用Java动态代理实现AOPJava认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022__E7_94_A8Java_E5_8A_A8_c104_644625.htm 目前整个开发社区对AOP(Aspect Oriented Programing)推崇备至，也涌现出大量支持AOP的优秀Framework,--Spring, JAC, Jboss AOP 等等。AOP似乎一时之间成了潮流。Java初学者不禁要发出感慨，OOP还没有学通呢，又来AOP。本文不是要在理论上具体阐述何为AOP, 为何要进行AOP . 要详细了解学习AOP可以到它老家http://aosd.net去瞧瞧。这里只是意图通过一个简单的例子向初学者展示一下如何来进行AOP. 为了简单起见，例子没有没有使用任何第三方的AOP Framework, 而是利用Java语言本身自带的动态代理功能来实现AOP. 让我们先回到AOP本身，AOP主要应用于日志记录，性能统计，安全控制,事务处理等方面。它的主要意图就要将日志记录，性能统计，安全控制等等代码从商业逻辑代码中清楚的划分出来，我们可以把这些行为一个一个单独看作系统所要解决的问题，就是所谓的面向问题的编程(不知将AOP译作面向问题的编程是否欠妥)。通过对这些行为的分离，我们希望可以将它们独立地配置到商业方法中，而要改变这些行为也不需要影响到商业方法代码。 假设系统由一系列的BusinessObject所完成业务逻辑功能，系统要求在每一次业务逻辑处理时要做日志记录。这里我们略去具体的业务逻辑代码。 public interface BusinessInterface { public void processBusiness(). } public class BusinessObject implements BusinessInterface { private Logger logger = Logger.getLogger(this.getClass().getName()). public void

processBusiness(){ try { logger.info("start to processing..."). //business logic here. System.out.println(“ here is business logic ”). logger.info("end processing..."). } catch (Exception e){ logger.info("exception happends..."). //exception handling } } } 这里处理商业逻辑的代码和日志记录代码混合在一起，这给日后的维护带来一定的困难，并且也会造成大量的代码重复。完全相同的log代码将出现在系统的每一个BusinessObject中。按照AOP的思想，我们应该把日志记录代码分离出来。要将这些代码分离就涉及到一个问题，我们必须知道商业逻辑代码何时被调用，这样我们好插入日志记录代码。一般来说要截获一个方法，我们可以采用回调方法或者动态代理。动态代理一般要更加灵活一些，目前多数的AOP Framework也大都采用了动态代理来实现。这里我们也采用动态代理作为例子。 JDK1.2以后提供了动态代理的支持，程序员通过实现java.lang.reflect.InvocationHandler接口提供一个执行处理器，然后通过java.lang.reflect.Proxy得到一个代理对象，通过这个代理对象来执行商业方法,在商业方法被调用的同时，执行处理器会被自动调用。 有了JDK的这种支持，我们所要做的仅仅是提供一个日志处理器。 public class LogHandler implements InvocationHandler { private Logger logger ＝ Logger.getLogger(this.getClass().getName()). private Object delegate. public LogHandler(Object delegate){ this.delegate = delegate. } 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com