**Hibernate5 第二天**

**【学习目标】**

通过本内容的学习，学生：

## 1、能够说出Hibernate中常用的主键生成策略

1. 说出increment是如何生成主键的。
2. 说出identity是如何生成主键的。
3. 说出sequence是如何生成主键的。
4. 说出native是如何生成主键的。
5. 说出uuid是如何生成主键的。
6. 说出assigned是如何生成主键的。

## 2，能够描述Hibernate持久化类及编写规则

1. 说出Hibernate持久化类的概念。
2. 说出Hibernate持久化类的编写规则。

## 3，能够描述Hibernate持久化对象的三种状态及特征

1. 说出瞬时态对象的特点。
2. 说出持久态对象的特点。
3. 说出脱管态对象的特点。

## 4，能够描述Hibernate持久化对象的三种状态的转换关系

1. 说出如何从瞬时态对象转成持久态对象
2. 说出如何从瞬时态对象转成脱管态对象
3. 说出如何从持久态对象转成瞬时态对象
4. 说出如何从持久态对象转成脱管态对象
5. 说出如何从脱管态对象转成瞬时态对象
6. 说出如何从脱管态对象转成持久态对象

## 5，能够描述Hibernate的一级缓存

1. 说出一级缓存的概念。
2. 说出一级缓存的作用。
3. 说出一级缓存的快照区。

## 6，能够应用Hibernate进行事务控制

1. 说出事务的概念。
2. 说出事务的四大特性。
3. 说出如果不考虑隔离性引发的读问题。
4. 配置Hibernate中事务的隔离级别。
5. 配置Hibernate中Session绑定当前线程。

## 7，能够描述Criteria对象的作用及常用的方法

1. 说出Criteria对象的作用。
2. 说出Criteria对象的常用方法。
3. 独立使用QBC编写完成查询所有客户的代码

# 主键的生成策略

## 主键类型之自然主键和代理主键

  1. 创建表的时候  
    \* 自然主键:对象本身的一个属性.创建一个人员表,每个人都有一个身份证号.(唯一的)使用身份证号作为表的主键.自然主键.（开发中不会使用这种方式）  
    \* 代理主键:不是对象本身的一个属性.创建一个人员表,为每个人员单独创建一个字段.用这个字段作为主键.代理主键.（开发中推荐使用这种方式）  
      
  2. 创建表的时候尽量使用代理主键创建表

## 主键的生成策略

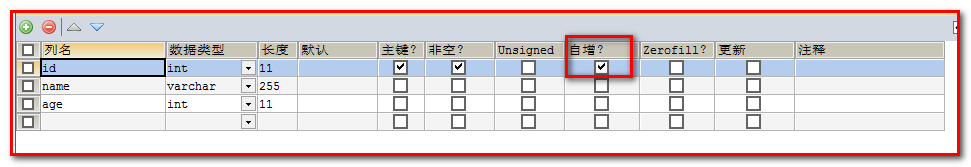
1.increment（了解）:适用于short,int,long作为主键.不是使用的数据库自动增长机制.  
     \* Hibernate中提供的一种增长机制  
         \* 先进行查询    :select max(id) from person;  
         \* 再进行插入    :获得最大值+1作为新的记录的主键  
          
     \* 问题:不能在集群环境下或者有并发访问的情况下使用  
      
    2. identity:适用于short,int,long作为主键。但是这个必须使用在有自动增长数据库中.采用的是数据库底层的自动增长机制  
        \* 底层使用的是数据库的自动增长(auto\_increment).像Oracle数据库没有自动增长  
      
    3. sequence:适用于short,int,long作为主键.底层使用的是序列的增长方式  
        \* Oracle数据库底层没有自动增长,想自动增长需要使用序列  
      
    4. uuid:适用于char,varchar类型的作为主键  
        \* 使用随机的字符串作为主键  
      
    5. native:本地策略.根据底层的数据库不同,自动选择适用于该种数据库的生成策略.(short,int,long)  
        \* 如果底层使用的MySQL数据库:相当于identity  
        \* 如果底层使用Oracle数据库:相当于sequence  
      
     6. assigned:主键的生成不用Hibernate管理了.必须手动设置主键

### increment主键策略

由hibernate来生成OID和维护的，原理是select max(id) +1

如果数据表中没有数据，则初始的时候,hibernate给值是1,再次给值2

1. 新建web项目hibernate5\_day02,添加hibernate、mysql驱动以及日志架包
2. 新建hibernate\_day02数据库,在数据库中新建person表如下：



注意主键自增选项打钩

1. 在com.igeek.demo1下新建Person类

|  |
| --- |
| **package** com.igeek.demo1;  **public** **class** Person {  **private** **int** id;  **private** String name;  **private** **int** age;  //省略getter和setter方法 |

