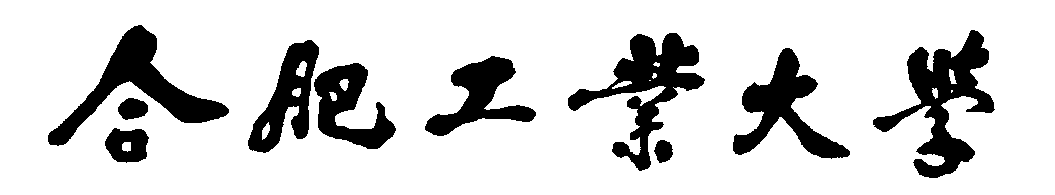
****

专业课程

（计算机与信息学院）

大数据处理技术课程报告

|  |  |
| --- | --- |
| 专 业 班 级 | 计算机创新实验18-1班 |
| 学生姓名及学号 | 刘嘉伟-2018213106 |
| 任 课 教 师 | 吴共庆 |
| 实验指导教师 | 吴共庆 |
| 实验地点 | 宿舍 |
| 2018～2019学年第 二 学期 | |

目录

[1． CRM概述 1](#_Toc23979721)

[1.1什么是CRM 1](#_Toc23979722)

[1.2CRM的功能模块 1](#_Toc23979723)

[JavaEE开发三层结构 1](#_Toc23979724)

[2．Hibernate的概述 2](#_Toc23979725)

[2.1什么是Hibernate 2](#_Toc23979726)

[2.2为什么要学习Hibernate 2](#_Toc23979727)

[2.3什么是orm思想 3](#_Toc23979728)

[2.4Hibernate的入门 4](#_Toc23979729)

[2.4.1下载Hibernate5 4](#_Toc23979730)

[2.4.2Hibernate环境搭建 4](#_Toc23979731)

[2.4.3实现添加操作 8](#_Toc23979732)

[2.5Hibernate配置文件及API讲解 10](#_Toc23979733)

[Hibernate映射配置文件 10](#_Toc23979734)

[Hibernate核心配置文件 10](#_Toc23979735)

[Hibernate核心API 10](#_Toc23979736)

[2． 使用Hibernate完成CRM客户管理中的查询所有客户操作 12](#_Toc23979737)

[2.1需求描述 12](#_Toc23979738)

[2.2实体类的编写规则 12](#_Toc23979739)

[2.3Hibernate主键生成策略 13](#_Toc23979740)

[2.4对实体类的操作 14](#_Toc23979741)

[2.4.1.Crud操作 14](#_Toc23979742)

[2.4.1.2修改操作 15](#_Toc23979743)

[2.4.1.3删除操作 16](#_Toc23979744)

[2.4.2实体类对象状态 17](#_Toc23979745)

[2.5Hibernate的一级缓存 18](#_Toc23979746)

[2.5.1什么是缓存 18](#_Toc23979747)

[2.5.2Hibernate一级缓存特性 18](#_Toc23979748)

[2.5.3验证一级缓存存在 19](#_Toc23979749)

[一级缓存执行过程 20](#_Toc23979750)

[2.6Hibernate事务操作 20](#_Toc23979751)

[2.6.1事务相关概念 20](#_Toc23979752)

[2.6.2事务代码规范写法 20](#_Toc23979753)

[2.7Hibernate的api使用 21](#_Toc23979754)

[Query对象 21](#_Toc23979755)

[3． Hibernate的一对多的关联关系映射并操作 23](#_Toc23979756)

[3.1表与表之间一对多的关系 23](#_Toc23979757)

[3.2Hibernate的一对多操作 24](#_Toc23979758)

[3.2.1一对多映射配置 24](#_Toc23979759)

[3.2.2一对多的联级操作 29](#_Toc23979760)

[3.3Hibernate的多对多操作 32](#_Toc23979761)

[3.3.1多对多的映射配置 32](#_Toc23979762)

[3.3.2多对多的联级操作 35](#_Toc23979763)

[4． Hibernate的条件查询 38](#_Toc23979764)

[4.1HQL查询 38](#_Toc23979765)

[4.1.1查询所有 38](#_Toc23979766)

[4.2.2条件查询 39](#_Toc23979767)

[4.2.3模糊查询 40](#_Toc23979768)

[4.3.4排序查询 40](#_Toc23979769)

[4.3.5分页查询 41](#_Toc23979770)

[4.3.6投影查询 42](#_Toc23979771)

[4.3.7聚集函数的使用 43](#_Toc23979772)

[4.4HQL多表查询 44](#_Toc23979773)

[4.4.1HQL内连接 44](#_Toc23979774)

[4.4.2HQL左外连接 45](#_Toc23979775)

[4.5Hibernate的检索策略 46](#_Toc23979776)

[4.5.1检索策略测概念 46](#_Toc23979777)

[4.5.2批量抓取 47](#_Toc23979778)

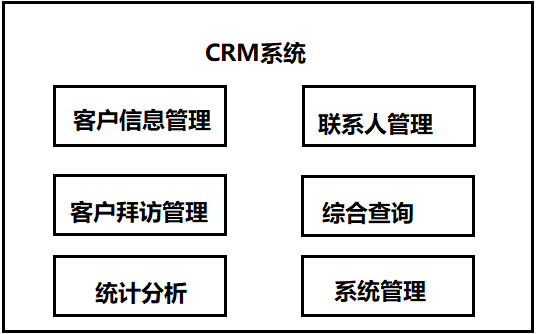
# CRM概述

## 1.1什么是CRM

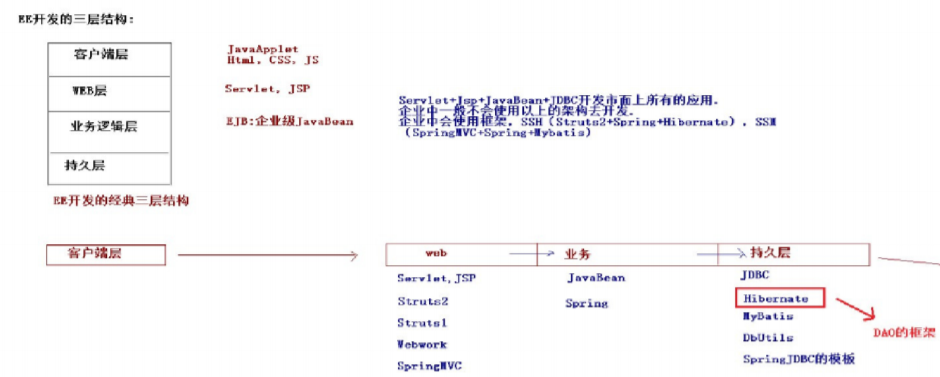
CRM(Customer Relation Management)客户关系管理，是利用相应的信息技术以及互联网技术来协调企业与顾客间在销售、营销和服务上的交互，向客户提供创新式的个性化的客户交互和服务的过程。其最终目标是将面向客户的各项信息和活动集成起来，组织一个以客户为中心的企业，实现面向客户的活动的全面管理。

## 1.2CRM的功能模块

CRM系统实现了对企业销售、营销、服务等各阶段的客户信息、客户活动进行统一管理。CRM系统功能涵盖企业销售、营销、用户服务等各个业务流程，业务流程中与客户相关活动都会在CRM系统统一管理，下边列出一些基本的功能模块，包括：客户信息管理、联系人管理、商机管理、统计分析等。



## JavaEE开发三层结构

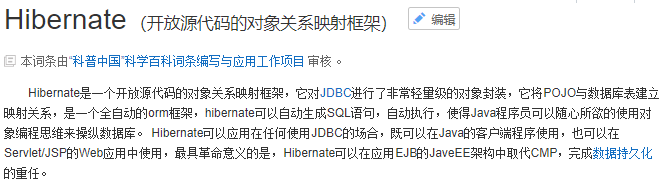


# 2．Hibernate的概述

## 2.1什么是Hibernate

Hibernate框架是当今主流的Java持久层框架之一，由于它具有简单易学、灵活性强、拓展性强等特点，能够大大地简化程序的代码量提高工作效率，因此受到广大开发人员地喜爱。

