MyBatis源码相对于Spring应该是层次比较清晰，容易理解的；其实简单来说就是解耦、封装，让开发者更关注业务层的开发，实现简单而又方便的调用。

MyBatis大概两部分内容：xml文件加载初始化、mapper接口动态代理加强。

注：以下介绍主要是基于使用注解的SpringBoot配置。

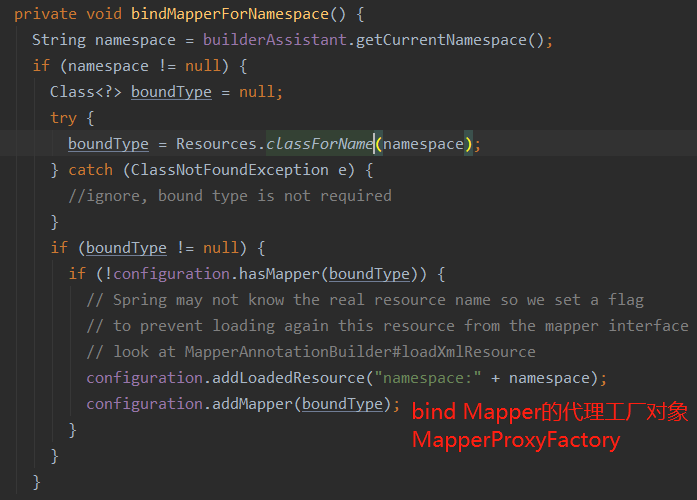
1. xml文件加载初始化

XMLConfigBuilder：解析Mybatis-config.xml文件节点</configuration>；这一块可以忽略，因为现在几乎都是Spring集成Mybatis，通过application.properties来配置，跳过config文件解析直接进入第二阶段XMLMapperBuilder解析mapper.xml文件。

XMLMapperBuilder：解析mapper.xml文件节点</mapper>；Mybatis结合springBoot或者Spring使用时，在配置dataSource创建SqlSessionFactory时启动的XMLMapperBuilder对象。主要有ResultMap、Sql节点等。

XMLStatementBuilder：解析select|insert|update|delete，封装成MappedStatement。

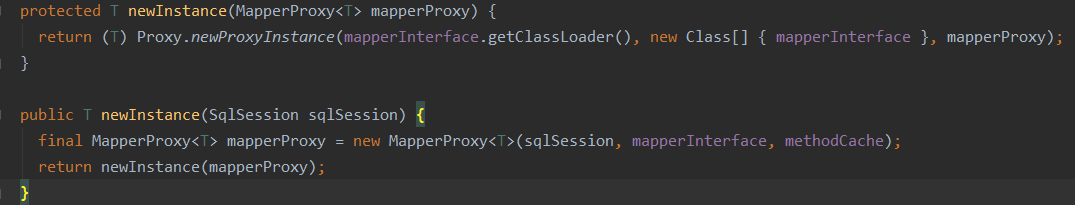
最后绑定mapper class和对代理工厂对象，mapperRegistry是其注册中心；创建Mapper接口的代理工厂类MapperProxyFactory，该类主要为Mapper接口方法生成代理类（MapperProxy（JDK动态代理）），已进行增强。



