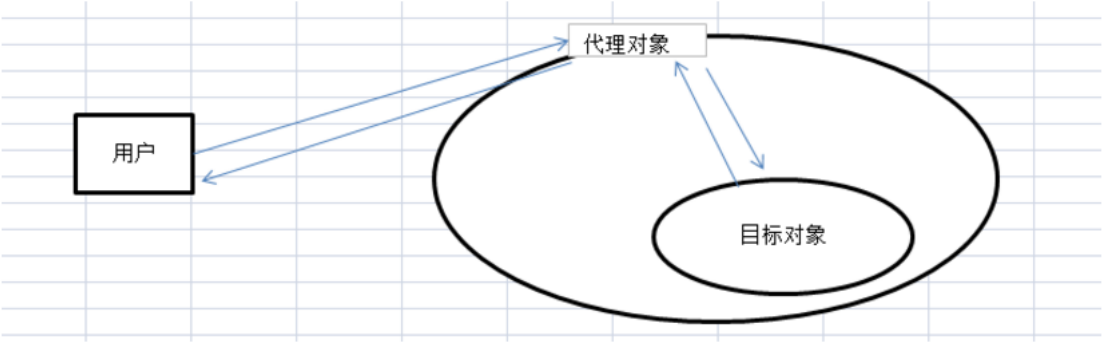
Java的三种代理模式：

代理(Proxy)是一种设计模式，提供了对目标对象另外的访问方式，即通过代理对象访问目标对象，这样的好处是：可以在目标对象实现的基础上，增强额外的功能操作，即扩展目标对象的功能。

使用代理的主要原因是，有一种编程思想，不能随意去修改别人写好的代码或者方法，如果需要修改，可以通过代理的方式来扩展该方法。



代理有三种模式：静态代理、动态代理、Cglib代理

静态代理：在使用时，需要定义接口或者父类，被代理对象与代理对象一起实现相同的接口或者是集成相同的父类。

```
/**
 * 接口
 */
public interface IUserDao {

    void save();
}
```

目标对象:UserDao.java

```
/**
 * 接口实现
 * 目标对象
 */
public class UserDao implements IUserDao {
    public void save() {
        System.out.println("----已经保存数据!----");
    }
}
```

代理对象:UserDaoProxy.java

```
/**
 * 代理对象,静态代理
 */
public class UserDaoProxy implements IUserDao{
    //接收保存目标对象
    private IUserDao target;
    public UserDaoProxy(IUserDao target){
        this.target=target;
    }

    public void save() {
        System.out.println("开始事务...");
        target.save();//执行目标对象的方法
        System.out.println("提交事务...");
    }
}
```

测试类:App.java

```
/**
 * 测试类
 */
public class App {
    public static void main(String[] args) {
        //目标对象
        UserDao target = new UserDao();

        //代理对象,把目标对象传给代理对象,建立代理关系
        UserDaoProxy proxy = new UserDaoProxy(target);

        proxy.save();//执行的是代理的方法
    }
}
```

静态代理总结：

1.可以做到在不修改目标对象功能的前提下，对目标功能进行扩展

2.缺点：因为代理对象和目标对象实现一样的接口，当对接口进行改变时，目标对象和代理对象都要维护。为了解决这种缺点，使用动态代理。

动态代理：

代理对象不需要实现接口

代理工厂类:ProxyFactory.java

```
/**
 * 创建动态代理对象
 * 动态代理不需要实现接口,但是需要指定接口类型
 */
public class ProxyFactory{

    //维护一个目标对象
    private Object target;
    public ProxyFactory(Object target){
        this.target=target;
    }

   //给目标对象生成代理对象
    public Object getProxyInstance(){
        return Proxy.newProxyInstance(
                target.getClass().getClassLoader(),
                target.getClass().getInterfaces(),
                new InvocationHandler() {
                    @Override
                    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
                        System.out.println("开始事务2");
                        //执行目标对象方法
                        Object returnValue = method.invoke(target, args);
                        System.out.println("提交事务2");
                        return returnValue;
                    }
                }
        );
    }

}
```