Hibernate是一个开源的ORM框架，它对JDBC进行了轻量级的对象封装，使得Java开发人员可以使用面向对象的编程思想来操作数据库。

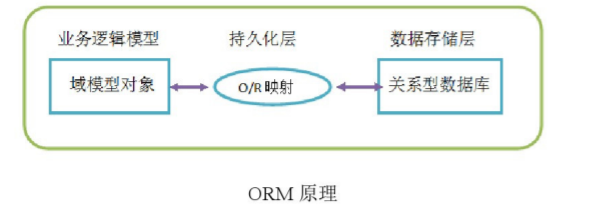


Hibernate就是一个持久层的ORM的框架。

## 2.2为什么要学习Hibernate

使用传统的JDBC开发应用系统时，如果是小型应用系统，并不觉得有什么麻烦，但是对于大型应用系统的开发，使用JDBC就会显得力不从心。例如对几十、几百张包含几十个字段的表进行插入操作时，编写的SQL语句不但很长，而且繁琐，容易出错，需要写多条getXxx语句从结果集中提取出各个字段的信息，不但枯燥重复，并且工作量非常大。为了提高数据访问层的编程效率，Gavin King开发出了当今最流行的ORM框架，它就是Hibernate框架。

所谓的ORM就是利用描述对象和数据库表之间的映射的元数据，自动把Java应用程序中的对象，持久化到关系型数据库的表中。通过操作Java对象，就可以完成对数据库表的操作。可以把ORM理解为关系型数据和对象的一个纽带，开发人员只需要关注纽带一端的映射对象即可。ORM原理如图



与其他操作数据的技术相比，Hibernate具有如下几点优势：

●Hibernate对JDBC访问数据库的代码做了轻量级的封装，大大简化了数据访问层繁琐重复性代码，并且减少了内存消耗，加快了运行效率。

●Hibernate是一个基于JDBC的主流持久化框架，是一个优秀的ORM实现，它很大程度上简化了DAO（Data Access Object，数据访问对象）层的编码工作。

●Hibernate的性能非常好，映射的灵活性很出色。它支持很多关系型数据库，从一对一到多对多的各种复杂关系。

●可拓展性强，由于源代码的开源以及API的开放，当本身功能不够时，可以自行编码进行拓展。

## 2.3什么是orm思想

1 hibernate使用orm思想对数据库进行crud操作

2 在web阶段学习 javabean，更正确的叫法 实体类

3 orm：object relational mapping，对象关系映射

文字描述：

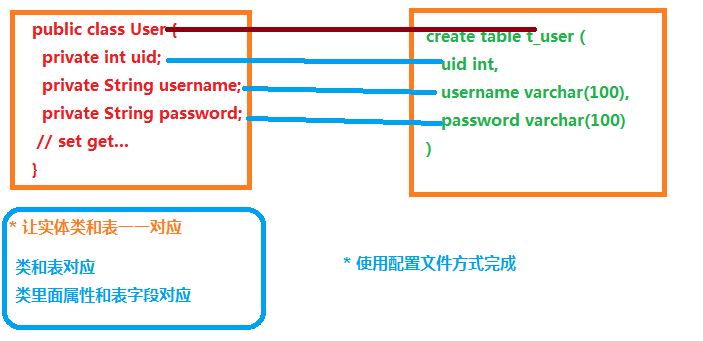
（1）让实体类和数据库表进行一一对应关系

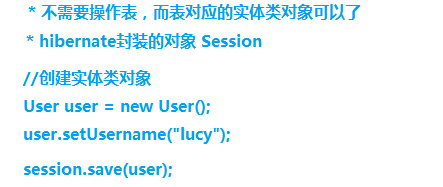
让实体类首先和数据库表对应

让实体类属性 和 表里面字段对应

（2）不需要直接操作数据库表，而操作表对应实体类对象

画图描述：

****

****

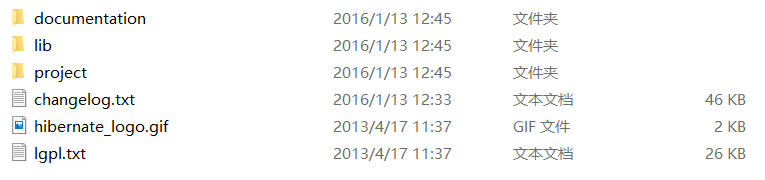
## 2.4Hibernate的入门

### 2.4.1下载Hibernate5

<https://sourceforge.net/projects/hibernate/files/hibernate-orm/5.0.7.Final/>



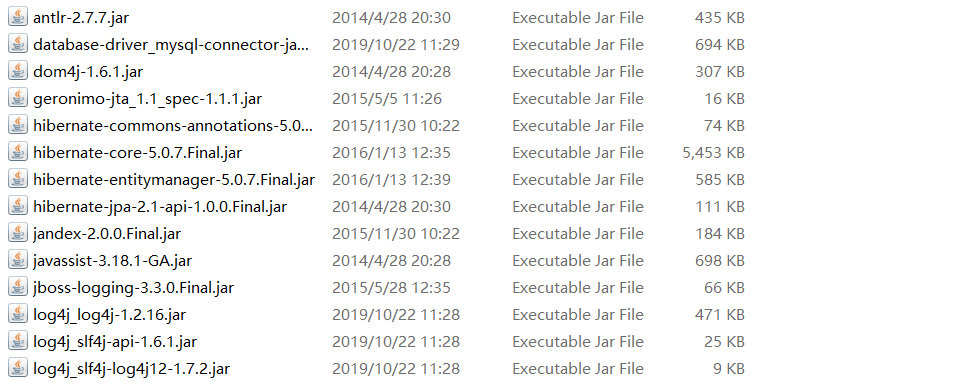
下载后，解压完的目录结构如图



### 2.4.2Hibernate环境搭建

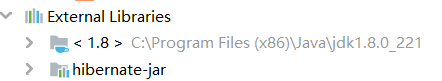
第一步 导入jar包

对其中学习用的jar包进行总和，放进一个jar包中，如下



打开IDEA，导入这些jar包文件夹

点击File->Project Sturcture->Libraries->+号->Java->选择文件夹位置



第二步 创建实体类

**public class** User {

/\*Hibernate要求实体类有一个属性唯一，也即主键约束\*/  
 **private int uid**;  
 **private** String **username**;  
 **private** String **password**;  
 **private** String **address**;  
  
 **public int** getUid() {  
 **return uid**;  
 }  
  
 **public void** setUid(**int** uid) {  
 **this**.**uid** = uid;  
 }  
  
 **public** String getUsername() {  
 **return username**;  
 }  
  
 **public void** setUsername(String username) {  
 **this**.**username** = username;  
 }  
  
 **public** String getPassword() {  
 **return password**;  
 }  
  
 **public void** setPassword(String password) {  
 **this**.**password** = password;  
 }  
  
 **public** String getAddress() {  
 **return address**;  
 }  
  
 **public void** setAddress(String address) {  
 **this**.**address** = address;  
 }  
}

注意：使用hibernate时，不需要自己手动创建表，hibernate自动会创建，但需要选择数据库！

第三步 配置实体类和数据库表的对应关系——映射关系

使用配置文件实现映射关系

1. 创建xml格式的配置文件

-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，实体类名称hbm.xml

1. 配置是xml格式，在配置文件中首次引入xml约束

-常见约束有dtd、schema，在hibernate里面引入的约束dtd约束



**<!DOCTYPE hibernate-mapping PUBLIC  
 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"*>***<**hibernate-mapping**>

1. 配置映射关系

<**hibernate-mapping**>  
 *<!-- 1配置类和表对应  
 class标签  
 name属性：实体类全路径  
 table属性：数据库表名称  
 -->* <**class name="D1.User" table="t\_user"**>  
 *<!-- 2配置实体类id和表id对应  
 hibernate要求实体类有一个属性唯一值  
 hibernate要求有字段作为唯一值  
 -->  
 <!-- id标签  
 name属性：实体类里面id属性名称  
 column属性：生成表字段名称  
 -->* <**id name="uuid" column="uid"**>  
 *<!-- 设置数据库表id的增长策略  
 native：生成表id值就是主键自动增长  
 -->* <**generator class="uuid"**></**generator**>  
 </**id**>  
 *<!-- 配置其他属性和表字段对应  
 name属性：实体类属性名称  
 column：生成表字段名称  
 -->* <**property name="username" column="username"**></**property**>  
 <**property name="password" column="password"**></**property**>  
 <**property name="address" column="address"**></**property**>  
  
