## Hibernate课程安排

1. Hibernate入门和基本操作
2. Hibernate概念和api使用
3. Hibernate配置一对多和多对多
4. Hibernate查询操作

## javaWeb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、Ibatis

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：是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：是用户看到并与之交互的界面
3. Controller 控制器：接受用户的输入并调用模型和视图去完成用户的需求

## Hibernate概述：

## 什么是框架？

框架是用来提高开发效率的

框架可以理解为是一个项目的半成品，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

我们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

弊端：编写代码时过于繁琐、查询封装结果集写吐了

## 什么是Hibernate框架？（重点）

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开源轻量级框架

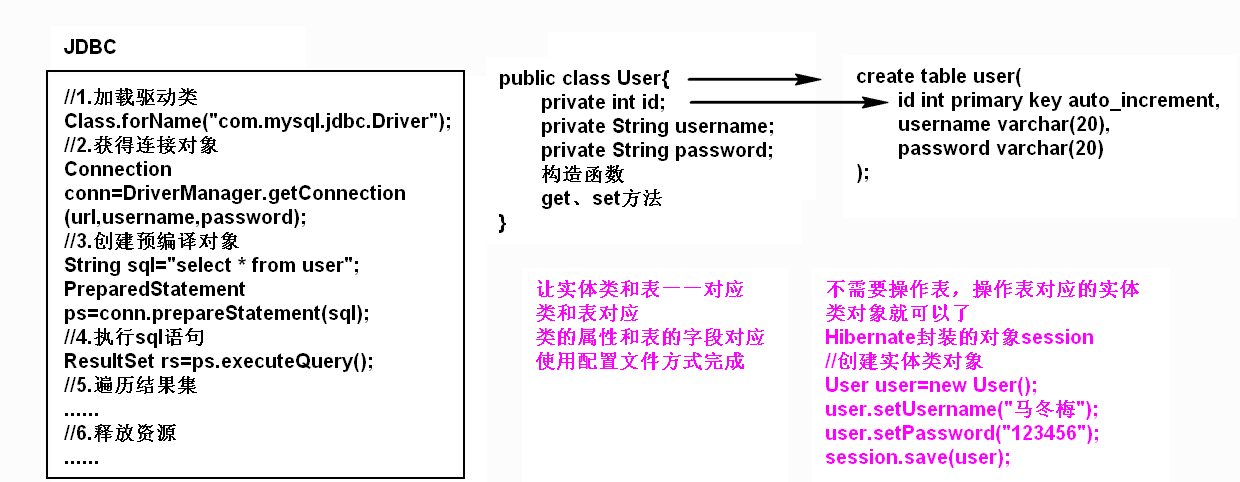
## 什么是orm思想？

Hibernate使用orm思想对数据库进行增删改查操作

orm:Object relational mapping 对象关系映射

就是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

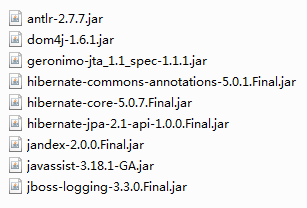


## Hibernate入门：

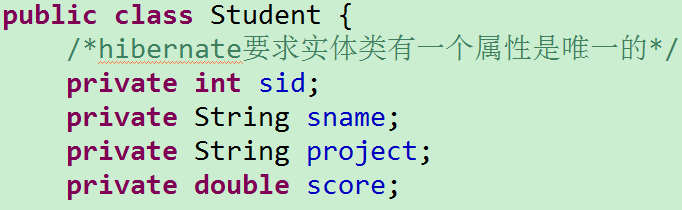
## 搭建Hibernate环境

**第一步：导入hibernate的jar包**

位置：hibernate-release-5.0.7.Final\lib\required



**第二步：创建实体类**



1. 使用hibernate的时候，不需要自己手动创建表，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

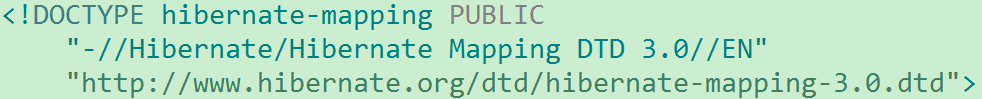
1. 创建映射配置文件



映射配置文件的名称和位置没有固定要求

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

<hibernate-mapping>

<!-- 映射实体类与表的关系

name属性：实体类的全路径

table属性：数据库要映射的表名

-->

<class name=*"com.pojo.Student"* table=*"t\_student"*>

<!-- 映射主键属性名与主键列的对应关系

name属性：实体类id属性名称

column属性：表中主键的列名

-->

<id name=*"sid"* column=*"sid"*>

<!-- 主键生成策略（明天讲）

native:生成表的id值就是主键自动递增

-->

<generator class=*"native"*></generator>

</id>

<!-- 映射其它属性和表字段的对应关系

name属性：实体类属性名称

column属性：表中字段的名称

-->

<property name=*"sname"* column=*"sname"*></property>

<property name=*"project"* column=*"project"*></property>

<property name=*"score"* column=*"score"*></property>

</class>

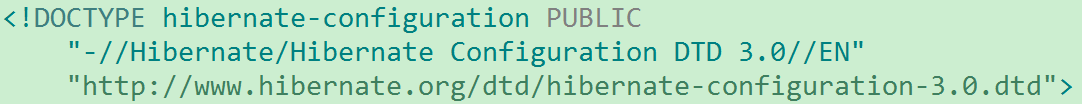
</hibernate-mapping>

**第四步：创建主配置文件（核心配置文件）**

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



（3）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

<hibernate-configuration>

<session-factory>

<!-- 第一部分：配置数据库信息 -->

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/test</property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.password"*>root</property>

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<!-- 第二部分：配置hibernate信息（可选的） -->

<!-- 输出sql语句 -->

<property name=*"hibernate.show\_sql"*>true</property>

<!-- 格式化sql语句 -->

<property name=*"hibernate.format\_sql"*>true</property>

<!-- 自动生成数据库表

update：如果已经有表，更新；如果没有，创建

-->

<property name=*"hibernate.hbm2ddl.auto"*>update</property>

<!-- 配置数据库方言 -->

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<!-- 第三部分：把映射文件放到主配置文件中 -->

<mapping resource=*"com/pojo/Student.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

## 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

@Test

**public** **void** testAdd(){

//第一步：加载hibernate主配置文件

Configuration conf=**new** Configuration();

conf.configure();

//第二步：创建sessionFactory对象

SessionFactory sf = conf.buildSessionFactory();

//第三步：使用sessionFactory创建session对象

Session session = sf.openSession();

//第四步：开启事务

Transaction transaction = session.beginTransaction();

//第五步：具体的逻辑，增删改查操作

Student stu=**new** Student();

stu.setSname("马冬梅");

stu.setProject("mysql从删库到跑路");

stu.setScore(100);

session.save(stu);

//第六步：提交事务

transaction.commit();

//第七步：关闭资源

session.close();

sf.close();

}

## Hibernate核心API：

**Configuration**



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

**SessionFactory（重点）**

1. 使用Configuration对象创建SessionFactory对象
2. 创建SessionFactory过程中做的事情

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把表创建



1. 创建SessionFactory过程中，特别耗资源
2. 在hibernate操作过程中，建议一个项目一般创建一个SessionFactory对象
3. 解决办法
4. 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，执行一次

**public** **class** HibernateUtil {

**private** **static** SessionFactory *sf*;

**static**{

//加载主配置文件

Configuration conf=**new** Configuration();

conf.configure();

*sf*=conf.buildSessionFactory();

