# Hibernate框架:

# JavaEE内容回顾:

## JavaEE三层结构:

1. 显示层(web层): Servlet, JSP, Struts2, SpringMVC
2. 业务逻辑层(Service层): JavaBean, Spring的IOC(控制反转)
3. 持久层(Dao层): JDBC, Hibernate, Mybatis, Ibtis

Servlet+JSP+JavaBean+JDBC,使用这套架构可以开发市面上的所有应用，但是企业中不会使用（过于底层），企业开发一般使用SSH（Struts2，Spring，Hibernate），SSM（Spring，Mybatis，SpringMVC）。

## MVC思想：

1. Model模型：应用程序的主体部分，模型表示业务数据和业务逻辑。
2. View视图：用户看到并与之交互的界面。
3. Controller控制器：接收用户的输入并调用模型和视图去完成用户的需求。

# Hibernate概述：

框架：框架是用来提高开发效率的，可以理解为一个项目的半成品，封装好了一些功能，我们需要使用这些功能时，调用即可，不需要再手动实现。目前已经掌握的持久层技术：

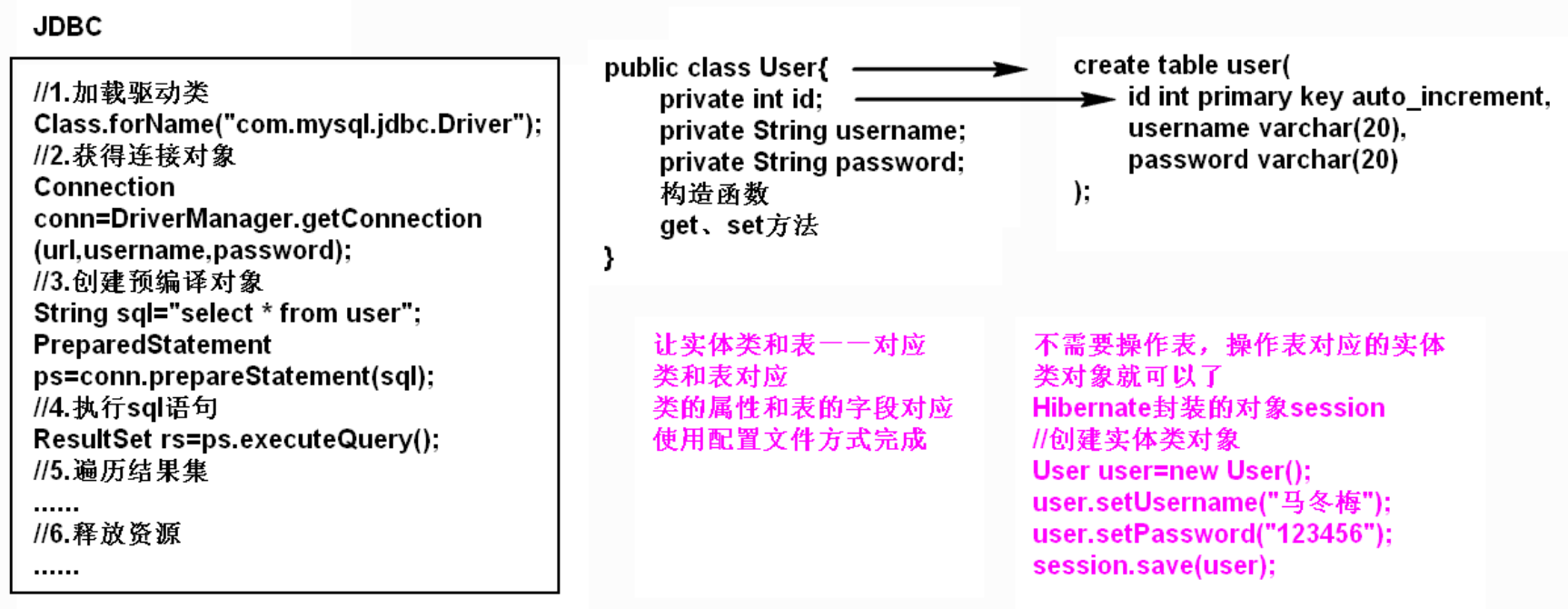
JDBC：它是操作数据库最底层的方式；优势：底层，效率高；劣势：编写代码（查询结果集）过于繁琐；

## 什么是Hibernate：

1. Hibernate框架应用在JavaEE三层结构中的DAO层。
2. Hibernate框架实现对数据库的增删改查操作，Hibernate底层代码就是JDBC。Hibernate对JDBC进行了封装，使用Hibernate的好处：不需要写复杂的JDBC代码了，不需要写sql语句实现功能；
3. Hibernate是开源轻量级框架。

## ORM思想（Object Relational Mapping）：

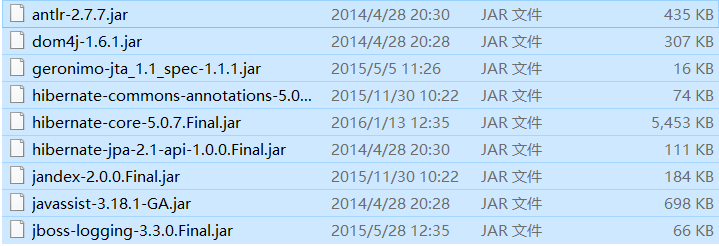
Hibernate使用orm思想对数据库进行增删改查操作；ORM：（对象关系映射）。建立实体类和数据表的对应关系，实现操作实体类对象就相当于操作数据库表。



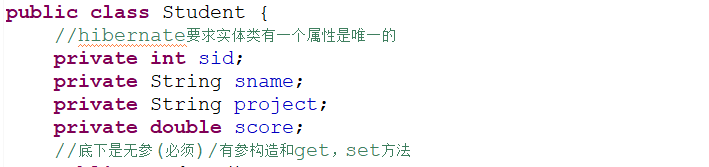
## Hibernate入门

### 搭建Hibernate环境：

第一步：导入导入Hibernate的jar包，位置：hibernate-release-5.0.7.Final\lib\required



第二步：创建实体类

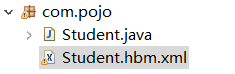


使用hibernate的时候，不需要自己动手创建表，hibernate会帮助我们创建表。

第三步：配置实体类和数据库表的一一对应关系（映射关系）

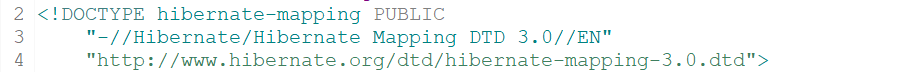
使用配置文件实现映射关系

1. 创建映射配置文件：



映射配置文件的名称和位置没有固定要求，但是建议在实体类所在的包里创建，文件名格式：实体类名.hbm.xml

1. 在配置文件中引入dtd约束：

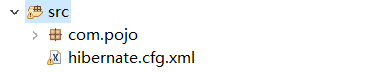


1. 配置映射关系

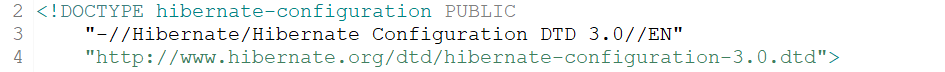


第四步：创建核心配置文件/主配置文件：

1. 主配置文件的名称和位置是固定的：



1. 在配置文件中引入dtd约束：



1. Hibernate操作过程中只会加载主配置文件，其它配置文件不会加载。位置：hibernate-release-5.0.7.Final\project\etc\hibernate.properties



### 实现添加操作

第一步：加载hibernate主配置文件

第二步：创建sessionFactory对象

第三步：使用sessionFactory创建session对象

第四步：开启事务

第五步：具体的逻辑（增删改查操作）

第六步：提交事务

第七步：关闭资源



### Hibernate核心API：

1. 读取src下名称为hibernate.cfg.xml的配置文件，然后创建对象，把配置文件放到对象里面（加载主配置文件）



1. SessionFactory（重点）

使用Configuration对象创建SessionFactory对象。

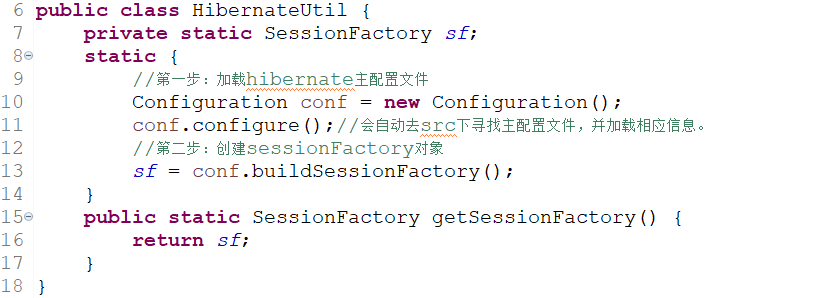
在创建过程中：

1. 根据主配置文件中的数据库配置，映射文件，在数据库里面根据映射关系创建表



1. 创建SessionFactory的过程中，特别耗资源。在hibernate操作过程中，建议一个项目一般创建一个SessionFactory对象即可。
2. 解决办法

创建一个工具类，写静态代码块实现，静态代码块在加载的时候执行，并且执行一次。





1. Session（重点）
2. Session类似于jdbc中的Connection
3. 调用Session里面不同的方法实现增删改查操作
   1. 添加save方法
   2. 修改update方法
   3. 删除delete方法
   4. 根据id查询get方法
4. Session对象是单线程对象
   1. Session对象不能共用，只能自己使用。
5. Transaction
6. 事务对象
7. 事务的提交和回滚方法

提交：transaction.commit();

回滚：transaction.rollback();

