**Java中的动态代理**

# 什么是动态代理？

所谓代理，通俗的说代理其实就是代替别人干活。

在开发过程中，如果不想让别人直接访问某个业务对象，就需要一个代理对象，拦截对这个业务对象的访问，并且这个代理对象能提供这个业务对象相同的功能。动态代理技术就是用来产生一个对象的代理对象的。动态代理其实是编程语言提供的一种语法，能够将对对象中不同方法的调用重定向到一个统一的处理函数中来，做自定义的逻辑处理，但是对于调用者透明。

那为什么叫‘动态’代理呢？，这是因为这种语法产生的代理类是和业务无关的，它不关心它所代理的类到底有多少个方法，即使被代理类后来增加了方法，这个代理类也能够自动感知，无需修改代理类的代码。实际上，被代理的类不一定位于本机类，动态代理语法提供了一种抽象方式，被代理的类也可以位于远程主机上，这也是RPC框架实现原理的一部分。

理解了动态代理的概念后，可以发现动态代理有这么两个部分：

1. 给调用者使用的代理类(有可能是接口)
2. 一个统一处理函数，收集不同函数转发过来的请求，可自定义处理逻辑集中处理。Java中它可能会成为一个较为独立的部分，因此也可能是类。

例如，python通过重写\_\_getattribute\_\_方法可以实现动态代理：

# Java动态代理

下面看看Java动态代理机制如何代理一个本地对象

## 代理接口

Java中实现动态代理，必须定义接口才能实现目标对象的方法增强。

interface IHello {  
 void say(String s);  
}

## 被代理类

class RealHello implements IHello {  
  
 @Override  
 public void say(String s) {  
 System.*out*.println("hello " + s);  
 }  
  
}

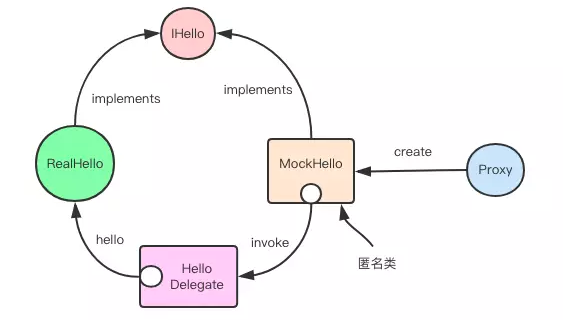
## 代理增强器

class HelloProxyHandler implements InvocationHandler {  
  
 private IHello target; // 原始对象  
  
 public HelloProxyHandler(IHello target) {  
 this.target = target;  
 }  
  
 @Override  
 public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
 */\*\*  
 \* 这里可以任意附加一些逻辑  
 \*/* System.*out*.println("before print");  
 method.invoke(target, args); // 调用原始对象的方法  
 System.*out*.println("after print");  
 return null;  
 }  
  
}

## 创建代理对象

public static IHello enhanceHello(IHello target) {  
 return (IHello) Proxy.*newProxyInstance*(DynamicProxyTest.class.getClassLoader(), new Class<?>[]{IHello.class},  
 new HelloProxyHandler(target));  
}

以上各对象之间的关系可以用下图表示：



我们调用Proxy.newProxyInstance产生了一个匿名类实例，该实例同样实现了IHello接口，它的作用就是用来替代原生的RealHello实例。这个匿名实例持有HelloProxyHandler实例的引用，当你对这个匿名实例进行方法调用时，它会将调用逻辑委托给HelloProxyHandler实例的invoke方法。HelloProxyHandler实例内部又持有原生RealHello对象的引用，所以用户就可以在invoke方法里实现任意附加逻辑，以及对原生RealHello对象的调用。

# 动态代理应用

## 代理模式/装饰器模式的动态实现

## 实现AOP

使用动态代理类替换被代理类的使用场景， 并在处理函数中插入自定义的钩子，这就相当于对该类的所有方法定义了一个切面。

## 实现RPC

RPC的实现上，它将函数调用方和函数的提供方分散在两个不同的进程上，中间使用网络通信来进行数据交互，RPC调用其实是对远程另外一台机器进程上的对象的代理

仔细思考RPC调用的数据流流向，就能梳理出这样的思路：

1. 调用方调用本地的RPC代理方法，将参数提供给该方法。
2. 不同的RPC代理方法被转发到一个统一的处理中心，该处理中心知道调用的是那个函数，参数是什么。
3. 该处理中心将调用的信息封装打包，通过网络发送给另外一个进程。
4. 另外一个进程接受到调用进程发送过来的数据包。
5. 该进程根据数据包中记录的RPC调用信息，将调用分发给对应的被代理对象的对应方法去执行。
6. 返回的话思路类似。

显而易见，第二步，需要使用动态代理将分散的函数调用转发到一个统一的处理中心；第五步，将统一收集来的调用信息分发给具体的函数执行，显然使用反射做到这一点。  
有了这个思路，通过利用动态代理，反射，和网络编程技术，实现一个简易版的RPC框架也就不难了。

# 最后

动态代理的核心在于：将分散的对对象不同方法的调用转发到一个同一的处理函数中来。