}

**public** **static** SessionFactory getSessionFactory(){

**return** *sf*;

}

}

**Session（重点）**

1. Session类似于jdbc中的Connection
2. 调用Session里面不同的方法实现增删改查操作
3. 添加 save方法
4. 修改 update方法
5. 删除 delete方法
6. 根据id查询 get方法
7. Session对象是单线程对象

（1）Session对象不能共用，只能自己使用

**Transaction**

1. 事务对象



1. 事务的提交和回滚方法

提交：transaction.commit();

回滚：transaction.rollback();

1. 事务的概念

把多个相关的操作捆绑一个整体，要么都成功，要么都失败！

1. 事务四个特性

原子性、隔离性、一致性、持久性

原子性：事务包含的所有操作要么全部成功，要么全部失败回滚；成功必须要完全应用到数据库，失败则不能对数据库产生影响

一致性：事务执行前和执行后必须处于一致性状态

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

持久性：一个事务一旦被提交了，那么对数据库中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（一般使用id）
4. 实体类属性不建议使用基本数据类型，使用基本数据类型对应的包装类
5. 八种基本数据类型对应的包装类

int--Integer

char--Character

其它的都是首字母大写

1. 表示学生分数 int score;

如果学生得了0分：score=0;

如果学生没有参加考试：score=0;不能准确表示学生是否参加考试

解决办法：

Integer score=0;表示学生得了0分

Integer score=null;表示学生没有参加考试

## 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

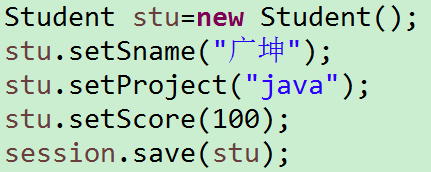


1. increment：由hibernate自动以递增的方式生成唯一标识符，每次递增1
2. identity：主键自增，由数据库负责生成主键（mysql）
3. sequence：Hibernate根据底层数据库序列生成标识符，条件是数据库支持序列（Oracle）
4. uuid：生成随机字符串作为主键，主键类型为字符串
5. hilo：高低位算法，适用于既不支持自增，也不支持序列的数据库
6. native：根据数据库来创建不同的生成策略，identity|sequence|hilo自动三选一
7. assigned：由外部程序负责生成，hibernate不负责维护主键生成

## 实体类操作

**添加操作：**

1. **调用session里面的save方法**



**根据id查询：**

1. **调用session里面的get方法**



**修改操作：**

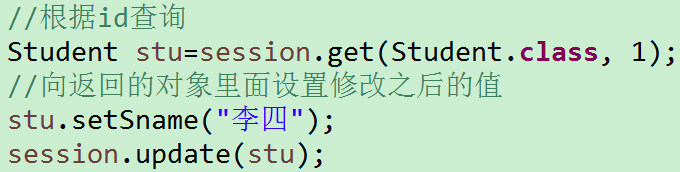
1. **调用session里面的update方法**

**根据id查询**

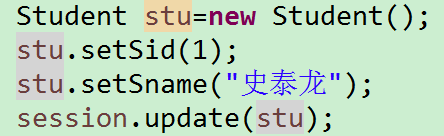
**向返回的对象里面设置修改之后的值**

**执行过程：到student对象里面找到id值，根据id进行修改**

**第一种方式：**

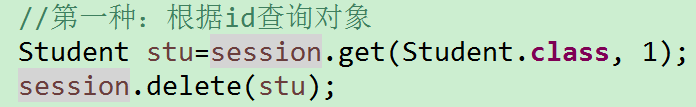


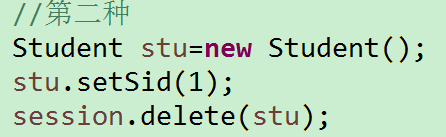
**第二种方式：**



**删除操作：**

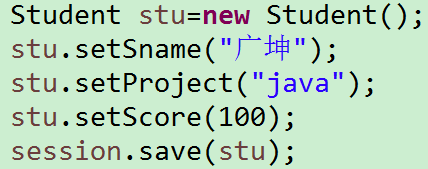
1. **调用session里面的delete方法**





## 实体类对象状态（概念）

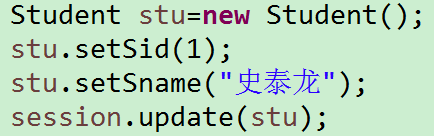
1. 实体类对象状态有三种
2. 瞬时态：对象里面没有id，对象与session没有关联



