MyBatis源码篇

一、 简介

1> MyBatis的核心组件

- **SqlSessionFactoryBuilder**——构造器
  根据配置信息或代码来生成SqlSessionFactory。
- **SqlSessionFactory**——工厂
  用来生成SqlSession。
- **SqlSession**——会话
  可以用来发送SQL去执行并返回结果，也可以获取Mapper的接口
- **SQL Mapper**
  它由一个Java接口和XML文件/注解构成。需要给出对应的SQL和映射规则。它负责发送SQL去执行，并返回结果。

二、SqlSession执行流程

2.1> Mapper的动态代理

我们自定义的Mapper接口想要发挥功能，必须有具体的实现类，在MyBatis中是通过为Mapper每个接口提供一个动态代理类来实现的。
整个过程主要有四个类，分别如下：

- **MapperRegistry**
  是Mapper接口及其对应的代理对象工厂的注册中心。
- **MapperProxyFactory**
  是MapperProxy的工厂类，主要方法就是包装了Java动态代理的Proxy.newProxyInstance()方法。
- **MapperProxy**
  是一个动态代理类，它实现了InvocationHandler接口。对于代理对象的调用都会被代理到InvocationHandler的invoke方法上。
- **MapperMethod**
  包含了具体增删改查方法的实现逻辑。

2.2> SqlSession中的对象

Mapper执行的过程是通过Executor、StatementHandler、ParameterHandler和ResultHandler来完成数据库操作和结果返回。

- **Executor**
  代表执行器，由它来调度StatementHander、ParameterHandler、ResultHandler等来执行对应的SQL。
- **StatementHander**
  它的作用是使用数据库的PreparedStatement来执行操作，起到承上启下的作用。
- **ParamenterHander**
  用于SQL对参数的处理。
- **ResultHander**
  进行最后数据集（ResultSet）的封装返回处理。

三、MyBatis四大组件解读

3.1> Executor

执行器是一个真正执行Java和数据库交互的类，一共有三种执行器。我们可以在MyBatis的配置文件中设置defaultExecutorType属性进行选择。

- **SIMPLE**
  SimpleExecutor：简易执行器，默认。
- **REUSE**
  ReuseExecutor：重用预处理语句执行器。

- BATCH

  BatchExecutor：重用语句和批量更新执行器。

3.1.1> 创建Executor

Java 复制代码

```
Configuration.java
/**
 * 创建Executor
 *
 * @param transaction
 * @param executorType  SIMPLE(默认), REUSE, BATCH
 * @return
 */
public Executor newExecutor(Transaction transaction, ExecutorType executorType) {
    executorType = executorType == null ? defaultExecutorType : executorType;
    executorType = executorType == null ? ExecutorType.SIMPLE : executorType;
    Executor executor;
    if (ExecutorType.BATCH == executorType) {
        executor = new BatchExecutor(this, transaction);
    } else if (ExecutorType.REUSE == executorType) {
        executor = new ReuseExecutor(this, transaction);
    } else {
        executor = new SimpleExecutor(this, transaction);
    }
    if (cacheEnabled) {
        executor = new CachingExecutor(executor);
    }
    // 在executor完成创建之后，会通过interceptorChain来添加插件
    executor = (Executor) interceptorChain.pluginAll(executor);
    return executor;
}
```


创建Executor的具体逻辑在Configure类中，可以看到，在Executor创建完成之后，会通过interceptorChain来添加插件，通过代理到方式，在调度真实的Executor方法之前执行插件代码来完成功能。

Executor的具体执行逻辑

我们通过SimpleExecutor来看一下Executor的具体执行逻辑：

> 第一步：构建**StatementHandler**
> 调用Configuration的newStatementHandler方法来构建StatementHandler。
> 第二步：构建**Statement**
> 使用prepareStatement方法，对SQL编译并对参数进行初始化；在prepareStatement方法中，调用了StatementHandler的prepared进行了预编译和基础设置，然后通过StatementHandler的parameterize来设置参数并执行。
> 第三步：构建**ResultHandler**
> Statement作为StatementHandler的入参来执行，并把结果传递给ResultHandler。

