hibernate乐观锁与悲观锁

在使用hibernate的过程我们会遇到多个人对同一数据同时进行修改，这个时候就会发生脏数据，造成数据的不一致性。为了避免更新数据的丢失，Hibernate采用锁机制。Hibernate提供了两种锁机制：悲观锁和乐观锁。

**悲观锁**：在数据有加载的时候就给其进行加锁，直到该锁被释放掉，其他用户才可以进行修改。

**乐观锁**：在对数据进行修改的时候，对数据采用版本号或者时间戳等方式来比较，数据是否一致性来实现加锁。

# 悲观锁

悲观锁是依靠数据库提供的锁机制。Hibernate是通过使用数据库的for update子句实现了悲观锁机制。Hibernate有如下五种加锁机制

* LockMode.NONE：无锁机制
* LockMode.WRITE：Hibernate在Insert和Update记录的时候会自动获取
* LockMode.READ：Hibernate在读取记录的时候会自动获取
* LockMode.UPGRADE：利用数据库的for update子句加锁
* LockMode.UPGRADE\_NOWAIT：Oracle的特定实现，利用Oracle的for update nowait子句实现加锁

悲观加锁一般通过以下三种方法实现：

* Criteria.setLockMode
* Query.setLockMode
* Session.lock

下面示例是对查询进行加锁：

|  |
| --- |
| public void query(int id){  Session session = HibernateUtil.getSession();  Transaction tx = session.beginTransaction();  String hql = "from Users as u where id= :id";  List list = session.createQuery(hql)  .setLockMode("u", LockMode.UPGRADE) //执行加锁  .setInteger("id", id)  .list();  for(Iterator iterator = list.iterator();iterator.hasNext();){  Users users = (Users) iterator.next();  System.out.println(users.getBirthday());  }  } |
| 产生的SQL语句：select users0\_.id as id0\_, users0\_.ver as ver0\_,  users0\_.birthday as birthday0\_, users0\_.first\_name as first4\_0\_, users0\_.last\_name as last5\_0\_ from Users users0\_  with (updlock, rowlock) where users0\_.id=?  悲观锁在对数据进行加锁后，会一直“霸占”该数据，直到释放掉，其他用户才可以对该数据进行更新。这里就存在一个问题，如果它一直占着不放，那么其他用户永远也不可能对该数据进行更新，这样就很不利于并发了。对于这个问题的解决方案，可以利用乐观锁。 |

# 乐观锁

乐观锁大多是基于数据版本记录机制实现。何谓数据版本？即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据库表增加一个“version”字段来实现。读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。我们可以通过class描述符的optimistic-lock属性结合version描述符指定乐观锁。

* none：无乐观锁
* version：通过版本机制实现乐观锁
* dirty：通过检查发生变动过的属性实现乐观锁
* all：通过检查所有属性实现乐观锁

实现乐观锁的持久化类我们需要为该持久化类增加一个version属性，并且提供相应的getter和setter方法。如下

|  |
| --- |
| public class Users {  private int id;  private Date birthday;  private Name name;  private int version;  //舍掉getter和setter方法  } |
| 配置文件：  <hibernate-mapping package="com.hibernate.domain">  <class name="Users" optimistic-lock="version">  <id name="id">  <generator class="native" />  </id>  <version name="version" />    <property name="birthday" />    <!-- 映射组件元素 -->  <component name="name">  <!-- 映射组件的name属性指向包含实体 -->  <property name="firstName" column="first\_name"/>  <property name="lastName" column="last\_name"/>  </component>  </class>  </hibernate-mapping>  注意：version 节点必须出现在ID 节点之后。  在这里我们声明了一个version属性，该属性用于存放用户的版本信息。我们对user表每一次更新操作，都会引起version属性的变化：加1。如果我们尝试在tx.commit 之前，启动另外一个Session，对名为同一个用户进行操作，就是并发更新的情形了：  public void update(){  //开启事务tx1  Session session1 = HibernateUtil.getSession();  Transaction tx1 = session1.beginTransaction();  Users users1 = (Users) session1.get(Users.class, 1); //获取id为1的用户    //开启事务tx2  Session session2 = HibernateUtil.getSession();  Transaction tx2 = session2.beginTransaction();  Users users2 = (Users) session2.get(Users.class, 1); //获取id为1的用户    users1.getName().setFirstName("first name1");  users2.getName().setFirstName("first name2");    tx1.commit(); //..........1  tx2.commit(); //..........2    session1.close();  session2.clear();    } |

执行以上代码，代码将在.....2处抛出StaleObjectStateException异 常，并指出版本检查失败。



在这里是先提交者成功，后提交者失败。当前事务正在试图提交一个过期数据。通过捕捉这个异常，我们就可以在乐观锁校验失败时进行相应处理。