1. 表与表之间关系回顾（重点）

1. 一对多

（1）分类和商品关系，一个分类里面有多个商品，一个商品只能属于一个分类

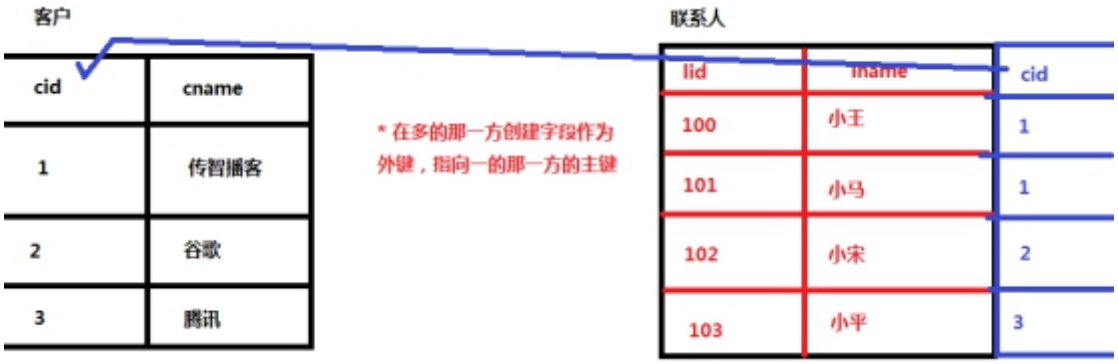
（2）客户和联系人是一对多关系

- 客户：与公司有业务往来，百度、新浪、360
- 联系人：公司里面的员工，百度里面有很多员工，联系员工

** 公司和公司员工的关系

- 客户是一，联系人是多
- 一个客户里面有多个联系人，一个联系人只能属于一个客户

（3）一对多建表：通过外键建立关系



2. 多对多

（1）订单和商品关系，一个订单里面有多个商品，一个商品属于多个订单

（2）用户和角色多对多关系

- 用户： 小王、小马、小宋
- 角色：总经理、秘书、司机、保安

** 比如小王 可以 是总经理，可以是司机

** 比如小宋 可以是司机，可以是秘书，可以保安

** 比如小马 可以是 秘书，可以是总经理

- 一个用户里面可以有多个角色，一个角色里面可以有多个用户

（3）多对多建表：创建第三张表维护关系

3. 一对一

（1）在中国，一个男人只能有一个妻子，一个女人只能有一个丈夫

2. Hibernate的一对多操作（重点）

1. 一对多映射配置（重点）

以客户和联系人为例：客户是一，联系人是多

第一步 创建两个实体类，客户和联系人

第二步 让两个实体类之间互相表示

（1）在客户实体类里面表示多个联系人

- 一个客户里面有多个联系人

```
//在客户实体类里面表示多个联系人，一个客户有多个联系人
//hibernate要求使用集合表示多的数据，使用set集合
private Set<LinkMan> setLinkMan = new HashSet<LinkMan>();
public Set<LinkMan> getSetLinkMan() {
    return setLinkMan;
}
public void setSetLinkMan(Set<LinkMan> setLinkMan) {
    this.setLinkMan = setLinkMan;
}
```

（2）在联系人实体类里面表示所属客户

- 一个联系人只能属于一个客户

```java
// 在联系人实体类里面表示所属客户,一个联系人只能属于一个客户
private Customer customer;
public Customer getCustomer() {
    return customer;
}
public void setCustomer(Customer customer) {
    this.customer = customer;
}
```

第三步 配置映射关系

（1）一般一个实体类对应一个映射文件

（2）把映射最基本配置完成

（3）在映射文件中，配置一对多关系

- 在客户映射文件中，表示所有联系人

```xml
<!-- 在客户映射文件中，表示所有联系人
     使用set标签表示所有联系人
     set标签里面有name属性，
       属性值写在客户实体类里面表示联系人的set集合名称
-->
<set name="setLinkMan">
    <!-- 一对多建表，有外键
         hibernate机制，双向维护外键，在一和多那一方都配置外键
         column属性值，外键名称
     -->
    <key column="clid"></key>
    <!-- 客户所有的联系人，class里面写联系人实体类全路径 -->
    <one-to-many class="cn.itcast.entity.LinkMan"/>
</set>
```