1. 事务的概念及特性：

把多个相关的操作捆绑成一个整体，要么都成功，要么都失败。

* + 1. 原子性：原子性是指事务包含的所有操作要么全部成功全部应用，要么全部失败回滚。成功则必须要完全应用到数据库。失败则不能对数据库产生影响。
    2. 隔离性：当多个用户并发访问数据库时，数据库为每一个用户开启的事务不被其它事务的操作所干扰，多个并发事务之间要相互隔离。
    3. 一致性：一致性是指一个事务执行之前和执行之后都必须处于一致性状态。
    4. 持久性：持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的。

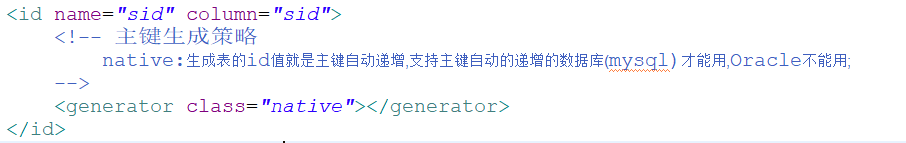
# 实体类编写规则:

1. 实体类里面的属性私有化
2. 私有属性使用公开的get和set方法操作
3. 要求实体类有属性作为唯一值
4. 实体类属性不建议使用基本数据类型，应使用基本数据类型对应的包装类（为了能准确区别0和null）；八种基本数据类型对应的包装类：int 🡪 Integer；char 🡪 Character; 其它都是首字母大写；

# Hibernate主键生成策略：

Hibernate要求实体类里面有一个属性作为唯一值，对应数据库表的主键，主键可以有不同的生成策略；

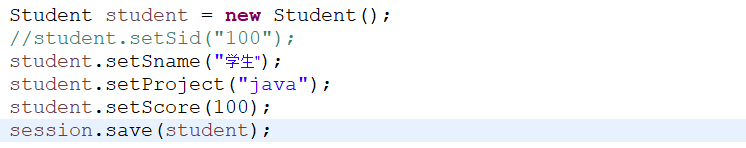
1. increment：由hibernate自动以递增的方式生成唯一标识符，每次递增1；只能单线程使用；
2. identity：主键自增，由数据库负责生成主键（要求数据库支持主键自动递增，mysql可以，oracle不可以自动递增）；
3. sequence：hibernate根据底层数据库序列生成标识符，条件是数据库支持序列（Oracle支持，mysql不支持）；
4. uuid：生成随机字符串作为主键，主键类型为字符串；
5. hilo：高低位算法，适用于既不支持自增，也不支持序列的数据库。
6. native：根据数据库来创建不同的生成策略（根据具体情况选择identity/seuence/hilo）；
7. assigned：由外部程序负责生成，hibernate不负责维护主键生成；设置为uuid和native时，即使设置id，也不会用其设置的id值；只有assigned，设置的id值才会起作用；



# 实体类操作：

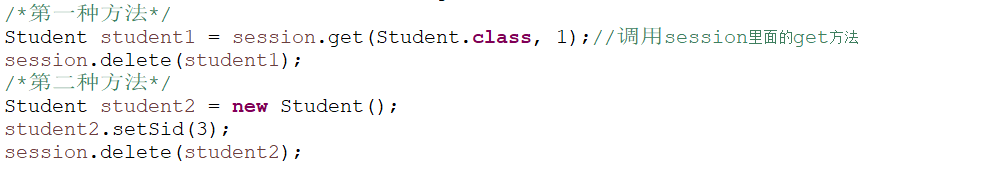
## **5.1 添加**

调用session里面的save方法。等效于saveOrUpdate()方法，save(持久态对象)也是更新。



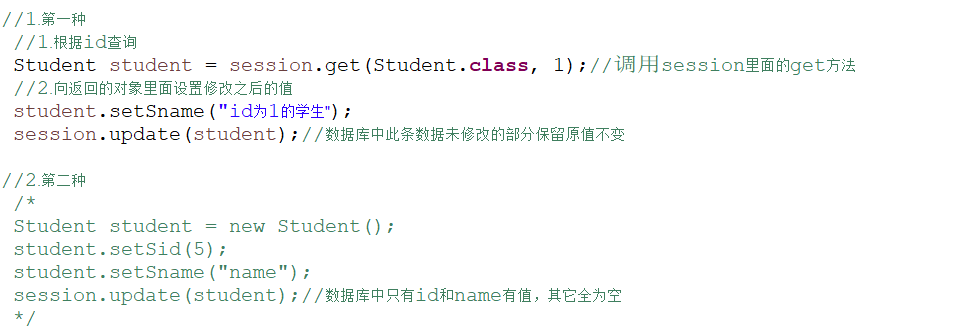
## **5.2 删除**

调用session里面的delete方法。推荐第一种；



## **5.3 修改**

调用session里面的update方法（根据id查询，向返回的对象里面设置修改之后的值）。执行过程：到student对象里面找到id值，然后根据id进行修改。强烈推荐第一种；

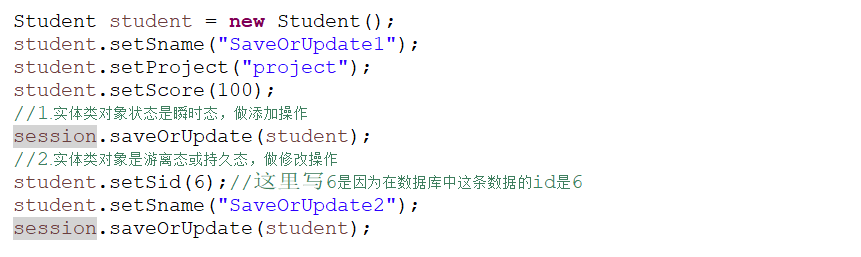


## **5.4 根据id查**

调用session里面的get方法。



## 5.5 saveOrUpdate方法



实现添加或修改操作；

# 实体类对象状态（概念）：

1. 瞬时态：对象里面没有id，对象与session没有关联；
2. 持久态：对象里面有id值，对象与session有关联；
3. 托管态（游离态）：对象有id值，对象与session没有关联；

## 状态转换：

瞬时态到持久态：执行save操作；

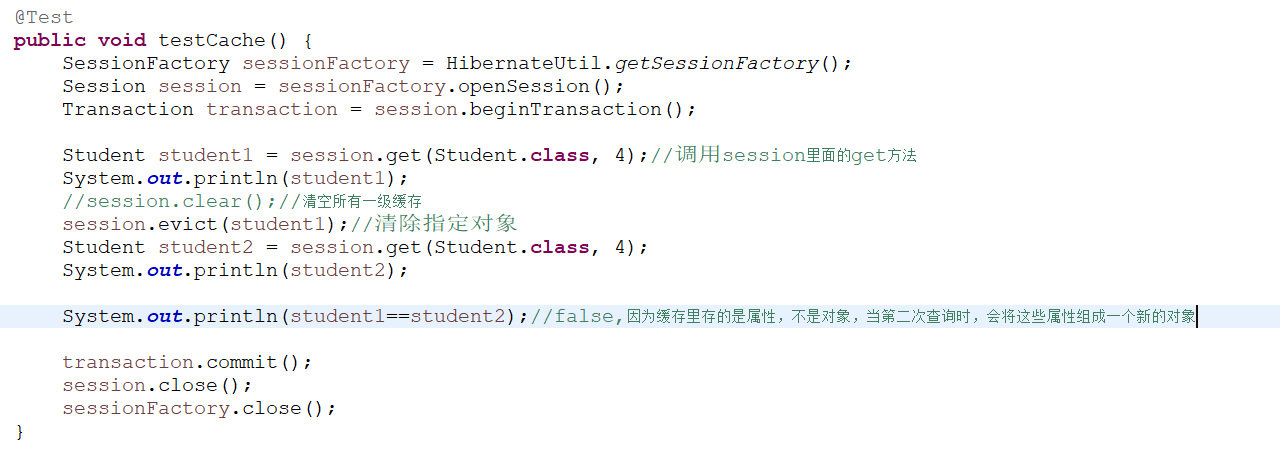
持久态到瞬时态：执行delete操作；

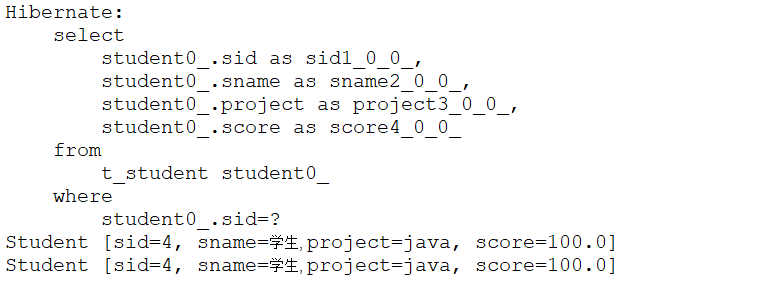
持久态到游离态：session关闭；

游离态到持久态：执行update操作；

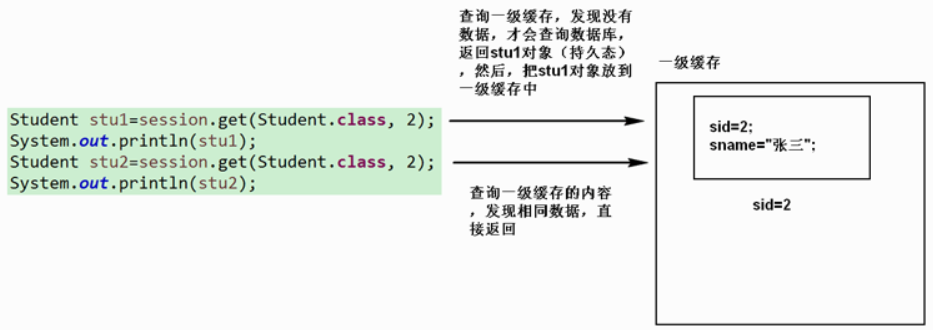
# Hibernate的一级缓存：

1. 什么是缓存：
   1. 数据存储到数据库中，数据库本身是文件系统，使用流的方式操作文件效率不是很高；
   2. 把数据放到内存里面，不需要使用流的方式，可以直接读取内存中的数据；
   3. 把数据存到内存中能提高读取效率；
2. hibernate缓存：
   1. hibernate中提供了很多优化方式，hibernate的缓存就是一种优化方式；
   2. hibernate缓存的特点：一级缓存，二级缓存；
3. 一级缓存：
   1. hibernate的一级缓存默认是打开的；
   2. hibernate的一级缓存只在session范围内有效（从session创建到session关闭）；
   3. hibernate的一级缓存中存储的数据必须是持久态数据；
4. 二级缓存：
   1. 默认是关闭的，需要配置开启；
   2. 二级缓存的使用范围是sessionFactory范围；
5. 验证一级缓存：
   1. 第一次根据某一个在数据库中存在的id查询，第二次再根据此id查询；

