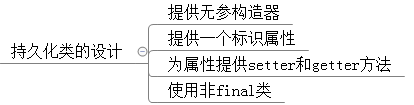
# 持久化类的设计



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

nt – 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;

# 对实体类crud操作

## (save, persist)添加操作

调用session里面的save方法实现

1>使一个临时对象转变为持久化对象

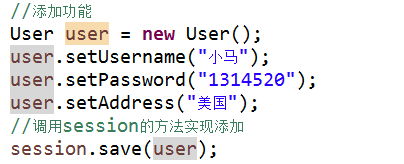
2>Session 的 save() 方法完成以下操作:

**把 News 对象加入到 Session 缓存中, 使它进入持久化状态**

**选用映射文件指定的标识符生成器, 为持久化对象分配唯一的 OID。**

**计划执行一条 insert 语句：在 flush 缓存的时候**

3>Hibernate 通过持久化对象的 OID 来维持它和数据库相关记录的对应关系. 当 News 对象处于持久化状态时, **不允许程序随意修改它的 ID**



persist() 和 save() 区别：

在调用 persist 方法之前, 若对象已经有 id 了, 则不会执行 INSERT, 而抛出异常

/\*\*

\* 1. save() 方法

\* 1). 使一个临时对象变为持久化对象

\* 2). 为对象分配 ID.

\* 3). 在 flush 缓存时会发送一条 INSERT 语句.

\* 4). 在 save 方法之前的 id 是无效的

\* 5). 持久化对象的 ID 是不能被修改的!

\*/

@Test

public void testSave(){ //插入数据

News news = new News();

news.setTitle("CC");

news.setAuthor("cc");

news.setDate(new Date());

news.setId(100);

System.out.println(news);

session.save(news);

System.out.println(news);

}

/\*\*

\* persist(): 也会执行 INSERT 操作

\*

\* 和 save() 的区别 :

\* 在调用 persist 方法之前, 若对象已经有 id 了, 则不会执行 INSERT, 而抛出异常

\*/

@Test

public void testPersist(){ //插入

News news = new News();

news.setTitle("EE");

news.setAuthor("ee");

news.setDate(new Date());

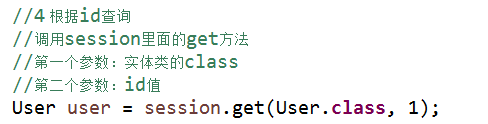
//news.setId(200);

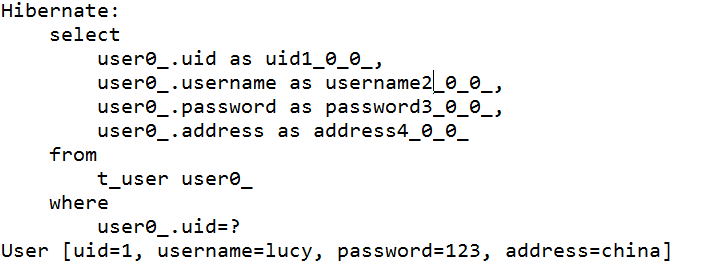
session.persist(news);

}

## (get, load)根据id查询—oid

1 调用session里面的get方法实现







@Test

public void testGet(){ //查询

News news = (News) session.get(News.class, 1);

// session.close();

System.out.println(news);

}

/\*\*

\* get VS load:

\*

\* 1. 执行 get 方法: 会立即加载对象.

\* 执行 load 方法, 若不适用该对象, 则不会立即执行查询操作, 而返回一个代理对象

\*

\* get 是 立即检索, load 是延迟检索.

\*

\* 2. load 方法可能会抛出 LazyInitializationException 异常: 在需要初始化

\* 代理对象之前已经关闭了 Session

\*

\* 3. 若数据表中没有对应的记录, Session 也没有被关闭.

\* get 返回 null

\* load 若不使用该对象的任何属性, 没问题; 若需要初始化了, 抛出异常.

\*/

@Test

public void testLoad(){ //查询

News news = (News) session.load(News.class, 10);

System.out.println(news.getClass().getName());

// session.close(); //若数据表中没有对应的记录, Session 也没有被关闭.