3.2> StatementHandler

3.3> ParameterHandler

3.4> ResultSetHandler

DefaultResultSetHandler

四、常用类

- DefaultSqlSession
  getMapper、selectOne、selectList
- Configuration
  所有MyBatis的配置信息都会加载到这个类中，其中提供了getMappedStatement等方法获取响应配置。
- MapperRegistry
- MapperProxyFactory
  每个Mapper.java/xml 对应一个MapperProxyFactory。用于调用**newInstance方法**，生成MapperProxy代理对象
- MapperProxy
  实现InvocationHandler（JDK动态代理），初始化MapperMethod，调用它的**execute方法**执行sql指令。
- MapperMethod
  - SqlCommand
    1> 通过**statementId**（mapperInterface.getName() + "." + methodName）获得

MappedStatement。

2> 存储MappedStatement的**id**和**SqlCommandType**（枚举类型：UNKNOWN, INSERT, UPDATE, DELETE, SELECT, FLUSH;）

- MethodSignature

  1> 存储解析后的一些方法签名，比如：returnsMany、returnsMap、returnType等等。

  2> 初始化**ParamNameResolver**实例。

- ParamNameResolver

  该类用于解析方法入参，通过**getNamedParams**方法，获得参数名称与值的对应map。

- MappedStatement

  用于保存**映射器的一个节点**（select|insert|delete|update）。

  包括许多我们配置的SQL、SQL的id、缓存信息、resultMap、parameterType、resultType、languageDriver等重要配置内容。

- Executor

  接口类，通用执行器。提供update、query、commit、rollback、getTransaction、close等方法

- CachingExecutor

  基于缓存实现的执行器。

- SqlSource

  接口类，只提供一个getBoundSql方法，返回BoundSql实例。

- BoundSql

  存储着sql，sql传参等。

- PropertyTokenizer

  属性标记器，通过构造函数，将属性解析为name、indexedName、index、children。

  举例：

Plain Text 复制代码

```
例1: 参数: fullname=user[1].linkman.name
    children=linkman.name
    indexedName=user[1]
    name=user
    index=1
例2: 参数: fullname=user
    children=null
    indexedName=user
    name=user
    index=null
```

- BaseExecutor

  提供执行查询的query方法和queryFromDatabase方法。

- SimpleExecutor

提供doQuery、doUpdate等真正的执行方法。

prepareStatement方法，是执行sql的关键方法。使用JDBC对**sql进行预编译**以及**设置对应的参数**。

- StatementHandler

  接口类，提供基础的语句执行方法，例如：prepare、batch、update、query、getBoundSql、getParameterHandler等。

  其中prepare方法，用于执行sql的预编译。
- BaseStatementHandler

  抽象类，实现StatementHandler接口。实现了prepare方法，该方法里面调用了instantiateStatement抽象方法。由其他继承该抽象类的子类进行实现。用于生成sql的预编译语句。
- PreparedStatementHandler

  StatementHandler的实现类。实现了BaseStatementHandler中的抽象方法instantiateStatement。该方法用于生成sql的预编译语句。
- ParameterHandler

  接口类。提供setParameters方法为预编译sql语句设置入参。
- DefaultParameterHandler

  ParameterHandler的实现类。实现了setParameters方法。针对预处理语句，设置入参。
- ResultSetHandler

  处理查询后的结果。
- ResultSetWrapper

  结果集包装类。

  存储查询后的结果集ResultSet，并通过构造方法初始化结果集的列名集合（columnNames），对应的实体属性类型集合（classNames）和对应的JDBC类型集合（jdbcTypes）。
- TypeHandlerRegistry

  TypeHandler的注册器。默认注册了很多基础类型的TypeHandler，也支持调用register方法注册自定义的TypeHandler。
- ObjectFactory

