作者 正文

goncha
等级:

文章: 104
积分: 477
来自: 上海
我现在离线

发表时间：2004-08-02

< > 猎头职位：上海：上海，北京：招聘java开发工程师

近段时间正好在使用HIBERNATE与数据库打交道. 由于使用环境中读操作占了相当大的比例, 所以想起用HIBERNATE的CACHE功能. 在论坛里搜了一把, 发现了不少关于CACHE的帖子. 但好像都是关于JCS的, 那时似乎还没有QueryCache. 所以就把这两天自己尝试的内容记了下来.

Cache In Hibernate

HIBERNATE中的CACHE有两级. 一级是在Session范围内的CACHE. 即每个Session有自己的一个CACHE, 当前操作的对象都会被保留在CACHE中. 但是Session关闭后这个CACHE也就没有. 可见这级CACHE的生命期是很短的. 另一级CACHE是在SessionFactory范围的, 可以被来自同一个SessionFactory的Session共享.
在HIBERNATE的文档中称其为SECOND LEVEL CACHE. 显然后者的优势较明显, 也比较复合当前的使用环境.

相关文章:

- 请教：关于缓存对象的问题
- Hibernate入门之自己写的小例子的总结
- Hibernate二级缓存,日志说明一切????

推荐圈子: JBPM @net
更多相关推荐

还有一个类型的CACHE就是QueryCache. 它的作用就是缓存一个Query以及Query返回对象的Identifier以及对象的类型. 有了QueryCache后就可以高效的使用SECOND LEVEL CACHE.

Second Level Cache

这里使用ehcache来作为HIBERNATE的SECOND LEVEL CACHE. 由于这是HIBERNATE默认的CACHE提供者, 所以无须做什么设置. 两个很简单的POJO, Person & Address, 一对多的关系. HBM如下:

Java代码

```
1.  <hibernate-mapping
2.          package="goncha.hb.bean">
3.          <class name="Person" table="hb.person">
4.                  <!-- enable second level cache -->
5.                  <cache usage="read-write"/>
6.
7.                  <id name="id">
8.                          <generator class="native"/>
9.                  </id>
10.
11.                 <property name="name" not-null="true" unique="true"/>
12.                 <property name="age" type="java.lang.Integer" not-null="true"/>
13.
14.                 <!-- relationship -->
15.                 <set name="addresses" lazy="true" inverse="true" cascade="save-update">
16.                         <key column="owner_id"/>
17.                         <one-to-many class="Address"/>
18.                 </set>
19.         </class>
20. </hibernate-mapping>
21.
22.
23. <hibernate-mapping
24.         package="goncha.hb.bean">
```

```
        <class name="Address" table="hb.address">
                <!-- enable second level cache -->
                <cache usage="read-write"/>

                <id name="id">
                        <generator class="native"/>
                </id>

                <property name="location" not-null="true" unique="true"/>
                <property name="phone" not-null="true" unique="true"/>

                <many-to-one name="owner" column="owner_id" not-null="true"/>
        </class>
</hibernate-mapping>
```

这两个类都使用了SECOND LEVEL CACHE, 并且两者之间存在关系映射.

ehcache的配置文件ehcache.xml:

Java代码

```
<ehcache>

    <diskStore path="java.io.tmpdir"/>



    <defaultCache

        maxElementsInMemory="10000"

        eternal="false"

        timeToIdleSeconds="120"

        timeToLiveSeconds="120"

        overflowToDisk="true"

        />



    <cache name="goncha.hb.bean.Person"

        maxElementsInMemory="10"

        eternal="false"

        timeToIdleSeconds="100"

        timeToLiveSeconds="100"

        overflowToDisk="false"

        />



    <cache name="goncha.hb.bean.Address"

        maxElementsInMemory="10"

        eternal="false"

        timeToIdleSeconds="100"

        timeToLiveSeconds="100"
```

```
48.
49.         overflowToDisk="false"
50.
51.         />
52.
53.
54.
55. </ehcache>
```

