# 入门步骤：添加

共十一步骤：

第一步 导入hibernate的jar包

1)导入bibernate-release-5.0.7.Final.zip中的lib\required文件夹下所有

Antlr-2.7.7.jar,dom4j-1.6.1.jar,Geronimo-jta\_1.1\_spec-1.1.1.jar

Hibernate-commons-annotations-5.0.1,hibernate-core-5.0.7,hibernate-jpa-2.1-api-1.0.0Jandex-2.0.0,javassist-3.18.1,jboss-logging-3.3.0

2)导入Hibernate-release-5.0.7.Final.zip中的lib\Jpa文件下只有一个的jar

Hibernate-entitymanager-5.0.7

3)其它 jar：因为使用Hibernate时候，有日志信息输出，hibernate本身么有日志输出的jar包。导入其它日志的jar包：

log4J-1.2.16,slf4j-api-1.6.1,slf4j-logj12-.17.2

第二步 创建实体类（JavaBean）要求实体类有一个属性是唯一要对应表中的主键

使用hibernate时候，不需要自己手动创建表，hibernate在核心配置文件配置好标签：

<property name="hibernate.hbm2ddl.auto">update</property>//配置此标签Hibernate会帮忙创建表

第三步 配置实体类和数据库表对应关系（映射关系）：类名要和表名一致，类中的属性要和表中字段一一对应。 不用自己创建表，Hibernate创建SessionFactory会创建

1）创建xml格式的配置文件：映射配置文件名称和位置么有固定要求，建议在实体类所在包里面创建：实体类名称.hbm.xml

2） 在配置文件中首先引入xml约束：它是dtd约束，配置实体类和表的映射关系。

下面是XML内容：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<!—1、 配置类和表对应-->

<class name="com.wwl.entity.User" table="t\_user">

<!—2、 配置属性对应主键id-->

<id name="uid" column="uid">

<!—3、设置数据库id增长策略-->

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

</id>

<!—4、 配置其它属性对应字段 -->

<property name="code" column="t\_code"></property>

<property name="password" column="t\_password"></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xml 与位置：必须工程类根目录，名称：必须hibernate.cfg.xml

2）Hibernate操作过程中，只会加载核心配置文件，其它配置文件不会加载（其它配置文件需要在核心配置文件中配置）

<?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>

<!-- 1、配置数据库信息 必须的-->

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

<property name="hibernate.connection.url">jdbc:mysql://localhost:3360/wwl</property>

<property name="hibernate.connection.username">wwl</property>

<property name="hibernate.connection.password">111111</property>

<!-- 2、配置hibernate优化信息，可选的 -->

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

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

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

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

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

<!-- 3、把配置的映射文件放到核心配置文件中 必须的 -->

<mapping resource="com/wwl/entity/User-hbm.xml"/>

</session-factory>

</hibernate-configuration>

5到11是代码部分：第五步 加载hibernate核心配置文件 hibernate-cfg.xml

创建对象configurati：方法configure()有重载方法：可以更改核心配置文件的路径（一般不建议这样做）

Configuration cfg = new Configuration();

cfg.configure();

第六步 通过configurati对象创建SessionFactory对象

读取hibernate核心配置文件内容，创建SessionFactory获得配置中的信息存入对象中

在此过程中，根据类中配置映射关系，会在数据库生成相对应的表

SessionFactory sessionF=cfg.buildSessionFactory();

第七步 使用SessiongFactory创建session对象,相当于JDBC的connection

第一种：线程不安全的

Session session=sessionF.openSession();

第二种：绑定本地线程，需在核心配置文件中配置

Session session =sessionFactory.getCurrentSession();

第八步 开启事务：必须的步骤，Hibernate需要在事务中操作，没有开启会报错

Transaction tx= session.beginTransaction();

第九步 写具体逻辑CRUD操作

//添加功能

User u= new User();

u.setCode("温文亮");

u.setPassword("123456");

session.save(u);

第十步 提交事务

tx.commit();

第十一步 关闭资源：正式环境不建议关闭sessionFactory的资源

session.close();

sessionF.close();

最后是简单的调用操作：new对象然后调用方法

# 概念

版本

Hibernate3.X，Hibernate4.x(过度版本，没什么用)，Hibernate5.X（现在学）

Hibernate【过冬；（动物）冬眠；（人等）避寒】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、扩展性强等特点，能够大大地简化程序的代码量，提高工作效率

什么是ORM思想

Hibernate是一个开放源代码的ORM（Object Relational【相关的，亲属的】 Mapping，对象关系映射）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