1. 在com.igeek.demo1下编写配置文件Person.hbm.xml

|  |
| --- |
| <?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>  <class name=*"com.igeek.demo1.Person"* table=*"person"*>  <id name=*"id"* column=*"id"*>  <generator class=*"increment"*></generator>  </id>  <property name=*"name"* column=*"name"*></property>  <property name=*"age"* column=*"age"*></property>  </class>  </hibernate-mapping> |

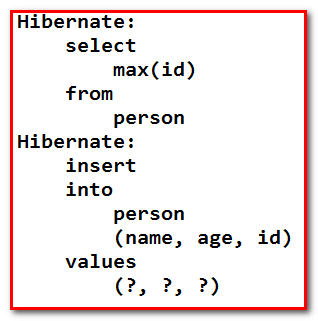
1. 在src下编写配置文件hibernate.cfg.xml

|  |
| --- |
| <?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>  <property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"hibernate.connection.url"*>jdbc:mysql:///hibernate\_day02</property>  <property name=*"hibernate.connection.username"*>root</property>  <property name=*"hibernate.connection.password"*>123456</property>  <property name=*"hibernate.dialect org.hibernate.dialect.MySQLDialect"*></property>  <!-- 打印sql语句的配置 -->  <property name=*"hibernate.show\_sql"*>true</property>  <property name=*"hibernate.format\_sql"*>true</property>  <property name=*"hibernate.hbm2ddl.auto"*>update</property>    <mapping resource=*"com/igeek/demo1/Person.hbm.xml"*/>    </session-factory>  </hibernate-configuration> |

1. 拷贝第一天编写的hibernate工具类放在src目录下
2. 在com.igeek.demo1下编写测试类TestPrimaryKey.java

|  |
| --- |
| **package** com.igeek.demo1;  **import** org.hibernate.Session;  **import** org.hibernate.Transaction;  **import** org.junit.Test;  **import** com.igeek.util.HibernateUtils;  **public** **class** TestPrimaryKey {  @Test  **public** **void** run1(){  //1.获取session  Session session = HibernateUtils.*openSession*();  //2.开启事务  Transaction tr = session.beginTransaction();  //3.创建Person对象  Person person = **new** Person();  person.setName("小凤");  person.setAge(18);  //4.保存数据  session.save(person);  //5.提交事务  tr.commit();  //6.释放资源  session.close();  }  } |

1. 测试查看结果



适用场景：



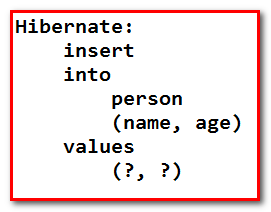
问题：可能出现多线程冲突问题，两个线程同时查询max(id)，同时+1 ,insert

### identity主键策略

1. 修改Person.hbm.xml

|  |
| --- |
| <generator class=*"identity"*></generator> |

1. 测试



### sequence主键策略—oracle演示

### uuid主键策略

用于String类型，生成代理主键，采用uuid （32位）作为主键值

Hibernate会产生不重复的32位字符串作为主键

1.

【示例】

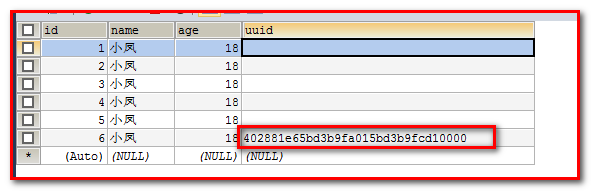
1. 修改Person类，添加uuid属性

|  |
| --- |
| **private** String uuid;    **public** String getUuid() {  **return** uuid;  }  **public** **void** setUuid(String uuid) {  **this**.uuid = uuid;  } |

1. 修改Person.hbm.xml的id节点

|  |
| --- |
| <id name=*"uuid"* column=*"uuid"*>  <generator class=*"uuid"*></generator>  </id> |

1. 测试



### assigned主键策略

唯一的一个自然主键设置方式，手动设置主键的值，

【示例】

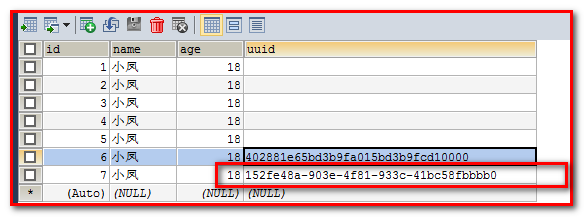
1. 修改Person.hbm.xml的id生成策略

|  |
| --- |
| <id name=*"uuid"* column=*"uuid"*>  <generator class=*"assigned"*></generator>  </id> |