一段简单的测试代码, 可以清楚的看到HIBERNATE如何来使用SECOND LEVEL CACHE. 在运行这段代码前需要作一些工作: 在HIBERNATE的配置文件中允许输出生成的SQL语句; 在LOG4J的配置文件中, 处了net.sf.hibernate.cache在DEBUG级别, 其余都在WARN级别, 为了没有干扰

Java代码

```
1. # log4j.properties
2. log4j.logger.net.sf.hibernate=warn
3. log4j.logger.net.sf.hibernate.cache=debug
4.
5.
6. # hibernate.properties
7. hibernate.show_sql true
```

至此就可以运行以下代码

Java代码

```
1. //: Main.java
2.
3. package goncha.hb;
4.
5. import java.util.Iterator;
6. import java.util.List;
7. import java.util.Set;
8.
9. import net.sf.hibernate.Query;
10. import net.sf.hibernate.Session;
11. import net.sf.hibernate.SessionFactory;
12. import net.sf.hibernate.Transaction;
13. import net.sf.hibernate.cfg.Configuration;
14.
15. import goncha.hb.bean.*;
16.
17.
18. public class Main {
19.
20.     public static void main(String[] args); throws Exception {
21.         SessionFactory sessions = buildSessionFactory();;
22.
23.         pushDataIntoCache(sessions);;
24.         popDataFromCache(sessions);;
25.     }
26.
27.     static SessionFactory buildSessionFactory(); throws Exception {
28.         Configuration config = new Configuration();;
29.         config.addClass(Person.class);.addClass(Address.class);;
30.         return config.buildSessionFactory();;
31.     }
32.
33.     static void pushDataIntoCache(SessionFactory sessions); throws Exception {
34.         System.out.println("======= Push some data into cache =======");;
35.
36.         Session sess = sessions.openSession();;
37.         Transaction tx = sess.beginTransaction();;
38.
39.         Person goncha = (Person);sess.load(Person.class, new Integer(1););;;
40.         System.out.println(goncha);;
41.
```

```
        Person chengang = (Person);sess.load(Person.class, new Integer(2));;;
        System.out.println(chengang);;

        goncha.getAddresses();.size();;
        tx.commit();;
        sess.close();;
    }

    static void popDataFromCache(SessionFactory sessions); throws Exception {
        System.out.println("======= Pop some data into cache =======");;

        Session sess = sessions.openSession();;
        Transaction tx = sess.beginTransaction();;

        Person goncha = (Person);sess.load(Person.class, new Integer(1));;;
        System.out.println(goncha);;

        Person chengang = (Person);sess.load(Person.class, new Integer(2));;;
        System.out.println(chengang);;

        Person tommy = (Person);sess.load(Person.class, new Integer(3););;;
        System.out.println(tommy);;

        Person mary = (Person);sess.load(Person.class, new Integer(4););;;
        System.out.println(mary);;


        List persons = sess.find("from Person as person");;

        Set addresses = goncha.getAddresses();;
        Iterator it = addresses.iterator();;
        while(it.hasNext();); {
            System.out.println((Address);it.next(););;
        }

        tx.commit();;
        sess.close();;
    }
}
```

