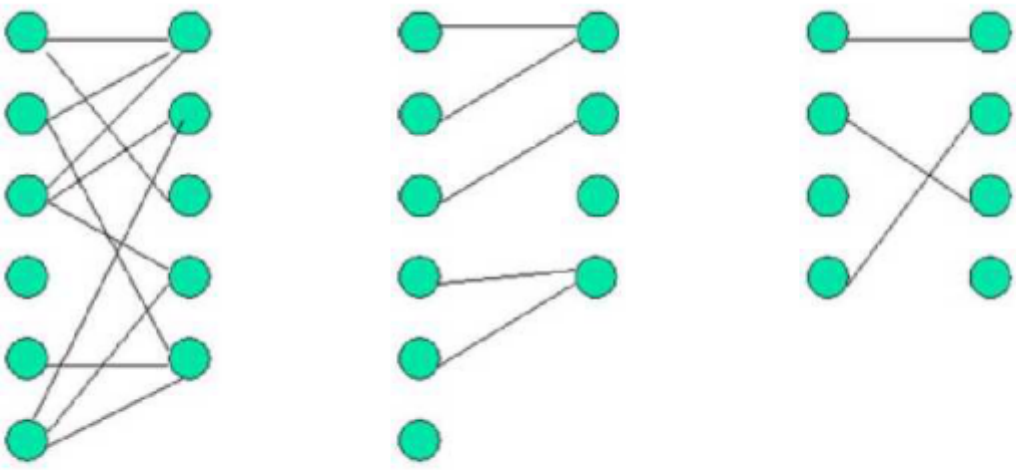
- 添加联系人：添加联系人时指定所属客户，添加信息包括联系人名称、联系电话等
- 修改联系人：允许修改联系人所属客户、联系人名称、联系人电话等信息
- 删除联系人：删除客户的同时删除下属的联系人，可以单独删除客户的某个联系人

1.2相关知识点

1.2.1 表关系的分析

Hibernate框架实现了 0RM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

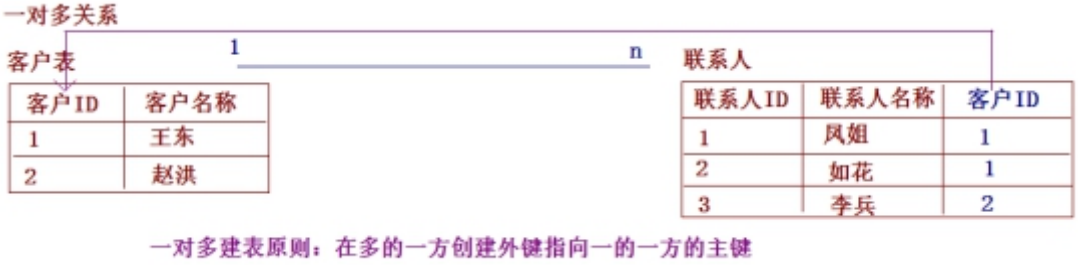
一对一
1: 1

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。在数据库中, 实体表之间的关系映射是采用外键来描述的，具体如下。

1.2.1.1 表与表的三种关系

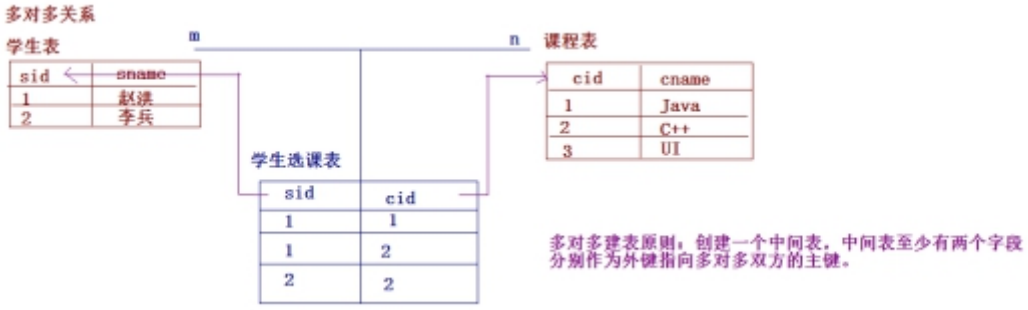
一对多关系

- 什么样关系属于一对多?
  - 一个部门对应多个员工，一个员工只能属于某一个部门。
  - 一个客户对应多个联系人，一个联系人只能属于某一个客户。
- 一对多的建表原则:



多对多关系

- 什么样关系属于多对多?
  - 一个学生可以选择多门课程，一门课程也可以被多个学生选择。
  - 一个用户可以选择多个角色，一个角色也可以被多个用户选择。
- 多对多的建表原则:



一对一关系(了解)

- 什么样关系属于一对一?
  - 一个公司只能有一个注册地址，一个注册地址只能被一个公司注册。
- 一对一的建表原则:

