已锁定 主题：**hibernate二级缓存攻略**

该帖已经被评为精华帖

作者 正文

AreYouOK?

等级:

文章: 46

积分: 390

来自: 北京

我现在离线

发表时间：2006-03-03

很多人对二级缓存都不太了解，或者是有错误的认识，我一直想写一篇文章介绍一下hibernate的二级缓存的，今天终于忍不住了。
我的经验主要来自hibernate2.1版本，基本原理和3.0、3.1是一样的，请原谅我的顽固不化。

hibernate的session提供了一级缓存，每个session，对同一个id进行两次load，不会发送两条sql给数据库，但是session关闭的时候，一级缓存就失效了。

二级缓存是SessionFactory级别的全局缓存，它底下可以使用不同的缓存类库，比如ehcache、oscache等，需要设置hibernate.cache.provider_class，我们这里用ehcache，在2.1中就是
hibernate.cache.provider_class=net.sf.hibernate.cache.EhCacheProvider
如果使用查询缓存，加上
hibernate.cache.use_query_cache=true

相关文章:

- 神啊save我吧，我快被中文的《深入java虚拟机(2nd)》搞疯了
- 希望大家多讨论一些软件方面的问题
- 烂译列表
- Learning GNU Emacs or Programming in Emacs Lisp
- Apple笔记本（400元）

推荐圈子: JBoss SEAM
更多相关推荐

缓存可以简单的看成一个Map，通过key在缓存里面找value。

Class的缓存
对于一条记录，也就是一个PO来说，是根据ID来找的，缓存的key就是ID，value是POJO。无论list，load还是iterate，只要读出一个对象，都会填充缓存。但是list不会使用缓存，而iterate会先取数据库select id出来，然后一个id一个id的load，如果在缓存里面有，就从缓存取，没有的话就去数据库load。假设是读写缓存，需要设置：
<cache usage="read-write"/>
如果你使用的二级缓存实现是ehcache的话，需要配置ehcache.xml
<cache name="com.xxx.pojo.Foo" maxElementsInMemory="500" eternal="false" timeToLiveSeconds="7200" timeToIdleSeconds="3600" overflowToDisk="true" />
其中eternal表示缓存是不是永远不超时，timeToLiveSeconds是缓存中每个元素（这里也就是一个POJO）的超时时间，如果eternal="false"，超过指定的时间，这个元素就被移走了。timeToIdleSeconds是发呆时间，是可选的。当往缓存里面put的元素超过500个时，如果overflowToDisk="true"，就会把缓存中的部分数据保存在硬盘上的临时文件里面。
每个需要缓存的class都要这样配置。如果你没有配置，hibernate会在启动的时候警告你，然后使用defaultCache的配置，这样多个class会共享一个配置。
当某个ID通过hibernate修改时，hibernate会知道，于是移除缓存。
这样大家可能会想，同样的查询条件，第一次先list，第二次再iterate，就可以使用到缓存了。实际上这是很难的，因为你无法判断什么时候是第一次，而且每次查询的条件通常是不一样的，假如数据库里面有100条记录，id从1到100，第一次list的时候出了前50个id，第二次iterate的时候却查询到30至70号id，那么30-50是从缓存里面取的，51到70是从数据库取的，共发送1+20条sql。所以我一直认为iterate没有什么用，总是会有1+N的问题。
（题外话：有说法说大型查询用list会把整个结果集装入内存，很慢，而iterate只select id比较好，但是大型查询总是要分页查的，谁也不会真的把整个结果集装进来，假如一页20条的话，iterate共需要执行21条语句，list虽然选择若干字段，比iterate第一条select id语句慢一些，但只有一条语句，不装入整个结果集hibernate还会根据数据库方言做优化，比如使用mysql的limit，整体看来应该还是list快。）
如果想要对list或者iterate查询的结果缓存，就要用到查询缓存了