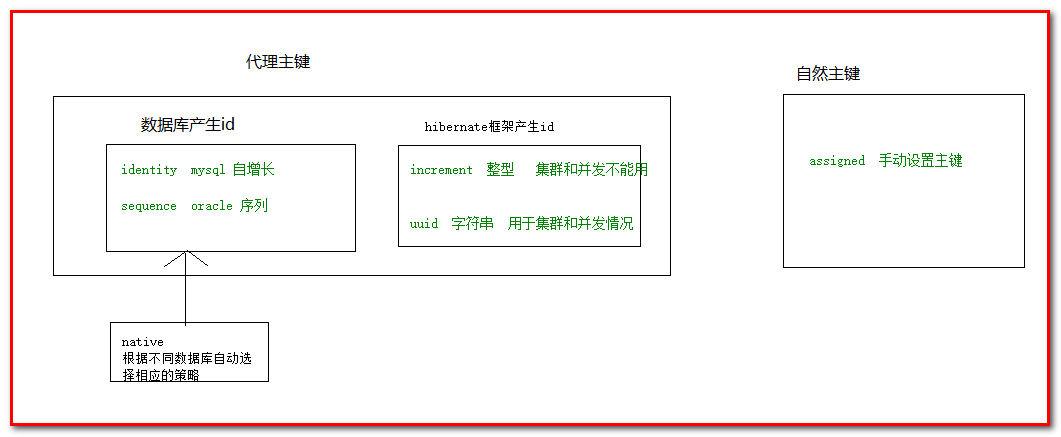
1. 修改测试代码

|  |
| --- |
| //3.创建Person对象  Person person = **new** Person();  person.setName("小凤");  person.setAge(18);  //手动设置主键  person.setUuid(UUID.*randomUUID*().toString()); |

1. 测试



### 小结



# Hibernate的持久化类概念和操作

## 什么是持久化类

   1. 持久化类:就是一个Java类（咱们编写的JavaBean），这个Java类与表建立了映射关系就可以成为是持久化类。  
     \* 持久化类 = JavaBean + xxx.hbm.xml  
      
    2. Hibernate框架环境中存在持久化类的概念

## 持久化类的编写规则

    1. 提供一个无参数 public访问控制符的构造器    -- 底层需要进行反射

     2. 提供一个标识属性，映射数据表主键字段      -- 唯一标识OID.数据库中通过主键.Java

对象通过地址确定对象.持久化类通过唯一标识OID确定记录

     3. 所有的属性需要提供get和set方法

     4. 标识属性应尽量使用基本数据类型的包装类型

     5. 选择实现序列化接口

## 持久化类示例

1. 在com.igeek.demo2下新建User类：

|  |
| --- |
| **package** com.igeek.demo2;  **public** **class** User **implements** Serializable{  //唯一标识OID  **private** Integer id;  **private** String name;  **private** Integer age;    **public** Integer getId() {  **return** id;  }  **public** **void** setId(Integer id) {  **this**.id = id;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** Integer getAge() {  **return** age;  }  **public** **void** setAge(Integer age) {  **this**.age = age;  }  } |

1. 在com.igeek.demo2下新建User.hbm.xml

|  |
| --- |
| <?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>  <class name=*"com.igeek.demo2.User"* table=*"t\_user"*>  <id name=*"id"* column=*"id"*>  <generator class=*"native"*></generator>  </id>  <property name=*"name"* column=*"name"*></property>  <property name=*"age"* column=*"age"*></property>  </class>  </hibernate-mapping> |

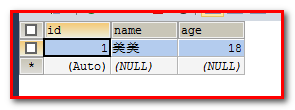
1. 在hibernate.cfg.xml中关联User.hbm.xml

|  |
| --- |
| <mapping resource=*"com/igeek/demo2/User.hbm.xml"*/> |

1. 在com.igeek.demo2下新建TestUser测试类

|  |
| --- |
| **package** com.igeek.demo2;  **import** org.hibernate.Session;  **import** org.hibernate.Transaction;  **import** org.junit.Test;  **import** com.igeek.util.HibernateUtils;  **public** **class** TestUser {  @Test  **public** **void** run1(){  //获取session  Session session = HibernateUtils.*openSession*();  Transaction tr = session.beginTransaction();  //新建User对象  User user = **new** User();  user.setName("美美");  user.setAge(18);  //保存  session.save(user);  tr.commit();  //关闭资源  session.close();  }  } |

