



北京圣思园科技有限公司
<http://www.shengsiyuan.com>

主讲人：张龙

代理模式

- 代理模式的作用是：为其他对象提供一种代理以控制对这个对象的访问。
- 在某些情况下，一个客户不想或者不能直接引用另一个对象，而代理对象可以在客户端和目标对象之间起到中介的作用



代理模式

- 代理模式一般涉及到的角色有
 - 抽象角色：声明真实对象和代理对象的共同接口
 - 代理角色：代理对象角色内部含有对真实对象的引用，从而可以操作真实对象，同时代理对象提供与真实对象相同的接口以便在任何时刻都能代替真实对象。同时，代理对象可以在执行真实对象操作时，附加其他的操作，相当于对真实对象进行封装
 - 真实角色：代理角色所代表的真实对象，是我们最终要引用的对象



代理模式

- 参见程序Subject.java
- 参见程序RealSubject.java
- 参见程序ProxySubject.java
- 参见程序Client.java



代理模式

- 由以上代码可以看出，客户实际需要调用的是 **RealSubject** 类的 **request()** 方法，现在用 **ProxySubject** 来代理 **RealSubject** 类，同样达到目的，同时还封装了其他方法(**preRequest()**, **postRequest()**)，可以处理一些其他问题。
- 另外，如果要按照上述的方法使用代理模式，那么真实角色必须是事先已经存在的，并将其作为代理对象的内部属性。但是实际使用时，一个真实角色必须对应一个代理角色，如果大量使用会导致类的急剧膨胀；此外，如果事先并不知道真实角色，该如何使用代理呢？这个问题可以通过 **Java** 的动态代理类来解决



动态代理类

- Java动态代理类位于java.lang.reflect包下，一般主要涉及到以下两个类：
- (1)Interface InvocationHandler: 该接口中仅定义了一个方法
 - public Object invoke(Object obj,Method method, Object[] args)
- 在实际使用时，第一个参数obj一般是指代理类，method是被代理的方法，如上例中的request()，args为该方法的参数数组。这个抽象方法在代理类中动态实现。
- (2)Proxy: 该类即为动态代理类，作用类似于上例中的ProxySubject，其中主要包含以下内容



动态代理类

- `protected Proxy(InvocationHandler h)`: 构造函数，用于给内部的`h`赋值。
- `static Class getProxyClass (ClassLoader loader, Class[] interfaces)`: 获得一个代理类，其中`loader`是类装载器，`interfaces`是真实类所拥有的全部接口的数组。
- `static Object newProxyInstance(ClassLoader loader, Class[] interfaces, InvocationHandler h)`: 返回代理类的一个实例，返回后的代理类可以当作被代理类使用(可使用被代理类的在`Subject`接口中声明过的方法)



动态代理类

- 所谓Dynamic Proxy是这样一种class: 它是在运行时生成的class, 在生成它时你必须提供一组interface给它, 然后该class就宣称它实现了这些 interface。你当然可以把该class的实例当作这些interface中的任何一个来用。当然, 这个Dynamic Proxy其实就是一个Proxy, 它不会替你作实质性的工作, 在生成它的实例时你必须提供一个handler, 由它接管实际的工作



动态代理类

- 在使用动态代理类时，我们必须实现 `InvocationHandler` 接口
- 参见程序 `Subject.java`
- 参见程序 `RealSubject.java`
- 参见程序 `DynamicSubject.java`
- 参见程序 `Client.java`



动态代理类

- 通过这种方式，被代理的对象 (**RealSubject**) 可以在运行时动态改变，需要控制的接口 (**Subject** 接口) 可以在运行时改变，控制的方式 (**DynamicSubject** 类) 也可以动态改变，从而实现了非常灵活的动态代理关系



动态代理

- 动态代理是指客户通过代理类来调用其它对象的方法
- 动态代理使用场合：
 - 调试
 - 远程方法调用(RMI)



动态代理



动态代理步骤

1. 创建一个实现接口 `InvocationHandler` 的类，它必须实现 `invoke` 方法

2. 创建被代理的类以及接口

3. 通过 `Proxy` 的静态方法

`newProxyInstance(ClassLoader loader, Class[] interfaces, InvocationHandler h)` 创建一个代理

4. 通过代理调用方法



程序实践

- 参见程序VectorProxy.java
- 参见程序Foo.java
- 参见程序FooImpl.java
- 参见程序FooImpl2.java
- 参见程序
CommonInvocationHandler.java
- 参见程序Demo.java

