**3.2.3  JDK生成AopProxy代理对象**

我们都知道对于AopProxy代理对象，可以由JDK或CGLIB来生成，而JdkDynamicAopProxy和Cglib2AopProxy实现的都是AopProxy接口，它们的层次关系如图3-9所示。

|  |
| --- |
| <http://images.51cto.com/files/uploadimg/20091222/150436303.jpg> |
|  |

我们先看看在AopProxy接口实现中，JdkDynamicAopProxy是怎样完成AopProxy代理对象生成工作的。这个代理对象的生成过程如代码清单3-16所示。在JdkDynamicAopProxy中，使用了JDK的Proxy类来生成代理对象，在生成Proxy对象之前，首先需要从advised对象中取得代理对象的代理接口配置，然后调用Proxy的newProxyInstance方法，最终得到对应的Proxy代理对象；在生成代理对象的时候，需要指明三个参数，分别是类装载器、代理接口和Proxy回调方法所在的对象，这个对象需要实现InvocationHandler接口；InvocationHandler接口定义了invoke方法，提供代理对象的回调入口；对于JdkDynamicAopProxy，它本身实现了InvocationHandler接口和invoke方法，这个invoke方法是Proxy代理对象的回调方法，所以可以使用this来把JdkDynamicAopProxy指派给Proxy对象；也就是说JdkDynamicAopProxy对象本身，在Proxy代理的接口方法被调用时，会促发invoke方法的回调。在这个回调方法里，完成了AOP编织实现的封装。在这里，我们先重点关注AopProxy代理对象的生成，Proxy代理对象的invoke实现，将是我们后面详细分析AOP实现原理的重要部分。

代码清单3-16  JdkDynamicAopProxy生成Proxy代理对象

1. **public** Object getProxy(ClassLoader classLoader) {
2. **if** (logger.isDebugEnabled()) {
3. logger.debug("Creating JDK dynamic proxy:
4. target source is " + **this**.advised.
5. getTargetSource());
6. }
7. Class[] proxiedInterfaces = AopProxyUtils.
8. completeProxiedInterfaces(**this**.advised);
9. findDefinedEqualsAndHashCodeMethods(proxiedInterfaces);
10. //这里调用JDK生成Proxy的地方。
11. **return** Proxy.newProxyInstance
12. (classLoader, proxiedInterfaces, **this**);
13. }