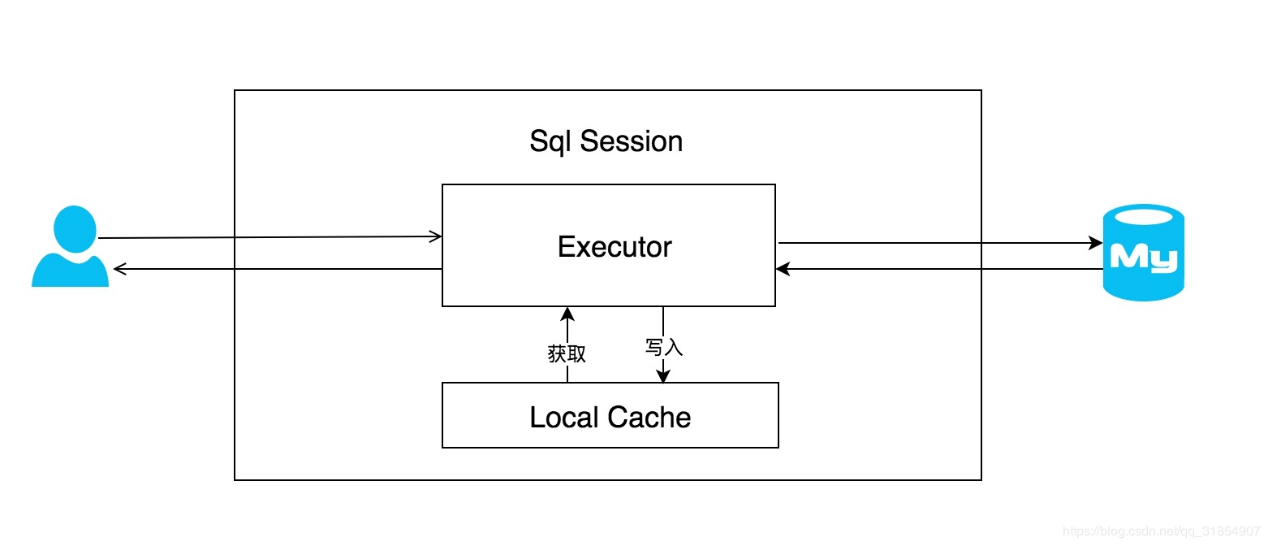
**学习笔记**

**Mybatis缓存**

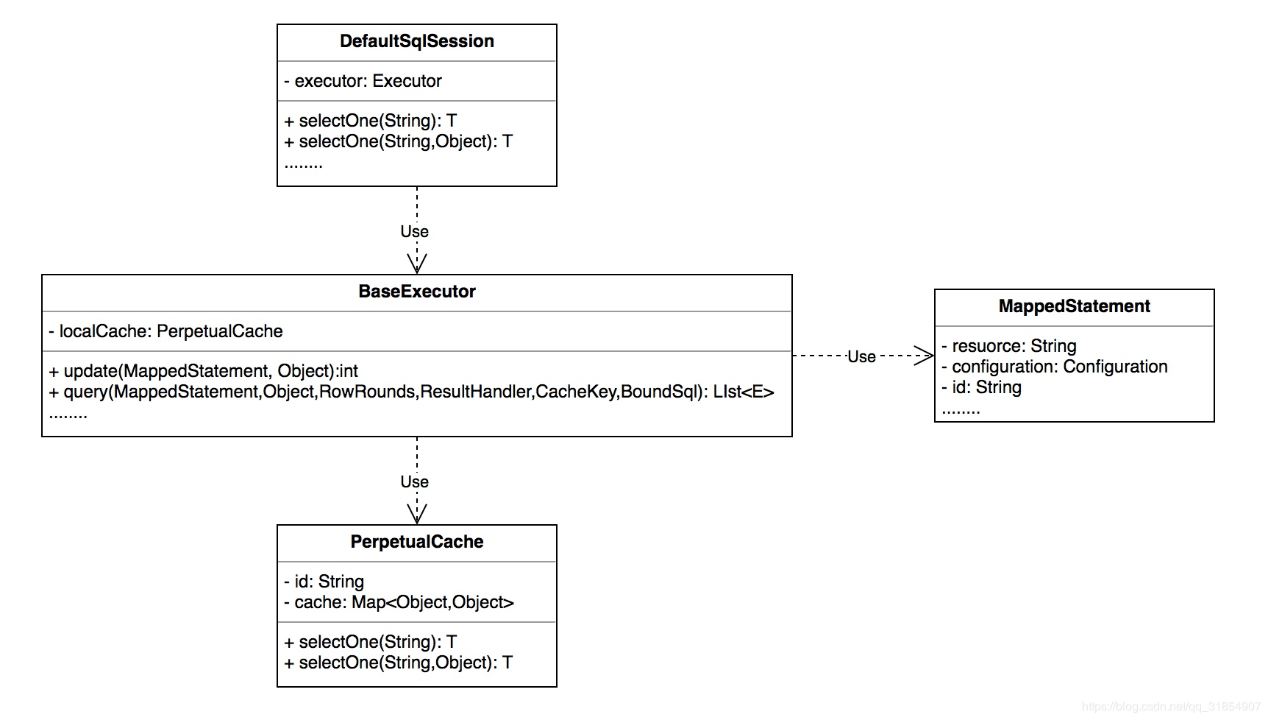
**一级缓存**

**一级缓存介绍**

在应用运行过程中，我们有可能在一次数据库会话中，执行多次查询条件完全相同的SQL，MyBatis提供了一级缓存的方案优化这部分场景，如果是相同的SQL语句，会优先命中一级缓存，避免直接对数据库进行查询，提高性能。具体执行过程如下图所示。