查询缓存
首先需要配置hibernate.cache.use_query_cache=true

如果用ehcache，配置ehcache.xml，注意hibernate3.0以后不是net.sf的包名了
<cache name="net.sf.hibernate.cache.StandardQueryCache"
  maxElementsInMemory="50" eternal="false" timeToIdleSeconds="3600"
  timeToLiveSeconds="7200" overflowToDisk="true"/>
<cache name="net.sf.hibernate.cache.UpdateTimestampsCache"
  maxElementsInMemory="5000" eternal="true" overflowToDisk="true"/>
然后
query.setCacheable(true);//激活查询缓存
query.setCacheRegion("myCacheRegion");//指定要使用的cacheRegion，可选
第二行指定要使用的cacheRegion是myCacheRegion，即你可以给每个查询缓存做一个单独的配置，使用setCacheRegion来做这个指定，需要在ehcache.xml里面配置它：
<cache name="myCacheRegion" maxElementsInMemory="10" eternal="false" timeToIdleSeconds="3600" timeToLiveSeconds="7200" overflowToDisk="true" />
如果省略第二行，不设置cacheRegion的话，那么会使用上面提到的标准查询缓存的配置，也就是net.sf.hibernate.cache.StandardQueryCache

对于查询缓存来说，缓存的key是根据hql生成的sql，再加上参数，分页等信息（可以通过日志输出看到，不过它的输出不是很可读，最好改一下它的代码）。
比如hql：
from Cat c where c.name like ?
生成大致如下的sql：
select * from cat c where c.name like ?
参数是"tiger%"，那么查询缓存的key*大约*是这样的字符串（我是凭记忆写的，并不精确，不过看了也该明白了）：
select * from cat c where c.name like ? , parameter:tiger%
这样，保证了同样的查询、同样的参数等条件下具有一样的key。
现在说说缓存的value，如果是list方式的话，value在这里并不是整个结果集，而是查询出来的这一串ID。也就是说，不管是list方法还是iterate方法，第一次查询的时候，它们的查询方式很它们平时的方式是一样的，list执行一条sql，iterate执行1+N条，多出来的行为是它们填充了缓存。但是到同样条件第二次查询的时候，就都和iterate的行为一样了，根据缓存的key去缓存里面查到了value，value是一串id，然后在到class的缓存里面去一个一个的load出来。这样做是为了节约内存。
可以看出来，查询缓存需要打开相关类的class缓存。list和iterate方法第一次执行的时候，都是既填充查询缓存又填充class缓存的。
这里还有一个很容易被忽视的重要问题，即打开查询缓存以后，即使是**list**方法也可能遇到**1+N**的问题！相同条件第一次list的时候，因为查询缓存中找不到，不管class缓存是否存在数据，总是发送一条sql语句到数据库获取全部数据，然后填充查询缓存和class缓存。但是第二次执行的时候，问题就来了，如果你的class缓存的超时时间比较短，现在class缓存都超时了，但是查询缓存还在，那么list方法在获取id串以后，将会一个一个去数据库load！因此，class缓存的超时时间一定不能短于查询缓存设置的超时时间！如果还设置了发呆时间的话，保证class缓存的发呆时间也大于查询的缓存的生存时间。这里还有其他情况，比如class缓存被程序强制evict了，这种情况就请自己注意了。

另外，如果hql查询包含select字句，那么查询缓存里面的value就是整个结果集了。

当hibernate更新数据库的时候，它怎么知道更新哪些查询缓存呢？
hibernate在一个地方维护每个表的最后更新时间，其实也就是放在上面net.sf.hibernate.cache.UpdateTimestampsCache所指定的缓存配置里面。
当通过hibernate更新的时候，hibernate会知道这次更新影响了哪些表。然后它更新这些表的最后更新时间。每个缓存都有一个生成时间和这个缓存所查询的表，当hibernate查询一个缓存是否存在的时候，如果缓存存在，它还要取出缓存的生成时间和这个缓存所查询的表，然后去查找这些表的最后更新时间，如果有一个表在生成时间后更新过了，那么这个缓存是无效的。
可以看出，只要更新过一个表，那么凡是涉及到这个表的查询缓存就失效了，因此查询缓存的命中率可能会比较低。