  MyBatis使用ObjectFactory对象工厂来创建所有的需要新建的对象。提供create方法来创建对象
- DefaultObjectFactory

  默认的对象工厂，使用instantiateClass方法利用反射创建对象。
- ResultHandler

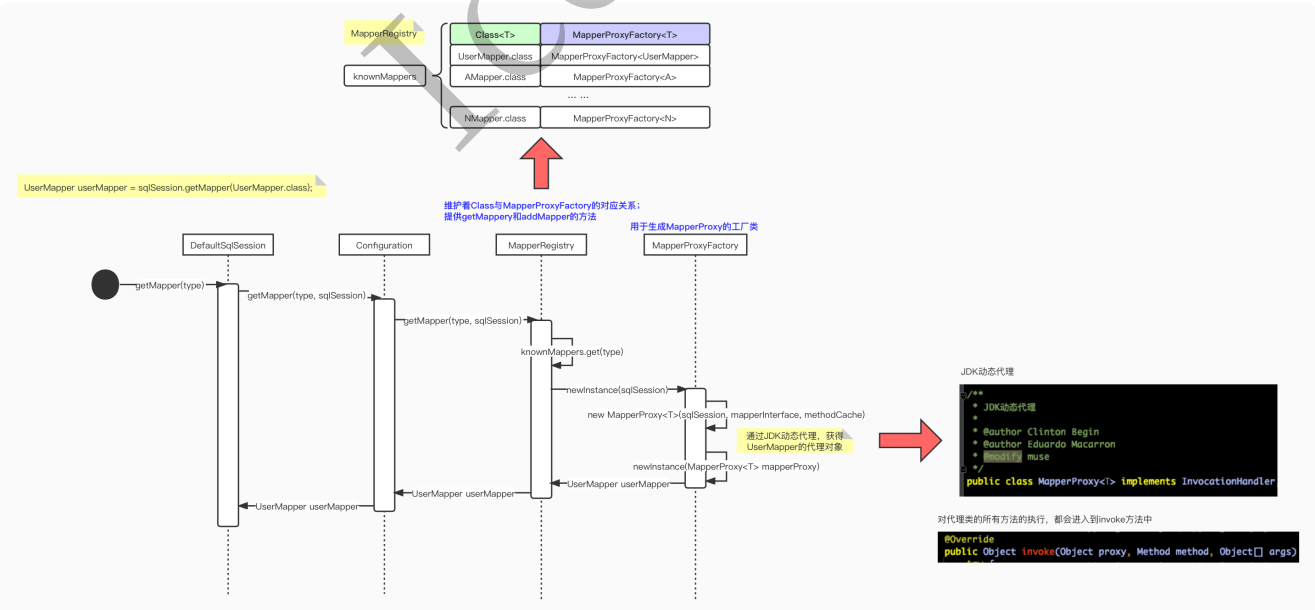
  接口类。查询结果处理类。

  该接口只有一个方法handleResult，用于处理查询结果，并存放于ResultHandler实例中。
- DefaultResultHandler

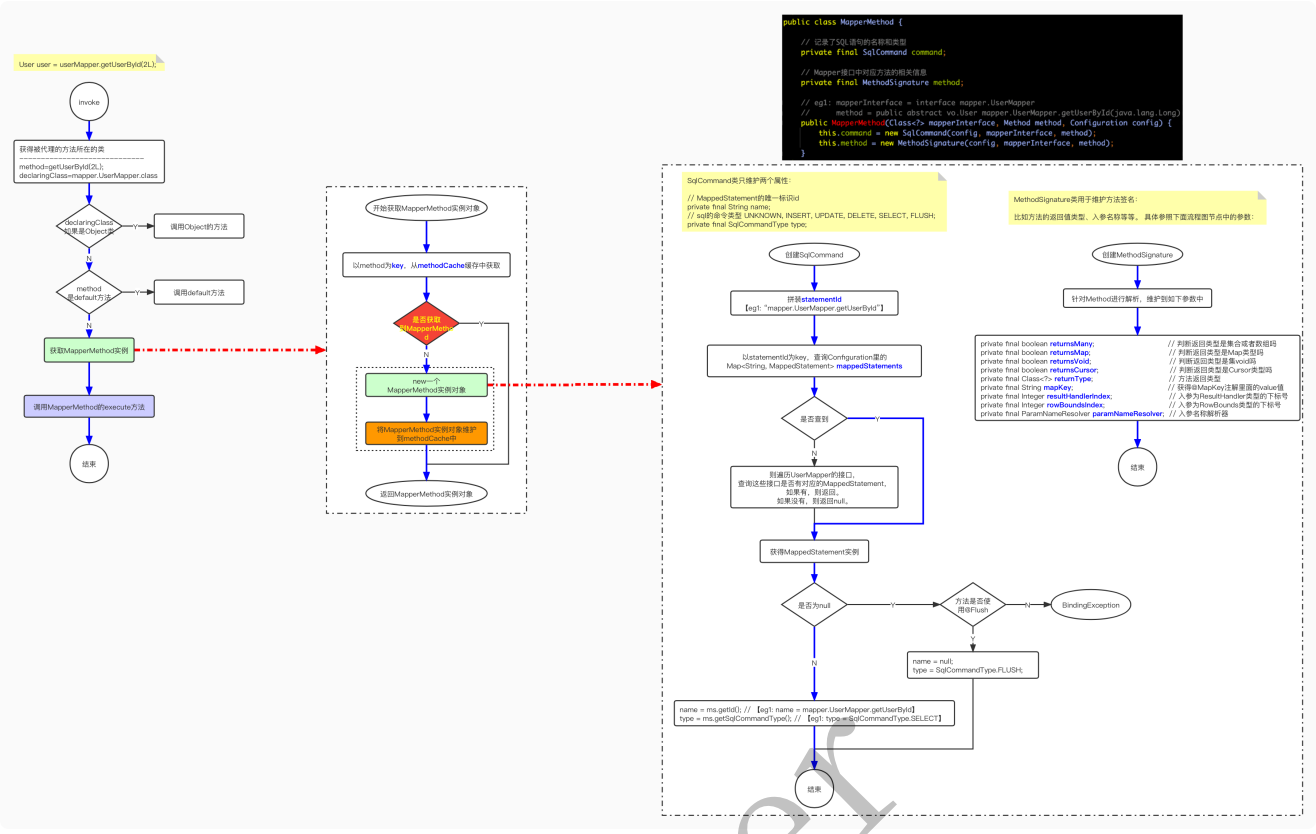
  ResultHandler的实现类，里面有属性List

- DefaultResultSetHandler
  处理数据库操作返回的结果集。**applyAutomaticMappings**方法将查询的结果集赋值给POJO实体对象。
- MetaClass
  元数据类。
- MetaObject
  元对象。存储着原始对象**originalObject**、对象工厂**objectFactory**、加工后的对象**objectWrapper**、加工对象工厂**objectWrapperFactory**、反射器工厂**reflectorFactory**。
- ReflectorFactory
  反射器工厂类
- Reflector
  反射器
- AutoMappingBehavior
  枚举类，自动映射配置枚举类。NONE、PARTIAL、FULL
- ResultContext
  结果上下文
- DefaultResultContext
  默认结果上下文实现类