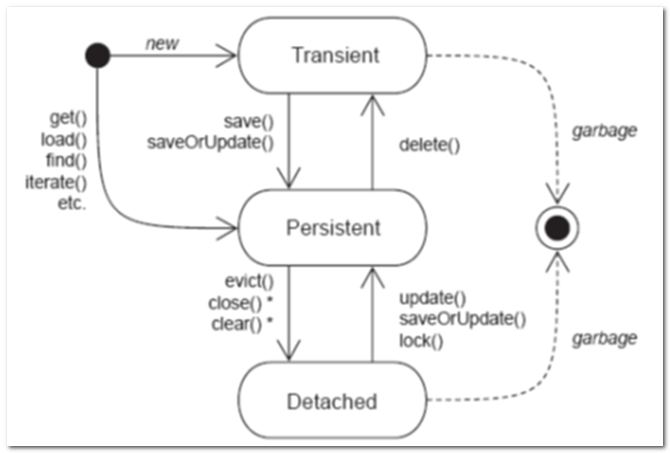
1. 测试

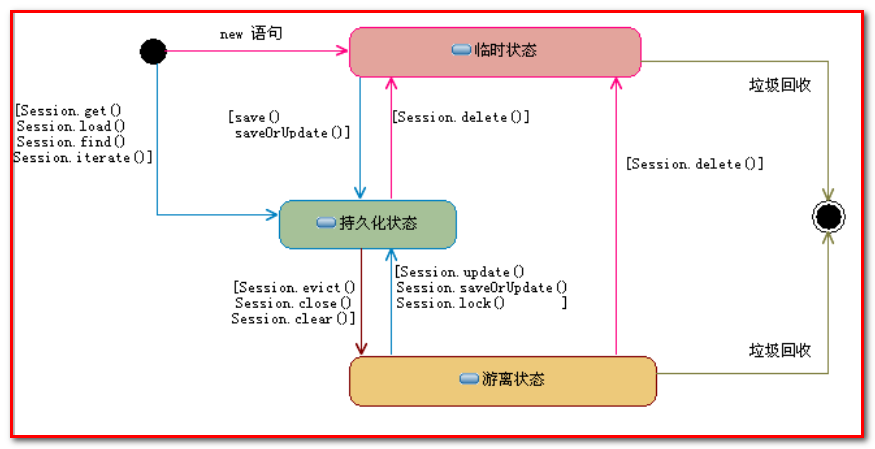


# Hibernate持久化对象的状态

## 持久化对象的状态

1. Hibernate的持久化类  
           \* 持久化类:Java类与数据库的某个表建立了映射关系.这个类就称为是持久化类  
               \* 持久化类 = Java类 + hbm的配置文件  
         
     2. Hibernate的持久化类的状态



  
        \* Hibernate为了管理持久化类：将持久化类分成了三个状态  
            \* 瞬时态:Transient  Object  
            \* 持久态:Persistent Object  
            \* 脱管态:Detached Object（游离态）

## Hibernate持久化对象的状态的转换

1. 瞬时态    -- 没有持久化标识OID, 没有被纳入到Session对象的管理  
           \* 获得瞬时态的对象  
               \* User user = new User()  
           \* 瞬时态对象转换持久态  
               \* session.save()  
         
       2. 持久态    -- 有持久化标识OID,已经被纳入到Session对象的管理  
           \* 获得持久态的对象  
               \* get()/load();  
           \* 持久态转换成瞬时态对象  
               \* delete();  --- 比较有争议的，进入特殊的状态(删除态:Hibernate中不建议使用的)  
           \* 持久态对象转成脱管态对象  
               \* session的close()/evict()/clear();  
         
       3. 托管态    -- 有持久化标识OID，没有被纳入到Session对象的管理  
         
       4. 注意：持久态对象有自动更新数据库的能力!!!

### 持久化对象的状态转换示例

1. 在com.igeek.demo3下新建TestUser类，编写测试方法，并通过调试看id值

|  |
| --- |
| **package** com.igeek.demo3;  **import** org.hibernate.Session;  **import** org.hibernate.Transaction;  **import** org.junit.Test;  **import** com.igeek.demo2.User;  **import** com.igeek.util.HibernateUtils;  **public** **class** TestUser {  //持久化状态转换  @Test  **public** **void** run1(){  Session session = HibernateUtils.*openSession*();  Transaction tr = session.beginTransaction();  //瞬时态：  //特点：没有OID，new出来的一个对象，不与session关联，不受session管理，数据库中没有对应的记录  User user = **new** User();  user.setName("小凤");  user.setAge(20);  //在save执行之前，user都只是瞬时态  //当save执行之后，user就处于持久态  session.save(user);  //持久态  //特点：有OID，数据库存在对应的记录，与session关联，受session管理  tr.commit();  session.close();  //只要session已关闭，那么就处于脱管态  //有OID，数据库中存在对应的记录，但是不与session关联，不受session管理  System.***out***.println(user);  }  } |

