Mybatis高级（二）

八点正式上课

T A H N K Y O U F O R W A T C H I N G

主讲老师Lison：525765982

课程咨询依娜老师：2470523467

目 录

CONTENTS

mybatis核心流程三大阶段

初始化阶段

读取XML配置文件和注解中的配置信息，创建配置对象，并完成各个模块的初始化的工作；

代理阶段

封装iBatis的编程模型，使用mapper接口开发的初始化工作；

数据读写阶段

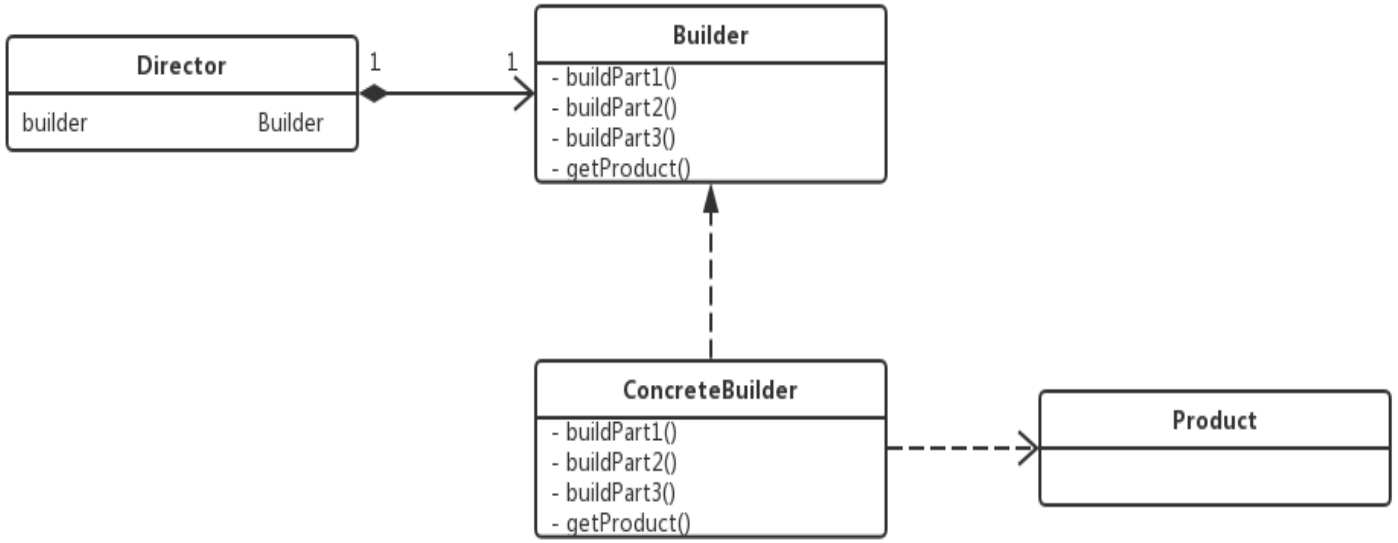
通过SqlSession完成SQL的解析，参数的映射、SQL的执行、结果的解析过程；

目 录

CONTENTS

Mybatis的初始化 建造者模式

- 建造者模式（Builder Pattern）使用多个简单的对象一步一步构建成一个复杂的对象。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。

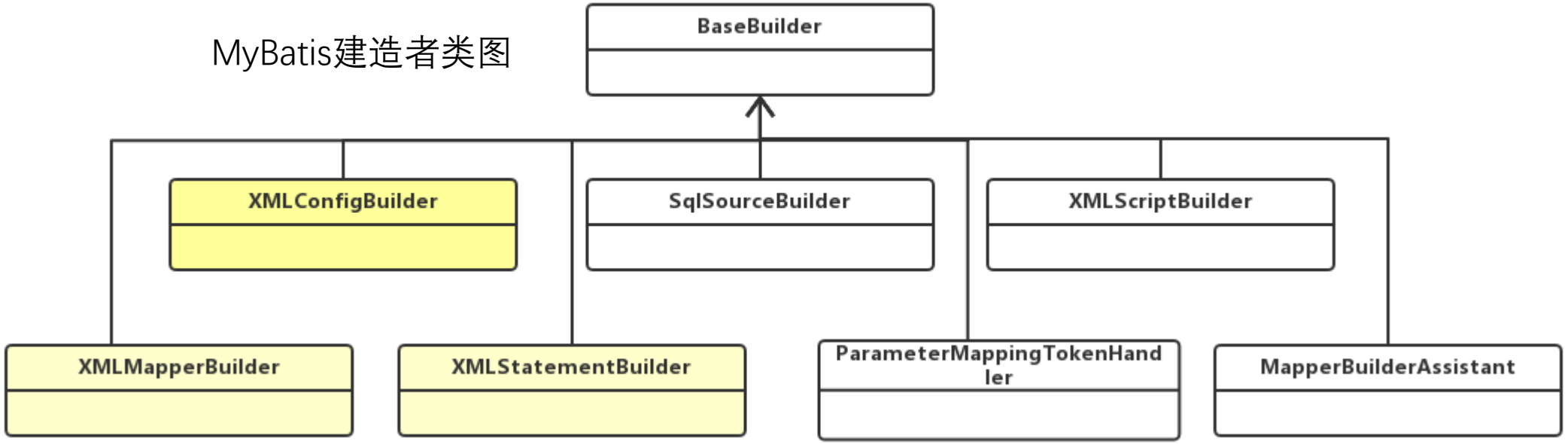


- ✓ **Builder**：给出一个抽象接口，以规范产品对象的各个组成成分的建造。这个接口规定要实现复杂对象的哪些部分的创建，并不涉及具体的对象部件的创建；
- ✓ **ConcreteBuilder**：实现Builder接口，针对不同的商业逻辑，具体化复杂对象的各部分的创建。 在建造过程完成后，提供产品的实例；
- ✓ **Director**：调用具体建造者来创建复杂对象的各个部分，在指导者中不涉及具体产品的信息，只负责保证对象各部分完整创建或按某种顺序创建；
- ✓ **Product**：要创建的复杂对象

建造者模式 使用场景

- ✓ 需要生成的对象具有复杂的内部结构，实例化对象时要屏蔽掉对象内部的细节，让上层代码与复杂对象的实例化过程解耦，可以使用建造者模式；简而言之，如果“遇到多个构造器参数时要考虑用构建器”；
- ✓ 一个对象的实例化是依赖各个组件的产生以及装配顺序，关注的是一步一步地组装出目标对象，可以使用建造器模式；

