# **MyBatis代码整洁之道之设计模式**

[IMG_256](https://www.zhihu.com/people/zhui-zhu-41-99-2)

**[追逐](https://www.zhihu.com/people/zhui-zhu-41-99-2" \t "https://zhuanlan.zhihu.com/p/_blank)**

热衷于收集和创造优质技术文章

​关注他

到目前为止，MyBatis的源代码也看的七七八八了，不得不说MyBatis的代码写的真的很不错，逻辑清晰的同时功能还很齐全，同时可以发现,MyBatis中使用了非常多的设计模式，接下来简单总结下：

### **建造者模式**

在MyBaits的各种对象构建类中，大多数都继承自BaseBuilder类，并且大多数类都是通过建造者模式实现。

比如：

* SqlSessionFactoryBuilder 构建SqlSessionFactory
* XMLStatementBuilder 构建XMLStatement
* SqlSourceBuilder 构建SqlSource
* CacheBuilder构建Cache
* …

这些类都有一个共同点，那就是所创建的对象的过程比较复杂，就比如SqlSessionFactory的创建，是需要解析整个配置文件。

**因此，他们将对象的创建与使用分离开来，使用单独的建造者类来创建这个对象。**

同时，对于CacheBuilder这种具有可选参数的类，MyBatis还是用了经典的连点设计：

Cache cache = new CacheBuilder(currentNamespace)

.implementation(valueOrDefault(typeClass, PerpetualCache.class))

.addDecorator(valueOrDefault(evictionClass, LruCache.class))

.clearInterval(flushInterval)

.size(size)

.readWrite(readWrite)

.blocking(blocking)

.properties(props)

.build();

因此，对于以后我们的代码中，如果创建一个对象非常复杂，则可以选择使用建造者模式将对象的创建和使用分离开来。

### **工厂模式**

MyBatis中也存在很多Factory，主要是用来根据不同的参数初始化对象。

比如：

* SqlSessionFactory 创建SqlSession
* ProxyFactory 创建代理类
* LogFactory 获取用户当前配置的Log
* DefaultReflectorFactory创建Reflector
* …

这些类的作用都是为了在创建对象前/后进行一些额外的处理，或者是修改获取这个类的获取方式，比如DefaultReflectorFactory是为了添加一层缓存，创建的时候首先读取缓存，比如LogFactory 是为了查找是否有用户自己配置的Log，如果有，则使用用户配置类等。

这里也可以看出来，**建造者模式和工厂模式的区别在于建造者模式主要用于解决创建对象的复杂问题，而工厂模式是为了简单修改获取对象的方式以及根据用户的参数创建不同对象**

但是其实两者都是为了将对象的创建和使用分开。

同时值得一提的是，MyBatis这里都是使用的工厂模式，而不是简单的简单工厂，这样使得代码根据低耦合，同时可以让用户自己实习这些工厂类。

### **单例模式**

说实话，MyBatis中单例模式用的还是比较少，因为使用单例就有点像是在使用静态类。不过对于一些简单的工具类，MyBatis依然使用了单例模式。其中创建单例模式的方式使用的是最方便的静态内部类创建法。

比如：

* VFS类

### **适配器模式**

在MyBatis中，可以自己定义Log的实现，使得整个集成MyBatis的系统可以使用同一的日志系统。然而对于多种Log的实现，MyBatis想要使用一个接口统一的使用它，就需要用到适配器模式了。

在MyBatis的Log包中，定义了一个Log接口，然后将各种其他Log都实现了这个Log接口，比如

* JakartaCommonsLoggingImpl
* Jdk14LoggingImpl
* Log4jImpl
* Log4j2Impl
* …

这样，在MyBatis中就可以无差别的使用这些实现类

### **代理模式**

一般来说，我们用代理模式的情况很少，但是相信动态代理大家一定听说过。

由于Java提供的动态类实在方便并且功能强大，所以很多地方都会有它的身影。

比如：

* 通过动态代理实现Mapper接口的创建
* 通过动态代理实现插件的包装
* 通过动态代理实现延迟执行SQL
* …

### **装饰器模式**

MyBatis的Cache可以被用户定义很多不同的特性，被缓存替换规则，是否有定时刷新，使用弱引用引用对象等等。。

刚开始接触这些配置的时候，就觉得如果让我设计，还挺麻烦，多么多功能。

后来看源码才发现设计的巧妙，MyBatis直接使用装饰器模式将HashMap层层封装，需要什么功能就添加一层包装，这样在后面的功能扩展中，完全不用修改任何的代码，真是将装饰器模式用到了淋漓尽致。

### **组合模式**

MyBatis的动态SQL中，可以定义一些类似编程语言结构。并且MyBatis允许嵌套定义这种结构，比如：

<update id="update" parameterType="User">

UPDATE users

<trim prefix="SET" prefixOverrides=",">

<if test="name != null and name != ''">

name = #{name}

</if>

<if test="age != null and age != ''">

, age = #{age}

</if>

<if test="birthday != null and birthday != ''">

, birthday = #{birthday}

</if>

</trim>

<where> 1=1

<if test="id != null">

and id = ${id}

</if>

</where>

</update>

这种类似一个树型结构。

恰好，组合模式便是用来针对这种情况的，使用组合模式，能够非常方便的遍历以及处理整个树的所有节点。并且不用各种if-else的判断

### **模板方法模式**

模板方法模式也是我们日常开发中比较喜欢使用的模式，它比较简单，而且能够有效地消除重复代码，在MyBatis中，有很多BaseXX，目的就是为了使用模板方法模式

比如：

* BaseExecutor
* BaseStatementHandler
* BaseTypeHandler
* …

模板方法很大限度的使用了继承的特性，它将不变的代码放在父类，变化留给子类，重用了代码，并且使代码的使用顺序也被固定在子类当中。

### **责任链模式**

在MyBatis中，用户可以自定义插件修改一些核心类的行为，而实现方式便是通过责任链。

MyBatis使用InterceptorChain保存所有插件，通过动态代理将插件包装成需要使用的目标类，完美的实现的插件的功能。

不过在使用插件的过程中，需要注意调用invocation#proceed()，否则可能会出现断链的情况。

### **策略模式**

MyBatis中，可以自定义使用三种不同的Executor，也可以自定义使用三种不同的Statement。

MyBatis的处理方式便是将这些类通过3种不同的类实现，并实现同一个接口。然后通过依赖注入的方式注入到使用类中。

这便是策略模式，策略模式使用非常简单，其实就是对依赖注入的实施。

策略模式一般都会和简单工厂配合使用，因为需要根据条件，选择不同的策略。

### **迭代器模式**

在使用MyBatis定义SQL的时候，可以使用#{}获取参数，同时还可以使用#{xxx.xx}获取对象的属性，同时还可以使用#{xxx[1]}获取集合中的指定坐标的元素。

这么多种访问方式，并且还可以嵌套使用，

比如：

student.names[0].firstName

在MyBatis中，负责解析这种语法的类为PropertyTokenizer,这个类实现了Iterator接口，使得其他代码可以使用迭代器来访问它。

但是，在MyBatis中，并没有使用foreach语法糖来访问它，而是直接使用了hasNext(),next()方法结合递归进行访问。

—