Mybatis

一、HelloWorld

- 使用SqlSession根据方法id进行操作
- 使用SqlSession获取映射器进行操作

二、全局配置文件

configuration 配置：

• properties 属性

如果属性在不只一个地方进行了配置，那么 MyBatis 将按 照下面的顺序来加载：

– 在 properties 元素体内指定的属性首先被读取。

– 然后根据 properties 元素中的 resource 属性读取类路径下属性文件或根 据 url 属性指定的路径读取属性文件，并覆盖已读取的同名属性。

– 最后读取作为方法参数传递的属性，并覆盖已读取的同名属性

• settings 设置

极为重要的调整设置，它们会改变 MyBatis 的运行时行为。

设置参数	描述	有效值	默认值
cacheEnabled	该配置影响的所有映射器中配置的缓存的全局开关。	true | false	TRUE
lazyLoadingEnabled	延迟加载的全局开关。当开启时，所有关联对象都会延迟加载。特定关联关系中可通过设置 fetchType 属性来覆盖该项的开关状态。	true | false	FALSE
useColumnLabel	使用列标签代替列名。不同的驱动在这方面会有不同的表现，具体可参考相关驱动文档或通过测试这两种不同的模式来观察所用驱动的结果。	true | false	TRUE
defaultStatementTimeout	设置超时时间，它决定驱动等待数据库响应的秒数。	Any positive integer	Not Set (null)
mapUnderscoreToCamelCase	是否开启自动驼峰命名规则（camel case）映射，即从经典数据库列名 A_COLUMN 到经典 Java 属性名 aColumn 的类似映射	true | false	FALSE

• typeAliases 类型命名

• typeHandlers 类型处理器

无论是 MyBatis 在预处理语句（PreparedStatement）中 设置一个参数时，还是从结果集中取出一个值时， 都会用类型处理器将获取的值以合适的方式转换成 Java 类型。

XXXTypeHandler

自定义类型处理器

我们可以重写类型处理器或创建自己的类型处理 器来处理不支持的或非标准的类型。

步骤：

1）、实现org.apache.ibatis.type.TypeHandler接口或 者继承org.apache.ibatis.type.BaseTypeHandler

2）、指定其映射某个JDBC类型（可选操作）

3）、在mybatis全局配置文件中注册

• objectFactory 对象工厂

• plugins 插件

通过插件来修改MyBatis的一些核心行为。插件通过动态代理机制，可以介入四大对象的任何 一个方法的执行。

• environments 环境

• environment 环境变量

• transactionManager 事务管理器

type = JDBC | MANAGED | 自定义

• dataSource 数据源

type = UNPOOLED | POOLED | JNDI | 自定义

实际开发中我们使用Spring管理数据源，并进行事务控制的配置来覆盖上述配置

• databaseIdProvider 数据库厂商标识

• mappers 映射器

1. mapper逐个注册SQL映射文件

2. 或者使用批量注册：

这种方式要求SQL映射文件名必须和接口名相同并且在同一目录下

三、映射文件

映射文件指导着MyBatis如何进行数据库增删改查， 有着非常重要的意义；

• cache –命名空间的二级缓存配置

•cache-ref – 其他命名空间缓存配置的引用。

•resultMap – 自定义结果集映射

id 和 result 映射一个单独列的值到**简单数据类型**

1. association 1. 嵌套结果集 2. 分段查询（select、column） 3. 分段查询&延迟加载
2. collection 1. 集合类型&嵌套结果集 2. 分步查询&延迟加载
3. 多列值封装map传递 1. 分步查询的时候通过column指定，将对应的列的数据 传递过去，我们有时需要传递多列数据。 2. 使用{key1=column1,key2=column2...}的形式

•sql –抽取可重用语句块。

•insert – 映射插入语句

主键生成方式

1. 若数据库**支持自动生成主键**的字段（比如 MySQL 和 SQL Server），则可以设置 **useGeneratedKeys=”true”, 然后再把 keyProperty** 设置到目标属性上。

