JDK proxy

前提条件：目标类实现了接口，就会使用JDK proxy进行动态代理，proxy是基于接口的动态代理

1.需要定一个实现了InvocationHandler接口的代理类，通过Proxy类的静态方法得到代理对象

```
/**
 * classLoader:将代理对象加载到jvm虚拟机
 * interfaces：要代理的接口集和
 * invocationHandler：拦截器，调用动态代理的对象时，走到这个handler中的invoke中
 */
UserService o = (UserService) Proxy.newProxyInstance(AOP.class.getClassLoader(),
        new Class[]{UserService.class},
        new MyInvocationHandler());

o.addUser( username: "12");
```

MyInvocationHandler的实现

```
1 usage
public class MyInvocationHandler implements InvocationHandler {
    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        System.out.println("调用了被代理的方法");
        return null;
    }
}
```

2.此时调用代理对象o的addUser方法，执行的就是我们传进去的代理类的invoke函数

```
C:\jdk\jdk1.8\bin\java.exe "-javaagent:C:\software\Idea\
调用了被代理的方法

Process finished with exit code 0
```

cglib

cglib是一个基于类的动态代理，可以代理任何没有实现接口的类

通过cglib库，可以直接在运行时创建目标代理对象的子类，通过重写父类的方法从而实现对于类的动态代理

cglib的原理是在运行时动态生成字节码，所以相比proxy虽然更加灵活但是会有一定的性能开销