Hibernate缓存机制

# Hibernate缓存

hibernate中用到了缓存的概念，那么什么是缓存呢？这里介绍的缓存并不是指计算机的内存或者CPU的一二级缓存，这里的缓存是指为了降低程序对物理数据源访问的频次，从而提高程序运行性能的一种策略。

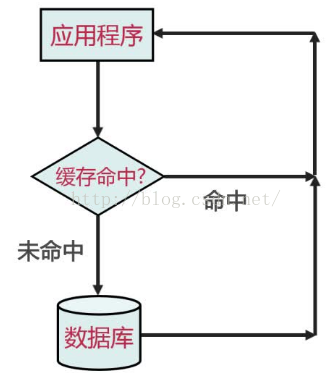
Hibernate要使用缓存的原因：

（1）Hibernate是一个ORM框架，ORM框架访问数据库的效率直接影响应用程序的运行速度，提升和优化ORM框架的执行效率至关重要；

（2）Hibernate的缓存是提升和优化Hibernate执行效率的重要手段，所以学会Hibernate缓存的使用和配置是优化的关键；

（3）评判一个ORM框架是否优秀，访问数据库的频次就一个重要的标准

Hibernate缓存就是将数据库中的数据存储在内存中，缓存的一般工作原理可以通过下图来进行分析



应用程序要查询某一条数据时首先会去缓存中查找，如果在缓存中找到相应的数据，则为缓存命中，将命中的数据直接从缓存中取出返回给应用程序，如果未命中则去数据库查询相应的数据，将数据库中的数据返回给应用程序，同时也会在缓存中保存数据以备下次查询。在Hibernate中用到了两种缓存，分别是一级缓存和二级缓存.

# 一级缓存

一级缓存是Session级共享的。对于一级缓存而言，所有经过Session操作的实体，不管是使用save()、upadate()或者saveOrUpdate保存一个对象，还是使用load()、get()、list()、iterate()、scroll()方法获得一个对象时，该对象都将被放入一级缓存中。

|  |
| --- |
| session = HibernateUtil.getSession();  Users users = (Users) session.get(Users.class, id);  System.out.println(users.getBirthday());  Users users2 = (Users) session.get(Users.class, id);  System.out.println(users.getName());  解析：该程序当第一次查询Users对象时，hibernate会先到一级缓存中查找缓存中是否有该实体，如果有，就直接拿，否则就到数据库中去读取。这里缓存中没有，所以hibernate会到数据库中去读取数据。这里产生一条SQL语句。同时hibernate会将该实体对象放入到一级缓存中去。当第二次还查询该实体对象的时候，hibernate同样会先到一级缓存中去读，结果一级缓存中存在该实体对象，所以直接拿。故上面的程序实例只会产生一条SQL语句。  在Session调用flush()方法或者close()方法之前，这些对象都会一直缓存在一级缓存中。由于一级缓存不能控制缓存的对象数据，所以在大批量操作数据的时候可能会造成内存溢出。清除缓存有两个方法：evil()、clear()。其中evil()用于清除一条记录，它接受一个持久化类参数。Clear用于清除session里面所有的记录。 |

# 二级缓存

## 基本配置使用

SessionFactory级别的二级缓存是全局的，应用的所有Session都共享这个二级缓存。但是二级缓存默认是关闭的，必须由程序显示开启。使用二级缓存一般有如下步骤：

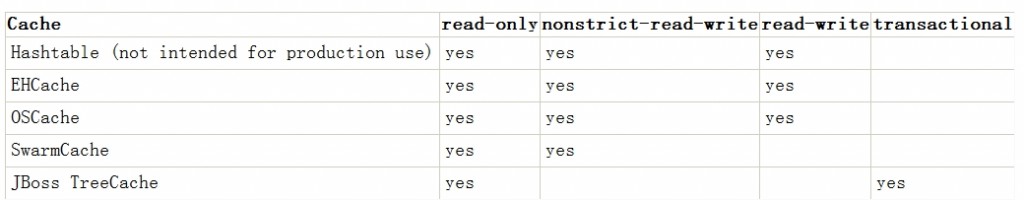
|  |
| --- |
| #1在hibernate.cfg.xml中开启二级缓存  <property name="cache.use\_second\_level\_cache">true</property>  #2设置二级缓存的实现类，直接使用第三方提供的缓存即可。  <property name="cache.provider\_class">org.hibernate.cache.EHCacheProvider</property>    #3复制二级缓存的JAR包。  #4将缓存实现所需要的配置文件添加到系统的类加载路径中。对于EHCache缓存而言，它需要一个ehcache.xml配置文件。配置文件代码如下：  <ehcache>  <defaultCache  maxElementsInMemory="10000"  eternal="false"  timeToIdleSeconds="120"  timeToLiveSeconds="120"  overflowToDisk="true"  />  </ehcache>  #5设置对那些实体类。实体的那些集合属性开启二级缓存。  方法一：修改要使用缓存的映射文件。在持久化映射文件的<class.../>元素、或者<set.../>、<list.../>等集合元素内使用<cache.../>元素指定缓存策略  <hibernate-mapping package="com.hibernate.domain">  <class name="Users">  <cache usage="read-only"/> <!-- 缓存策略为只读 -->  <id name="id">  <generator class="native" />  </id>    <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>  方法二：在hibernate.cfg.xml文件中使用<class-cache.../>或者<collection-cache.../>元素对知道那个的持久化类、集合属性启用二级缓存  <class-cache usage="read-only" class="com.hibernate.domain.Users"/>  #6测试代码  public void query(int id){  Session session = null;  try {  session = HibernateUtil.getSession();  Users users = (Users) session.get(Users.class, id);  System.out.println(users.getBirthday());  } finally{  if(session!=null)  session.close();  }    try {  session = HibernateUtil.getSession();  Users users = (Users) session.get(Users.class, id);  System.out.println(users.getName());  }finally{  if(session!=null)  session.close();  }  } |