2. 而对于不支持自增型主键的数据库（例如 Oracle），则可以使用 **selectKey** 子元素： **selectKey 元素将会首先运行，id 会被设置，然 后插入语句会被调用**

selectKey

keyProperty	**selectKey 语句结果应该被设置的目标属性。**
keyColumn	匹配属性的返回结果集中的列名称。
resultType	**结果的类型。**MyBatis 通常可以推算出来，但是为了更加确定写上也不会有什么问题。MyBatis 允许任何简单类型用作主键的类型，包括字符串。
order	**可以被设置为 BEFORE 或 AFTER。如果设置为 BEFORE，那么它会首先选择主键，设置 keyProperty 然后执行插入语句。**如果设置为 AFTER，那么先执行插入语句，然后是 selectKey 元素
statementType	与前面相同，MyBatis 支持 STATEMENT，PREPARED 和 CALLABLE 语句的映射类型，分别代表 PreparedStatement 和 CallableStatement 类型

•update – 映射更新语句

•delete – 映射删除语句

insert、update、delete元素

id	**命名空间中的唯一标识符**
parameterType	将要传入语句的参数的完全限定类名或别名。这个属性是可选的，因为 **MyBatis 可以通过 TypeHandler 推断出具体传入语句的参数类型**，默认值为 unset。
flushCache	将其设置为 true，任何时候只要语句被调用，都会导致本地缓存和二级缓存都会被清空，默认值：true（对应插入、更新和删除语句）。
timeout	这个设置是在抛出异常之前，驱动程序**等待数据库返回请求结果的秒数**。默认值为 unset（依赖驱动）。
statementType	STATEMENT，PREPARED 或 CALLABLE 的一个。这会让 MyBatis 分别使用 Statement，**PreparedStatement** 或 CallableStatement，**默认值：PREPARED**。
useGeneratedKeys	（仅对 insert 和 update 有用）这**会令 MyBatis 使用 JDBC 的 getGeneratedKeys 方法来取出由数据库内部生成的主键**（比如：像 MySQL 和 SQL Server 这样的关系数据库管理系统的自动递增字段），默认值：false
keyProperty	（仅对 insert 和 update 有用）唯一标记一个属性，**MyBatis 会通过 getGeneratedKeys 的返回值或者通过 insert 语句的 selectKey 子元素设置它的键值**，默认：unset。
keyColumn	（仅对 insert 和 update 有用）通过生成的键值设置表中的列名，这个设置仅在某些数据库（像 PostgreSQL）是必须的，当主键列不是表中的第一列的时候需要设置。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。
databaseId	**如果配置了 databaseIdProvider，MyBatis 会加载所有的不带 databaseId 或匹配当前 databaseId 的语句**；如果带或者不带的语句都有，则不带的会被忽略。