MyBatis建造者类图

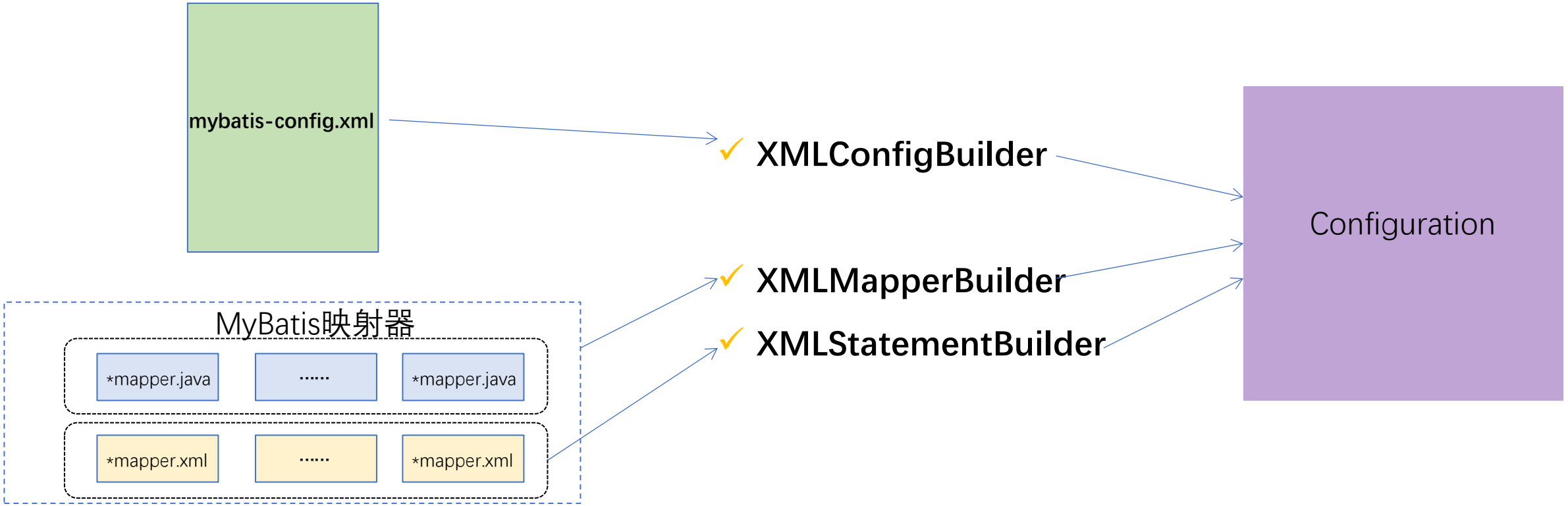


- ✓ **XMLConfigBuilder：** 主要负责解析mybatis-config.xml；
- ✓ **XMLMapperBuilder：** 主要负责解析映射配置文件；
- ✓ **XMLStatementBuilder：** 主要负责解析映射配置文件中的SQL节点；

与工厂模式区别

- **对象复杂度**
  - ✓ 建造者建造的对象更加复杂，是一个复合产品，它由各个部件复合而成，部件不同产品对象不同，生成的产品粒度细；
  - ✓ 在工厂方法模式里，我们关注的是一个产品整体，无须关心产品的各部分是如何创建出来的；
- **客户端参与程度**
  - ✓ 建造者模式，导演对象参与了产品的创建，决定了产品的类型和内容，参与度高；适合实例化对象时属性变化频繁的场景；
  - ✓ 工厂模式，客户端对产品的创建过程参与度低，对象实例化时属性值相对比较固定；