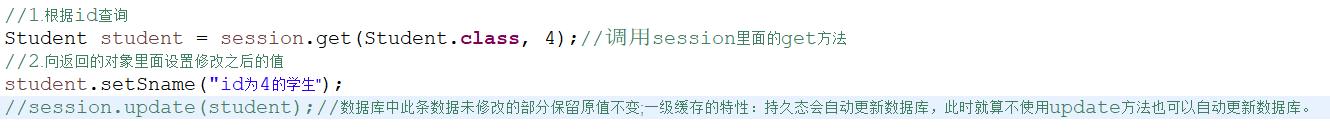




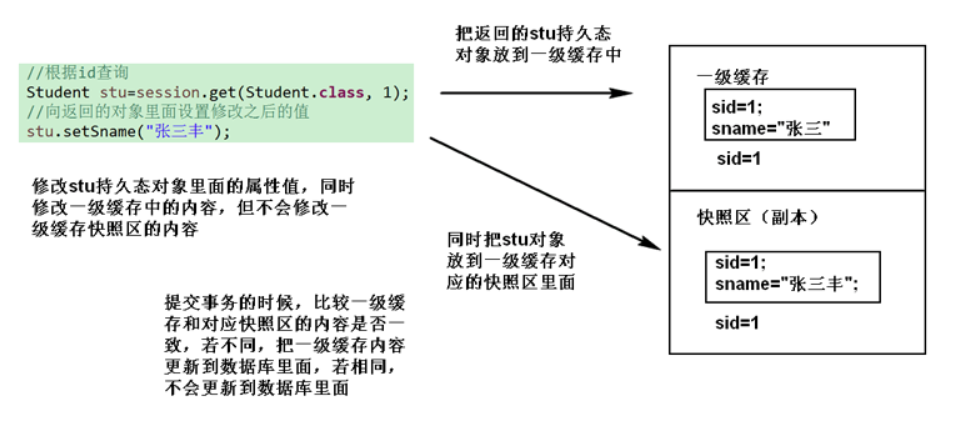
* 1. 原理：第一次向数据库发送sql语句，第二次没有发送sql语句，查询的是一级缓存的内容。
  2. hibernate一级缓存的执行过程：



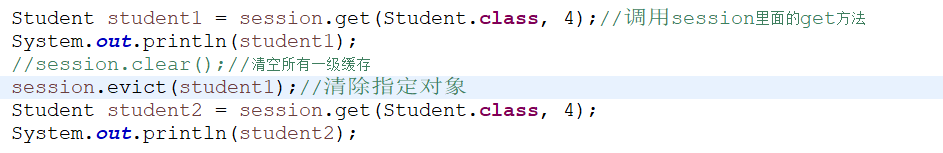
1. 一级缓存的特性：
   1. 持久态会自动更新数据库



* 1. 执行过程（了解）



1. 清空所有一级缓存和清除指定对象

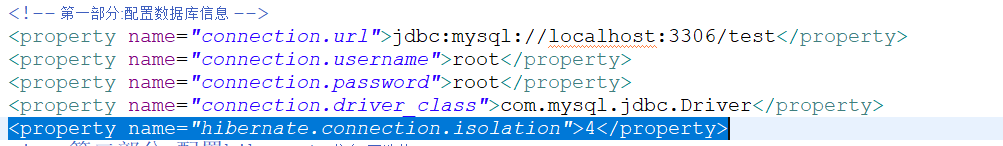


# Hibernate事务操作：

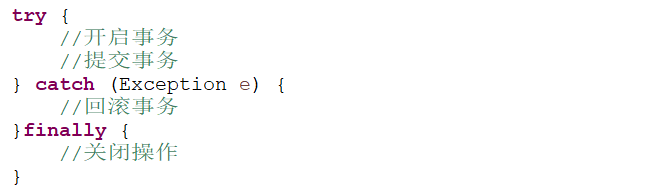
1. 事务的相关概念：
   1. 什么是事务
   2. 事务的特性
   3. 不考虑隔离性产生的问题

脏读/不可重复读/虚读（幻读）

* 1. 设置事务的隔离级别：mysql默认隔离级别repeatable read；

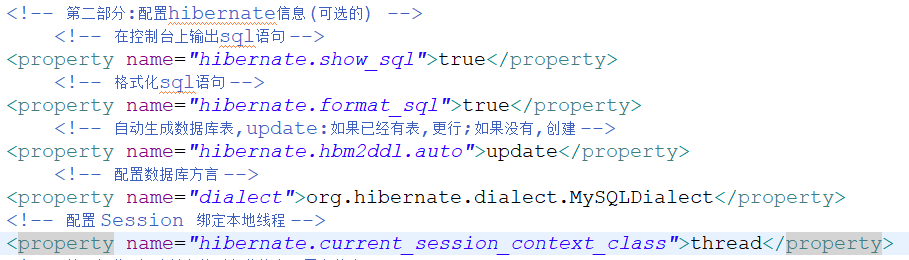


1. hibernate事务代码规范写法：
   1. 代码结构

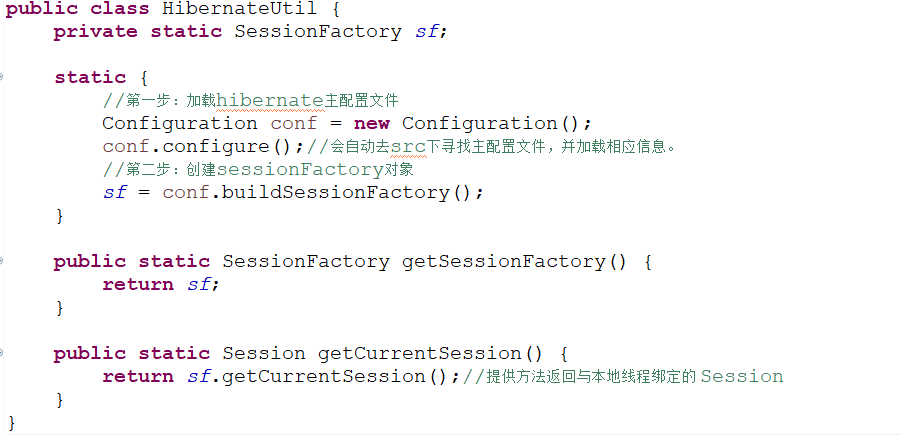


# Hibernate绑定Session：

1. Session 类似于 JDBC 的连接 Connection；
2. Session 对象是单线程对象，只能自己使用，不能共用；将 Session 与本地线程绑定，保证 Session 对象绝对是一个单线程对象；
3. Hibernate 帮助我们实现了 Session 与本地线程绑定（底层是 ThreadLocal）；
4. 获取与本地线程绑定的 Session
   1. 在 Hibernate 核心配置文件中进行配置：



* 1. 调用SessionFactory 的 getCurrentSession() 方法获取 Session；使用本地线程绑定的session，即使是查询，也需要开启事务，否则报错。



1. 与本地线程绑定的Session 最后不用手动关闭，当线程执行结束后，Session 将会自动关闭。如果手动关闭，将报错



# Hibernate的api使用：

## Query对象

1. 使用query对象，不需要写sql语句，但是需要写hql（hibernate query language）语句；hql是hibernate提供的查询语言，hql语句和普通sql语句很相似；
2. hql和sql语句的区别：
   1. 使用sql操作表和表的字段；
   2. 使用hql操作实体类和属性；
3. 查询所有的hql语句：from实体类名称；
4. Query对象的使用：
   1. 创建Query对象；
   2. 调用Query对象里的方法，得到结果；



## Criteria对象

1. 使用这个对象的时候，不需要写语句，直接调用方法实现；
2. 实现过程：
   1. 创建Criteria对象
   2. 调用Criteria对象里面的方法得到结果