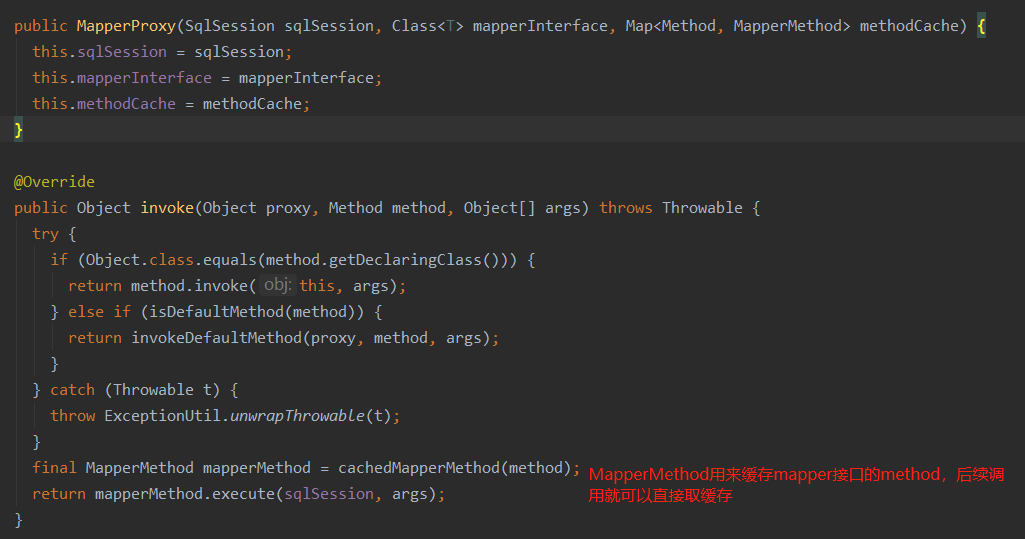
1. Mapper接口动态代理加强

问题思考：Mybatis是如何通过Mapper接口进行sql操作的？

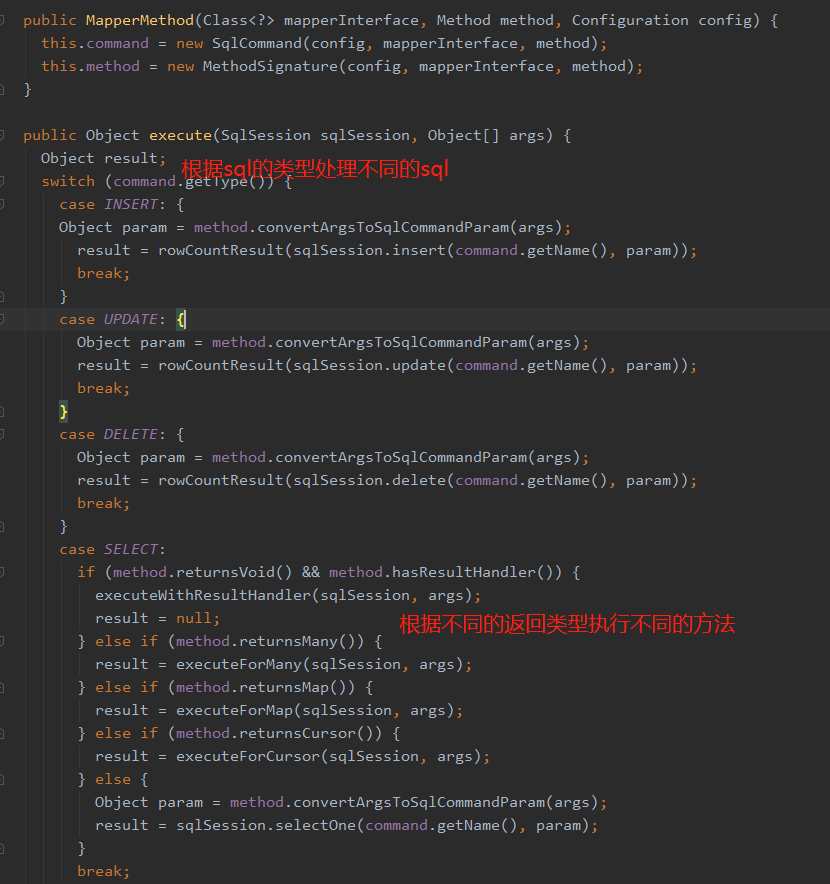
MapperProxy就是Mapper接口的代理类，这个代理类由代理工厂类MapperProxyFactory生成如图所示（此工厂类在第一步中已经将Mapper接口类型作为key进行缓存），MapperProxy的必须属性参数有mapper相关的sqlsession，mapper接口类型Class。