 </**class**>  
</**hibernate-mapping**>

第四步 创建hibernate的核心配置文件

1. 核心配置文件格式是xml,但是核心配置文件名称和位置固定

-位置：必须位于src下面

-名称：必须hibernate.cfg.xml



1. 引入dtd约束



**<!DOCTYPE hibernate-mapping PUBLIC  
 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"*>***

1. Hibernate操作过程中，只会加载核心配置文件，其他配置文件不会加载

-第一部分：配置数据库信息 必须的

-第二部分：配置hibernate信息 可选的

-第三部分：把映射文件加载到核心配置文件中

<**hibernate-configuration**>  
 <**session-factory**>  
 *<!-- 第一部分：配置hibernate信息 必须的-->* <**property name="hibernate.connection.driver\_class"**>com.mysql.jdbc.Driver</**property**>  
 <**property name="hibernate.connection.url"**>jdbc:mysql:///hibernate</**property**>  
 <**property name="hibernate.connection.username"**>root</**property**>  
 <**property name="hibernate.connection.password"**>258079</**property**>  
 *<!-- 第二部分：配置hibernate的信息 可选的-->  
 <!-- 输出底层sql语句-->* <**property name="hibernate.show\_sql"**>true</**property**>  
 *<!-- 输出底层sql语句格式-->* <**property name="hibernate.format\_sql"**>true</**property**>  
 *<!-- hibernate帮我们创建表，需要配置之后  
 update：如果已经有表，更新，如果没有表，创建  
 -->* <**property name="hibernate.hbm2ddl.auto"**>update</**property**>  
 *<!-- 配置数据库的方言  
 在MySQL里实现分页，关键字limit，只能使用MySQL里面  
 在Oracle里实现rownum  
 让hibernate框架实现不同的自己特有的语句  
 -->* <**property name="hibernate.dialect"**>org.hibernate.dialect.MySQL5Dialect</**property**>  
 *<!-- 第三部分：把映射文件放到核心配置文件中 必须的-->* <**mapping resource="D1/User.hbm.xml"**/>  
 </**session-factory**>  
</**hibernate-configuration**>

**注意：需要自行创建一个名为hibernate的数据库**

### 2.4.3实现添加操作

第一步 加载hibernate核心配置文件

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

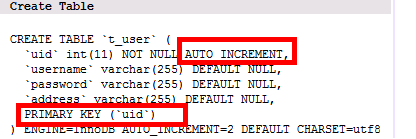
第五步 写具体逻辑 crud操作

第六步 提交事务

第七步 关闭资源  
**import** org.hibernate.Session;  
**import** org.hibernate.SessionFactory;  
**import** org.hibernate.Transaction;  
**import** org.hibernate.cfg.Configuration;  
**import** utils.HibernateUtils;  
  
**public class** HibernateDemo {  
  
 **public static void** testAdd(){  
*// 第一步 加载hibernate核心配置文件；  
 //到src下面找到名称是hibernate.cfg.xml  
 //在hibernate里面封装对象  
 Configuration cfg = new Configuration();  
 cfg.configure();  
  
// 第二步 创建SessionFactory对象；  
 //读取hibernate核心配置文件的内容，创建sessionFactory  
 //在过程中，根据映射关系，在配置数据库里面把表创建  
 SessionFactory sessionFactory = cfg.buildSessionFactory();*  
*// 第三步 使用SessionFactory创建Session对象；  
 //类似于连接* Session session = sessionFactory.openSession() ;  
  
*// 第四步 开启事务；* Transaction tx = session.beginTransaction();  
  
*// 第五步 写具体逻辑crud操作；  
 //添加功能* User user = **new** User();  
 user.setUsername(**"小马"**);  
 user.setPassword(**"1314520"**);  
 user.setAddress(**"美国"**);  
 *//调用session的方法实现添加* session.save(user);  
  
*// 第六步 提交事务；* tx.commit();  
  
*// 第七步 关闭资源* session.close();  
 sessionFactory.close();  
 }  
  
 **public static void** main(String[] args) {  
 *testAdd*();  
 }  
}

看到效果：

1. 是否生成表



1. 看表是否有记录

****

## 2.5Hibernate配置文件及API讲解

### Hibernate映射配置文件

1 映射配置文件名称和位置没有固定要求

2 映射配置文件中，标签name属性值写实体类相关内容

（1）class标签name属性值实体类全路径

（2）id标签和property标签name属性值 实体类属性名称

3 id标签和property标签，column属性可以省略的

（1）不写值和name属性值一样的

4 property标签type属性，设置生成表字段的类型，自动对应类型

### Hibernate核心配置文件

1 配置写位置要求

2 配置三部分要求

（1）数据库部分必须的

（2）hibernate部分可选的

（3）映射文件必须的

3 核心配置文件名称和位置固定的

（1）位置：src下面

（2）名称：hibernate.cfg.xml

### Hibernate核心API

**Configuration**

*Configuration cfg = new Configuration();  
 cfg.configure();*

到src下面找到名称hibernate.cfg.xml配置文件，创建对象，把配置文件放到对象里面（加载核心配置文件）

**SessionFactory**

1 使用configuration对象创建sessionFactory对象

（1）创建sessionfactory过程中做事情：

- 根据核心配置文件中，有数据库配置，有映射文件部分，到数据库里面根据映射关系把表创建

2 创建sessionFactory过程中，这个过程特别耗资源的

（1）在hibernate操作中，建议一个项目一般创建一个sessionFactory对象

3 具体实现

（1）写工具类，写静态代码块实现

\* 静态代码块在类加载时候执行，执行一次

在src文件夹下创建一个utils的包，再写一个类

**package** utils;  
**import** org.hibernate.SessionFactory;  
**import** org.hibernate.cfg.Configuration;  
  
**public class** HibernateUtils {  
 **static** Configuration *cfg* = **null**;  
 **static** SessionFactory *sessionFactory* = **null**;  
 *//静态代码块实现* **static** {  
 *//加载核心配置文件  
 cfg* = **new** Configuration();  
 *cfg*.configure();  
 *sessionFactory* = *cfg*.buildSessionFactory();  
 }  
  
*// 提供方法返回sessionFactory* **public static** SessionFactory getSessionFactory(){  
 **return** *sessionFactory*;  
 }  
}

**Session**

1 session类似于jdbc中connection

2 调用session里面不同的方法实现crud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

3 session对象单线程对象

1. session对象不能共用，只能自己使用

**Transaction**

1 事务对象

Transaction tx = session.beginTransaction();

2 事务提交和回滚方法

tx.commit();

tx.rolback();

3 事务概念

原子性、一致性、隔离性、持久性

# 使用Hibernate完成CRM客户管理中的查询所有客户操作

## 2.1需求描述

CRM系统中客户信息管理模块功能包括（crud）：

●新增客户信息

●客户信息查询

●修改客户信息

●删除客户信息

## 2.2实体类的编写规则

1 实体类里面属性私有的

2 私有属性使用公开的set和get方法操作

3 要求实体类有属性作为唯一值（一般使用id值）

4 实体类属性建议不使用基本数据类型，使用基本数据类型对应的包装类

（1）八个基本数据类型对应的包装类

- int – Integer

- char—Character、

- 其他的都是首字母大写 比如 double – Double

（2）比如 表示学生的分数，假如 int score;

- 比如学生得了0分 ，int score = 0;

- 如果表示学生没有参加考试，int score = 0;不能准确表示学生是否参加考试

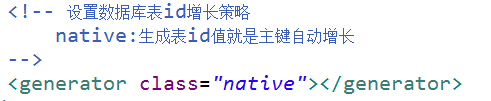
* 解决：使用包装类可以了， Integer score = 0，表示学生得了0分，

