Hibernate框架第二天

课程回顾：Hibernate框架的第一天

1. Hibernate框架的概述：ORM
2. 框架的入门的程序
    * 编写映射的配置文件
    * 编写核心的配置文件
    * 编写程序

3. 配置的文件
4. 使用的接口和方法

今天内容

1. Hibernate持久化对象的状态
2. Hibernate的一级缓存
3. Hibernate操作持久化对象的方法
4. Hibernate的基本查询

Hibernate的持久化类

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

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. 所有属性提供public访问控制符的 set或者get 方法
4. 标识属性应尽量使用基本数据类型的包装类型

区分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: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 user;
        * 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.

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
            * 没有持久化标识OID, 没有被纳入到Session对象的管理.

        * 持久态:Persistent Object
            * 有持久化标识OID,已经被纳入到Session对象的管理.

        * 脱管态:Detached Object
            * 有持久化标识OID,没有被纳入到Session对象的管理.

Hibernate持久化对象的状态的转换

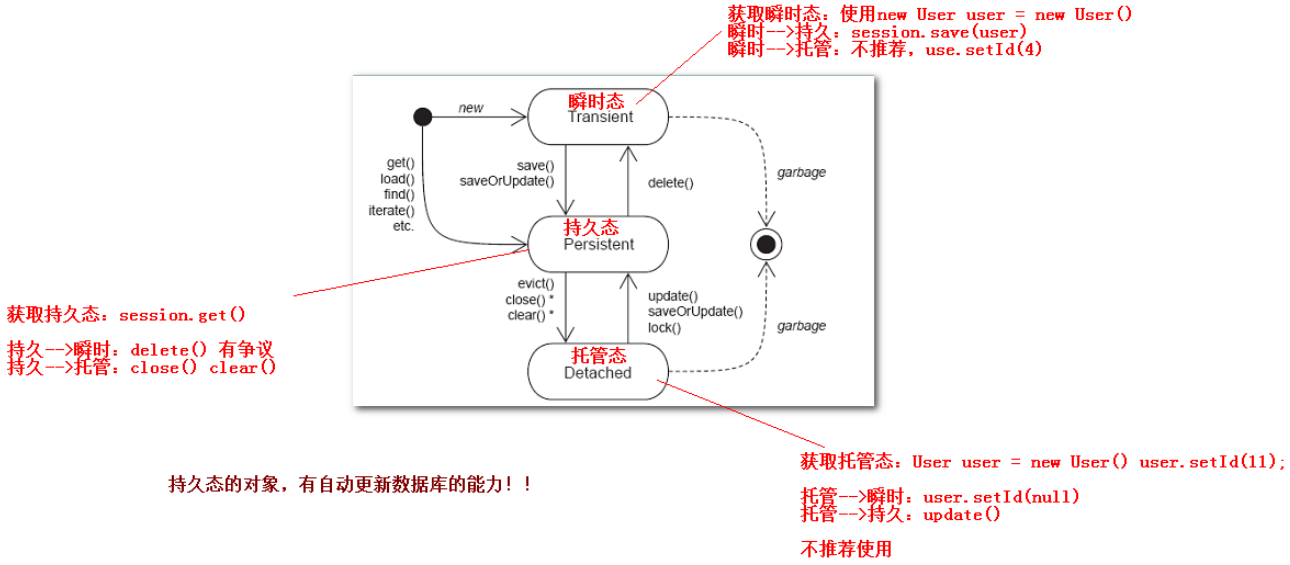
1. 瞬时态 -- 没有持久化标识OID, 没有被纳入到Session对象的管理
    * 获得瞬时态的对象
        * User user = new User()
    * 瞬时态对象转换持久态
        * save()/saveOrUpdate();
    * 瞬时态对象转换成脱管态
        * user.setId(1)

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. 脱管态 -- 有持久化标识OID,没有被纳入到Session对象的管理
    * 获得托管态对象:不建议直接获得脱管态的对象.
        * User user = new User();
        * user.setId(1);
    * 脱管态对象转换成持久态对象
        * update();/saveOrUpdate()/lock();
    * 脱管态对象转换成瞬时态对象
        * user.setId(null);

4. 注意：持久态对象有自动更新数据库的能力!!!

三个状态对象之间的转换



Hibernate的一级缓存

Session对象的一级缓存（重点）

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
   * 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）

快照的机制保存自动更新数据库的能力！！

```java
@Test
public void run4(){
    Session session = HibernateUtils.getSession();
    Transaction tr = session.beginTransaction();
    // 获取到持久态的对象
    User user = session.get(User.class,1);
    // 重新设置新的名称
    user.setName("隔离老王");

    tr.commit();
    session.close();
}
```

修改缓存区域中的数据的

先查询数据库

sesion销毁了，缓存中任何内容都没有了

提交之前：自己比对缓存区和快照的数据是否是一致的，如果是一致，没问题。

如果发现不一致，修改数据库中的值，同时把快照区的数据更新了。

控制Session的一级缓存（了解）

1. 学习Session接口中与一级缓存相关的方法
    * Session.clear()　　-- 清空缓存。
    * Session.evict(Object entity)　　-- 从一级缓存中清除指定的实体对象。
    * Session.flush()　　-- 刷出缓存