以下是程序的输出

Java代码

```
Buildfile: build.xml

init:

build:

run:
     [java] 23:14:04,878 DEBUG CacheFactory:32 - cache for: goncha.hb.bean.Person usage strategy:
read-write
     [java] 23:14:05,156 DEBUG CacheFactory:32 - cache for: goncha.hb.bean.Address usage strategy:
 read-write
     [java] Initializing c3p0 pool... com.mchange.v2.c3p0.PoolBackedDataSource@17bd6a1 [ connectio
nPoolDataSource -> com.mchange.v2.c3p0.WrapperConnectionPoolDataSource@8b819f [ acquireIncrement -
> 2, autoCommitOnClose -> false, connectionTesterClassName -> com.mchange.v2.c3p0.impl.DefaultConn
ectionTester, factoryClassLocation -> null, forceIgnoreUnresolvedTransactions -> false, idleConnec
tionTestPeriod -> 3000, initialPoolSize -> 2, maxIdleTime -> 5000, maxPoolSize -> 2, maxStatement
s -> 100, minPoolSize -> 2, nestedDataSource -> com.mchange.v2.c3p0.DriverManagerDataSource@147ee0
5 [ description -> null, driverClass -> null, factoryClassLocation -> null, jdbcUrl -> jdbc:postg
resql:test, properties -> {user=pgsql, password=pgsql} ] , propertyCycle -> 300, testConnectionOnC
heckout -> false ] , factoryClassLocation -> null, numHelperThreads -> 3 ]
     [java] 23:14:07,390  INFO UpdateTimestampsCache:35 - starting update timestamps cache at regi
on: net.sf.hibernate.cache.UpdateTimestampsCache
     [java] 23:14:07,396  INFO QueryCache:39 - starting query cache at region: net.sf.hibernate.ca
```

```
che.QueryCache
     [java] __======= Push some data into cache =======__
     [java] 23:14:07,492 DEBUG ReadWriteCache:68 - Cache lookup: 1
     [java] 23:14:07,494 DEBUG ReadWriteCache:84 - Cache miss: 1
     [java] Hibernate: select person0_.id as id0_, person0_.name as name0_, person0_.age as age0_ from hb.person person0_ where person0_.id=?
     [java] 23:14:07,584 DEBUG ReadWriteCache:132 - Caching: 1
     [java] 23:14:07,594 DEBUG ReadWriteCache:143 - Cached: 1
     [java] Person ['goncha'; 23]
     [java] 23:14:07,599 DEBUG ReadWriteCache:68 - Cache lookup: 2
     [java] 23:14:07,601 DEBUG ReadWriteCache:84 - Cache miss: 2
     [java] Hibernate: select person0_.id as id0_, person0_.name as name0_, person0_.age as age0_ from hb.person person0_ where person0_.id=?
     [java] 23:14:07,609 DEBUG ReadWriteCache:132 - Caching: 2
     [java] 23:14:07,611 DEBUG ReadWriteCache:143 - Cached: 2
     [java] Person ['chengang'; 23]
     [java] Hibernate: select addresses0_.id as id__, addresses0_.owner_id as owner_id__, addresses0_.id as id0_, addresses0_.location as location0_, addresses0_.phone as phone0_, addresses0_.owner_id as owner_id0_ from hb.address addresses0_ where addresses0_.owner_id=?
     [java] 23:14:07,624 DEBUG ReadWriteCache:132 - Caching: 1
     [java] 23:14:07,630 DEBUG ReadWriteCache:143 - Cached: 1
     [java] 23:14:07,632 DEBUG ReadWriteCache:132 - Caching: 2
     [java] 23:14:07,633 DEBUG ReadWriteCache:143 - Cached: 2
     [java] 23:14:07,635 DEBUG ReadWriteCache:132 - Caching: 6
     [java] 23:14:07,637 DEBUG ReadWriteCache:143 - Cached: 6
     [java] __======= Pop some data into cache =======__
     [java] 23:14:07,669 DEBUG ReadWriteCache:68 - Cache lookup: 1
     [java] 23:14:07,671 DEBUG ReadWriteCache:78 - Cache hit: 1
     [java] Person ['goncha'; 23]
     [java] 23:14:07,675 DEBUG ReadWriteCache:68 - Cache lookup: 2
     [java] 23:14:07,678 DEBUG ReadWriteCache:78 - Cache hit: 2
     [java] Person ['chengang'; 23]
     [java] 23:14:07,680 DEBUG ReadWriteCache:68 - Cache lookup: 3
     [java] 23:14:07,682 DEBUG ReadWriteCache:84 - Cache miss: 3
     [java] Hibernate: select person0_.id as id0_, person0_.name as name0_, person0_.age as age0_ from hb.person person0_ where person0_.id=?
     [java] 23:14:07,691 DEBUG ReadWriteCache:132 - Caching: 3
     [java] 23:14:07,693 DEBUG ReadWriteCache:143 - Cached: 3
     [java] Person ['tommy'; 21]
     [java] 23:14:07,698 DEBUG ReadWriteCache:68 - Cache lookup: 4
     [java] 23:14:07,699 DEBUG ReadWriteCache:84 - Cache miss: 4
     [java] Hibernate: select person0_.id as id0_, person0_.name as name0_, person0_.age as age0_ from hb.person person0_ where person0_.id=?
     [java] 23:14:07,704 DEBUG ReadWriteCache:132 - Caching: 4
     [java] 23:14:07,706 DEBUG ReadWriteCache:143 - Cached: 4
     [java] Person ['mary'; 18]
     [java] Hibernate: select person0_.id as id, person0_.name as name, person0_.age as age from hb.person person0_
     [java] 23:14:07,768 DEBUG ReadWriteCache:132 - Caching: 5
     [java] 23:14:07,770 DEBUG ReadWriteCache:143 - Cached: 5
     [java] 23:14:07,772 DEBUG ReadWriteCache:132 - Caching: 6
     [java] 23:14:07,774 DEBUG ReadWriteCache:143 - Cached: 6
     [java] Hibernate: select addresses0_.id as id__, addresses0_.owner_id as owner_id__, addresses0_.id as id0_, addresses0_.location as location0_, addresses0_.phone as phone0_, addresses0_.owner_id as owner_id0_ from hb.address addresses0_ where addresses0_.owner_id=?
     [java] 23:14:07,809 DEBUG ReadWriteCache:132 - Caching: 1
     [java] 23:14:07,811 DEBUG ReadWriteCache:152 - Item was already cached: 1
     [java] 23:14:07,816 DEBUG ReadWriteCache:132 - Caching: 2
     [java] 23:14:07,817 DEBUG ReadWriteCache:152 - Item was already cached: 2
     [java] 23:14:07,819 DEBUG ReadWriteCache:132 - Caching: 6
     [java] 23:14:07,820 DEBUG ReadWriteCache:152 - Item was already cached: 6
     [java] Address ['goncha'; 'newyork']
     [java] Address ['goncha'; 'shanghai']
     [java] Address ['goncha'; 'guangzhou']


BUILD SUCCESSFUL
```

```
71.  Total time: 7 seconds
```