#{key}：获取参数的值，预编译到SQL中。安全。

${key}：获取参数的值，拼接到SQL中。有SQL注入问题。ORDER BY ${name}

•select – 映射查询语句

parameterType	将会传入这条语句的参数类的完全限定名或别名。这个属性是可选的，因为 **MyBatis 可以通过 TypeHandler 推断出具体传入语句的参数**，默认值为 unset。
resultType	从这条语句中返回的**期望类型的类的完全限定名或别名。注意如果是集合，那应该是集合可以包含的类型**，而不能是集合本身。该属性**和 resultMap，不能同时使用。**
resultMap	外部 resultMap 的命名引用。和 resultType 属性不能同时使用。
flushCache	将其设置为 true，任何时候只要语句被调用，都会导致本地缓存和二级缓存都会被清空，默认值：false
useCache	将其设置为 true，将会导致本条语句的结果被二级缓存，默认值：对 select 元素为 true。
timeout	这个设置是在抛出异常之前，**驱动程序等待数据库返回请求结果的秒数**。默认值为 unset（依赖驱动）。
fetchSize	影响驱动程序**每次批量返回的结果行数**。默认值为 unset（依赖驱动）。
statementType	STATEMENT，PREPARED 或 CALLABLE 的一个。这会让 MyBatis 分别使用 Statement，PreparedStatement 或 CallableStatement，默认值：PREPARED。
resultSetType	FORWARD_ONLY，SCROLL_SENSITIVE 或 SCROLL_INSENSITIVE 中的一个，默认值为 unset （依赖驱动）
databaseId	**如果配置了 databaseIdProvider，MyBatis 会加载所有的不带 databaseId 或匹配当前 databaseId 的语句**；如果带或者不带的语句都有，则不带的会被忽略。
resultOrdered	这个设置仅针对嵌套结果 select 语句适用：如果为 true，就假设包含了嵌套结果集或是分组，这样当返回一个主结果行，就不会发生有对前面结果集引用的情况。这就使得在获取嵌套的结果集的时候不至于导致内存不够用。默认值：false
resultSets	这个设置仅对多结果集的情况适用，它将列出语句执行后返回的结果集并每个结果集给一个名称，名称是逗号分隔的

四、动态SQL

– if

– choose (when, otherwise)

– trim (where, set)

– foreach

– bind

五、缓存机制

- 一级缓存
  - 同一次会话期间只要查询过的数据都会保存在当 前SqlSession的一个Map中

    • key:hashCode+查询的SqlId+编写的sql查询语句+参数
  - 一级缓存失效的四种情况

    1、不同的SqlSession对应不同的一级缓存

    2、同一个SqlSession但是查询条件不同

    3、同一个SqlSession两次查询期间执行了任何一次增 删改操作

    4、同一个SqlSession两次查询期间手动清空了缓存
- 二级缓存

  • 二级缓存(second level cache)，全局作用域缓存

  • 二级缓存默认不开启，需要手动配置

  • MyBatis提供二级缓存的接口以及实现，缓存实现要求 POJO实现Serializable接口

  • 二级缓存在 SqlSession关闭或提交之后才会生效

- 缓存相关属性
  - eviction="FIFO"：缓存回收策略：
  - flushInterval：刷新间隔，单位毫秒
  - size：引用数目，正整数
  - readOnly：只读，true/false • true：只读缓存；会给所有调用者返回缓存对象的相同实例。因此这些对象 不能被修改。这提供了很重要的性能优势。 • false：读写缓存；会返回缓存对象的拷贝（通过序列化）。这会慢一些， 但是安全，因此默认是 false。
- 缓存有关配置

  1、全局setting的cacheEnable： 2、select标签的useCache属性： 3、sql标签的flushCache属性： 4、sqlSession.clearCache()： 5、当在某一个作用域 (一级缓存Session/二级缓存Namespaces) 进行了 C/U/D 操作后，默认该作用域下所 有 select 中的缓存将被clear。
- 第三方缓存整合
  - EhCache 是一个纯Java的进程内缓存框架，具有快速、精干等特点，是Hibernate中默认的CacheProvider。
  - MyBatis定义了Cache接口方便我们进行自定义扩展。