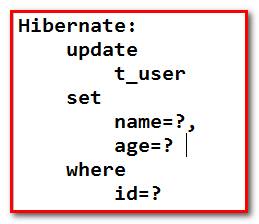
### 持久态对象自动更新数据库

通过改变查询出来的PO的属性值，来查看一级缓存的更改；通过提交事务，来使用快照更新数据。

1.在TestUser类中添加测试代码：

|  |
| --- |
| @Test  **public** **void** run2(){  //获取session  Session session = HibernateUtils.*openSession*();  Transaction tr = session.beginTransaction();  //调用get方法，现在user是持久态  User user = session.get(User.**class**, 1);  //设置属性  user.setName("熊大");  //正常 ,调用修改方法  //session.update(user);  tr.commit();  //关闭资源  session.close();  } |

1. 测试发现没有执行session.update(user),数据库中数据已修改



### 三者的区别总结

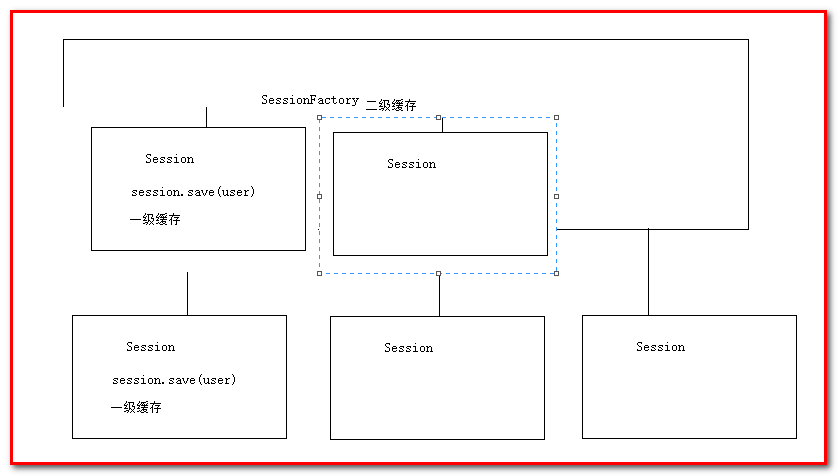
对于三者：在session中存在的，就是持久化对象，不存在的就是瞬时或脱管对象。

对于瞬时和脱管对象：有oid（持久化标识）的就脱管对象，没有的就是瞬时对象。

OID一定是与数据库主键一一对应的

|  |  |  |  |
| --- | --- | --- | --- |
|  | 是否有持久化标识OID | session是否存在 | 数据库中是否有 |
| 瞬时态对象-临时状态 | n | n | n |
| 持久态对象 | y | y | y/（n：没有提交） |
| 脱管态对象-游离 | y | n | y |

# Hibernate的一级缓存



## Session对象的一级缓存

1. 什么是缓存？  
       \* 其实就是一块内存空间,将数据源（数据库或者文件）中的数据存放到缓存中.再次获取的时候 ,直接从缓存中获取.可以提升程序的性能！  
         
     2. Hibernate框架提供了两种缓存  
       \* 一级缓存    -- 自带的不可卸载的.一级缓存的生命周期与session一致.一级缓存称为session级别的缓存.  
      \* 二级缓存    -- 默认没有开启，需要手动配置才可以使用的.二级缓存可以在多个session中共享数据,二级缓存称为是sessionFactory级别的缓存.  
         
     3. Session对象的缓存概述  
       \* Session接口中,有一系列的java的集合,这些java集合构成了Session级别的缓存(一级缓存).将对象存入到一级缓存中,session没有结束生命周期,那么对象在session中存放着  
       \* 内存中包含Session实例 --> Session的缓存（一些集合） --> 集合中包含的是缓存对象  
         
     4. 证明一级缓存的存在，编写查询的代码即可证明  
       \* 在同一个Session对象中两次查询，可以证明使用了缓存  
         
     5. Hibernate框架是如何做到数据发生变化时进行同步操作的呢？  
       \* 使用get方法查询User对象  
       \* 然后设置User对象的一个属性，注意：没有做update操作。发现，数据库中的记录也改变了。  
       \* 利用快照机制来完成的（SnapShot）

### 证明一级缓存的存在性示例1

通过多次查询同一个po对象数据，得到同一个对象，且第二次不再从数据库查询，证明一级缓存存在。

1. 在com.igeek.demo4下新建TestUser类

|  |
| --- |
| **package** com.igeek.demo4;  **import** org.hibernate.Session;  **import** org.hibernate.Transaction;  **import** org.junit.Test;  **import** com.igeek.demo2.User;  **import** com.igeek.util.HibernateUtils;  **public** **class** TestUser {  //查看一级缓存是否存在  @Test  **public** **void** run1(){  Session session = HibernateUtils.*openSession*();  Transaction tr = session.beginTransaction();  //第一次查询，hibernate必然发出sql语句  User user =session.get(User.**class**, 1);  System.***out***.println(user);  //第二次查询，hibernate从一级缓存中取数据，不发sql  User user2 = session.get(User.**class**, 1);  System.***out***.println(user2);  tr.commit();  session.close();  }  } |