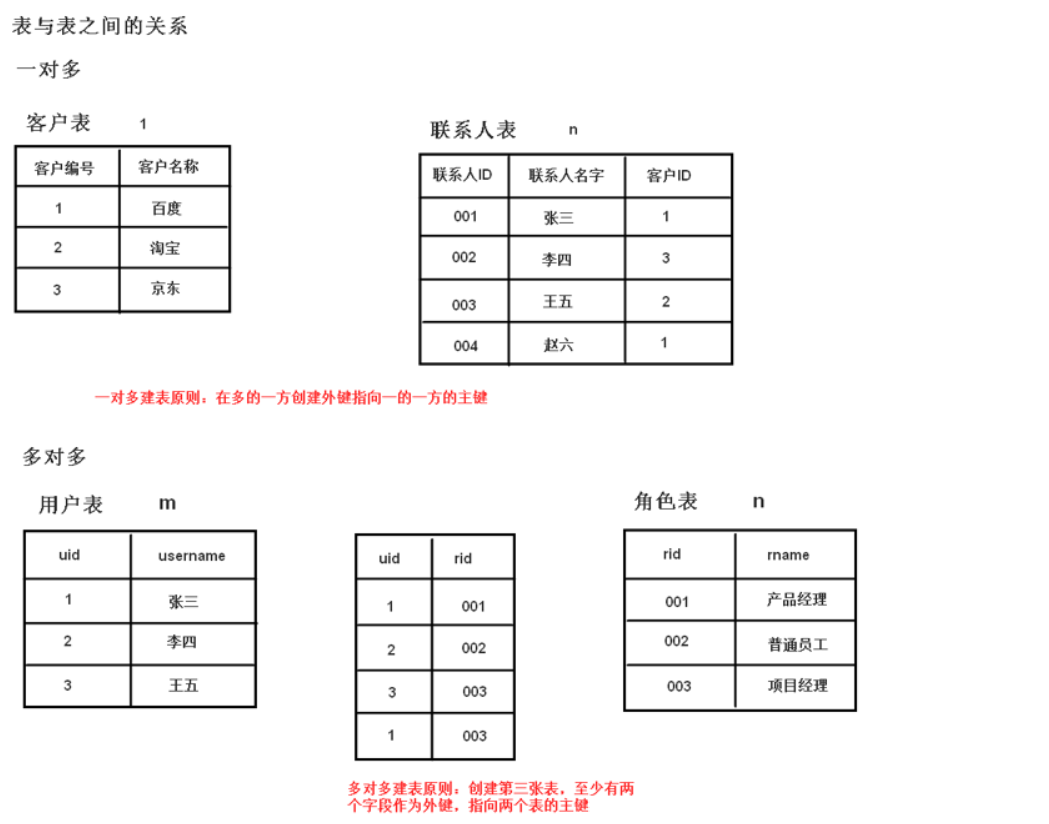
## SQLQuery对象

1. 使用SQLQuery对象，调用底层sql实现
2. 实现过程
   1. 创建对象
   2. 调用对象的方法得到结果



# 表与表之间关系回顾

1. 一对多
   1. 一个部门有多个员工，一个员工只能属于一个部门；
   2. 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；
2. 多对多
   1. 订单和商品的关系，一个订单里面可以有多个商品，一个商品可以属于多个订单；
   2. 一个用户可以有多个角色，一个角色可以有多个用户；
3. 一对一：一夫一妻，一个人只有一个身份证号等；



# Hibernate的一对多操作（重点）

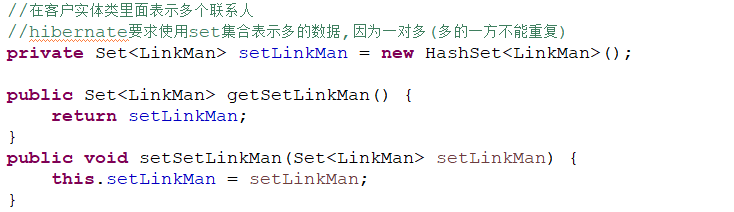
此部分以Customer(1)和LinkMan(多)为例。

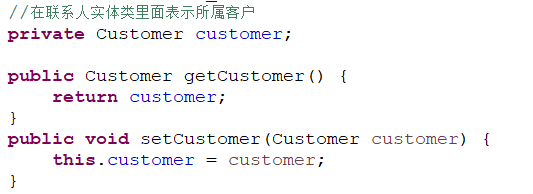
## 12.1 一对多映射配置

以客户和联系人为例，客户对联系人（一对多）；

第一步：创建两个实体类，客户和联系人；

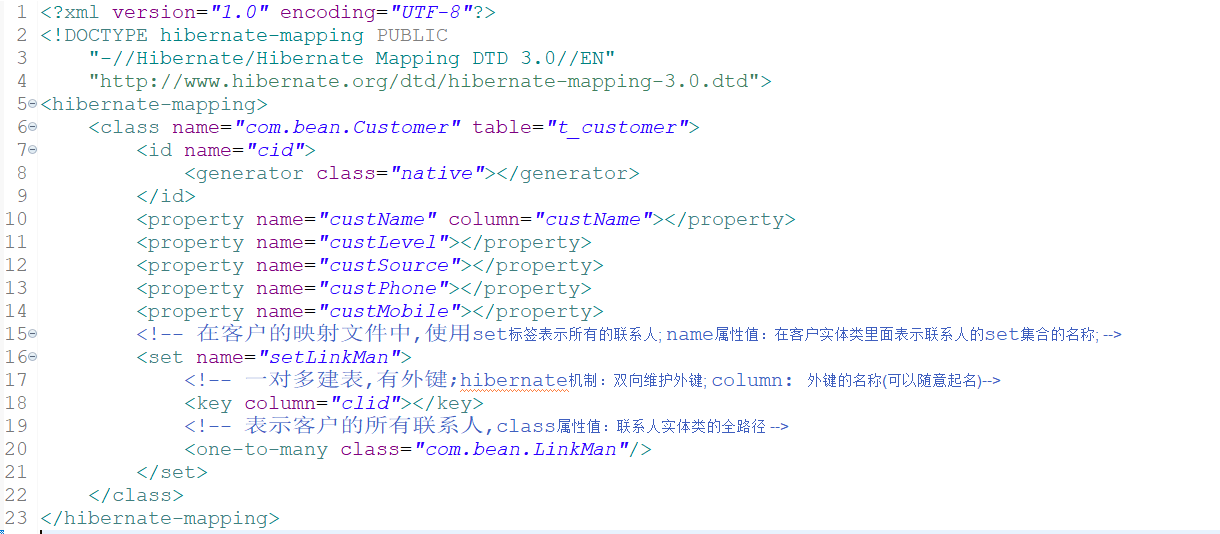
第二步：让两个实体之间相互表示：在客户实体类里面表示多个联系人（一个客户可以对应多个联系人）；在联系人实体类里面表示所属客户（一个联系人只能对应属于一个客户）；





第三步：配置映射关系

* + 1. 一个实体对应一个映射文件
    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（重点）
    4. 在客户的映射文件中，表示所有的联系人



