Hibernate 缓存机制

1. Cache 简介：

缓存(Cache)是计算机领域非常通用的概念。它介于应用程序和永久性数据存储源(如硬盘上的文件或者数据库)之间，其作用是降低应用程序直接读写永久性数据存储源的频率，从而提高应用的运行性能。缓存中的数据是数据存储源中数据的拷贝，应用程序在运行时直接读写缓存中的数据，只在某些特定时刻按照缓存中的数据来同步更新数据存储源。

缓存的物理介质通常是内存，而永久性数据存储源的物理介质通常是硬盘或磁盘，应用程序读写内在的速度显然比读写硬盘的速度快，如果缓存中存放的数据量非常大，也会用硬盘作为缓存的物理介质。

缓存的实现不仅需要作为物理介质的硬件，同时还需要用于管理缓存的并发访问和过期等策略的软件。因此，缓存是通过软件和硬件共同实现的。

1.1. 持久化层的缓存的范围

缓存的范围决定了缓存的生命周期以及可以被谁访问。缓存的范围分为三类。

1) 事务范围：缓存只能被当前事务访问。缓存的生命周期依赖于事务的生命周期，当事务结束时，缓存也就结束生命周期。在此范围下，缓存的介质是内存。事务可以是数据库事务或者应用事务，每个事务都有独自的缓存，缓存内的数据通常采用相互关联的对象形式。

2) 进程范围：缓存被进程内的所有事务共享。这些事务有可能是并发访问缓存，因此必须对缓存采取必要的事务隔离机制。缓存的生命周期依赖于进程的生命周期，进程结束时，缓存也就结束了生命周期。进程范围的缓存可能会存放大量的数据，所以存放的介质可以是内存或硬盘。缓存内的数据既可以是相互关联的对象形式也可以是对象的松散数据形式。松散的对象数据形式有点类似于对象的序列化数据，但是对象分解为松散的算法比对象序列化的算法要求更快。

3) 集群范围：在集群环境中，缓存被一个机器或者多个机器的进程共享。缓存中的数据被复制到集群环境中的每个进程节点，进程间通过远程通信来保证缓存中的数据的一致性，缓存中的数据通常采用对象的松散数据形式。

对大多数应用来说，应该慎重地考虑是否需要使用集群范围的缓存，因为访问的速度不一定会比直接访问数据库数据的速度快多少。

持久化层可以提供多种范围的缓存。如果在事务范围的缓存中没有查到相应的数据，还可以到进程范围或集群范围的缓存内查询，如果还是没有查到，那么只有到数据库中查询。事务范围的缓存是持久化层的第一级缓存，通常它是必需的；进程范围或集群范围的缓存是持久化层的第二级缓存，通常是可选的。

1.2. 持久化层的缓存的并发访问策略

当多个并发的事务同时访问持久化层的缓存的相同数据时，会引起并发问题，必须采用必要的事务隔离措施。

在进程范围或集群范围的缓存，即第二级缓存，会出现并发问题。因此可以设定以下四种类型的并发访问策略，每一种策略对应一种事务隔离级别。

1) 事务型(Transactional)策略：仅仅在受管理环境中适用。它提供了 Repeatable Read 事务隔离级别。对于经常被读但很少修改的数据，可以采用这种隔离类型，因为它可以防止脏读和不可重复读这类的并发问题。

2) 读写型(read-write)策略：提供了 Read Committed 事务隔离级别。仅仅在非集群的环境中适用。对于经常被读但很少修改的数据，可以采用这种隔离类型，因为它可以防止脏读这类的并发问题。

3) 非严格读写型(nonstrict-read-write)策略：不保证缓存与数据库中数据的一致性。如果存在两个事务同时访问缓存中相同数据的可能，必须为该数据配置一个很短的数据过期时间，从而尽量避免脏读。对于极少被修改，并且允许偶尔脏读的数据，可以采用这种并发访问策略。

4) 只读型策略(read-only)：对于从来不会修改的数据，如参考数据，可以使用这种并发访问策略。

事务型并发访问策略是事务隔离级别最高，只读型的隔离级别最低。事务隔离级别越高，并发性能就越低。

2. Hibernate 中的缓存：

Hibernate 中提供了两级 Cache，第一级别的缓存是 Session 级别的缓存，它是属于事务范围的缓存。这一级别的缓存由 hibernate 管理的，一般情况下无需进行干预；第二级别的缓存是 SessionFactory 级别的缓存，它是属于进程范围或群集范围的缓存。这一级别的缓存可以进行配置和更改，并且可以动态加载和卸载。