1.2.2 Hibernate的一对多关联映射

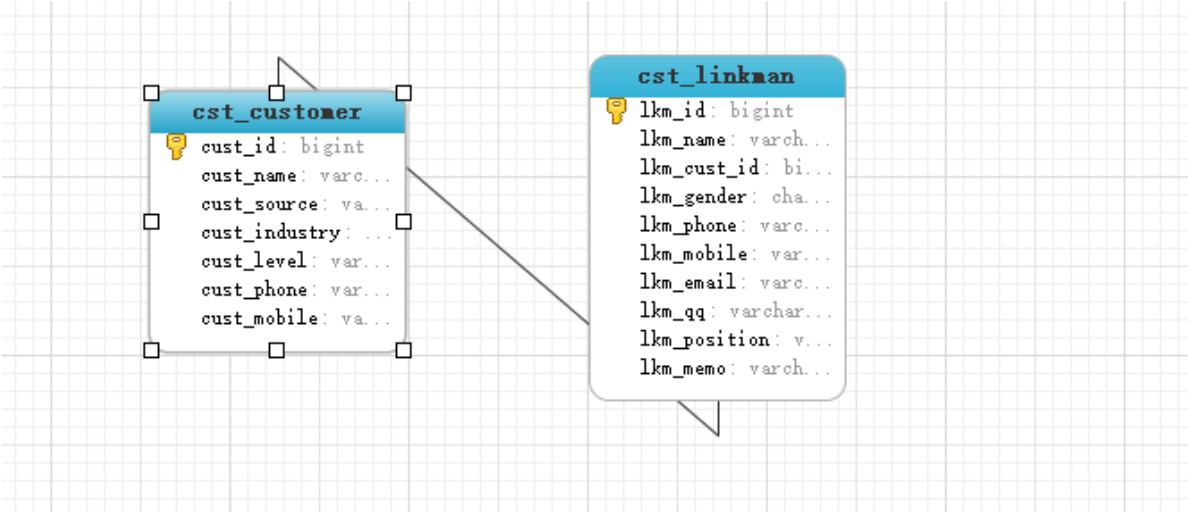
1.2.2.1 创建表

cst_customer表

```
CREATE TABLE `cst_customer` (
  `cust_id` bigint(32) NOT NULL AUTO_INCREMENT COMMENT '客户编号(主键)',
  `cust_name` varchar(32) NOT NULL COMMENT '客户名称(公司名称)',
  `cust_source` varchar(32) DEFAULT NULL COMMENT '客户信息来源',
  `cust_industry` varchar(32) DEFAULT NULL COMMENT '客户所属行业',
  `cust_level` varchar(32) DEFAULT NULL COMMENT '客户级别',
  `cust_phone` varchar(64) DEFAULT NULL COMMENT '固定电话',
  `cust_mobile` varchar(16) DEFAULT NULL COMMENT '移动电话',
  PRIMARY KEY (`cust_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

cst_linkman 表

```
CREATE TABLE `cst_linkman` (
  `lkm_id` bigint(32) NOT NULL AUTO_INCREMENT COMMENT '联系人编号(主键)',
  `lkm_name` varchar(16) DEFAULT NULL COMMENT '联系人姓名',
  `lkm_cust_id` bigint(32) DEFAULT NULL COMMENT '客户id',
  `lkm_gender` char(1) DEFAULT NULL COMMENT '联系人性别',
  `lkm_phone` varchar(16) DEFAULT NULL COMMENT '联系人办公电话',
  `lkm_mobile` varchar(16) DEFAULT NULL COMMENT '联系人手机',
  `lkm_email` varchar(64) DEFAULT NULL COMMENT '联系人邮箱',
  `lkm_qq` varchar(16) DEFAULT NULL COMMENT '联系人qq',
  `lkm_position` varchar(16) DEFAULT NULL COMMENT '联系人职位',
  `lkm_memo` varchar(512) DEFAULT NULL COMMENT '联系人备注',
  PRIMARY KEY (`lkm_id`),
  KEY `FK_cst_linkman_lkm_cust_id` (`lkm_cust_id`),
  CONSTRAINT `FK_cst_linkman_lkm_cust_id` FOREIGN KEY (`lkm_cust_id`) REFERENCES
`cst_customer` (`cust_id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```



1.2.2.2 创建实体

- 一的一方的实体

```
/**
* @Title: Customer.java
* @Package com.admiral.domain
```

```java
* @Description: 客户的实体
* @author 白世鑫
* @date 2020-9-22
* @version V1.0
*/
package com.admiral.domain;

import java.util.HashSet;
import java.util.Set;

public class Customer {

    private Long cust_id;
    private String cust_name;
    private String cust_source;
    private String cust_industry;
    private String cust_level;
    private String cust_phone;
    private String cust_mobile;

    // 通过 ORM 方式表示:一个客户对应多个联系人
    // 放置多的一方的集合.Hibernate 默认使用的是 Set 集合
    private Set<LinkMan> linkMans = new HashSet<LinkMan>();

    public Customer() {
        super();
    }

    public Customer(Long cust_id, String cust_name, String cust_source, String
cust_industry, String cust_level,
            String cust_phone, String cust_mobile) {
        super();
        this.cust_id = cust_id;
        this.cust_name = cust_name;
        this.cust_source = cust_source;
        this.cust_industry = cust_industry;
        this.cust_level = cust_level;
        this.cust_phone = cust_phone;
        this.cust_mobile = cust_mobile;
    }

    public Set<LinkMan> getLinkMans() {
        return linkMans;
    }

    public void setLinkMans(Set<LinkMan> linkMans) {
        this.linkMans = linkMans;
    }

    public Long getCust_id() {
        return cust_id;
    }

    public void setCust_id(Long cust_id) {
        this.cust_id = cust_id;
    }

    public String getCust_name() {
```

```
        return cust_name;
    }

    public void setCust_name(String cust_name) {
        this.cust_name = cust_name;
    }

    public String getCust_source() {
        return cust_source;
    }

    public void setCust_source(String cust_source) {
        this.cust_source = cust_source;
    }

    public String getCust_industry() {
        return cust_industry;
    }

    public void setCust_industry(String cust_industry) {
        this.cust_industry = cust_industry;
    }

    public String getCust_level() {
        return cust_level;
    }

    public void setCust_level(String cust_level) {
        this.cust_level = cust_level;
    }

    public String getCust_phone() {
        return cust_phone;
    }

    public void setCust_phone(String cust_phone) {
        this.cust_phone = cust_phone;
    }

    public String getCust_mobile() {
        return cust_mobile;
    }

    public void setCust_mobile(String cust_mobile) {
        this.cust_mobile = cust_mobile;
    }

    @Override
    public String toString() {
        return "Customer [cust_id=" + cust_id + ", cust_name=" + cust_name + ",
cust_source=" + cust_source
                + ", cust_industry=" + cust_industry + ", cust_level=" +
cust_level + ", cust_phone=" + cust_phone
                + ", cust_mobile=" + cust_mobile + "]";
    }

}
```