表示学生没有参加考试，Integer score = null;

## 2.3Hibernate主键生成策略

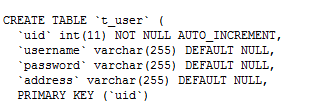
1 hibernate要求实体类里面有一个属性作为唯一值，对应表主键，主键可以不同生成策略

2 hibernate主键生成策略有很多的值

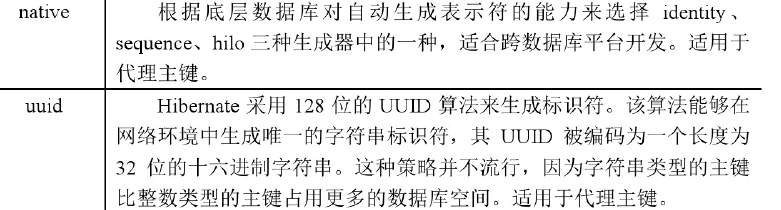


3 在class属性里面有很多值

（1）native： 根据使用的数据库帮选择哪个值



（2）uuid：之前web阶段写代码生成uuid值，hibernate帮我们生成uuid值



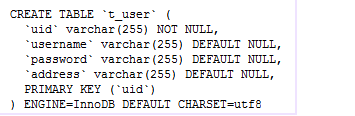
4 演示生成策略值 uuid

（1）使用uuid生成策略，实体类id属性类型 必须 字符串类型



（2）配置部分写uuid值，将原表删除，再运行一次



****

****

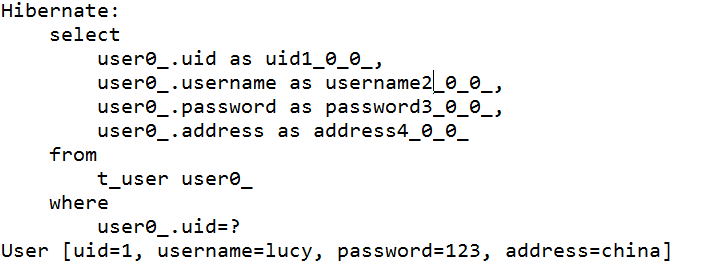
## 2.4对实体类的操作

### 2.4.1.Crud操作

#### 2.4.1.1根据id查询

调用session里面的get方法实现

**public static void** testGet(){  
 *//1 调用工具类得到sessionFactory* SessionFactory sessionFactory = HibernateUtils.*getSessionFactory*();  
 *//2 获取session* Session session = sessionFactory.openSession();  
 *//3 开启事务* Transaction tx = session.beginTransaction();  
 *//4 根据id查询  
 //调用session里面的get方法  
 //第一个参数：实体类的class  
 //第二和参数：id值* User user = session.get(User.**class**,1);  
 System.***out***.println(user);  
 *//5 提交事务* tx.commit();  
 *//6 关闭* session.close();  
 sessionFactory.close();  
}

****

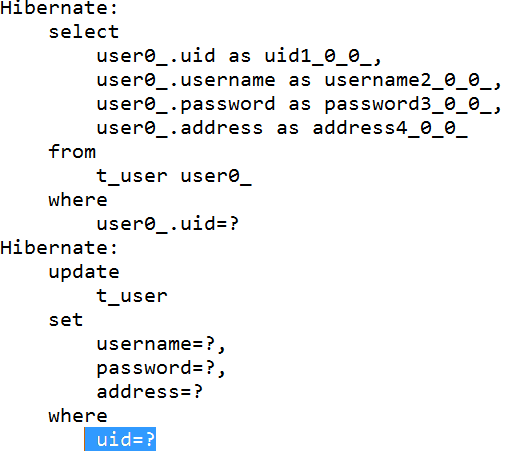
### 2.4.1.2修改操作

1 根据id查询出对象

2 用对象里的set方法修改字段值

3 更新

**public static void** testUpdate(){  
 *//1 调用工具类得到sessionFactory* SessionFactory sessionFactory = HibernateUtils.*getSessionFactory*();  
 *//2 获取session* Session session = sessionFactory.openSession();  
 *//3 开启事务* Transaction tx = session.beginTransaction();  
 *//4 修改操作  
 //4.1根据id进行查询* User user = session.get(User.**class**,2);  
 *//4.2向返回的user对象里面设置修改后的值* user.setUsername(**"东方不败"**);  
 *//4.3调用session的方法update修改  
 //执行过程：到user对象里面* session.update(user);  
 *//5 提交事务* tx.commit();  
 *//6 关闭* session.close();  
 sessionFactory.close();  
}

****

### 2.4.1.3删除操作

1 根据id查询出对象

2 调用session的delete方法删除对象

**public static void** testDelete(){  
 *//1 调用工具类得到sessionFactory* SessionFactory sessionFactory = HibernateUtils.*getSessionFactory*();  
 *//2 获取session* Session session = sessionFactory.openSession();  
 *//3 开启事务* Transaction tx = session.beginTransaction();  
 *//4 删除操作  
 //第一种 根据id查询对象  
// User user = session.get(User.class,2);  
// session.delete(user);  
  
 //第二种* User user = **new** User();  
 user.setUid(3);  
 session.delete(user);  
  
  
 *//5 提交事务* tx.commit();  
 *//6 关闭* session.close();  
 sessionFactory.close();  
 }

### 2.4.2实体类对象状态

1实体类有三种状态

（1） 瞬时态：对象里面没有id值，对象与session没有关联

User u = new User();

u.SetUsername(“jack”);

u.setPassword(“123”);

u.setAddress(“China”);

（2） 持久态：对象里面有id值，对象与session关联

User user = Session.get(User.class,1);

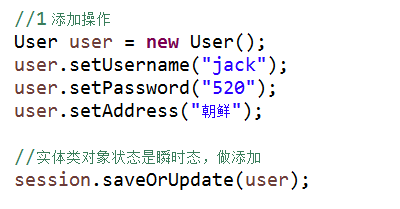
（3） 托管态：对象有id值，对象与session没有关联

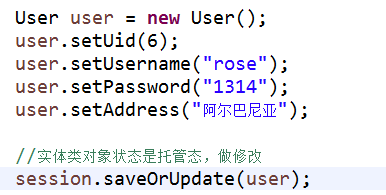
User user = new User();

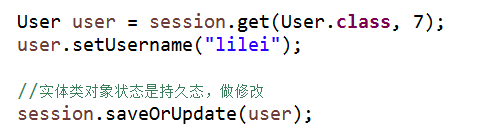
User.setUid(3);

