<https://blog.csdn.net/hongtaolong/article/details/88688634>

spring中的aop使用动态代理增强，mybatis中使用动态代理生成mapper，动态代理主要有JDK和CGLIB两种方式。

动态代理：是使用反射和字节码的技术，在运行期创建指定接口或类的子类，以及其实例对象的技术，通过这个技术可以无侵入的为代码进行增强.

mybatis的使用中我们知道不需要创建dao中的mapper接口的子类，也能调用到相应的方法，其实就是生成的实现了mapper接口的动态的代理类.

1.jdk实现的动态代理由两个重要的成员组成，分别是Proxy、InvocationHandler

Proxy：是所有动态代理的父类，它提供了一个静态方法来创建动态代理的class对象和实例

InvocationHandler：每个动态代理实例都有一个关联的InvocationHandler，在代理实例上调用方法是，方法调用将被转发到InvocationHandler的invoke方法

2.CGLIB（Code Generation Library）是一个基于ASM的字节码生成库，它允许我们在运行时对字节码进行修改和动态生成。CGLIB通过继承的方式实现代理（局限，final方法是不能够被重写，所以它不能增强被final修饰的方法）

CGLIB的实现也有两个重要的成员组成，Enhancer、MethodInterceptor，其实这两个的使用和jdk实现的动态代理的Proxy、InvocationHandler非常相似

Enhancer：来指定要代理的目标对象，实际处理代理逻辑的对象，最终通过调用create（）方法得到代理对象、对这个对象所有的非final方法的调用都会转发给MethodInterceptor

MethodInterceptor：动态代理对象的方法调用都会转发到intercept方法进行增强.

**1、JDK原声动态代理时java原声支持的、不需要任何外部依赖、但是它只能基于接口进行代理（因为它已经继承了proxy了，java不支持多继承）**

**2、CGLIB通过继承的方式进行代理、无论目标对象没有没实现接口都可以代理，但是无法处理final的情况（final修饰的方法不能被覆写）**

<https://blog.csdn.net/weixin_36759405/article/details/82770422>

JDK动态代理：利用反射机制生成一个实现代理接口的匿名类，在调用具体方法前调用InvokeHandler来处理。

CGlib动态代理：利用ASM（开源的Java字节码编辑库，操作字节码）开源包，将代理对象类的class文件加载进来，通过修改其字节码生成子类来处理。

区别：JDK代理只能对实现接口的类生成代理；CGlib是针对类实现代理，对指定的类生成一个子类，并覆盖其中的方法，这种通过继承类的实现方式，不能代理final修饰的类。82770422

1.JDK代理使用的是反射机制实现aop的动态代理，CGLIB代理使用字节码处理框架asm，通过修改字节码生成子类。所以jdk动态代理的方式创建代理对象效率较高，执行效率较低，cglib创建效率较低，执行效率高；2.JDK动态代理机制是委托机制，具体说动态实现接口类，在动态生成的实现类里面委托hanlder去调用原始实现类方法，CGLIB则使用的继承机制，具体说被代理类和代理类是继承关系，所以代理类是可以赋值给被代理类的，如果被代理类有接口，那么代理类也可以赋值给接口。

反射：**反射是框架设计的灵魂**

**（使用的前提条件：必须先得到代表的字节码的Class，Class类用于表示.class文件（字节码））**

[**JAVA反射机制**](https://baike.baidu.com/item/JAVA反射机制/6015990)**是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能称为java语言的反射机制。**

**与反射有关的类：**

Class类：类的实体，在运行的Java应用程序中表示类的接口

Field类：类的成员变量（或类的属性）

Method类：类的方法

Constructor类：类的构造器。

Class类的

|  |  |
| --- | --- |
| getClassLoader() | 获得类的加载器，在JDK动态代理中的invoke方法的第一个参数 |

|  |  |
| --- | --- |
| getInterfaces() | 获得当前类实现的类或是接口 |

**带有Declared修饰的方法可以反射到私有的方法，没有Declared修饰的只能用来反射公有的方法。**其他的Annotation、Field、Constructor也是如此。

加载的时候：Class对象的由来是将class文件读入内存，并为之创建一个Class对象。

比如Student stu = new Student();