1. 持久态：对象里面有id值，对象与session有关联



1. 托管态（游离态）：对象有id值，与session没有关联



Hibernate的三种状态之间如何转换？\*\*\*\*\*

瞬时态，持久态，游离态

瞬时态到持久态：执行save操作

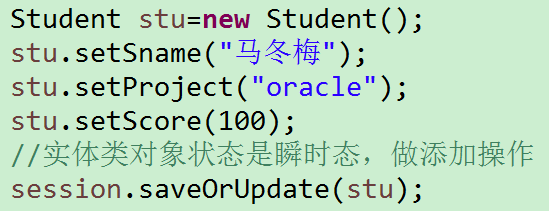
持久态到瞬时态：执行delete操作

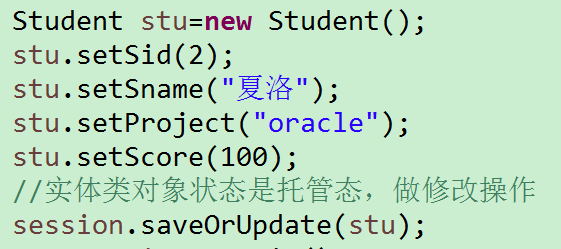
持久态到游离态：session关闭

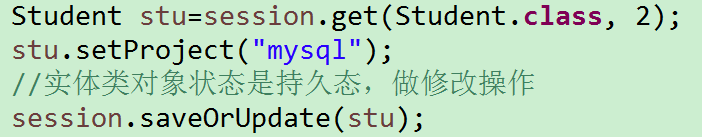
游离态到持久态：执行update操作

**演示操作实体类对象的方法**

1. saveOrUpdate方法：实现添加或修改操作







## Hibernate的一级缓存

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，提高读取效率

Hibernate的缓存

1. Hibernate框架中提供很多优化方式，Hibarnate的缓存就是一种优化方式
2. Hibernate缓存的特点

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

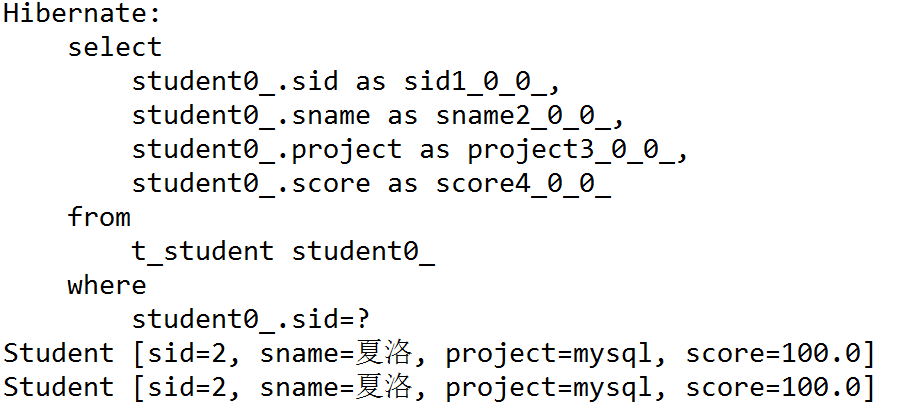
二级缓存

（1）二级缓存默认不是打开的，需要配置

（2）二级缓存的使用范围，是sessionFactory范围

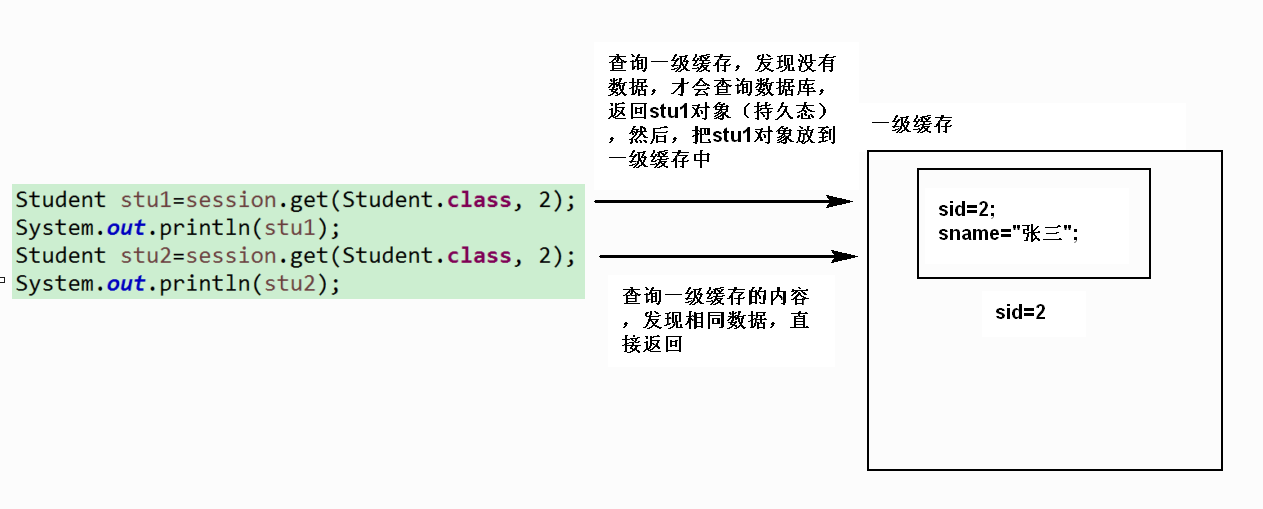
验证一级缓存

第一次根据id=2查询；第二次再根据id=2查询



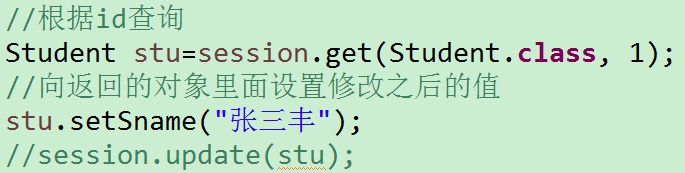
第一次执行查询向数据库发送sql语句，第二次没有发送sql语句，查询的是一级缓存的内容

**Hibernate一级缓存的执行过程**

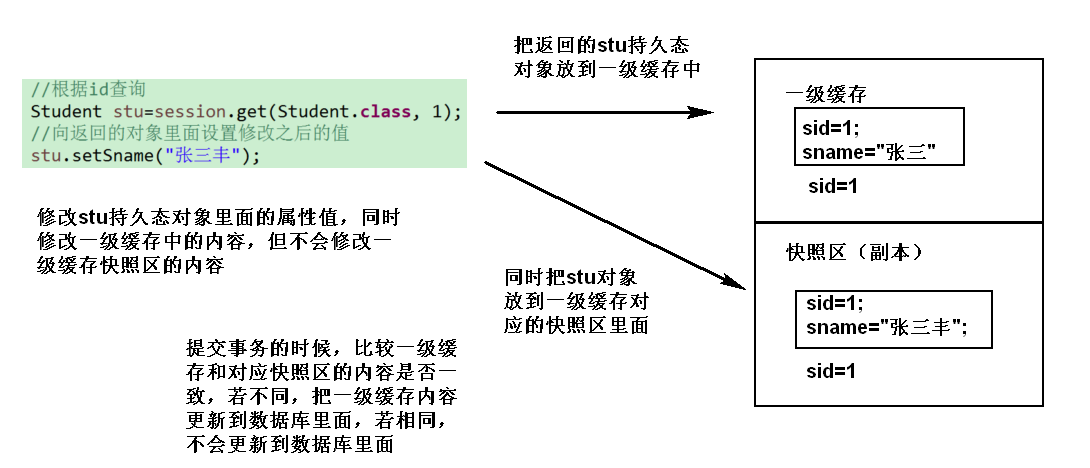


一级缓存的特性

1. 持久态会自动更新数据库



1. 执行过程（了解）



清空一级缓存

session.clear();//清除所有

session.evict(stu1);//清除stu1在一级缓存的数据

## Hibernate事务操作

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
4. 脏读
5. 不可重复的
6. 虚读（幻读）
7. 设置事务的隔离级别

Mysql默认隔离级别repeatable read

<property name=*"hibernate.connection.isolation"*>4</property>

**Hibernate事务代码规范写法**

1. 代码结构

try{

开启事务

提交事务

}catch(Exception e){

回滚事务

}finally{

关闭操作

}

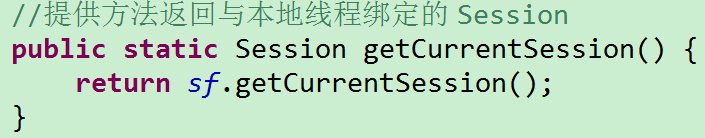
## Hibernate绑定Session

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
4. 获取与本地线程绑定的 Session
5. 在 Hibernate 核心配置文件中进行配置

<!-- 配置 Session 绑定本地线程 -->

<property name=*"hibernate.current\_session\_context\_class"*>thread</property>

1. 调用SessionFactory 的 getCurrentSession() 方法获取 Session



5、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

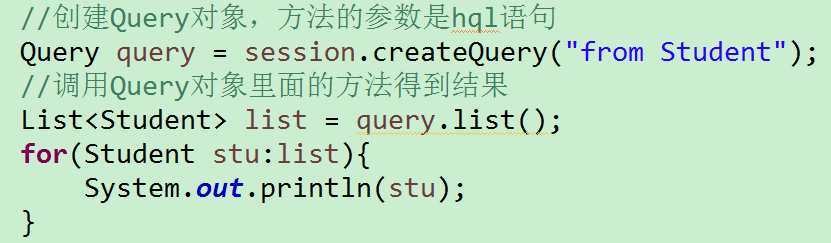
**Query对象**

1. 使用Query对象，不需要写sql语句，但是需要写hql语句
2. hql：hibernate query language，hibernate提供的查询语言，hql语句和普通sql语句很相似
3. hql和sql语句的区别

--使用sql操作表和表的字段

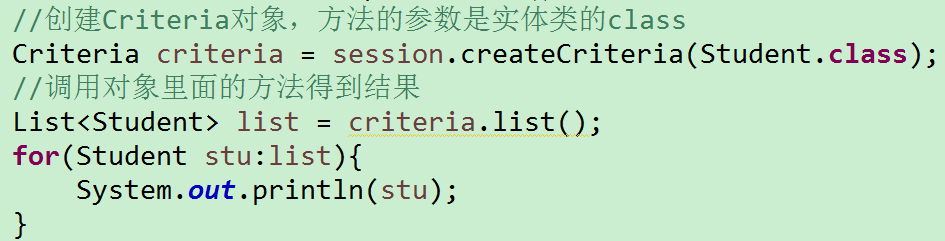
--使用hql语句操作实体类和属性

1. 查询所有的hql语句
2. from 实体类名称
3. Query对象的使用
4. 创建Query对象
5. 调用Query对象里面的方法得到结果



**Criteria对象**

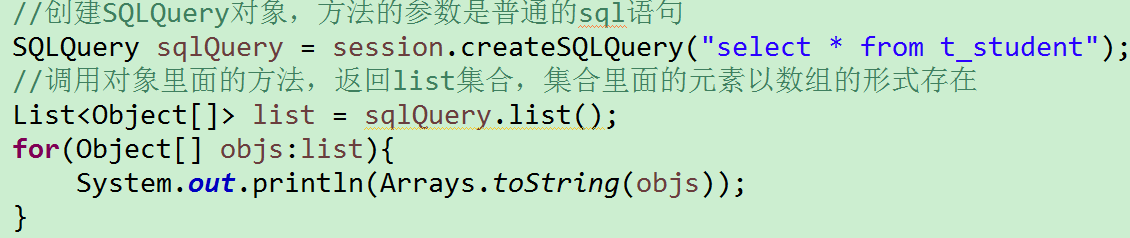
1. 使用这个对象的时候，不需要写语句，直接调用方法实现
2. 实现过程
3. 创建Criteria对象
4. 调用对象里面的方法得到结果