- 当执行一条查询SQL时，流程为 从二级缓存中进行查询 进入一级缓存中查询 执行 JDBC 查询。

六、Mybatis-Spring整合

整合关键配置

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
<!-- 指定mybatis全局配置文件位置 -->
<property name="configLocation" value="classpath:mybatis/mybatis-config.xml"></property>
<!--指定数据源 -->
<property name="dataSource" ref="dataSource"></property>
<!--mapperLocations：所有sql映射文件所在的位置 -->
<property name="mapperLocations" value="classpath:mybatis/mapper/*.xml"></property>
<!--typeAliasesPackage：批量别名处理-->
<property name="typeAliasesPackage" value="Zh1Cheung.bean"></property>
</bean>
<!--自动的扫描所有的mapper的实现并加入到ioc容器中 -->
<bean id="configure" class="org.mybatis.spring.mapper.MapperScannerConfigurer">
<!--basePackage:指定包下所有的mapper接口实现自动扫描并加入到ioc容器中 -->
<property name="basePackage" value="Zh1Cheung.dao"></property>
</bean>
```

七、逆向工程

MBG使用

使用步骤： 1）编写MBG的配置文件（重要几处配置） 1）jdbcConnection配置数据库连接信息 2）javaModelGenerator配置javaBean的生成策略 3）sqlMapGenerator 配置sql映射文件生成策略 4）javaClientGenerator配置Mapper接口的生成策略 5）table 配置要逆向解析的数据表 tableName：表名 domainObjectName：对应的javaBean名

2）运行代码生成器生成代码 • 注意： Context标签 targetRuntime="MyBatis3"可以生成带条件的增删改查 targetRuntime="MyBatis3Simple"可以生成基本的增删改查 如果再次生成，建议将之前生成的数据删除，避免xml向后追加内容出现的问题

$号使用在具体pojo类也就是非基本类型的取值，而#号使用在具体有基本类型的取值

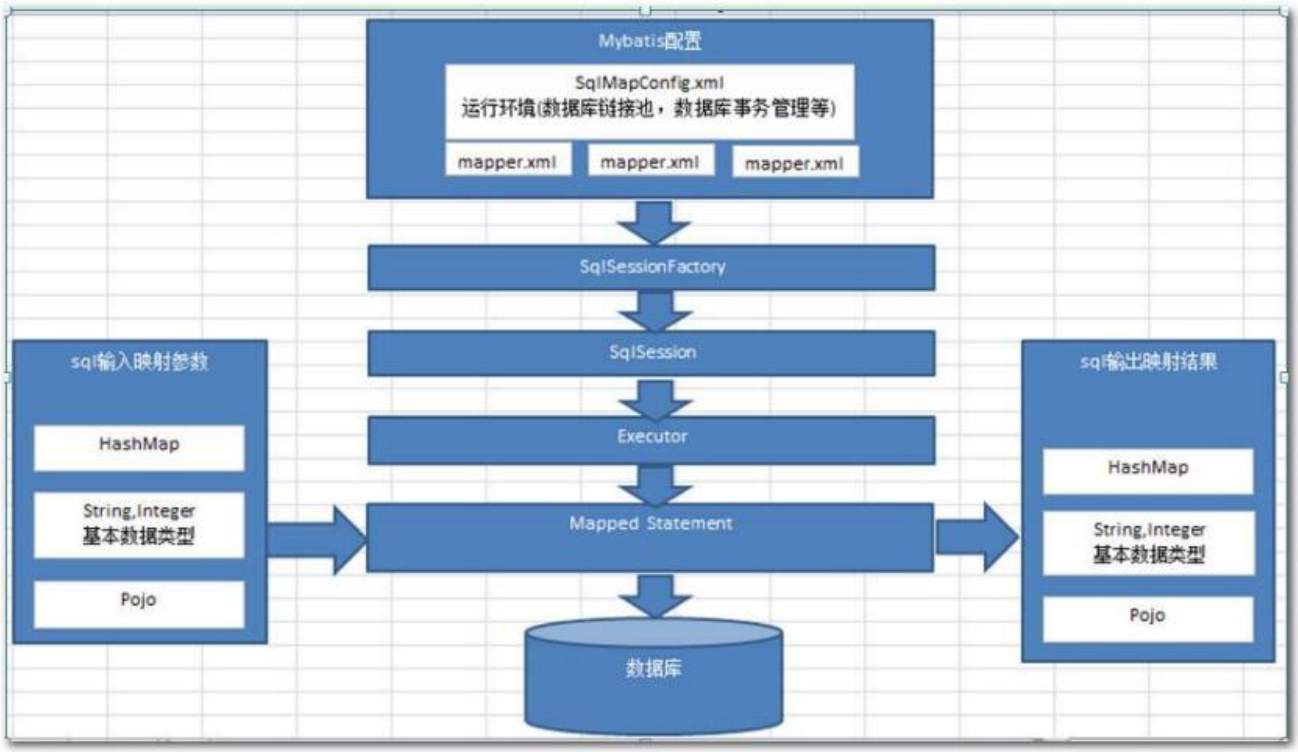