* + 1. 在联系人的映射文件中表示所属客户



第四步：创建核心配置文件，将映射文件引入到主配置文件中



## 12.2一对多级联保存

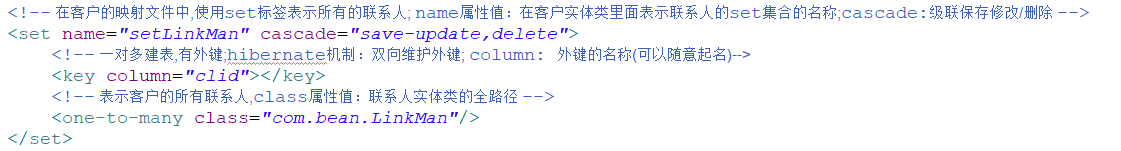
添加一个客户，为这个客户添加多个联系人

* + - 1. 复杂写法：



* + - 1. 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



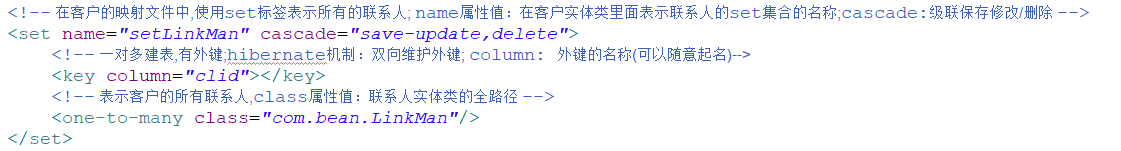
第二步：创建客户和联系人对象，最终只需要保存客户就可以了



## 12.3一对多级联删除

删除某一个客户，这个客户里面的所有联系人也删除；

第一步：在客户映射文件中进行配置



第二步：在代码中直接删除客户即可（底层执行过程：根据id查询对象；根据外键id查询联系人；把联系人的外键设置为null；删除联系人和客户）



## 12.4一对多的级联修改

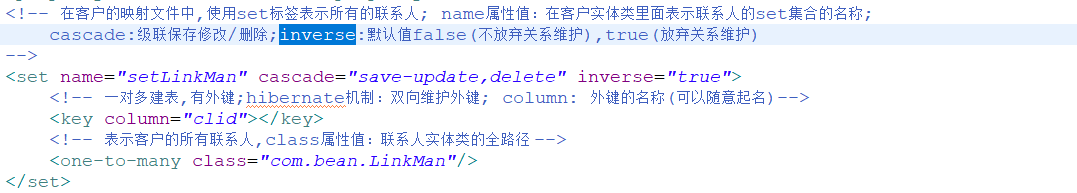


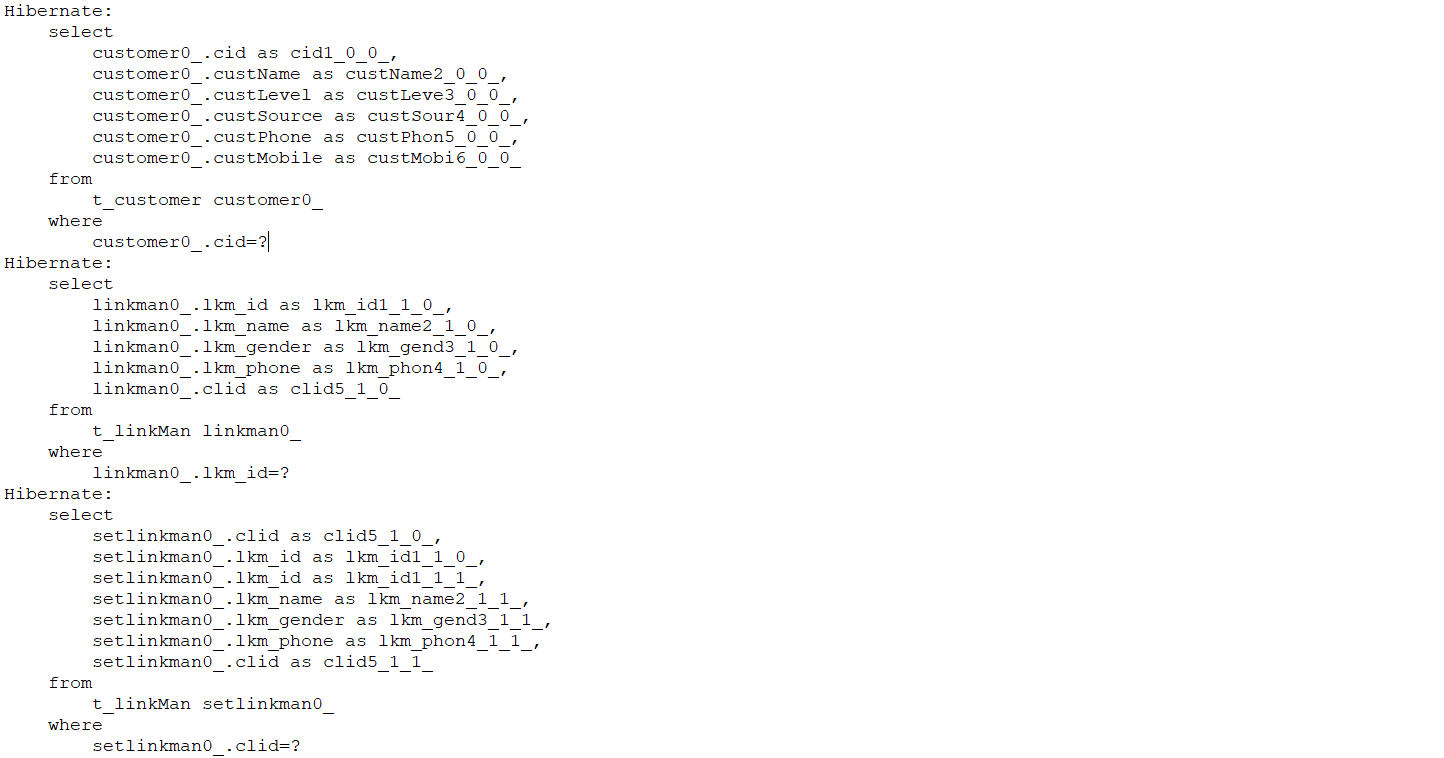
## 12.5 inverse属性

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，修改客户的时候，修改一次外键，修改联系人的时候也修改了一次外键。这会造成效率问题