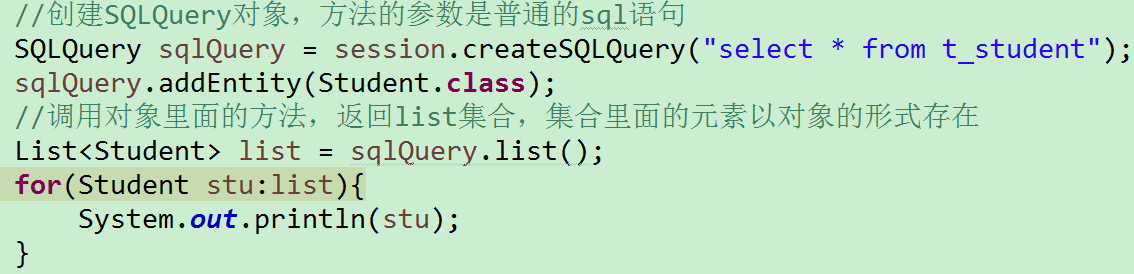
**SQLQuery对象**

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

**返回list集合每部分是数组**

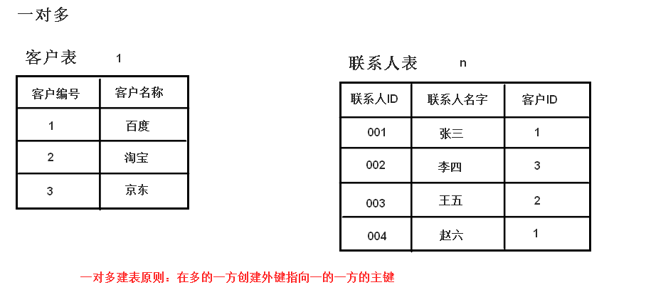


**返回list集合每部分是对象**

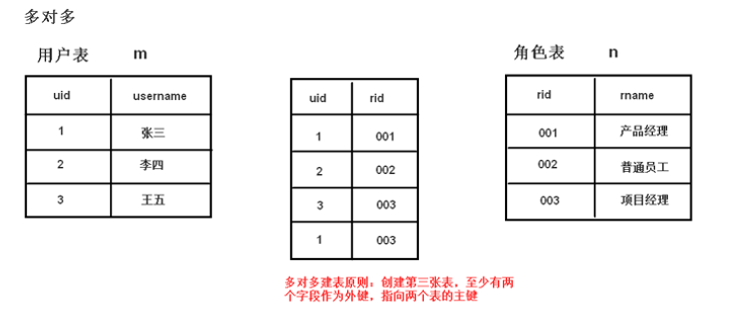


## 表与表之间关系回顾

1. 一对多
2. 一个部门有多个员工，一个员工只能属于一个部门
3. 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
4. 一对多建表：通过外键建立关系



1. 多对多
2. 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
3. 一个用户可以有多个角色，一个角色里面可以有多个用户
4. 多对多建表：创建第三张表维护关系



1. 一对一

一夫一妻

## Hibernate的一对多操作（重点）

**一对多映射配置：**

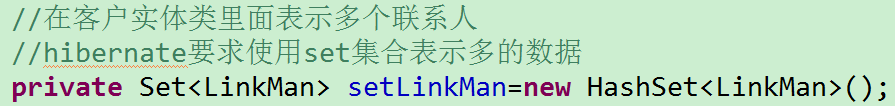
以客户和联系人为例：客户是一，联系人是多

**第一步：创建两个实体类，客户和联系人**

**第二步：让两个实体之间相互表示**

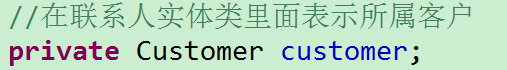
1. 在客户实体类里面表示多个联系人

一个客户可以对应多个联系人



1. 在联系人实体类里面表示所属客户

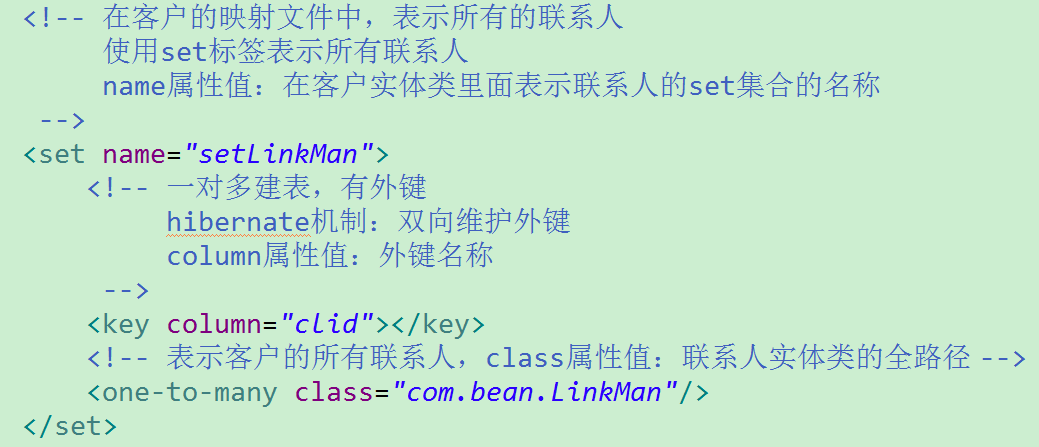
一个联系人只能属于一个客户



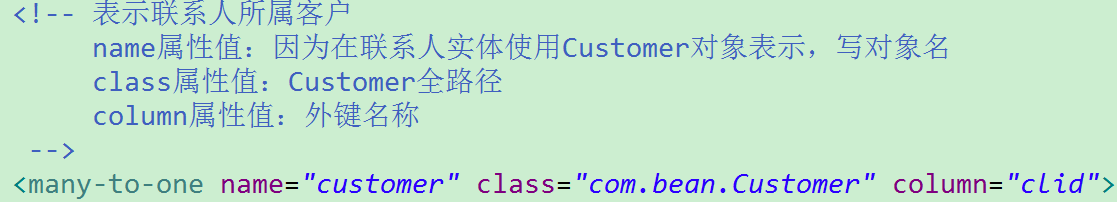
**第三步：配置映射关系**

1. 一般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）

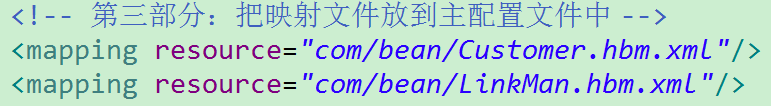
在客户的映射文件中，表示所有的联系人



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，把映射文件引入到主配置文件中



**一对多级联操作**

1. **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

复杂写法

//一对多级联保存

@Test

**public** **void** testAdd(){

Session session=**null**;