// System.out.println(news);

}

## (update)修改操作

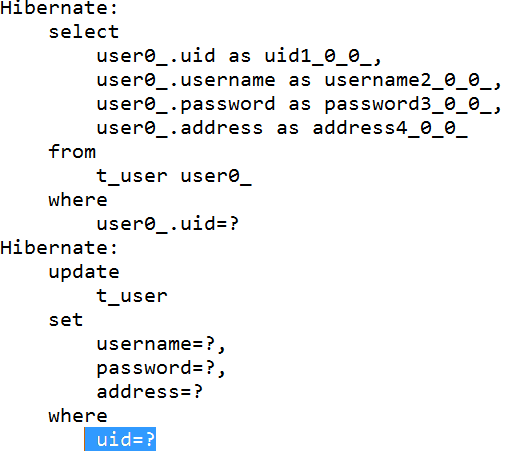
1 首先查询，修改值

（1）根据id查询，返回对象

也可以像delete那样定义实体类修改，不过在赋值时就需要写完所有字段，如果只写修改字段，得到的结果就只有 修改字段 ，其他字段就会为空 ；原因也简单，因为你新的实体类里面只有修改字段



2.sql输出



/\*\*

\* update:

\* 1. 若更新一个持久化对象, 不需要显示的调用 update 方法. 因为在调用 Transaction

\* 的 commit() 方法时, 会先执行 session 的 flush 方法.

\* 2. 更新一个游离对象, 需要显式的调用 session 的 update 方法. 可以把一个游离对象

\* 变为持久化对象

\*

\* 需要注意的:

\* 1. 无论要更新的游离对象和数据表的记录是否一致, 都会发送 UPDATE 语句.

\* 如何能让 updat 方法不再盲目的出发 update 语句呢 ? 在 .hbm.xml 文件的 class 节点设置

\* select-before-update=true (默认为 false). 但通常不需要设置该属性.

\*

\* 2. 若数据表中没有对应的记录, 但还调用了 update 方法, 会抛出异常

\*

\* 3. 当 update() 方法关联一个游离对象时,

\* 如果在 Session 的缓存中已经存在相同 OID 的持久化对象, 会抛出异常. 因为在 Session 缓存中

\* 不能有两个 OID 相同的对象!

\*

\*/

@Test

public void testUpdate(){

/\*//1.

News news = (News) session.get(News.class, 1);

news.setAuthor("SUN1"); \*/

//2.

News news = (News) session.get(News.class, 1);

transaction.commit();

session.close();

session = sessionFactory.openSession();

transaction = session.beginTransaction();

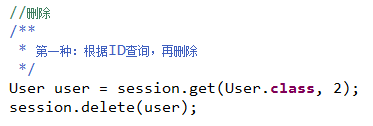
news.setAuthor("SUN2");

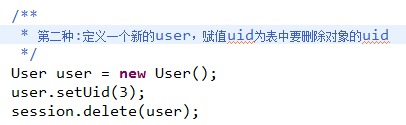
session.update(news);

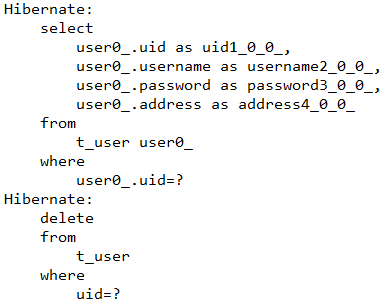
}

## (delete, evict)删除操作—oid

### delete







可以删除一个游离对象, 也可以删除一个持久化对象

Session 的 delete() 方法处理过程

计划执行一条 delete 语句

把对象从 Session 缓存中删除, 该对象进入删除状态.

### cfg.xml 配置文件

<!-- 删除对象后，使其OID置为null -->

<property name=*"use\_identifier\_rollback"*>true</property>

默认值为 false, 若把它设为 true, 将改变 delete() 方法的运行行为: delete() 方法会把持久化对象或游离对象的 OID 设置为 null, 使它们变为临时对象

/\*\*

\* delete: 执行删除操作. 只要 OID 和数据表中一条记录对应, 就会准备执行 delete 操作

\* 若 OID 在数据表中没有对应的记录, 则抛出异常

\*

\* 可以通过设置 hibernate 配置文件 hibernate.use\_identifier\_rollback 为 true,

\* 使删除对象后, 把其 OID 置为 null

\*/

@Test

public void testDelete(){

/\*News news = new News();

news.setId(5);

session.delete(news); \*/

News news = (News) session.get(News.class, 6);

session.delete(news);

System.out.println(news);

}

### 移除对象evict()

从 session 缓存中把指定的持久化对象移除

/\*\*

\* evict: 从 session 缓存中把指定的持久化对象移除

\*/

@Test

public void testEvict(){

News news1 = (News) session.get(News.class, 1);

news1.setTitle("AA1");

News news2 = (News) session.get(News.class, 2);