1. 测试观察结果只发出一条sql语句，说明一级缓存存在。

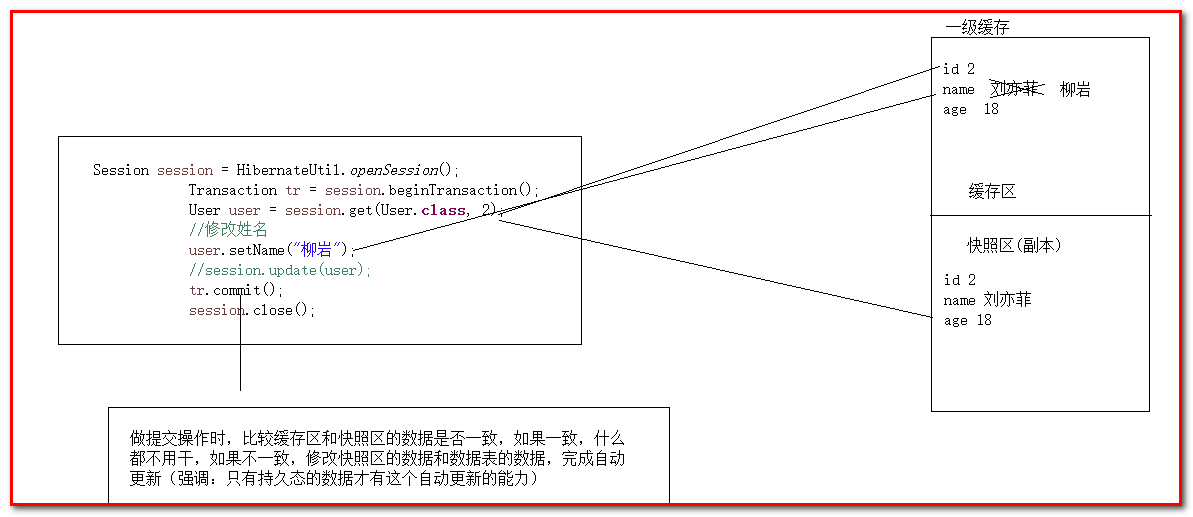
### 证明一级缓存的存在性示例2

1. 在刚才的TestUser中添加测试方法如下：

|  |
| --- |
| @Test  **public** **void** run2(){  Session session = HibernateUtils.*openSession*();  Transaction tr = session.beginTransaction();  //1.新建一个user对象  User user = **new** User();  user.setName("冠希");  user.setAge(30);  //2.保存对象，发送sql语句，返回保存的OID,并且把对象保存在一级缓存中  Serializable id = session.save(user);  tr.commit();  //3.通过返回id直接从一级缓存中获取数据，不会发送sql  User user2 = session.get(User.**class**, id);  //两个对象是同一个  System.***out***.println(user);  System.***out***.println(user2);    session.close();  } |

1. 查看结果，只发送一条sql语句，说明一级缓存存在。

### 一级缓存中的快照机制



# Hibernate中的事务

## 事务相关的概念

1. 什么是事务  
        \* 事务就是逻辑上的一组操作，组成事务的各个执行单元，操作要么全都成功，要么全都失败.  
        \* 转账的例子：冠希给美美转钱，扣钱，加钱。两个操作组成了一个事情！  
      
    2. 事务的特性  
        \* 原子性    -- 事务不可分割.  
        \* 一致性    -- 事务执行的前后数据的完整性保持一致.  
        \* 隔离性    -- 一个事务执行的过程中,不应该受到其他的事务的干扰.  
        \* 持久性    -- 事务一旦提交,数据就永久保持到数据库中.  
      
    3. 如果不考虑隔离性:引发一些读的问题  
        \* 脏读          -- 一个事务读到了另一个事务未提交的数据.  
        \* 不可重复读       -- 一个事务读到了另一个事务已经提交的update数据,导致多次查询结果不一致.  
        \* 虚读          -- 一个事务读到了另一个事务已经提交的insert数据,导致多次查询结构不一致.  
      