- 多的一方的实体

```
/**
* @Title: LinkMan.java
* @Package com.admiral.domain
* @Description: 联系人实体
* @author 白世鑫
* @date 2020-9-22
* @version V1.0
*/
package com.admiral.domain;

public class LinkMan {

    private Long lkm_id;
    private String lkm_name;
    private String lkm_gender;
    private String lkm_phone;
    private String lkm_mobile;
    private String lkm_email;
    private String lkm_qq;
    private String lkm_position;
    private String lkm_memo;

    // 通过 ORM 方式表示:一个联系人只能属于一个客户
    // 放置一的一方的对象.
    private Customer customer;

    public LinkMan() {
        super();
    }

    public LinkMan(Long lkm_id, String lkm_name, String lkm_gender, String
lkm_phone, String lkm_mobile,
            String lkm_email, String lkm_qq, String lkm_position, String
lkm_memo) {
        super();
        this.lkm_id = lkm_id;
        this.lkm_name = lkm_name;
        this.lkm_gender = lkm_gender;
        this.lkm_phone = lkm_phone;
        this.lkm_mobile = lkm_mobile;
        this.lkm_email = lkm_email;
        this.lkm_qq = lkm_qq;
        this.lkm_position = lkm_position;
        this.lkm_memo = lkm_memo;
    }

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
```

```
    }

    public Long getLkm_id() {
        return lkm_id;
    }

    public void setLkm_id(Long lkm_id) {
        this.lkm_id = lkm_id;
    }

    public String getLkm_name() {
        return lkm_name;
    }

    public void setLkm_name(String lkm_name) {
        this.lkm_name = lkm_name;
    }

    public String getLkm_gender() {
        return lkm_gender;
    }

    public void setLkm_gender(String lkm_gender) {
        this.lkm_gender = lkm_gender;
    }

    public String getLkm_phone() {
        return lkm_phone;
    }

    public void setLkm_phone(String lkm_phone) {
        this.lkm_phone = lkm_phone;
    }

    public String getLkm_mobile() {
        return lkm_mobile;
    }

    public void setLkm_mobile(String lkm_mobile) {
        this.lkm_mobile = lkm_mobile;
    }

    public String getLkm_email() {
        return lkm_email;
    }

    public void setLkm_email(String lkm_email) {
        this.lkm_email = lkm_email;
    }

    public String getLkm_qq() {
        return lkm_qq;
    }

    public void setLkm_qq(String lkm_qq) {
        this.lkm_qq = lkm_qq;
    }
```

```
    public String getLkm_position() {
        return lkm_position;
    }

    public void setLkm_position(String lkm_position) {
        this.lkm_position = lkm_position;
    }

    public String getLkm_memo() {
        return lkm_memo;
    }

    public void setLkm_memo(String lkm_memo) {
        this.lkm_memo = lkm_memo;
    }

    @Override
    public String toString() {
        return "LinkMan [lkm_id=" + lkm_id + ", lkm_name=" + lkm_name + ",
lkm_gender=" + lkm_gender + ", lkm_phone="
                + lkm_phone + ", lkm_mobile=" + lkm_mobile + ", lkm_email=" +
lkm_email + ", lkm_qq=" + lkm_qq
                + ", lkm_position=" + lkm_position + ", lkm_memo=" + lkm_memo +
"]";
    }

}
```

1.2.2.3 创建映射

- 多的一方的配置

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:      类中的全路径
            table属性:     表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:   数据库名,可以省略
    -->
    <class name="com.admiral.domain.LinkMan" table="cst_linkman">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="lkm_id" column="lkm_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="lkm_name" />
```

```
        <property name="lkm_gender" />
        <property name="lkm_phone" />
        <property name="lkm_mobile" />
        <property name="lkm_email" />
        <property name="cust_qq" />
        <property name="lkm_position" />
        <property name="lkm_memo" />


        <!-- 配置多对一关系:放置的是一的一方的对象 -->
        <many-to-one name="customer" column="lkm_cust_id"
class="com.admiral.domain.Customer" />

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

- 一的一方的配置

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:     类中的全路径
            table属性:    表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:  数据库名,可以省略
    -->
    <class name="com.admiral.pojo.Customer" table="cst_customer">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="cust_id" column="cust_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="cust_name" column="cust_name" />
        <property name="cust_source" column="cust_source" />
        <property name="cust_industry" column="cust_industry" />
        <property name="cust_level" column="cust_level" />
        <property name="cust_phone" column="cust_phone" />
        <property name="cust_mobile" column="cust_mobile" />

        <!-- 配置一对多关系:放置的多的一方的集合 -->
        <set name="linkMans">
            <key column="lkm_cust_id" />
            <one-to-many class="com.admiral.domain.LinkMan"/>
        </set>

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

1.2.2.4 将映射添加到配置文件

```
<?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>
        <!-- ================必要的配置=============== -->
        <!-- 必要的配置信息,连接数据库的基本参数 -->
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql:///hibernate_03</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">111111</property>
        <!-- Hibernate 的方言:根据配置的方言生成对应的 SQL 语句 -->
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- 配置C3P0连接池 -->
        <property
name="connection.provider_class">org.hibernate.connection.C3P0ConnectionProvider
</property>
        <!--在连接池中可用的数据库连接的最少数目 -->
        <property name="c3p0.min_size">5</property>
        <!--在连接池中所有数据库连接的最大数目 -->
        <property name="c3p0.max_size">20</property>
        <!--设定数据库连接的过期时间,以秒为单位, 如果连接池中的某个数据库连接处于空闲状态的时
间超过了timeout时间,就会从连接池中清除 -->
        <property name="c3p0.timeout">120</property>
        <!--每3000秒检查所有连接池中的空闲连接 以秒为单位 -->
        <property name="c3p0.idle_test_period">3000</property>

        <!-- ================可选的配置=============== -->
        <!-- Hibernate 显示 SQL 语句: -->
        <property name="hibernate.show_sql">true</property>
        <!-- Hibernate 格式化 SQL 语句: -->
        <property name="hibernate.format_sql">true</property>
        <!-- Hibernate 自动创建表 -->
        <property name="hibernate.hbm2ddl.auto">update</property>
        <!-- 配置 Session 绑定本地线程 -->
        <property
name="hibernate.current_session_context_class">thread</property>


        <!-- 加载映射文件 -->
        <mapping resource="com/admiral/domain/Customer.hbm.xml" />
        <mapping resource="com/admiral/domain/LinkMan.hbm.xml" />
```

```
    </session-factory>
</hibernate-configuration>
```