这个结果很出乎意外, HERBNATE只能在Session.load()方法中使用CACHE. pushDataIntoCache()方法成功地给CACHE注入两个Person对象("goncha", "chengang")以及三个Address对象(与"goncha"关联). 再看看popDataFromCache()方法. 使用Session.load()时, "goncha"与"chengang"对应的Person对象都是从CACHE中获得, 其余两个是CACHE MISS的, 正常举动. 而Session.find()的数据获取与CACHE没有关系, 完全由JDBC来操办. 最后只是把从JDBC获得的对象注入CACHE. 通过关系查找对象的过程和Session.find()没有异样, 甚至CACHE还报出对象已存在的消息.

不知这样的测试是否会偏面. 但是在具体的使用环境中就是Session.find()和Relationship Collection占了多数. 像上面的例子那样, CACHE不仅没有用到还白白占了内存. 所以在HIBERNATE中使用CACHE还是需要根据具体情况来定制.

Query Cache

老实说, 要做到在JDBC查询之前决定哪些数据需要从JDBC来还是CACHE来不是件容易事. 但是HIBERNATE还是很好地完成了这个任务. 前面说过QueryCache用来缓存查询语句, 及查询结果集中对象的Identifier与Type. 当再次使用已缓存的Query时, 就可以通过对象的Identifier与Type在SECOND LEVEL CACHE中查找实际的对象.

使用QueryCache时需要在hibernate配置文件中设置如下属性:

Java代码

```
1.  hibernate.cache.use_query_cache true
```

在程序中需要为Query对象设置Cachable属性:

Java代码

```
1.  Query query = sess.createQuery("from AccountingPeriod as period "
2.          + "where period.id.dealerId = ? order by period.id.orderType asc, "
3.          + "period.id.accountingPeriod asc");;
4.  query.setCacheable(true);;
5.  query.setInteger(0, dealerId.intValue());;;
6.  List rs = query.list();;
```

对于查询结果的CACHE处理算是解决了. 但是, 通过Relationship获得Collection的方式好似还不能利用CACHE来提高性能. 有时间再仔细研究一下文档.

