Mybatis

1、JDBC 执行流程

1、加载JDBC驱动、建立并获取数据库连接

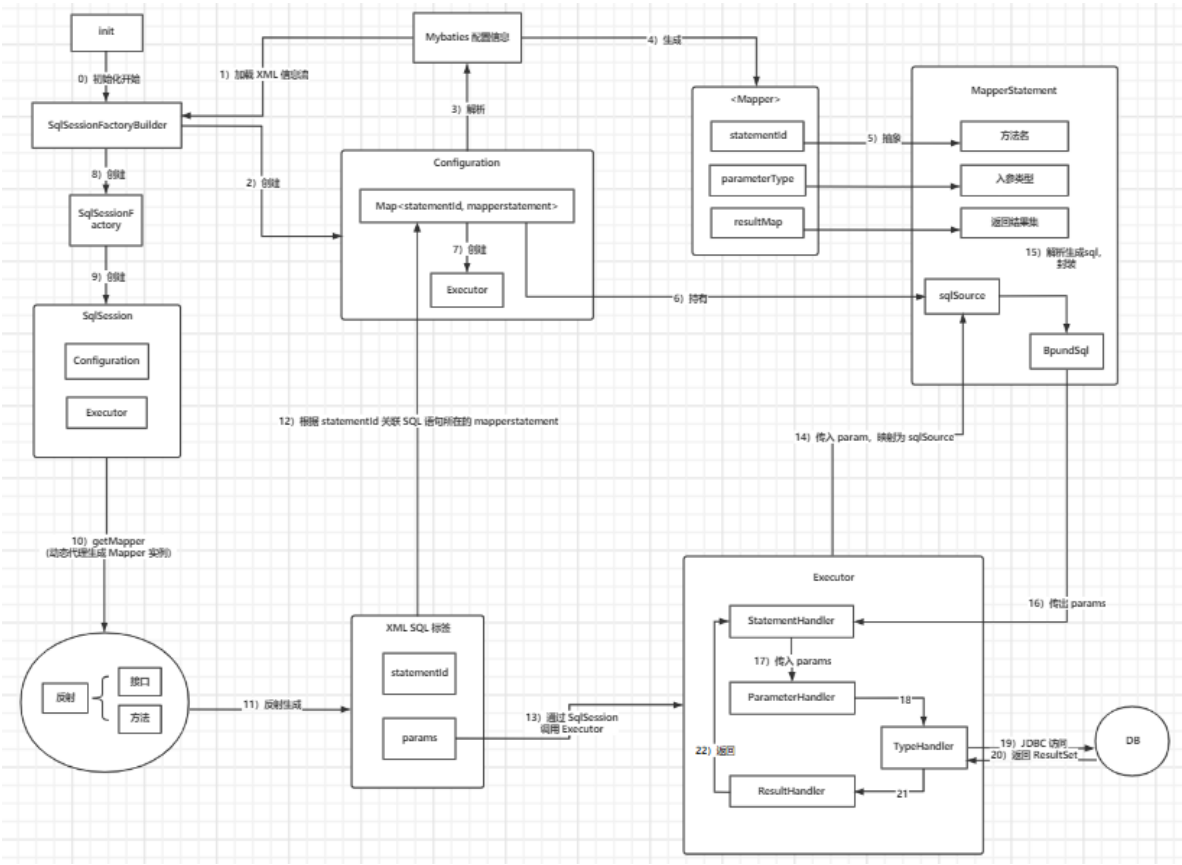
3、创建JDBC Statements对象、设置SQL语句的传入参数

5、执行SQL语句并获得查询结果、对查询结果进行转换处理并将处理结果返回

6、释放相关资源（关闭Connection,关闭Statement,关闭ResultSet）

> **加载驱动、建立并获取连接、创建语句对象、设置语句和参数、执行语句返回结果、对结果转换、释放资源**

2、实现原理



Mybatis的实现原理，主要由两个方面来体现，初始化和调用。

初始化主要是配置文件的解析过程，首先是配置文件的解析，因为一个语句标签主要包含三部分内容，分别是语句信息、参数信息、和结果集信息，所以具体解析也是围绕这三部分进行，将这三部分信息解析成对象存储在全局的配置对象 Configuration 中，用不同的 Map 存储不同种类的标签信息，key 就是对应的标签中的 id，value 就是存储不同标签信息的对象，Configuration（MappedStatement、ParameterMap、ResultMap）。

