作业：

一对多与多对一练习，需求自拟！

-🡪 用户与地址

🡪 用户与订单

🡪 老师与学员

🡪 部门/ 员工

回顾复习:

1. 介绍hibernate中对象的几种状态？

2. 介绍hibernate中懒加载属性

3. session缓存?

4. list与iterator查询区别?

5. 一对多与多对一，实现？

目标：

1. 关联映射中关联关系的维护

Inverse属性

2. 多对多映射 / cascade 属性

3. 一对一对映射

4. 其他映射

# 1. 关联映射中关联关系的维护

1. 需求： 部门与员工

关系：一对多与多对一的关系！

1. 数据库

主表： 部门表

外键表： 员工表 (部门的外键，引入部门的主键)

1. Javabean
2. 映射
3. 测试

关系维护：

A. 保存

B. 通过其中一方获取另外一方

C. 解除关系

D. 删除数据，对关系的影响！

1. Inverse属性

Inverse属性**在维护关联关系的时候的作用！**

Inverse属性，**表示的是控制权是否转移！**

**I**nverse属性， **在一的一方设置，如<set> 标签节点**

Inverse 反转

true 反转, 控制权转移，当前方没有控制权

false 不反转, 控制权不转移，当前方有控制权！

# 2. 多对多映射

1. 需求：

老师与学员

项目与开发人员

一个项目，对应多个开发人员

一个开发人员，参与多个项目

这种就是多对多关系！

1. 设计数据库：
2. Javabean / 映射
3. 测试：
   1. 基本多对多映射，是否配置正确
   2. 关联关系的维护
   3. Inverse属性，在多对多种的影响

总结：

一对多与多对多中，inverse属性在维护关联关系时候作用的区别?

一对多， 是否设置外键的值！

多对多， 是否删除中间表数据！

# 3. 一对一

需求： 存储用户、身份证信息

用户： 姓名、性别、年龄、身份证编号；

身份证：身份证编号、签发地、有效日期；

他们的关系就是一对一的关系！

数据库：

也可以通过主键生成策略来做。

# 3. 组件映射

对象之间的关系：

组合关系， 一个类中包含对另外一个类的引用，这2个类就是组合关系！

继承关系，一个类继承另外一个类！

映射，

组合关系的映射就是“组件映射”！

继承关系的映射就是”继承映射”！

* 组件类和包含的组件类同时映射到一个表

**需求：**

**汽车与车轮**

映射：

|  |
| --- |
| <component name=*"wheel"* class=*"Wheel"*>  <property name=*"count"*></property>  <property name=*"size"*></property>  </component> |

# 3. 继承映射

需求：

动物，猫，狗！

## 简单继承映射

特点：

父类、子类写到一个映射文件中！

有多少个子类，要写多少个映射文件.

总结：

缺点就是映射文件过多！

|  |
| --- |
| <hibernate-mapping package=*"cn.itcast.f\_extends1"*>  <class name=*"Dog"* table=*"t\_dog"*>  <!-- 继承父类的属性，直接写 -->  <id name=*"id"*>  <generator class=*"native"*></generator>  </id>  <property name=*"name"* length=*"20"*></property>    <!-- 子类属性 -->  <property name=*"play"* length=*"20"*></property>    </class>  </hibernate-mapping> |

## 复杂继承映射

### 方式1：整个继承结构一张表

特点：

所有的子类都写到一个映射文件中！

用一张表存储所有的子类信息！

|  |
| --- |
| <subclass name=*"Cat"* discriminator-value=*"cat\_"*>  <property name=*"catching"*></property>  </subclass> |

总结：

生成的表，不符合数据库设计原则！

因为所有子类都用一张表存储，存在很大的冗余字段！

### 方式2：三张表

特点：

完全按照面向对象设计表结构！

|  |
| --- |
| <joined-subclass name=*"Cat"* table=*"t\_cat"*>  <key column=*"id"*></key>  <property name=*"catching"*></property>  </joined-subclass> |

总结：

设计的表，是符合数据库设计原则（三大范式）!

但是，表结构、关系变得复杂，影响数据访问效率！

### 方式3：每个子类用一张表

特点：

一个映射文件

有多少个子类对应多少个表，父类不对应表！

|  |
| --- |
| <!-- 注意：使用union-subclass要求主键不能自增长！ -->  <!-- 子类：猫 -->  <union-subclass name=*"Cat"* table=*"t\_cat"*>  <property name=*"catching"*></property>  </union-subclass> |

总结：

推荐使用！

注意：主键不能为自增长！

# 3. cascade 级联操作

cascade 表示级联操作，在一对多、多对一、一对一种都可以设置

save-update 级联保存或更新

delete 级联删除

save-update,delete 级联保存、更新、删除

all 同上

none 不级联(默认值)

未完成：

hibernate查询：

二级缓存/连接池支持

Hibernate对session管理(案例)

# 4. hibernate查询

Hibernate提供的查询：

1. 主键查询

get/load

2) 对象导航方式

3) HQL查询

Hibernate提供的面向对象的查询方式！