链滴

mybatis 源码解读及缓存机制

作者：lvyeqqq
原文链接：https://ld246.com/article/1553854772490
来源网站：链滴

1.MyBatis的SqlSession的四大对象：

Executor、StatemenHandler、ParameterHandler、ResultHandler

2.SqlSessionFactoryBuilder、SqlSessionFactory、SqlSession简介

官方对SqlSessionFactoryBuilder、SqlSessionFactory、SqlSession的作用域范围分别是：方法范
、应用范围、请求或方法范围。
其中，SqlSessionFactoryBuilder这个类可以被实例化，使用和丢弃，一点那创
建了SqlSessionFactory，就不需要它了。因此 SqlSessionFactoryBuilder 实例的
最佳范围是方法范围（也就是局部方法变量）。
SqlSessionFactory 一旦被创建就应该在应用的运行期间一直存在，没有任何理
由对它进行清除或重建。使用 SqlSessionFactory 的最佳实践是在应用运行期间
不要重复创建多次，多次重建 SqlSessionFactory 被视为一种代码“坏味道（bad
smell）”。因此 SqlSessionFactory 的最佳范围是应用范围。有很多方法可以
做到，最简单的就是使用单例模式或者静态单例模式。

每个线程都应该有它自己的 SqlSession 实例。SqlSession 的实例不是线程安全
的，因此是不能被共享的，所以它的最佳的范围是请求或方法范围。

3.sqlsession 创建

参见5中mybatis的初始化

4. mybatis 的一级缓存

SqlSession级别的缓存，操作数据库时需要构造SQLSession对象， 在对象中有一个数据结构（Hash
ap）用于存储缓存数据，不同的SQLSession对象之间的
缓存数据是不共享的，即独立的.
spring 中 结合 mybatis中，默认情况下，数据库处于自动提交模式，每一条sql
语句处于一个单独的事务中，语句执行完毕时，如果执行成功则隐式提交事
务。而mybatis的一级缓存在这种情况下是无效的，想要一级缓存起作用，则要

开启事务：

下面用spring整合mybatis来测试一下mybatis的一级缓存：

1、下面是service层实现， 可以看到，我两次查询了同一个数据，理论上由于mybatis中默认开启一级缓存， 那么第二次肯定时要从缓存中获取，而不是创建SqlSession对象重新从数据库获取

```
@Autowired
private LsjmUserMapper lsjmUserMapper;

@Override
public LsjmUser getUser() {
        // 第一次查询
        LsjmUser user = lsjmUserMapper.getUserByName("300");
        System.out.println(user.toString());

        // 第二次查询
        LsjmUser user1 = lsjmUserMapper.getUserByName("300");
        System.out.println(user1.toString());
        return user;
}
```

前台页面触发这个service后，控制台打印：

从日志信息可以很明显的看到，代码中的两次查询构建了两个SqlSession对象，也就是说第二次查询并没有从前一次的SqlSession缓存中获取，而是自己新建一个SQLSession对象，重新查询，，**看似，mybatis的一级缓存失效了?**

开启事务后

下面Service层中的代码同样对同一个数据查询了两次，这次开启了事务管理

```java
@Autowired
private LsjmUserMapper lsjmUserMapper;

@Override
@Transactional   // 开启事务控制，当前，spring配置文件中得先配置好
public LsjmUser getUser() {
        // 第一次查询
        LsjmUser user = lsjmUserMapper.getUserByName("300");
        System.out.println(user.toString());

        // 第二次查询
        LsjmUser user1 = lsjmUserMapper.getUserByName("300");
        System.out.println(user1.toString());
        return user;
}
```