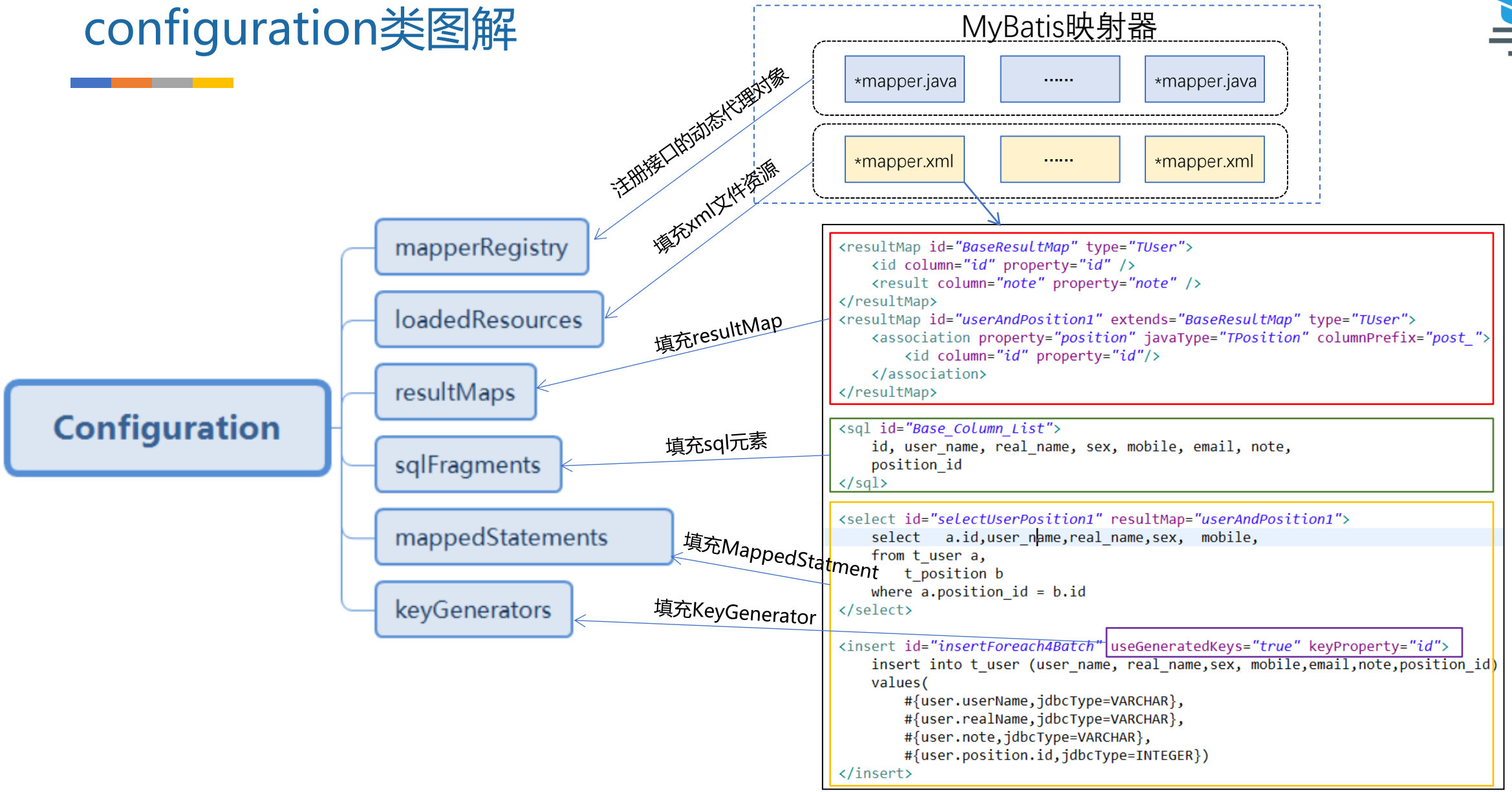
Mybatis的初始化



映射器的关键类

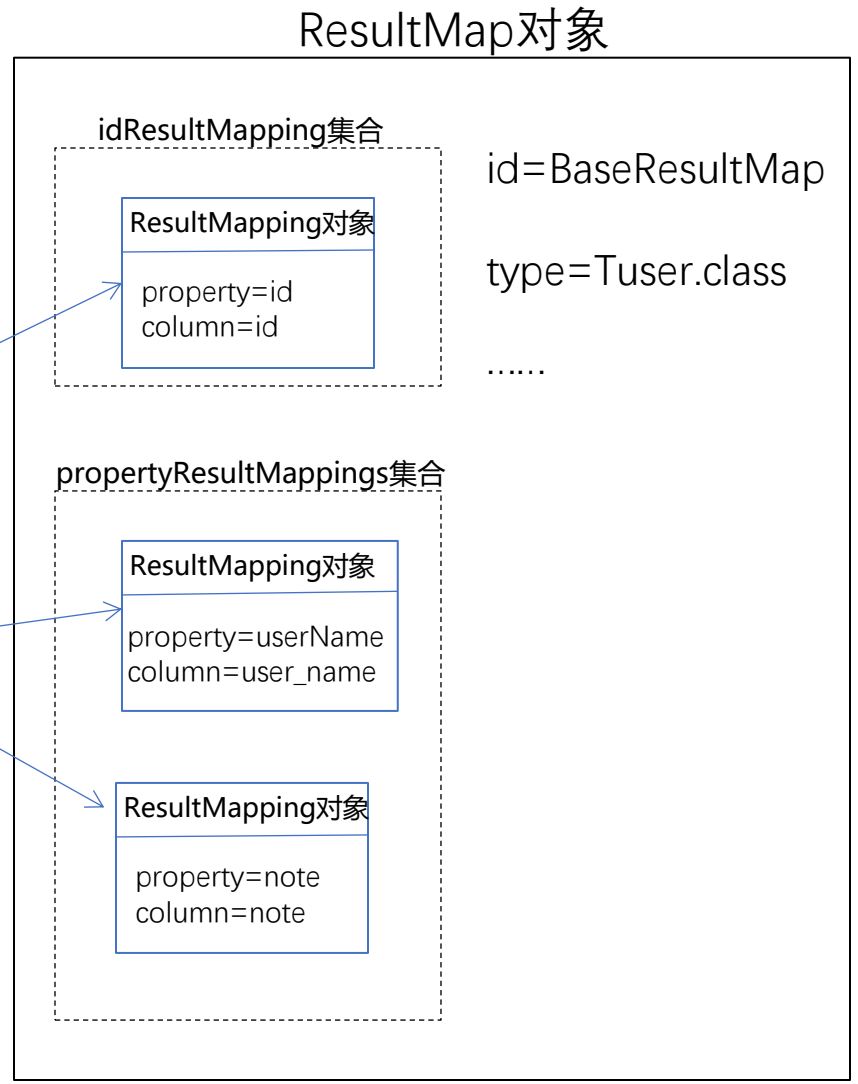
- **Configuration ：** Mybatis启动初始化的核心就是将所有xml配置文件信息加载到Configuration对象中， Configuration是单例的，生命周期是应用级的；
- **MapperRegistry：**mapper接口动态代理工厂类的注册中心。在MyBatis中，通过mapperProxy实现InvocationHandler接口，MapperProxyFactory用于生成动态代理的实例对象；
- **ResultMap：**用于解析mapper.xml文件中的resultMap节点，使用ResultMapping来封装id，result等子元素；
- **MappedStatement：**用于存储mapper.xml文件中的select、insert、update和delete节点，同时还包含了这些节点的很多重要属性；
- **SqlSource：**mapper.xml文件中的sql语句会被解析成SqlSource对象，经过解析SqlSource包含的语句最终仅仅包含？占位符，可以直接提交给数据库执行；