具体的过程，首先会解析这些包含 sql 的 XML 配置文件，然后把相关的语句 id、语句信息、参数信息、结果集等信息都存储在一个 MapperStatement 对象中，比如 XML 中有增删改查 4 个 sql 配置，那么就会对应的有 4 个 MapperStatement 对象。然后再把这些对象作为 value ，以他们对应的 xml 中的语句 id 作为 key，存到一个 map 中，就是刚才说的配置信息对象中的 Map。那么这一步主要就是对于 XML 信息的解析，直白的说就是用一个 Map 存起来。

解析完配置以后，就要初始化一些功能组件。首先通过 SqlSessionFactoryBuilder 建造者模式创建一个 SqlSessionFactory，用来生产一个 sql 通信会话的工厂，再由这个工厂创建出一个接口层访问对象 sqlSession，当程序执行到一个 Mpper 接口的某个 sql 方法时，sqlSession 以动态代理的方式生成一个 Mapper 代理对象，然后这个代理对象会通过反射机制得到调用者的接口名、方法名、和参数信息等。然后将接口的全限定名和方法名拼接起来就形成一个和 XML 配置文件中相关的语句 id，然后通过这个语句 id 作为 key，在配置信息对象 Configuration 中的三个映射器 Map 找到对应的配置，比如有语句信息、参数信息、和结果集的映射信息等。

这就完成了从接口方法到具体 XML 文件的映射，有了这些信息以后，sqlSession 先把这些信息连同调用者的输入参数一起传递给内部的语句生成器和参数处理器，语句生成器负责解析和编译这些零散的信息形成 sql 语句，如果 是带 #{ } 被解析为一个参数占位符？， ${ } 仅仅为一个纯碎的 string 替换。

而参数处理器负责将输入参数转换成 JDBC 语句所需要的参数，同时类型转换器把相关的 java 对象转为 jdbc 数据类型。到这里一个带有参数的 jdbc sql 语句就形成了，然后交给执行引擎 Executor，它再委托给具体与数据库交互的一个语句处理器 statementHandler 对象，由它进行数据库访问和操作，并将结果返回，然后交给一个结果集映射器，把结果映射成 java 对象，并缓存起来，然后再经过一步步地回传，把结果通过 sqlSession 传递给应用层的调用者。

> 输入参数通过Executor传递给封装在映射器MapperStatement中的sqlSource对象，该对象用来解析生成SQL语句并封装到BoundSql对象中。

【组件】

1、SqlSession：作为mybatis工作的主要顶层API，表示和数据库交互的会话，完成必要的数据库增删改查功能。

2、Executor：mybatis执行器，是Mybatis调度的核心，负责SQL语句的生成和查询缓存的维护。

3、StatementHandler：封装了JDBCStatement操作，负责对JDBC Statement的操作。

4、ParameterHandler：负责对用户传递的参数转换成JDBC Statement所需要的参数。

5、ResultSetHandler：负责将JDBC返回的ResultSet结果集转换成List类型的集合。

6、TypeHandler：负责Java数据类型和jdbc数据类型之间的映射和转换。

7、MappedStatement：维护了一条select/update/delete/insert节点的封装。

8、Sqlsource：负责根据用户传递的parameterObject，动态生成SQL语句，将信息封装在BoundSql对象中，并返回。

9、BoundSql：表示动态生成的SQL语句以及相应的参数信息。

10、Configuration：Mybatis所有的配置信息都维护在这个对象中。

【使用】

1、添加mybatis-spring的包

2、配置SqlSessionFactory：配置数据源 datasource（Druid德鲁伊）、全局配置文件的位置 configLocation

3、配置映射器，启动自动扫描

【总结】

初始化

1）配置文件解析

- MapperStatement（语句 id、语句信息、参数信息、结果集）
- Map<id, MapperStatement>

2）功能组件构造

- SqlSessionFactory
- sqlSession（sql 语句的拼装、参数的传递、类型转换、具体执行、以及结果集的映射和返回）
- Mapper（动态代理、映射）