## 缓存策略

二级缓存存在如下四种缓存策略：read-only、read-write、nonstrict-read-write、transaction。

* read-only：只读策略。如果应用程序只需要读取持久化实体的对象，无须对其进行修改，那么就可以对其设置为"只读"缓存策略。这是最简单，也是实用性最好的方法。甚至在集群中，它也能完美地运作。
* read-write：读/写策略：如果应用程序需要更新数据，那么使用读/写缓存 比较合适。 如果应用程序要求“序列化事务”的隔离级别（serializable transaction isolation level），那么就决不能使用这种缓存策略。
* nonstrict-read-write：非严格读/写策略。如果应用程序只偶尔需要更新数据（也就是说，两个事务同时更新同一记录的情况很不常见），也不需要十分严格的事务隔离， 那么比较适合使用非严格读/写缓存策略。
* transaction：事物缓存。Hibernate的事务缓存策略提供了全事务的缓存支持。这样的缓存只能用于JTA环境中，你必须指定 为其hibernate.transaction.manager\_lookup\_class属性。

没有一种缓存提供商能够支持上列的所有缓存并发策略。下表中列出了各种提供器、及其各自适用的并发策略。



## 管理缓存

对于二级缓存而言，SessionFactory提供了许多方法用于清除缓存中实例、整个类、集合实例或者整个集合。

|  |
| --- |
| sessionFactory.evict(Users.class, id); //清除指定id的Users对象    sessionFactory.evict(Users.class); // 清除所有的Users对象    sessionFactory.evictCollection("Users.name",id); //清除指定id的Users所关联集合属性    sessionFactory.evictCollection("Users.name"); //清除所有Users所关联集合属性 |

SessionFactory还提供了一个getCache()方法，该方法返回一个Cache对象，通过该对象即可操作二级缓存中的实体、集合等。

* cacheMode参数用于控制具体的Session如何与二级缓存进行交互。
* CacheMode.NORMAL - 从二级缓存中读、写数据。
* CacheMode.GET - 从二级缓存中读取数据，仅在数据更新时对二级缓存写数据。
* CacheMode.PUT - 仅向二级缓存写数据，但不从二级缓存中读数据。
* CacheMode.REFRESH - 仅向二级缓存写数据，但不从二级缓存中读数据。通过 hibernate.cache.use\_minimal\_puts的设置，强制二级缓存从数据库中读取数据，刷新缓存内容。

如果需要查看二级缓存或查询缓存区域的内容，可以使用Hibernate的统计（Statistics） API。为了开启二级缓存的统计功能，在hibernate.cfg.xml文件中进行配置。

|  |
| --- |
| <property name="generate\_statistics">true</property>  <property name="cache.use\_structured\_entries">true</property>  #查看  Map cacheEntries = HibernateUtil.getSessionFactory().getStatistics()  .getSecondLevelCacheStatistics("com.hibernate.domain.Users").getEntries();  System.out.println(cacheEntries); |

# 查询缓存

一级、二级缓存都是对整个实体进行缓存，它不会缓存普通属性，如果想对普通属性进行缓存，则可以使用查询缓存。使用查询缓存时，不仅需要所使用的HQL语句、SQL语句相同，还要求所传入的参数也相同。 要使用查询缓存就需要在hibernate.cfg.xml中开启查询缓存：

|  |
| --- |
| <property name="cache.use\_query\_cache">true</property> |

该设置将会创建两个缓存区域 - 一个用于保存查询结果集； 另一个则用于保存最近查询的一系列表的时间戳。 如若需要进行缓存，请调用 Query.setCacheable(true)方法，该方法用于开启查询缓存。这个调用会让查询在执行过程中时先从缓存中查找结果， 并将自己的结果集放到缓存中去。

如果你要对查询缓存的失效政策进行精确的控制，你必须调用Query.setCacheRegion()方法， 为每个查询指定其命名的缓存区域。

|  |
| --- |
| public void cacheQuery(){  Session session = HibernateUtil.getSession();  List list = session.createQuery("from User u where u.id=:id")  .setInteger("id", 2)  .setCacheable(true)  .setCacheRegion("name")  .list();  } |