configuration类图解

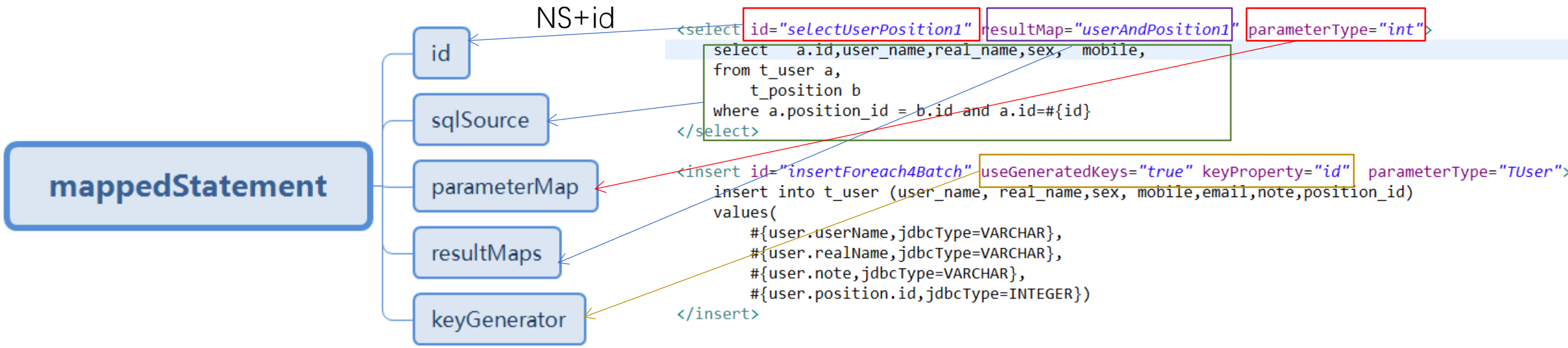


ResultMap图解

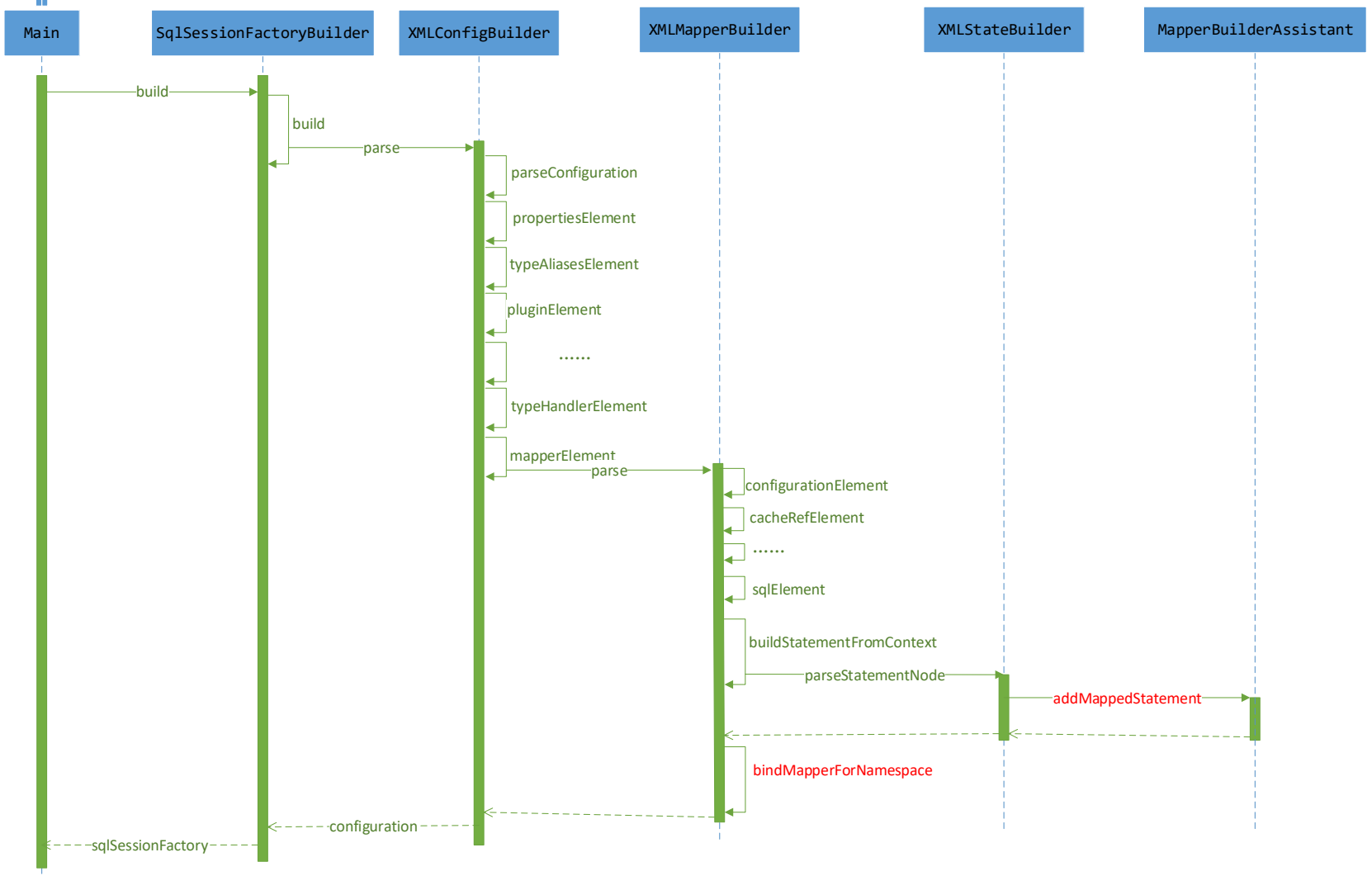
```
<resultMap id="BaseResultMap" type="TUser">
    <id column="id" property="id" />
    <result column="user_name" property="userName" />
    <result column="note" property="note" />
</resultMap>
```



mappedStatment图解



MyBatis初始化过程



目 录

CONTENTS

第三次作业

SqlSession对数据库执行一次查询操作的时序图？描述主要流程

为什么使用mapper接口就能操作数据库？

配置文件解读 +动态代理的增强

解读MapperMethod

快速入门

```
@Test
// 快速入门
public void quickStart() throws IOException {
    // 2.获取sqlSession
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 3.获取对应mapper
    TUserMapper mapper = sqlSession.getMapper(TUserMapper.class);
    // 4.执行查询语句并返回结果
    TUser user = mapper.selectByPrimaryKey(1);
    System.out.println(user.toString());
}
```

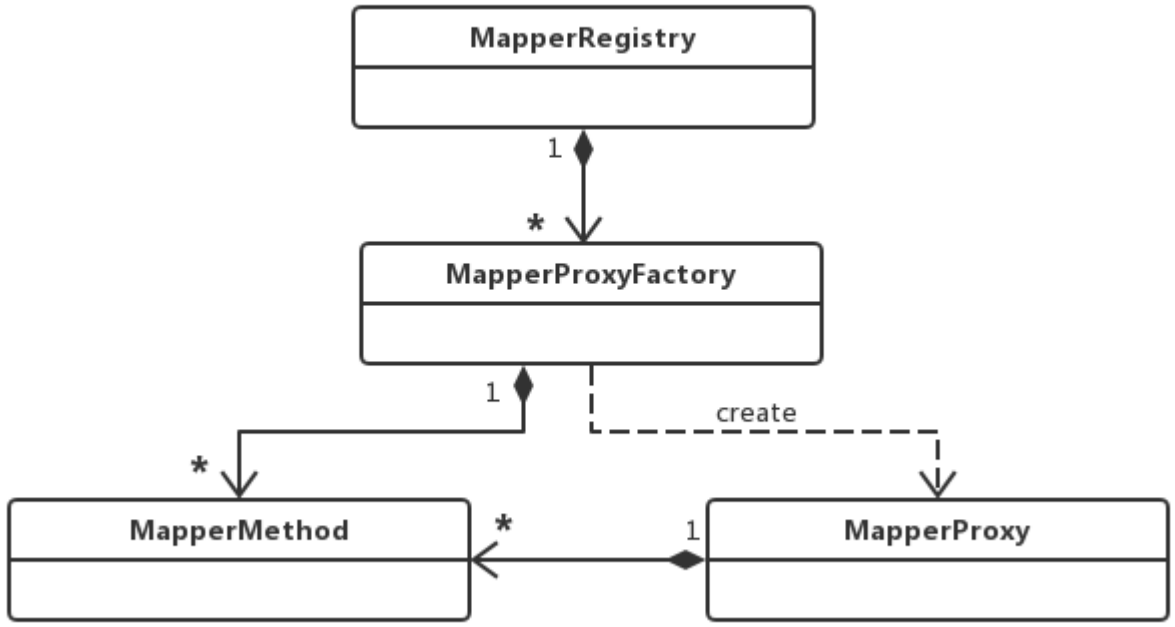
追本溯源

```
@Test
// 快速入门 本质分析
public void originalOperation() throws IOException {
    // 2.获取sqlSession
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 3.执行查询语句并返回结果
    TUser user = sqlSession.selectOne("com.enjoylearning.mybatis.mapper."
                                      + "TUserMapper.selectByPrimaryKey", 1);
    System.out.println(user.toString());
}
```

翻译

- ✓ 找到session中对应的方法执行；
- ✓ 找到命名空间和方法名
- ✓ 传递参数

binding模块分析



- **MapperRegistry ：** mapper接口和对应的代理对象工厂的注册中心；
- **MapperProxyFactory：**用于生成mapper接口动态代理的实例对象；
- **MapperProxy：**实现了InvocationHandler接口，它是增强mapper接口的实现；
- **MapperMethod：**封装了Mapper接口中对应方法的信息，以及对应的sql语句的信息；它是mapper接口与映射配置文件中sql语句的桥梁；

让我们从XMLMapperBuilder.bindMapperForNamespace()开始入手吧！

解读MapperMethod

- **MapperMethod：**封装了Mapper接口中对应方法的信息，以及对应的sql语句的信息；它是mapper接口与映射配置文件中sql语句的桥梁；MapperMethod对象不记录任何状态信息，所以它可以在多个代理对象之间共享；
  - ✓ **SqlCommand ：** 从configuration中获取方法的命名空间.方法名以及SQL语句的类型；
  - ✓ **MethodSignature：**封装mapper接口方法的相关信息（入参，返回类型）；
  - ✓ **ParamNameResolver：** 解析mapper接口方法中的入参；

翻译过程

- ✓ 找到session中对应的方法执行；← MapperMethod.**SqlCommand.** type +MapperMethod.MethodSignature.returnType
- ✓ 找到命名空间和方法名 ← MapperMethod.**SqlCommand.** name
- ✓ 传递参数 ← MapperMethod.MethodSignature.paramNameResolver

