## 表与表之间的关系

* 1. 一对一：例如t\_person表和t\_card表，人和身份证表。这种情况需要找出主从关系，谁是主表，谁是从表。人可以没有身份证，但身份证必须要有人才行。所以人是主表，而身份证是从表。
  2. 一对多（多对一）：最为常见的就是一对多！一对多和多对一，这是从哪个角度去看得出来的。t\_stu 和t\_section 的关系，从t\_user 来看就是一对多，而从t\_section的角度来看就是多对一！这种情况都是在多方创建外键！
  3. 多对多：例如t\_stu和t\_teacher,即一个学生可以有多个老师，而一个老师也可以有多个学生。这种情况通常需要中间表来处理多对多的关系。例如在创建一张表t\_stu\_tea表，给出两个外键，一个相对t\_stu表的外键，另一个相对t\_teacher表的外键。

## 技术分析之Hibernate的关联关系映射之一对多映射（重点）

### 1. JavaWEB中一对多的设计及其建表原则

### 2. 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

### 3. 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**

public class Customer {

private Long cust\_id;

private String cust\_name;

private Long cust\_user\_id;

private Long cust\_create\_id;

private String cust\_source;

private String cust\_industry;

private String cust\_level;

private String cust\_linkman;

private String cust\_phone;

private String cust\_mobile;

private Set<Linkman> linkmans = new HashSet<Linkman>();

}

**联系人的JavaBean如下**

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;

private Customer customer;

}

### 4. 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**

<class name="com.qf.domain.Customer" table="cst\_customer">

<id name="cust\_id" column="cust\_id">

<generator class="native"/>

</id>

<property name="cust\_name" column="cust\_name"/>

<property name="cust\_user\_id" column="cust\_user\_id"/>

<property name="cust\_create\_id" column="cust\_create\_id"/>

<property name="cust\_source" column="cust\_source"/>

<property name="cust\_industry" column="cust\_industry"/>

<property name="cust\_level" column="cust\_level"/>

<property name="cust\_linkman" column="cust\_linkman"/>

<property name="cust\_phone" column="cust\_phone"/>

<property name="cust\_mobile" column="cust\_mobile"/>

<set name="linkmans">

<key column="lkm\_cust\_id"/>

<one-to-many class="com.qf.domain.Linkman"/>

</set>

</class>

**联系人的映射配置文件如下**

<class name="com.qf.domain.Linkman" table="cst\_linkman">

<id name="lkm\_id" column="lkm\_id">

<generator class="native"/>

</id>

<property name="lkm\_name" column="lkm\_name"/>

<property name="lkm\_gender" column="lkm\_gender"/>

<property name="lkm\_phone" column="lkm\_phone"/>

<property name="lkm\_mobile" column="lkm\_mobile"/>

<property name="lkm\_email" column="lkm\_email"/>

<property name="lkm\_qq" column="lkm\_qq"/>

<property name="lkm\_position" column="lkm\_position"/>

<property name="lkm\_memo" column="lkm\_memo"/>

<many-to-one name="customer" class="com.qf.domain.Customer" column="lkm\_cust\_id"/>

</class>

## 技术分析之保存客户和联系人的数据

### 进行双向关联进行数据的保存

### 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"

### 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 customers where cid = 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" />

### 技术分析之级联的取值（cascade的取值）和孤儿删除

**1. 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 save-update,delete -- 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。

<many-to-one cascade="delete-orphan" />

### 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">

### 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）

2. inverse用来维护外键的

## 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**

### 1. 编写用户和角色的JavaBean

**用户的JavaBean代码如下**

public class User {

private Long user\_id;

private String user\_code;

private String user\_name;

private String user\_password;

private String user\_state;

private Set<Role> roles = new HashSet<Role>();

}

**角色的JavaBean代码如下**

public class Role {

private Long role\_id;

private String role\_name;

private String role\_memo;

private Set<User> users = new HashSet<User>();

}

### 2. 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**

<class name="com.qf.domain.User" table="sys\_user">

<id name="user\_id" column="user\_id">

<generator class="native"/>

</id>

<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">

<key column="user\_id"/>

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

</set>

</class>

**角色的映射配置文件如下**

<class name="com.qf.domain.Role" table="sys\_role">

<id name="role\_id" column="role\_id">

<generator class="native"/>

</id>

<property name="role\_name" column="role\_name"/>

<property name="role\_memo" column="role\_memo"/>

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

<key column="role\_id"/>

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

</set>

</class>

### 3. 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

**1. 级联保存**

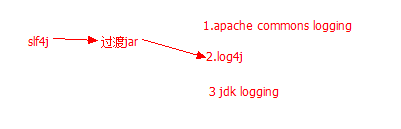
<set cascade="save-update">

## 级联删除（在多对多中是很少使用的）

1. 级联删除

## 整合log4j(了解)

* slf4j 核心jar ： slf4j-api-1.6.1.jar 。slf4j是日志框架，将其他优秀的日志第三方进行整合。



* 整合导入jar包

log4j 核心包：log4j-1.2.17.jar

过渡jar（整合jar）：slf4j-log4j12-1.7.5.jar

* 导入配置文件

log4j.properties ，此配置文件通知log4j 如何输出日志

* 配置文件内容：

1.记录器

2.输出源

3.布局

* 记录器

例如： log4j.rootLogger=info, stdout,file

格式：log4j.rootLogger=日志级别, 输出源1，输出源2，。。。。

log4j 日志级别 ： fatal 致命错误 error 错误 warn 警告 info 信息 debug 调试信息 trace 堆栈信息 （由高到低顺序）

* 输出源：

例如：log4j.appender.file=org.apache.log4j.FileAppender

格式：log4j.appender.输出源的名称=输出源的实现类

名称：自定义

实现类：log4j提供

输出源属性例如：log4j.appender.file.File=d\\:mylog.log

输出源属性格式：log4j.appender.名称.属性=值

每一个输出源对应一个实现类，实现类都属性（setter），底层执行setter方法进行赋值

* 常见的输出源实现类

org.apache.log4j.FileAppender 输出文件中

file ,表示文件输出位置

org.apache.log4j.ConsoleAppender 输出到控制台

Target ，表示使用哪种输出方式，在控制台打印内容，取值：System.out / System.err

* 布局 -- 确定输出格式

例如：log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

格式：log4j.appender.数据源.layout=org.apache.log4j.PatternLayout

布局属性：log4j.appender. 数据源.layout.ConversionPattern=值

12:56:30,123 info

* 扩展：对指定的目录设置日志级别

例如：log4j.logger.org.hibernate.transaction=debug

格式：log4j.logger.包结构=级别