2 演示操作实体类对象的方法

（1）saveOrUpdate方法：实现添加、实现修改

****

****

****

## 2.5Hibernate的一级缓存

### 2.5.1什么是缓存

数据存到数据库里面，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。

（1）把数据存到内存里面，不需要使用流方式，可以直接读取内存中数据

（2）把数据放到内存中，提供读取效率

### 2.5.2Hibernate一级缓存特性

1 hibernate框架中提供很多优化方式，hibernate的缓存就是一个优化方式

2 hibernate缓存特点：

第一类 hibernate的一级缓存

（1）hibernate的一级缓存默认打开的

（2）hibernate的一级缓存使用范围，是session范围，从session创建到session关闭范围

（3）hibernate的一级缓存中，存储数据必须 持久态数据

第二类 hibernate的二级缓存

（1）目前已经不使用了，替代技术 redis

（2）二级缓存默认不是打开的，需要配置

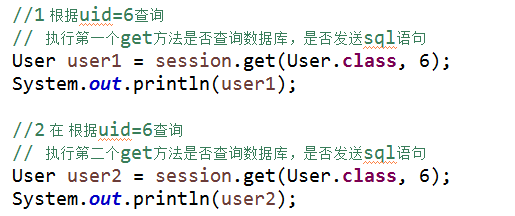
（3）二级缓存使用范围，是sessionFactory范围

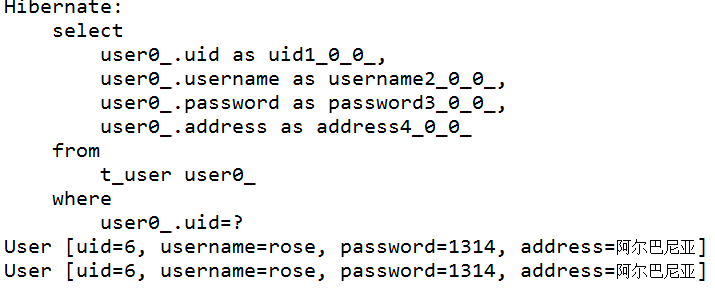
### 2.5.3验证一级缓存存在

1 验证方式

（1）首先根据uid=1查询，返回对象

（2）其次再根据uid=1查询，返回对象

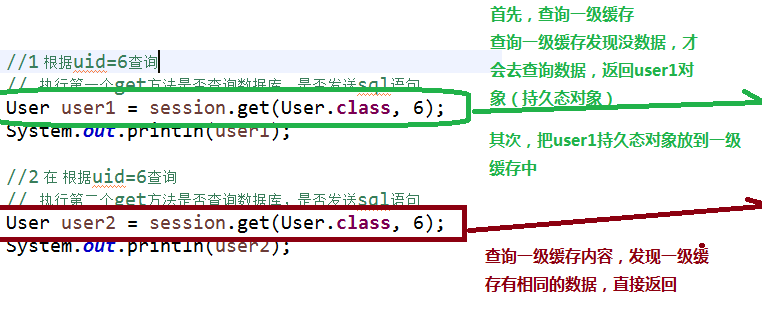




第一步执行get方法之后，发送sql语句查询数据库

第二个执行get方法之后，没有发送sql语句，查询一级缓存内容

### 一级缓存执行过程

****

## 2.6Hibernate事务操作

### 2.6.1事务相关概念

1 什么是事务

2 事务特性

3 不考虑隔离性产生问题

（1）脏读

（2）不可重复读

（3）虚读

1. 设置事务隔离级别

mysql默认隔离级别 repeatable read

### 2.6.2事务代码规范写法

1 代码结构

try {

开启事务

提交事务

}catch() {

回滚事务

}finally {

关闭

}

*//事务代码规范* **public static void** testTX(){  
 Transaction tx = **null**;  
*// SessionFactory sessionFactory = null;* Session session = **null**;  
 **try**{  
*// sessionFactory = HibernateUtils.getSessionFactory();  
// session = sessionFactory.openSession();  
  
 //与本地线程绑定的session* session = HibernateUtils.*getSessionObject*();  
  
 *//开启事务* tx = session.beginTransaction();  
  
 *//添加* User user = **new** User();  
 user.setUsername(**"小马1"**);  
 user.setAddress(**"美国"**);  
 user.setPassword(**"250"**);  
  
 session.save(user);  
 *//提交事务* tx.commit();  
 }**catch** (Exception e){  
 *//回滚事务* tx.rollback();  
 }  
 **finally** {  
 session.close();  
*// sessionFactory.close();* }  
 }

## 2.7Hibernate的api使用

### Query对象

1 使用query对象，不需要写sql语句，但是写hql语句

（1）hql：hibernate query language，hibernate提供查询语言，这个hql语句和普通sql语句很相似

（2）hql和sql语句区别：

- 使用sql操作表和表字段

- 使用hql操作实体类和属性

2 查询所有hql语句：

（1）from 实体类名称

3 Query对象使用

（1）创建Query对象

（2）调用query对象里面的方法得到结果

**public static void** testQuery(){  
 Transaction tx = **null**;  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
  
 *//开启事务* tx = session.beginTransaction();  
  
 *//1 创建Query对象* Query query = session.createQuery(**"from User"**);  
  
 *//2 调用Query对象里面的方法得到结果* List<User> list = query.list();  
  
 **for**(User user : list){  
 System.***out***.println(user);  
 }  
 *//提交事务* tx.commit();  
 }**catch** (Exception e){  
 *//回滚事务* tx.rollback();  
 }  
 **finally** {  
 *//关闭操作* session.close();  
 sessionFactory.close();  
 }  
}

# Hibernate的一对多的关联关系映射并操作

## 3.1表与表之间一对多的关系

一对多

（1）分类和商品关系，一个分类里面有多个商品，一个商品只能属于一个分类

（2）客户和联系人是一对多关系

- 客户：与公司有业务往来，百度、新浪、360

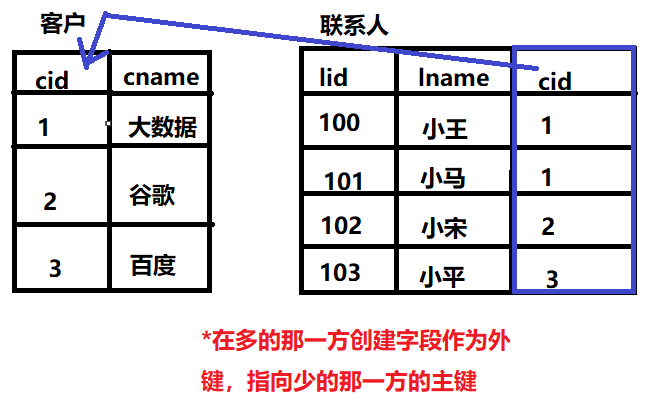
- 联系人：公司里面的员工，百度里面有很多员工，联系员工

\*\* 公司和公司员工的关系

- 客户是一，联系人是多

- 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

（3）一对多建表：通过外键建立关系



## 3.2Hibernate的一对多操作

### 3.2.1一对多映射配置

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 让两个实体类之间互相表示

（1）在客户实体类里面表示多个联系人

- 一个客户里面有多个联系人

**import** java.util.HashSet;  
**import** java.util.Set;  
**public class** Customer {  
 *//客户id* **private** Integer **cid**;  
 *//客户名称* **private** String **custName**;  
 *//客户级别* **private** String **custLevel**;  
 *//客户来源* **private** String **custSource**;  
 *//联系电话* **private** String **custPhone**;  
 *//手机* **private** String **custMobile**;  
 *//在客户实体类里面表示多个联系人，一个客户有多个联系人  
 //hibernate要求使用set集合表示多的数据* **private** Set<LinkMan> **linkManSet** = **new** HashSet<LinkMan>();  
 **public** Set<LinkMan> getLinkManSet() {  
 **return linkManSet**;  
 }  
 **public void** setLinkManSet(Set<LinkMan> linkManSet) {  
 **this**.**linkManSet** = linkManSet;  
 }  
 **public** Integer getCid() {  
 **return cid**;  
 }  
 **public void** setCid(Integer cid) {  
 **this**.**cid** = cid;  
 }  
 **public** String getCustName() {  
 **return custName**;  
 }  
 **public void** setCustName(String custName) {  
 **this**.**custName** = custName;  
 }  
 **public** String getCustLevel() {  
 **return custLevel**;  
 }  
 **public void** setCustLevel(String custLevel) {  
 **this**.**custLevel** = custLevel;  
 }  
 **public** String getCustSource() {  
 **return custSource**;  
 }  
 **public void** setCustSource(String custSource) {  
 **this**.**custSource** = custSource;  
 }  
 **public** String getCustPhone() {  
 **return custPhone**;  
 }  
 **public void** setCustPhone(String custPhone) {  
 **this**.**custPhone** = custPhone;  
 }  
 **public** String getCustMobile() {  
 **return custMobile**;  
 }  
 **public void** setCustMobile(String custMobile) {  
 **this**.**custMobile** = custMobile;  
 }  
}

（2）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

**public class** LinkMan {  
 **private** Integer **lkm\_id**;  
 **private** String **lkm\_name**;  
 **private** String **lkm\_gender**;  
 **private** String **lkm\_phone**;  
  
 *// 再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* **private** Customer **customer**;  
  