Collection缓存
需要在hbm的collection里面设置
<cache usage="read-write"/>
假如class是Cat，collection叫children，那么ehcache里面配置
<cache name="com.xxx.pojo.Cat.children"
  maxElementsInMemory="20" eternal="false" timeToIdleSeconds="3600" timeToLiveSeconds="7200"
  overflowToDisk="true" />
Collection的缓存和前面查询缓存的list一样，也是只保持一串id，但它不会因为这个表更新过就失效，一个collection缓存仅在这个collection里面的元素有增删时才失效。
这样有一个问题，如果你的collection是根据某个字段排序的，当其中一个元素更新了该字段时，导致顺序改变时，collection缓存里面的顺序没有做更新。

缓存策略
只读缓存（read-only）：没有什么好说的
读/写缓存（read-write）:程序可能要的更新数据
不严格的读/写缓存（nonstrict-read-write）：需要更新数据，但是两个事务更新同一条记录的可能性很小，性能比读写缓存好
事务缓存（transactional）：缓存支持事务，发生异常的时候，缓存也能够回滚，只支持jta环境，这个我没有怎么研究过

读写缓存和不严格读写缓存在实现上的区别在于，读写缓存更新缓存的时候会把缓存里面的数据换成一个锁，其他事务如果去取相应的缓存数据，发现被锁住了，然后就直接取数据库查询。
在hibernate2.1的ehcache实现中，如果锁住部分缓存的事务发生了异常，那么缓存会一直被锁住，直到60秒后超时。

不严格读写缓存不锁定缓存中的数据。

使用二级缓存的前置条件
你的hibernate程序对数据库有独占的写访问权，其他的进程更新了数据库，hibernate是不可能知道的。你操作数据库必需直接通过hibernate，如果你调用存储过程，或者自己使用jdbc更新数据库，hibernate也是不知道的。hibernate3.0的大批量更新和删除是不更新二级缓存的，但是据说3.1已经解决了这个问题。
这个限制相当的棘手，有时候hibernate做批量更新、删除很慢，但是你却不能自己写jdbc来优化，很郁闷吧。
SessionFactory也提供了移除缓存的方法，你一定要自己写一些JDBC的话，可以调用这些方法移除缓存，这些方法是：

```
void evict(Class persistentClass)
          Evict all entries from the second-level cache.
void evict(Class persistentClass, Serializable id)
          Evict an entry from the second-level cache.
void evictCollection(String roleName)
          Evict all entries from the second-level cache.
void evictCollection(String roleName, Serializable id)
          Evict an entry from the second-level cache.
void evictQueries()
          Evict any query result sets cached in the default query cache region.
void evictQueries(String cacheRegion)
          Evict any query result sets cached in the named query cache region.
```

不过我不建议这样做，因为这样很难维护。比如你现在用JDBC批量更新了某个表，有3个查询缓存会用到这个表，用evictQueries(String cacheRegion)移除了3个查询缓存，然后用evict(Class persistentClass)移除了class缓存，看上去好像完整了。不过哪天你添加了一个相关查询缓存，可能会忘记更新这里的移除代码。如果你的jdbc代码到处都是，在你添加一个查询缓存的时候，还知道其他什么地方也要做相应的改动吗？

总结：
不要想当然的以为缓存一定能提高性能，仅仅在你能够驾驭它并且条件合适的情况下才是这样的。hibernate的二级缓存限制还是比较多的，不方便用jdbc可能会大大的降低更新性能。在不了解原理的情况下乱用，可能会有1+N的问题。不当的使用还可能导致读出脏数据。
如果受不了hibernate的诸多限制，那么还是自己在应用程序的层面上做缓存吧。
在越高的层面上做缓存，效果就会越好。就好像尽管磁盘有缓存，数据库还是要实现自己的缓存，尽管数据库有缓存，咱们的应用程序还是要做缓存。因为底层的缓存它并不知道高层要用这些数据干什么，只能做的比较通用，而高层可以有针对性的实现缓存，所以在更高的级别上做缓存，效果也要好些吧。

终于写完了，好累......

