**Hibernate的策略**

* Hibernate的持久化类的编写规则
  + 无参数构造
  + 属性私有
  + 属性尽量使用包装类
  + 提供一个唯一OID与主键对应
  + 不要使用final修饰
* Hibernate的主键生成策略
  + 主键分类
    - 自然主键
    - 代理主键
  + 主键生成策略
    - increment
    - identity
    - sequence
    - uuid
    - native
    - assigned
    - foreign
* Hibernate的持久化类的三种状态
  + 瞬时态：没有唯一标识OID，没有被session管理
  + 持久态：有唯一标识OID，已经被session管理
  + 脱管态：有唯一标识OID，没有被session管理
  + 状态转换：（了解）
* Hibernate的一级缓存
  + 一级缓存：Hibernate优化手段，称为是session级别缓存。
  + 一级缓存：快照区
* Hibernate的事务管理
  + 事务的回顾
    - 事务的概念
    - 事务的特性
    - 引发安全性问题
    - 安全性问题解决
  + Hibernate解决读问题
    - 配置设置隔离级别
  + Hibernate解决Service事务
    - 采用的是线程绑定的方式：
* Hibernate的其他的API
  + Query ：HQL 面向对象方式的查询。
  + Criteria ：QBC 完成面向对象化。
  + SQLQuery ：SQL查询

#### 一对多的级联操作

* 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

#### 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

Cascade ： “delete”

Cascade ： “save-update”

* 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--）**
* Hibernate的一对多
  + 表与表之间关系
    - 一对多关系
    - 多对多关系
    - 一对一关系
  + Hibernate的一对多配置
    - 搭建Hibernate基本环境
    - 创建表
    - 创建实体
      * 一的一方
        + 放的是多的一方的集合
      * 多的一方
        + 放的是一的一方的对象
    - 创建映射
      * 一的一方
        + 配置的<set>集合
      * 多的一方
        + 配置<many-to-one>
    - 编写测试类
  + Hibernate的一对多的操作
    - 级联操作：cascade，用于操作其关联的对象。
      * 级联保存或更新
      * 级联删除
    - 测试对象导航
    - 放弃外键维护权：inverse，用户控制是否有外键维护能力
* Hibernate的多对多
  + Hibernate的多对多配置
    - 搭建Hibernate环境
    - 创建表
    - 创建实体
      * 放置的是对方的集合
    - 创建映射
      * 配置的是对象的<set>
    - 编写测试类
  + Hibernate的多对多操作
    - 级联操作
      * 级联保存或更新
      * 级联删除（了解）
    - 其他的操作
      * 给用户选择角色
      * 给用户改选角色
      * 给用户删除角色

### Hibernate的查询方式：OID查询

OID检索：Hibernate根据对象的OID（主键）进行检索

#### 使用get方法

Customer customer = session.get(Customer.class,1l);

#### 使用load方法

Customer customer = session.load(Customer.class,1l);

### Hibernate的查询方式：对象导航检索

对象导航检索：Hibernate根据一个已经查询到的对象，获得其关联的对象的一种查询方式。

LinkMan linkMan = session.get(LinkMan.class,1l);

**Customer customer = linkMan.getCustomer();**

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

**Set<LinkMan> linkMans = customer.getLinkMans();**

### Hibernate的查询方式：HQL检索

HQL查询：Hibernate Query Language，Hibernate的查询语言，是一种面向对象的方式的查询语言，语法类似SQL。通过session.createQuery()，用于接收一个HQL进行查询方式。

#### HQL的多表查询

* SQL的多表查询
  + 连接查询
    - 交叉连接：笛卡尔积

select \* from A,B;

* + - 内连接 :inner join (inner 可以省略)
      * 隐式内连接：

select \* from A,B where A.id = B.aid;

* + - * 显示内连接：

select \* from A inner join B on A.id = B.aid;

* + - 外连接 :
      * 左外连接:left outer join(outer 可以省略)

select \* from A left outer join B on A.id= B.aid;

* + - * 右外连接:right outer join(outer 可以省略)

select \* from A right outer join B on A.id = B.aid;

* + 子查询
* HQL的多表查询
  + 连接查询
    - 交叉连接
    - 内连接
      * 显示内连接
      * 隐式内连接
      * **迫切内连接**
    - 外连接
      * 左外连接
      * 右外连接
      * **迫切左外连接**

### Hibernate的查询方式：QBC检索

QBC查询：Query By Criteria，条件查询。是一种更加面向对象化的查询的方式。

#### 延迟加载的分类

* 类级别的延迟加载
  + 指的是通过load方法查询某个对象的时候，是否采用延迟。session.load(Customer.class,1l);
  + 类级别延迟加载通过<class>上的lazy进行配置，如果让lazy失效
    - 将lazy设置为false
    - 将持久化类使用final修饰
    - Hibernate. Initialize()
* 关联级别的延迟加载
  + 指的是在查询到某个对象的时候，查询其关联的对象的时候，是否采用延迟加载。

Customer customer = session.get(Customer.class,1l);

**customer.getLinkMans();----通过客户获得联系人的时候，联系人对象是否采用了延迟加载，称为是关联级别的延迟。**

* + **抓取策略往往会和关联级别的延迟加载一起使用，优化语句。**

### 抓取策略

#### 抓取策略的概述

* 通过一个对象抓取到关联对象需要发送SQL语句，SQL语句如何发送，发送成什么样格式通过策略进行配置。
  + 通过<set>或者<many-to-one>上通过fetch属性进行设置
  + fetch和这些标签上的lazy如何设置优化发送的SQL语句

#### <set>上的fetch和lazy

* fetch：抓取策略，控制SQL语句格式
  + select ：默认值，发送普通的select语句，查询关联对象
  + join ：发送一条迫切左外连接查询关联对象
  + subselect ：发送一条子查询查询其关联对象
* lazy：延迟加载，控制查询关联对象的时候是否采用延迟
  + true ：默认值，查询关联对象的时候，采用延迟加载
  + false ：查询关联对象的时候，不采用延迟加载
  + extra ：及其懒惰。
* **在实际开发中，一般都采用默认值。如果有特殊的需求，可能需要配置join。**

#### <many-to-one>上的fetch和lazy

* fetch ：抓取策略，控制SQL语句格式。
  + select ：默认值，发送普通的select语句，查询关联对象。
  + join ：发送一条迫切左外连接。
* lazy ：延迟加载，控制查询关联对象的时候是否采用延迟。
  + proxy ：默认值，proxy具体的取值，取决于另一端的<class>上的lazy的值。
  + false ：查询关联对象，不采用延迟。
  + no-proxy ：（不会使用）
* **在实际开发中，一般都采用默认值。如果有特殊的需求，可能需要配置join。**