#可以进行预编译，进行类型匹配，#变量名# 会**转化为 jdbc 的 类型** $不进行数据类型匹配，$变量名$就直接把$name$**替换为 name的内容** 例如： select * from tablename where id = #id# ，假设id的值为12,其中如果数据库字段id为字符型，那么#id#表示的就是'12'，如果id为整型，那么#id#就是 12 会转化为jdbc的 select * from tablename where id=? ，把? 参数设置为id的值 select * from tablename where id = id ，如果字段id为整型，Sql语句就不会出错，但是如果字段id为字符型，那么Sql语句应该写成 select * from table where id = 'id'

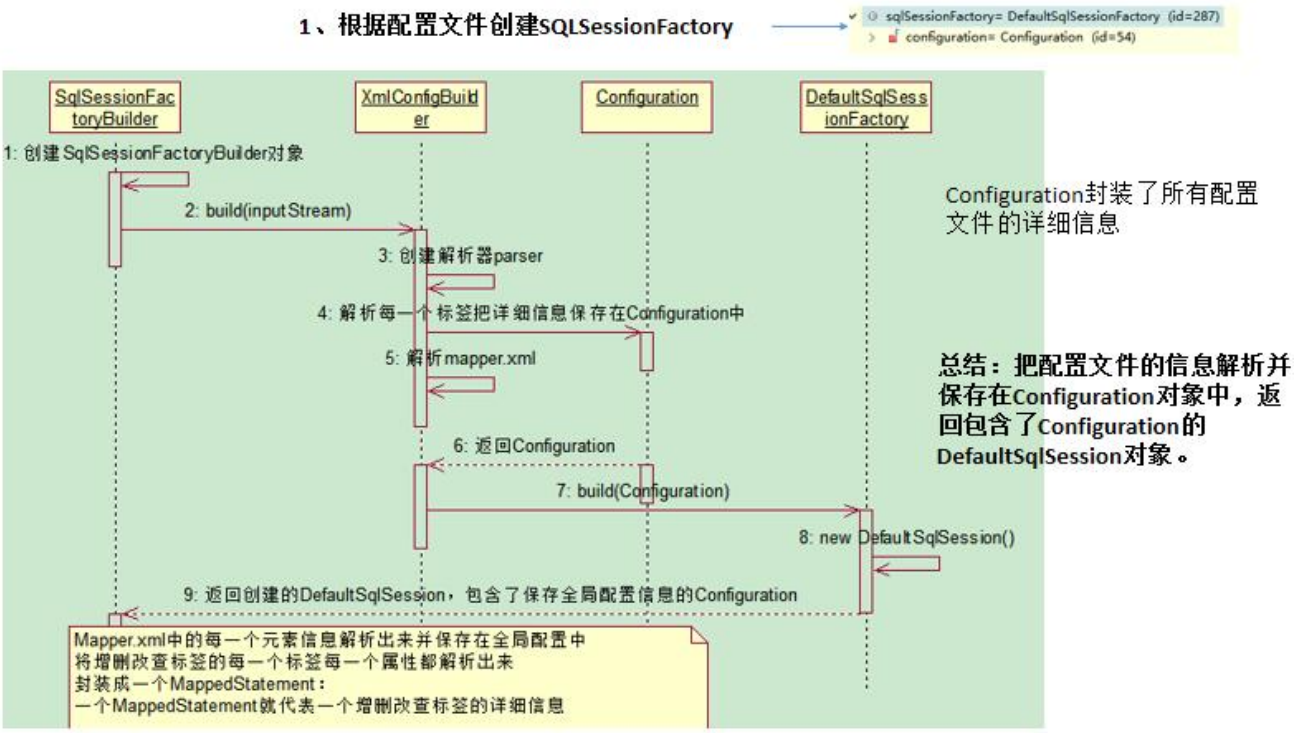
Criteria包含一个Cretiron的集合,每一个Criteria对象内包含的Cretiron之间是由AND连接的,是逻辑与的关系。

Example内有一个成员叫oredCriteria,是Criteria的集合,就想其名字所预示的一样，这个集合中的Criteria是由OR连接的，是逻辑或关系

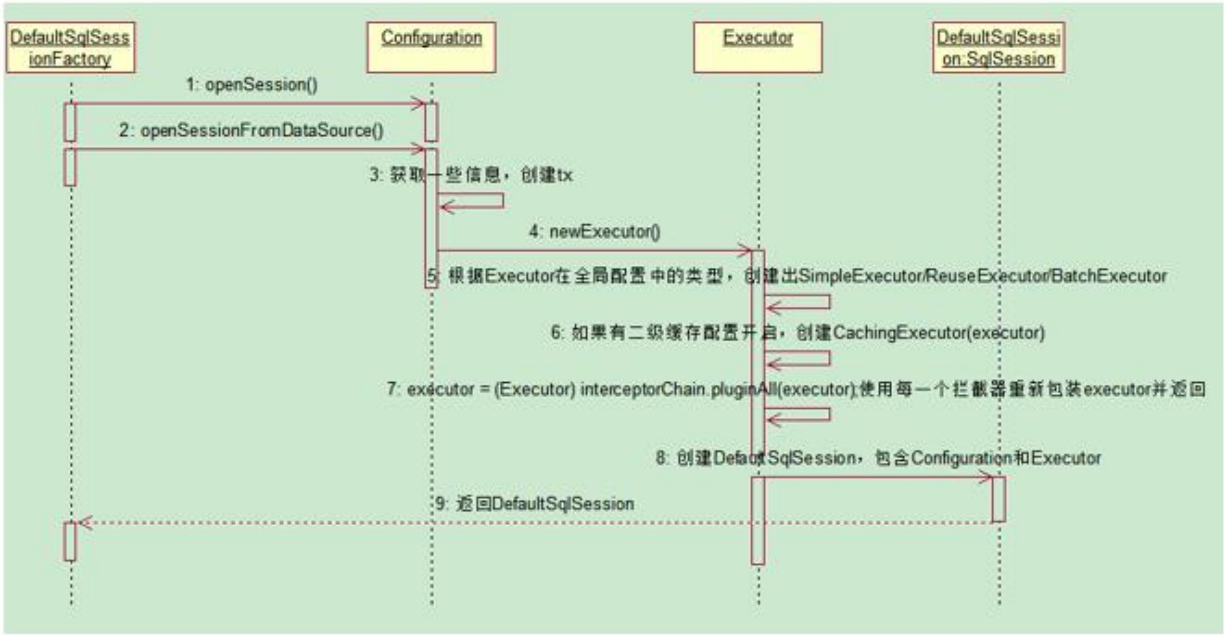
虽然可以明确的引用对象的属性名了,但如果要在if元素中测试传入的user参数,仍然要使用**_parameter**来引用传递进来的实际参数,因为传递进来的User对象的名字是不可考的。如果测试对象的属性,则直接引用属性名字就可以了。

