MyBatis 的一级缓存和二级缓存

一、前言

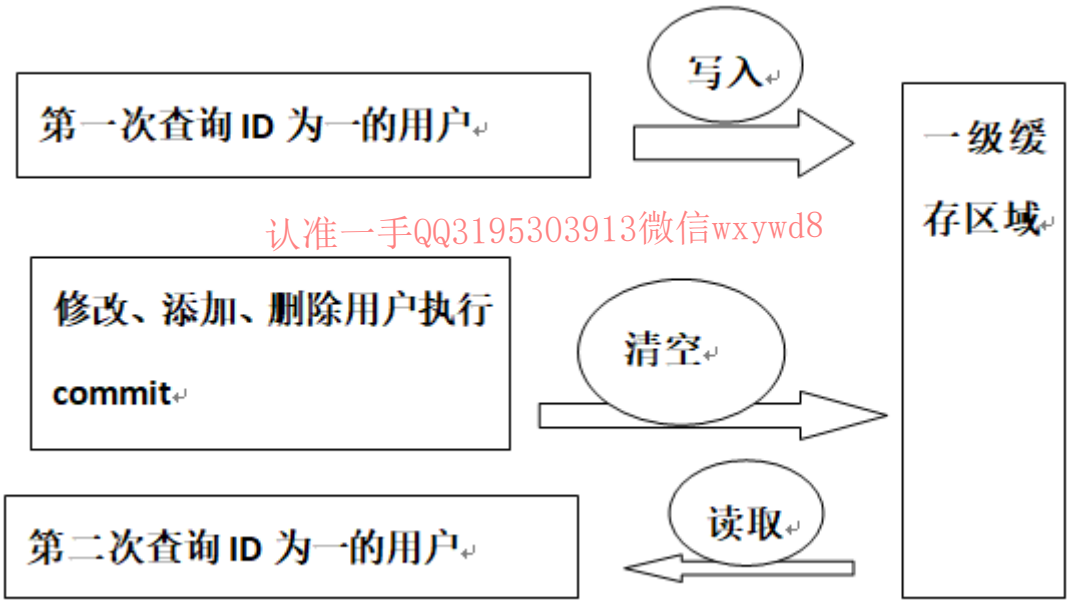
先说缓存，合理使用缓存是优化中最常见的，将从数据库中查询出来的数据放入缓存中，下次使用时不必从数据库查询，而是直接从缓存中读取，避免频繁操作数据库，减轻数据库的压力，同时提高系统性能。

二、一级缓存

一级缓存 是 SqlSession级别的缓存。在操作数据库时需要构造sqlSession对象，在对象中有一个数据结构用于存储缓存数据。

不同的sqlSession之间的缓存数据区域是互相不影响的。也就是他只能作用在同一个sqlSession中，不同的sqlSession中的缓存是互相不能读取的。

一级缓存的工作原理：



用户发起查询请求，查找某条数据，sqlSession 先去缓存中查找，是否有该数据，如果有，读取；如果没有，从数据库中查询，并将查询到的数据放入一级缓存区域，供下次查找使用。

但 sqlSession 执行commit，即增删改操作时会清空缓存。这么做的目的是避免脏读。

如果commit不清空缓存，会有以下场景：

A查询了某商品库存为10件，并将10件库存的数据存入缓存中，之后被客户买走了10件，数据被delete了，但是下次查询这件商品时，并不从数据库中查询，而是从缓存中查询，就会出现错误。

