Java设计模式之Decorator模式Java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_Java_E8_AE_BE_E8_AE_A1_c104_644807.htm JDK为程序员提供了大量的类库，而为了保持类库的可重用性，可扩展性和灵活性，其中使用到了大量的设计模式，本文将介绍JDK的I/O包中使用到的Decorator模式，并运用此模式，实现一个新的输出流类。 Decorator模式简介 Decorator模式又名包装器(Wrapper)，它的主要用途在于给一个对象动态的添加一些额外的职责。与生成子类相比，它更具有灵活性。 有时候，我们需要为一个对象而不是整个类添加一些新的功能，比如，给一个文本区添加一个滚动条的功能。我们可以使用继承机制来实现这一功能，但是这种方法不够灵活，我们无法控制文本区加滚动条的方式和时机。而且当文本区需要添加更多的功能时，比如边框等，需要创建新的类，而当需要组合使用这些功能时无疑将会引起类的爆炸。 我们可以使用一种更为灵活的方法，就是把文本区嵌入到滚动条中。而这个滚动条的类就相当于对文本区的一个装饰。这个装饰(滚动条)必须与被装饰的组件(文本区)继承自同一个接口，这样，用户就不必关心装饰的实现，因为这对他们来说是透明的。装饰会将用户的请求转发给相应的组件(即调用相关的方法)，并可能在转发的前后做一些额外的动作(如添加滚动条)。通过这种方法，我们可以根据组合对文本区嵌套不同的装饰，从而添加任意多的功能。这种动态的对对象添加功能的方法不会引起类的爆炸，也具有了更多的灵活性。 以上的方法就是Decorator模式，它通过给对象添加装饰来动态的添加新的功能。 Component

为组件和装饰的公共父类，它定义了子类必须实现的方法。ConcreteComponent是一个具体的组件类，可以通过给它添加装饰来增加新的功能。 Decorator是所有装饰的公共父类，它定义了所有装饰必须实现的方法，同时，它还保存了一个对于Component的引用，以便将用户的请求转发给Component，并可能在转发请求前后执行一些附加的动作。ConcreteDecoratorA和ConcreteDecoratorB是具体的装饰，可以使用它们来装饰具体的Component。 Java IO包中的Decorator模式 JDK提供的java.io包中使用了Decorator模式来实现对各种输入输出流的封装。以下将以java.io.OutputStream及其子类为例，讨论一下Decorator模式在IO中的使用。 首先来看一段用来创建IO流的代码： 以下是代码片段： try { DataOutputStream out = new DataOutputStream(new FileOutputStream("test.txt")). } catch (FileNotFoundException e) { e.printStackTrace(). } 这段代码对于使用过JAVA输入输出流的人来说再熟悉不过了，我们使用DataOutputStream封装了一个FileOutputStream。这是一个典型的Decorator模式的使用，FileOutputStream相当于 Component，DataOutputStream就是一个Decorator。将代码改成如下，将会更容易理解： 以下是代码片段： try { OutputStream out = new FileOutputStream("test.txt"). out = new DataOutputStream(out). } catch(FileNotFoundException e) { e.printStatckTrace(). } 由于FileOutputStream和DataOutputStream有公共的父类OutputStream，因此对对象的装饰对于用户来说几乎是透明的。下面就来看看OutputStream及其子类是如何构成Decorator模式的： OutputStream是一个抽象类，它是所有输出流的公

共父类，其源代码如下： 以下是代码片段： 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