1、 hibernate使用orm思想对数据库进行CRUD操作（面向对象思想增删改查）

2、 javaBean：也叫实体类和数据库表是一一对应关系：

实体类名和数据库表名对应，实体类中的属性和表里面字段一一对应

不需要直接操作数据库表，而是操作表对应实体类对象

# 实体类对象状态

1、瞬时态：第一：对象中的主键属性没有赋值，第二：这个对象与session还没有关联。

比如：new了一个对象（这对象的id对应表中的主键），赋值的时候：id没有进行赋值，如下例子：

User u= new User();

u.setCode("温文亮");

u.setPassword("123456"); 这个对象u就是瞬时状态

2、托管态：第一：对象有id值，第二：对象与session没有关联，老师说这种用的不多

User u= new User();

u.setUid(3); 这个对象u就是托管态

3、持久态：第一：对象中的属性id有值，第二：对象跟session有关联

如下例子：id设置为2，User对象

User user=session.get(User.class, 2);这个对象user就是持久态

hibernate缓存，什么是缓存？

数据存到数据库里面，数据库本身是文件系统，java中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）把数据方法存储内存中，提供读取效率

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

Hibernate缓存特点：

第一类 hibernate的一级缓存

hibernate的一级缓存默认是打开的，

hibernate的一级缓存适用范围，是session范围从session创建到session关闭范围

hibernate的一级缓存中，存储数据都是：持久态数据

第二类 hibernate的二级缓存

1）目前没有人使用了，替代技术：radis

2）二级缓存默认不是打开的，需要配置

3）二级缓存适用范围：是sessionFactory范围

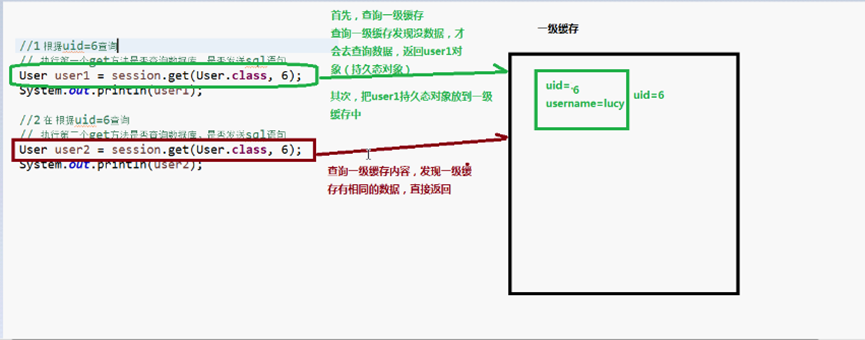
验证一级缓存存在

验证方式：debug方式

1、首先根据get(A.class,1)，返回对象

2、再次通过get(A.class,1)，返回对象：这次查询的是缓存中的

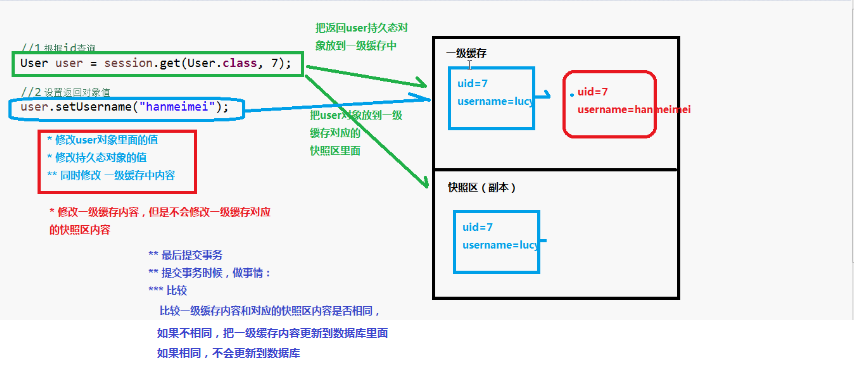
一级缓存的执行过程



一级缓存特性：持久态自动更新数据库



自动更新数据库原理：



Hibernate事务操作：都要居于开启事务的环境中操作

原子性：要么都成功，要么都失败

一致性：小马给小息转账，必须小马少1000，小息多1000，或者小马和小息都没变（失败）

隔离性：多个人操作同一个东西，

持久性：存入数据库了就是持久性了

不考虑隔离性产生问题：

1、脏读。2、不可重复读。3、虚读

# Hibernate绑定session

使用：sessionFactory.getCurrentSession()，绑定threadLocal本地安全线程：自己用不用担心安全问题，但系统是100万人用怎么保证session是唯一对象也就是是绝对单线程？

本地线程进行绑定就是唯一对象！

1、Session类似于jdbc的connection。之前web阶段学过threadLocal（以本地线程绑定）

2、Hibernate已帮实现与本地线程绑定session，底层就是threadLocal

配置方法：

1）在hibernate核心配置文件中配置

<!-- 绑定本地线程：底层就是用的threadLocal类事项的 -->

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

2）调用sessionFactory里面的方法得到

Session session=sessionFactory.getCurrentSession();

3）获取与本地线程绑定session时候，session.colse会报错:（因为绑定本地线程都是还在运行的，你关闭跟它绑定在一起的session就会造成冲突）所以就不需要手动关闭了