Hibernate 还为查询结果提供了一个查询缓存，它依赖于第二级缓存。

2.1. 一级缓存和二级缓存的比较：

第一级缓存 第二级缓存

存放数据的形式 相互关联的持久化对象 对象的散装数据

缓存的范围 事务范围，每个事务都有单独的第一级缓存 进程范围或集群范围，缓存被同一个进程或集群范围内的所有事务共享

并发访问策略 由于每个事务都拥有单独的第一级缓存，不会出现并发问题，无需提供并发访问策略 由于多个事务会同时访问第二级缓存中相同数据，因此必须提供适当的并发访问策略，来保证特定的事务隔离级别

数据过期策略 没有提供数据过期策略。处于一级缓存中的对象永远不会过期，除非应用程序显式清空缓存或者清除特定的对象 必须提供数据过期策略，如基于内存的缓存中的对象的最大数目，允许对象处于缓存中的最长时间，以及允许对象处于缓存中的最长空闲时间

物理存储介质 内存 内存和硬盘。对象的散装数据首先存放在基于内在的缓存中，当内存中对象的数目达到数据过期策略中指定上限时，就会把其余的对象写入基于硬盘的缓存中。

缓存的软件实现 在 Hibernate 的 Session 的实现中包含了缓存的实现 由第三方提供，Hibernate 仅提供了缓存适配器(CacheProvider)。用于把特定的缓存插件集成到 Hibernate 中。

启用缓存的方式 只要应用程序通过 Session 接口来执行保存、更新、删除、加载和查询数据库数据的操作，Hibernate 就会启用第一级缓存，把数据库中的数据以对象的形式拷贝到缓存中，对于批量更新和批量删除操作，如果不希望启用第一级缓存，可以绕过 Hibernate API，直接通过 JDBC API 来执行指操作。 用户可以在单个类或类的单个集合的粒度上配置第二级缓存。如果类的实例被经常读但很少被修改，就可以考虑使用第二级缓存。只有为某个类或集合配置了第二级缓存，Hibernate 在运行时才会把它的实例加入到第二级缓存中。

用户管理缓存的方式 第一级缓存的物理介质为内存，由于内存容量有限，必须通过恰当的检索策略和检索方式来限制加载对象的数目。Session 的 evit()方法可以显式清空缓存中特定对象，但这种方法不值得推荐。 第二级缓存的物理介质可以是内存和硬盘，因此第二级缓存可以存放大量的数据，数据过期策略的 maxElementsInMemory 属性值可以控制内存中的对象数目。管理第二级缓存主要包括两个方面：选择需要使用第二级缓存的持久类，设置合适的并发访问策略：选择缓存适配器，设置合适的数据过期策略。

2.2. 一级缓存的管理：

当应用程序调用 Session 的 save()、update()、savaeOrUpdate()、get()或 load()，以及调用查询接口的 list()、iterate()或 filter()方法时，如果在 Session 缓存中还不存在相应的对象，Hibernate 就会把该对象加入到第一级缓存中。当清理缓存时，Hibernate 会根据缓存中对象的状态变化来同步更新数据库。

Session 为应用程序提供了两个管理缓存的方法：

evict(Object obj)：从缓存中清除参数指定的持久化对象。

clear()：清空缓存中所有持久化对象。

2.3. 二级缓存的管理：

2.3.1. Hibernate 的二级缓存策略的一般过程如下：

1) 条件查询的时候，总是发出一条 select * from table_name where …. （选择所有字段）这样的 SQL 语句查询数据库，一次获得所有的数据对象。

2) 把获得的所有数据对象根据 ID 放入到第二级缓存中。

3) 当 Hibernate 根据 ID 访问数据对象的时候，首先从 Session 一级缓存中查；查不到，如果配置了二级缓存，那么从二级缓存中查；查不到，再查询数据库，把结果按照 ID 放入到缓存。

4) 删除、更新、增加数据的时候，同时更新缓存。

Hibernate 的二级缓存策略，是针对于 ID 查询的缓存策略，对于条件查询则毫无作用。为此，Hibernate 提供了针对条件查询的 Query Cache。

