---git:jiaofu

接口: public interface UserService

实现类**: public class** UserServiceImpl **implements** UserService 动态代理会生成

```
extends Proxy implements 代理的接口
```

public final class \$Proxy@ 这样的类,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为hanlder去调用原始实现类方法。

与

```
UserServiceImpl$$EnhancerByCGLIB$$38614660$$FastClassByCGLIB$$3ef05'

UserServiceImpl$$EnhancerByCGLIB$$38614660.class

UserServiceImpl$$FastClassByCGLIB$$9080f1f4.class
```

第二个方法里面是

```
package juc.cglibtext;

import java.lang.reflect.Method;
import net.sf.cglib.proxy.Callback;
import net.sf.cglib.proxy.Factory;
import net.sf.cglib.proxy.MethodInterceptor;
import net.sf.cglib.proxy.MethodProxy;

public class UserServiceImpl$$EnhancerByCGLIB$$38614660
extends UserServiceImpl
implements Factory

{
```

CGLIB包的底层是通过使用一个小而快的字节码处理框架ASM(Java字节码操控框架),来转换字节码并生成新的类

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,CGLIB原理:动态生成一个要代理类的子类,子类重写要代理的类的所有不是final的方法。在子类中采用方法拦截的技术拦截所有父类方法的调用,顺势织入横切逻辑。它比使用java反射的JDK动态代理要快。

另外JDK代理只能对接口进行代理,Cglib则是对实现类进行代理。

---git:jiaofu