1.2.2.5 编写测试代码

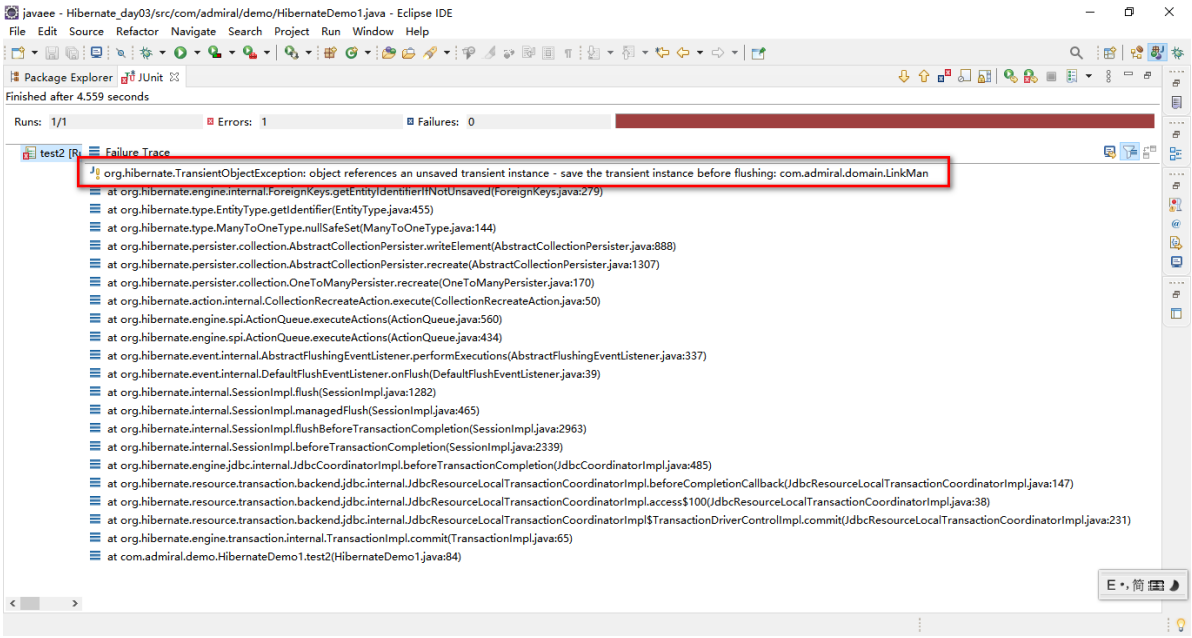
```
/**
* @Title: HibernateDemo1.java
* @Package com.admiral.demo
* @Description:
* @author 白世鑫
* @date 2020-9-22
* @version V1.0
*/
package com.admiral.demo;

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.junit.Test;

import com.admiral.domain.Customer;
import com.admiral.domain.LinkMan;
import com.admiral.utils.HibernateUtils;

public class HibernateDemo1 {

    @Test
    // 保存2个客户和3个联系人
    public void test1() {

        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        Customer customer1 = new Customer();
        customer1.setCust_name("小白白");
        Customer customer2 = new Customer();
        customer2.setCust_name("小黑黑");


        LinkMan linkMan1 = new LinkMan();
        linkMan1.setLkm_name("张三丰");
        LinkMan linkMan2 = new LinkMan();
        linkMan2.setLkm_name("欧阳锋");
        LinkMan linkMan3 = new LinkMan();
        linkMan3.setLkm_name("黄老邪");

        //设置关系
        linkMan1.setCustomer(customer1);
        linkMan2.setCustomer(customer1);
        linkMan3.setCustomer(customer2);

        customer1.getLinkMans().add(linkMan1);
        customer1.getLinkMans().add(linkMan2);
        customer2.getLinkMans().add(linkMan3);

        //保存数据
```

```
        session.save(linkMan1);
        session.save(linkMan2);
        session.save(linkMan3);
        session.save(customer1);
        session.save(customer2);


        transaction.commit();

    }
}
```

1.2.3 一对多的相关操作

- 一对多关系只保存一边是否可以

```
    @Test
    // 只保存一边是否可以
    public void test2() {

        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        Customer customer1 = new Customer();
        customer1.setCust_name("小白白");


        LinkMan linkMan1 = new LinkMan();
        linkMan1.setLkm_name("张三丰");

        //设置关系
        linkMan1.setCustomer(customer1);

        customer1.getLinkMans().add(linkMan1);

        //保存数据
        //只保存一边是否可以:不可以,报一个瞬时对象异常,因为持久态对象关联了瞬时态对象.
        session.save(customer1);


        transaction.commit();

    }
```



1.2.3.1 级联保存或更新

- 什么叫做级联
  - 级联指的是，操作一个对象的时候，是否会同时操作其关联的对象。
- 级联是有方向性
  - 操作一的一方的时候，是否操作到多的一方
  - 操作多的一方的时候，是否操作到一的一方

- 保存客户级联联系人

```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>
    <!--
        建立类和表的映射关系
            name属性:     类中的全路径
            table属性:    表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:  数据库名,可以省略
    -->
    <class name="com.admiral.domain.Customer" table="cst_customer">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="cust_id" column="cust_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="cust_name" column="cust_name" />
        <property name="cust_source" column="cust_source" />
        <property name="cust_industry" column="cust_industry" />
        <property name="cust_level" column="cust_level" />
        <property name="cust_phone" column="cust_phone" />
        <property name="cust_mobile" column="cust_mobile" />

        <!-- 配置一对多关系:放置的多的一方的集合 -->
        <set name="linkMans" cascade="save-update">
            <key column="lkm_cust_id" />
            <one-to-many class="com.admiral.domain.LinkMan"/>
        </set>

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