解决方式：让其中的一方不维护外键，让一的一方放弃外键，具体实现如下：

在放弃关系维护的映射文件中进行配置，在set标签上使用inverse属性。







# Hibernate多对多操作

此部分以User(多)和Role(多)为例。

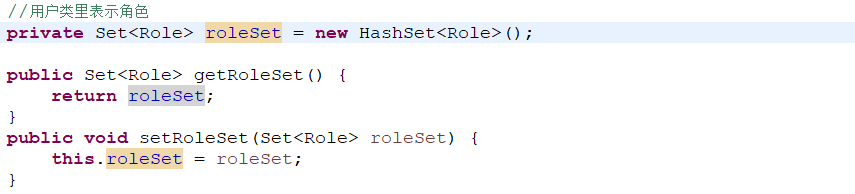
## 13.1 多对多映射配置

以用户和角色为例

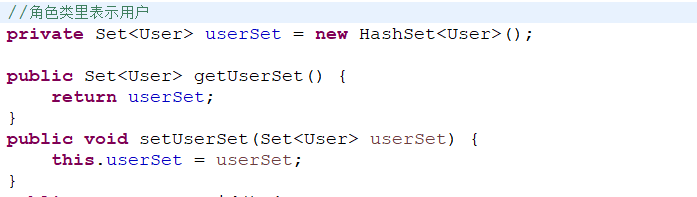
第一步：创建两个实体类，用户和角色

第二部：让两个实体类之间相互表示

1. 在用户里面表示所有角色，使用set集合



1. 在角色里面表示所有用户，使用set集合



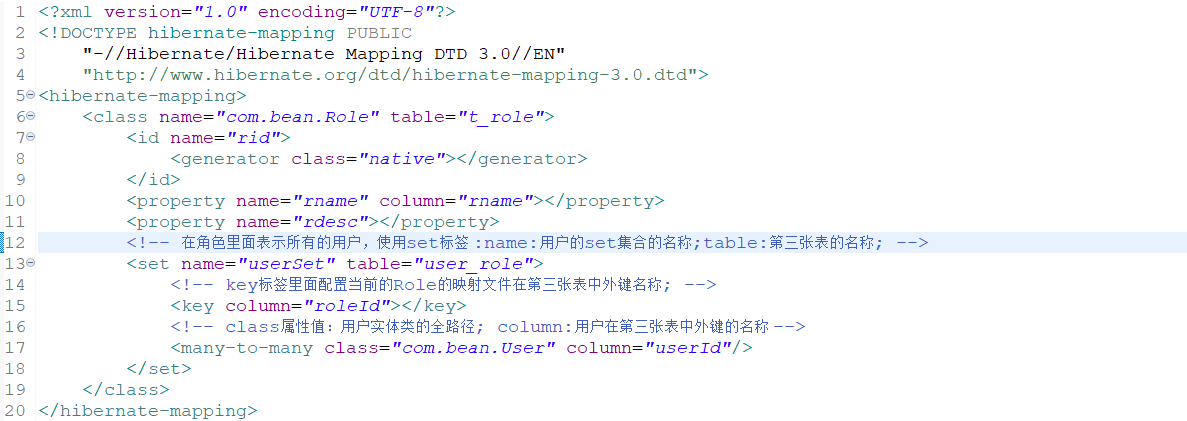
第三步：配置映射关系

1. 基本配置
2. 配置多对多关系：

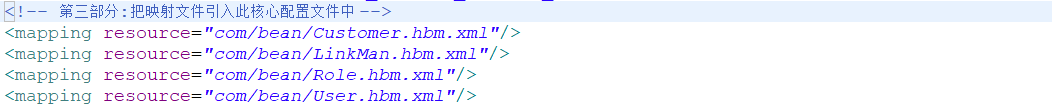
在用户里面表示所有的角色，使用set标签；



在角色里面表示用户，使用set标签。



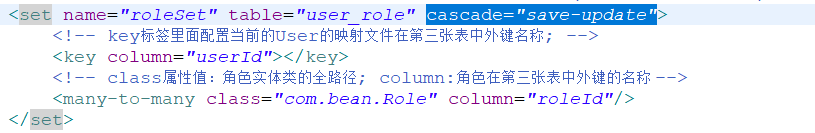
第四步：在著配置文件里引入映射文件



## 13.2 多对多级联保存

根据用户保存角色

第一步：在用户配置文件中的set标签进行配置（在哪里配置，就保存哪一个就ok）

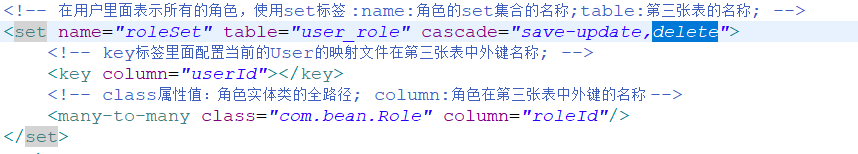


第二步：写代码实现功能，创建用户和角色对象，把角色放到用户里面，最终保存用户就可以了。



## 13.3 多对多级联删除（了解）

第一步：在配置文件中的set标签进行配置



第二步：删除用户：



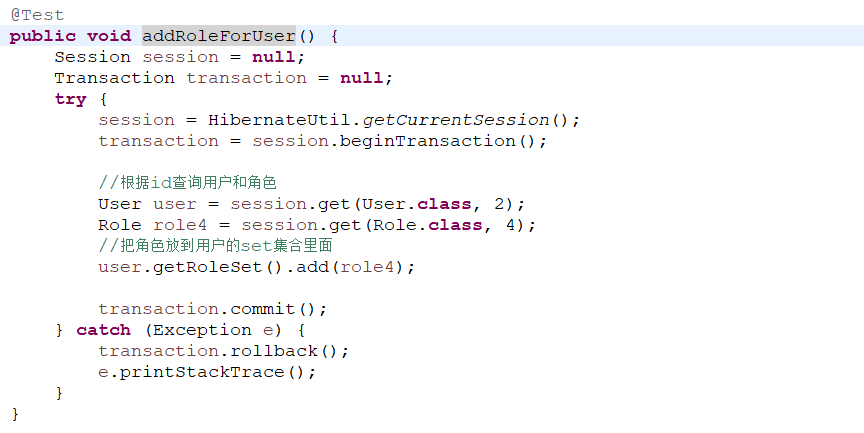
## 13.4 维护第三张表的关系

因为用户和角色是多对多关系，维护关系通过第三张表维护

* + - 1. 为用户分配角色

第一步：根据id查询用户和角色

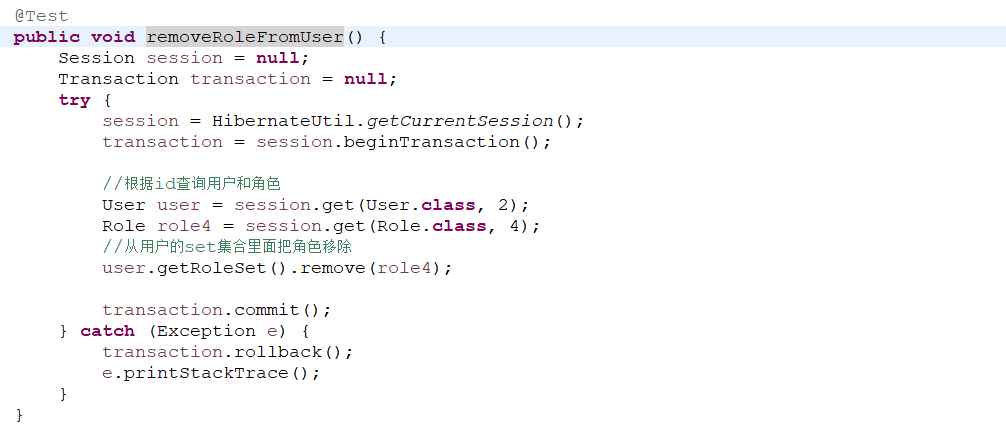
第二步：把角色放到用户的set集合里面



* + - 1. 删除用户对应的角色

第一步：根据id查询用户和角色

第二步：从用户的set集合里面把角色移除

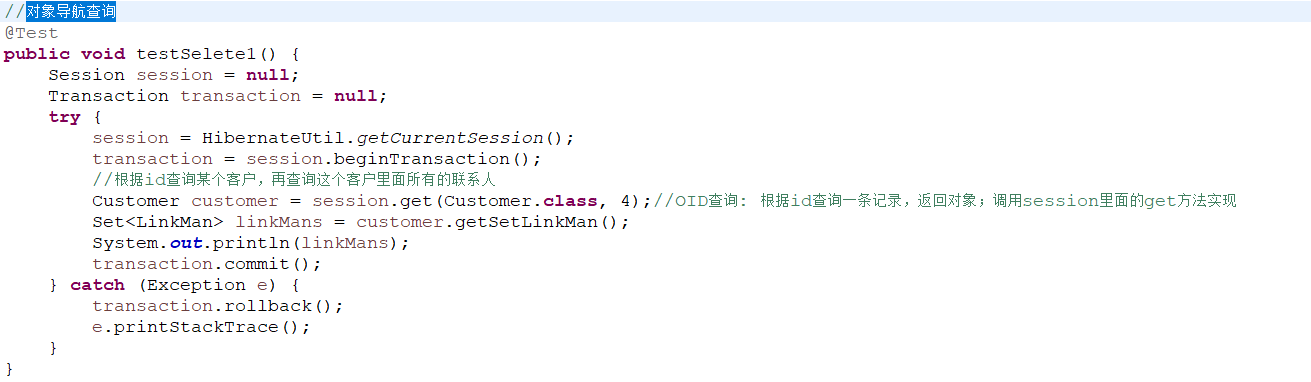