 **public** Customer getCustomer() {  
 **return customer**;  
 }  
  
 **public void** setCustomer(Customer customer) {  
 **this**.**customer** = customer;  
 }  
  
 **public** Integer getLkm\_id() {  
 **return lkm\_id**;  
 }  
  
 **public void** setLkm\_id(Integer lkm\_id) {  
 **this**.**lkm\_id** = lkm\_id;  
 }  
  
 **public** String getLkm\_name() {  
 **return lkm\_name**;  
 }  
  
 **public void** setLkm\_name(String lkm\_name) {  
 **this**.**lkm\_name** = lkm\_name;  
 }  
  
 **public** String getLkm\_gender() {  
 **return lkm\_gender**;  
 }  
  
 **public void** setLkm\_gender(String lkm\_gender) {  
 **this**.**lkm\_gender** = lkm\_gender;  
 }  
  
 **public** String getLkm\_phone() {  
 **return lkm\_phone**;  
 }  
  
 **public void** setLkm\_phone(String lkm\_phone) {  
 **this**.**lkm\_phone** = lkm\_phone;  
 }  
}

第三步 配置映射关系

（1）一般一个实体类对应一个映射文件

（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

- 在客户映射文件中，表示所有联系人

<**class name="entity.Customer" table="t\_Customer"**>  
 *<!-- 2配置实体类id和表id对应  
 hibernate要求实体类有一个属性唯一值  
 hibernate要求有字段作为唯一值  
 -->  
 <!-- id标签  
 name属性：实体类里面id属性名称  
 column属性：生成表字段名称  
 -->* <**id name="cid" column="cid"**>  
 *<!-- 设置数据库表id的增长策略  
 native：生成表id值就是主键自动增长  
 -->* <**generator class="native"**></**generator**>  
 </**id**>  
 *<!-- 配置其他属性和表字段对应  
 name属性：实体类属性名称  
 column：生成表字段名称  
 -->* <**property name="custLevel" column="custLevel"**></**property**>  
 <**property name="custMobile" column="custMobile"**></**property**>  
 <**property name="custName" column="custName"**></**property**>  
 <**property name="custPhone" column="custPhone"**></**property**>  
 <**property name="custSource" column="custSource"**></**property**>  
  
 *<!-- 在客户映射文件中，表示所有联系人  
 使用set标签表示所有联系人  
 set标签里面有name属性：  
 属性值写在客户实体类里面表示联系人的set集合名称  
  
 inverse属性默认值：false不放弃维护  
 true表示放弃关系维护  
 -->* <**set name="linkManSet" inverse="true"**>  
*<!-- 一对多建表，有外键-->  
<!-- hibernate机制：双向维护外键，在一和多那一方都配置外键-->* <**key column="clid"**></**key**>  
*<!-- 客户所有的联系人，class里面写联系人实体类全路径-->* <**one-to-many class="entity.LinkMan"**/>  
 </**set**>  
 </**class**>

- 在联系人映射文件中，表示所属客户

<**class name="entity.LinkMan" table="t\_LinkMan"**>  
 *<!-- 2配置实体类id和表id对应  
 hibernate要求实体类有一个属性唯一值  
 hibernate要求有字段作为唯一值  
 -->  
 <!-- id标签  
 name属性：实体类里面id属性名称  
 column属性：生成表字段名称  
 -->* <**id name="lkm\_id" column="lkm\_id"**>  
 *<!-- 设置数据库表id的增长策略  
 native：生成表id值就是主键自动增长  
 -->* <**generator class="native"**></**generator**>  
 </**id**>  
 *<!-- 配置其他属性和表字段对应  
 name属性：实体类属性名称  
 column：生成表字段名称  
 -->* <**property name="lkm\_gender" column="lkm\_gender"**></**property**>  
 <**property name="lkm\_name" column="lkm\_name"**></**property**>  
 <**property name="lkm\_phone" column="lkm\_phone"**></**property**>  
  
*<!-- 表示联系人所属的客户-->  
<!-- name属性：因为在联系人实体类使用customer对象表示，写customer名称-->  
<!-- class属性：customer全路径-->  
<!-- column属性：外键名称-->* <**many-to-one name="customer" class="entity.Customer" column="clid"**></**many-to-one**>  
  
 </**class**>

第四步 创建核心配置文件，把映射文件引入到核心配置文件中

*<?***xml version="1.0" encoding="UTF-8"***?>***<!DOCTYPE hibernate-configuration PUBLIC  
 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  
 "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd"*>***<**hibernate-configuration**>  
 <**session-factory**>  
 *<!-- 第一部分：配置hibernate信息 必须的-->* <**property name="hibernate.connection.driver\_class"**>com.mysql.jdbc.Driver</**property**>  
 <**property name="hibernate.connection.url"**>jdbc:mysql:///hibernate\_day03</**property**>  
 <**property name="hibernate.connection.username"**>root</**property**>  
 <**property name="hibernate.connection.password"**>258079</**property**>  
 *<!-- 第二部分：配置hibernate的信息 可选的-->  
 <!-- 输出底层sql语句-->* <**property name="hibernate.show\_sql"**>true</**property**>  
 *<!-- 输出底层sql语句格式-->* <**property name="hibernate.format\_sql"**>true</**property**>  
 *<!-- hibernate帮我们创建表，需要配置之后  
 update：如果已经有表，更新，如果没有表，创建  
 -->* <**property name="hibernate.hbm2ddl.auto"**>update</**property**>  
 *<!-- 配置数据库的方言  
 在MySQL里实现分页，关键字limit，只能使用MySQL里面  
 在Oracle里实现rownum  
 让hibernate框架实现不同的自己特有的语句  
 -->* <**property name="hibernate.dialect"**>org.hibernate.dialect.MySQL5Dialect</**property**>  
  
 *<!-- 在hibernate核心配置文件中配置-->* <**property name="hibernate.current\_session\_context\_class"**>thread</**property**>  
 *<!-- 第三部分：把映射文件放到核心配置文件中 必须的-->* <**mapping resource="entity/Customer.hbm.xml"**/>  
 <**mapping resource="entity/LinkMan.hbm.xml"**/>  
  
 <**mapping resource="manytomany/Role.hbm.xml"**/>  
 <**mapping resource="manytomany/User.hbm.xml"**/>  
  
 </**session-factory**>  
</**hibernate-configuration**>

### 3.2.2一对多的联级操作

#### 一对多的联级保存

- 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

- 在客户映射文件里面set标签进行配置

****

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

//演示一对多级联保存

**public** **void** testAddDemo2() {

SessionFactory sessionFactory = **null**;

Session session = **null**;

Transaction tx = **null**;

**try** {

//得到sessionFactory

sessionFactory = HibernateUtils.*getSessionFactory*();

//得到session

session = sessionFactory.openSession();

//开启事务

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("百度");

customer.setCustLevel("普通客户");

customer.setCustSource("网络");

customer.setCustPhone("110");

customer.setCustMobile("999");

LinkMan linkman = **new** LinkMan();

linkman.setLkm\_name("小宏");

linkman.setLkm\_gender("男");

linkman.setLkm\_phone("911");

//2 把联系人放到客户里面

customer.getSetLinkMan().add(linkman);

//3 保存客户

session.save(customer);

//提交事务

tx.commit();

}**catch**(Exception e) {

tx.rollback();

}**finally** {

session.close();

//sessionFactory不需要关闭

sessionFactory.close();

}

}

#### 一对多的联级删除

1 删除某个客户，把客户里面所有的联系人删除

2 具体实现

第一步 在客户映射文件set标签，进行配置

（1）使用属性cascade属性值 delete

****

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delete方法删除

*//演示一对多级联删除***public static void** testDeleteDemo(){  
 Transaction tx = **null**;  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
  
 *//开启事务* tx = session.beginTransaction();  
  
 *//1 根据id查询客户对象* Customer customer = session.get(Customer.**class**,3);  
 *//2 调用方法删除* session.delete(customer);  
  
  
 *//提交事务* tx.commit();  
 }**catch** (Exception e){  
 *//回滚事务* tx.rollback();  
 }  
 **finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

#### 一对多修改操作

*//演示一对多修改操作***public static void** testUpdate(){  
 Transaction tx = **null**;  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
  
 *//开启事务* tx = session.beginTransaction();  
  
