**查询缓存**

# mybatis缓存介绍

如下图，是mybatis一级缓存和二级缓存的区别图解：

sqlSession1

sqlSession 2

sqlSession3...

一级缓存

一级缓存

一级缓存

Mapper(namespace)二级缓存

Mybatis一级缓存的作用域是同一个SqlSession，在同一个sqlSession中两次执行相同的sql语句，第一次执行完毕会将数据库中查询的数据写到缓存（内存），第二次会从缓存中获取数据将不再从数据库查询，从而提高查询效率。当一个sqlSession结束后该sqlSession中的一级缓存也就不存在了。Mybatis默认开启一级缓存。

Mybatis二级缓存是多个SqlSession共享的，其作用域是mapper的同一个namespace，不同的sqlSession两次执行相同namespace下的sql语句且向sql中传递参数也相同即最终执行相同的sql语句，第一次执行完毕会将数据库中查询的数据写到缓存（内存），第二次会从缓存中获取数据将不再从数据库查询，从而提高查询效率。Mybatis默认没有开启二级缓存需要在setting全局参数中配置开启二级缓存。

# 一级缓存

## 原理

下图是根据id查询用户的一级缓存图解：

第一次查询id为1的用户

SqlSession

一级缓存区域

Map

SqlSession

写入

修改、添加、删除用户执行commit

清空

第二次查询id为1的用户

读取

一级缓存区域是根据SqlSession为单位划分的。

每次查询会先从缓存区域找，如果找不到从数据库查询，查询到数据将数据写入缓存。

Mybatis内部存储缓存使用一个HashMap，key为hashCode+sqlId+Sql语句。value为从查询出来映射生成的java对象

sqlSession执行insert、update、delete等操作commit提交后会清空缓存区域。

## 测试1

//获取session

SqlSession session = sqlSessionFactory.openSession();

//获限mapper接口实例

UserMapper userMapper = session.getMapper(UserMapper.**class**);

//第一次查询

User user1 = userMapper.findUserById(1);

System.*out*.println(user1);

//第二次查询，由于是同一个session则不再向数据发出语句直接从缓存取出

User user2 = userMapper.findUserById(1);

System.*out*.println(user2);

//关闭session

session.close();

## 测试2

//获取session

SqlSession session = sqlSessionFactory.openSession();

//获限mapper接口实例

UserMapper userMapper = session.getMapper(UserMapper.**class**);

//第一次查询

User user1 = userMapper.findUserById(1);

System.*out*.println(user1);

//在同一个session执行更新

User user\_update = **new User()**;

user\_update.setId(1);

user\_update.setUsername("李奎");

userMapper.updateUser(user\_update);

session.commit();

//第二次查询，虽然是同一个session但是由于执行了更新操作session的缓存被清空，这里重新发出sql操作

User user2 = userMapper.findUserById(1);

System.*out*.println(user2);

# 二级缓存

## 原理

下图是多个sqlSession请求UserMapper的二级缓存图解。

UserMapper

sqlSession1

执行UserMapper查询

sqlSession2

执行UserMapper查询

sqlSession3..

执行UserMapper提交（插入、更新、删除）

UserMapper的二级缓存区域

写入

读取

清空

二级缓存区域是根据mapper的namespace划分的，相同namespace的mapper查询数据放在同一个区域，如果使用mapper代理方法每个mapper的namespace都不同，此时可以理解为二级缓存区域是根据mapper划分。

每次查询会先从缓存区域找，如果找不到从数据库查询，查询到数据将数据写入缓存。

Mybatis内部存储缓存使用一个HashMap，key为hashCode+sqlId+Sql语句。value为从查询出来映射生成的java对象

sqlSession执行insert、update、delete等操作commit提交后会清空缓存区域。

## 开启二级缓存：

在核心配置文件SqlMapConfig.xml中加入

<setting name=*"cacheEnabled"* value=*"true"*/>

|  |  |  |  |
| --- | --- | --- | --- |
|  | 描述 | 允许值 | 默认值 |
| cacheEnabled | 对在此配置文件下的所有cache 进行全局性开/关设置。 | true false | true |

要在你的Mapper映射文件中添加一行：  <cache /> ，表示此mapper开启二级缓存。

## 实现序列化

二级缓存需要查询结果映射的pojo对象实现java.io.Serializable接口实现序列化和反序列化操作，注意如果存在父类、成员pojo都需要实现序列化接口。

public class Orders implements Serializable

public class User implements Serializable

....