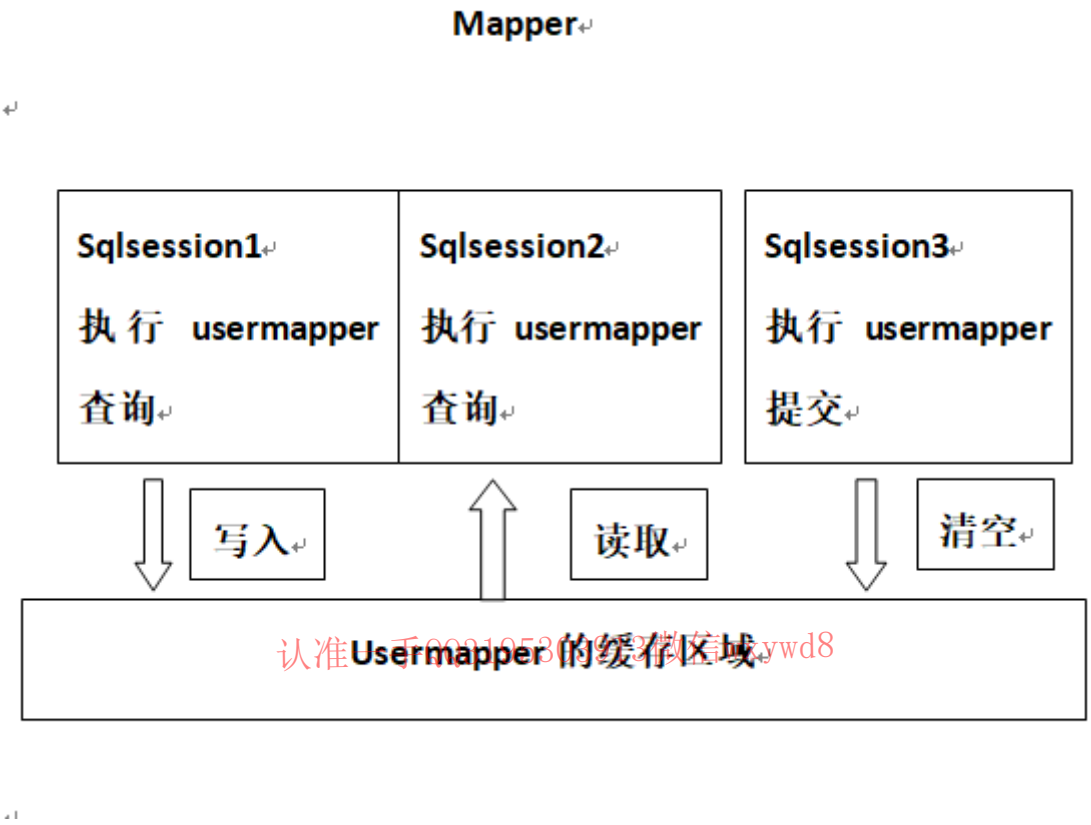
既然有了一级缓存，那么为什么要提供二级缓存呢?

1. 二级缓存是 mapper 级别的缓存，多个SqlSession去操作同一个Mapper的sql语句，多个SqlSession可以共用二级缓存，二级缓存是跨SqlSession的。二级缓存的作用范围更大。
2. 还有一个原因，实际开发中，MyBatis 通常和 Spring 进行整合开发。
   Spring 将事务放到 Service 中管理，对于每一个 service 中的 sqlsession 是不同的，
   这是通过 `mybatis-spring` 中的 `org.mybatis.spring.mapper.MapperScannerConfigurer` 创

建 sqlsession 自动注入到 service 中的。
每次查询之后都要进行关闭 sqlSession ， 关闭之后数据被清空。
所以 spring 整合之后，如果没有事务，**一级缓存** 是没有意义的。

三、二级缓存

二级缓存原理：



二级缓存是 mapper 级别的缓存，多个 SqlSession 去操作同一个Mapper的sql语句，多个 SqlSession 可以共用二级缓存，二级缓存是跨 SqlSession 的。

UserMapper有一个二级缓存区域（按 namespace 划分），每个 mapper 也有自己的二级缓存区域（按namespace分）。

每一个 namespace 的 mapper 都有一个二级缓存区域，如果相同两个 mapper 的 namespace ，这两个mapper执行sql查询到数据将存在相同的二级缓存区域中。

3.1、开启二级缓存：

1，打开总开关

在MyBatis的全局配置文件中加入：

```
<settings>
   <!-- 开启二级缓存  -->
    <setting name="cacheEnabled" value="true"/>
</settings>
```

2，在需要开启二级缓存的 mapper.xml 中加入 cache 标签 ：

```
<cache/>
```

3，让使用二级缓存的 POJO 类实现 Serializable 接口

```
public class User implements Serializable {
}
```

