06-MyBatis基于XML的详细使用——缓存

1、介绍

MyBatis 内置了一个强大的事务性查询缓存机制，它可以非常方便地配置和定制。 为了使它更加强大而且易于配置，我们对 MyBatis 3 中的缓存实现进行了许多改进。

默认情况下，只启用了本地的会话缓存，它仅仅对一个会话中的数据进行缓存。 要启用全局的二级缓存，只需要在你的 SQL 映射文件中添加一行：

```
<cache/>
```

当添加上该标签之后，会有如下效果：

- 映射语句文件中的所有 select 语句的结果将会被缓存。
- 映射语句文件中的所有 insert、update 和 delete 语句会刷新缓存。
- 缓存会使用最近最少使用算法（LRU, Least Recently Used）算法来清除不需要的缓存。
- 缓存不会定时进行刷新（也就是说，没有刷新间隔）。
- 缓存会保存列表或对象（无论查询方法返回哪种）的 1024 个引用。
- 缓存会被视为读/写缓存，这意味着获取到的对象并不是共享的，可以安全地被调用者修改，而不干扰其他调用者或线程所做的潜在修改。

在进行配置的时候还会分为一级缓存和二级缓存：

一级缓存：线程级别的缓存，是本地缓存，sqlSession级别的缓存

二级缓存：全局范围的缓存，不止局限于当前会话

2、一级缓存的使用

一级缓存是sqlsession级别的缓存，默认是存在的。在下面的案例中，大家发现我发送了两个相同的请求，但是sql语句仅仅执行了一次，那么就意味着第一次查询的时候已经将结果进行了缓存。

```
@Test
    public void test01() {

        SqlSession sqlSession = sqlSessionFactory.openSession();
        try {
            EmpDao mapper = sqlSession.getMapper(EmpDao.class);
            List<Emp> list = mapper.selectAllEmp();
            for (Emp emp : list) {
                System.out.println(emp);
            }
             System.out.println("---------------------------------");
             List<Emp> list2 = mapper.selectAllEmp();
             for (Emp emp : list2) {
                 System.out.println(emp);
            }
        } catch (Exception e) {
             e.printStackTrace();
        } finally {
             sqlSession.close();
        }
    }
```

在大部分的情况下一级缓存是可以的，但是有几种特殊的情况会造成一级缓存失效：

1、一级缓存是sqlSession级别的缓存，如果在应用程序中只有开启了多个sqlsession，那么会造成缓存失效

```
@Test
    public void test02(){
        SqlSession sqlSession = sqlSessionFactory.openSession();
        EmpDao mapper = sqlSession.getMapper(EmpDao.class);
        List<Emp> list = mapper.selectAllEmp();
        for (Emp emp : list) {
```

```
            System.out.println(emp);
        }
        System.out.println("================================");
        SqlSession sqlSession2 = sqlSessionFactory.openSession();
        EmpDao mapper2 = sqlSession2.getMapper(EmpDao.class);
        List<Emp> list2 = mapper2.selectAllEmp();
        for (Emp emp : list2) {
            System.out.println(emp);
        }
        sqlSession.close();
        sqlSession2.close();
    }
```

2、在编写查询的sql语句的时候，一定要注意传递的参数，如果参数不一致，那么也不会缓存结果

3、如果在发送过程中发生了数据的修改，那么结果就不会缓存

```
@Test
    public void test03(){
        SqlSession sqlSession = sqlSessionFactory.openSession();
        EmpDao mapper = sqlSession.getMapper(EmpDao.class);
        Emp empByEmpno = mapper.findEmpByEmpno(1111);
        System.out.println(empByEmpno);
        System.out.println("================================");
        empByEmpno.setEname("zhangsan");
        int i = mapper.updateEmp(empByEmpno);
        System.out.println(i);
        System.out.println("================================");
        Emp empByEmpno1 = mapper.findEmpByEmpno(1111);
        System.out.println(empByEmpno1);
        sqlSession.close();
    }
```

