参考：

https://www.cnblogs.com/daniels/p/8242592.html

https://blog.csdn.net/Dome /article/details/82427386

https://blog.csdn.net/lsgqjh/article/details/68486433

# 前言

代理模式算是一个很重要的模式，Spring的一个核心AOP机制的核心就是代理，然后代理又分为静态代理和动态代理，其中动态代理又涉及到Java一个核心技术反射的运用，所以把代理模式好好理解理解，向上能去理解AOP，向下能去理解反射。

该篇也不将仅仅围绕代理模式本身去讲，借由动态代理也会酌情去介绍下反射，同时也会去将代理结合到Spring的AOP去理解一番。

# 静态代理

其实静态代理现在不推荐用也没谁去用，但要了解代理模式，从静态代理开始才是最佳的开门方式。

代理干了啥事呢，要是用现实中一个职业去描述可能比较容易理解，中介，它差不多就是干了中介的活。比如我要买房，我可以自己去买房，不过买房各种手续非常的繁琐，我工作繁忙又没时间，所以就用到中介，那我买房的这个过程就由我自己去头疼好长时间转换为，我向中介描述的我的房屋需求以及付钱，然后我只需要安心等中介给我处理完。代理就是这么回事，由代理类去处理类本身不想处理的事情，最后的结果就是类本身做的事情做完了连带不想做的事情也做完了。

静态代理算是代理实现的一种早期手段，为啥说不推荐使用呢，因为每个代理类只绑定一个接口，换句话说，如果通过静态代理去实现代

理，我们的代码里会遍布代理类，这显然是极简主义者不能接受的。为啥又说静态代理是一个不错的敲门砖呢，正因为它的不绕弯子绑定接口直接实现代理，整个过程非常清晰，让初学者容易先建立代理模式代码体现的体系。

好了，废话不多说，我模拟了一个静态代理的实现。看代码。

```
public interface UserDao {

	void add();
}
```

```
public class UserDaoImpl implements UserDao {

	@Override
	public void add() {
		System.out.println("加一个用户");
	}

}
```

```
public class UserProxy implements UserDao {

	private UserDao userDao;

	public UserProxy(UserDao userDao) {
		this.userDao = userDao;
	}

	@Override
	public void add() {
		System.out.println("加前准备");
		userDao.add();
		System.out.println("加后记录");
	}

	public static void main(String[] args) {
		UserDao userDao = new UserDaoImpl();

		UserProxy userProxy = new UserProxy(userDao);
		userProxy.add();
	}

}
```

一个很简单的模拟，把UserDaoImpl想象成客户A，客户A现在要加一个用户，但是加用户可能宏观上还有很多准备工作，客户A并不想干，

把UserProxy想象成中介，中介能打包票客户A你授权我帮你加客户，一条龙服务帮你做完。至于同时实现了的UserDao接口可以想象成泛称的客户这个群体。

## JDK动态代理

见分支

## CGLIB动态代理

见分支