- 在联系人映射文件中，表示所属客户

```xml
<!-- 表示联系人所属客户
    name属性，因为在联系人实体类使用customer对象表示，写customer名称
    class属性，customer全路径
    column属性，外键名称
-->
<many-to-one name="customer" class="cn.itcast.entity.Customer" column="clid">
```

第四步 创建核心配置文件，把映射文件引入到核心配置文件中

```xml
<!-- 第三部分，把映射文件放到核心配置文件中 必须的-->
<mapping resource="cn/itcast/entity/Customer.hbm.xml"/>
<mapping resource="cn/itcast/entity/LinkMan.hbm.xml"/>
```

测试：

```
CREATE TABLE `t_linkman` (
  `lkm_id` int(11) NOT NULL AUTO_INCREMENT,
  `lkm_name` varchar(255) DEFAULT NULL,
  `lkm_gender` varchar(255) DEFAULT NULL,
  `lkm_phone` varchar(255) DEFAULT NULL,
  `clid` int(11) DEFAULT NULL,
  PRIMARY KEY (`lkm_id`),
  KEY `FKjtgu0oocf35ij4fmulu123vwk` (`clid`),
  CONSTRAINT `FKjtgu0oocf35ij4fmulu123vwk` FOREIGN KEY (`clid`) REFERENCES `t_customer` (`cid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

2. 一对多级联操作概述

级联操作

1 级联保存

（1）添加一个客户，为这个客户添加多个联系人

2 级联删除

（1）删除某一个客户，这个客户里面的所有的联系人也删除

3. 一对多级联保存

1. 添加客户，为这个客户添加一个联系人

（1）复杂写法：

```
//演示一对多级联保存
@Test
public void testAddDemo1() {
        SessionFactory sessionFactory = null;
        Session session = null;
        Transaction tx = null;
    try  {
        //得到sessionFactory
        sessionFactory = HibernateUtils.getSessionFactory();
        session = sessionFactory.openSession();
        //开启事务
        tx = session.beginTransaction();
        // 添加一个客户，为这个客户添加一个联系人
        //1 创建客户和联系人对象
        Customer customer = new Customer();
        customer.setCustName("传智播客");
        customer.setCustLevel("vip");
        customer.setCustSource("网络");
        customer.setCustPhone("110");
        customer.setCustMobile("999");
```

```
LinkMan linkman = new LinkMan();
linkman.setLkm_name("lucy");
linkman.setLkm_gender("男");
linkman.setLkm_phone("911");
//2 在客户表示所有联系人，在联系人表示客户
// 建立客户对象和联系人对象关系
//2.1 把联系人对象 放到客户对象的set集合里面
customer.getSetLinkMan().add(linkman);
//2.2 把客户对象放到联系人里面
linkman.setCustomer(customer);
//3 保存到数据库
session.save(customer);
session.save(linkman);
//提交事务
tx.commit();
}catch(Exception e) {
        tx.rollback();
}finally {
        session.close();
        //sessionFactory不需要关闭
        sessionFactory.close();
        }
}
```

cid	custName	custLevel	custSource	custPhone	custMobile
2	传智播客	vip	网络	110	999

lkm_id	lkm_name	lkm_gender	lkm_phone	clid
2	lucy	男	911	2

（2）简化写法

- 一般根据客户添加联系人

第一步 在客户映射文件中进行配置

- 在客户映射文件里面set标签进行配置

```
<set name="setLinkMan" cascade="save-update">
```

第二步 创建客户和联系人对象，只需要把联系人放到客户里面就可以了，最终只需要保存客户就可以了

```
//演示一对多级联保存
```

```
@Test
public void testAddDemo2() {
        SessionFactory sessionFactory = null;
        Session session = null;
        Transaction tx = null;
try {
        //得到sessionFactory
        sessionFactory = HibernateUtils.getSessionFactory();
        //得到session
        session = sessionFactory.openSession();
        //开启事务
        tx = session.beginTransaction();
        // 添加一个客户，为这个客户添加一个联系人
        //1 创建客户和联系人对象
        Customer customer = new Customer();
        customer.setCustName("百度");
        customer.setCustLevel("普通客户");
        customer.setCustSource("网络");
        customer.setCustPhone("110");
        customer.setCustMobile("999");