前台页面触发Service后：控制台打印日志：

可以看出来第一次查询时，构造了一个SqlSession对象，从数据库查询数据，然后将查询的结果存储到一级缓存SqlSession中，第二次查询时，直接Fetched SqlSession，而不是再重新建一个，此时就是从缓存中直接取数据了

```
mybatis.spring.SqlSessionUtils.debug(JakartaCommonsLoggingImpl.java:54)] - Creating a new SqlSession
mybatis.spring.SqlSessionUtils.debug(JakartaCommonsLoggingImpl.java:54)] - Registering transaction synchronization for SqlSession [org.apache.ibatis.session.de
mybatis.spring.transaction.SpringManagedTransaction.debug(JakartaCommonsLoggingImpl.java:54)] - JDBC Connection [com.mysql.jdbc.JDBC4Connection@7142cdd7] will
haol.mapper.LsjmUserMapper.getUserByName.debug(JakartaCommonsLoggingImpl.java:54)] - ==>  Preparing: SELECT uname, pwd, phone, address, breed_type, field_size
haol.mapper.LsjmUserMapper.getUserByName.debug(JakartaCommonsLoggingImpl.java:54)] - ==> Parameters: 300(String)
haol.mapper.LsjmUserMapper.getUserByName.debug(JakartaCommonsLoggingImpl.java:54)] - <==      Total: 1
mybatis.spring.SqlSessionUtils.debug(JakartaCommonsLoggingImpl.java:54)] - Releasing transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSes
, breed_type=养殖户, field_size=2000-4000]
mybatis.spring.SqlSessionUtils.debug(JakartaCommonsLoggingImpl.java:54)] - Fetched SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@2502e36e] f
mybatis.spring.SqlSessionUtils.debug(JakartaCommonsLoggingImpl.java:54)] - Releasing transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSes
, breed_type=养殖户, field_size=2000-4000]
mybatis.spring.SqlSessionUtils.debug(JakartaCommonsLoggingImpl.java:54)] - Transaction synchronization committing SqlSession [org.apache.ibatis.session.default
mybatis.spring.SqlSessionUtils.debug(JakartaCommonsLoggingImpl.java:54)] - Transaction synchronization deregistering SqlSession [org.apache.ibatis.session.defa
mybatis.spring.SqlSessionUtils.debug(JakartaCommonsLoggingImpl.java:54)] - Transaction synchronization closing SqlSession [org.apache.ibatis.session.defaults.D
springframework.jdbc.datasource.DataSourceTransactionManager.processCommit(AbstractPlatformTransactionManager.java:763)] - Initiating transaction commit
```

5.原理分析

mybatis的初始化：

1.首先会创建SqlSessionFactoryBuilder建造者对象，然后由它进行创建SqlSessionFactory

2.然后会解析xml配置文件，实际为configuration节点的解析操作，还要解析transactionManager及datasource，最后将解析后的结果存到configuration对象中。

3.解析完MyBatis配置文件后，configuration就初始化完成了，然后根据configuration对象来创建SqlSession，到这里时，MyBatis的初始化的征程已经走完了。

mybatis 的SQL查询流程：

1.调用selectOne方法进行SQL查询，selectOne方法最后调用的是selectList，在selectList中，会查询configuration中存储的MappedStatement对象，mapper文件中一个sql语句的配置对应一个MappedStatement对象，然后调用执行器进行查询操作。

2.执行器在query操作中，优先会查询缓存是否命中，命中则直接返回，否则从数据库中查询。

3. 真正的doQuery操作是由SimplyExecutor代理来完成的，该方法中有2个子流程，一个是SQL参数的设置，另一个是SQL查询操作和结果集的封装。

3.1 首先获取数据库connection连接，然后准备statement，然后就设置SQL查询中的参数值。打开一个connection连接，在使用完后不会close，而是存储下来，当下次需要打开连接时就直接返回。

final.参考

1、MyBatis一级缓存的生命周期和SqlSession一致。
2、MyBatis一级缓存内部设计简单，只是一个没有容量限定的HashMap，在缓存的功能性上有所欠缺。
3、MyBatis的一级缓存最大范围是SqlSession内部，有多个SqlSession或者分布式的环境下，数据库写操作会引起脏数据，建议设定缓存级别为Statement。
4、MyBatis的二级缓存相对于一级缓存来说，实现了SqlSession之间缓存数据的共享，同时粒度更加的细，能够到namespace级别，通过Cache接口实现类不同的组合，对Cache的可控性也更强。
5、MyBatis在多表查询时，极大可能会出现脏数据，有设计上的缺陷，安全使用二级缓存的条件比较苛刻。
6、在分布式环境下，由于默认的MyBatis Cache实现都是基于本地的，分布式环境下必然会出现读取到脏数据，需要使用集中式缓存将MyBatis的Cache接口实现，有一定的开发成本，直接使用Redis、Memcached等分布式缓存可能成本更低，安全性也更高。
7. 个人建议MyBatis缓存特性在生产环境中进行关闭，单纯作为一个ORM框架使用可能更为合适。