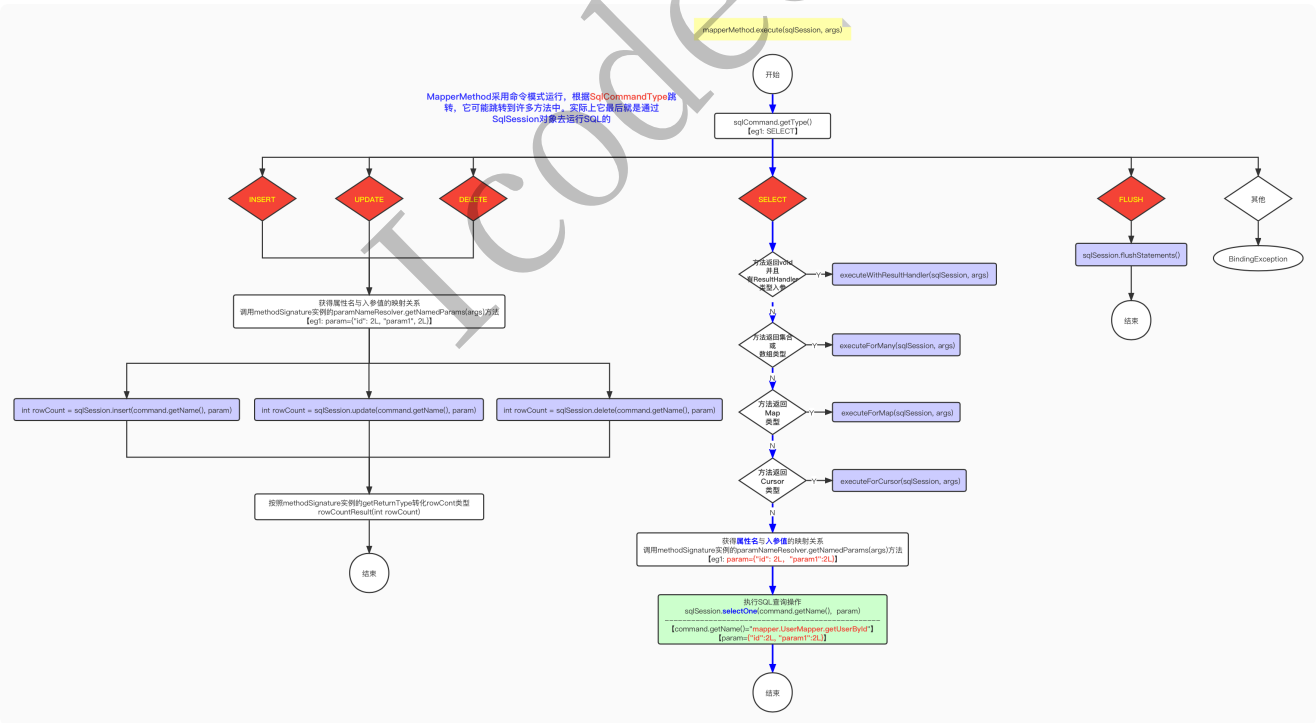
- 阶段1：获得 `Mapper动态代理` 阶段



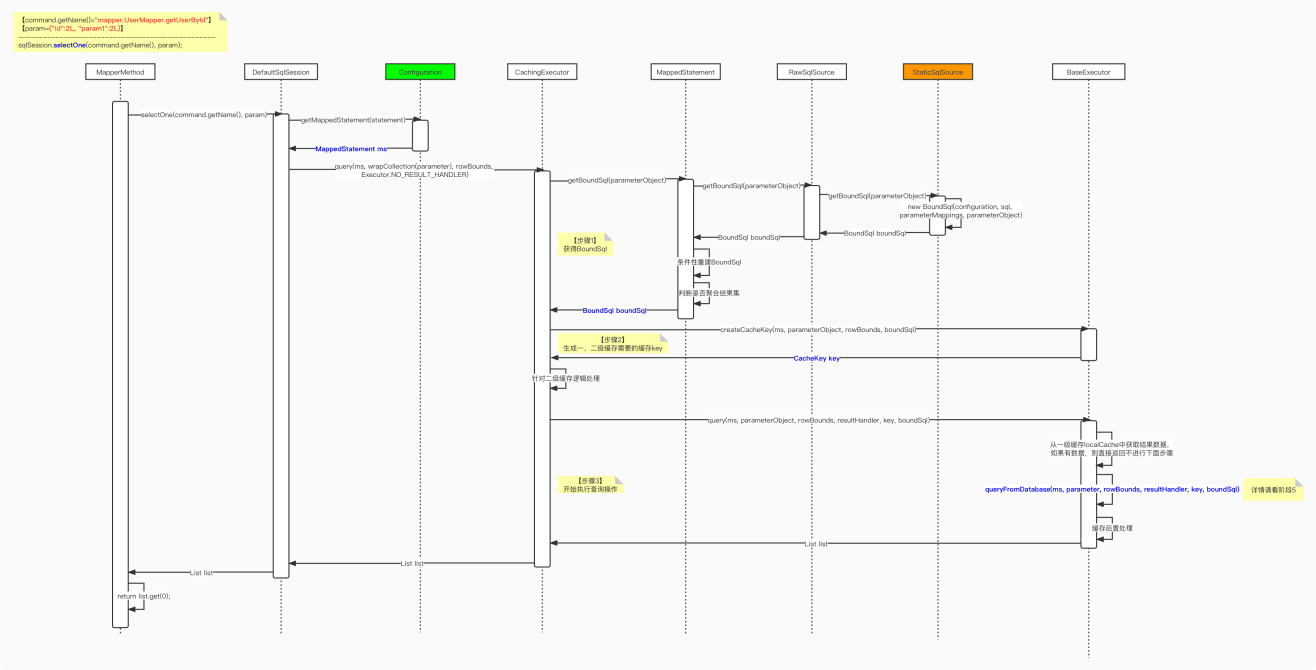
- 阶段2：获得 `MapperMethod` 对象



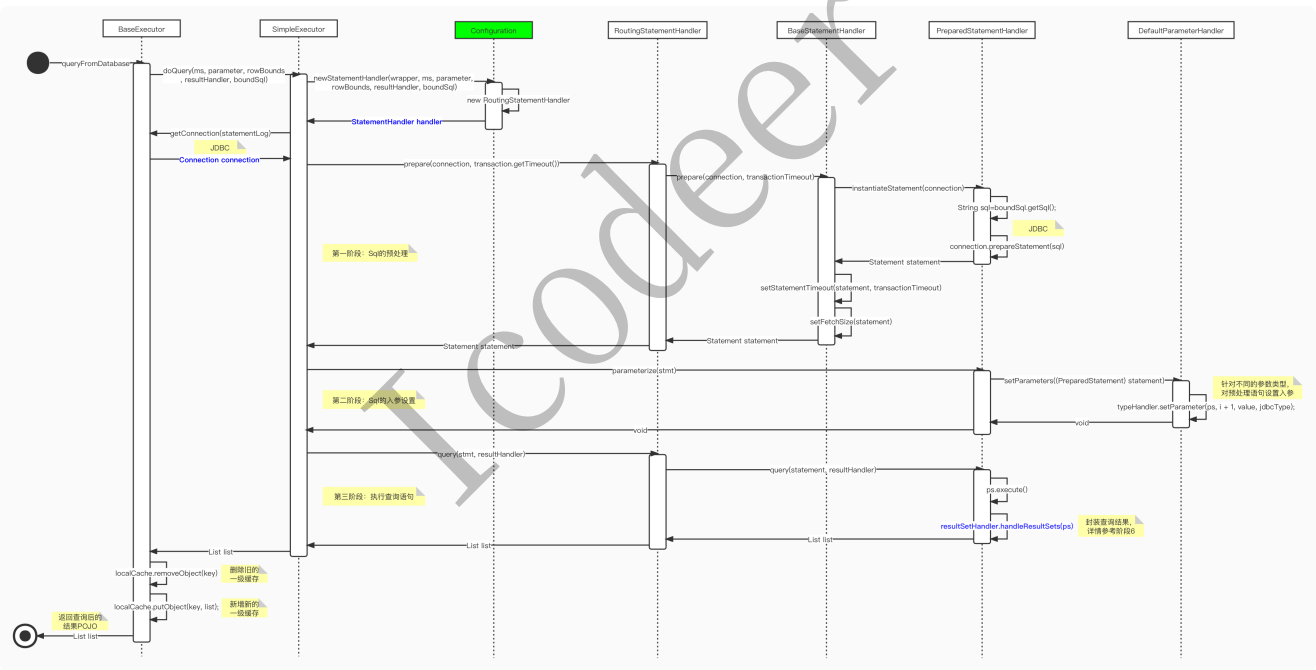