调用

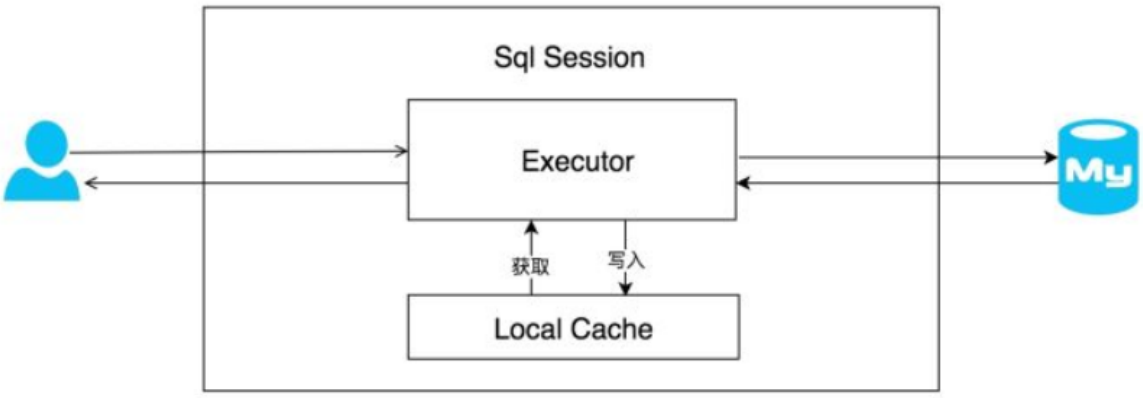
sqlSession -> 语句生成器 -> 参数处理器 -> 类型转换器 -> 执行引擎 -> 结果映射器

3、一级缓存

概述

Mybatis 的一级缓存存在于 SqlSession 的生命周期中，在同一个 SqlSession 中查询时，Mybatis 会把执行的方法和参数通过算法生成缓存的键值，将键值和查询结果缓存在一个 Map。如果同一个 SqlSession 中执行的方法和参数完全一样，那么通过算法生成的键值就一样，就会返回缓存中的同个对象。

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。

在同个 SqlSession 中的增删改操作会刷新缓存，但是查询不会，所以如果查询之前先对缓存中的对象进行 set 操作，此时再读的时候就会读到这个对象 set 之后的最新数据，如果查询的时候不希望使用一级缓存，可以在 xml 语句中增加 flushCache="true" 来声明。