3.2、测试

```
@Test
public void testCache2() throws Exception {
    SqlSession sqlSession1 = sqlSessionFactory.openSession();
    SqlSession sqlSession2 = sqlSessionFactory.openSession();

    UserMapper userMapper1 = sqlSession1.getMapper(UserMapper.class);
    User user1 = userMapper1.findUserById(1);
    System.out.println(user1);
    sqlSession1.close();

    UserMapper userMapper2 = sqlSession2.getMapper(UserMapper.class);
    User user2 = userMapper2.findUserById(1);
    System.out.println(user2);
    sqlSession2.close();
}
```

输出结果：

```
DEBUG [main] - Cache Hit Ratio [com.iot.mybatis.mapper.UserMapper]: 0.0
DEBUG [main] - Opening JDBC Connection
DEBUG [main] - Created connection 103887628.
DEBUG [main] - Setting autocommit to false on JDBC Connection
[com.mysql.jdbc.JDBC4Connection@631330c]
DEBUG [main] - ==>  Preparing: SELECT * FROM user WHERE id=?
DEBUG [main] - ==> Parameters: 1(Integer)
DEBUG [main] - <==      Total: 1
User [id=1, username=张三, sex=1, birthday=null, address=null]
DEBUG [main] - Resetting autocommit to true on JDBC Connection
[com.mysql.jdbc.JDBC4Connection@631330c]
DEBUG [main] - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@631330c]
DEBUG [main] - Returned connection 103887628 to pool.
DEBUG [main] - Cache Hit Ratio [com.iot.mybatis.mapper.UserMapper]: 0.5
User [id=1, username=张三, sex=1, birthday=null, address=null]
```


我们可以从打印的信息看出，两个sqlSession，去查询同一条数据，只发起一次select查询语句，第二次直接从Cache中读取。

前面我们说到，Spring和MyBatis整合时， 每次查询之后都要进行关闭sqlSession，关闭之后数据被清空。所以spring整合之后，如果没有事务，一级缓存是没有意义的。

那么如果开启二级缓存，关闭 sqlsession 后，会把该 sqlsession 一级缓存中的数据添加到namespace的二级缓存中。这样，缓存在sqlsession关闭之后依然存在。

3.3、cache 标签的属性

```
<cache
    eviction="FIFO"              // 回收策略为先进先出
    flushInterval="60000"        // 自动刷新时间60s
    size="512"                   // 最多缓存512个引用对象
    readOnly="true"/>            // 只读
```

cache 标签可指定如下属性，每种属性的指定都是针对都是针对底层Cache的一种装饰，采用的是装饰器的模式。

- blocking：默认为false，当指定为true时将采用BlockingCache进行封装，blocking，阻塞的意思。

  使用BlockingCache会在查询缓存时锁住对应的Key，如果缓存命中了则会释放对应的锁，否则会在查询数据库以后再释放锁，这样可以阻止并发情况下多个线程同时查询数据，详情可参考BlockingCache的源码。
  简单理解，也就是设置true时，在进行增删改之后的并发查询，只会有一条去数据库查询，而不会并发。

- eviction：eviction 就是 驱逐的意思，也就是元素驱逐算法，默认是LRU。

  LRU 对应的就是LruCache，其默认只保存1024个Key，超出时按照最近最少使用算法进行驱逐，详情请参考LruCache的源码。

  如果想使用自己的算法，则可以将该值指定为自己的驱逐算法实现类，只需要自己的类实现Mybatis的Cache接口即可。

  除了LRU以外，系统还提供：

  - FIFO（先进先出，对应FifoCache）
  - SOFT（采用软引用存储Value，便于垃圾回收，对应SoftCache）
  - WEAK（采用弱引用存储Value，便于垃圾回收，对应WeakCache）这三种策略。

