Spring AOP JDKDynamicProxy动态代理，CGLib动态代理分析与总结

JDK动态代理：基于接口代理

1. 代理设计模式会有三个角色：抽象角色（代理类实现了被代理类相同的接口）
2. 代理角色：对抽象类实例某一个方法的调用会转移到对被代理实例的调用
3. 真实角色： 被代理类的真实实例

JDK代理会有以下及步骤：

1. 首先实现一个InvocationHandler 传入真正被代理的实例；通过该类的invoke（Object proxy, Method method, Object[] args）方法反射调被代理的方法上，并且阔以在调用的前后做增强。
2. 通过Proxy类的newProxyInstance(ClassLoader loader, Class<?> []interfaces ,InvocationHandler handler) 静态方法去实现一个代理类，并且被指定的classLoader进行加载，通过反射的方式进行对该代理类实例化；可以用指定接口中的某一个接口进行向下转型。
3. 通过返回的代理类的实例调用某一个方法，例如subject.request()会转向到InvocationHandler中的invoke方法，该InvocationHandler 实例会在代理类实例化时传入到构造方法中。

通过对程序进行系统参数的设置，可以将生成的代理类的字节码保存到本地（ProxyGenerator类中动态生成字节码），通过分析它会有如下的结构：

静态变量成员有3+n 个Method：n表示代理接口的n个方法；这些方法都是通过class.forName放射得到方法;另外3个表示反射object的toString ， hanshCode ,equals这三个方法；最终所有方法的执行都会转向到invoke方法反射去调用被代理类的方法。

CGLib动态代理的方式：