测试类:App.java

```
/**
 * 测试类
 */
public class App {
    public static void main(String[] args) {
        // 目标对象
        IUserDao target = new UserDao();
        // 【原始的类型 class cn.itcast.b_dynamic.UserDao】
        System.out.println(target.getClass());

        // 给目标对象，创建代理对象
        IUserDao proxy = (IUserDao) new ProxyFactory(target).getProxyInstance();
        // class $Proxy0   内存中动态生成的代理对象
        System.out.println(proxy.getClass());

        // 执行方法    【代理对象】
        proxy.save();
    }
}
```

代理对象不需要实现接口，但是目标对象一定要实现接口，否则不能用动态代理。

1.3.Cglib代理

上面的静态代理和动态代理模式都是要求目标对象是实现一个接口的目标对象,但是有时候目标对象只是一个单独的对象,并没有实现任何的接口,这个时候就可以使用以目标对象子类的方式类实现代理,这种方法就叫做:Cglib代理

Cglib代理,也叫作子类代理,它是在内存中构建一个子类对象从而实现对目标对象功能的扩展.

- JDK的动态代理有一个限制,就是使用动态代理的对象必须实现一个或多个接口,如果想代理没有实现接口的类,就可以使用Cglib实现.
- Cglib是一个强大的高性能的代码生成包,它可以在运行期扩展java类与实现java接口.它广泛的被许多AOP的框架使用,例如Spring AOP和synaop,为他们提供方法的interception(拦截)
- Cglib包的底层是通过使用一个小而块的字节码处理框架ASM来转换字节码并生成新的类.不鼓励直接使用ASM,因为它要求你必须对JVM内部结构包括class文件的格式和指令集都很熟悉.

Cglib子类代理实现方法:
1.需要引入cglib的jar文件,但是Spring的核心包中已经包括了Cglib功能,所以直接引入 `pring-core-3.2.5.jar` 即可.
2.引入功能包后,就可以在内存中动态构建子类
3.代理的类不能为final,否则报错
4.目标对象的方法如果为final/static,那么就不会被拦截,即不会执行目标对象额外的业务方法.

代码示例:
目标对象类:UserDao.java

```
/**
 * 目标对象,没有实现任何接口
 */
public class UserDao {

    public void save() {
        System.out.println("----已经保存数据!----");
    }
}
```

Cglib代理工厂:ProxyFactory.java

Cglib代理工厂:ProxyFactory.java

```
/**
 * Cglib子类代理工厂
 * 对UserDao在内存中动态构建一个子类对象
 */
public class ProxyFactory implements MethodInterceptor{
    //维护目标对象
    private Object target;

    public ProxyFactory(Object target) {
        this.target = target;
    }

    //给目标对象创建一个代理对象
    public Object getProxyInstance(){
        //1.工具类
        Enhancer en = new Enhancer();
        //2.设置父类
        en.setSuperclass(target.getClass());
        //3.设置回调函数
        en.setCallback(this);
        //4.创建子类(代理对象)
        return en.create();

    }

    @Override
    public Object intercept(Object obj, Method method, Object[] args, MethodProxy proxy) throws Throwable {
        System.out.println("开始事务...");

        //执行目标对象的方法
        Object returnValue = method.invoke(target, args);

        System.out.println("提交事务...");

        return returnValue;
    }
}
```

测试类:

测试类:

```
/**
 * 测试类
 */
public class App {

    @Test
    public void test(){
        //目标对象
        UserDao target = new UserDao();

        //代理对象
        UserDao proxy = (UserDao)new ProxyFactory(target).getProxyInstance();

        //执行代理对象的方法
        proxy.save();
    }
}
```

在Spring的AOP编程中:
如果加入容器的目标对象有实现接口,用JDK代理
如果目标对象没有实现接口,用Cglib代理

cglib可以对任意类生成代理对象，它的原理是对目标对象进行继承代理，如果目标对象被final修饰，那么该类无法被cglib代理。