2.3.2. 什么样的数据适合存放到第二级缓存中？

1 很少被修改的数据

2 不是很重要的数据，允许出现偶尔并发的数据

3 不会被并发访问的数据

4 参考数据,指的是供应用参考的常量数据，它的实例数目有限，它的实例会被许多其他类的实例引用，实例极少或者从来不会被修改。

2.3.3. 不适合存放到第二级缓存的数据？

1 经常被修改的数据

2 财务数据，绝对不允许出现并发

3 与其他应用共享的数据。

2.3.4. 常用的缓存插件

Hibernater 的二级缓存是一个插件，下面是几种常用的缓存插件：

l EhCache：可作为进程范围的缓存，存放数据的物理介质可以是内存或硬盘，对 Hibernate 的查询缓存提供了支持。

l OSCache：可作为进程范围的缓存，存放数据的物理介质可以是内存或硬盘，提供了丰富的缓存数据过期策略，对 Hibernate 的查询缓存提供了支持。

l SwarmCache：可作为群集范围内的缓存，但不支持 Hibernate 的查询缓存。

l JBossCache：可作为群集范围内的缓存，支持事务型并发访问策略，对 Hibernate 的查询缓存提供了支持。

2.3.5. 配置二级缓存的主要步骤：

1) 选择需要使用二级缓存的持久化类，设置它的命名缓存的并发访问策略。这是最值得认真考虑的步骤。

2)　选择合适的缓存插件，然后编辑该插件的配置文件。

2.4.　使用 EhCache 配置二级缓存：

2.4.1.　配置准备：

1)　把 ehcache-1.2.3.jar 加入到当前应用的 classpath 中。

2)　在 hibernate.cfg.xml 文件中加入 EhCache 缓存插件的提供类。

< !--配置缓存插件 -->

< property name="hibernate.cache.provider_class">

org.hibernate.cache.EhCacheProvider

< /property>

3)　拷贝 ehcache.xml 文件到类路径(项目工程的 src 目录下)，这个文件在 Hibernate 安装目录的 etc 下。

2.4.2.　配置步骤：

Hibernate 允许在类和集合的粒度上设置第二级缓存。在映射文件中，< class>和< set>元素都有一个< cache>子元素，这个子元素用来配置二级缓存。

示例：以 category(产品类别)和 product(产品)的映射为例：

1)　修改要配置缓存的那个持久化类的对象关系映射文件：

Category.hbm.xml

< ?xml version="1.0" encoding="utf-8"?>

< !DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

< hibernate-mapping>

< class name="org.qiujy.domain.cachedemo.Category" table="categories">

< !?

配置缓存,必须紧跟在 class 元素后面

对缓存中的 Category 对象采用读写型的并发访问策略

-->

< cache usage="read-write"/>

< id name="id" type="java.lang.Long">

< column name="id" />

< generator class="native" />

< /id>

< !-- 配置版本号,必须紧跟在 id 元素后面 -->

< version name="version" column="version" type="java.lang.Long" />

< property name="name" type="java.lang.String">

< column name="name" length="32" not-null="true"/>

< /property>

< property name="description" type="java.lang.String">

< column name="description" length="255"/>

< /property>

< set name="products" table="products" cascade="all" inverse="true">

< !-- Hibernate 只会缓存对象的简单属性的值,

要缓存集合属性,必须在集合元素中也加入< cache>子元素

而 Hibernate 仅仅是把与当前持久对象关联的对象的 OID 存放到缓存中。

如果希望把整个关联的对象的所有数据都存入缓存,

则要在相应关联的对象的映射文件中配置< cache>元素

-->

< cache usage="read-write"/>

< key column="categoryId" not-null="true"/>

< one-to-many class="org.qiujy.domain.cachedemo.Product"/>

```
        < /set>
    < /class>
< /hibernate-mapping>
```

Product.hbm.xml

```
< ?xml version="1.0" encoding="utf-8"?>
< !DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
< hibernate-mapping>
    < class name="org.qiujy.domain.cachedemo.Product" table="products">
        < cache usage="read-write"/>
        < id name="id" type="java.lang.Long">
            < column name="id" />
            < generator class="native" />
        < /id>
        < !--  配置版本号,必须紧跟在 id 元素后面  -->
        < version name="version" column="version" type="java.lang.Long" />
        < property name="name" type="java.lang.String">
            < column name="name" length="32" not-null="true"/>
        < /property>
        < property name="description" type="java.lang.String">
            < column name="description" length="255"/>
        < /property>
        < property name="unitCost" type="java.lang.Double">
            < column name="unitCost" />
```

```
        < /property>
        < property name="pubTime" type="java.util.Date">
            < column name="pubTime" not-null="true" />
        < /property>
        < many-to-one name="category"
                column="categoryId"
               class="org.qiujy.domain.cachedemo.Category"
               cascade="save-update"
                not-null="true">
         < /many-to-one>

    < /class>
< /hibernate-mapping>
```