返回顶楼

luosheng

等级:

发表时间：2004-08-02

文章: 4

积分: 252

我现在离线

<class name="eg.Cat" .... >
    <jcs-cache usage="read-write"/>
    ....
    <set name="kittens" ... >
        <jcs-cache usage="read-write"/>
        ....
    </set>
</class>

这样才可利用collection cache.

返回顶楼

回帖地址

0 0 请登录后投票

goncha

等级:

发表时间：2004-08-02

刚才看了一下文档和Mapping的DTD定义，在<set></set>，<map></map>等集合定义中也可以使用<cache>．文档中声明了JCS在将来的版本中将不被支持了．

引用

> JCS support is now deprecated and will be removed in a future version of Hibernate.

文章: 104
积分: 477
来自: 上海

返回顶楼　　回帖地址　　0　　0 请登录后投票

firebody

发表时间：2004-08-08

等级: 

楼主的好文章。感谢你的知识!
其实说到cache还是有一些值得注意的:
cache必须注意保持同步。
在delete集合元素的时候，更应该值得注意cache是否已经与数据库同步，否则在读取集合的时候会发生异常。
关于同步cache在必要的时候，可以采用手工同步的办法

文章: 996
积分: 3215

返回顶楼　　回帖地址　　0　　0 请登录后投票

goncha

发表时间：2004-08-10

等级: 

的确是个问题, 还没有尝试过.

文章: 104
积分: 477
来自: 上海

返回顶楼　　回帖地址　　0　　0 请登录后投票

b051

发表时间：2004-08-19

等级: 

楼主的这篇文章我这几天看了好多遍，也在网上搜了很久，
有若干问题，不知道怎么理解，或者怎么解决。
1。我的ehcache的debug消息怎么和楼主的不一样？
2。MISS是不是“cache中没有，现在加入”的意思？
3。query.setCacheable(true)，如果是在一个经常调用&需要ps的地方，是不是在一个static的field里get这个query就可以了？怎么看某个query在不在cache中？
4。ehcache还有hibernate的文档中说管理二级缓存应该在sessionFactory中使用evict，我使用了，但是好像没有生效，它仍然看那个Entity是不是expired?false。
5。怎么写这个部分的测试代码？debug我实在不想看了，首先是我不确定他的每句话都是在干什么，其次也忒多了，不过如果很久还没有人回复，那我自己也能写个简单点的来啃。
6。idle值有必要设的比live短吗？
7。如果不设定cache某个collection，是不是cache中就肯定不会出现collection？（我想知道debug里没有写清的是session cache还是二级）

文章: 153
积分: 595
来自: 上海

（以上无涉及session级cache）

返回顶楼 回帖地址 0 0 请登录后投票

b051
等级:

文章: 153
积分: 595
来自: 上海
我现在离线

发表时间：2004-08-19

还有一些是不是杞人忧天的问题，
其实我想更多的知道他中间的过程（如果有人知道了，就告知，hibernate源代码的cache那部分没有看太懂）
read-write，如果写成了read-write，那么session1正在执行write操作，还没有结束动作，同时sessionFactory被执行了evict，那么session1之后的动作是些什么？（我知道肯定会出错了，那么该如何解决）
如果在mapping中改成read-only，那么前边的问题没有了，可是我通过什么来执行写操作呢？
或者我把问题改一下，cache中的内容，是不是只是由ehcache的那两个小参数——idle和live——维护的。在某个具体过程中，我有没有暂时不选择cache中内容的权力。

是不是我把事情想复杂了？

返回顶楼 回帖地址 0 0 请登录后投票

goncha
等级:

文章: 104
积分: 477
来自: 上海
我现在离线

发表时间：2004-08-19

没有搞复杂. 这是在具体应用中一定会遇到的情况. 由于前一段时间使用Hibernate大都是处理只读的操作所以没有遇到类似的问题. 有时间了好好试一下b051提到的情况.
至于暂时不选择cache中内容, 好像没有注意到有相关的API.