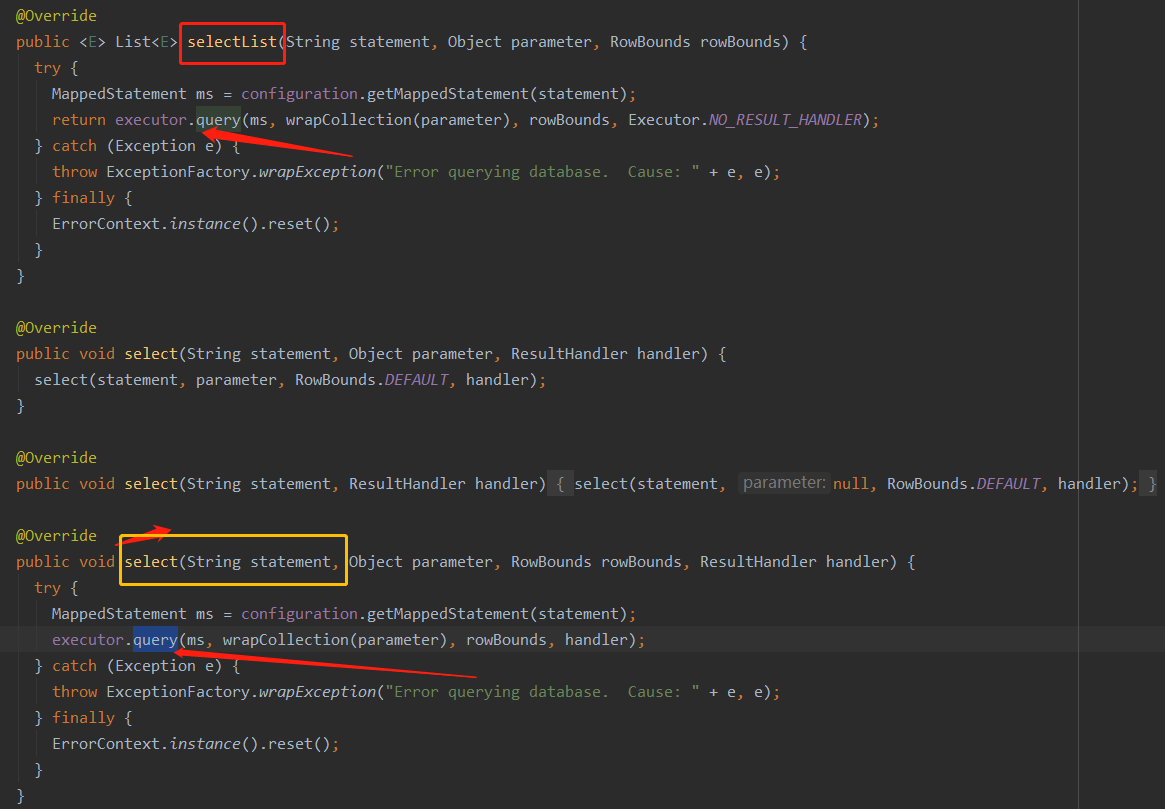
1、动态代理：当调用Mapper接口中的方法就会交由MapperProxy进行代理，



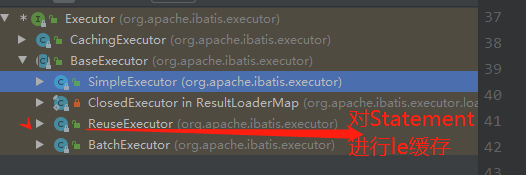
method.execute(sqlsession,args)真正调用sqlSession的地方。走进去看一下sqlSession触发不同的sql类型，但是对于select由于返回类型可能不同，再继续处理。