目 录

CONTENTS

MyBatis核心流程分析

MyBatis核心流程分析

配置加载阶段

建造者模式

mybatis初始化过程

binding模块分析

binding模块分析

Mybatis的接口层

策略模式

sqlSession相关的类

核心组件Excutor

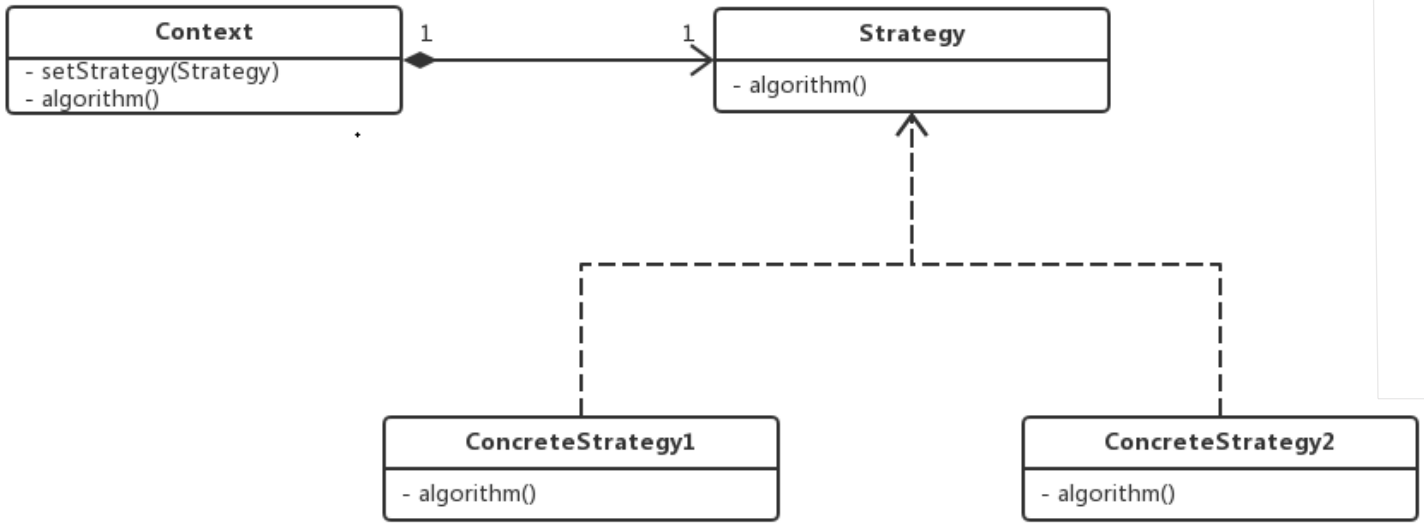
模板模式

Excutor组件分析

Excutor的三个小弟

创建SqlSession　策略模式

- **策略模式（Strategy Pattern）**策略模式定义了一系列的算法，并将每一个算法封装起来，而且使他们可以相互替换，让算法独立于使用它的客户而独立变化。



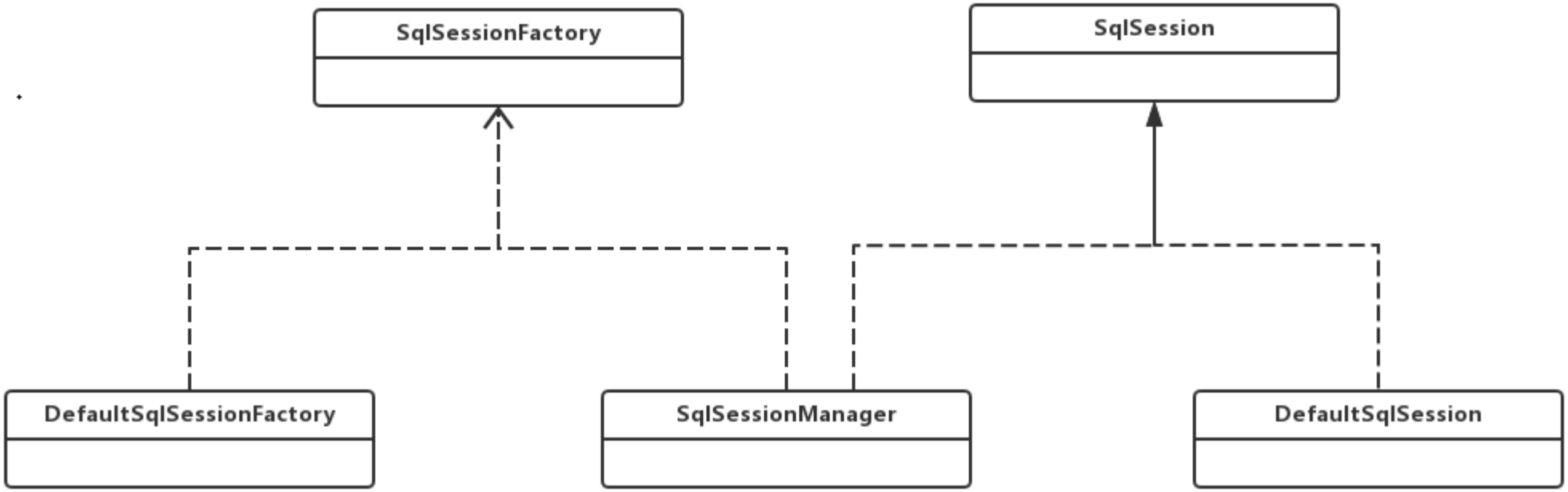
策略模式的使用场景：

- ✓ 针对同一类型问题的多种处理方式，仅仅是具体行为有差别时；
- ✓ 出现同一抽象类有多个子类，而又需要使用 if-else 或者 switch-case 来选择具体子类时。