配置

我们来看看如何使用MyBatis一级缓存。开发者只需在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"/>
```

小结

1. MyBatis一级缓存的生命周期和SqlSession一致。
2. 增删改可以刷新缓存，查询不会
3. 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。
4. 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。

   eg： 开启两个SqlSession，在sqlSession1中查询数据，使一级缓存生效，在sqlSession2中更新数据库。 sqlSession2更新了id为1的学生的姓名，从凯伦改为了小岑，但session1之后的查询中，id为1的学生的名字还是凯伦，出现了脏数据，也证明了之前的设想，一级缓存只在数据库会话内部共享。

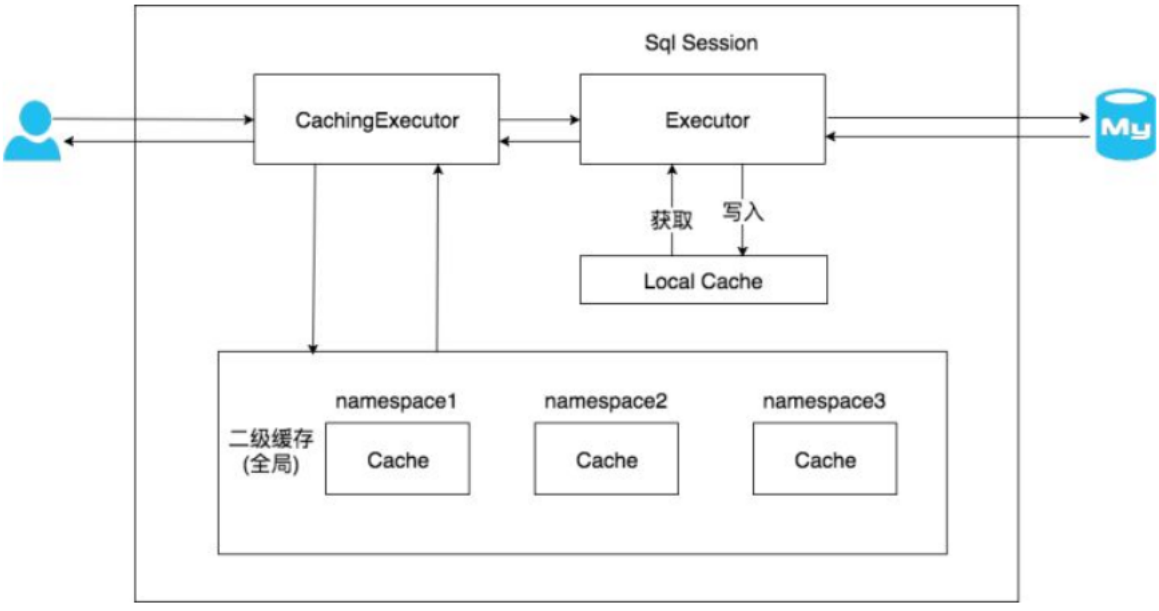
> **SqlSession -> Executor -> LocalCache -> Map**
>
> **生命周期是 SqlSession、增删改可以刷新缓存，查询不会、内部用 Map 实现缓存**
>
> **分布式环境多个 SqlSession 的写操作会产生脏数据**

> **在 SqlSession 内部有一个 Executor，Executor 持有一个缓存，每次调用方调用时，会根据方法和参数通过算法生成缓存的键值，将键值和查询结果缓存在一个 Map，下次再用同样的方式请求，就会直接返回这个结果对象。而不同的 SqlSession 即使是相同的请求方式，生成的 key 相同，但是缓存的实例对象不是同一个，就会造成多个 SqlSession 并发写的时候，一个 session 执行了写操作修改了字段，但其缓存只在本 session 中有效，另一个 session 的缓存看不到这种修改结果，造成数据过期的脏数据的问题。**

4、二级缓存

概述

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，进入一级缓存的查询流程前，会先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，在 mysbtis 中就是同个 mapper 接口对应的同个 xml 文件，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。

配置

Mybatis 的二级缓存存在于 SqlSessionFactory 的生命周期中，可以让多个 SqlSession 共享。二级缓存默认是关闭的，如果要开启需要两步，第一步是开启全局设置的开关，

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

第二步是在对应的 Mapper 接口的 XML 文件中配置一个 cache 标签，并设置相应的属性，因为 Mybatis 的二级缓存是和命名空间绑定的，也就是每个 xml 文件。