    4. 通过设置数据库的隔离级别来解决上述读的问题  
        \* 未提交读:以上的读的问题都有可能发生  
        \* 已提交读:避免脏读,但是不可重复读，虚读都有可能发生  
        \* 可重复读:避免脏读，不可重复读.但是虚读是有可能发生  
        \* 串行化:以上读的情况都可以避免  
      
    5. 如果想在Hibernate的框架中来设置隔离级别，需要在hibernate.cfg.xml的配置文件中通过标签来配置（了解）  
        \* 通过：hibernate.connection.isolation = 4 来配置  
        \* 取值  
            \* 1—Read uncommitted isolation  
            \* 2—Read committed isolation  
            \* 4—Repeatable read isolation  
            \* 8—Serializable isolation

## 绑定本地的Session

1. 之前在讲JavaWEB的事务的时候，需要在业务层使用Connection来开启事务  
        \* 一种是通过参数的方式传递下去  
        \* 另一种是把Connection绑定到ThreadLocal对象中  
      
    2. 现在的Hibernate框架中，使用session对象开启事务，所以需要来传递session对象，框架提供了ThreadLocal的方式  
        \* 需要在hibernate.cfg.xml的配置文件中提供配置  
    \* <property name="hibernate.current\_session\_context\_class">thread</property>  
          
        \* 修改HibernateUtil的工具类，使用SessionFactory的getCurrentSession()方法，获取当前的Session对象。并且该Session对象不用手动关闭，线程结束了，会自动关闭。  
            public static Session getCurrentSession(){  
                return factory.getCurrentSession();  
            }  
          
        \* 注意：想使用getCurrentSession()方法，必须要先配置才能使用

### 绑定本地session示例

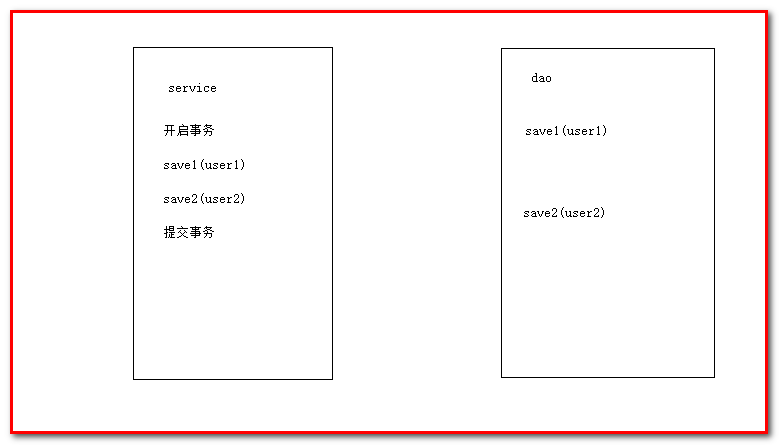
1. 修改hibernate.cfg.xml配置文件

|  |
| --- |
| <!-- 把session放在线程中 -->  <property name=*"hibernate.current\_session\_context\_class"*>thread</property> |

1. 修改HibernateUtil的工具类，使用SessionFactory的getCurrentSession()方法，获取当前的Session对象。

|  |
| --- |
| /\*\*  \* 这种方式获取的session不用手动关闭，线程结束了会自动关闭  \* **@return**  \*/  **public** **static** Session getCurrentSession(){  **return** ***FACTORY***.getCurrentSession();  } |

1. 在com.igeek.demo5下新建UserDao、UserService、TestUser3个类，代码如下



UserDao.java

|  |
| --- |
| **package** com.igeek.demo5;  **import** org.hibernate.Session;  **import** com.igeek.demo2.User;  **import** com.igeek.util.HibernateUtils;  **public** **class** UserDao {  **public** **void** save1(User user1){  //获取session对象  Session session = HibernateUtils.*getCurrentSession*();  //不需要开启事务  session.save(user1);  }    **public** **void** save2(User user2){  //获取session对象  Session session = HibernateUtils.*getCurrentSession*();  //不需要开启事务  session.save(user2);  }  } |

UserService.java

|  |
| --- |
| **package** com.igeek.demo5;  **import** org.hibernate.Session;  **import** org.hibernate.Transaction;  **import** com.igeek.demo2.User;  **import** com.igeek.util.HibernateUtils;  **public** **class** UserService {  /\*\*  \* 使用事务的方式  \* **@param** user1  \* **@param** user2  \*/  **public** **void** save(User user1,User user2){  //先获取到session对象，session会创建，会把session绑定到当前线程上  Session session = HibernateUtils.*getCurrentSession*();  //开启事务  Transaction tr = session.beginTransaction();  **try** {  //调用dao层  UserDao dao = **new** UserDao();  dao.save1(user1);  dao.save2(user2);  //提交事务  tr.commit();  }**catch**(Exception e){  //回滚事务  tr.rollback();  e.printStackTrace();  }**finally** {  //释放资源，强调问题，当前线程结束了，session会自动关闭  }  }  } |