返回顶楼 回帖地址 0 0 请登录后投票

buaawhl
等级:

文章: 3370
积分: 5204
来自: china
我现在离线

发表时间：2004-12-17

> **goncha** 写道
>
> 老实说, 要做到在JDBC查询之前决定哪些数据需要从JDBC来还是CACHE来不是件容易事. 但是HIBERNATE还是很好地完成了这个任务. 前面说过QueryCache用来缓存查询语句, 及查询结果集中对象的Identifier与Type. 当再次使用已缓存的Query时, 就可以通过对象的Identifier与Type在SECOND LEVEL CACHE中查找实际的对象.

在hibernate中，
query cache 的 Key 是 QueryKey。(包括hql, parameters, start, maxRows.)
value是 list of DB object。

session factory level cache 的 key是 id.
value 是 DB object.

请问这两个cache之间的对象能够同步吗？

比如，query cache里面，
"from A where ..." 这个query 对应的
object list是 {a (id = 1) , b (id = 2), c (id = 3)}

session factory level cache 的 里面，

有 key = 1，value = a {id = 1}.

请问，这两个 a 是同一个object吗？
如果不是，两者之间会同步保持相同数据吗？

hibernate的相关源代码为
Loader类，QueryKey关键字

Java代码

```
final boolean cacheable = factory.isQueryCacheEnabled(); && queryParameters.isCacheable();;

if (cacheable); {
    QueryCache queryCache = factory.getQueryCache(queryParameters.getCacheRegion(); );;
    QueryKey key = new QueryKey( getSQLString();, queryParameters );;
    List result = null;
    if ( !queryParameters.isForceCacheRefresh(); ); {
    result = queryCache.get(key, resultTypes, querySpaces, session);;
  }
  if (result==null); {
    result = doList(session, queryParameters);;
    queryCache.put(key, resultTypes, result, session);;
  }
  return getResultList(result);;
}
else {
  return getResultList( doList(session, queryParameters); );;
}
```

StandardQueryCache类。get()方法。

Java代码

```
List cacheable = (List); queryCache.get(key);;
if (cacheable==null); {return null;}

List result = new ArrayList( cacheable.size();-1 );;
Long timestamp = (Long); cacheable.get(0);;
if ( ! isUpToDate(spaces, timestamp); ); {
  log.debug("cached query results were not up to date");;
  return null;
}

for ( int i=1; i<cacheable.size();; i++ ); {
  if ( returnTypes.length==1 ); {
    result.add( returnTypes[0].assemble( (Serializable); cacheable.get(i);, session, null ); );;
  }
  else {
    result.add( TypeFactory.assemble( (Serializable[]); cacheable.get(i);, returnTypes, session, nu
ll ); );;
  }
}
return result;
```

如果有同步，是在returnTypes[0].assemble() 或者 TypeFactory.assemble 里面做的吗？

StandardQueryCache类。put()方法。

Java代码

```
public void put(QueryKey key, Type[] returnTypes, List result, SessionImplementor session); throws
 HibernateException {
if ( log.isDebugEnabled(); ); log.debug("caching query results in region: " + regionName);;
List cacheable = new ArrayList( result.size();+1 );;
cacheable.add( new Long( session.getTimestamp(); ); );;
for ( int i=0; i<result.size();; i++ ); {
if ( returnTypes.length==1 ); {
        cacheable.add( returnTypes[0].disassemble( result.get(i);, session ); );;
```

```
}
else {
        cacheable.add( TypeFactory.disassemble( (Object[]); result.get(i);, returnTypes, session );
 );;
}
}
queryCache.put(key, cacheable);;
}
```

或者是在put的时候，type.dissemble做的？

再来看，ObjectType的assemble和dissemble代码。

Java代码

```
public Object assemble(
Serializable cached,
SessionImplementor session,
Object owner);
throws HibernateException {

ObjectTypeCacheEntry e = (ObjectTypeCacheEntry); cached;
return (cached==null); ? null : session.load(e.clazz, e.id);;
}

/**
* @see net.sf.hibernate.type.Type#disassemble(Object, SessionImplementor);
*/
public Serializable disassemble(Object value, SessionImplementor session);
throws HibernateException {
return (value==null); ?
null :
new ObjectTypeCacheEntry(
        HibernateProxyHelper.getClass(value);,
        session.getEntityIdentifierIfNotUnsaved(value);
);;
}
```