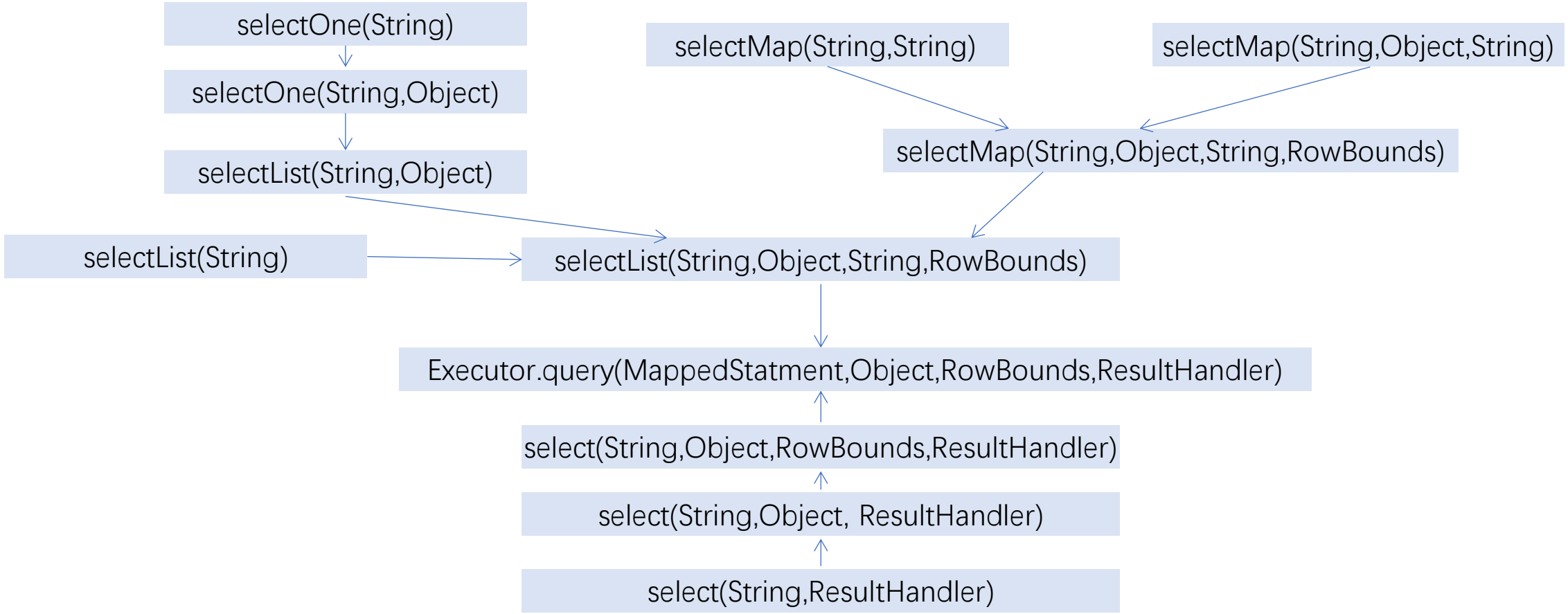
- ✓ **Context：**算法调用者，使用setStrategy方法灵活的选择策略（strategy）；
- ✓ **Strategy：**算法的统一接口；
- ✓ **ConcreteStrategy：**算法的具体实现；