        LinkMan linkman = new LinkMan();
        linkman.setLkm_name("小宏");
        linkman.setLkm_gender("男");
        linkman.setLkm_phone("911");
        //2 把联系人放到客户里面
        customer.getSetLinkMan().add(linkman);
        //3 保存客户
        session.save(customer);
        //提交事务
        tx.commit();
}catch(Exception e) {
        tx.rollback();
}finally {
        session.close();
        //sessionFactory不需要关闭
        sessionFactory.close();
        }
```

}

4.一对多级联删除

1 删除某个客户，把客户里面所有的联系人删除

2 具体实现

第一步 在客户映射文件set标签，进行配置

（1）使用属性cascade属性值 delete

```
<set name="setLinkMan" cascade="save-update,delete">
```

第二步 在代码中直接删除客户

（1）根据id查询对象，调用session里面delete方法删除

```
// 1 根据id查询客户对象
Customer customer = session.get(Customer.class, 3);
//2 调用方法删除
session.delete(customer);
```

3 执行过程：

（1）根据id查询客户

```
select
    customer0_.cid as cid1_0_0_,
    customer0_.custName as custName2_0_0_,
    customer0_.custLevel as custLeve3_0_0_,
    customer0_.custSource as custSour4_0_0_,
    customer0_.custPhone as custPhon5_0_0_,
    customer0_.custMobile as custMobi6_0_0_
from
    t_customer customer0_
where
    customer0_.cid=?
```

（2）根据外键id值查询联系人

```
select
    setlinkman0_.clid as clid5_1_0_,
    setlinkman0_.lkm_id as lkm_id1_1_0_,
    setlinkman0_.lkm_id as lkm_id1_1_1_,
    setlinkman0_.lkm_name as lkm_name2_1_1_,
    setlinkman0_.lkm_gender as lkm_gend3_1_1_
    setlinkman0_.lkm_phone as lkm_phon4_1_1_,
    setlinkman0_.clid as clid5_1_1_
from
    t_linkman setlinkman0_
where
    setlinkman0_.clid=?
```

（3）把联系人外键设置为null

```
update
    t_linkman
set
    clid=null
where
    clid=?
```

（4）删除联系人和客户

```
Hibernate:
    delete
    from
        t_linkman
    where
        lkm_id=?
Hibernate:
    delete
    from
        t_customer
    where
        cid=?
```

5. 一对多修改操作（inverse属性）

1 让lucy联系人所属客户不是传智播客，而是百度

```
//1 根据id查询lucy联系人，根据id查询百度的客户
Customer baidu = session.get(Customer.class, 1);
LinkMan lucy = session.get(LinkMan.class, 2);
//2 设置持久态对象值
//把联系人放到客户里面
baidu.getSetLinkMan().add(lucy);
//把客户放到联系人里面
lucy.setCustomer(baidu);
```

2 inverse属性

（1）因为hibernate双向维护外键，在客户和联系人里面都需要维护外键，修改客户时候修改一次外键，修改联系人时候也修改一次外键，造成效率问题

```
    update
        t_linkman
    set
        lkm_name=?,
        lkm_gender=?,
        lkm_phone=?,
        clid=?
    where
        lkm_id=?
ibernate:
    update
        t_linkman
    set
        clid=?
    where
        lkm_id=?
```

（2）解决方式：让其中的一方不维护外键

- 一对多里面，让其中一方放弃外键维护
- 一个国家有总统，国家有很多人，总统不能认识国家所有人，国家所有人可以认识总统

（3）具体实现：

在放弃关系维护映射文件中，进行配置，在set标签上使用inverse属性

```
inverse属性默认值：false不放弃关系维护
                  true表示放弃关系维护

name="setLinkMan" inverse="true">
```

3. Hibernate多对多操作

1. 多对多映射配置

以用户和角色为例演示

第一步 创建实体类，用户和角色

第二步 让两个实体类之间互相表示

（1）一个用户里面表示所有角色，使用set集合

```java
//一个用户可以有多个角色
private Set<Role> setRole = new HashSet<Role>();

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

（2）一个角色有多个用户，使用set集合

```java
// 一个角色有多个用户
private Set<User> setUser = new HashSet<User>();

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

第三步 配置映射关系

（1）基本配置

（2）配置多对多关系

- 在用户里面表示所有角色，使用set标签