从代码看不出来什么。
有时间，我看看能不能把hibernate source编译，跟踪一下，到底是怎么回事。如果有人知道这方面，愿意回答，就非常感谢了。

---

goncha 写道

> 这个结果很出乎意外, HERBNATE只能在Session.load()方法中使用CACHE. pushDataIntoCache()方法成功地给CACHE注入两个Person对象("goncha", "chengang")以及三个Address对象(与"goncha"关联). 再看看popDataFromCache()方法. 使用Session.load()时, "goncha"与"chengang"对应的Person对象都是从CACHE中获得, 其余两个是CACHE MISS的, 正常举动. 而Session.find()的数据获取与CACHE没有关系, 完全由JDBC来操办. 最后只是把从JDBC获得的对象注入CACHE. 通过关系查找对象的过程和Session.find()没有异样, 甚至CACHE还报出对象已存在的消息.
>
> 不知这样的测试是否会偏面. 但是在具体的使用环境中就是Session.find()和Relationship Collection占了多数. 像上面的例子那样, CACHE不仅没有用到还白白占了内存. 所以在HIBERNATE中使用CACHE还是需要根据具体情况来定制.

对，是这样的。对应代码为：
SessionImpl.java.

Java代码

```
private void endLoadingCollections(CollectionPersister persister, List resultSetCollections);{

//now finish them
for ( int i=0; i<count; i++ ); {
...
```

```
6.  if ( noQueuedAdds && persister.hasCache(); && !ce.doremove ); {
7.  persister.getCache();.put(
8.  lce.id, lce.collection.disassemble(persister);, getTimestamp();, version, versionComparator
9.  );;
10. }
11. }
```

这里只是把list的object按照id放入到缓存里面。
这也许就是应该 使用query cache的场合吧？

返回顶楼　　回帖地址　　0　　0 请登录后投票

b051
等级:

文章: 153
积分: 595
来自: 上海

我现在离线

发表时间：2004-12-18

刚写了一个测试：

Java代码

```
1.  log.info("Start");;
2.        Session session = Hibernator.currentSession();;
3.        String name = "test.admin";
4.        Users admin = Users.getUser(name);;
5.        Query query = session.createQuery("select user.name "
6.                + "from Users as user where user.username=:name");;
7.        query.setCacheable(true);;
8.        query.setParameter("name", name);;
9.        log.info(query.list(););;
10.       admin.setName("1");;
11.       log.info(query.list(););;
```