  
 *//1 根据id查询lucy联系人，根据id查询百度的客户* Customer baidu = session.get(Customer.**class**, 4);  
 LinkMan lucy = session.get(LinkMan.**class**, 2);  
 *//2 设置持久态对象值  
 //把联系人放到客户里面* baidu.getLinkManSet().add(lucy);  
 lucy.setCustomer(baidu);  
  
 session.update(baidu);  
 *//提交事务* tx.commit();  
 }**catch** (Exception e){  
 *//回滚事务* tx.rollback();  
 }  
 **finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

## 3.3Hibernate的多对多操作

### 3.3.1多对多的映射配置

第一步 创建实体类，用户和角色

第二步 让两个实体类之间互相表示

（1）一个角色有多个用户，使用set集合

**import** java.util.HashSet;  
**import** java.util.Set;  
  
**public class** Role {  
 **private** Integer **role\_id**; *//角色id* **private** String **role\_name**; *//角色名称* **private** String **role\_memo**; *//角色描述  
  
 //一个角色有多个用户* **private** Set<User> **userSet** = **new** HashSet<User>();  
  
 **public** Set<User> getUserSet() {  
 **return userSet**;  
 }  
  
 **public void** setUserSet(Set<User> userSet) {  
 **this**.**userSet** = userSet;  
 }  
  
 **public** Integer getRole\_id() {  
 **return role\_id**;  
 }  
  
 **public void** setRole\_id(Integer role\_id) {  
 **this**.**role\_id** = role\_id;  
 }  
  
 **public** String getRole\_name() {  
 **return role\_name**;  
 }  
  
 **public void** setRole\_name(String role\_name) {  
 **this**.**role\_name** = role\_name;  
 }  
  
 **public** String getRole\_memo() {  
 **return role\_memo**;  
 }  
  
 **public void** setRole\_memo(String role\_memo) {  
 **this**.**role\_memo** = role\_memo;  
 }  
}

1. 一个用户里面表示所有角色，使用set集合

**import** java.util.HashMap;  
**import** java.util.HashSet;  
**import** java.util.Set;  
  
**public class** User {  
 **private** Integer **user\_id**;  
 **private** String **user\_name**;  
 **private** String **user\_password**;  
  
 *//一个用户可以有多个角色* **private** Set<Role> **setRole** = **new** HashSet<Role>();  
  
 **public** Set<Role> getSetRole() {  
 **return setRole**;  
 }  
  
 **public void** setSetRole(Set<Role> setRole) {  
 **this**.**setRole** = setRole;  
 }  
  
 **public** Integer getUser\_id() {  
 **return user\_id**;  
 }  
  
 **public void** setUser\_id(Integer user\_id) {  
 **this**.**user\_id** = user\_id;  
 }  
  
 **public** String getUser\_name() {  
 **return user\_name**;  
 }  
  
 **public void** setUser\_name(String user\_name) {  
 **this**.**user\_name** = user\_name;  
 }  
  
 **public** String getUser\_password() {  
 **return user\_password**;  
 }  
  
 **public void** setUser\_password(String user\_password) {  
 **this**.**user\_password** = user\_password;  
 }  
}

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

- 在角色里面表示所有用户，使用set标签

请自行进行配置

第四步 在核心配置文件中引入映射文件

与之前一致

### 3.3.2多对多的联级操作

#### 多对多的联级保存

*//演示多对多级联保存*

**public static void** testSave(){  
 Transaction tx = **null**;  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
  
 *//开启事务* tx = session.beginTransaction();  
  
 *//添加两个用户，为每个用户添加两个角色  
 //1创建对象* User user1 = **new** User();  
 user1.setUser\_name(**"lucy"**);  
 user1.setUser\_password(**"123"**);  
  
 User user2 = **new** User();  
 user2.setUser\_name(**"mary"**);  
 user2.setUser\_password(**"456"**);  
  
 Role role1 = **new** Role();  
 role1.setRole\_name(**"总经理"**);  
 role1.setRole\_memo(**"总经理"**);  
  
 Role role2 = **new** Role();  
 role2.setRole\_name(**"秘书"**);  
 role2.setRole\_memo(**"秘书"**);  
  
 Role role3 = **new** Role();  
 role3.setRole\_name(**"保安"**);  
 role3.setRole\_memo(**"保安"**);  
  
 *//2 建立关系，把角色放到用户里面  
 // user1 -- r1/r2* user1.getSetRole().add(role1);  
 user1.getSetRole().add(role2);  
  
 *// user2 -- r2/r3* user2.getSetRole().add(role2);  
 user2.getSetRole().add(role3);  
  
 *//3 保存用户* session.save(user1);  
 session.save(user2);  
  
 *//提交事务* tx.commit();  
 }**catch** (Exception e){  
 *//回滚事务* tx.rollback();  
 }  
 **finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

#### 多对多的联级删除

*//演示多对多级联删除***public static void** testDelete(){  
 Transaction tx = **null**;  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
  
 *//开启事务* tx = session.beginTransaction();  
  
 User user = session.get(User.**class**,5);  
 session.delete(user);  
  
 *//提交事务* tx.commit();  
 }**catch** (Exception e){  
 *//回滚事务* tx.rollback();  
 }  
 **finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

#### 多对多第三张表的维护

*//演示维护第三张表***public static void** testTalble2(){  
 Transaction tx = **null**;  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
  
 *//开启事务* tx = session.beginTransaction();  
  
 *//让某个用户有某个角色  
 //让lucy有经纪人角色  
 //1查询lucy和经纪人* User lucy = session.get(User.**class**,6);  
 Role role = session.get(Role.**class**,9);  
  
 *//2把角色放到用户的set集合里面* lucy.getSetRole().remove(role);  
  
  
 *//提交事务* tx.commit();  
 }**catch** (Exception e){  
 *//回滚事务* tx.rollback();  
 }  
 **finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

# Hibernate的条件查询

## 4.1HQL查询

### 4.1.1查询所有

1 查询所有客户记录

（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

2 查询所有： from 实体类名称

*//演示hql查询所有*

**public static void** testSelect1() {  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 Transaction tx = **null**;  
 **try** {  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
 tx = session.beginTransaction();  
  
 *//1创建query对象* Query query = session.createQuery(**"from Customer"**);  
 *//2调用方法得到结果* List<Customer> list = query.list();  
 **for** (Customer customer : list) {  
 System.***out***.println(customer.getCid() + **"::"** + customer.getCustName());  
 }  
  
 tx.commit();  
 } **catch** (Exception e) {  
 tx.rollback();  
 } **finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

### 4.2.2条件查询

1 hql条件查询语句写法：

（1） from 实体类名称 where 实体类属性名称=? and实体类属性名称=?

from 实体类名称 where 实体类属性名称 like ?

*//演示条件查询***public static void** testSelect2() {  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 Transaction tx = **null**;  
 **try** {  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
 tx = session.beginTransaction();  
  
 *//1创建query对象* Query query = session.createQuery(**"from Customer c where c.cid=? and c.custName=?"**);  
  
 *//2设置条件值  
 //向？里面设置值  
 //setParamater方法两个参数  
 //第一个参数：int类型是？的值，？位置从0开始  
 //第二个参数：具体参数值  
 //设置第一个？值* query.setParameter(0, 1);  
 query.setParameter(1, **"百度"**);  
  
 *//3调用方法得到结果* List<Customer> list = query.list();  
 **for** (Customer customer : list) {  
 System.***out***.println(customer.getCid() + **"::"** + customer.getCustName());  
 }  
  
 tx.commit();  
 } **catch** (Exception e) {  
 tx.rollback();  
 } **finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

### 4.2.3模糊查询

*//演示模糊查询***public static void** testSelect3() {  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 Transaction tx = **null**;  
 **try** {  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
 tx = session.beginTransaction();  
  
 *//1创建query对象* Query query = session.createQuery(**"from Customer c where c.custName like ?"**);  
  
 *//2设置？的值  
 // %浪%* query.setParameter(0, **"%浪%"**);  
  