# 14.Hibernate查询方式

## 14.1对象导航查询

根据id查询某个客户，再查询这个客户里面所有的联系人；

查询某个客户里面所有的联系人的过程，使用对象导航查询实现：



## 14.2 OID查询

根据id查询一条记录，返回对象；调用session里面的get方法实现。

## 14.3 HQL查询

1. Query对象，写hql语句，实现查询；HIbernate提供的一种查询语言，hql语句（操作实体类和属性）和sql语句（操作数据库表和字段）很相似；

2.使用hql查询操作的时候，使用Query对象

（1）创建Query对象，写hql语句

（2）调用query对象里面的方法得到结果

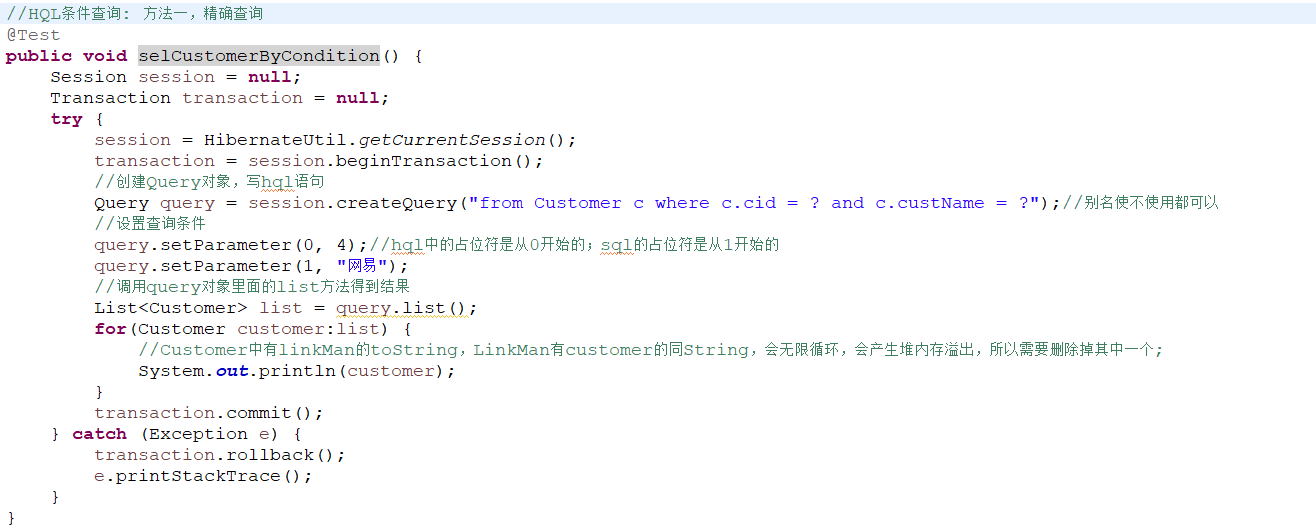
3. 常用的hql语句：

（1）查询所有：from 实体类名



1. 条件查询：

方法一：from 实体类名 where 实体类属性名称=？ and 实体类属性名称=？

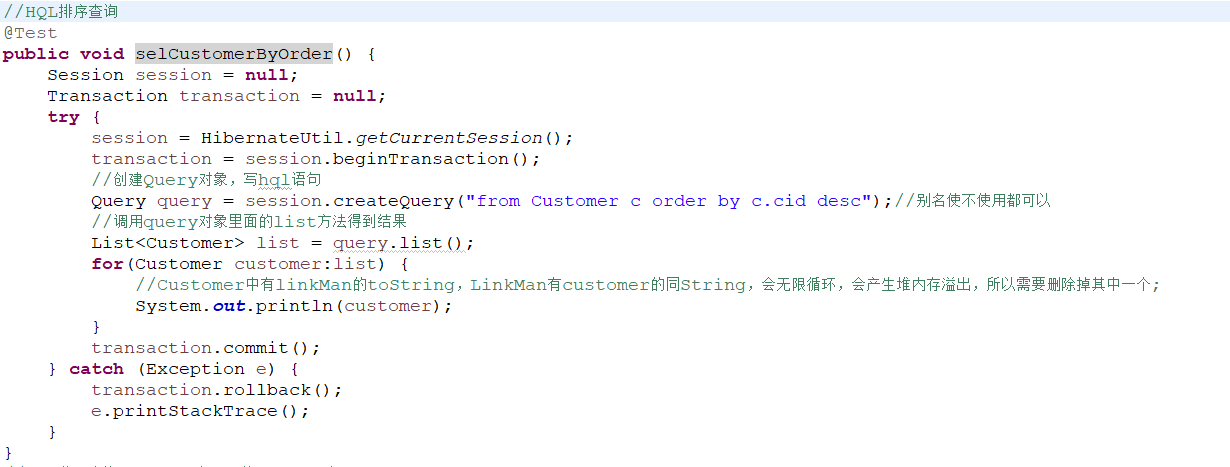




方法二（模糊查询）：from 实体类名 where 实体类属性名称 like ？



1. 排序查询：from 实体类名 order by 实体类属性名称 asc/desc



1. 分页查询：



1. 投影查询：查询一个或多个属性，但不查询全部

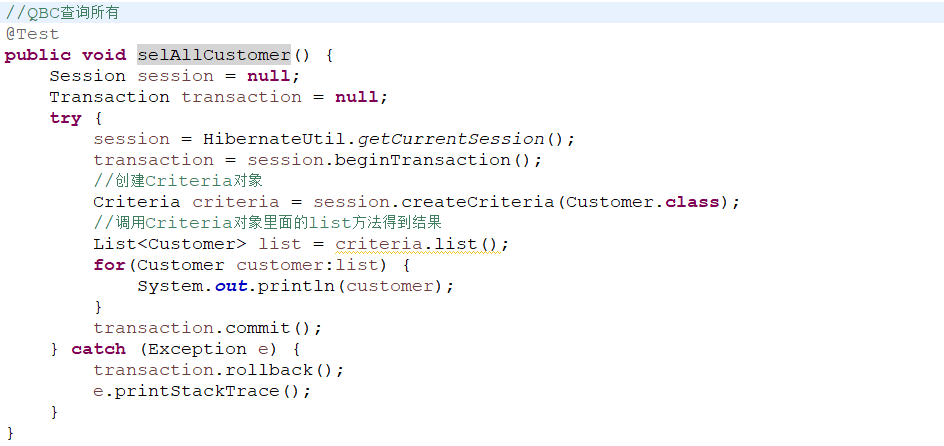


1. 聚集函数的使用：



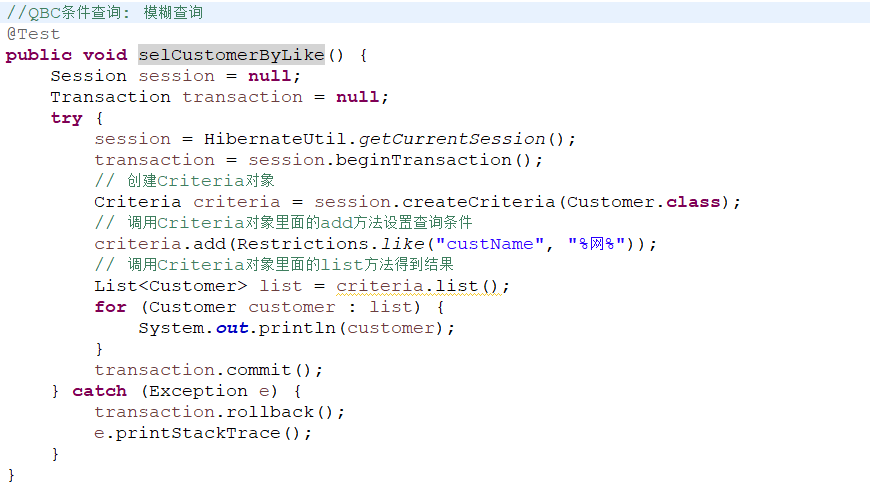
## 14.4 QBC查询

1. HQL查询需要写hql语句实现，QBC查询不需要写语句了，使用方法实现；
2. 使用QBC查询的时候，操作实体类和属性；
3. 使用QBC查询的时候，使用Criteria实现；
4. 查询所有：



1. 条件查询：

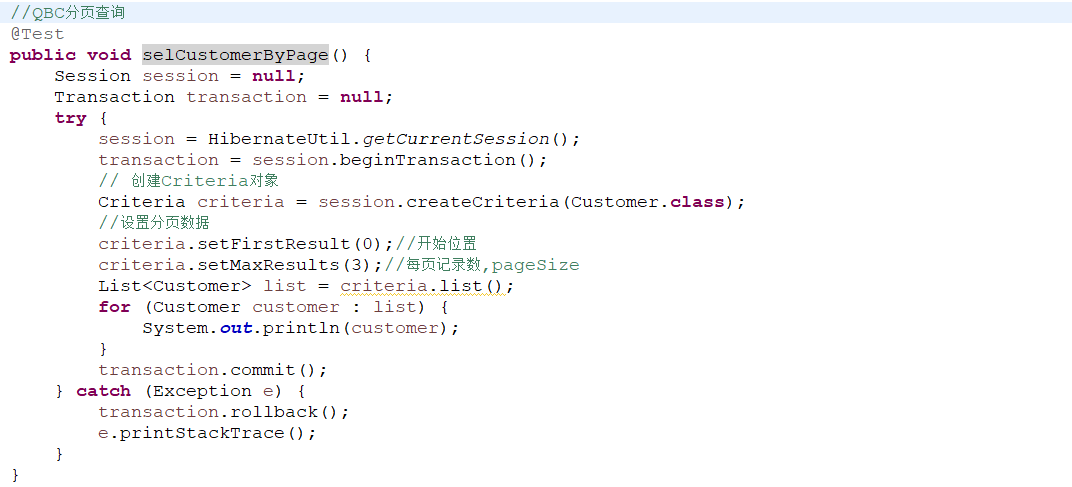




1. 排序查询：



1. 分页查询：



1. 统计查询：



1. 离线查询：



## 14.5 本地SQL查询

1. SQLQuery对象，使用普通sql查询

## 14.6 HQL多表查询

### 1. Mysql的多表查询（参考MySQL.sql的笔记）

（1）内连接（等值连接/非等值连接/自连接）

（2）外连接（左外连接/右外连接/）（mysql不支持全外连接）

