# ****一、遇到的神奇的事情****

使用jpa操作数据库，当我使用findAll（）方法查处一个List的对象后，给对这个list的实体进行了一些操作，并没有调用update 或者 saveOrUpdate方法，更改后的数据却神奇的保存到数据库里面去了。

最后简单粗暴的解决办法是把这份从数据里面查出来的List 复制了一份，然后再操作，再返回。数据就正常了，数据库也没更新。后面找了资料才发现是jpa是对hibernate的封装，底层是hibernate，这是hibernate的持久状态搞的鬼。

# ****二、hibernate 的三种状态****

## ****１． 瞬时状态 (Transient)****

当我们通过Java的new关键字来生成一个实体对象时，这时这个实体对象就处于自由状态，如下：

Customer customer=new Customer(“zx”,27,images);

这时customer对象就处于自由状态，为什么说customer对象处于自由状态呢？这是因为，此时customer只是通过JVM获得了一块内存空间，还并没有通过Session对象的save()方法保存进数据库，因此也就还没有纳入Hibernate的缓存管理中，也就是说customer对象现在还自由的游荡于Hibernate缓存管理之外。所以我们可以看出自由对象最大的特点就是，在数据库中不存在一条与它对应的记录。

**瞬时对象特点：**

不和 Session 实例关联

在数据库中没有和瞬时对象关联的记录

## ****２． 持久状态 (Persistent)****

持久化对象就是已经被保存进数据库的实体对象，并且这个实体对象现在还处于Hibernate的缓存管理之中。这是对该实体对象的任何修改，都会在清理缓存时同步到数据库中。如下所示：

Customer customer=new Customer(“zx”,27,images);

tx=session.beginTransaction();

session.save(customer);

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

customer.setAge(28);

tx.commit();

这时我们并没有显示调用session.update()方法来保存更新，但是对实体对象的修改还是会同步更新到数据库中，因为此时customer对象通过save方法保存进数据库后，已经是持久化对象了，然后通过load方法再次加载它，它仍然是持久化对象，所以它还处于Hibernate缓存的管理之中，这时当执行tx.commit()方法时，Hibernate会自动清理缓存，并且自动将持久化对象的属性变化同步到到数据库中。

持久的实例在数据库中有对应的记录，并拥有一个持久化标识 (identifier).

持久对象总是与 Session 和 Transaction 相关联，在一个 Session 中，对持久对象的改变不会马上对数据库进行变更，而必须在 Transaction 终止，也就是执行 commit() 之后，才在数据库中真正运行 SQL 进行变更，持久对象的状态才会与数据库进行同步。在同步之前的持久对象称为脏 (dirty) 对象。

**瞬时对象转为持久对象：**

通过 Session 的 save() 和 saveOrUpdate() 方法把一个瞬时对象与数据库相关联，这个瞬时对象就成为持久化对象。

使用 fine(),get(),load() 和 iterater() 待方法查询到的数据对象，将成为持久化对象。

**持久化对象的特点：**

和 Session 实例关联

在数据库中有和持久对象关联的记录

## ****３． 脱管状态 (Detached)****

当一个持久化对象，脱离开Hibernate的缓存管理后，它就处于游离状态，游离对象和自由对象的最大区别在于，游离对象在数据库中可能还存在一条与它对应的记录，只是现在这个游离对象脱离了Hibernate的缓存管理，而自由对象不会在数据库中出现与它对应的数据记录。如下所示：

Customer customer=new Customer(“zx”,27,images);

tx=session.beginTransaction();

session.save(customer);

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

customer.setAge(28);

tx.commit();

session.close();

当session关闭后，customer对象就不处于Hibernate的缓存管理之中了，但是此时在数据库中还存在一条与customer对象对应的数据记录，所以此时customer对象处于游离态

与持久对象关联的 Session 被关闭后，对象就变为脱管对象。对脱管对象的引用依然有效，对象可继续被修改。

**脱管对象特点：**

本质上和瞬时对象相同

只是比爱瞬时对象多了一个数据库记录标识值 id.