 *//3调用方法得到结果* List<Customer> list = query.list();  
 **for** (Customer customer : list) {  
 System.***out***.println(customer.getCid() + **"::"** + customer.getCustName());  
 }  
  
  
 tx.commit();  
 } **catch** (Exception e) {  
 tx.rollback();  
 } **finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

### 4.3.4排序查询

1 hql排序语句写法

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

*//演示排序查询***public static void** testSelect4() {  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 Transaction tx = **null**;  
 **try** {  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
 tx = session.beginTransaction();  
  
 *//1创建query对象* Query query = session.createQuery(**"from Customer order by cid desc"**);  
  
  
 *//2调用方法得到结果* List<Customer> list = query.list();  
 **for** (Customer customer : list) {  
 System.***out***.println(customer.getCid() + **"::"** + customer.getCustName());  
 }  
  
  
 tx.commit();  
 } **catch** (Exception e) {  
 tx.rollback();  
 } **finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

### 4.3.5分页查询

1 mysql实现分页

（1）使用关键字 limit实现

****

2 在hql中实现分页

（1）在hql操作中，在语句里面不能写limit，hibernate的Query对象封装两个方法实现分页操作

*//演示分页查询***public static void** testSelect5() {  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 Transaction tx = **null**;  
 **try** {  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
 tx = session.beginTransaction();  
  
 *//1创建query对象* Query query = session.createQuery(**"from Customer"**);  
  
 *//2设置分页的数据  
 //2.1设置开始的位置* query.setFirstResult(0);  
 *//2.2设置每页显示记录数* query.setMaxResults(3);  
  
  
 *//3调用方法得到结果* List<Customer> list = query.list();  
 **for** (Customer customer : list) {  
 System.***out***.println(customer.getCid() + **"::"** + customer.getCustName());  
 }  
  
  
 tx.commit();  
 } **catch** (Exception e) {  
 tx.rollback();  
 } **finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

### 4.3.6投影查询

1 投影查询：查询不是所有字段值，而是部分字段的值

2 投影查询hql语句写法：

（1）select 实体类属性名称1, 实体类属性名称2 from 实体类名称

（2）select 后面不能写 \* ，不支持的

*//演示投影查询***public static void** testSelect6(){  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 Transaction tx = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
 tx = session.beginTransaction();  
  
 *//1创建query对象* Query query = session.createQuery(**"select custName from Customer"**);  
  
  
  
 *//2调用方法得到结果* List<Customer> list = query.list();  
 **for** (Object object:list){  
 System.***out***.println(object);  
 }  
  
  
 tx.commit();  
 }**catch** (Exception e){  
 tx.rollback();  
 }**finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

### 4.3.7聚集函数的使用

1 常用的聚集函数

（1）count、sum、avg、max、min

2 hql聚集函数语句写法

（1）查询表记录数

- select count(\*) from 实体类名称

*//演示聚集函数使用* **public static void** testSelect7(){  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 Transaction tx = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
 tx = session.beginTransaction();  
  
 *//1创建query对象* Query query = session.createQuery(**"select count(\*) from Customer"**);  
  
 *//2调用方法得到结果  
 //query对象里面方法，直接返回对象形式* Object o = query.uniqueResult();  
  
 *//返回int类型(不能直接转成int类型）  
// int count = (int) o;  
  
 //首先把object变成long类型，再变成int类型* Long lobj = (**long**)o;  
 **int** count = lobj.intValue();  
  
 System.***out***.println(count);  
  
 tx.commit();  
 }**catch** (Exception e){  
 tx.rollback();  
 }**finally** {  
 session.close();  
 sessionFactory.close();  
 }  
 }

## 4.4HQL多表查询

### 4.4.1HQL内连接

1 内连接查询hql语句写法：以客户和联系人为例

（1）from Customer c inner join c.setLinkMan

返回list，list里面每部分是数组形式

2 演示迫切内连接

（1）迫切内连接和内连接底层实现一样的

（2）区别：使用内连接返回list中每部分是数组，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

*//演示hql内连接查询* **public static void** testInnerJoin(){  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 Transaction tx = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
 tx = session.beginTransaction();  
  
 *//1创建query对象  
 //内连接* Query query = session.createQuery(**"from Customer c inner join c.linkManSet"**);  
 *//迫切内连接  
// Query query = session.createQuery("from Customer c inner join fetch c.linkManSet");  
  
 //内连接的list每部分是数组形式  
 //迫切内连接返回的是对象形式* List list = query.list();  
 System.***out***.println(list);  
  
 tx.commit();  
 }**catch** (Exception e){  
 tx.rollback();  
 }**finally** {  
 session.close();  
 sessionFactory.close();  
 }  
 }

### 4.4.2HQL左外连接

1 左外连接hql语句：

（1）from Customer c left outer join c.setLinkMan

（2）迫切左外连接from Customer c left outer join fetch c.setLinkMan

2 左外连接返回list中每部分是数组，迫切左外连接返回list每部分是对象

*//演示hql左外连接查询* **public static void** testOuterJoin(){  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 Transaction tx = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
 tx = session.beginTransaction();  
  
 *//1创建query对象  
 //左外连接* Query query = session.createQuery(**"from Customer c left outer join c.linkManSet"**);  
 *//迫切左外连接  
// Query query = session.createQuery("from Customer c inner join fetch c.linkManSet");  
  
 //左外连接的list每部分是数组形式  
 //迫切左外连接返回的是对象形式* List list = query.list();  
  
  
 tx.commit();  
 }**catch** (Exception e){  
 tx.rollback();  
 }**finally** {  
 session.close();  
 sessionFactory.close();  
 }  
 }

## 4.5Hibernate的检索策略

### 4.5.1检索策略测概念

1 hibernate检索策略分为两类：

（1）立即查询：根据id查询，调用get方法，一调用get方法马上发送语句查询数据库

（2）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，只有得到对象里面的值时候才会发送语句查询数据库

*//演示检索策略* **public static void** testSelect2(){  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 Transaction tx = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
 tx = session.beginTransaction();  
  
 *//根据cid=1客户  
 //执行get方法之后，是否发送sql语句  
 //调用sql方法马上发送sql语句查询数据库  
// Customer customer = session.get(Customer.class, 1);  
//  
// System.out.println(customer.getCid());  
  
 //1 调用load方法之后，不会马上发送sql语句  
 //（1）返回对象里面只有id值  
  
 //2 得到对象里面不是id的其他值时候才会发送语句* Customer customer = session.load(Customer.**class**,2);  
 System.***out***.println(customer.getCid());  
 System.***out***.println(customer.getCustName());  
  
  
 tx.commit();  
 }**catch** (Exception e){  
 tx.rollback();  
 }**finally** {  
 session.close();  
 sessionFactory.close();  
 }  
 }

### 4.5.2批量抓取

1 查询所有的客户，返回list集合，遍历list集合，得到每个客户，得到每个客户的所有联系人

（1）上面操作代码，发送多条sql语句

2 在客户的映射文件中，set标签配置

1. batch-size值，值越大发送语句越少



*//演示批量抓取***public static void** testSelect3(){  
 SessionFactory sessionFactory = **null**;  
 Session session = **null**;  
 Transaction tx = **null**;  
 **try**{  
 sessionFactory = HibernateUtils.*getSessionFactory*();  
 session = sessionFactory.openSession();  
 tx = session.beginTransaction();  
  
 *//查询所有客户* Criteria criteria = session.createCriteria(Customer.**class**);  
 List<Customer> list = criteria.list();  
 *//得到每个客户里面所有联系人* **for**(Customer customer:list){  
 System.***out***.println(customer.getCid()+**"::"**+customer.getCustName());  
 Set<LinkMan> linkManSet = customer.getLinkManSet();  
 **for**(LinkMan linkMan:linkManSet){  
 System.***out***.println(linkMan.getLkm\_id()+**"::"**+linkMan.getLkm\_name());  
 }  
 }  
  
 tx.commit();  
 }**catch** (Exception e){  
 tx.rollback();  
 }**finally** {  
 session.close();  
 sessionFactory.close();  
 }  
}

观察查询语句的次数，可以发现，bach-size取值为10时，只进行了两次查询