```java
    @Test
    // 保存客户级联联系人
    public void test3() {
```

```
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();
        Customer customer1 = new Customer();
        customer1.setCust_name("小白白");
        LinkMan linkMan1 = new LinkMan();
        linkMan1.setLkm_name("张三丰");
        //设置关系
        linkMan1.setCustomer(customer1);
        customer1.getLinkMans().add(linkMan1);
        //保存数据
        //只保存一边是否可以:不可以,报一个瞬时对象异常,因为持久态对象关联了瞬时态对象.
        session.save(customer1);
        transaction.commit();

    }
}
```

- 保存联系人级联客户

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:       类中的全路径
            table属性:      表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:    数据库名,可以省略
    -->
    <class name="com.admiral.domain.LinkMan" table="cst_linkman">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="lkm_id" column="lkm_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="lkm_name" />
        <property name="lkm_gender" />
        <property name="lkm_phone" />
        <property name="lkm_mobile" />
        <property name="lkm_email" />
        <property name="lkm_qq" />
        <property name="lkm_position" />
        <property name="lkm_memo" />


        <!-- 配置多对一关系:放置的是一的一方的对象 -->
        <many-to-one name="customer" cascade="save-update" column="lkm_cust_id" class="com.admiral.domain.Customer" />

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

```
    @Test
    // 保存联系人级联客户
    public void test4() {

        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();
        Customer customer1 = new Customer();
        customer1.setCust_name("小白白");
        LinkMan linkMan1 = new LinkMan();
        linkMan1.setLkm_name("张三丰");
        //设置关系
        linkMan1.setCustomer(customer1);
```

```
        customer1.getLinkMans().add(linkMan1);
        //保存数据
        //只保存一边是否可以:不可以,报一个瞬时对象异常,因为持久态对象关联了瞬时态对象.
        session.save(linkMan1);
        transaction.commit();

    }
```

1.2.3.2 测试对象的导航的问题

```
    @Test
    // 测试对象导航
    public void test5() {

        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        Customer customer1 = new Customer();
        customer1.setCust_name("小白白");

        LinkMan linkMan1 = new LinkMan();
        linkMan1.setLkm_name("张三丰");
        LinkMan linkMan2 = new LinkMan();
        linkMan2.setLkm_name("欧阳锋");
        LinkMan linkMan3 = new LinkMan();
        linkMan3.setLkm_name("黄老爷");

        linkMan1.setCustomer(customer1);
        customer1.getLinkMans().add(linkMan2);
        customer1.getLinkMans().add(linkMan3);

//      session.save(linkMan1);//发送4条 insert 语句
//      session.save(customer1);//发送3条 insert 语句
        session.save(linkMan2);//发送1条 insert 语句


        transaction.commit();

    }
```

1.2.3.3 Hibernate 的级联删除

- 级联删除:
  - 删除一边的时候，同时将另一方的数据也一并删除。
- 删除客户级联删除联系人

```
    <!-- 配置一对多关系:放置的多的一方的集合 -->
    <set name="linkMans" cascade="save-update,delete">
        <key column="lkm_cust_id" />
        <one-to-many class="com.admiral.domain.LinkMan"/>
    </set>

</class>
```

```
    @Test
    // 级联删除
    public void test6() {

        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        //默认情况:修改了联系人的外键,然后删除客户.
//      Customer customer = session.get(Customer.class, 1L);
//      session.delete(customer);

        //级联删除
        Customer customer = session.get(Customer.class, 1L);
        session.delete(customer);

        transaction.commit();

    }
```

- 删除联系人级联删除客户（基本不用）

1.2.3.4 双向关联产生多余的SQL语句

- 解决多余的SQL语句
  - 单向维护：
  - 使一方放弃外键维护权：
    - 一的一方放弃。**在set上配置inverse="true"**
  - **一对多的关联查询的修改的时候。（CRM练习--）**

```
    @Test
    public void test7() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        LinkMan linkMan = session.get(LinkMan.class, 2L);

        Customer customer = session.get(Customer.class, 2L);

        linkMan.setCustomer(customer);
        customer.getLinkMans().add(linkMan);

        transaction.commit();
    }
```

```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>
    <!--
        建立类和表的映射关系
            name属性:       类中的全路径
            table属性:      表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:    数据库名,可以省略
    -->
    <class name="com.admiral.domain.Customer" table="cst_customer">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="cust_id" column="cust_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="cust_name" column="cust_name" />
        <property name="cust_source" column="cust_source" />
        <property name="cust_industry" column="cust_industry" />
        <property name="cust_level" column="cust_level" />
        <property name="cust_phone" column="cust_phone" />
        <property name="cust_mobile" column="cust_mobile" />

        <!-- 配置一对多关系:放置的多的一方的集合 -->
        <set name="linkMans" cascade="save-update,delete" inverse="true">
            <key column="lkm_cust_id" />
            <one-to-many class="com.admiral.domain.LinkMan"/>
        </set>

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

1.2.3.5 区分 cascade 和 inverse

```java
    @Test
    // 区分 cascade 和 inverse
    public void test8() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        Customer customer = new Customer();
        customer.setCust_name("小红红");

        LinkMan linkMan = new LinkMan();
        linkMan.setLkm_name("小龙女");

        customer.getLinkMans().add(linkMan);
        //客户会插入到数据库,联系人也会插入到数据库,但是外键为 null
        session.save(customer);

        transaction.commit();
    }
```