错误信息：org.hibernate.SessionException: Session was already closed

# 配置文件XML

## hibernate映射配置文件详解

1）创建xml格式的配置文件：映射配置文件名称和位置么有固定要求：建议在实体类所在包里面创建：实体类名称.hbm.xml ： User-hbm.xml

2）配置xml格式：在配置文件中首先引入xml约束：它是dtd约束

3）配置实体类和和表的映射关系,下面是xml内容：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

第一步、配置实体类和数据库表名对应

属性：name：实体类的全路径

属性：table：数据库表名称

注意：不用自己建表：table属性自己先写一个上去，后续hibernate会给你创建

<class name="com.wwl.entity.User" table="t\_user">

第二步、配置数据库主键id和实体类中的唯一属性对应关系

hibernate要求实体类有一个属性是唯一值

id标签： name： 类中的成员变量

column【列】：数据库表的主键字段名称

generator标签： 设置数据库id增长策略：如Mysql主键自动增长

主键增长策略6个值：incremnt、identity、sequence、native、uuid、assigncd

Incremnt：每添加一条数据，id加1

Identity：使用Mysql自动增长数据类型，如MysqL有。oracle不支持

Sequence：数据库支持序列，如Oracle支持，Mysql不支持

Assigncd

主要记住一下两个：

Native：根据底层数据库自动来选择Identity、Sequence、hilo，适合跨数据库平台开发

UUID：Hibernate采用128位的UUID算法来生成标识符，使用uuid生成策略，实体类id属性必须是String

<id name="uid" column="uid">

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

</id>

第三步、配置实体类中其它成员变量和数据库其它字段一一对应

如果忽略column属性会根据name的值创建表字段名称、

<property name="code" column="t\_code"></property>

<property name="password" column="t\_password"></property>

第四步、级联关联操作：看级联操作

</class>

</hibernate-mapping>

**总结：**

**1、id标签和property标签中的column属性是可以省略的**

**如：<property name="code" column="t\_code"></property>可以写成：**

**<property name="code" ></property>这样写在表中自动匹配name中的值**

**2、property标签type属性，设置生成表字段的类型不用写，自动对应类型**

## Hibernate核心配置文件详解

注意容易忘记不在：<session-factory>下写；配置3步骤

<hibernate-configuration>

<session-factory>

第一步 配置连接数据库信息 必须的

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

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/wwl</property>

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

<property name="hibernate.connection.password">123456</property>

第二步 配置hibernate优化信息，可选的

<!-- 控制台显示底层sql语句 -->

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

<!-- 对底层的sql语句进行格式化：也就是显示的sql语句排列好给你看 -->

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

<!-- hibernate帮创建表，需要配置下面:hibernate.hbm2ddl.auto

update:如果已经有表，更新，如果没有，创建。这配置在获取工厂类执行的-->

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

<!-- 配置数据库方言:如：

在mysql里面实现分页关键字limit，只能msyql使用，在oracle数据库，是rownum

让hibernate框架识别不同数据库的自己特有的语句 -->

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

<!-- 绑定本地线程：底层就是用的threadLocal类实现的，配置了这条可以使用

sessionFactory.getCurrentSession(); -->

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

第三步 把配置好的映射文件放到核心配置文件中 必须的

<mapping resource="com/wwl/entity/User-hbm.xml"/>

</session-factory>

</hibernate-configuration>

# Hibernate核心api

1. Configuration：用来找核心配置文件：hibernate-cfg.xml

cfg.configure();有重载的方法：可以指定路径下找那个核心配置文件：一般都不这样做

Configuration cfg = new Configuration();

cfg.configure();

2、SessionFactory

使用configuration对象创建sessionFactory对象

创建sessionfactory过程中做的事情：

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

创建sessionFactory过程中，这个过程特别耗费资源的：

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

具体实现：写一个工具类，类中写静态代码块实现，静态代码块在类加载的时候执行一次，所以就不进

行关闭资源了

3、Session

1）、session类似于jdbc中的connection类

2）、调用session里面不同的方法实现crud操作：

添加save、删除delte、修改update、更具id查询get方法

3）、session对象单线程对象：sessionFactory.getCurrentSession()

单线程：对象不能共用，只能自己使用

4、Transaction

原子性：要么都成功，要么都失败

一致性：小马给小息转账，必须小马少1000，小息多1000，或者小马和小息都没变（失败）

隔离性：多个人操作同一个东西，

持久性：存入数据库了就是持久性了

# CRUD操作

修改

步骤：通过id进行修改：先查询获得对象（持久化态），再进行类.set，调用Update方法

//自己写工具类得到sessionFactory

SessionFactory sessionFactory=HibernateUtil.getSessionFactory();

Session session=sessionFactory.openSession();

//开启事务

Transaction tx=session.beginTransaction();

//修改操作：先根据id查询,

User s=session.get(User.class, 2);

//2,向自己的值进行set操作

s.setCode("凉凉");

//3,调用update方法

**session.update(s);**

//提交事务

tx.commit();

//关闭资源

session.close();

sessionFactory.close();