4、在两次查询期间，手动去清空缓存，也会让缓存失效

```
@Test
    public void test03(){
        SqlSession sqlSession = sqlSessionFactory.openSession();
        EmpDao mapper = sqlSession.getMapper(EmpDao.class);
        Emp empByEmpno = mapper.findEmpByEmpno(1111);
        System.out.println(empByEmpno);
        System.out.println("================================");
        System.out.println("手动清空缓存");
```

```
        sqlSession.clearCache();
        System.out.println("================================");
        Emp empByEmpno1 = mapper.findEmpByEmpno(1111);
        System.out.println(empByEmpno1);
        sqlSession.close();
    }
```

2、二级缓存

二级缓存是全局作用域缓存，默认是不开启的，需要手动进行配置。

Mybatis提供二级缓存的接口以及实现，缓存实现的时候要求实体类实现Serializable接口，二级缓存在sqlSession关闭或提交之后才会生效。

1、缓存的使用

步骤：

1、全局配置文件中添加如下配置：

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

2、需要在使用二级缓存的映射文件出使用<cache/>标签标注

3、实体类必须要实现Serializable接口

```
@Test
    public void test04(){
        SqlSession sqlSession = sqlSessionFactory.openSession();
        SqlSession sqlSession2 = sqlSessionFactory.openSession();
        EmpDao mapper = sqlSession.getMapper(EmpDao.class);
        EmpDao mapper2 = sqlSession2.getMapper(EmpDao.class);
        Emp empByEmpno = mapper.findEmpByEmpno(1111);
        System.out.println(empByEmpno);
        sqlSession.close();

        Emp empByEmpno1 = mapper2.findEmpByEmpno(1111);
        System.out.println(empByEmpno1);
        sqlSession2.close();
    }
```

2、缓存的属性

eviction:表示缓存回收策略，默认是LRU

LRU：最近最少使用的，移除最长时间不被使用的对象

FIFO：先进先出，按照对象进入缓存的顺序来移除

SOFT：软引用，移除基于垃圾回收器状态和软引用规则的对象

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

flushInternal:刷新间隔，单位毫秒

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，正整数

代表缓存最多可以存储多少个对象，太大容易导致内存溢出

readonly：只读，true/false

true：只读缓存，会给所有调用这返回缓存对象的相同实例，因此这些对象不能被修改。

false：读写缓存，会返回缓存对象的拷贝（序列化实现），这种方式比较安全，默认值

```
//可以看到会去二级缓存中查找数据，而且二级缓存跟一级缓存中不会同时存在数据，因为二级缓存中的数据是在sqlsession 关闭之后才生效的
@Test
    public void test05(){
        SqlSession sqlSession = sqlSessionFactory.openSession();
        EmpDao mapper = sqlSession.getMapper(EmpDao.class);
        Emp empByEmpno = mapper.findEmpByEmpno(1111);
        System.out.println(empByEmpno);
        sqlSession.close();

         SqlSession sqlSession2 = sqlSessionFactory.openSession();
         EmpDao mapper2 = sqlSession2.getMapper(EmpDao.class);
         Emp empByEmpno2 = mapper2.findEmpByEmpno(1111);
         System.out.println(empByEmpno2);
         Emp empByEmpno3 = mapper2.findEmpByEmpno(1111);
         System.out.println(empByEmpno3);
         sqlSession2.close();
    }
```