案例二：完成多对多的关联关系映射并操作

1.3案例需求

1.3.1 需求描述

1.4相关知识点

1.4.1 Hibernate的多对多关联关系映射

1.4.1.1 创建表

- sys_user 表

```
CREATE TABLE `sys_user` (
  `user_id` bigint(32) NOT NULL AUTO_INCREMENT COMMENT '用户id',
  `user_code` varchar(32) DEFAULT NULL COMMENT '用户账号',
  `user_name` varchar(64) DEFAULT NULL COMMENT '用户名称',
  `user_password` varchar(32) DEFAULT NULL COMMENT '用户密码',
  `user_state` char(1) DEFAULT NULL COMMENT '1:正常,0:暂停',
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8;
```

- sys_role 表

```
CREATE TABLE `sys_role` (
  `role_id` bigint(32) NOT NULL AUTO_INCREMENT,
  `role_name` varchar(32) DEFAULT NULL COMMENT '角色名称',
  `role_memo` varchar(128) DEFAULT NULL COMMENT '备注',
  PRIMARY KEY (`role_id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;
```

- sys_user_role 表

```
CREATE TABLE `sys_user_role` (
  `role_id` bigint(32) NOT NULL COMMENT '角色id',
  `user_id` bigint(32) NOT NULL COMMENT '用户id',
  PRIMARY KEY (`role_id`,`user_id`),
  KEY `FK_user_role_user_id` (`user_id`),
  CONSTRAINT `FK_user_role_role_id` FOREIGN KEY (`role_id`) REFERENCES
`sys_role` (`role_id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `FK_user_role_user_id` FOREIGN KEY (`user_id`) REFERENCES
`sys_user` (`user_id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

1.4.1.2 创建实体

```
/**
* @Title: User.java
* @Package com.admiral.domain
* @Description: 用户实体
```

```java
* @author 白世鑫
* @date 2020-9-23
* @version V1.0
*/
package com.admiral.domain;

import java.util.HashSet;
import java.util.Set;

public class User {
    private Long user_id;
    private String user_code;
    private String user_name;
    private String user_password;
    private String user_state;

    // 设置多对躲关系,表示一个用户选择多个角色
    // 放置的是角色的集合
    private Set<Role> roles = new HashSet<Role>();

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }

    public Long getUser_id() {
        return user_id;
    }

    public void setUser_id(Long user_id) {
        this.user_id = user_id;
    }

    public String getUser_code() {
        return user_code;
    }

    public void setUser_code(String user_code) {
        this.user_code = user_code;
    }

    public String getUser_name() {
        return user_name;
    }

    public void setUser_name(String user_name) {
        this.user_name = user_name;
    }

    public String getUser_password() {
        return user_password;
    }

    public void setUser_password(String user_password) {
        this.user_password = user_password;
```

```
    }

    public String getUser_state() {
        return user_state;
    }

    public void setUser_state(String user_state) {
        this.user_state = user_state;
    }
}
```

```
/**
* @Title: Role.java
* @Package com.admiral.domain
* @Description: 角色实体
* @author 白世鑫
* @date 2020-9-23
* @version V1.0
*/
package com.admiral.domain;

import java.util.HashSet;
import java.util.Set;

public class Role {

    private Long role_id;
    private String role_name;
    private String role_memo;

    // 设置多对躲关系,表示一个角色被多个用户选择
    // 放置的是用户的集合
    private Set<User> users = new HashSet<User>();

    public Set<User> getUsers() {
        return users;
    }

    public void setUsers(Set<User> users) {
        this.users = users;
    }

    public Long getRole_id() {
        return role_id;
    }

    public void setRole_id(Long role_id) {
        this.role_id = role_id;
    }

    public String getRole_name() {
        return role_name;
    }
```

```
    public void setRole_name(String role_name) {
        this.role_name = role_name;
    }

    public String getRole_memo() {
        return role_memo;
    }

    public void setRole_memo(String role_memo) {
        this.role_memo = role_memo;
    }

}
```

1.4.1.3 创建映射

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:      类中的全路径
            table属性:     表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:   数据库名,可以省略
    -->
    <class name="com.admiral.domain.User" table="sys_user">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="user_id" column="user_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="user_code" column="user_code" />
        <property name="user_name" column="user_name" />
        <property name="user_password" column="user_password" />
        <property name="user_state" column="user_state" />

        <!-- 配置多对多关系:放置的是角色的集合 -->
        <!--
            name:        对方的集合的属性名称
            tables:      多对多关系需要使用中间表,放的是中间表的名称
         -->
        <set name="roles" table="sys_user_role">
            <!--
```

```
                column: 当前对象对应中间表的外键的名称
            -->
            <key column="user_id"/>
            <!--
                class:      对方的类的全路径
                column:     对方的对象在中间表中的外键的名称
             -->
            <many-to-many class="com.admiral.domain.Role" column="role_id"/>
        </set>

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

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:     类中的全路径
            table属性:    表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:  数据库名,可以省略
    -->
    <class name="com.admiral.domain.Role" table="sys_role">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="role_id" column="role_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="role_name" column="role_name" />
        <property name="role_memo" column="role_memo" />

        <!-- 配置多对多关系:放置的是用户的集合 -->
        <!--
            name         :对方的属性的集合名称
            table        :中间表的表名
         -->
        <set name="users" table="sys_user_role">
            <!--
                column       :当前对象对应中间表中外键的名称
             -->
            <key column="role_id"/>
            <!--
                class        :对方的类的全路径
                column       :对方的对象在中间表中外键的名称
             -->
            <many-to-many class="com.admiral.domain.User" column="user_id"/>
        </set>
```

