CGLIB字节码增强

- 没有接口，只有实现类
- 采用字节码增强框架cglib，在运行时，创建目标类的子类，从而对目标类进行增强
- 导入jar包：
  - 手动导包
    - 核心：cglib-2.2.jar
    - 依赖：asm-3.3.jar
  - spring-core.jar包中已经整合了~

```
public class MyBeanFactory {

    public static UserServiceImpl createService(){

        //1.目标类
        final UserServiceImpl userService = new UserServiceImpl();
        //2.切面类
        final MyAspect myAspect = new MyAspect();

         //3.代理类 采用cglib，底层创建目标类的子类
        //3.1核心类
        Enhancer enhancer = new Enhancer();
        //3.2确定父类
        enhancer.setSuperclass(userService.getClass());
        /*
         * 3.3设置回调函数,MethodIntrospector接口等效jdk InvocationHandler接口
         * intercept()  等效    jdk invoke()
         *  参数1,2,3:与invoke一样
         *  参数4: MethodProxy 方法的代理
         */
        enhancer.setCallback(new MethodInterceptor(){
            public Object intercept(Object proxy, Method method, Object[] args,
                    MethodProxy methodProxy) throws Throwable {
                //前
                myAspect.before();

                //执行目标类的方法
                Object obj = method.invoke(userService, args);
                //执行代理类的父类,执行目标类（目标类和代理类 父子关系）
                //后
                myAspect.after();
                return obj;
            }});
        //3.4创建代理
        UserServiceImpl proService = (UserServiceImpl) enhancer.create();

        return proService;

    }
```

```
public class MyBeanFactory {

 public static UserServiceImpl createService(){

 //1.目标类
 final UserServiceImpl userService = new UserServiceImpl();
 //2.切面类
 final MyAspect myAspect = new MyAspect();
```

```
    //3.代理类 采用cglib，底层创建目标类的子类
    //3.1核心类
    Enhancer enhancer = new Enhancer();
    //3.2确定父类
    enhancer.setSuperclass(userService.getClass());
    /*
     * 3.3设置回调函数,MethodIntrospector接口等效jdk InvocationHandler接口
     * intercept() 等效 jdk invoke()
     *  参数1,2,3:与invoke一样
     *  参数4：MethodProxy 方法的代理
     *
     */

    enhancer.setCallback(new MethodInterceptor(){

      public Object intercept(Object proxy, Method method, Object[] args,
          MethodProxy methodProxy) throws Throwable {

        //前
        myAspect.before();

        //执行目标类的方法
        Object obj = method.invoke(userService, args);
        //执行代理类的父类,执行目标类（目标类和代理类 父子关系）
    //    methodProxy.invokeSuper(proxy, args);

        //后
        myAspect.after();
        return obj;
      }});
    //3.4创建代理
    UserServiceImpl proService = (UserServiceImpl) enhancer.create();

    return proService;

  }
}
```