```
<!-- 在用户里面表示所有角色，使用set标签
    name属性：角色set集合名称
    table属性：第三张表名称
-->
<set name="setRole" table="user_role">
    <!-- key标签里面配置
        配置当前映射文件在第三张表外键名称
    -->
    <key column="userid"></key>
    <!-- class：角色实体类全路径
         column：角色在第三张表外键名称
     -->
    <many-to-many class="cn.itcast.manytomany.Role" column="roleid"></
</set>
```

- 在角色里面表示所有用户，使用set标签

```
<!-- 在角色里面表示所有用户，使用set标签 -->
<set name="setUser" table="user_role">
    <!-- 角色在第三张表外键 -->
    <key column="roleid"></key>
    <many-to-many class="cn.itcast.manytomany.User" column="userid"
</set>
```

第四步 在核心配置文件中引入映射文件

```
<mapping resource="cn/itcast/manytomany/User.hbm.xml"/>
<mapping resource="cn/itcast/manytomany/Role.hbm.xml"/>
```

测试：

```
CONSTRAINT `FKkcalytill1i8ffamb9x4it65m` FOREIGN KEY (`userid`) REFERENCES `t_user` (`user_id`),
CONSTRAINT `FKcwof70ufe1sg8jh46fhrgv5wp` FOREIGN KEY (`roleid`) REFERENCES `t_role` (`role_id`)
```

2. 多对多级联保存

根据用户保存角色

第一步 在用户配置文件中set标签进行配置，cascade值save-update

```
-->
<set name="setRole" table="user_role" cascade="save-update">
```

第二步 写代码实现

（1）创建用户和角色对象，把角色放到用户里面，最终保存用户就可以了

```
//演示多对多修级联保存
@Test
public void testSave() {
    SessionFactory sessionFactory = null;
    Session session = null;
    Transaction tx = null;
```

```
try {
        //得到sessionFactory
        sessionFactory = HibernateUtils.getSessionFactory();
        //得到session
        session = sessionFactory.openSession();
        //开启事务
        tx = session.beginTransaction();
        //添加两个用户，为每个用户添加两个角色
        //1 创建对象
        User user1 = new User();
        user1.setUser_name("lucy");
        user1.setUser_password("123");
        User user2 = new User();
        user2.setUser_name("mary");
        user2.setUser_password("456");

        Role r1 = new Role();
        r1.setRole_name("总经理");
        r1.setRole_memo("总经理");
        Role r2 = new Role();
        r2.setRole_name("秘书");
        r2.setRole_memo("秘书");
        Role r3 = new Role();
        r3.setRole_name("保安");
        r3.setRole_memo("保安");
        //2 建立关系，把角色放到用户里面
        // user1 -- r1/r2
        user1.getSetRole().add(r1);
        user1.getSetRole().add(r2);
        // user2 -- r2/r3
        user2.getSetRole().add(r2);
        user2.getSetRole().add(r3);
        //3 保存用户
        session.save(user1);
        session.save(user2);
        //提交事务
        tx.commit();
 }catch(Exception e) {
```

```
            tx.rollback();
        }finally {
            session.close();
            //sessionFactory不需要关闭
            sessionFactory.close();
            }
        }
```

userid	roleid
1	1
1	2
2	2
2	3

3. 多对多级联删除（了解）

第一步 在set标签进行配置，cascade值delete

第二步 删除用户

```
cascade="save-update,delete">
```

```
User user = session.get(User.class, 1);
session.delete(user);
```

4. 维护第三张表关系

1 用户和角色多对多关系，维护关系通过第三张表维护

2 让某个用户有某个角色

第一步 根据id查询用户和角色

第二步 把角色放到用户里面

（1）把角色对象放到用户set集合

3 让某个用户没有某个角色

第一步 根据id查询用户和角色

```
// 让某个用户有某个角色
//让lucy有经纪人角色
//1 查询lucy和经纪人
User lucy = session.get(User.class, 1);
Role role = session.get(Role.class, 1);

//2 把角色放到用户的set集合里面
lucy.getSetRole().add(role);
```

第二步 从用户里面把角色去掉

（1）从set集合里面把角色移除

```
// 让某个用户没有有某个角色
User user = session.get(User.class, 2);
Role role = session.get(Role.class, 3);

//2 从用户里面把角色去掉
user.getSetRole().remove(role);
```