    这里，根据个人需求选择了，没什么要求的话，默认的LRU即可。

- flushInterval：清空缓存的时间间隔，单位是毫秒，默认是不会清空的。

  当指定了该值时会再用ScheduleCache包装一次，其会在每次对缓存进行操作时判断距离最近一次清空缓存的时间是否超过了flushInterval指定的时间，如果超出了，则清空当前的缓存，详情可参考ScheduleCache的实现。

- readOnly：是否只读 。默认为false。
  当指定为false时，底层会用SerializedCache包装一次，其会在写缓存的时候将缓存对象进行序列化，然后在读缓存的时候进行反序列化，这样每次读到的都将是一个新的对象，即使你更改了读取到的结果，也不会影响原来缓存的对象，即非只读，你每次拿到这个缓存结果都可以进行修改，而不会影响原来的缓存结果；

  当指定为true时，那就是每次获取的都是同一个引用，对其修改会影响后续的缓存数据获取，这种情况下是不建议对获取到的缓存结果进行更改，意为只读。 （不建议设置为true）

  这是Mybatis二级缓存读写和只读的定义，可能与我们通常情况下的只读和读写意义有点不同。每次都进行序列化和反序列化无疑会影响性能，但是这样的缓存结果更安全，不会被随意更改，具体可根据实际情况进行选择。详情可参考SerializedCache的源码。

- size：用来指定缓存中最多保存的Key的数量。

  其是针对LruCache而言的，LruCache默认只存储最多1024个Key，可通过该属性来改变默认值，当然，如果你通过eviction指定了自己的驱逐算法，同时自己的实现里面也有setSize方法，那么也可以通过cache的size属性给自定义的驱逐算法里面的size赋值。

- type：指定当前底层缓存实现类，默认是PerpetualCache。

  如果我们想使用自定义的Cache，则可以通过该属性来指定，对应的值是我们自定义的Cache的全路径名称。

3.4、cache-ref 标签

```
<cache-ref namespace="cn.chenhaoxiang.dao.UserMapper"/>
```

`cache-ref` 可以用来指定其它 Mapper.xml 中定义的Cache，有的时候可能我们多个不同的 Mapper 需要共享同一个缓存的
是希望在MapperA中缓存的内容在MapperB中可以直接命中的，这个时候我们就可以考虑使用cache-ref，这种场景只需要保证它们的缓存的Key是一致的即可命中，二级缓存的Key是通过Executor接口的createCacheKey()方法生成的，其实现基本都是BaseExecutor。

四、总结：

对于查询多、commit少且用户对查询结果实时性要求不高，此时采用 mybatis 二级缓存技术降低数据库访问量，提高访问速度。

但不能滥用二级缓存，二级缓存也有很多弊端，从MyBatis默认二级缓存是关闭的就可以看出来。
二级缓存是建立在同一个 namespace下的，如果对表的操作查询可能有多个 namespace，那么得到的数据就是错误的。

举个简单的例子:

订单 和 订单详情 分别是 orderMapper、orderDetailMapper。

在查询订单详情（orderDetailMapper）时，我们需要把订单信息（orderMapper）也查询出来，那么这个订单详情（orderDetailMapper）的信息被二级缓存在 orderDetailMapper 的 namespace中，这个时候有人要修改订单的基本信息（orderMapper），那就是在 orderMapper 的 namespace 下修改，他是不会影响到 orderDetailMapper 的缓存的，那么你再次查找订单详情时，拿到的是缓存的数据，这个数据其实已经是过时的。

二级缓存的使用原则

1. **只能在一个命名空间下使用二级缓存**
   由于二级缓存中的数据是基于namespace的，即不同 namespace 中的数据互不干扰。
   在多个namespace中存在对同一个表的操作，那么这多个namespace中的数据可能就会出现不一致现象。
2. **在单表上使用二级缓存**
   如果一个表与其它表有关联关系，那么就非常有可能存在多个 namespace 对同一数据的操作。
   而不同 namespace 中的数据相互干扰，所以就有可能出现多个 namespace 中的数据不一致现象。
3. **查询多于修改时使用二级缓存**
   在查询操作远远多于增删改操作的情况下可以使用二级缓存。因为任何增删改操作都将刷新二级缓存，对二级缓存的频繁刷新将降低系统性能。