SqlSession相关类UML

- **SqlSession**是MyBaits对外提供的最关键的核心接口，通过它可以执行数据库读写命令、获取映射器、管理事务等；



SqlSession查询接口嵌套关系

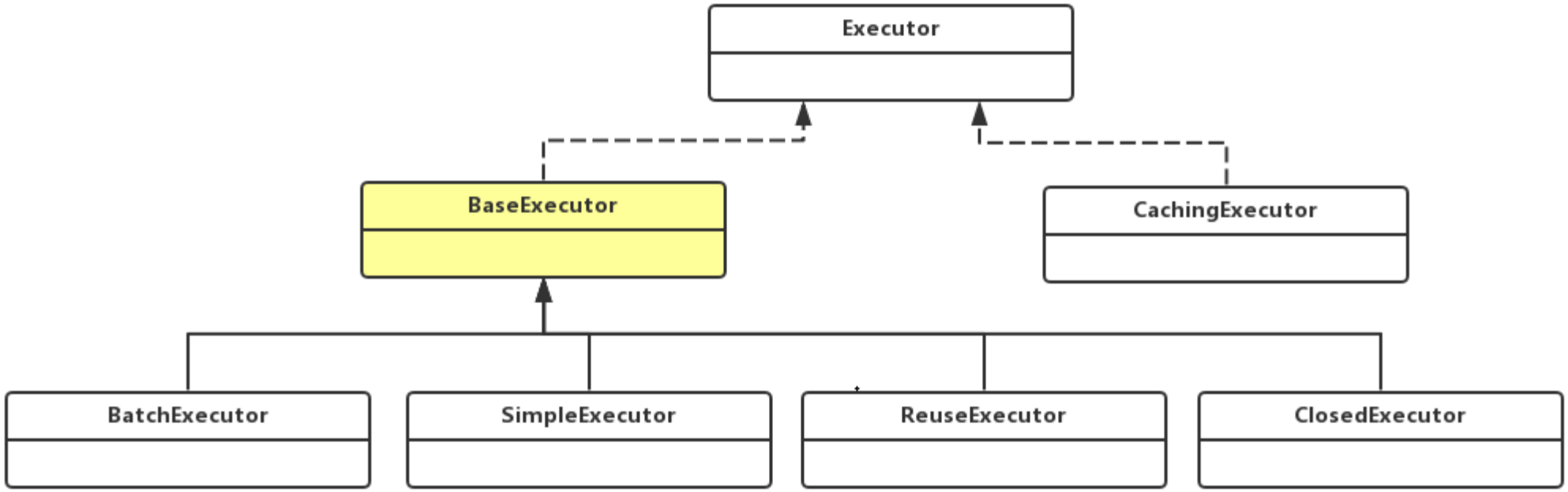


目 录

CONTENTS

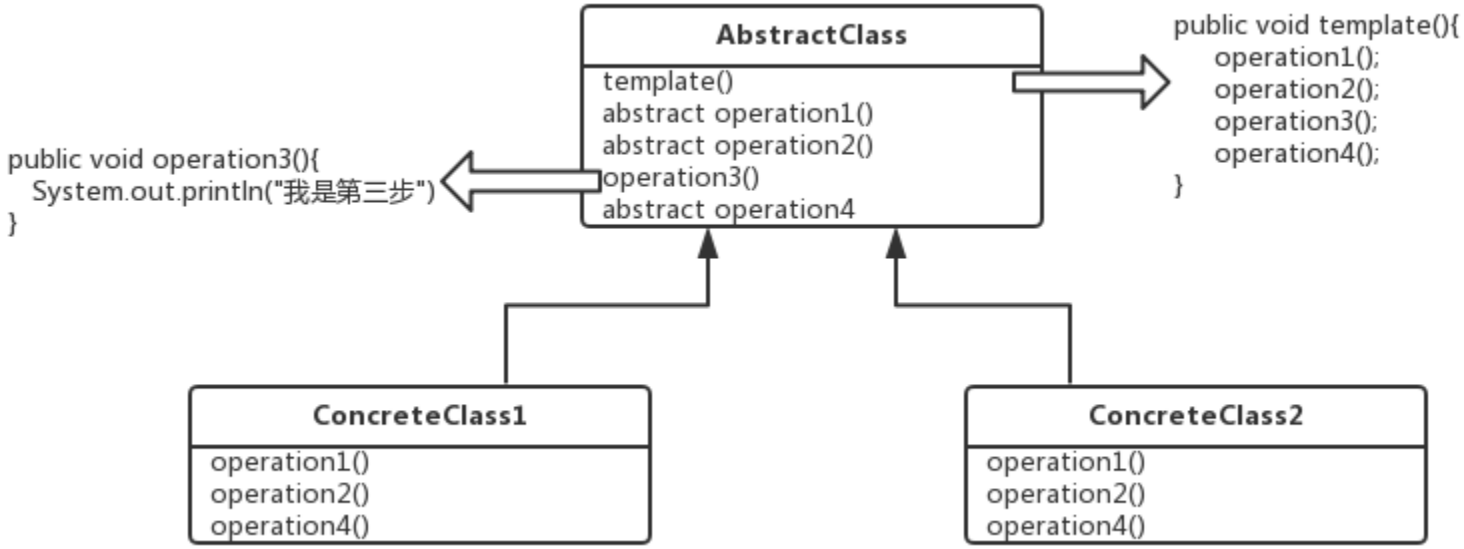
Executor组件分析

- **Executor**是MyBaits核心接口之一，定义了数据库操作最基本的方法，SqlSession的功能都是基于它来实现的；



模板模式

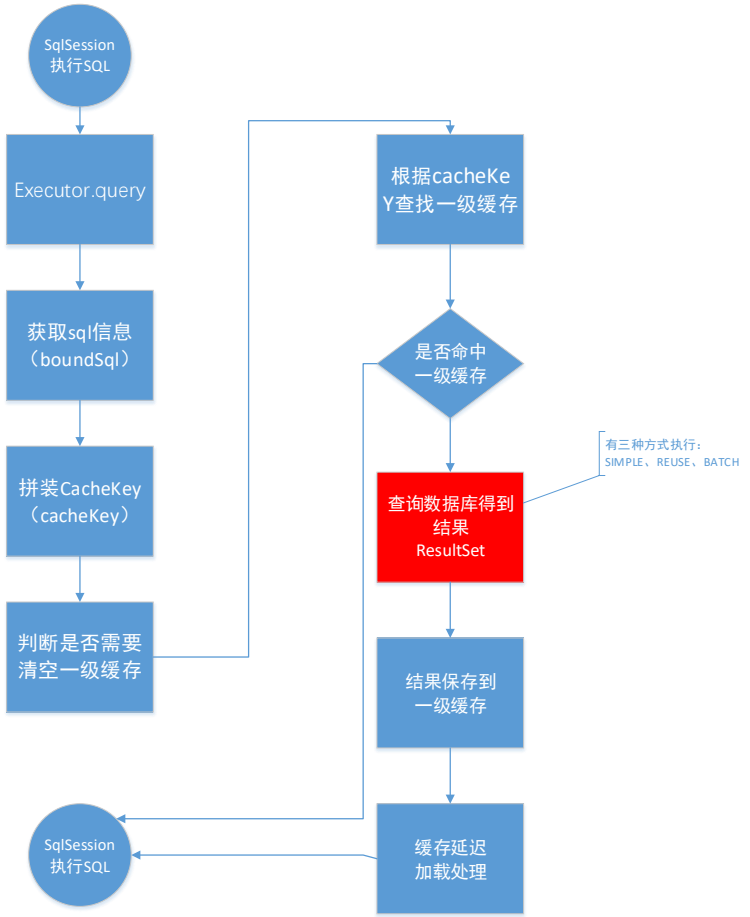
- **模板模式（Template Pattern）**：一个抽象类公开定义了执行它的方法的方式/模板。它的子类可以按需要重写方法实现，但调用将以抽象类中定义的方式进行。定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定实现；



BaseExecutor代码分析 模板模式应用场景

遇到由一系列步骤构成的过程需要执行，这个过程从高层次上看是相同的，但是有些步骤的实现可能不同，这个时候就需要考虑用模板模式了。比如：Executor查询操作流程:

- **BaseExecutor**：抽象类，实现了executor接口的大部分方法，主要提供了缓存管理和事务管理的能力，其他子类需要实现的抽象方法为：doUpdate,doQuery等方法；



Executor的三个实现类解读

- SimpleExecutor：默认配置，使用PrepareStatement对象访问数据库，每次访问都要创建新的PrepareStatement对象；
- ReuseExecutor：使用预编译PrepareStatement对象访问数据库，访问时，会重用缓存中的statement对象；
- BatchExecutor：实现批量执行多条SQL语句的能力；

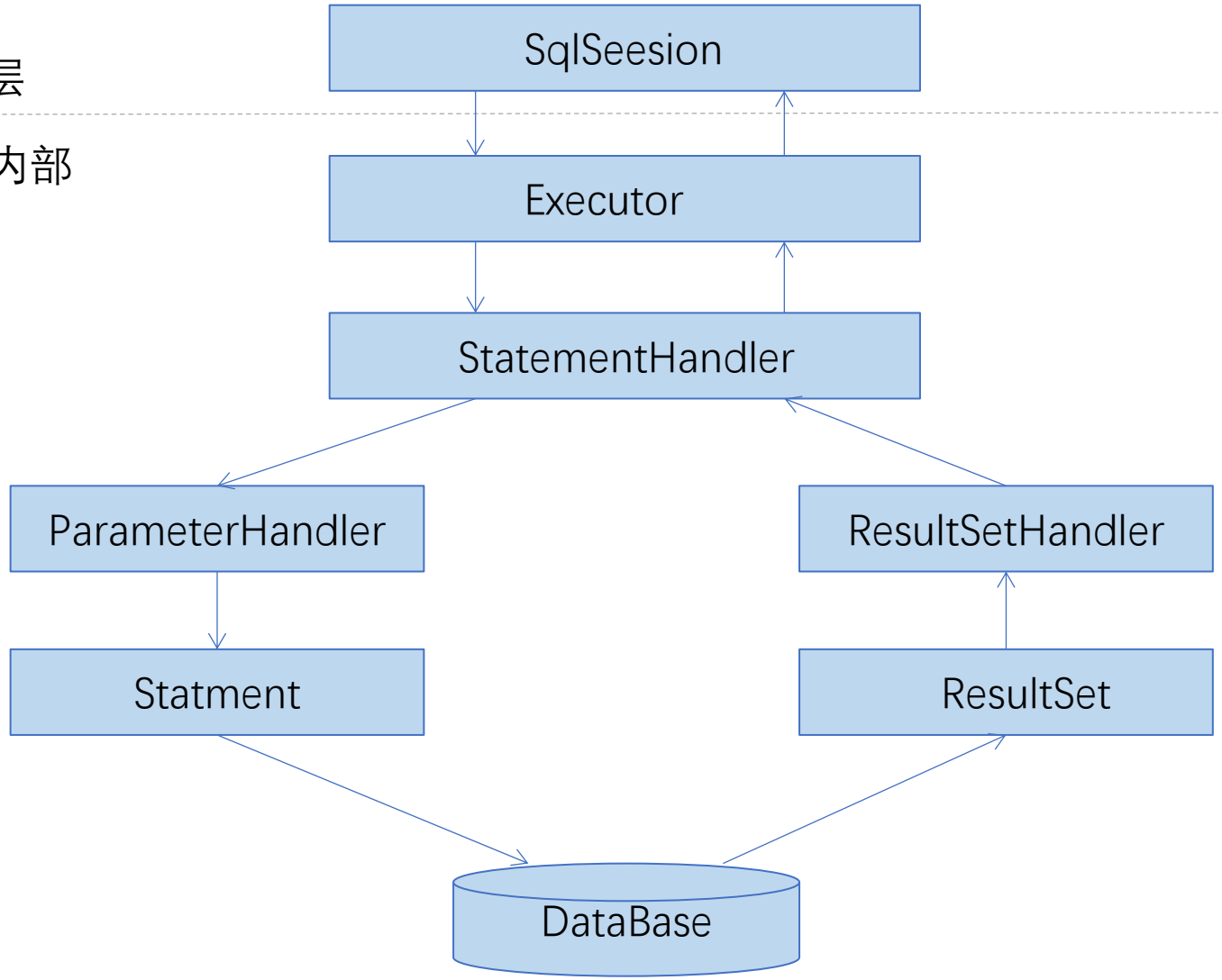
Executor的三个重要小弟

- 通过对SimpleExecutor doQuery()方法的解读发现，Executor是个指挥官，它在调度三个小弟工作：
  - StatementHandler：它的作用是使用数据库的Statement或PrepareStatement执行操作，启承上启下作用；
  - ParameterHandler：对预编译的SQL语句进行参数设置，SQL语句中的的占位符 “？” 都对应BoundSql.parameterMappings集合中的一个元素，在该对象中记录了对应的参数名称以及该参数的相关属性
  - ResultSetHandler：对数据库返回的结果集（ResultSet）进行封装，返回用户指定的实体类型；