Transaction tx=**null**;

**try**{

session=HibernateUtil.*getCurrentSession*();

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("腾讯");

customer.setCustLevel("普通客户");

customer.setCustSource("网络");

customer.setCustPhone("123456");

customer.setCustMobile("147123456789");

LinkMan linkMan=**new** LinkMan();

linkMan.setLkm\_name("张三丰");

linkMan.setLkm\_gender("男");

linkMan.setLkm\_phone("10086");

//2.在客户中表示所有联系人，在联系人中表示客户

//建立客户对象和联系人对象的关系

//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.保存数据到数据库

session.save(customer);

session.save(linkMan);

tx.commit();

}**catch**(Exception e){

e.printStackTrace();

tx.rollback();

}

}

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，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

//一对多级联保存(简化写法)

@Test

**public** **void** testInsert(){

Session session=**null**;

Transaction tx=**null**;

**try**{

session=HibernateUtil.*getCurrentSession*();

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=session.get(Customer.**class**, 1);

LinkMan linkMan=**new** LinkMan();

linkMan.setLkm\_name("张三");

linkMan.setLkm\_gender("男");

linkMan.setLkm\_phone("10086");

//2.把联系人放到客户里面

customer.getSetLinkMan().add(linkMan);

//3.保存到数据库

session.saveOrUpdate(customer);

tx.commit();

}**catch**(Exception e){

e.printStackTrace();

tx.rollback();

}

}

1. **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

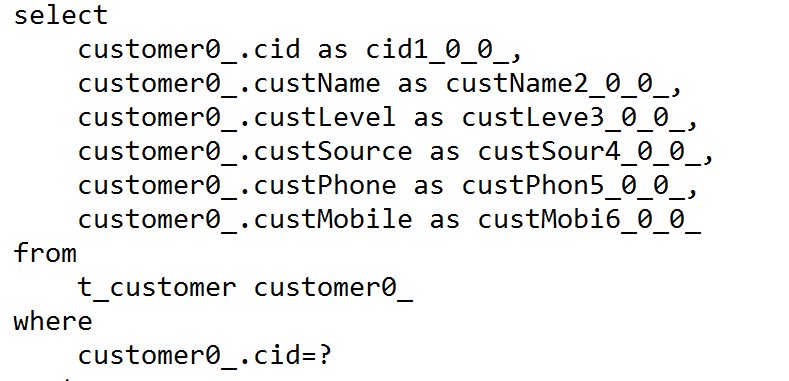


第二步：在代码中直接删除客户

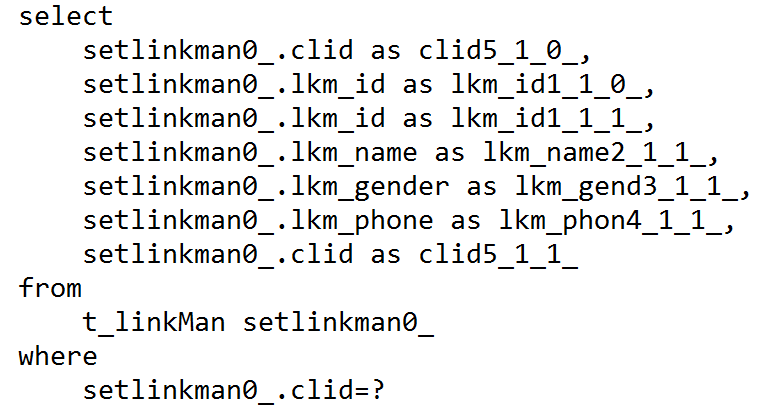
1. 根据id查询对象，调用session里面的delete方法删除

执行过程：

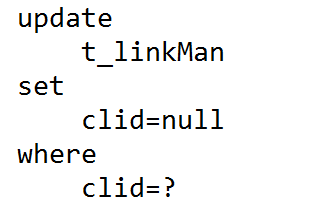
1. 根据id查询客户对象



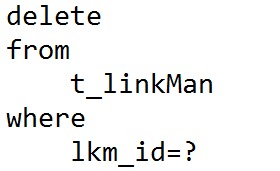
1. 根据外键id查询联系人

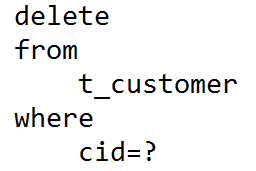


1. 把联系人的外键设置为null



1. 删除联系人和客户





**一对多级联修改**

//一对多级联修改

@Test

**public** **void** testUpdate(){

Session session=**null**;

Transaction tx=**null**;

**try**{

session=HibernateUtil.*getCurrentSession*();

tx=session.beginTransaction();

Customer customer=session.get(Customer.**class**, 3);

LinkMan linkMan=session.get(LinkMan.**class**, 3);

linkMan.setCustomer(customer);

customer.getSetLinkMan().add(linkMan);

tx.commit();

}**catch**(Exception e){

e.printStackTrace();

tx.rollback();

}

}

**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，修改客户的时候，修改一次外键；修改联系人的时候也修改一次外键，造成效率问题

Hibernate:

update

t\_linkMan

set

lkm\_name=?,

lkm\_gender=?,

lkm\_phone=?,

clid=?

where

lkm\_id=?

Hibernate:

update

t\_linkMan

set

clid=?

where

lkm\_id=?