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

1.4.1.4 在核心配置中加入映射文件

```
<?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>
        <!-- =================必要的配置================ -->
        <!-- 必要的配置信息,连接数据库的基本参数 -->
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql:///hibernate_03</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">111111</property>
        <!-- Hibernate 的方言:根据配置的方言生成对应的 SQL 语句 -->
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- 配置C3P0连接池 -->
        <property
name="connection.provider_class">org.hibernate.connection.C3P0ConnectionProvider
</property>
        <!--在连接池中可用的数据库连接的最少数目 -->
        <property name="c3p0.min_size">5</property>
        <!--在连接池中所有数据库连接的最大数目 -->
        <property name="c3p0.max_size">20</property>
        <!--设定数据库连接的过期时间,以秒为单位, 如果连接池中的某个数据库连接处于空闲状态的时
间超过了timeout时间,就会从连接池中清除 -->
        <property name="c3p0.timeout">120</property>
        <!--每3000秒检查所有连接池中的空闲连接 以秒为单位 -->
        <property name="c3p0.idle_test_period">3000</property>

        <!-- =================可选的配置================ -->
        <!-- Hibernate 显示 SQL 语句: -->
        <property name="hibernate.show_sql">true</property>
        <!-- Hibernate 格式化 SQL 语句: -->
        <property name="hibernate.format_sql">true</property>
        <!-- Hibernate 自动创建表 -->
        <property name="hibernate.hbm2ddl.auto">create</property>
        <!-- 配置 Session 绑定本地线程 -->
        <property
name="hibernate.current_session_context_class">thread</property>


        <!-- 加载映射文件 -->
```

```
        <mapping resource="com/admiral/domain/Customer.hbm.xml" />
        <mapping resource="com/admiral/domain/LinkMan.hbm.xml" />
        <mapping resource="com/admiral/domain/Role.hbm.xml" />
        <mapping resource="com/admiral/domain/User.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

1.4.1.5 编写测试类

```
/**
* @Title: HibernateDemo2.java
* @Package com.admiral.demo2
* @Description: Hibernate 多对多关系测试
* @author 白世鑫
* @date 2020-9-23
* @version V1.0
*/
package com.admiral.demo2;

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.junit.Test;

import com.admiral.domain.Role;
import com.admiral.domain.User;
import com.admiral.utils.HibernateUtils;

public class HibernateDemo2 {

    @Test
    public void test1() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        //创建两个用户
        User user1 = new User();
        user1.setUser_name("小红红");
        User user2 = new User();
        user2.setUser_name("小花花");

        //创建三个角色
        Role role1 = new Role();
        role1.setRole_name("研发部");
        Role role2 = new Role();
        role2.setRole_name("财务部");
        Role role3 = new Role();
        role3.setRole_name("公关部");

        //设置双向的关联关系
        user1.getRoles().add(role1);
        user1.getRoles().add(role2);
        user2.getRoles().add(role2);
        user2.getRoles().add(role3);
```

```
        role1.getUsers().add(user1);
        role2.getUsers().add(user1);
        role2.getUsers().add(user2);
        role3.getUsers().add(user2);

        //保存操作:多对多建立了双向关系必须有一方放弃外键的维护权
        //一般是被动方放弃维护权
        session.save(user1);
        session.save(user2);
        session.save(role1);
        session.save(role2);
        session.save(role3);


        transaction.commit();
    }
}
```

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:       类中的全路径
            table属性:      表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:    数据库名,可以省略
    -->
    <class name="com.admiral.domain.Role" table="sys_role">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="role_id" column="role_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="role_name" column="role_name" />
        <property name="role_memo" column="role_memo" />

        <!-- 配置多对多关系:放置的是用户的集合 -->
        <!--
            name          :对方的属性的集合名称
            table         :中间表的表名
         -->
        <set name="users" table="sys_user_role" inverse="true">
            <!--
                column        :当前对象对应中间表中外键的名称
             -->
            <key column="role_id"/>
            <!--
                class         :对方的类的全路径
                column        :对方的对象在中间表中外键的名称
             -->
            <many-to-many class="com.admiral.domain.User" column="user_id"/>
        </set>


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

1.4.2 多对多的相关操作

- 只保存一边是否可以

```
    @Test
    //只保存一边是否可以?不可以
    public void test2() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();
```

```
        // 创建两个用户
        User user1 = new User();
        user1.setUser_name("小红红");

        // 创建三个角色
        Role role1 = new Role();
        role1.setRole_name("研发部");

        // 设置双向的关联关系
        user1.getRoles().add(role1);
        role1.getUsers().add(user1);

        // 只保存一边是否可以
        session.save(user1);

        transaction.commit();
    }
```

1.4.2.1 级联保存或更新

- 保存用户级联保存角色

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:     类中的全路径
            table属性:    表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:  数据库名,可以省略
    -->
    <class name="com.admiral.domain.User" table="sys_user">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="user_id" column="user_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="user_code" column="user_code" />
        <property name="user_name" column="user_name" />
        <property name="user_password" column="user_password" />
        <property name="user_state" column="user_state" />




        <set name="roles" table="sys_user_role" cascade="save-update">
            <!--
                column: 当前对象对应中间表的外键的名称
            -->
            <key column="user_id"/>
            <!--
                class:      对方的类的全路径
                column:     对方的对象在中间表中的外键的名称
             -->
            <many-to-many class="com.admiral.domain.Role" column="role_id"/>
        </set>

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

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:      类中的全路径
            table属性:     表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:   数据库名,可以省略
    -->
    <class name="com.admiral.domain.User" table="sys_user">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="user_id" column="user_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="user_code" column="user_code" />
        <property name="user_name" column="user_name" />
        <property name="user_password" column="user_password" />
        <property name="user_state" column="user_state" />

        <!-- 配置多对多关系:放置的是角色的集合 -->
        <!--
            name:        对方的集合的属性名称
            tables:      多对多关系需要使用中间表,放的是中间表的名称
         -->
        <set name="roles" table="sys_user_role" cascade="save-update">
            <!--
                column: 当前对象对应中间表的外键的名称
            -->
            <key column="user_id"/>
            <!--
                class:       对方的类的全路径
                column:      对方的对象在中间表中的外键的名称
             -->
            <many-to-many class="com.admiral.domain.Role" column="role_id"/>
        </set>

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

```
    @Test
    //保存用户级联保存角色
    //在 User.hbm.xml 中的 set 上配置 cascade="save-update"
    public void test3() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        // 创建两个用户
        User user1 = new User();
        user1.setUser_name("小红红");

        // 创建三个角色
        Role role1 = new Role();
        role1.setRole_name("研发部");

        // 设置双向的关联关系
```

```
        user1.getRoles().add(role1);
        role1.getUsers().add(user1);