输出是：

Java代码

```
1.  [12-18 17:05:05]Start
2.  {此处省略97行,内容为将诸相关entity加载入二级缓存}
3.  [12-18 17:05:05]key: 297e6d8400e457e70100e457ee570003
4.  [12-18 17:05:05]bean.UsersCache: MemoryStore miss for 297e6d8400e457e70100e457ee570003
5.  [12-18 17:05:05]bean.Users cache - Miss
6.  [12-18 17:05:05]Element for 297e6d8400e457e70100e457ee570003 is null
7.  [12-18 17:05:05]297e6d8400e4578a0100e457924d0007 now: 1103360705906
8.  [12-18 17:05:05]297e6d8400e4578a0100e457924d0007 Creation Time: 1103360705906 Next To Last Access
    Time: 0
9.  [12-18 17:05:05]297e6d8400e4578a0100e457924d0007 mostRecentTime: 1103360705906
10. [12-18 17:05:05]297e6d8400e4578a0100e457924d0007 Age to Idle: 300000 Age Idled: 0
11. [12-18 17:05:05]bean.Users: Is element with key 297e6d8400e4578a0100e457924d0007 expired?: false

12. [12-18 17:05:05]key: sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.
    USERNAME=? );; parameters: ; named parameters: {name=test.admin}
13. [12-18 17:05:05]net.sf.hibernate.cache.StandardQueryCacheCache: MemoryStore miss for sql: select u
    sers0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERNAME=? );; parameters: ; named
    parameters: {name=test.admin}
14. [12-18 17:05:05]net.sf.hibernate.cache.StandardQueryCache cache - Miss
15. [12-18 17:05:05]Element for sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (u
    sers0_.USERNAME=? );; parameters: ; named parameters: {name=test.admin} is null
16. [12-18 17:05:05]sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERN
    AME=? );; parameters: ; named parameters: {name=test.admin} now: 1103360705937
17. [12-18 17:05:05]sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERN
    AME=? );; parameters: ; named parameters: {name=test.admin} Creation Time: 1103360705937 Next To L
    ast Access Time: 0
18. [12-18 17:05:05]sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERN
    AME=? );; parameters: ; named parameters: {name=test.admin} mostRecentTime: 1103360705937
19. [12-18 17:05:05]sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERN
    AME=? );; parameters: ; named parameters: {name=test.admin} Age to Idle: 0 Age Idled: 0
20. [12-18 17:05:05]net.sf.hibernate.cache.StandardQueryCache: Is element with key sql: select users0_
    .REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERNAME=? );; parameters: ; named parame
    ters: {name=test.admin} expired?: false
21. [12-18 17:05:05][null]
```

```
[12-18 17:05:05]net.sf.hibernate.cache.UpdateTimestampsCache: Is element with key MDC_USERS expired?: false
[12-18 17:05:05]key: 297e6d8400e457e70100e457ee570003
[12-18 17:05:05]bean.UsersCache: MemoryStore hit for 297e6d8400e457e70100e457ee570003
[12-18 17:05:05]297e6d8400e457e70100e457ee570003 now: 1103360705937
[12-18 17:05:05]297e6d8400e457e70100e457ee570003 Creation Time: 1103360705906 Next To Last Access Time: 0
[12-18 17:05:05]297e6d8400e457e70100e457ee570003 mostRecentTime: 1103360705906
[12-18 17:05:05]297e6d8400e457e70100e457ee570003 Age to Idle: 300000 Age Idled: 31
[12-18 17:05:05]bean.Users: Is element with key 297e6d8400e457e70100e457ee570003 expired?: false

[12-18 17:05:05]key: sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERNAME=? );; parameters: ; named parameters: {name=test.admin}
[12-18 17:05:05]net.sf.hibernate.cache.StandardQueryCacheCache: MemoryStore hit for sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERNAME=? );; parameters: ; named parameters: {name=test.admin}
[12-18 17:05:05]sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERNAME=? );; parameters: ; named parameters: {name=test.admin} now: 1103360705937
[12-18 17:05:05]sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERNAME=? );; parameters: ; named parameters: {name=test.admin} Creation Time: 1103360705937 Next To Last Access Time: 0
[12-18 17:05:05]sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERNAME=? );; parameters: ; named parameters: {name=test.admin} mostRecentTime: 1103360705937
[12-18 17:05:05]sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERNAME=? );; parameters: ; named parameters: {name=test.admin} Age to Idle: 0 Age Idled: 0
[12-18 17:05:05]net.sf.hibernate.cache.StandardQueryCache: Is element with key sql: select users0_.REALNAME as x0_0_ from MDC_USERS users0_ where (users0_.USERNAME=? );; parameters: ; named parameters: {name=test.admin} expired?: false
[12-18 17:05:05]key: MDC_USERS
[12-18 17:05:05]net.sf.hibernate.cache.UpdateTimestampsCacheCache: MemoryStore hit for MDC_USERS
[12-18 17:05:05]net.sf.hibernate.cache.UpdateTimestampsCache: Is element with key MDC_USERS expired?: false
[12-18 17:05:05][1]
```

返回顶楼　　回帖地址　　1　　0 请登录后投票

« 上一页 1 2 下一页 »

论坛首页 → Java编程和Java企业应用版 → Hibernate

跳转论坛: Java编程和Java企业应用