```
<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。

特点

语句会被缓存

所有 Insert、update、delete 增删改语句会刷新缓存

可以设置缓存回收算法：LRU（最近最少使用）、FIFO等

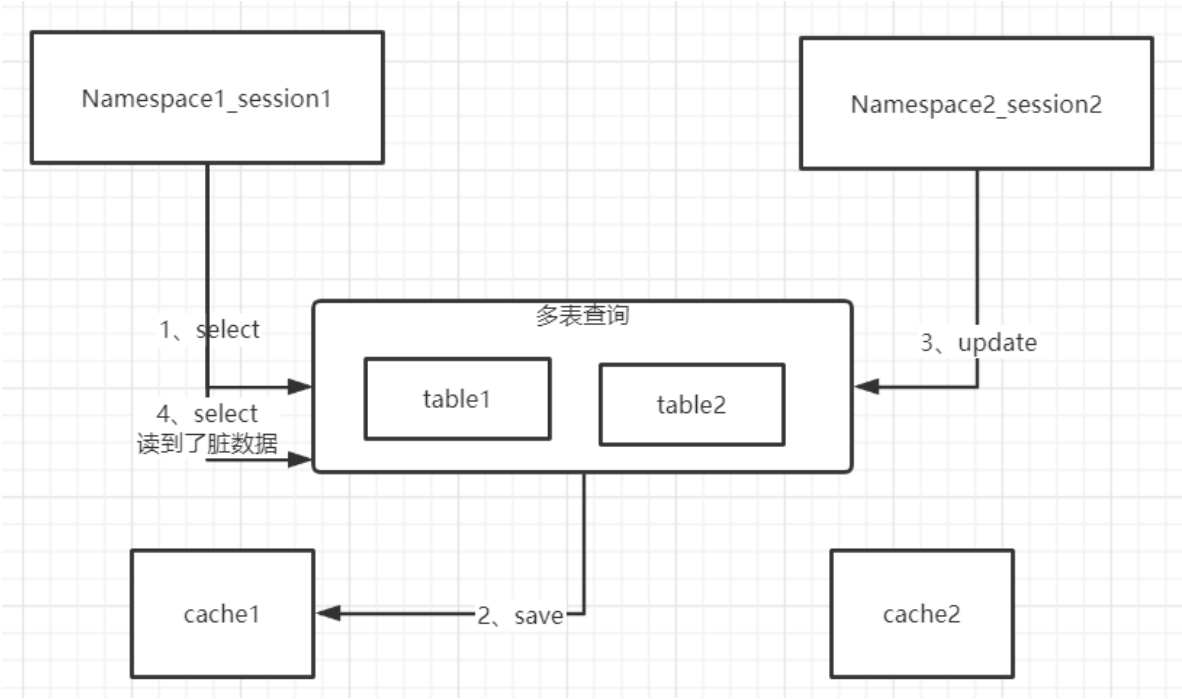
可以设置缓存的对象的个数（默认是1024个）

可以设置缓存的读写性质，如果是只读的话，缓存会返回相同的实例，性能最高；如果是可读可写的话，缓存会通过序列化返回缓存对象的拷贝，会慢一些，同时需要缓存的对象实现了序列化的接口，但可以保证线程安全。

多表查询脏数据问题

引入两张表，一张class，一张classroom。class中保存了班级的id和班级名，classroom中保存了班级id和学生id。我们在StudentMapper中增加了一个查询方法getStudentByIdWithClassInfo，用于查询学生所在的班级，涉及到多表查询。在ClassMapper中添加了updateClassName，根据班级id更新班级名的操作。

当sqlsession1的studentmapper查询数据后，二级缓存生效。保存在StudentMapper的namespace下的cache中。当sqlSession3的classMapper的updateClassName方法对class表进行更新时，updateClassName不属于StudentMapper的namespace，所以StudentMapper下的cache没有感应到变化，没有刷新缓存。当StudentMapper中同样的查询再次发起时，从缓存中读取了脏数据。



为了解决上面的问题，可以使用 Cache ref，可以让多个命名空间的 Mapper 共享一个缓存，这样的话，session2 的修改对于 session1 就可见了。不过这样做的后果是，缓存的粒度变粗了，多个 Mapper namespace下的所有操作都会对缓存使用造成影响。

小结

1. **MyBatis的二级缓存相对于一级缓存来说，其生命周期扩大了，从 sqlSession 到 sqlSessionFactory。**
2. **第二是实现了不同SqlSession之间缓存数据的共享。**
3. **同时缓存的粒度更加的细，从原先的同个 sqlSession 内部到 namespace 级别。**
4. 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。
5. 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。
6. 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。

> **两次配置（回收算法、缓存个数、读写性质：读相同实例、写序列化拷贝）**
>
> **多表查询脏数据（不同命名空间即不同会话不能共享一个缓存）、使用 cache_ref 让多个会话共享一个缓存**

> **这种缓存也会引发脏数据的问题，只不过脏数据的范围扩大了。比如有两个命名空间1和2，即有两个 Mapper 文件，分别对应着两张表table1和2，这时命名空间2有一个修改的操作是同时修改两张表的，这个操作产生的缓存只会存在命名空间2对应的缓存中，不会被命名空间1感知，所以命名空间1的数据变成了过期数据，就引发了读到脏数据的问题。**

5、Q&A

1、#{}和${}的区别是什么?