注意：不要自己创建User空对象，进行设置值，一旦这样做，有些属性你没有进行设置表里给变成null

}

保存

Save（），saveOrUpdate（）；步骤：创建要保存对象，进行每个属性赋值

删除

两种方法

1、通过id进行删除：先查询获得对象（持久化态），再调用delete方法

2、创建要删除的对象，把id值进行赋值,调用delete方法，这方式很少用

//自己写工具类得到sessionFactory

SessionFactory sessionFactory=HibernateUtil.getSessionFactory();

Session session=sessionFactory.openSession();

//开启事务

Transaction tx=session.beginTransaction();

//删除操作：

//第一种：先根据id先查询出来对象

User s=session.get(User.class, 3);

//第二种：如果确定有了这个数据，创建这个对象，传进id值

/\*User u= new User();

u.setUid(3);\*/

//调用delete方法

session.delete(s);

//提交事务

tx.commit();

//关闭资源

session.close();

sessionFactory.close();

查询

通过id获取一条类的对象数据：get（User.class,3）;

SessionFactory sessionFactory=HibernateUtil.getSessionFactory();

Session session=sessionFactory.openSession();

// 开启事务

Transaction tx= session.beginTransaction();

// 写具体逻辑crud操作

//通过id进行查询

User s=session.get(User.class, 2);

# 级联操作

## 一对多

**一对多级联保存操作：三步**

保存‘一’和‘多’的表，根据外键进行关联

第一步 在’一’配置的映射文件xml中配置set标签：

set标签中cascade属性使用值 save-update，如有多个值用逗号隔开：save-update,delete

<!-- 配置一对多：name:本类中的Set名称，cascade：batch-size：为了减少select语句数目，可以设置元素的batch-size属性：值越大输出语句越少-->

<set name="person" cascade="save-update,delete" batch-size="100">

<!-- 此column对应另一个表的外键名称-->

<key column="cid"></key>

<!-- 多的一方：类的全路径名称 -->

<one-to-many class="com.wwl.entity.Person"/>

</set>

第二步：在‘多’的映射文件配置many-to-one标签，它cascade属性也要save-update

注意：第二步的配置老师是没有配置的老师没问题的，我的有问题（可能版本问题我的比较高）报错：

object references an unsaved transient instance - save the transient instance before flushing：意思是有一个不是持久态不让保存

<!—属性：name：本类维护的对象名称，class：是”少”的一方类的全路径名 column：本表外键cid名称 -->

<many-to-one name="customer" class="com.wwl.entity.Customer" column="cid"

cascade="save-update"></many-to-one>

第三步 在代码中直接保存‘一’的即可：它会关联到多的表进行保存操作，看代码

//级联保存

Customer：客户公司名称（一），Person：此公司的联系人（多）

Customer c= new Customer();

c.setcName("传智播客");

c.setType("普通客户");

c.setAddress("广州");

Person p = new Person();

p.setpName("小红");

p.setPhone("183568554");

p.setGender("女");

//把联系人放到我们客户里面：等于把‘多’的放到‘少’的就可以了

c.getPerson().add(p);

//保存客户:保存少的就可以了

session.save(c);

底层保存原理：不用进行映射配置

//一对多保存hibernate保存原理

public static void testMan(){

Session session=HibernateUtil.getMySession();

Transaction tx = session.beginTransaction();

try{

Customer c= new Customer();

c.setcName("百度");

c.setType("钻石客户");

c.setAddress("深圳");

Person p = new Person();

p.setpName("梁龙");

p.setPhone("153625845");

p.setGender("男");

c.getPerson().add(p);

p.setCustomer(c);

//经测试：建议‘一’的先保存，‘多’的后保存:hibernate操作数据库：先存Customer，后存Person，根据pid进行修改cid；

//‘多’的先保存，‘一’的后保存：先存Person，后存Customer，修改根据pid修改全部数据，然后又根据pid修改cid

session.save(p);

session.save(c);

tx.commit();

}catch(Exception e){

tx.rollback();

}finally{

}

**一对多级联删除**

删除’一’的一方，把关联‘多’的也进行删除

具体实现

第一步 在’少’的映射文件set标签，进行配置：

cascade属性使用值 delete，如有多个值用逗号隔开：save-update,delete

<set name="person" cascade="save-update,delete">

第二步 在代码中直接删除‘少’的即可

根据id查询对象，调用session里面delete方法删除

//级联删除

Session session=HibernateUtil.getMySession();

Transaction tx = session.beginTransaction();

try{

//根据id查询客户对象

Customer c=session.get(Customer.class, 3);

session.delete(c);

tx.commit();

}catch(Exception e){

tx.rollback();

}finally{

}

Hibernate级联删除的原理步骤

先查‘少’再查‘多’的表，然后把‘多’的外键设置为null，删除‘多’表再删‘少’的表

**一对多级联修改**

//级联修改

public static void testUpdate2(){

Session session=HibernateUtil.getMySession();

Transaction tx = session.beginTransaction();

try{

//根据id查询客户对象

Customer c=session.get(Customer.class, 15);