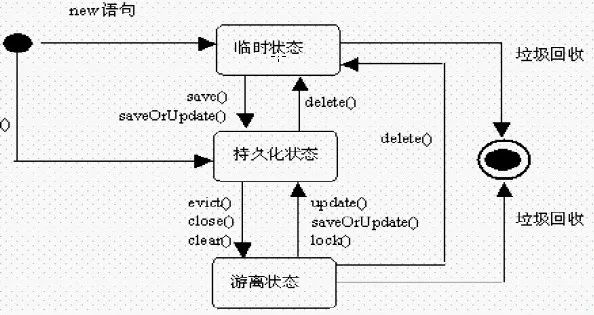
**持久对象转为脱管对象：**

当执行 close() 或 clear(),evict() 之后，持久对象会变为脱管对象。

**瞬时对象转为持久对象：**

通过 Session 的 update(),saveOrUpdate() 和 lock() 等方法，把脱管对象变为持久对象。

# ****三、三种状态的转换****



# ****四、举例子****

结合 save(),update(),saveOrUpdate() 方法说明对象的状态

(1) Save() 方法将瞬时对象保存到数据库，对象的临时状态将变为持久化状态。当对象在持久化状态时，它一直位于 Session 的缓存中，对它的任何操作在事务提交时都将同步到数据库，因此，对一个已经持久的对象调用 save()或 update() 方法是没有意义的。如：

Student stu = new Strudnet();

stu.setCarId(“200234567”);

stu.setId(“100”);

// 打开 Session, 开启事务

session.save(stu);

stu.setCardId(“20076548”);

session.save(stu); // 无效

session.update(stu); // 无效

// 提交事务，关闭 Session

(2) update() 方法两种用途重新关联脱管对象为持久化状态对象，显示调用 update() 以更新对象。调用 update() 只为了关联一个脱管对象到持久状态，当对象已经是持久状态时，调用 update() 就没有多大意义了。如：

// 打开 session ，开启事务

stu = (Student)session.get(Student.class,”123456”);

stu.setName(“Body”);

session.update(stu); // 由于 stu 是持久对象，必然位于 Session 缓冲中，

对 stu 所做的变更将 // 被同步到数据库中。所以 update() 是没有意义的，可以不要这句效果一样的。

// 提交事务，关闭 Session

Hibernate 总是执行 update 语句，不管这个脱管对象在离开 Session 之后有没有更改过，在清理缓存时 Hibernate总是发送一条 update 语句，以确保脱管对象和数据库记录的数据一致，如：

Student stu = new Strudnet();

stu.setCarId(“1234”);

// 打开 Session1, 开启事务

session1.save(stu);

// 提交事务，关闭 Session1

stu.set(“4567”); // 对脱管对象进行更改

// 打开 Session2, 开启事务

session2.update(stu);

// 提交事务，关闭 Session2

注：即使把 session2.update(stu); 这句去掉，提交事务时仍然会执行一条 update() 语句。

如果希望只有脱管对象改变了， Hibernate 才生成 update 语句，可以把映射文件中 <class> 标签的 select-before-update 设为 true, 这种会先发送一条 select 语句取得数据库中的值，判断值是否相同，如果相同就不执行 update语句。不过这种做法有一定的缺点，每次 update 语句之前总是要发送一条多余的 select 语句，影响性能。对于偶尔更改的类，设置才是有效的，对于经常要更改的类这样做是影响效率的。

(3) saveOrUpdate() 方法兼具 save() 和 update() 方法的功能，对于传入的对象， saveOrUpdate() 首先判断其是脱管对象还是临时对象，然后调用合适的方法。

# ****五、整合spring****

在使用Spring进行系统开发的时候，数据库连接一般都是配置在Spring的配置文件中，并且由Spring来管理的。在利用Spring + Hibernate进行开发时也是如此。下面是一个简单的Spring + Hibernate Dao的例子：

程序代码

****[c-sharp]**** [view plain](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "view plain" \t "https://blog.csdn.net/mafly008/article/details/_blank)[copy](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "copy" \t "https://blog.csdn.net/mafly008/article/details/_blank)

1. **public** **class** DaoReal extends HibernateDaoSupport implements Dao {
2. **public** List<User> getAll() {
3. **return** super.getHibernateTemplate().find("from User");
4. }
5. }