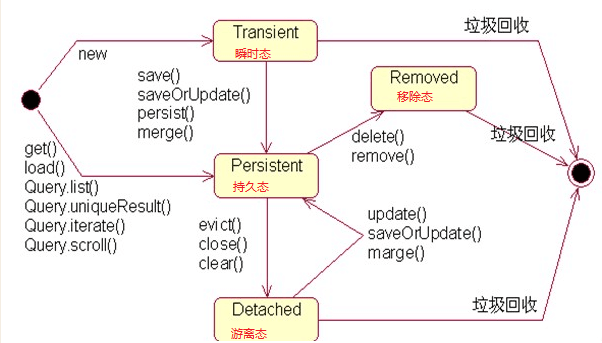
news2.setTitle("BB1");

session.evict(news1); //移除，1修改被跳过

}

# 实体类对象状态

## 状态转换图

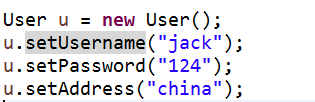




## 三种状态

### 瞬时态：

对象里面没有id值，对象与session没有关联



### 持久态：

对象里面有id值，对象与session关联



### 托管态：

对象有id值，对象与session没有关联



### 补充：移除态

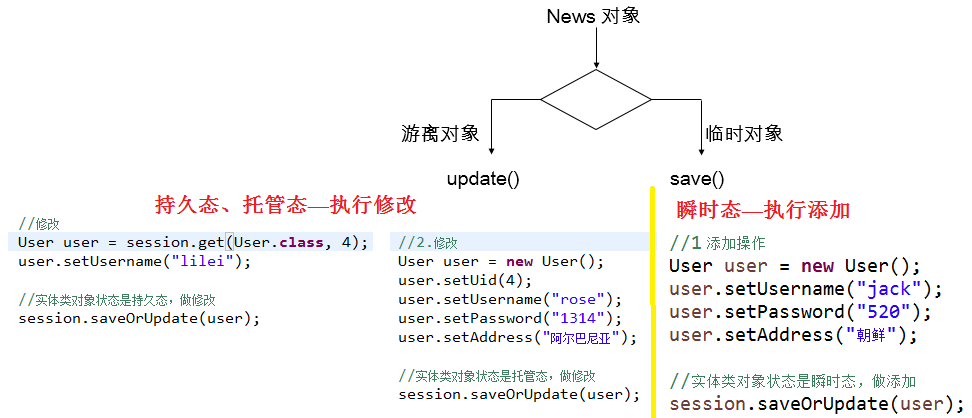
是实体从数据库删除后的状态

## saveOrUpdate

实现添加、实现修改

### 概述

同时包含了 save() 与 update() 方法的功能



### 定对象为临时对象的标准

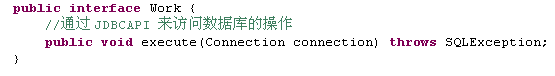
**Java 对象的 OID 为 null**

映射文件中为 <id> 设置了 **unsaved-value**  属性, 并且 Java 对象的 OID 取值与这个 unsaved-value 属性值匹配

### Session 的 merge() 方法

## 调用存储过程

Work 接口: 直接通过 JDBC API 来访问数据库的操作



Session 的 doWork(Work) 方法用于执行 Work 对象指定的操作, 即调用 Work 对象的 execute() 方法. Session 会把当前使用的数据库连接传递给 execute() 方法.

@Override

**public** **void** execute(Connection connection) **throws** SQLException {

System.***out***.println(connection);

//调用存储过程.

String sql = "{call testProcedure()}";

CallableStatement cstmt = connection.prepareCall(sql);

cstmt.executeUpdate();

}

});

## 与触发器协同工作

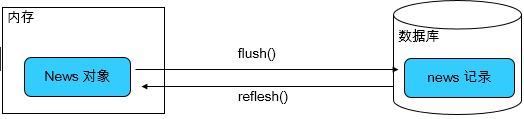
Hibernate 与数据库中的触发器协同工作时, 会造成两类问题

触发器使 Session 的缓存中的持久化对象与数据库中对应的数据不一致:触发器运行在数据库中, 它执行的操作对 Session 是透明的

Session 的 update() 方法盲目地激发触发器: 无论游离对象的属性是否发生变化, 都会执行 update 语句, 而 update 语句会激发数据库中相应的触发器

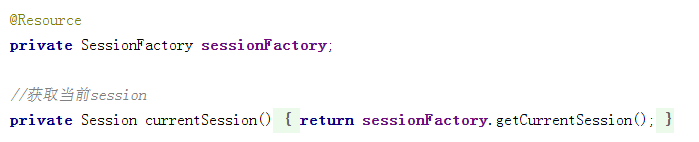
解决方案:

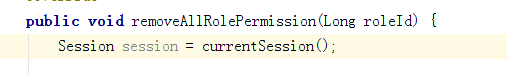
在执行完 Session 的相关操作后, 立即调用 Session 的 flush() 和 refresh() 方法, 迫使 Session 的缓存与数据库同步(refresh() 方法重新从数据库中加载对象)



在映射文件的的 <class> 元素中设置 select-before-update 属性: 当 Session 的 update 或 saveOrUpdate() 方法更新一个游离对象时, 会先执行 Select 语句, 获得当前游离对象在数据库中的最新数据, 只有在不一致的情况下才会执行 update 语句

# Session的获得





源代码文件：http://download.csdn.net/detail/qq\_26553781/9728141