#{ }变量是在进行 sql 动态解析预编译阶段将被解析为一个参数占位符？，然后在将要执行的时候由 sqlSession 的参数处理器进行参数的替换，可以防止 SQL 注入。而 ${ } 只是在动态 SQL 解析阶段简单替换字符串，其 SQL 语句已经不包含变量了，是常量数据。

例如：select * from emp where name=#{empName}，参数传入empName->Smith，解析执行后的SQL是：select * from emp where name=?。但是对于select * from emp where name=${empName}，参数传入empName->Smith，解析执行后的SQL是：select * from emp where name='Smith'。

> **#{ }变量是在进行 sql 动态解析预编译阶段将被解析为一个参数占位符？，然后在将要执行的时候由 sqlSession 的参数处理器进行参数的替换，可以防止 SQL 注入。而 ${ } 只是在动态 SQL 解析阶段简单替换字符串，其 SQL 语句已经不包含变量了，是常量数据。**

SQL 注入：

```
select * from user where (name = '"+ username +"');

username 被恶意填入：username = "1' OR '1'='1";

就会变成 select * from user where (name = '1' or '1'='1')

实际上是 select * from user;导致不用用户名密码登录也行。
```

SQL 注入就是通过注入一些带有特殊符号与恶意执行的SQL语句，填入被注入的语句，然后消除了某些条件，或者达成了实行新的注入的SQL语句的目的。因为 ${ } 只是简单地做字符串的拼接，就有可能会使得可以通过两个引号拼接字符串1来达成条件的消除，引发隐患。而 #{ } 是预编译的时候现将条件变成占位符，然后在后面进行参数的设置，而不是直接设置字符串，就可以消除这种由字符串替换带来的安全隐患。

2、通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？

Dao接口，就是人们常说的Mapper接口，接口的全限名，就是映射文件中的namespace的值，接口的方法名，就是映射文件中MappedStatement的id值，接口方法内的参数，就是传递给sql的参数。Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可唯一定位一个MappedStatement，举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到namespace为com.mybatis3.mappers.StudentDao下面id = findStudentById的MappedStatement。在Mybatis中，每一个 ▼ 、、、标签均会被解析为MappedStatement对象，标签内的sql会被解析为BoundSql对象。

> **Configuration（MappedStatement、ParameterMap、ResultMap）**

8、当实体类中的属性名和表中的字段名不一样，应该怎么办？Mybatis是如何将sql执行结果封装为目标对象并返回的？都有哪些映射形式？

1）通过在sql语句中定义别名，这样的话表字段就可以通过别名和实体类的属性建立联系

```
<select id="getByOrderId" parameterType="java.lang.Long"
resultType="com.demo.entity.OrderInfo">
select order_id OrderId, order_sn orderSn, total_fee totalFee, create_time
createTime
from order_info where order_id=#{orderId}
</select>
```

2）通过resultMap来映射数据表的字段名和实体类的属性名之间的对应关系（推荐）。

```
<resultMap id = "BaseResultMap" type="com.demo.entity.OrderInfo">
    <id property="OrderId" column="order_id"/>
    <result property="orderSn" column="order_sn"/>
    <result property="totalFee" column="total_fee"/>
    <result property="createTime" column="create_time"/>
</resultMap>
<select id="getByOrderId" parameterType="java.lang.Long"
resultMap="BaseResultMap">
select order_id, order_sn, total_fee, create_time
from order_info where order_id=#{orderId}
</select>
```

Mybatis会忽略列名大小写，有了列名与属性名的映射关系后，Mybatis通过反射创建对象 resultMap 定义的对象，然后使用反射给对象的属性逐一赋值，值就是表的字段并返回。

> **别名、resultMap、反射创建对象，设置属性值**

9、如何获取自动生成的主键?

我：一般我们插入数据的话，如果想要知道刚刚插入的数据的主键是多少，可以通过以下方式来获取。

通过**LAST_INSERT_ID**（）获取刚插入记录的自增主键值，在insert语句执行之后，执行select LAST_INSERT_ID()就可以获取自增主键。

```
<insert id='insert' parameterType="com.demo.entity.OrderInfo"
<selectKey keyProperty="orderId" order="AFTER" resultType="java.lang.Long">

select LAST_INSERT_ID()
</selectKey>
insert into order_info(order_sn,total_fee,create_time)
values(#{orderSn},#{totalFee},#{createTime})</insert>
```

> **在语句标签中使用 LAST_INSERT_ID**

10、你知道mybatis的哪些动态sql？除了常见的select|insert|update|delete标签之外，还有哪些标签?