每个SqlSession中持有了Executor，每个Executor中有一个LocalCache。当用户发起查询时，MyBatis根据当前执行的语句生成MappedStatement，在Local Cache进行查询，如果缓存命中的话，直接返回结果给用户，如果缓存没有命中的话，查询数据库，结果写入Local Cache，最后返回结果给用户。具体实现类的类关系图如下图所示。



**一级缓存配置**

需在MyBatis的配置文件中，添加如下语句，就可以使用一级缓存。共有两个选项，SESSION或者STATEMENT，默认是SESSION级别，即在一个MyBatis会话中执行的所有语句，都会共享这一个缓存。一种是STATEMENT级别，可以理解为缓存只对当前执行的这一个Statement有效。

<setting name="localCacheScope" value="SESSION"/>

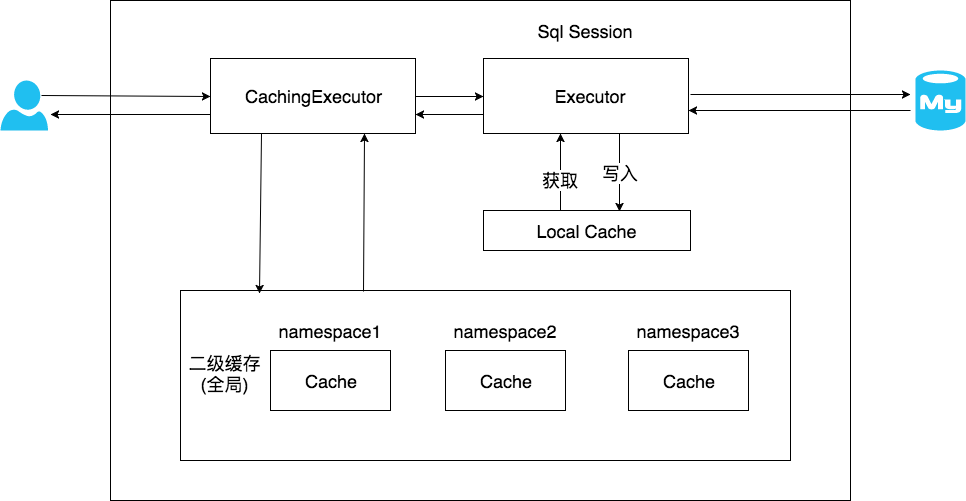
**总结**

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。

**二级缓存**

**二级缓存介绍**

在上文中提到的一级缓存中，其最大的共享范围就是一个SqlSession内部，如果多个SqlSession之间需要共享缓存，则需要使用到二级缓存。开启二级缓存后，会使用CachingExecutor装饰Executor，进入一级缓存的查询流程前，先在CachingExecutor进行二级缓存的查询，具体的工作流程如下所示。



二级缓存开启后，同一个namespace下的所有操作语句，都影响着同一个Cache，即二级缓存被多个SqlSession共享，是一个全局的变量。

当开启缓存后，数据的查询执行的流程就是 二级缓存 -> 一级缓存 -> 数据库。

**二级缓存配置**

要正确的使用二级缓存，需完成如下配置的。

1. 在MyBatis的配置文件中开启二级缓存。

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

1. 在MyBatis的映射XML中配置cache或者 cache-ref 。

cache标签用于声明这个namespace使用二级缓存，并且可以自定义配置。

<cache/>

* type：cache使用的类型，默认是PerpetualCache，这在一级缓存中提到过。
* eviction： 定义回收的策略，常见的有FIFO，LRU。
* flushInterval： 配置一定时间自动刷新缓存，单位是毫秒。
* size： 最多缓存对象的个数。
* readOnly： 是否只读，若配置可读写，则需要对应的实体类能够序列化。
* blocking： 若缓存中找不到对应的key，是否会一直blocking，直到有对应的数据进入缓存。

cache-ref代表引用别的命名空间的Cache配置，两个命名空间的操作使用的是同一个Cache。

<cache-ref namespace="mapper.StudentMapper"/>

**总结**

1. 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。
2. 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。
3. 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。

**Mybatis使用的设计模式**

| **模式** | **Mybatis体现** |
| --- | --- |
| builder模式 | SqlSessionFactoryBuilder、Environment |
| 工厂方法模式 | SqlSessionFactory、TransactionFactory、LogFactory |
| 单例模式 | ErrorContext、LogFactory |
| 代理模式 | MapperProxy、ConnectionLogger(JDK动态代理)、 executor.loader(使用cglib达到延迟记载的效果) |
| 组合模式 | SqlNode |
| 模版方法模式 | BaseExecutor、SimpleExecutor、BaseTypeHandler和其子类 |
| 适配器模式 | Log的Mybatis接口和它对jdbc、log4j等各种日志框架的适配实现 |
| 装饰器模式 | cache.decorators子包中等各个装饰者的实现 |
| 迭代器模式 | PropertyTokenizer |

环境设置：必须是如下搭配

<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.20</version>  
</dependency>  
<dependency>  
 <groupId>com.mchange</groupId>  
 <artifactId>c3p0</artifactId>  
 <version>0.9.5.2</version>  
</dependency>

<dataSource>  
 <property name="driverClass" value="com.mysql.cj.jdbc.Driver"></property>  
 <property name="jdbcUrl" value="jdbc:mysql:///m\_mbatis?serverTimezone=UTC"></property>  
 <property name="username" value="root"></property>  
 <property name="password" value="root"></property>  
</dataSource>

而且必须链接时加Timezone。

Debug时，用step-in看

List<Object> list = simpleExecutor.query(configuration, mappedStatement, params);

的public <E> List<E> query(Configuration configuration, MappedStatement mappedStatement, Object... params) throws Exception {

方法可以看到很多执行细节和老师的原理分析有参照对应关系。