2)　编辑 ehcache.xml 文件：

```
< ehcache>
    < diskStore path="c:\\ehcache\"/>
    < defaultCache
        maxElementsInMemory="10000"
        eternal="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        overflowToDisk="true"
        />
    < !--  设置 Category 类的缓存的数据过期策略  -->
```

```
    < cache name="org.qiujy.domain.cachedemo.Category"
        maxElementsInMemory="100"
        eternal="true"
        timeToIdleSeconds="0"
        timeToLiveSeconds="0"
        overflowToDisk="false"
        />
     < !--  设置 Category 类的 products 集合的缓存的数据过期策略  -->
     < cache name="org.qiujy.domain.cachedemo.Category.products"
        maxElementsInMemory="500"
        eternal="false"
        timeToIdleSeconds="300"
        timeToLiveSeconds="600"
        overflowToDisk="true"
        />
    < cache name="org.qiujy.domain.cachedemo.Product"
        maxElementsInMemory="500"
        eternal="false"
        timeToIdleSeconds="300"
        timeToLiveSeconds="600"
        overflowToDisk="true"
        />
< /ehcache>
```

配置的元素说明：

元素或属性 描述

< diskStore> 设置缓存数据文件的存放目录

< defaultCache> 设置缓存的默认数据过期策略

< cache> 设定具体的命名缓存的数据过期策略

每个命名缓存代表一个缓存区域，每个缓存区域有各自的数据过期策略。命名缓存机制使得用户能够在每个类以及类的每个集合的粒度上设置数据过期策略。

cache 元素的属性

name 设置缓存的名字,它的取值为类的全限定名或类的集合的名字

maxInMemory 设置基于内存的缓存中可存放的对象最大数目

eternal 设置对象是否为永久的,true 表示永不过期,此时将忽略 timeToIdleSeconds 和 timeToLiveSeconds 属性;

默认值是 false

timeToIdleSeconds 设置对象空闲最长时间,超过这个时间,对象过期。当对象过期时,EHCache 会把它从缓存中清除。

如果此值为 0,表示对象可以无限期地处于空闲状态。

timeToLiveSeconds 设置对象生存最长时间,超过这个时间,对象过期。

如果此值为 0,表示对象可以无限期地存在于缓存中。

overflowToDisk 设置基于内在的缓存中的对象数目达到上限后,是否把溢出的对象写到基于硬盘的缓存中

3) 写一测试类：

```
package org.qiujy.test.cache;

import java.util.List;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.qiujy.common.HibernateSessionFactory;
```

```
import org.qiujy.domain.cachedemo.Product;


public class TestCache {


    public static void main(String[] args) {


        //test cache.....
        Session session2 = HibernateSessionFactory.getSession();
        Transaction tx2 =null;


        try{
            tx2 = session2.beginTransaction();


            List list = session2.createQuery("from Product").list();


            for(int i = 0 ; i <   list.size(); i++){
                Product prod = (Product)list.get(i);
                System.out.println(prod.getName());
            }


           tx2.commit();
        }catch(HibernateException e){
            if(tx2 != null){
                tx2.rollback();
```

```
        }
        e.printStackTrace();
}finally{
        HibernateSessionFactory.closeSession();
}

//--------------------
Session session3 = HibernateSessionFactory.getSession();
Transaction tx3 =null;

try{
        tx3 = session3.beginTransaction();

        Product prod = (Product)session3.get(Product.class, new Long(1));
        System.out.println("从 cache 中得到,不执行 SQL---" + prod.getName());

       tx3.commit();
}catch(HibernateException e){
        if(tx3 != null){
              tx3.rollback();
        }
        e.printStackTrace();
}finally{
        HibernateSessionFactory.closeSession();
```

```
        }
    }
}
```

首先数据库插入 1000 条产品记录和 1 条类别记录。此 1000 个产品都属于这一类别。然后执行以上测试类，在 Session2 中查询所有的产品，输出它的产品名，Session2 会把这些数据加载到二级缓存中，由于有 1000 个对象，而配置中定义内存中只能存放 500 个，剩下的对象就会写到指定的磁盘目录中缓存起来。所以在磁盘相应位置可看到数据文件：