2、Executor执行：其实最终还是要使用Executor执行的

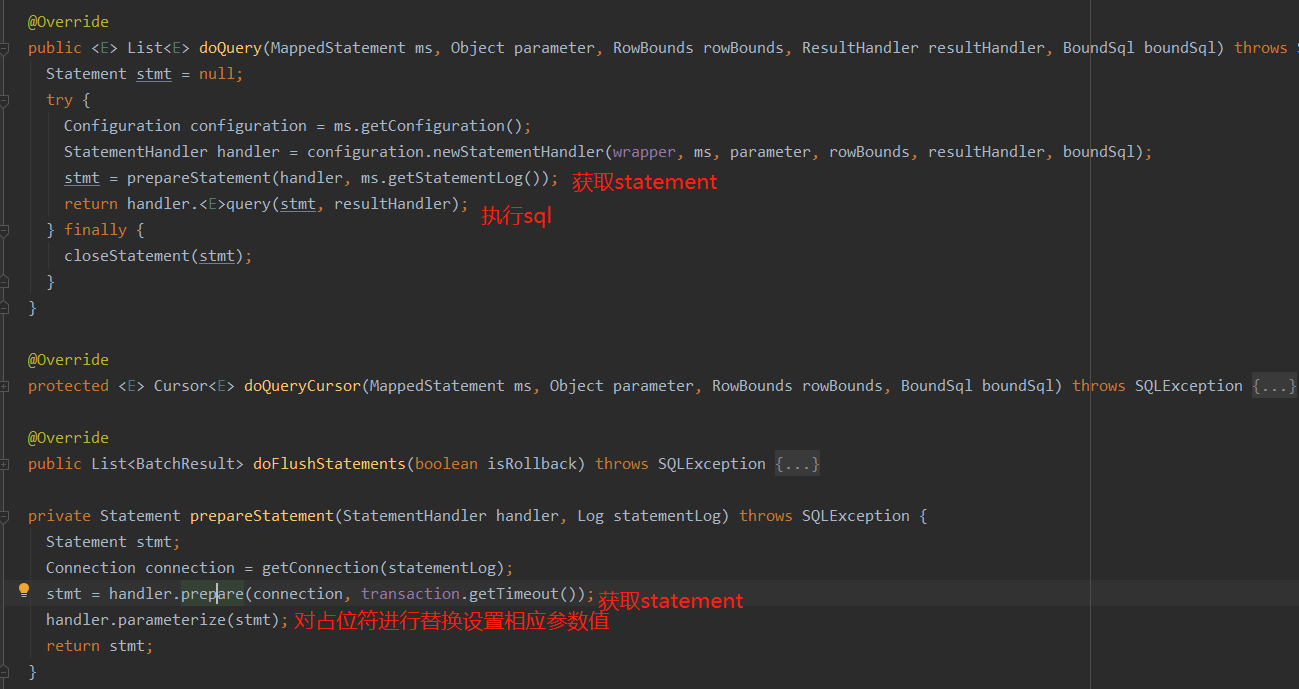


Executor有两个实现类，类图如下；Executor是由Configuration创建的，并且使用了装饰器模式以及动态代理进行加强（interceptorChain.pluginAll(executor)）

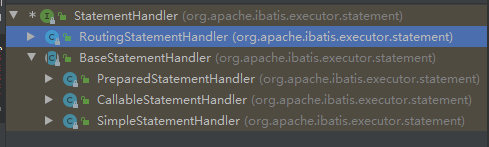


以SimpleExecutor为例看其处理过程：

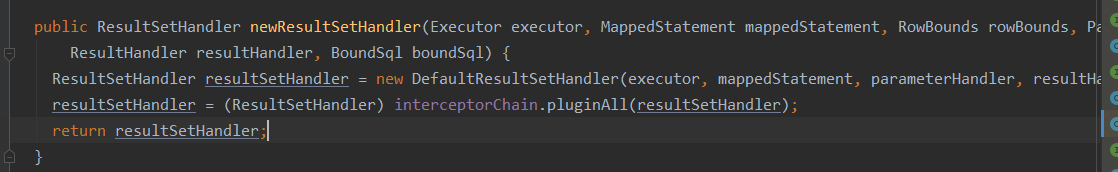
下图中doQuery()方法第四行代码configuration.newStatementHandler():在Configuration类中通过RoutingStatementHandler（静态代理）根据MappedStatement的StatementType（默认PREPARED）生成哪一种StatementHandler（类图如下），



StatementHandler继承关系：



3、ResultSetHandler解析查询结果：结果解析（ResultSetHandler）唯一默认实现类DefaultResultSetHandler，也会被加强；

