雨小烛的空间

Hibernate的flush机制

博客分类：
- Hibernate

Hibernate多线程.netAccess

随着Hibernate在Java开发中的广泛应用，我们在使用Hibernate进行对象持久化操作中也遇到了各种各样的问题。这些问题往往都是我们对Hibernate缺乏了解所致，这里我讲个我从前遇到的问题及一些想法，希望能给大家一点借鉴。

这是在一次事务提交时遇到的异常。

an assertion failure occured (this may indicate a bug in Hibernate, but is more likely due to unsafe use of the session)
net.sf.hibernate.AssertionFailure: possible nonthreadsafe access to session注：非possible non-threadsafe access to the session （那是另外的错误，类似但不一样）

这个异常应该很多的朋友都遇到过，原因可能各不相同。但所有的异常都应该是在flush或者事务提交的过程中发生的。这一般由我们在事务开始至事务提交的过程中进行了不正确的操作导致，也会在多线程同时操作一个Session时发生，这里我们仅仅讨论单线程的情况，多线程除了线程同步外基本与此相同。

至于具体是什么样的错误操作那？我给大家看一个例子（假设Hibernate配置正确，为保持代码简洁，不引入包及处理任何异常）

Java代码

```
1. SessionFactory sf = new Configuration().configure().buildSessionFactory() ;
2. Session s = sf.openSession();
3. Cat cat = new Cat();
4.
5. Transaction tran = s.beginTransaction(); (1)
6. s.save(cat); (2) ( 此处同样可以为update delete )
7. s.evict(cat); (3)
8. tran.commit(); (4)
9. s.close();(5)
```

这就是引起此异常的典型错误。我当时就遇到了这个异常，检查代码时根本没感觉到这段代码出了问题，想当然的认为在Session上开始一个事务，通过Session将对象存入数据库，再将这个对象从Session上拆离，提交事务，这是一个很正常的流程。如果这里正常的话，那问题一定在别处。

问题恰恰就在这里，我的想法也许没有错，但是一个错误的论据所引出的观点永远都不可能是正确的。因为我一直以为直接在对数据库进行操作，忘记了在我与数据库之间隔了一个Hibernate，Hibernate在为我们提供持久化服务的同时，也改变了我们对数据库的操作方式，这种方式与我们直接的数据库操作有着很多的不同，正因为我们对这种方式没有一个大致的了解造成了我们的应用并未得到预先设想的结果。

那Hibernate的持久化机制到底有什么不同那？简单的说，Hibernate在数据库层之上实现了一个缓存区，当应用save或者update一个对象时，Hibernate并未将这个对象实际的写入数据库中，而仅仅是在缓存中根据应用的行为做了登记，在真正需要将缓存中的数据flush入数据库时才执行先前登记的所有行为。

在实际执行的过程中，每个Session是通过几个映射和集合来维护所有与该Session建立了关联的对象以及应用对这些对象所进行的操作的，与我们这次讨论有关的有entityEntries（与Session相关联的对象的映射）、insertions（所有的插入操作集合）、deletions（删除操作集合）、updates（更新操作集合）。下面我就开始解释在最开始的例子中，Hibernate到底是怎样运作的。
(1)生成一个事务的对象，并标记当前的Session处于事务状态（注：此时并未启动数据库级事务）。
(2)应用使用s.save保存cat对象，这个时候Session将cat这个对象放入entityEntries，用来标记cat已经和当前的会话建立了关联，由于应用对cat做了保存的操作，Session还要在insertions中登记应用的这个插入行为（行为包括：对象引用、对象id、Session、持久化处理类）。
(3)s.evict(cat)将cat对象从s会话中拆离，这时s会从entityEntries中将cat这个对象移出。
(4)事务提交，需要将所有缓存flush入数据库，Session启动一个事务，并按照insert,update,……,delete的顺序提交所有之前登记的操作（注意：所有insert执行完毕后才会执行update，这里的特殊处理也可能会将你的程序搞得一团糟，如需要控制操作的执行顺序，要善于使用flush），现在cat不在entityEntries中，但在执行insert的行为时只需要访问insertions就足够了，所以此时不会有任何的异常。异常出现在插入后通知Session该对象已经插入完毕这个步骤上，这个步骤中需要将entityEntries中cat的existsInDatabase标志置为true，由于cat并不存在于entityEntries中，此时Hibernate就认为insertions和entityEntries可能因为线程安全的问题产生了不同步（也不知道Hibernate的开发者是否考虑到例子中的处理方式，如果没有的话，这也许算是一个bug吧），于是一个net.sf.hibernate.AssertionFailure就被抛出，程序终止。

我想现在大家应该明白例子中的程序到底哪里有问题了吧，我们的错误的认为s.save会立即的执行，而将cat对象过早的与Session拆离，造成了Session的insertions和entityEntries中内容的不同步。所以我们在做此类操作时一定要清楚Hibernate什么时候会将数据flush入数据库，在未flush之前不要将已进行操作的对象从Session上拆离。