2.5. 查询缓存(Query Cache)：

对于经常使用的查询语句，如果启用了查询缓存，当第一次执行查询语句时，Hibernate 会把查询结果存放在第二缓存中。以后再次执行该查询语句时，只需从缓存中获得查询结果，从而提高查询性能。

2.5.1. 查询缓存适用于以下场合：

l 在应用程序运行时经常使用的查询语句。

l 很少对与查询语句关联的数据库数据进行插入、删除或更新操作。

2.5.2. Hibernate 的 Query 缓存策略的过程如下：

1) Hibernate 首先根据这些信息组成一个 Query Key，Query Key 包括条件查询的请求一般信息：SQL， SQL 需要的参数，记录范围（起始位置 rowStart，最大记录个数 maxRows)，等。

2) Hibernate 根据这个 Query Key 到 Query 缓存中查找对应的结果列表。如果存在，那么返回这个结果列表；如果不存在，查询数据库，获取结果列表，把整个结果列表根据 Query Key 放入到 Query 缓存中。

3) Query Key 中的 SQL 涉及到一些表名，如果这些表的任何数据发生修改、删除、增加等操作，这些相关的 Query Key 都要从缓存中清空。

只有当经常使用同样的参数进行查询时，这才会有些用处。

启用查询缓存的步骤：

1) 配置二级缓存：

Hibernate 提供了三种和查询相关的缓存区域：

l 默认的查询缓存区域：org.hibernate.cache.StandardQueryCache

l 用户自定义的查询缓存区域：

l 时间戳缓存区域：org.hibernate.cache.UpdateTimestampCache

默认的查询缓存区域以及用户自定义的查询缓存区域都用于存放查询结果。而时间戳缓存区域存放了对与查询结果相关的表进行插入、更新或删除操作的时间戳。Hibernate 通过时间戳缓存区域来判断被缓存的查询结果是否过期。所以，当应用程序对数据库的相关数据做了修改，Hibernate 会自动刷新缓存的查询结果。但是如果其他应用程序对数据库的相关数据做了修改，则无法监测，此时必须由应用程序负责监测这一变化，然后手工刷新查询结果。Query 接口的 setForceCacheRefresh(true)可以手工刷新查询结果。

在 ehcache.xml 中添加如下配置：

```
< !--  设置默认的查询缓存的数据过期策略  -->
    < cache name="org.hibernate.cache.StandardQueryCache"
      maxElementsInMemory="50"
      eternal="false"
      timeToIdleSeconds="3600"
      timeToLiveSeconds="7200"
      overflowToDisk="true"/>

    < !--  设置时间戳缓存的数据过期策略  -->
    < cache name="org.hibernate.cache.UpdateTimestampsCache"
      maxElementsInMemory="5000"
      eternal="true"
      overflowToDisk="true"/>

    < !--  设置自定义命名查询缓存 customerQueries 的数据过期策略  -->
    < cache name="myCacheRegion"
       maxElementsInMemory="1000"
       eternal="false"
```

timeToIdleSeconds="300"

timeToLiveSeconds="600"

overflowToDisk="true"

/>

2) 打开查询缓存：在 hibernate.cfg.xml 添加如下配置

< !--启用查询缓存 -->

< property name="cache.use_query_cache">true< /property>

3) 在程序中使用：

虽然按以上设置好了查询缓存，但 Hibernate 在执行查询语句语句时仍不会启用查询缓存。对于希望启用查询缓存的查询语句，应该调用 Query 接口的 setCacheeable(true)方法：

测试类如下：

```
package org.qiujy.test.cache;


import java.util.List;


import org.hibernate.HibernateException;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.qiujy.common.HibernateSessionFactory;

import org.qiujy.domain.cachedemo.Product;


public class TessQueryCache {
```

```
public static void main(String[] args) {
    Session session = HibernateSessionFactory.getSession();
    Transaction tx =null;

    try{
        tx = session.beginTransaction();
        Query query = session.createQuery("from Product");
        //激活查询缓存
        query.setCacheable(true);
        //使用自定义的查询缓存区域,若不设置,则使用标准查询缓存区域
        query.setCacheRegion("myCacheRegion");

        List list = query.list();
        for(int i = 0 ; i <   list.size(); i++){
            Product prod = (Product)list.get(i);
            System.out.println(prod.getName());
        }

       tx.commit();
    }catch(HibernateException e){
        if(tx != null){
            tx.rollback();
        }
        e.printStackTrace();
```

```
        }finally{
            HibernateSessionFactory.closeSession();
        }
    }
}
```