1. 解决方式：让其中的一方不维护外键

一的一方放弃外键维护

1. 具体实现：

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

Inverse属性默认值：false不放弃关系维护，true表示放弃关系维护



## Hibernate多对多操作

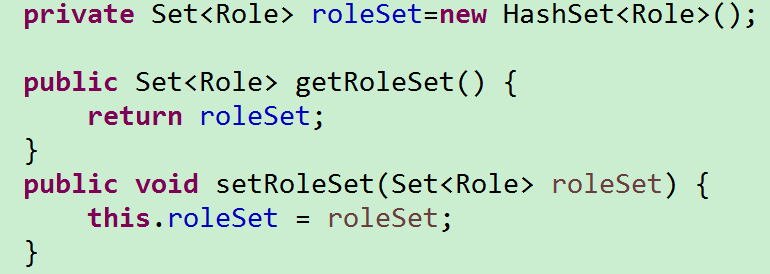
**多对多映射配置**

以用户和角色为例

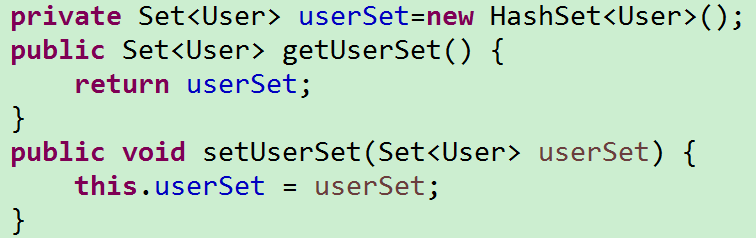
第一步：创建两个实体类，用户和角色

第二步：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



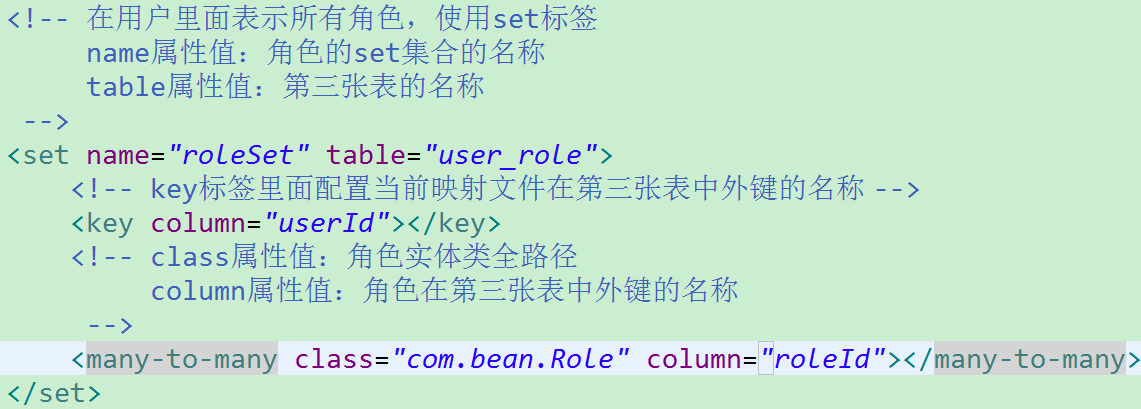
1. 在角色里面表示用户，使用set集合



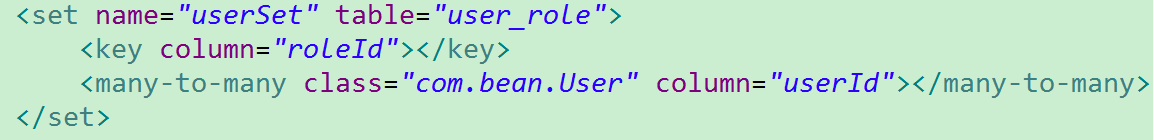
第三步：配置映射关系

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：在主配置文件中引入映射配置文件

**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



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

@Test

**public** **void** testAdd(){

Session session=**null**;

Transaction tx=**null**;

**try**{

session=HibernateUtil.*getCurrentSession*();

tx=session.beginTransaction();

//添加两个用户，为每个用户添加两个角色

User user1=**new** User();

user1.setUsername("马冬梅");

user1.setPassword("123456");

User user2=**new** User();

user2.setUsername("大春");

user2.setPassword("123456");

Role role1=**new** Role();

role1.setRname("项目经理");

role1.setRdesc("厉害了");

Role role2=**new** Role();

role2.setRname("产品经理");

role2.setRdesc("上天了");

Role role3=**new** Role();

role3.setRname("研发经理");

role3.setRdesc("不错呦");

user1.getRoleSet().add(role1);

user1.getRoleSet().add(role2);

user2.getRoleSet().add(role2);

user2.getRoleSet().add(role3);

session.save(user1);

session.save(user2);

tx.commit();

}**catch**(Exception e){

e.printStackTrace();

tx.rollback();

}

}

**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

@Test

**public** **void** testDelete(){

Session session=**null**;

Transaction tx=**null**;

**try**{

session=HibernateUtil.*getCurrentSession*();

tx=session.beginTransaction();

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

session.delete(user);

tx.commit();

}**catch**(Exception e){

e.printStackTrace();

tx.rollback();

}

}

**维护第三张表的关系**

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

@Test

**public** **void** testAddRole(){

Session session=**null**;

Transaction tx=**null**;

**try**{

session=HibernateUtil.*getCurrentSession*();

tx=session.beginTransaction();

//根据id查询用户和角色

User user=session.get(User.**class**, 2);

Role role=session.get(Role.**class**, 4);

//把角色放到用户的set集合里面

user.getRoleSet().add(role);

tx.commit();

}**catch**(Exception e){

e.printStackTrace();

tx.rollback();

}

}

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

@Test

**public** **void** testRemoveRole(){

Session session=**null**;

Transaction tx=**null**;

**try**{

session=HibernateUtil.*getCurrentSession*();

tx=session.beginTransaction();

//根据id查询用户和角色

User user=session.get(User.**class**, 2);

Role role=session.get(Role.**class**, 4);

//从用户的set集合里面把角色移除

user.getRoleSet().remove(role);

tx.commit();

}**catch**(Exception e){

e.printStackTrace();

tx.rollback();

}

}

## Hibernate查询方式

1. 对象导航查询

（1）根据id查询某个客户，再查询这个客户里面所有的联系人

1. oid查询

（1）根据id查询一条记录，返回对象

1. HQL查询

（1）Query对象，写hql语句实现查询

1. QBC查询

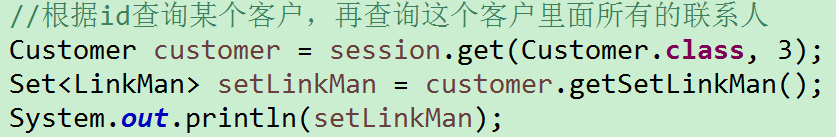
（1）Criteria

1. 本地SQL查询

（1）SQLQuery对象，使用普通sql实现查询

**对象导航查询**

1. 查询某个客户里面所有联系人的过程，使用对象导航查询实现



## OID查询

1. 根据id查询记录
2. 调用session里面的get方法实现



## HQL查询

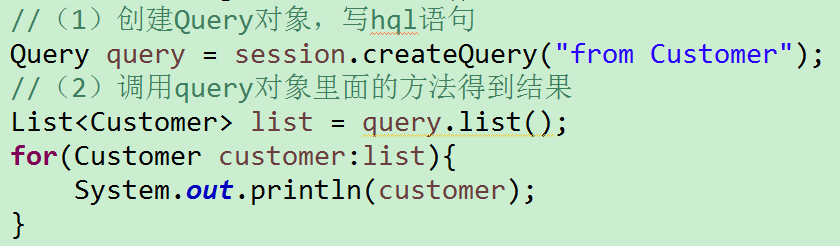
1. hql:hibernate query language，Hibernate提供的一种查询语言，hql语句和sql语句很相似

区别：普通sql操作数据库表和字段；hql操作实体类和属性

1. 常用的hql语句
2. 查询所有：from 实体类名
3. 条件查询：from 实体类名 where 实体类属性名称=? and 实体类属性名称=?
4. 排序查询：from 实体类名 order by 实体类属性名称 asc/desc
5. 使用hql查询操作的时候，使用Query对象
6. 创建Query对象，写hql语句
7. 调用query对象里面的方法得到结果

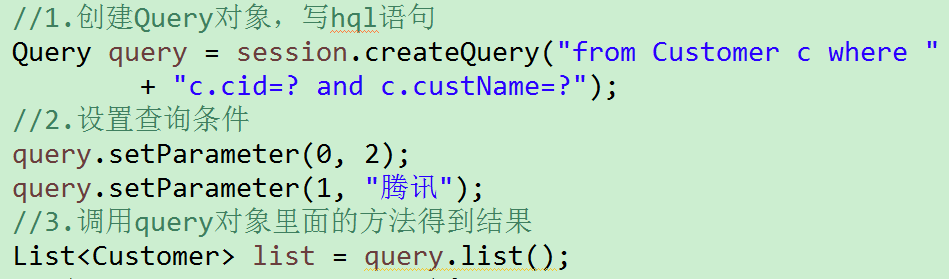
**查询所有**

1. 查询所有客户信息：from Customer

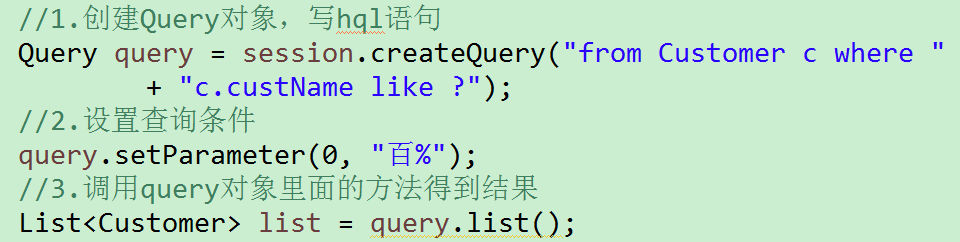


**条件查询**

1. hql条件查询语法：
2. from 实体类名 where 实体类属性名称=? and 实体类属性名称=?
3. from 实体类名 where 实体类属性名称 like ?