对于这个错误的解决方法是，我们可以在(2)和(3)之间插入一个s.flush()强制Session将缓存中的数据flush入数据库（此时Hibernate会提前启动事务，将(2)中的save登记的insert语句登记在数据库事务中，并将所有操作集合清空），这样在(4)事务提交时insertions集合就已经是空的了，即使我们拆离了cat也不会有任何的异常了。
前面简单的介绍了一下Hibernate的flush机制和对我们程序可能带来的影响以及相应的解决方法，Hibernate的缓存机制还会在其他的方面给我们的程序带来一些意想不到的影响。看下面的例子：
（name为cat表的主键）
Java代码

```
1. Cat cat = new Cat();
2. cat.setName(“tom”);
3. s.save(cat);
4.
5. cat.setName(“mary”);
6. s.update(cat);(6)
7.
8. Cat littleCat = new Cat();
9. littleCat.setName(“tom”);
10. s.save(littleCat);
11.
12. s.flush();
```

这个例子看起来有什么问题？估计不了解Hibernate缓存机制的人多半会说没有问题，但它也一样不能按照我们的思路正常运行，在flush过程中会产生主键冲突，可能你想问：“在save(littleCat)之前不是已经更改cat.name并已经更新了么？为什么还会发生主键冲突那？”这里的原因就是我在解释第一个例子时所提到的缓存flush顺序的问题，Hibernate按照insert,update,……,delete的顺序提交所有登记的操作，所以你的s.update(cat)虽然在程序中出现在s.save(littleCat)之前，但是在flush的过程中，所有的save都将在update之前执行，这就造成了主键冲突的发生。

这个例子中的更改方法一样是在(6)之后加入s.flush()强制Session在保存littleCat之前更新cat的name。这样在第二次flush时就只会执行s.save(littleCat)这次登记的动作，这样就不会出现主键冲突的状况。

再看一个例子（很奇怪的例子，但是能够说明问题）

Java代码

```
1. Cat cat = new Cat();
2. cat.setName( “tom” );
3. s.save(cat); (7)
4. s.delete(cat);(8)
5.
6. cat.id=null;(9)
7. s.save(cat);(10)
8. s.flush();
```

这个例子在运行时会产生异常net.sf.hibernate.HibernateException: identifier of an instance of Cat altered from 8b818e920a86f038010a86f03a9d0001 to null

这里例子也是有关于缓存的问题，但是原因稍有不同：
（7）和（2）的处理相同。
（8）Session会在deletions中登记这个删除动作，同时更新entityEntries中该对象的登记状态为DELETED。
（9）Cat类的标识符字段为id,将其置为null便于重新分配id并保存进数据库。
（10）此时Session会首先在entityEntries查找cat对象是否曾经与Session做过关联，因为cat只改变了属性值，引用并未改变，所以会取得状态为DELETED的那个登记对象。由于第二次保存的对象已经在当前Session中删除，save会强制Session将缓存flush才会继续，flush的过程中首先要执行最开始的save动作，在这个save中检查了cat这个对象的id是否与原来执行动作时的id相同。不幸的是，此时cat的id被赋为null，异常被抛出，程序终止（此处要注意，我们在以后的开发过程尽量不要在flush之前改变已经进行了操作的对象的id）。

这个例子中的错误也是由于缓存的延时更新造成的（当然，与不正规的使用Hibernate也有关系），处理方法有两种：
1、在（8）之后flush，这样就可以保证（10）处save将cat作为一个全新的对象进行保存。
2、删除（9），这样第二次save所引起的强制flush可以正常的执行，在数据库中插入cat对象后将其删除，然后继续第二次save重新插入cat对象，此时cat的id仍与从前一致。

这两种方法可以根据不同的需要来使用，呵呵，总觉得好像是很不正规的方式来解决问题，但是问题本身也不够正规，只希望能够在应用开发中给大家一些帮助，不对的地方也希望各位给与指正。

总的来说，由于Hibernate的flush处理机制，我们在一些复杂的对象更新和保存的过程中就要考虑数据库操作顺序的改变以及延时flush是否对程序的结果有影响。如果确实存在着影响，那就可以在需要保持这种操作顺序的位置加入flush强制Hibernate将缓存中记录的操作flush入数据库，这样看起来也许不太美观，但很有效。


老房改造装修

40平小复式装修

柴犬转让

虚拟办公室

希腊买房移民

分享到：
Java 获取客户端代码大全 | 使用apache commons-fileupload.jar 实现文 ...

- 2008-10-20 16:24
- 浏览 1063
- 评论(0)
- 查看更多

相关资源推荐

- hibernate的flush机制
- Hibernate 中的session 的flush、reflush 和clear 方法 ，及数据库的隔离级别
- Hibernate中Session的flush方法介绍
- hibernate flush 机制
- hibernate session中clear、evict、flush方法的区别
- hibernate的事务处理机制以及flush方法的作用
- hibernate中flush()、refresh()、clear()缓存操作
- spring管理的hibernate事务不会自动flush的问题-今天真遇到这问题了
- hibernate中的session.flush()和commit()的区别
- 深入Hibernate的flush机制

浅谈Hibernate的flush机制
Hibernate的flush机制详解
Hibernate session FlushMode有五种属性
springMVC整合hibernate的时候数据插入需要flush问题
hibernate在事务中的session.flush无效

Hibernate的flush机制深入
浅谈Hibernate的flush机制
Hibernate Clear 与 Flush 方法
Hibernate中持久化上下文的flush操作之一AUTO
Hibernate中session.flush()会不会去数据库执行SQL语句

评论

发表评论


您还没有登录,请您登录后再发表评论

声明：ITeye文章版权属于作者，受法律保护。没有作者书面许可不得转载。若作者同意转载，必须以超链接形式标明文章原始出处和作者。
© 2003-2018 ITeye.com. All rights reserved. [京ICP证110151号 京公网安备110105010620]