Person p= session.get(Person.class, 18);

//把联系人放到客户里面

c.getPerson().add(p);

//把客户放到联系人里面

p.setCustomer(c);

tx.commit();

}catch(Exception e){

tx.rollback();

}finally{

}

引出问题：

Incerse属性

本身进行修改了全部（包括外键），最后又通过id修改外键一次

为什么hibernate重复的修改了两次操作呢？

因为hibernate双向维护外键，在‘少’和‘多’里面都需要维护外键，修改‘少’时候修改一次外键，修改‘多’时候也修改一次外键，造成效率问题

解决方式：可以让另一方放弃维护外键：让‘少’的一方放弃维护外键

为什么让‘少’的一方放弃呢？：一个国家有总统，国家有很多人，总统不认识国家所有人，国家所有人都认识总统

具体实现

在放弃维护映射文件中，进行配置，在set标签上使用Incerse属性值为true

<set name="person" inverse="true">

## 多对多

多对多映射配置

第一步 创建实体类：让两个实体类之间互相表示：如角色和用户

（1） 用户里面表示所有角色，使用set集合

（2） 一个角色有多个用户，使用set集合

第二步 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

<hibernate-mapping>

<class name="com.wwl.entity.User" table="t\_user">

<id name="uid" column="uid">

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

</id>

<property name="code" column="t\_code"></property>

<property name="password" column="t\_password"></property>

<!-- 在用户里面表示所有角色，使用set标签

neme属性：角色set集合名称

table属性：第三张表名称-->

<set name="role" table="user\_role">

<!-- key标签里面配置

配置当前映射文件在第三张表外键名称 -->

<key column="userid"></key>

<!-- class:角色实体类全路径名

column 角色在第三张表外键名称 -->

<many-to-many class="com.wwl.entity.Role" column="roleid"></many-to-many>

</set>

</class>

</hibernate-mapping>

在角色里面表示所有用户，也使用set标签：一样的配置

第三步 在核心配置文件引入配置好的映射文件

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set标签进行配置。Cascade值save-update

<set name="role" table="user\_role" cascade="save-update">

第二步 写代码实现

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

//级联保存

public static void testMan3(){

Session session=HibernateUtil.getMySession();

Transaction tx = session.beginTransaction();

try{

User user= new User();

user.setCode("温文亮");

user.setPassword("男");

Role role = new Role();

role.setRole\_name("经理");

role.setRole\_memo("老大");

Role role2 = new Role();

role2.setRole\_name("秘书");

role2.setRole\_memo("辅助经理");

user.getRole().add(role);

user.getRole().add(role2);

//保存客户:这个是1个人有两个角色保存操作

session.save(user);

tx.commit();

}catch(Exception e){

tx.rollback();

}finally{

}

}

多对多级联删除（了解）会造成问题

第一步 在set标签进行配置， cascade值delete

第二步 删除用户

这个有问题的，不写了：如2个用户有相同的角色，删除第一个用户，会删除它全部数据，连角色也删除掉了，第2个就没了角色。

维护第三张表

用户和角色是多对多关系，维护关系是通过第三张表维护

1、 让某个用户有某个角色：这样删除不会有问题

第一步 根据 id 查询用户和角色

第二步 把角色放到用户里面

把角色对象放到用户set集合中

Session session=HibernateUtil.getMySession();

Transaction tx = session.beginTransaction();

try{

//持久态会进行更改,不用调用什么

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

Role role=session.get(Role.class, 3);

user.getRole().add(role);

tx.commit();

}catch(Exception e){

tx.rollback();

}finally{

}

2、 让某个用户没有某个角色：这样删除不会有问题

第一步 根据 id 查询用户和角色

第二步 从用户里面把角色去掉

从set里面移除

Session session=HibernateUtil.getMySession();

Transaction tx = session.beginTransaction();

try{

//持久态会进行更改,不用调用什么方法

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

Role role=session.get(Role.class, 3);

user.getRole().remove(role);

tx.commit();

}catch(Exception e){

tx.rollback();

}finally{

}

# Hibernate的三个查询对象

## Query对象

1使用query对象，不需要写sql语句而是写hql语句

（1） hql：hibernate query langnage hi贝尔纳特提供查询语言，这个hql语句和普通sql语句很相似

（2） hql和sql语句的区别：

使用sql操作表和表字段

使用hql操作实体类和属性

2、查询所有的hql语句

（1） from实体类名称

3、Query对象的使用

（1）创建Query对象

（2）调用Query中的方法

查询全部代码：

Session session=HibernateUtil.getMySession();//工具类获取session

Transaction tx=session.beginTransaction();

try{

Query query=session.createQuery("from User");//hql语句查询对象对应表全部数据

List<User> list=query.list();//返回对象集合

for(User u :list){

System.out.println(u.getPassword());

}

tx.commit();

}catch(Exception e){

e.printStackTrace();

tx.rollback();

