动态代理（**Dynamic Proxy**）

一、切面编程（在方法前后加上一些逻辑，如下：在**save(User user)**方法前后加逻辑）

准备工作：

```
1. public class UserDAOImpl implements UserDAO {
2.     public void save(User user) {
3.         System.out.println("user saved!");
4.     }
5. }
```

1、实现方法一：直接在方法里面逻辑前后进行添加

```
1. public class UserDAOImpl implements UserDAO {
2.     public void save(User user) {
3.         System.out.println("user start");    //在方法前添加逻辑
4.         System.out.println("user saved!");
5.     }
6. }
```

2、实现方法二：使用继承进行添加

```
1. public class UserDAOImpl2 extends UserDAOImpl {
2.     @Override
3.     public void save(User user) {
4.         System.out.println("user start");    //在方法前添加逻辑
5.         super.save(user);
6.     }
7. }
```

3、实现方法三：使用接口进行添加

添加接口UserDAO

```
1. public interface UserDAO {
2.     public void save(User user);
3. }
```

让UserDAOImpl继承接口

```
1. public class UserDAOImpl implements UserDAO {
2.     public void save(User user) {
3.         System.out.println("user saved!");
4.     }
5. }
```

让UserDAOImpl2继承接口

```
1. public class UserDAOImpl2 implements UserDAO {
```

```
    private UserDAOImpl userDAOImpl = new UserDAOImpl();
    public void save(User user) {
        System.out.println("user start");    //在方法前添加逻辑
        userDAOImpl.save(user);
    }
}
```

二、使用动态代理添加逻辑

将所需要添加的逻辑封装成一个类，让其继承InvocationHandler接口

```
public class LogInterceptor implements InvocationHandler {
    private Object target;  //需要添加逻辑的对象
    public LogInterceptor(Object target) {
        this.target = target;
    }

    public void log(Method m) {
        System.out.println(m.getName() + " start");
    }

     /*
      * proxy 代理对象
      * method 代理对象要执行的方法
      * args 所需执行方法的参数
      */
     public Object invoke(Object proxy, Method method, Object[] args)
             throws Throwable {
         log(method);    //在执行方法前添加逻辑
         method.invoke(target , args);
         return null;
    }
}
```

使用Proxy获得代理对象，代理对象与被代理对象要实现同样的接口

```
public class TestProxy {
    /*
     * userDAO 要代理的对象，实现了UserDAO接口
     * li 给代理添加逻辑的类
     * proxy 通过Proxy的静态方法生成的代理对象
     * 代理对象与被代理的对象的ClassLoader要一样
     * 通过传入的参数，实现与代理对象同样的接口
```

```
8.      */
9.     public void testProxy() {
10.         //new被代理对象
11.         UserDAO userDAO = new UserDAOImpl();
12.         //将被代理对象交给代理
13.         LogInterceptor li = new LogInterceptor(userDAO);
14.         //Proxy.newProxyInstance产生代理对象
15.         UserDAO proxy = (UserDAO)Proxy.newProxyInstance(
16.                 userDAO.getClass().getClassLoader(),
17.                     UserDAO.class.getInterfaces(), li);
18.         proxy.save(new User());
19.     }
20. }
```