返回顶楼

hongliang　　发表时间：2006-03-03

等级：

文章: 464
积分: 1129
来自: 上海
我现在离线

楼主理解的透彻，佩服。特别赞同最后一句话，“在越高的层面上做缓存，效果就会越好”。我是一直热衷于页面html化。。。

返回顶楼 回帖地址 0 0 请登录后投票

hintcnuie
等级:

文章: 60
积分: 137
来自: 北京
我现在离线

发表时间：2006-03-06

深入浅出，看过不少讲缓存的，但还是楼主讲的最透彻!

返回顶楼 回帖地址 0 0 请登录后投票

flyjie
等级:

文章: 112
积分: 446
来自: 杭州
我现在离线

发表时间：2006-03-06

写得好详细啊! 大家讨论一下在JTA环境下hibernate2用什么缓存策略。如A数据库读写频繁，但数据库量不大，B数据库查询多。C数据库是远程的.....目前hibernate大部分二缓存好像都不支持并发transactional~~

返回顶楼 回帖地址 0 0 请登录后投票

Allen
等级:

文章: 409
积分: 572
来自: Inner Party
我现在离线

发表时间：2006-03-09

关于Hibernate Second-Level Cache，Devx上面有一篇比较详尽的文章，值得参考：

http://www.devx.com/dbzone/Article/29685/1954?pf=true

返回顶楼 回帖地址 0 0 请登录后投票

hzlinux
等级:

发表时间：2006-03-12

高层缓存 比较难作，但比较有针对性

但是对开发人员的要求也更高

文章: 11
积分: 151
我现在离线

返回顶楼 回帖地址 0 0 请登录后投票

stevendu
等级:

发表时间：2006-03-16

已仔细拜读LZ大作，并打印收藏。
只是在缓存策略那一小节还有些模糊：
1：读/写缓存与不严格读写缓存的区别大致了解，但是在应用的时候应该如何选择还是个问题。
2：缓存中的数据被锁住之后，其他缓存直接从数据库中取数据，这样就造成数据不一致了吧？ 这时是否应该采用不严格的读写缓存？

文章: 19
积分: 243
我现在离线

返回顶楼 回帖地址 0 0 请登录后投票

zhaoxiqian
等级: 初级会员

发表时间：2006-03-22

好好，楼主讲的正是我需要的，为什么我前几天来的时候没有看到呢。。
该打```

文章: 8
积分: 87
我现在离线

返回顶楼 回帖地址 0 0 请登录后投票

fuguan
等级:

发表时间：2006-04-21

楼住写的很透入，好文章。但是 hibernate3.0的大批量更新和删除是不更新二级缓存的 这话好像不一定吧,hibernate它有自己的jdbc实现,只要实现是它的api 缓存的问题，hibernate也会管理的吧

文章: 9
积分: 154
我现在离线

返回顶楼 回帖地址 0 0 请登录后投票

AreYouOK?
等级:

发表时间：2006-04-21

fuguan 写道

> 楼住写的很透入，好文章。但是 hibernate3.0的大批量更新和删除是不更新二级缓存的 这话好像不一定吧,hibernate它有自己的jdbc实现,只要实现是它的api 缓存的问题，hibernate也会管理的吧

文章: 46
积分: 390
来自: 北京

我现在离线

http://opensource.atlassian.com/projects/hibernate/browse/HHH-352

不要想当然的认为，这件事情对于hibernate来说其实很为难，因为执行一个批量delete语句并不能知道删除了哪些记录，除非先select一下

返回顶楼　　回帖地址　　0　　0 请登录后投票