Executor内部运作过程

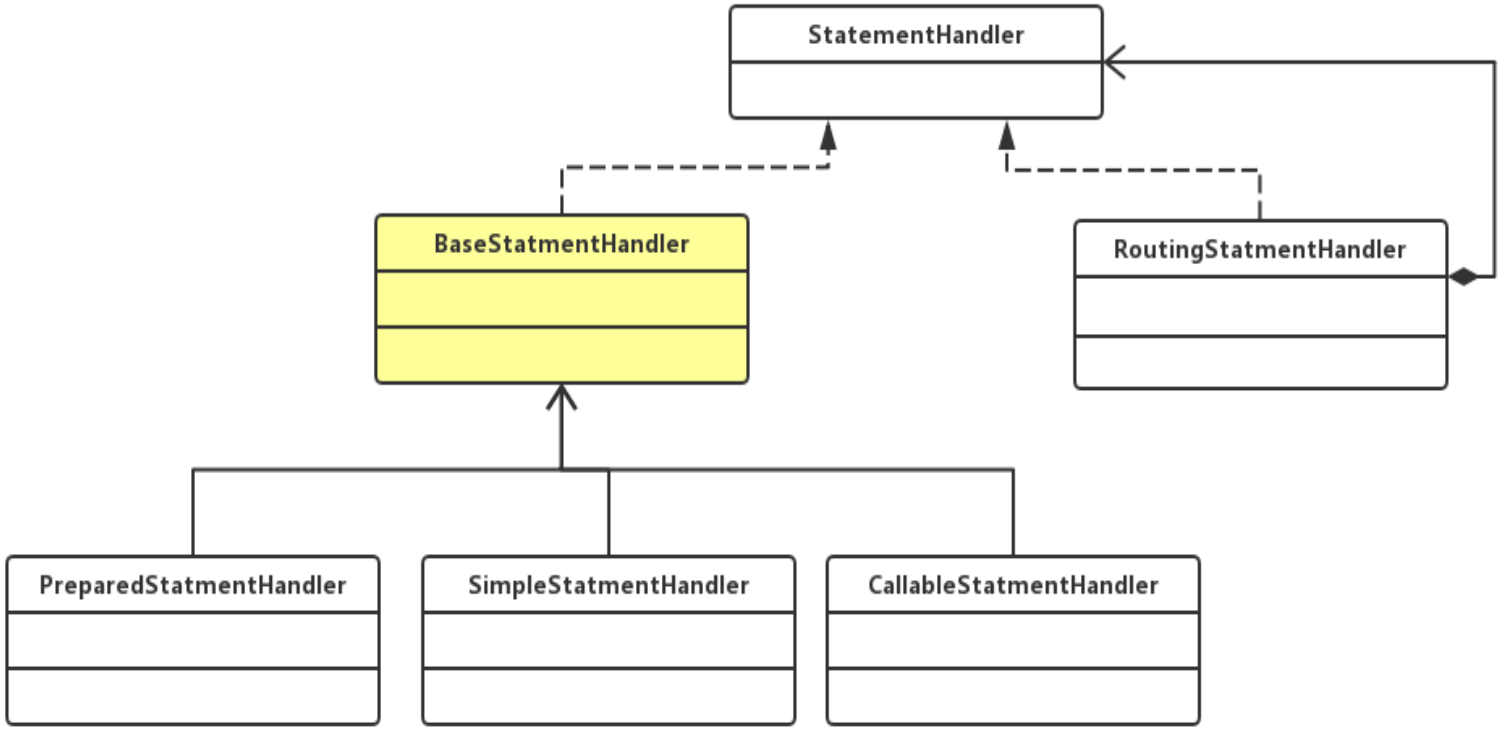
接口层

Excutor内部



StatementHandler分析

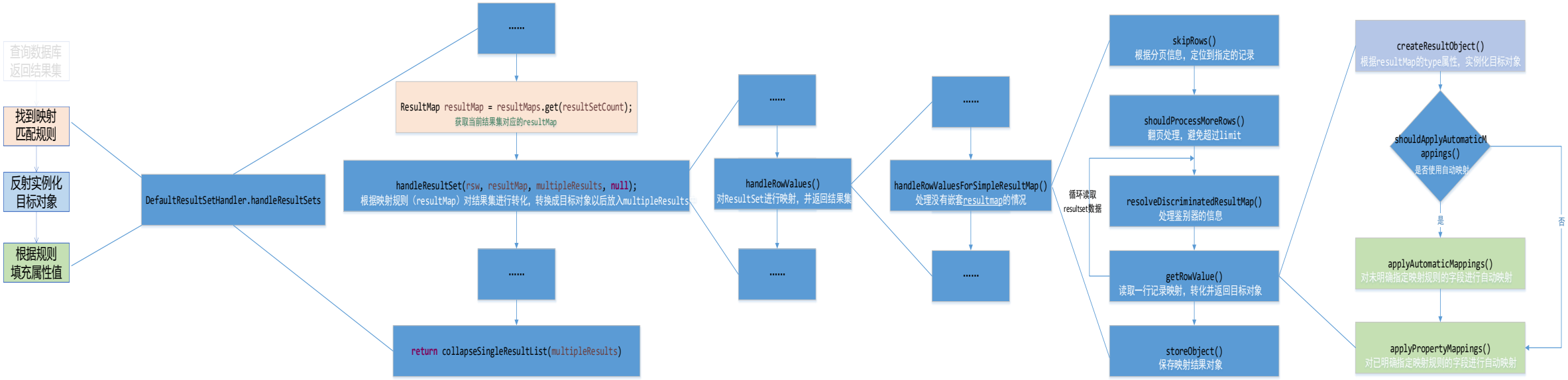
- StatementHandler完成Mybatis最核心的工作，也是Executor实现的基础；功能包括：创建statement对象，为sql语句绑定参数，执行增删改查等SQL语句、将结果映射集进行转化；



- ✓ **BaseStatementHandler**：所有子类的抽象父类，定义了初始化statement的操作顺序，由子类实现具体的实例化不同的statement（模板模式）；
- ✓ **RoutingStatementHandler**：Excutor组件真正实例化的子类，使用静态代理模式，根据上下文决定创建哪个具体实体类；
- ✓ **SimpleStatmentHandler**：使用statement对象访问数据库，无须参数化；
- ✓ **PreparedStatmentHandler**：使用预编译PrepareStatement对象访问数据库；
- ✓ **CallableStatmentHandler**：调用存储过程；

ResultSetHandler分析

- ResultSetHandler将从数据库查询得到的结果按照映射配置文件的映射规则，映射成相应的结果集对象；



spring集成Mybatis的原理分析

1. SqlSessionFactoryBean源码分析
2. MapperScannerConfigurer源码分析