八、工作原理

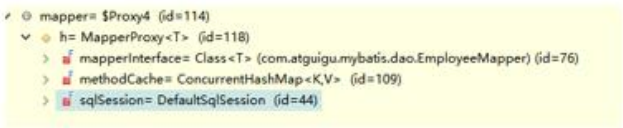


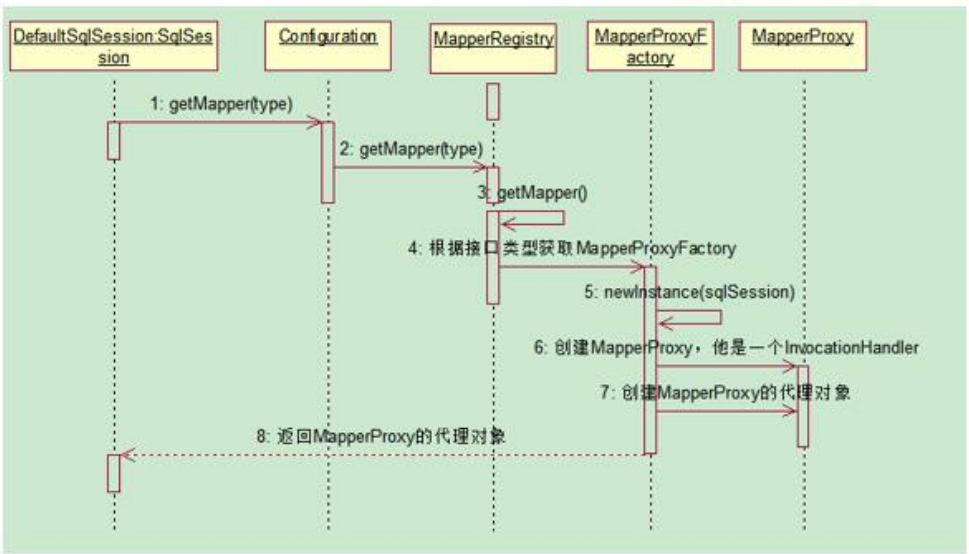


2、返回SqlSession的实现类DefaultSqlSession对象。
他里面包含了Executor和Configuration；
Executor会在这一步被创建

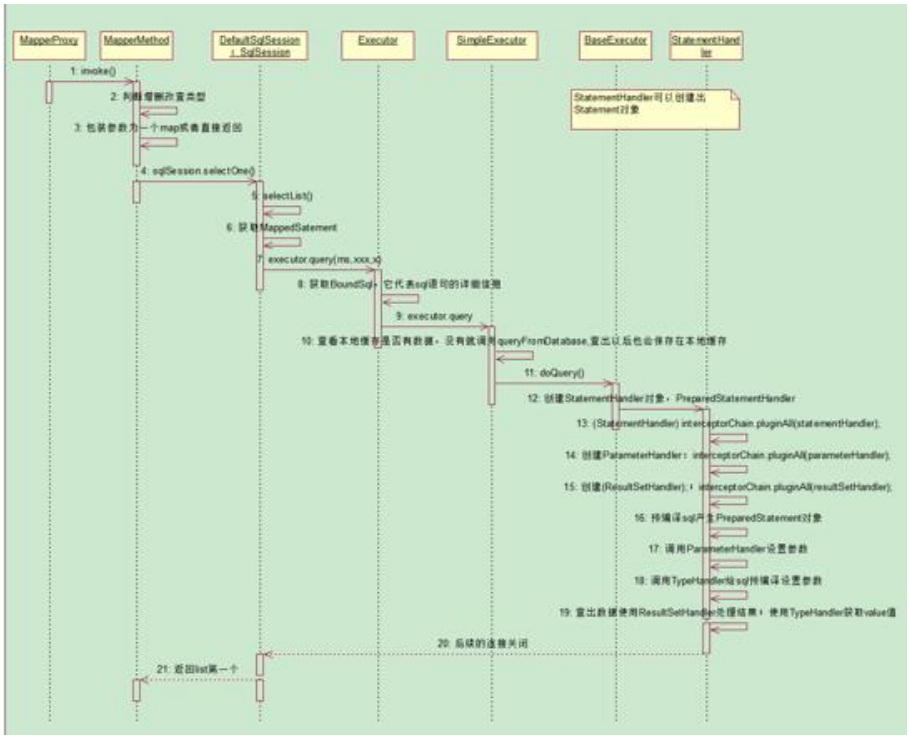


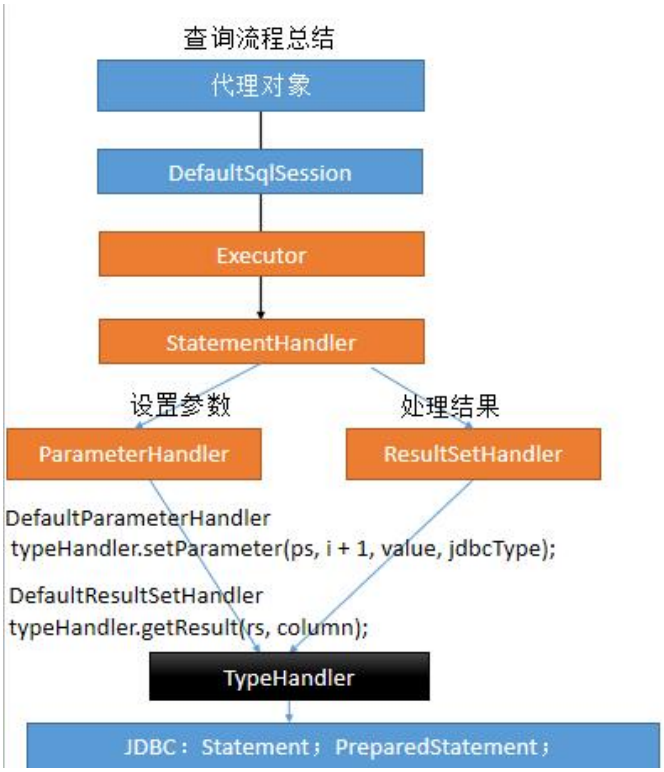
3、getMapper返回接口的代理对象
包含了SqlSession对象

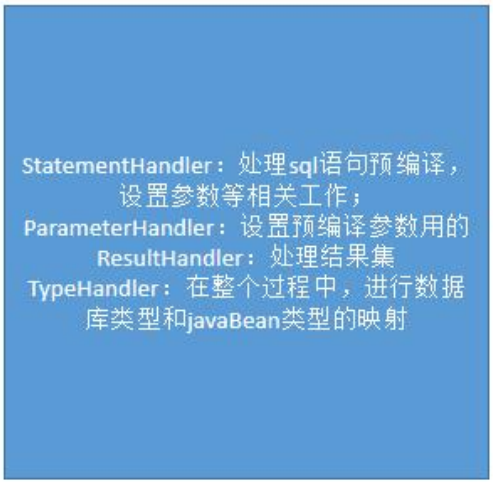




4、查询流程







九、插件开发

• MyBatis在四大对象的创建过程中，都会有插件进行介入。插件可以利用动态代理机制一层层的包装目标对象，而实现在目标对象执行目标方法之前进行拦截的效果。 • MyBatis 允许在已映射语句执行过程中的某一点进行 拦截调用。 • 默认情况下，MyBatis 允许使用插件来拦截的方法调 用包括： • Executor (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed) • ParameterHandler (getParameterObject, setParameters) •

ResultSetHandler (handleResultSets, handleOutputParameters) • StatementHandler (prepare, parameterize, batch, update, query)。

- 插件原理

  • 1）、按照插件注解声明，按照插件配置顺序调用插件plugin方法，生成被拦截对象的动态代理 • 2）、多个插件依次生成目标对象的代理对象，层层包裹，先声明的先包裹；形成代理链 • 3）、目标方法执行时依次从外到内执行插件的intercept方法。 • 4）、多个插件情况下，我们往往需要在某个插件中分离出目标对象。可以借助MyBatis提供的SystemMetaObject类来进行获取最后一层的h以及target属性的值

- interceptor接口

  • Intercept：拦截目标方法执行 • plugin：生成动态代理对象，可以使用MyBatis提 供的Plugin类的wrap方法 • setProperties：注入插件配置时设置的属性

- mybatis实用场景

  • 1）、PageHelper插件进行分页 • 2）、批量操作 • 3）、存储过程 • 4）、typeHandler处理枚举

  我们可以通过自定义TypeHandler的形式来在设置参数或者取出结果集的时候自定义参数封装策略。