Hibernate中的事务与并发

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
    * 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

丢失更新的问题

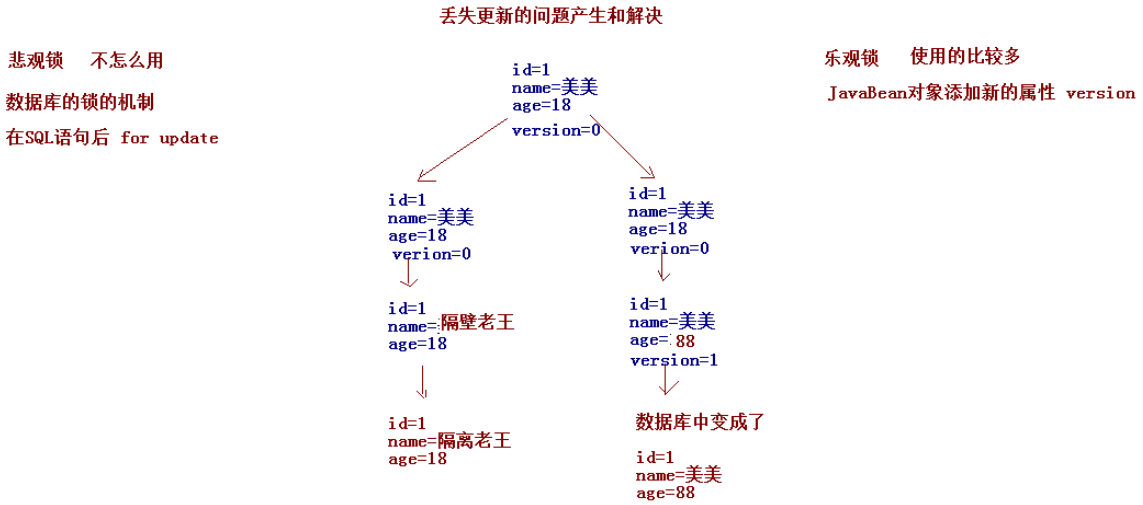
1. 如果不考虑隔离性，也会产生写入数据的问题，这一类的问题叫丢失更新的问题。
2. 例如：两个事务同时对某一条记录做修改，就会引发丢失更新的问题。
   * A事务和B事务同时获取到一条数据，同时再做修改
   * 如果A事务修改完成后，提交了事务
   * B事务修改完成后，不管是提交还是回滚，如果不做处理，都会对数据产生影响

3. 解决方案有两种
   * 悲观锁
     * 采用的是数据库提供的一种锁机制，如果采用做了这种机制，在SQL语句的后面添加 for update 子句
       * 当A事务在操作该条记录时，会把该条记录锁起来，其他事务是不能操作这条记录的。
       * 只有当A事务提交后，锁释放了，其他事务才能操作该条记录

   * 乐观锁
     * 采用版本号的机制来解决的。会给表结构添加一个字段version=0，默认值是0
       * 当A事务在操作完该条记录，提交事务时，会先检查版本号，如果发生版本号的值相同时，才可以提交事务。同时会更新版本号version=1.
       * 当B事务操作完该条记录时，提交事务时，会先检查版本号，如果发现版本不同时，程序会出现错误。

4. 使用Hibernate框架解决丢失更新的问题
   * 悲观锁
     * 使用session.get(Customer.class, 1,LockMode.UPGRADE); 方法

   * 乐观锁
     * 1.在对应的JavaBean中添加一个属性，名称可以是任意的。例如：private Integer version; 提供get和set方法
     * 2.在映射的配置文件中，提供<version name="version"/>标签即可。



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>

   * 重新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

```
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。
```

Hibernate框架的查询方式

Query查询接口

1. 具体的查询代码如下

```
// 1.查询所有记录
/*Query query = session.createQuery("from Customer");
List<Customer> list = query.list();
System.out.println(list);*/

// 2.条件查询:
/*Query query = session.createQuery("from Customer where name = ?");
query.setString(0, "李健");
List<Customer> list = query.list();
System.out.println(list);*/

// 3.条件查询:
/*Query query = session.createQuery("from Customer where name = :aaa and age = :bbb");
query.setString("aaa", "李健");
query.setInteger("bbb", 38);
List<Customer> list = query.list();
System.out.println(list);*/
```

Criteria查询接口（做条件查询非常合适）

1. 具体的查询代码如下

```
// 1.查询所有记录
/*Criteria criteria = session.createCriteria(Customer.class);
List<Customer> list = criteria.list();
System.out.println(list);*/

// 2.条件查询
/*Criteria criteria = session.createCriteria(Customer.class);
criteria.add(Restrictions.eq("name", "李健"));
List<Customer> list = criteria.list();
System.out.println(list);*/

// 3.条件查询
/*Criteria criteria = session.createCriteria(Customer.class);
criteria.add(Restrictions.eq("name", "李健"));
criteria.add(Restrictions.eq("age", 38));
List<Customer> list = criteria.list();
System.out.println(list);*/
```