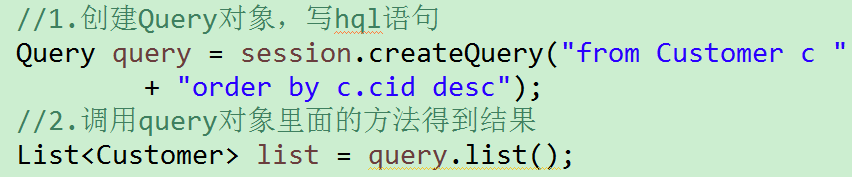


1. 模糊查询



1. 排序查询
2. hql排序查询语法

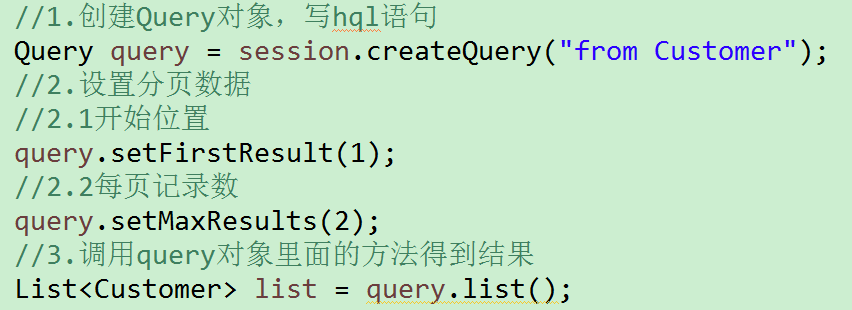
（1）from 实体类名 order by 实体类属性名称 asc/desc



**分页查询**

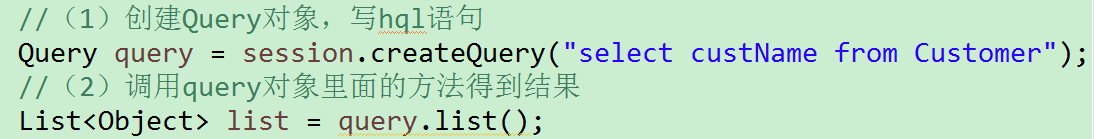
1. mysql实现分页查询，使用关键字limit实现
2. hql实现分页查询
3. 在hql语句里面不能写limit，hibernate的Query对象封装了两个方法实现分页操作

开始位置计算公式：（当前页-1）\*每页记录数



**投影查询**

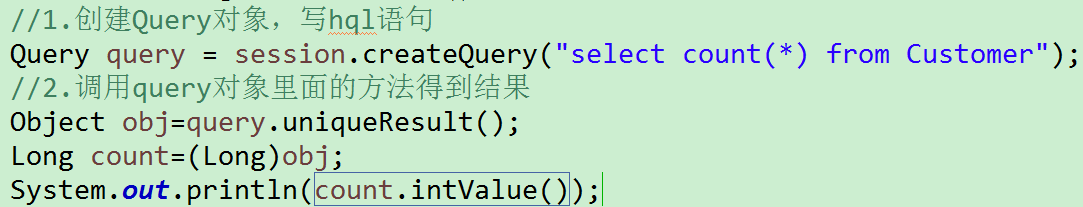
1. 投影查询：查询一个或多个属性，但不查询全部
2. 投影查询hql语句写法：
3. select 实体类属性名称1，实体类属性名称2 from 实体类名称
4. select后面不能写\*，不支持



**聚集函数使用**

1. 常用的聚集函数
2. 总数：count()、总和：sum()、平均：avg()、最大：max()、最小：min()
3. hql聚集函数语法：
4. select count(\*) from 实体类名称
5. Count函数返回的结果必须使用Long类型数据接收

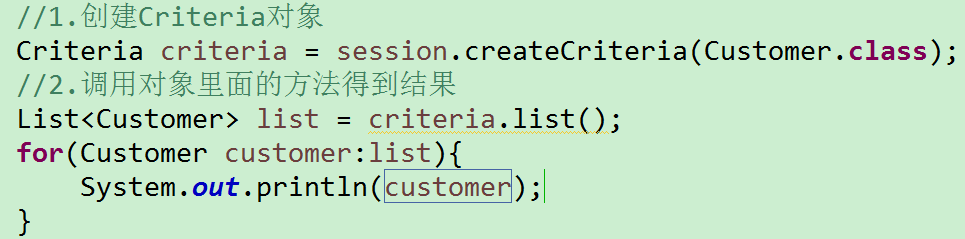




## QBC查询

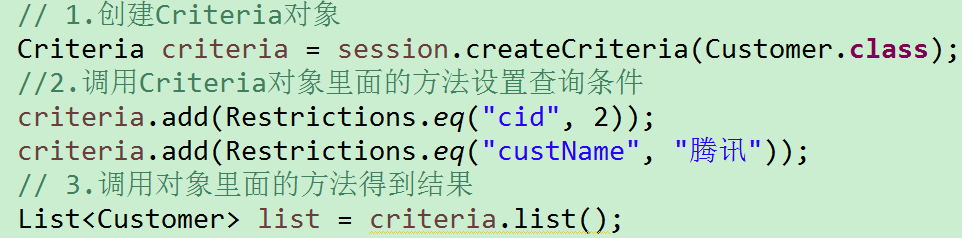
1. HQL查询需要写hql语句实现，QBC查询不需要写语句了，使用方法实现
2. 使用QBC查询的时候，操作实体类和属性
3. 使用QBC查询的时候，**使用**Criteria对象实现