　在上面的这个例子中，我们并没有关闭Session，但程序并没有任何问题，那是因为Spring已经帮我们关闭了。那么再看一个例子：

程序代码

****[c-sharp]**** [view plain](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "view plain" \t "https://blog.csdn.net/mafly008/article/details/_blank)[copy](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "copy" \t "https://blog.csdn.net/mafly008/article/details/_blank)

1. **public** **class** DaoReal extends HibernateDaoSupport implements Dao {
2. **public** List<User> getAll() {
3. **return** super.getSession().createQuery("from User").list();
4. }
5. }

这个例子会不会有问题呢？的确，上面的例子中隐藏了一个问题，数据库连接并没有被关闭，在我们的印象中这件事似乎应该是Spring的。程序执行后，好像也没有什么问题，但是连续执行该语句n次（n<=最大连接数，如果没有指定最大连接数，那么默认为10次）后，系统处于等待状态，不会继续执行了，控制台上并没有输出任何信息。打开log文件，发现系统抛出java.lang.IllegalStateException: Pool not open的异常，无法打开连接。这说明系统连接池中所有的连接都在使用中。那么我们手动关闭Session后，应该就没有问题了吧！是这样吗？修改我们的例子，如下：

程序代码

****[c-sharp]**** [view plain](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "view plain" \t "https://blog.csdn.net/mafly008/article/details/_blank)[copy](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "copy" \t "https://blog.csdn.net/mafly008/article/details/_blank)

1. **public** **class** DaoReal extends HibernateDaoSupport implements Dao {
2. **public** List<User> getAll() {
3. Session s = super.getSession();
4. **try** {
5. **return** s.createQuery("from User").list();
6. } **finally** {
7. s.close();
8. }
9. ｝
10. }

执行n遍后依然停止响应。问题出在哪里呢？其实Spring的Session总是与某个线程绑定的，而这个线程往往就是承载Servlet或Jsp的那个线程，也就是说，它的生命周期scope是request的。在上面的例子中，我们利用getSession强制获得了Hibernate的 Session，这个Session可能是当前事务中之前使用过的，或者可能是一个新的，并不在当前事务中，Spring只对当前事务中的Session 进行关闭。

这里有3个解决方案：  
方案一：

用此种方法，虽然没有手动关闭数据库连接，但spring已经帮我们关闭了

****[c-sharp]**** [view plain](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "view plain" \t "https://blog.csdn.net/mafly008/article/details/_blank)[copy](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "copy" \t "https://blog.csdn.net/mafly008/article/details/_blank)

1. **return** super.getHibernateTemplate().find(hql);

方案二：（我用这方法解决的）

设定HibernateTemplate的AllowCreate为True，允许创建一个新的session

Java代码

****[c-sharp]**** [view plain](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "view plain" \t "https://blog.csdn.net/mafly008/article/details/_blank)[copy](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "copy" \t "https://blog.csdn.net/mafly008/article/details/_blank)

1. **public** **class** ItemDAOImpl extends HibernateDaoSupport implements ItemDAO {
3. **public** List queryAll() throws Exception {
5. Session session=super.getSession(**true**);
7. String hql="from Item as i";
9. **try**{
11. List l=session.createQuery(hql).list();
13. }**finally**{
15. session.close();
16. }
17. }
18. }

方案三：

Spring API:  
geSession()是org.springframework.orm.hibernate3.support.HibernateDaoSupport 中的一个方法，  
它可以从当前事务或者一个新的事务获得一个hibernate　session.  
通常使用releaseSession(org.hibernate.Session)方法与getSession()配合。  
如果没有绑定线程，releaseSession关闭由这个DAO的SessionFactory创建的Hibernate Session。　  
修改后的代码如下：

Java代码

****[c-sharp]**** [view plain](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "view plain" \t "https://blog.csdn.net/mafly008/article/details/_blank)[copy](http://blog.csdn.net/kbeanwu/article/details/5744147" \o "copy" \t "https://blog.csdn.net/mafly008/article/details/_blank)

1. **public** **class** ItemDAOImpl extends HibernateDaoSupport implements ItemDAO {
3. **public** List queryAll() throws Exception {
5. Session session = super.getSession();
7. String hql = "from Item as i";
9. List l = session.createQuery(hql).list();
11. releaseSession(session);
13. }