还有很多其他的标签，加上动态sql的9个标签，
trim|where|set|foreach|if|choose|when|otherwise|bind等，其中为sql片段标签，通过标签引入sql片段，为不支持自增的主键生成策略标签。

if：做条件判断的，如果不使用这个标签，肯定要在代码中做判断，比如元素是否为空，字符串是否是空字符串，还比如一些特定的枚举值需要判断执行条件。

choose/when/otherwise：这个标签组合类似于if/else if.../else，就是多个选项中选择一个，如果都不满足条件，那只能执行中的内容了。

例如：

```
<select id="getStudentListChoose" parameterType="Student"
resultMap="BaseResultMap">
    SELECT * from STUDENT WHERE 1=1
    <where>
        <choose>
```

```
            <when test="Name!=null and student!='' ">
                AND name LIKE CONCAT(CONCAT('%', #{student}),'%')
            </when>
            <when test="hobby!= null and hobby!= '' ">
                AND hobby = #{hobby}
            </when>
            <otherwise>
                AND AGE = 15
            </otherwise>
        </choose>
    </where>
</select>
```

3.foreach标签：用于循环。例如：

```
<select id="listByOrderIds" resultMap="BaseResultMap">
    select * from order_info where order_id in
    <foreach collection="list" item="item" open="(" close=")" separator=",">
        #{item}
    </foreach>
</select>
```

4.另外还有set标签，where标签，trim标签等。

> **if（条件判断）、choose/when/otherwise（多条件判断类似于 if else ）、foreach（循环遍历）、set、where**

11、在mapper中如何传递多个参数?

我：有两种方法：

1、使用占位符的思想：

（1）在映射文件中使用#{0},#{1}代表传递进来的第几个参数。

（2）使用@param注解来命名参数（推荐使用） 例如：

```
//mapper接口
public OrderInfo getByOrderIdAndStatus(Long orderId, String status);

//mapper.xml文件
<select id="getByOrderIdAndStatus" resultMap="BaseResultMap">
    select * from order_info where order_id=#{0} and status=#{1}
</select>
//mapper接口
public OrderInfo getByOrderIdAndStatus(@param("orderId")Long orderId,
@param("status")String status);

//mapper.xml文件
<select id="getByOrderIdAndStatus" resultMap="BaseResultMap">
    select * from order_info where order_id=#{orderId} and status=#{status}
</select>
```

2、使用Map集合作为参数来装载

```
Map<String, Object> map = new HashMap();
map.put("orderId", 1L);
map.put("status", "NORMAL");
OrderInfo orderinfo = getByOrderIdAndStatus(map);

//mapper接口
public OrderInfo getByOrderIdAndStatus(Map<String, Object> map);

//mapper.xml文件
<select id="getByOrderIdAndStatus" parameterType="map"
resultMap="BaseResultMap">
    select * from order_info where order_id=#{orderId} and status=#{status}
</select>
```

> **@param 注解来修饰参数，可以让传入的参数按照注解的字符串匹配找到 xml 中相应的 sql 的条件。**
>
> **可以用 Map 集合来装载参数，接口中直接传入 Map 到 xml 文件中，然后 xml 的 sql 配置把参数类型改成 map，key 要和 sql 的条件相同**

12、为什么需要预编译

JDBC 中使用对象 **PreparedStatement** 来抽象预编译语句，使用预编译。

预编译阶段可以优化 sql 的执行。预编译之后的 sql 多数情况下可以直接执行，DBMS 不需要再次编译，越复杂的sql，编译的复杂度将越大，预编译阶段可以合并多次操作为一个操作。

预编译语句对象可以重复利用。把一个 sql 预编译后产生的 PreparedStatement 对象缓存下来，下次对于同一个sql，可以直接使用这个缓存的 PreparedState 对象。

mybatis 默认情况下，将对所有的 sql 进行预编译。

参考资料

聊聊MyBatis缓存机制

https://zhuanlan.zhihu.com/p/33179093