**查询所有**

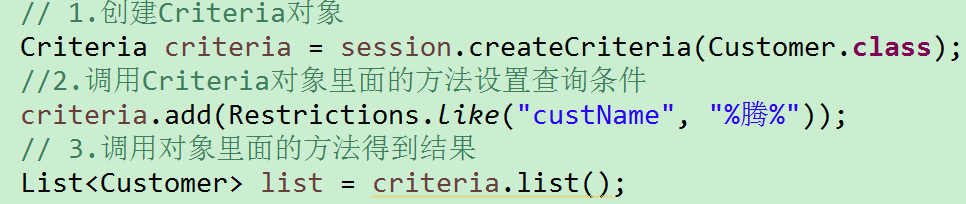


**条件查询**

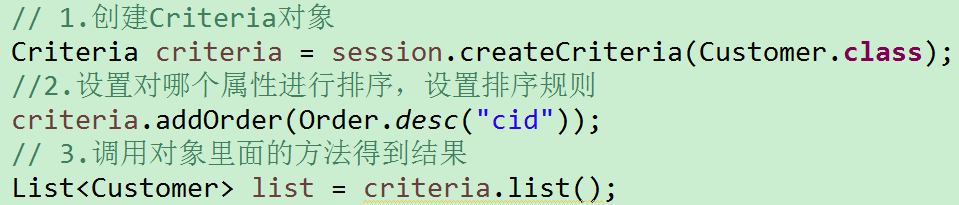
1. 不需要写语句，使用封装的方法实现



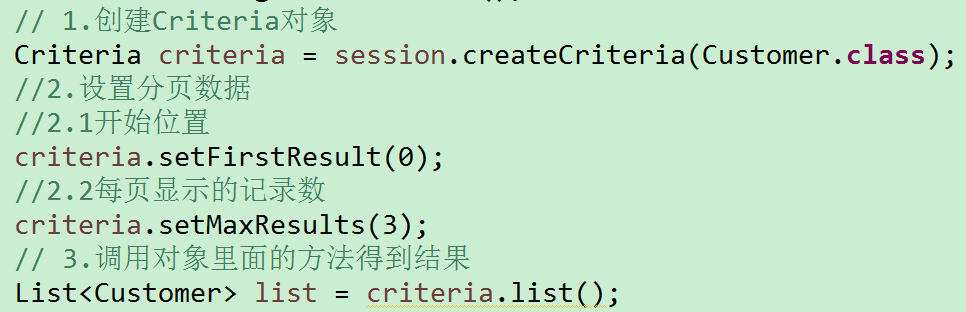
1. 模糊查询



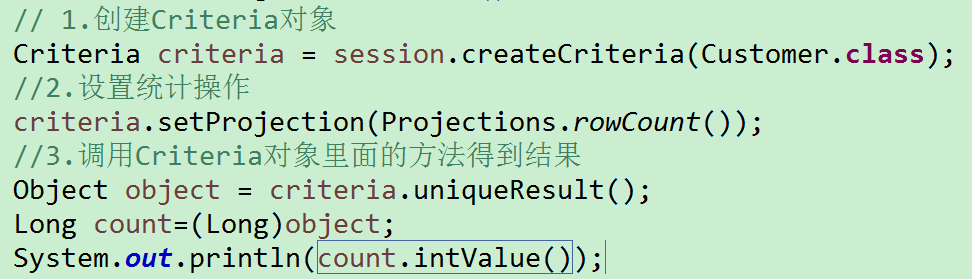
**排序查询**



**分页查询**



**统计查询**



**离线查询**

//1.创建对象

DetachedCriteria detachedCriteria = DetachedCriteria.*forClass*(Customer.**class**);

//2.最终执行的时候才需要用到session

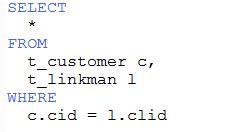
Criteria criteria = detachedCriteria.getExecutableCriteria(session);

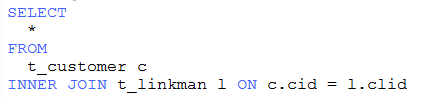
List<Customer> list = criteria.list();

**HQL多表查询**

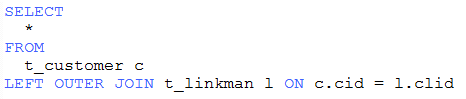
**Mysql的多表查询**

1. 内连接

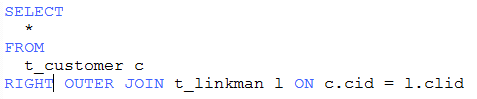




1. 左外连接



1. 右外连接

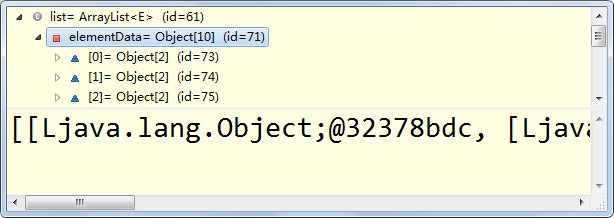


**HQL实现多表查询**

1. 内连接
2. 左外连接
3. 右外连接
4. 迫切内连接
5. 迫切左外连接

HQL内连接

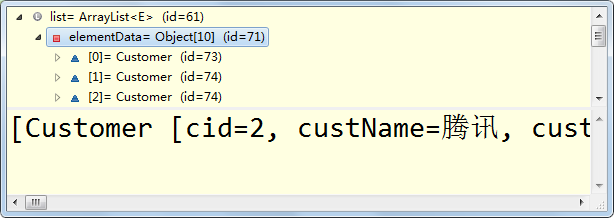
1. HQL内连接查询语法：（以客户和联系人为例）
2. from Customer c inner join c.setLinkMan
3. 返回list集合，list里面每部分是数组形式



迫切内连接

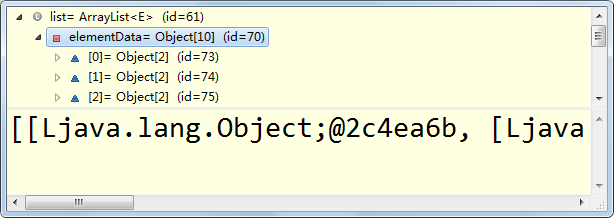
1. 迫切内联接和内连接底层实现是一样的
2. 区别：内连接返回list每部分是数组，迫切内连接返回list每部分是对象
3. HQL语句写法：from Customer c inner join fetch c.setLinkMan

通知hibernate将另一个对象的数据封装到该对象中



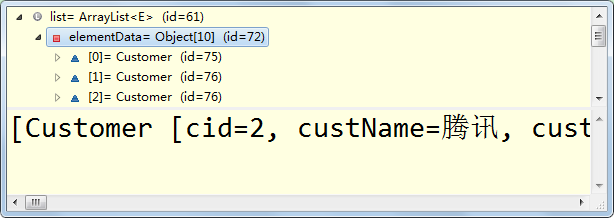
HQL左外连接

1. HQL左外连接语法：
2. from Customer c left join c.setLinkMan



迫切左外连接

1. from Customer c left join fetch c.setLinkMan



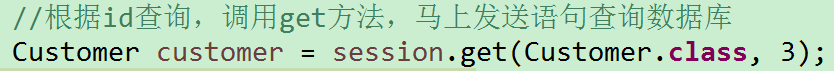
HQL右外连接

1. HQL右外连接语法：
2. from Customer c right join c.setLinkMan

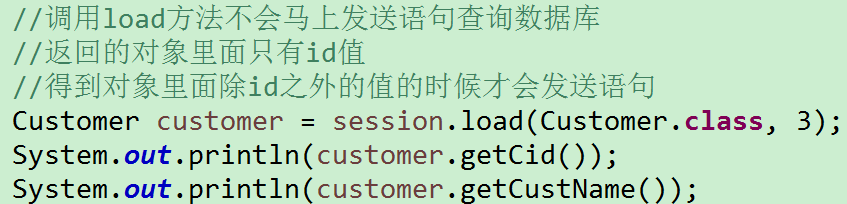
## **Hibernate检索策略**

检索策略概念：

1. 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，马上发送语句查询数据库

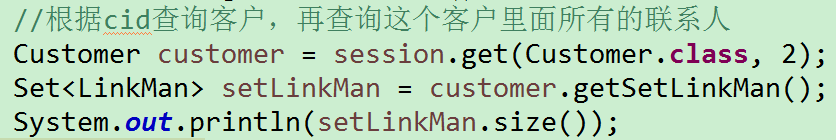


1. 延迟查询：根据id查询，还有load方法，调用load方法不会马上发送语句查询数据库，只有得到对象的值的时候才会发送语句查询数据库



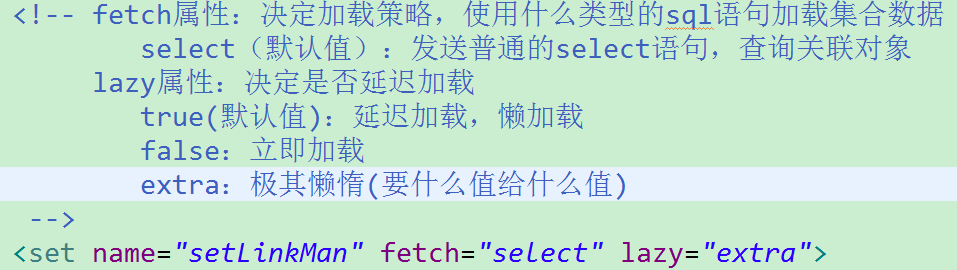
延迟查询分为两类：

1. 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
2. 关联级别延迟：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



关联级别延迟操作

1. 在映射文件中通过配置实现
2. 根据客户得到所有联系人，在客户的映射文件中配置



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