}finally{

//使用了绑定本地线程的查询不用关闭资源了

}

## Criteria对象

1使用这个对象查询操作，但是使用这个对象时候，就不用写语句。直接调用方法实现

2实现过程

（1） 创建criteria对象

（2） 调用对象里面的方法得到结果：对象集合

查询对象全部数据代码

Session session=HibernateUtil.getMySession();

Transaction tx=session.beginTransaction();

try{

Criteria cr=session.createCriteria(User.class);

List<User> list=cr.list();

for(User l :list){

System.out.println(l.getCode());

}

tx.commit();

}catch(Exception e){

tx.rollback();

}finally{

}

## SQLQuery对象

1使用hibernate时候，调用底层sql实现，就是原生sql语句实现

2实现过程

（1） 创建criteria对象

（2） 在查询之前可以设置对象变为对象集合

sq.addEntity(User.class);

（3） 调用对象里面的方法得到结果（默认是数组集合，不是对象集合）

查询User对象对应表的全部数据

Session session=HibernateUtil.getMySession();

Transaction tx = session.beginTransaction();

try{

SQLQuery sq=session.createSQLQuery("select \* from t\_user");//sql语句

//告诉对象SQLQuery得到的是User对象，不然最后查询出来是数组

sq.addEntity(User.class);

List<User> list=sq.list();

for(User l :list){

System.out.println(l.getCode());

}

tx.commit();

}catch(Exception e){

e.printStackTrace();

tx.rollback();

}finally{

}

# Hibernate查询方式

## 1、OID查询

根据id查询某一条记录，返回对象：get()

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

## 2、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；

其实就是通过get获得对象后，对象有维护别的对象进行取出来就是了

try{

//根据id=1客户，再查询这个客户里面所有联系人

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

//获得person对象

Set<Person> person=customer.getPerson();

//遍历

for(Person p :person){

System.out.println(p.getpName());

}

tx.commit();

}catch(Exception e){

tx.rollback();

}finally{

}

## 3、Hql查询

Query对象。写hql语句实现查询

（1）hql：hibernate query langnage hi贝尔纳特提供查询语言，这个hql语句和普通sql语句很相似

（2）hql和sql语句的区别：

使用sql操作表和表字段

使用hql操作实体类和属性

创建Query对象，构造方法参数中写HQL语句

常用的hql语句

查询所有

Query query=session.createQuery("from User");

List<User> list=query.list();

for(User u :list){

System.out.println(u.getPassword());

}

tx.commit();

条件查询

Hql条件查询语句写法：

from 实体类名称 where 实体类属性名称=？ And实体类属性名称=？

Query query=session.createQuery("from Customer where cid=? and cName=?");

设置占位符值,JDBC从1开始，它是从0开始的

query.setParameter(0, 1);

query.setParameter(1, "百度");

List<Customer> customer=query.list();

for (Customer cust : customer) {

System.out.println(cust.getcName());

}

tx.commit();

模糊查询

from 实体类名称 where 实体类属性名称 like？

Query query=session.createQuery("from Customer where cName like ?");

//设置占位符值,JDBC从1开始，它是从0开始的

query.setParameter(0, "%百%");

List<Customer> customer=query.list();

for (Customer cust : customer) {

System.out.println(cust.getcName());

}

tx.commit();

排序查询

排序查询：asc（升序）desc（降序）

From 实体类名称 order by 实体类属性名称 asc/desc

Query query=session.createQuery("from Customer order by cid desc ");

分页查询 不同的

普通sql：select \* from customer limit 0,3 0是起始位置，3是显示几条

Hql不认识只有mysql的limit关键字：它是只有方法的：

Query query=session.createQuery("from Customer");

设置分页方法，起始和显示

设置开始位置

query.setFirstResult(0);//从0开始

设置每页显示几条

query.setMaxResults(2);

调用方法得到结果

List<Customer> customer=query.list();

for (Customer cust : customer) {

System.out.println(cust.getcName());

}

tx.commit();

投影查询：查的是部分，不是全部

Select cid from customer：select后不是\*号就是投影查询

投影查询hql语句写法

select 实体类属性名称1，实体类属性名称2. From实体类名称

select 后面不能写 \*，不支持的

try{

Query query=session.createQuery("select cName from Customer");

//注意得到的就不是对象了，泛型要Object或String

List<Object> customer=query.list();

for (Object cust : customer) {

System.out.println(cust);

}

tx.commit();

}catch(Exception e){

e.printStackTrace();

tx.rollback();

聚集函数使用

常用的聚集函数

Count（统计）、sum（相加）、avg（平均数）、max（最大）、min（最小）

Query query=session.createQuery("select count(\*) from Customer");

//得到唯一的值，调用list方法也可以但要遍历麻烦，有一个方法可以直接得到对象

Object obj=query.uniqueResult();

//要变成int必须先包装类Long才有方法，再进行转int,测试也可以强转int

Long c= (long) obj;

//int d =(int) c;//强转

int d=c.intValue();

System.out.println(d);

tx.commit();