当我们new Student时，jvm会加载Student.class. JVM会去本地磁盘找student.class文件并加载到JVM内存中。将.class文件读入内存同时产生class对象。这个class对象是JVM自动创建的（一个类只产生一个class对象。

Java类加载机制使用双亲委托模式。

一个ClassLoader加载一个类时，首先需要将任务委托给其父加载器，直到Bootstrap ClassLoader。

如果父加载器未加载该类，则逐层返回给委托发起者即当前ClassLoader进行加载。

在正常应用中，用户不自定义类加载器。

类加载工作首先由App ClassLoader发起，然后委托给Extension ClassLoader，最后委托给Bootstrap ClassLoader。

http://www.imooc.com/article/261822?block\_id=tuijian\_wz，图都看不到

）

Class没有公共构造方法。class对象是在加载类时由Java虚拟机以及通过调用类加载器中的defineClass方法自动构造的。

## **获取Class对象的三种方式**

### **1.1 Object ——> getClass(); 1.2 任何数据类型（包括基本数据类型）都有一个“静态”的class属性 1.3 通过Class类的静态方法：forName（String  className）(常用)**

Class.forName(”fanshe.Student”);//注意此字符串必须是真实路径，就是带包名的类路径，包名.类名

三种方式常用第三种，第一种对象都有了还要反射干什么。第二种需要导入类的包，依赖太强，不导包就抛编译错误。一般都第三种，一个字符串可以传入也可写在配置文件中等多种方法。

配置文件以txt文件为例子（pro.txt）：

**className = cn.fanshe.Student**

**methodName = show**

**import java.io.FileNotFoundException;**

1. **import** java.io.FileReader;
2. **import** java.io.IOException;
3. **import** java.lang.reflect.Method;
4. **import** java.util.Properties;
6. /\*
7. \* 我们利用反射和配置文件，可以使：应用程序更新时，对源码无需进行任何修改
8. \* 我们只需要将新类发送给客户端，并修改配置文件即可
9. \*/
10. **public** **class** Demo {
11. **public** **static** **void** main(String[] args) **throws** Exception {
12. //通过反射获取Class对象
13. Class stuClass = Class.forName(getValue(”className”));//”cn.fanshe.Student”
14. //2获取show()方法
15. Method m = stuClass.getMethod(getValue(”methodName”));//show
16. //3.调用show()方法
17. m.invoke(stuClass.getConstructor().newInstance());
19. }
21. //此方法接收一个key，在配置文件中获取相应的value
22. **public** **static** String getValue(String key) **throws** IOException{
23. Properties pro = **new** Properties();//获取配置文件的对象
24. FileReader in = **new** FileReader(“pro.txt”);//获取输入流
25. pro.load(in);//将流加载到配置文件对象中
26. in.close();
27. **return** pro.getProperty(key);//返回根据key获取的value值
28. }
29. }

泛型用在编译期，编译过后泛型擦除（消失掉）。所以是可以通过反射越过泛型检查的。

ArrayList<String> strList = **new** ArrayList<>();

1. strList.add(”aaa”);
2. strList.add(”bbb”);
4. //  strList.add(100);
5. //获取ArrayList的Class对象，反向的调用add()方法，添加数据
6. Class listClass = strList.getClass(); //得到 strList 对象的字节码 对象
7. //获取add()方法
8. Method m = listClass.getMethod(”add”, Object.**class**);
9. //调用add()方法
10. m.invoke(strList, 100);

**https://yq.aliyun.com/articles/711621?spm=a2c4e.11155472.0.0.4f905aedDZmmNQ**