        // 只保存一边是否可以
        session.save(user1);

        transaction.commit();
    }
```

- 保存角色级联保存用户

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:       类中的全路径
            table属性:      表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:    数据库名,可以省略
    -->
    <class name="com.admiral.domain.Role" table="sys_role">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="role_id" column="role_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="role_name" column="role_name" />
        <property name="role_memo" column="role_memo" />

        <!--配置多对多关系:放置的是用户的集合 -->
        <!--
            name          :对方的属性的集合名称
            table         :中间表的表名
         -->
        <set name="users" table="sys_user_role" cascade="save-update" inverse="true">
            <!--
                column       :当前对象对应中间表中外键的名称
             -->
            <key column="role_id"/>
            <!--
                class        :对方的类的全路径
                column       :对方的对象在中间表中外键的名称
             -->
            <many-to-many class="com.admiral.domain.User" column="user_id"/>
        </set>


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

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:       类中的全路径
            table属性:      表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:    数据库名,可以省略
    -->
    <class name="com.admiral.domain.Role" table="sys_role">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="role_id" column="role_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
```

```
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="role_name" column="role_name" />
        <property name="role_memo" column="role_memo" />

        <!-- 配置多对多关系:放置的是用户的集合 -->
        <!--
            name         :对方的属性的集合名称
            table        :中间表的表名
         -->
        <set name="users" table="sys_user_role" cascade="save-update"
inverse="true">
            <!--
                column       :当前对象对应中间表中外键的名称
             -->
            <key column="role_id"/>
            <!--
                class        :对方的类的全路径
                column       :对方的对象在中间表中外键的名称
             -->
            <many-to-many class="com.admiral.domain.User" column="user_id"/>
        </set>


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

```
    @Test
    // 保存角色级联保存用户
    // 在 Role.hbm.xml 中的 set 上配置 cascade="save-update"
    public void test4() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        User user1 = new User();
        user1.setUser_name("小拜拜");

        Role role1 = new Role();
        role1.setRole_name("财务部");

        // 设置双向的关联关系
        user1.getRoles().add(role1);
        role1.getUsers().add(user1);

        // 只保存一边是否可以
        session.save(role1);

        transaction.commit();
    }
```

1.4.2.2 级联删除：（了解）

- 删除用户级联删除角色

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:      类中的全路径
            table属性:     表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:   数据库名,可以省略
    -->
    <class name="com.admiral.domain.User" table="sys_user">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="user_id" column="user_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="user_code" column="user_code" />
        <property name="user_name" column="user_name" />
        <property name="user_password" column="user_password" />
        <property name="user_state" column="user_state" />

        <!-- 配置多对多关系:放置的是角色的集合 -->
        <!--
            name:        对方的集合的属性名称
            tables:      多对多关系需要使用中间表,放的是中间表的名称
         -->
        <set name="roles" table="sys_user_role" cascade="save-update,delete">
            <!--
                column: 当前对象对应中间表的外键的名称
            -->
            <key column="user_id"/>
            <!--
                class:       对方的类的全路径
                column:      对方的对象在中间表中的外键的名称
             -->
            <many-to-many class="com.admiral.domain.Role" column="role_id"/>
        </set>

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

```
    @Test
    // 删除用户级联删除角色
    // 在 User.hbm.xml 中的 set 上配置 cascade="delete"
    public void test5() {
        Session session = HibernateUtils.getCurrentSession();
```

```
        Transaction transaction = session.beginTransaction();

        //先查询再删除
        User user = session.get(User.class, 1L);
        session.delete(user);


        transaction.commit();
    }
```

- 删除角色级联删除用户

```
<?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>
    <!--
        建立类和表的映射关系
            name属性:      类中的全路径
            table属性:     表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:   数据库名,可以省略
    -->
    <class name="com.admiral.domain.Role" table="sys_role">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="role_id" column="role_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="role_name" column="role_name" />
        <property name="role_memo" column="role_memo" />

        <!-- 配置多对多关系:放置的是用户的集合 -->
        <!--
            name         :对方的属性的集合名称
            table        :中间表的表名
         -->
        <set name="users" table="sys_user_role" cascade="save-update,delete"
inverse="true">
            <!--
                column       :当前对象对应中间表中外键的名称
             -->
            <key column="role_id"/>
            <!--
                class        :对方的类的全路径
                column       :对方的对象在中间表中外键的名称
             -->
            <many-to-many class="com.admiral.domain.User" column="user_id"/>
        </set>
```

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

```
    @Test
    // 删除角色级联删除用户
    // 在 Role.hbm.xml 中的 set 上配置 cascade="delete"
    public void test6() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        //先查询再删除
        Role role = session.get(Role.class, 3L);
        session.delete(role);


        transaction.commit();
    }
```

1.4.2.3 多对多的其他操作

- 给用户选择角色

```
    @Test
    // 给用户选择角色
    // 给1号用户添加3号角色
    public void test7() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        // 查询1号用户
        User user = session.get(User.class, 1L);
        // 查询3号角色
        Role role = session.get(Role.class, 3L);

        user.getRoles().add(role);

        transaction.commit();
    }
```

- 给用户改选角色

```
    @Test
    // 给用户改选角色
    // 将2号用户原有的2号角色改为1号角色
    public void tes8() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        // 查询2号用户
        User user = session.get(User.class, 2L);
```

```
        // 查询2号角色
        Role role2 = session.get(Role.class, 2L);
        // 查询1号角色
        Role role1 = session.get(Role.class, 1L);
        user.getRoles().remove(role2);
        user.getRoles().add(role1);

        transaction.commit();
    }
```

- 给用户删除角色

```
    @Test
    // 给用户删除角色
    // 将2号用户的3号角色删除
    public void tes9() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

        // 查询2号用户
        User user = session.get(User.class, 2L);
        // 查询3号角色
        Role role3 = session.get(Role.class, 3L);
        user.getRoles().remove(role3);

        transaction.commit();
    }
```