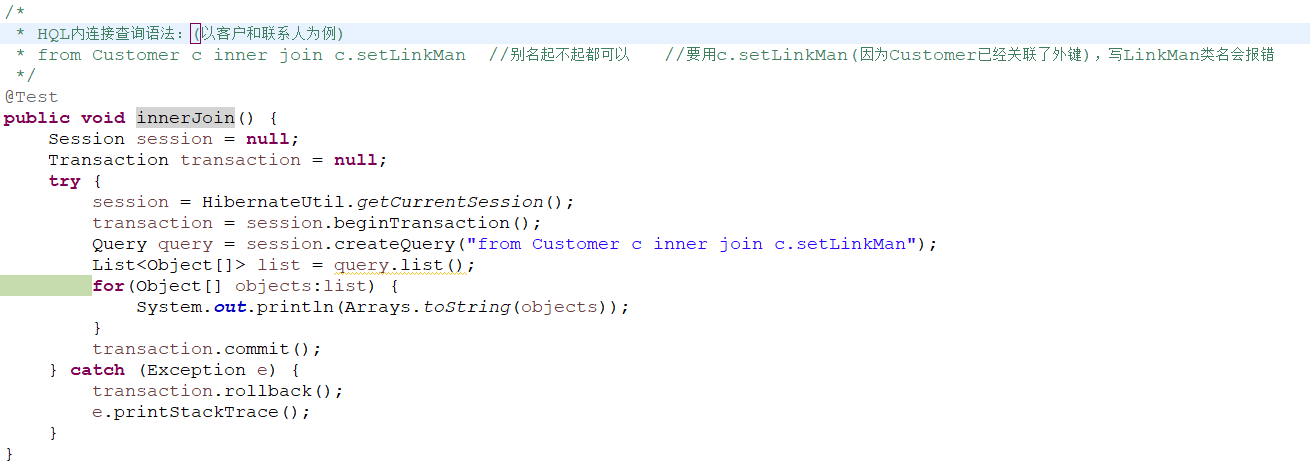
### 2. HQL实现多表查询

（1）内连接：返回list集合，list里面每部分是数组的形式



（用debug模式，断点运行到list时，可查看list的类型）

这里因为Customer4有两个联系人，所以会输出两遍（匹配两遍，将同一个对象封装两遍到list集合中）

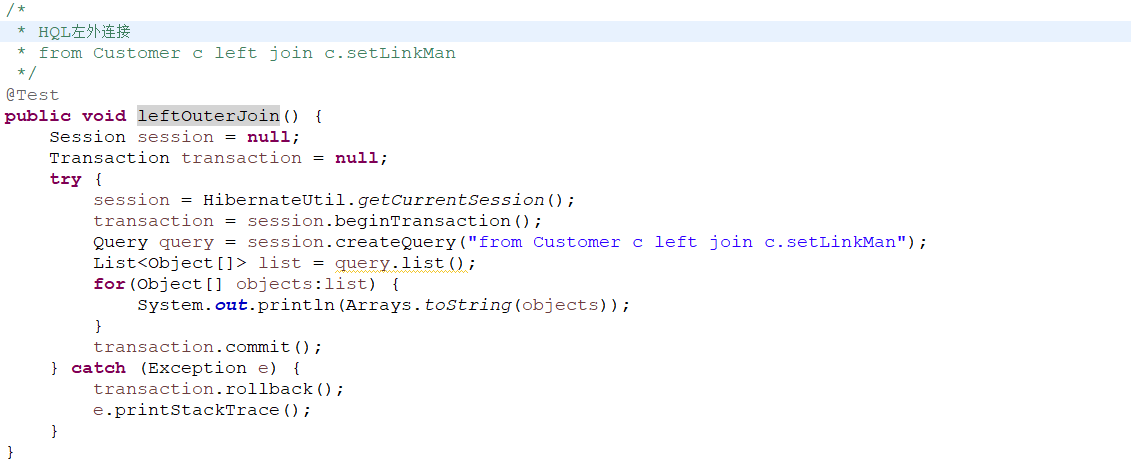


（2）迫切内连接：迫切内连接和内连接底层实现是一样的

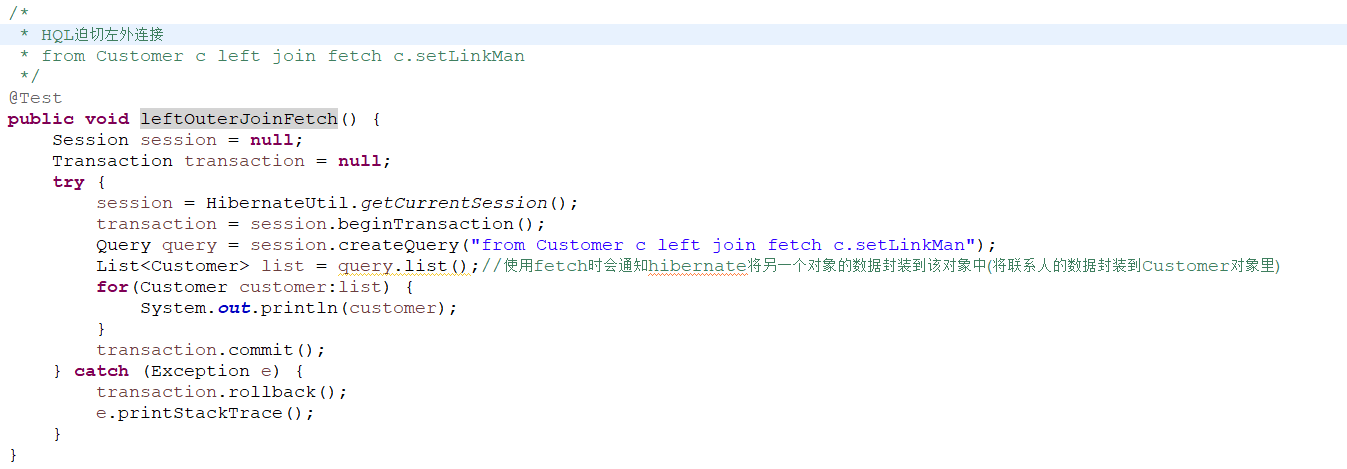
区别：内连接返回list每部分是数组，迫切内连接返回list每部分是对象。



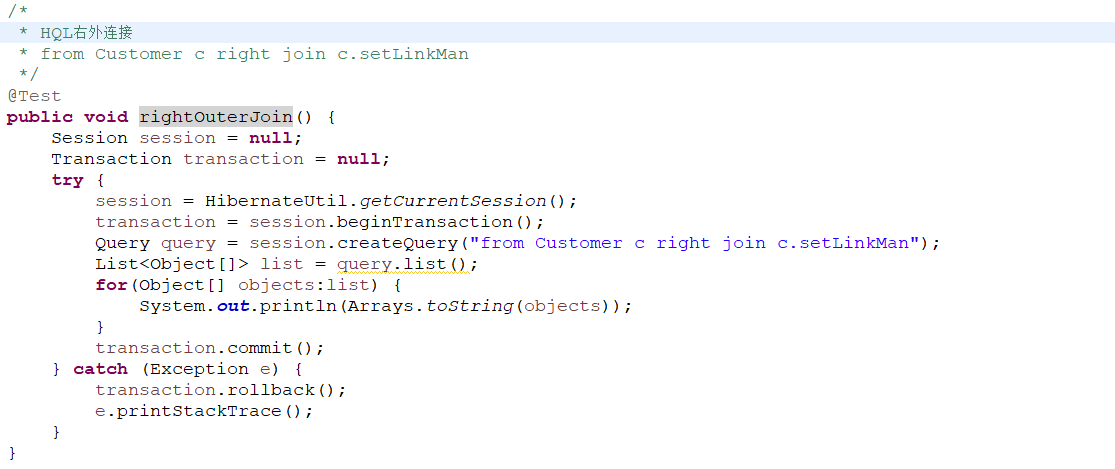
1. 左外连接：



（4）迫切左外连接：

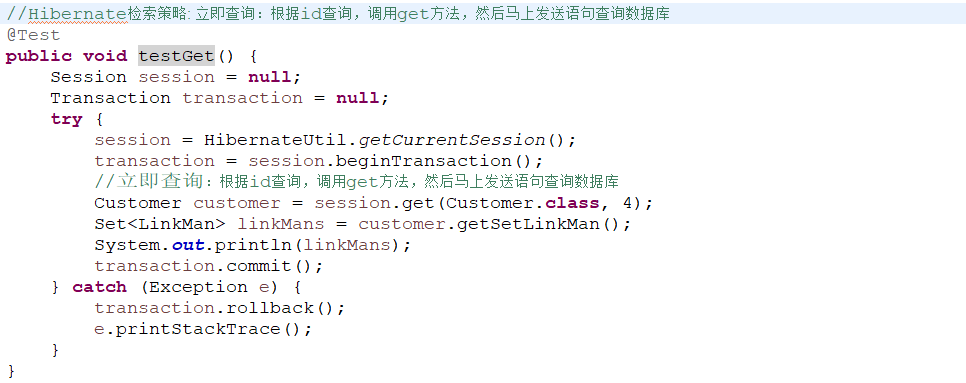


（5）右外连接：

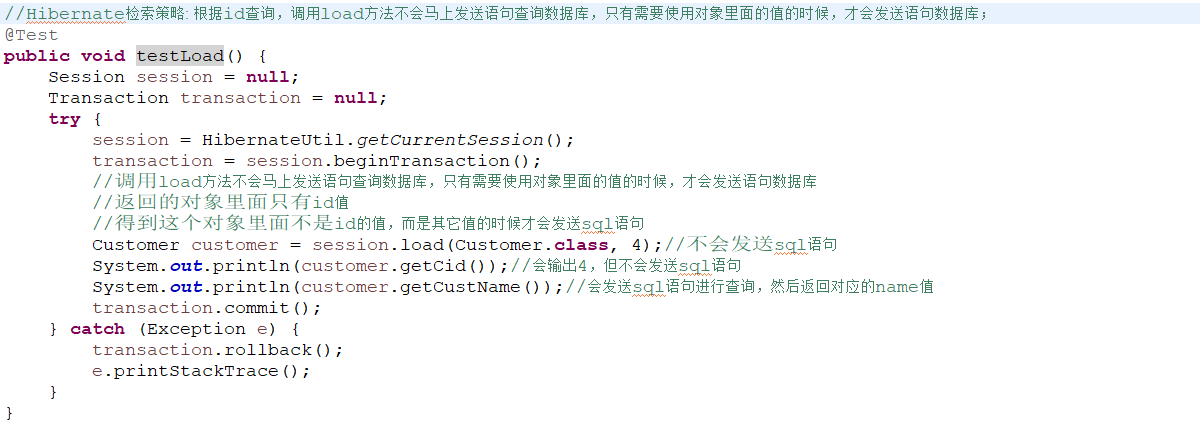


# Hibernate检索策略

1. 检索策略概念：hibernate的检索策略分为两类
2. 立即查询：根据id查询，调用get方法，然后马上发送语句查询数据库；

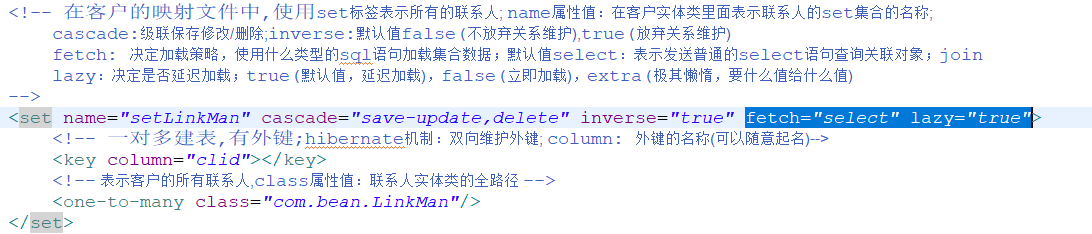


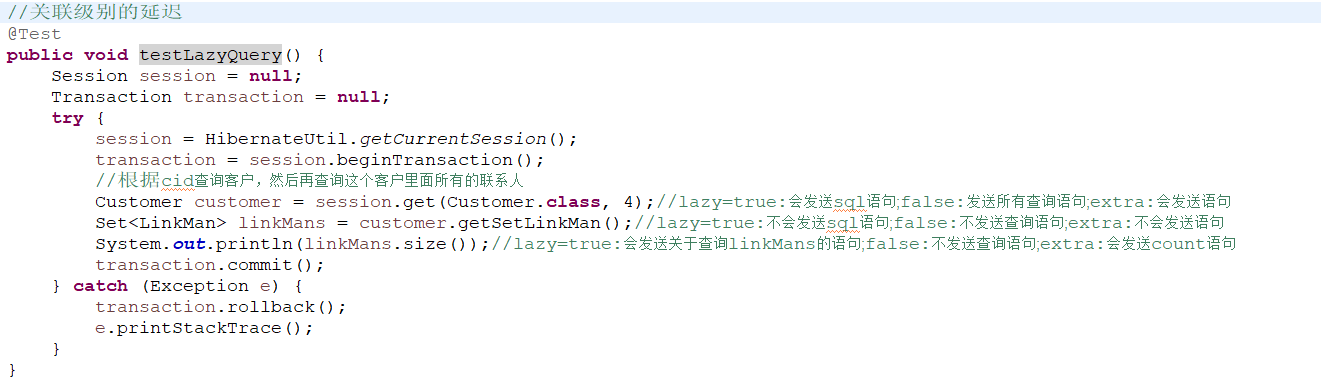
1. 延迟查询：根据id查询，除了get方法之外还有load方法，调用load方法不会马上发送语句查询数据库，只有需要使用对象里面的值的时候，才会发送语句数据库；



1. 延迟查询分为两类：
2. 类级别延迟：根据id查询返回实体类对象，调用load方法，不会马上发送语句查询数据库。
3. 关联级别查询：查询某个客户，再查询这个客户的所有联系人，查询客户的所有联系人的过程是否需要延迟。不论是get还是load获取，关联级别都有延迟。
4. 在映射文件中通过配置实现关联级别延迟操作的控制：

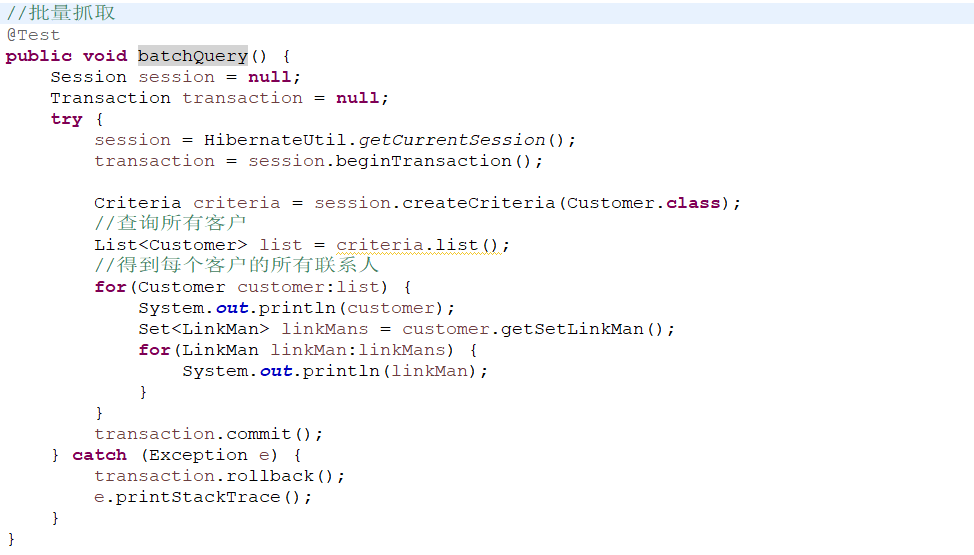
根据客户得到所有联系人，在客户的映射文件中配置





# 15.批量抓取

1. 查询所有客户，返回list集合，遍历list集合，获取每个客户，然后再获取每个客户的所有联系人；



2. 使用批量抓取可以减少sql语句的发送，提高检索效率