## 测试

//获取session1

SqlSession session1 = sqlSessionFactory.openSession();

UserMapper userMapper = session1.getMapper(UserMapper.**class**);

//使用session1执行第一次查询

User user1 = userMapper.findUserById(1);

System.*out*.println(user1);

//关闭session1

session1.close();

//获取session2

SqlSession session2 = sqlSessionFactory.openSession();

UserMapper userMapper2 = session2.getMapper(UserMapper.**class**);

//使用session2执行第二次查询，由于开启了二级缓存这里从缓存中获取数据不再向数据库发出sql

User user2 = userMapper2.findUserById(1);

System.*out*.println(user2);

//关闭session2

session2.close();

## 禁用二级缓存

在statement中设置useCache=false可以禁用当前select语句的二级缓存，即每次查询都会发出sql去查询，默认情况是true，即该sql使用二级缓存。

<select id="findOrderListResultMap" resultMap="ordersUserMap" useCache="false">

## 刷新缓存

在mapper的同一个namespace中，如果有其它insert、update、delete操作数据后需要刷新缓存，如果不执行刷新缓存会出现脏读。

 设置statement配置中的flushCache="true" 属性，默认情况下为true即刷新缓存，如果改成false则不会刷新。使用缓存时如果手动修改数据库表中的查询数据会出现脏读。

如下：

<insert id="insertUser" parameterType="cn.itcast.mybatis.po.User" flushCache="true">

## Mybatis Cache参数

flushInterval（刷新间隔）可以被设置为任意的正整数，而且它们代表一个合理的毫秒形式的时间段。默认情况是不设置，也就是没有刷新间隔，缓存仅仅调用语句时刷新。

size（引用数目）可以被设置为任意正整数，要记住你缓存的对象数目和你运行环境的可用内存资源数目。默认值是1024。

readOnly（只读）属性可以被设置为true或false。只读的缓存会给所有调用者返回缓存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。可读写的缓存会返回缓存对象的拷贝（通过序列化）。这会慢一些，但是安全，因此默认是false。

如下例子：

<cache  eviction="FIFO"  flushInterval="60000"  size="512"  readOnly="true"/>

这个更高级的配置创建了一个 FIFO 缓存,并每隔 60 秒刷新,存数结果对象或列表的 512 个引用,而且返回的对象被认为是只读的,因此在不同线程中的调用者之间修改它们会导致冲突。可用的收回策略有, 默认的是 LRU:

1. LRU – 最近最少使用的:移除最长时间不被使用的对象。
2. FIFO – 先进先出:按对象进入缓存的顺序来移除它们。
3. SOFT – 软引用:移除基于垃圾回收器状态和软引用规则的对象。
4. WEAK – 弱引用:更积极地移除基于垃圾收集器状态和弱引用规则的对象。

## mybatis整合ehcache

EhCache 是一个纯Java的进程内缓存框架，是一种广泛使用的开源Java分布式缓存，具有快速、精干等特点，是Hibernate中默认的CacheProvider。

### mybatis整合ehcache原理

mybatis提供二级缓存Cache接口，如下：



它的默认实现类：



通过实现Cache接口可以实现mybatis缓存数据通过其它缓存数据库整合，mybatis的特长是sql操作，缓存数据的管理不是mybatis的特长，为了提高缓存的性能将mybatis和第三方的缓存数据库整合，比如ehcache、memcache、redis等。

### 第一步：引入缓存的依赖包

JOZ58Q@U{X_}F`EN}BLVU1A

maven坐标：

<dependency>

<groupId>org.mybatis.caches</groupId>

<artifactId>mybatis-ehcache</artifactId>

<version>1.0.2</version>

</dependency>

### 第二步：引入缓存配置文件

classpath下添加：ehcache.xml

内容如下：

<ehcache xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:noNamespaceSchemaLocation=*"../config/ehcache.xsd"*>