// 缓存查询的顺序是先查询二级缓存再查询一级缓存

```
@Test
    public void test05(){
        SqlSession sqlSession = sqlSessionFactory.openSession();
        EmpDao mapper = sqlSession.getMapper(EmpDao.class);
```

```
        Emp empByEmpno = mapper.findEmpByEmpno(1111);
        System.out.println(empByEmpno);
        sqlSession.close();

        SqlSession sqlSession2 = sqlSessionFactory.openSession();
         EmpDao mapper2 = sqlSession2.getMapper(EmpDao.class);
         Emp empByEmpno2 = mapper2.findEmpByEmpno(1111);
         System.out.println(empByEmpno2);
         Emp empByEmpno3 = mapper2.findEmpByEmpno(1111);
         System.out.println(empByEmpno3);

         Emp empByEmpno4 = mapper2.findEmpByEmpno(7369);
         System.out.println(empByEmpno4);
         Emp empByEmpno5 = mapper2.findEmpByEmpno(7369);
         System.out.println(empByEmpno5);
         sqlSession2.close();
    }
```

3、二级缓存的作用范围：

如果设置了全局的二级缓存配置，那么在使用的时候需要注意，在每一个单独的select语句中，可以设置将查询缓存关闭，以完成特殊的设置

1、在setting中设置，是配置二级缓存开启，一级缓存默认一直开启

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

2、select标签的useCache属性：

在每一个select的查询中可以设置当前查询是否要使用二级缓存，只对二级缓存有效

3、sql标签的flushCache属性

增删改操作默认值为true，sql执行之后会清空一级缓存和二级缓存，而查询操作默认是false

4、sqlSession.clearCache()

只是用来清楚一级缓存

4、整合第三方缓存

1.整合redis

1.1.需要安装redis服务：https://github.com/MicrosoftArchive/redis/tags

1.2启动服务：双击redis-server.exe 或 安装到windows服务： windows下redis配 置密码（可选）

1.3测试redis是否能够正常运行：

1. 双击redis-cli.exe
2. 有密码的情况下输入密码

```
auth 密码
```

3.存储缓存set 命令

```
set key value
```

4.获取缓存 get命令

```
get key
```

1.4 添加redis-mybatis 缓存适配器 依赖

```
<dependencies>
 <!--添加依赖-->
 <dependency>
 <groupId>org.mybatis.caches</groupId>
 <artifactId>mybatis-redis</artifactId>
 <version>1.0.0-beta2</version>
 </dependency>
</dependencies>
```

1.5 添加redis.properties在resources根目录

```
host=localhost
port=6379
connectionTimeout=5000
soTimeout=5000
password=无密码可不填
database=0
clientName=
```

1.6 设置mybatis二级缓存实现类

```
<cache
 ...
 type="org.mybatis.caches.redis.RedisCache"
 .../>
```

2.整合ehcache

1、导入对应的maven依赖

```
<!-- https://mvnrepository.com/artifact/org.ehcache/ehcache -->
        <dependency>
```

```
            <groupId>org.ehcache</groupId>
            <artifactId>ehcache</artifactId>
            <version>3.8.1</version>
        </dependency>
        <!--
https://mvnrepository.com/artifact/org.mybatis.caches/mybatis-ehcache -->
        <dependency>
            <groupId>org.mybatis.caches</groupId>
             <artifactId>mybatis-ehcache</artifactId>
             <version>1.2.0</version>
         </dependency>

```

2、导入ehcache配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd">
<!-- 磁盘保存路径 -->
<diskStore path="D:\ehcache" />

<defaultCache
  maxElementsInMemory="1"
  maxElementsOnDisk="10000000"
   eternal="false"
   overflowToDisk="true"
   timeToIdleSeconds="120"
   timeToLiveSeconds="120"
   diskExpiryThreadIntervalSeconds="120"
   memoryStoreEvictionPolicy="LRU">
</defaultCache>
</ehcache>

<!--
属性说明：
l diskStore：指定数据在磁盘中的存储位置。
l 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：
l maxElementsInMemory - 在内存中缓存的element的最大数目
l maxElementsOnDisk - 在磁盘上缓存的element的最大数目，若是0表示无穷大
```

```
27  l 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
28  l 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
29
30  以下属性是可选的：
31  l 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
32  l 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
33  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.
34  l 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。
35  l 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
36  l 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）
37  -->
```

3、在mapper文件中添加自定义缓存

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