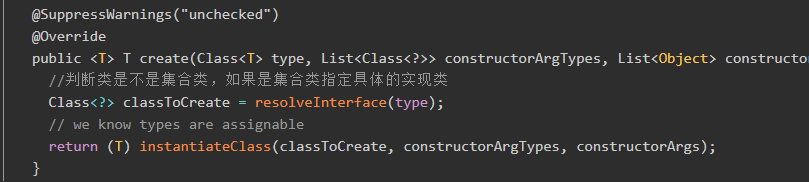


1. Mybatis涉及的模块以及设计模式使用：
2. 反射模块：查**询**的结果ResultSet通过反射构建bean。（实例化目标对象和对象属性赋值需要反射生成）---工厂模式

**反射的核心类：**

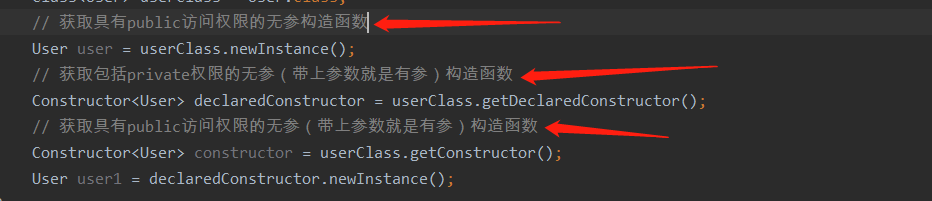
* **Reflector**：存放class定义的属性、方法信息
* **DefaultReflectorFactory**：依据class类新创建Reflector并缓存
* **MetaClass**：引用ReflectorFactory、Reflector，可以获取类的元信息
* **ObjectWrapper**：对bean、MetaClass的封装，可以实现对bean属性的设值以及获取
* **DefaultObjectFactory**：实例化对象
* **MetaObject**：主要封装了**bean**、**ObjectWrapper、DefaultObjectFactory、DefaultReflectorFactory，因此可以通过MetaObject设置和获取bean的属性。（主要是通过ObjectWrapper操作）**
* 用于实例化目标对象的类

ObjectFactory：MyBatis每次创建结果对象的新实例时，使用ObjectFactory构建POJO，默认实现**DefaultObjectFactory（反射实例化对象）**



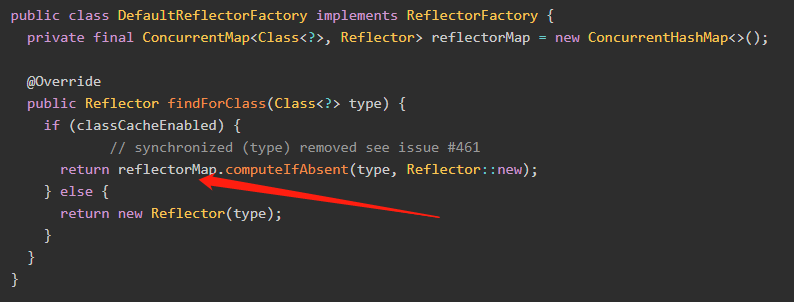


反射创建对象（注意区别）：

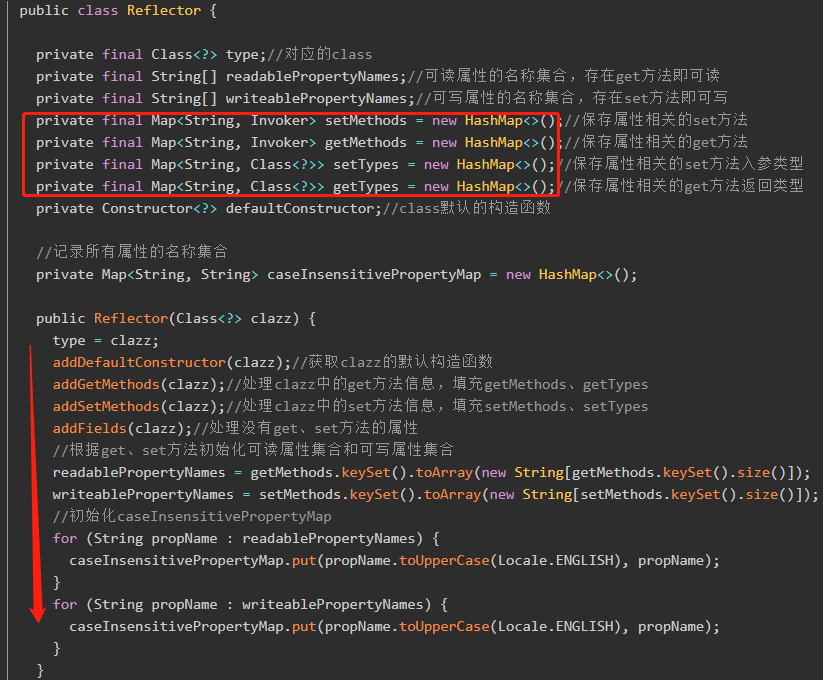


* 用于对象属性赋值的类

ReflectorFactory：创建Reflector的工厂类，Reflector是MyBatis反射模块的基础，每个Reflector对象都对应一个类，在其中缓存了反射操作所需要的类元信息。默认实现类DefaultReflectorFactory（**读取类元信息封装在Reflector中，每一个类都有一个Reflector**）。