<diskStore path=*"F:\develop\ehcache"* />

<defaultCache

maxElementsInMemory=*"1000"*

maxElementsOnDisk=*"10000000"*

eternal=*"false"*

overflowToDisk=*"false"*

timeToIdleSeconds=*"120"*

timeToLiveSeconds=*"120"*

diskExpiryThreadIntervalSeconds=*"120"*

memoryStoreEvictionPolicy=*"LRU"*>

</defaultCache>

</ehcache>

属性说明：

diskStore：指定数据在磁盘中的存储位置。

defaultCache：当借助CacheManager.add("demoCache")创建Cache时，EhCache便会采用<defalutCache/>指定的的管理策略

以下属性是必须的：

maxElementsInMemory - 在内存中缓存的element的最大数目

maxElementsOnDisk - 在磁盘上缓存的element的最大数目，若是0表示无穷大

eternal - 设定缓存的elements是否永远不过期。如果为true，则缓存的数据始终有效，如果为false那么还要根据timeToIdleSeconds，timeToLiveSeconds判断

overflowToDisk - 设定当内存缓存溢出的时候是否将过期的element缓存到磁盘上

以下属性是可选的：

timeToIdleSeconds - 当缓存在EhCache中的数据前后两次访问的时间超过timeToIdleSeconds的属性取值时，这些数据便会删除，默认值是0,也就是可闲置时间无穷大

timeToLiveSeconds - 缓存element的有效生命期，默认是0.,也就是element存活时间无穷大

diskSpoolBufferSizeMB 这个参数设置DiskStore(磁盘缓存)的缓存区大小.默认是30MB.每个Cache都应该有自己的一个缓冲区.

diskPersistent - 在VM重启的时候是否启用磁盘保存EhCache中的数据，默认是false。

diskExpiryThreadIntervalSeconds - 磁盘缓存的清理线程运行间隔，默认是120秒。每个120s，相应的线程会进行一次EhCache中数据的清理工作

memoryStoreEvictionPolicy - 当内存缓存达到最大，有新的element加入的时候， 移除缓存中element的策略。默认是LRU（最近最少使用），可选的有LFU（最不常使用）和FIFO（先进先出）

### 第三步：开启ehcache缓存

EhcacheCache是ehcache对Cache接口的实现：



修改mapper.xml文件，在cache中指定EhcacheCache。

<cache type=*"org.mybatis.caches.ehcache.**EhcacheCache"*/>

根据需求调整缓存参数：

<cache type=*"org.mybatis.caches.ehcache.EhcacheCache"* >

<property name=*"timeToIdleSeconds"* value=*"3600"*/>

<property name=*"timeToLiveSeconds"* value=*"3600"*/>

<!-- 同ehcache参数maxElementsInMemory -->

<property name=*"maxEntriesLocalHeap"* value=*"1000"*/>

<!-- 同ehcache参数maxElementsOnDisk -->

<property name=*"maxEntriesLocalDisk"* value=*"10000000"*/>

<property name=*"memoryStoreEvictionPolicy"* value=*"LRU"*/>

</cache>

## 应用场景

对于访问多的查询请求且用户对查询结果实时性要求不高，此时可采用mybatis二级缓存技术降低数据库访问量，提高访问速度，业务场景比如：耗时较高的统计分析sql、电话账单查询sql等。

实现方法如下：通过设置刷新间隔时间，由mybatis每隔一段时间自动清空缓存，根据数据变化频率设置缓存刷新间隔flushInterval，比如设置为30分钟、60分钟、24小时等，根据需求而定。

## 局限性

mybatis二级缓存对细粒度的数据级别的缓存实现不好，比如如下需求：对商品信息进行缓存，由于商品信息查询访问量大，但是要求用户每次都能查询最新的商品信息，此时如果使用mybatis的二级缓存就无法实现当一个商品变化时只刷新该商品的缓存信息而不刷新其它商品的信息，因为mybaits的二级缓存区域以mapper为单位划分，当一个商品信息变化会将所有商品信息的缓存数据全部清空。解决此类问题需要在业务层根据需求对数据有针对性缓存。