多表查询

内连接查询:显示两边有联系的，两种写法：

（1）Select \* from customer c,person p where c.cid=l.pid;

（2）Select \* from customer c inner join person p on c.cid=l.pid;

内连接也要看场景使用的，如果右边有多条记录对应左边一条，左边就会有重复的信息，相反：如果左边有多条记录对应右边一条，右边就有重复信息，所以总有一边是没有重复的，自己掂量了。。。

左外连接：左边表所有，右边表关联的有，无的显示null

Select \* from customer c left outer join person p on c.cid=l.pid;

经测试：显示左边表所有，右边表显示信息要和昨天一一对应的才会显示，重复问题和内连接一样

右外连接：右边表所有，左边表关联的有，无的显示null

Select \* from customer c right outer join person p on c.cid=l.pid;

经测试：显示右边表所有，左边表显示信息要和昨天一一对应的才会显示，重复问题和内连接一样

Hql多表查询

内连接：from Customer c inner join c.person c.person：类变量set集合名称

Query query=session.createQuery("from Customer c inner join c.person");

List<Object[]> l=query.list();

//返回的是Object数组,有8个记录，两个对象需要判断

for (Object[] customer : l) {

for(Object c:customer){

if(c instanceof Customer){//判断类型然后强制转换

Customer v=(Customer) c ;

System.out.println(v.getcName());

}else{

Person p =(Person) c;

System.out.println(p.getpName());

迫切内连接：from Customer c inner join fetch c.person

迫切内连接和内连接底层实现一样的，使用内连接返回list中每部分是数组，迫切内连接返回时list每部分都是对象

Configuration ca=new Configuration().configure();

//获得工厂类

SessionFactory sessionFactory=ca.buildSessionFactory();

//sessionFactory.openSession();//session不安全的

Session session=sessionFactory.getCurrentSession();//绑定了本地线程的

Transaction tx=session.beginTransaction();

try{//创建对象

Query query=session.createQuery("from Customer c inner join fetch c.person");

//获得4条记录都是Customer对象

List<Customer> l=query.list();

for(Customer c:l){

Set<Person> person=c.getPerson();

for(Person p:person){

//这里要注意了：如果此Customer对象对应persion有多条联系人记录，他会多出几条重复记录。

System.out.println(p.getpName());

}

}

tx.commit();

}catch(Exception e){

e.printStackTrace();

}

}

}

左外连接：From Customer c left outer join c.person

左外连接返回list中每部分是数组，迫切左外连接返回list是对象

迫切左外连接：From Customer c left outer join fetch c.person

右外连接：From Customer c right outer join c.person

注意：没有迫切右外连接

## 4、QBC查询

Criteria对象，不用写语句

使用方法实现，使用qbc操作实体类和属性

查询所有

1、 创建Criteria对象

2、 调用方法得到结果

条件查询：通过cid=1，cName=百度查询

Criteria criteria=session.createCriteria(Customer.class);

使用Criteria对象里面的方法设置条件值

首先使用add方法，表示设置条件值

在add方法里面使用类的方法实现条件设置

criteria.add(Restrictions.eq("cid", 1));//类似以cid=1；

criteria.add(Restrictions.eq("cName", "百度"));

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

for(Customer l:ls){

System.out.println(l.getcName());

}

tx.commit();

模糊查询

criteria.add(Restrictions.like("cName", "%度%"));

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

for(Customer l:ls){

System.out.println(l.getcName());

}

tx.commit();

排序查询

Criteria criteria=session.createCriteria(Customer.class);

设置对那个属性进行排序，设置排序规则，Order类就两个方法

criteria.addOrder(Order.desc("cid"));

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

for(Customer l:ls){

System.out.println(l.getcName());

}

tx.commit();

分页查询

Criteria criteria=session.createCriteria(Customer.class);

//设置分页数据

//设置起始位置

criteria.setFirstResult(0);//(当前页-1)\*每页记录数

//每页显示几条

criteria.setMaxResults(3);

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

for(Customer l:ls){

System.out.println(l.getcName());

}

tx.commit();

统计查询

Session session=HibernateUtil.getMySession();

Transaction tx=session.beginTransaction();

try{

//1创建对象

Criteria criteria=session.createCriteria(Customer.class);

//2设置操作：使用Projections类

criteria.setProjection(Projections.rowCount());

Object obj=criteria.uniqueResult();

Long l=(Long) obj;

int i=l.intValue();

System.out.println(i);

离线查询：上面都要通过session创建，不用session也可以创建对象

应用场景：主要用来拼接条件时要在别的层进行拼接条件，进行使用的

try{//创建对象

DetachedCriteria deta=DetachedCriteria.forClass(Customer.class);

//最终执行时候才需要到session

Criteria criteria=deta.getExecutableCriteria(session);

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

for (Customer customer : l) {

System.out.println(customer.getcName());

}

tx.commit();

## 5、本地sql查询

SQLQuery对象，使用普通sql实现查询

# Hibernate检索策略

1、hiberante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。

Configuration congfig = new Configuration().configure();