TestUser.java

|  |
| --- |
| **package** com.igeek.demo5;  **import** org.junit.Test;  **import** com.igeek.demo2.User;  **public** **class** TestUser {  @Test  **public** **void** run1(){  UserService service = **new** UserService();  User user1 = **new** User();  user1.setName("熊大");  User user2 = **new** User();  user2.setName("熊二");    service.save(user1, user2);  }  } |

1. 测试查看效果

# Hibernate框架的查询方式

## Criteria查询接口（做条件查询非常合适）

1. QBC：Query By Criteria    -- 按条件进行查询  
    2. QBC查询方式的使用  
        \* 先创建查询的接口  
            \* Criteria c = session.createCriteria(User.class);  
        \* 设置查询的条件  
            \* criteria.add(Restrictions.gt("age", 10));  
        \* 查询数据  
            \* List<User> list = criteria.list();

### 不带条件查询

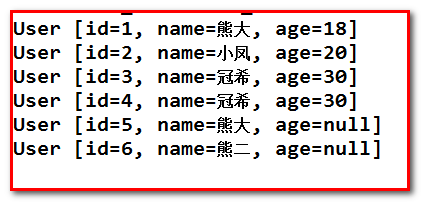
1. 在com.igeek.demo6下新建TestQBC类，编写测试代码如下：

|  |
| --- |
| **package** com.igeek.demo6;  **import** java.util.List;  **import** org.hibernate.Criteria;  **import** org.hibernate.Session;  **import** org.hibernate.Transaction;  **import** org.junit.Test;  **import** com.igeek.demo2.User;  **import** com.igeek.util.HibernateUtils;  /\*\*  \* QBC的查询测试  \* **@author** jerry  \*  \*/  **public** **class** TestQBC {  //不带条件的查询  @Test  **public** **void** run1(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();  //创建QBC查询接口  Criteria c = session.createCriteria(User.**class**);  //查询所有数据  List<User> users = c.list();  **for** (User user : users) {  System.***out***.println(user);  }    tr.commit();  //session不用关闭  }  } |

1. 修改com.igeek.demo6下的User类，添加toString()方法

|  |
| --- |
| @Override  **public** String toString() {  **return** "User [id=" + id + ", name=" + name + ", age=" + age + "]";  } |

1. 测试



### 条件查询

1. 在TestQBC类中加入条件查询的测试方法

|  |
| --- |
| //模糊查询  @Test  **public** **void** run2(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();  //创建QBC查询接口  Criteria c = session.createCriteria(User.**class**);  //设置条件 查询名字中带熊的记录  c.add(Restrictions.*like*("name", "%熊%"));    //查询所有数据  List<User> users = c.list();  **for** (User user : users) {  System.***out***.println(user);  }    tr.commit();  //session不用关闭  }  //多条件查询 and  @Test  **public** **void** run3(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();  //创建QBC查询接口  Criteria c = session.createCriteria(User.**class**);  //设置条件 age=18  //第一个参数是javabean中的属性  //第二个参数是要设置的值  c.add(Restrictions.*eq*("age", 18));  //设置第二个条件  c.add(Restrictions.*eq*("name", "熊大"));  //设置第N个条件  //查询所有数据  List<User> users = c.list();  **for** (User user : users) {  System.***out***.println(user);  }    tr.commit();  //session不用关闭  }    //or条件  @Test  **public** **void** run4(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();  //创建QBC查询接口  Criteria c = session.createCriteria(User.**class**);  //设置条件 查询名字中带熊或者年龄大于18的记录  c.add(Restrictions.*or*(Restrictions.*like*("name", "%熊%"),Restrictions.*gt*("age", 18)));  //查询所有数据  List<User> users = c.list();  **for** (User user : users) {  System.***out***.println(user);  }    tr.commit();  //session不用关闭  }    //between...and  @Test  **public** **void** run5(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();  //创建QBC查询接口  Criteria c = session.createCriteria(User.**class**);  //设置条件 查询年龄在20和30之间的记录  c.add(Restrictions.*between*("age", 20, 30));  //查询所有数据  List<User> users = c.list();  **for** (User user : users) {  System.***out***.println(user);  }    tr.commit();  //session不用关闭  }    //分页  @Test  **public** **void** run6(){  Session session = HibernateUtils.*getCurrentSession*();  Transaction tr = session.beginTransaction();  //创建QBC查询接口  Criteria c = session.createCriteria(User.**class**);  //设置条件  //mysql 中 分页limit index,size  //index 索引 从0开始  //size 每页条数  c.setFirstResult(0);  c.setMaxResults(2);  //查询所有数据  List<User> users = c.list();  **for** (User user : users) {  System.***out***.println(user);  }    tr.commit();  //session不用关闭  } |

1. 测试查看结果

课后作业

1. 使用QBC完成CRM系统中客户信息管理模块的查询功能