Reflector(核心元信息缓存类)：



MetaObject：

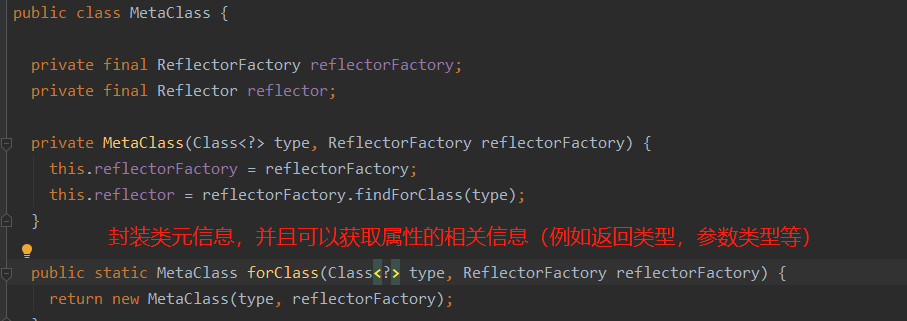
MetaObject封装了对象元信息，包装了MyBatis中的五个核心的反射类，也是提供给外部使用的反射工具类，可以利用它读取或者修改对象的属性信息（其实也是通过ObjectWrapper调用相关的方法，例如BeanWrapper）。



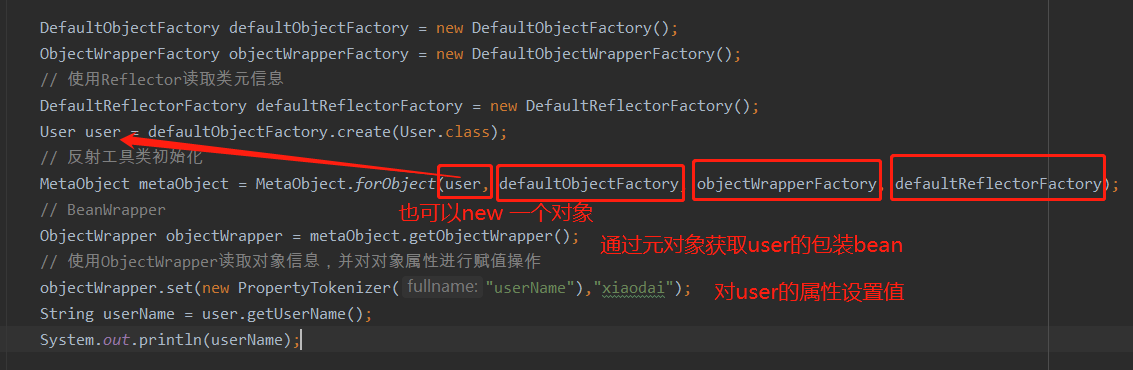
BeanWrapper作用如下:(是对bean的封装可以给bean属性设置值，也可以获取某个属性的值)



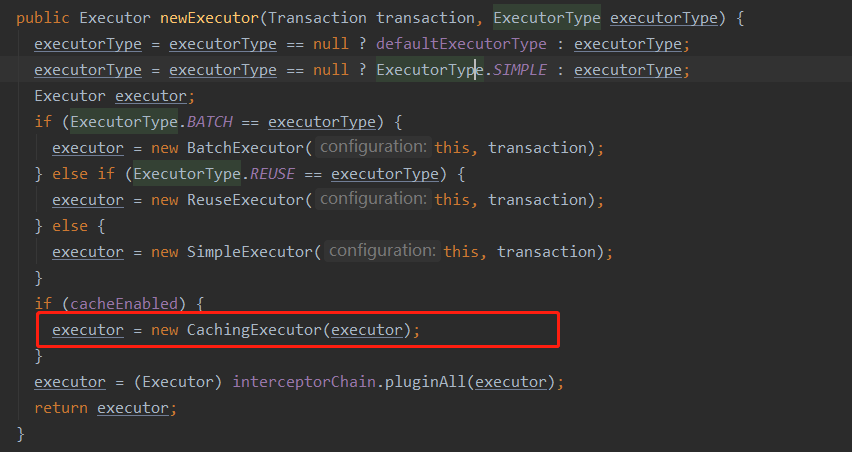
MetaClass(封装类元信息,并且也是通过Reflector来获取类的相关属性信息)：



总结：反射模块核心类使用如下



1. 插件模块:
2. 适配器模式：Logging模块可以引入第三方例如Slfj
3. 装饰器模式：cache模块，Configuration创建Executor时如果开启缓存则CachingExecutor会装饰BaseExecutor并会被interceptorChain.pluginAll(executor)进行加强。



1. 建造者模式：在解析mapper.xml时会通过MapperBuilderAssistant助理进行每一阶段的属性值的builder（例如熟悉addResultMap、addMappedStatement）。