SessionFactory sessionFactory=congfig.buildSessionFactory();

Session session=sessionFactory.openSession();

//延迟查询的效果：你什么时候要这个值我才去查数据库，你不用我就不查

load不会像get在这里进行发送语句

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

//这里不会跟数据库进行发送语句查询的动作。getCid()因为它调用的就是传的值 1

System.out.println(customer.getCid());

//customer.getcName()就会发送语句让数据库进行查询了

System.out.println(customer.getcName());

延迟查询分成两类

（1） 类级别延迟：根据id查询返回实体类对象，调用load方法不会马上查数据库

（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，这个过程称为关联级别延迟

//关联级别延迟：默认hibernate帮我优化了

public static void testJ(){

Configuration congfig = new Configuration().configure();

SessionFactory sessionFactory=congfig.buildSessionFactory();

Session session=sessionFactory.openSession();

//这里发送了语句查询Customer

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

Set<Person> person=customer.getPerson();

//这里根据id发送语句查询person

person.size();

}

如果需要更改：配置更改步骤（了解）：

1、在映射文件中进行配置实现

（1） 根据客户得到所有的联系人，在客户映射文件中配置

2、在set标签上使用属性

（1）fetch：值select（默认）

（2）lazy：值有3个：

-true：延迟（默认）。False：不延迟

extra：极其懒惰：你只要查统计数量的，就会只发统计数量的语句，而不是像上面两个属性通过id查询所有。效率最高

# 批量抓取（了解）

只用一个场景：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。这样查询是性能比较低的

//查询所有客户

Criteria criteria=session.createCriteria(Customer.class);

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

//得到所有客户

for (Customer c2 : c) {

System.out.println(c2.getCid()+":"+c2.getcName());

//每个客户里面所有的联系人

Set<Person> person=c2.getPerson();

for (Person p : person) {

System.out.println(p.getPid()+":"+p.getpName());

}

}

这样查询没有问题，但是性能很严重，发送了很多次语句

使用批量抓取步骤

在映射文件中的，set标签中属性batch-size设置：只要写上整数即可，原则：整数越大，发送语句越少

<set name="person" cascade="save-update,delete" batch-size="100">

得到的语句是这样紫的：

Hibernate:

select

this\_.cid as cid1\_0\_0\_,

this\_.cname as cname2\_0\_0\_,

this\_.address as address3\_0\_0\_,

this\_.type as type4\_0\_0\_

from

customer this\_

1:百度

Hibernate:

select

person0\_.cid as cid5\_1\_1\_,

person0\_.pid as pid1\_1\_1\_,

person0\_.pid as pid1\_1\_0\_,

person0\_.pname as pname2\_1\_0\_,

person0\_.gender as gender3\_1\_0\_,

person0\_.phone as phone4\_1\_0\_,

person0\_.cid as cid5\_1\_0\_

from

person person0\_

where

person0\_.cid in ( //重点关注 in

?, ?, ?, ?, ?

)

# 注意问题

一、实体类（持久化类）编写规则

1、 要求实体类有属性作为唯一值（一般使用id值）

2、 属性建议不要使用基本数据类型，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 \ 其它6种都是首字母大写的就不列举了

2）为什么要用基本数据类型的包装类呢？

答：基本类型有些表示不了，例如：学生们考试成绩得0分，基本数据类型能表示成：int score=0；但不能表示，这位学生有没有参加考试呢？只能包装类Integer score=null；可以表示。

二、全部CRUD都要在事务中执行：不然都报错

三、获取session两种方式：

Configuration configuration= new Configuration();

configuration.configure();//获取核心配置文件方法，重载可以自己指定位置

SessionFactory sessionFactory=configuration.buildSessionFactory();

第一种：不安全的(直接获取不用配置)：Session session=sessionFactory.openSession();

第二种：绑定本地线程安全的session是唯一对象的（前提必须要在核心配置文件配置）：

<!-- 绑定本地线程：底层就是用的threadLocal类事项的 -->

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

代码：

Session session=sessionFactory.getCurrentSession();

四、本身进行修改了全部（包括外键），最后又通过id修改外键一次

为什么hibernate重复的修改了两次操作呢？

因为hibernate双向维护外键，在‘少’和‘多’里面都需要维护外键，修改‘少’时候修改一次外键，修改‘多’时候也修改一次外键，造成效率问题

解决方式：可以让另一方放弃维护外键：让‘少’的一方放弃维护外键

为什么让‘少’的一方放弃呢？：一个国家有总统，国家有很多人，总统不认识国家所有人，国家所有人都认识总统

具体实现：Incerse属性

在放弃维护映射文件中，进行配置，在set标签上使用Incerse属性值为true

Hibernate:

update

person

set

pname=?,

gender=?,

phone=?,

cid=?

where

pid=? 问题点：明明已经修改了

Hibernate:

update

person

set

cid=? 这里又做一次通过主键id修改

where

pid=?