- 阶段3：根据 SQL指令跳转 执行语句



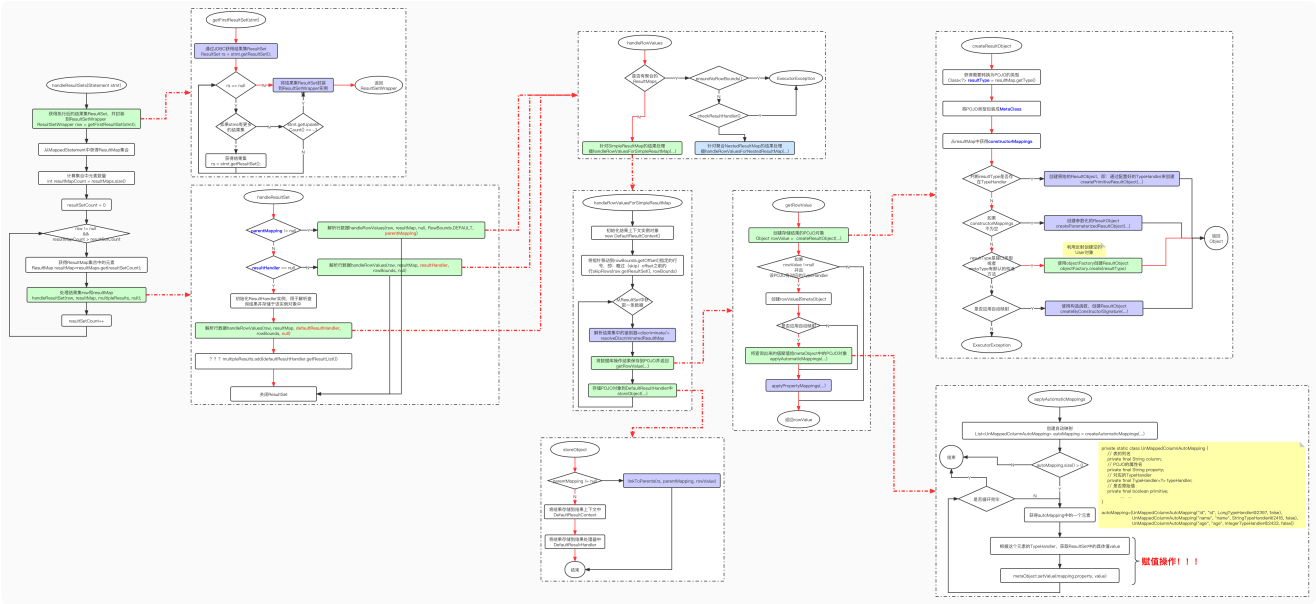
- 阶段4：查询前的 缓存处理



- 阶段5：`执行DB查询` 操作



- 阶段6：针对ResultSet结果集转换为POJO



吾尝终日而思矣，不如须臾之所学也；
吾尝跂而望矣，不如登高之博见也。
登高而招，臂非加长也，而见者远；
顺风而呼，声非加疾也，而闻者彰。
假舆马者，非利足也，而致千里；
假舟楫者，非能水也，而绝江河。
君子生非异也，善假于物也。
------ 摘自《劝学》

愿本文可以成为大家的“山”、“风”、“马”、“舟”，助大家在技术之路上乘风破浪，大展宏图~~

同时，也欢迎大家关注我的公众号“爪哇缪斯” ~\(^o^)/~ 「干货分享，每天更新」