使用内部和匿名类优化Java代码 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/493/2021_2022__E4_BD_BF_E7_94_A8_E5_86_85_E9_c67_493667.htm Java 1.1通过对Java语言规范进行修改，显著简化了一些实用结构的实现。在那些修改中，最引人注目的就是内部类和匿名类。如运用得当，它们可使程序更易理解和维护。下面来看看这些特性具体是如何工作的，如何正确使用它们，以及如何避免一些常见的错误。 内部类 简单地说，“ 内部类 ”是在另一个类的内部声明的类。从Java 1.1开始，你可在一个类中声明另一个类，这与声明字段和方法非常相似。包装了内部类声明的类就称为“ 外部类 ”。 实际上，Java语言规范还允许你做更多的事情，包括： 在另一个类或者一个接口中声明一个类。 在另一个接口或者一个类中声明一个接口。 在一个方法中声明一个类。 类和接口声明可嵌套任意深度。 清单A是类和接口的一些空白声明，它演示了这些可能性。 使用一个import语句，你可像使用其他任何标准类那样省略package名称。此外，在外部类中，可利用简单名称来引用所有内部类和接口（参见清单A中的new语句）。注意从Method1中引用Inner2仍需指定Interface1，因为Inner2在一个不同的级别上。 表A总结了清单A中声明的每个内部类和接口的完全限定名称。用了import语句之后，就可采用较短的形式。当然，在外部类中，你还可省略外部类的名称。 名称 类/接口 Inner1 mypackage.Inner1 Interface1 mypackage.Interface1 Inner2 mypackage.Interface1.Inner2 Interface2 mypackage.Interface1.Interface2 Inner3 Inner3对于Method1来说

是local的，所以它不可在方法外部访问 引用内部类 内部类最自然的一种应用就是声明只在另一个类的内部使用的类，或者声明与另一个类密切相关的类。如清单B所示，它是一个链表的简单实现。由于Node类通常只在LinkedList的范围内使用，所以最好将Node声明为LinkedList的一个内部类。 适用于类成员的访问控制修改符也适用于内部类；也就是说，内部类可以具有package、protected、private和public访问权限，它们的语义和正常的语义没有什么不同。由于Node要在LinkedList的外部使用，所以把它声明为public。 然而，修饰符static具有不同的含义。应用于内部类时，它声明的类具有与其他类相同的语义，也就是可进行实例化，并像一个标准类那样使用。惟一的区别就是它拥有对外部类的所有静态成员的完全访问权限。清单C展示了一个简单的程序，它创建一个链表，并将它打印到标准输出设备。 非静态内部类 如果内部类没有指定static修饰符，就拥有对外部类的所有成员的完全访问权限，包括实例字段和方法。为实现这一行为，非静态内部类存储着对外部类的实例的一个隐式引用。 所以，对一个非静态内部类进行实例化需要采用不同语法的new语句： .new 这种形式的new语句要求外部类的一个实例，使内部类能在那个实例的上下文中创建。注意清单A声明了几个非静态内部类，并用标准的new语句在Method1中实例化它们。之所以能那样做，是因为Method1是外部类的一个实例方法，所以new语句会在外部类的一个实例的上下文中隐式地执行。只有在外部类的外部或者在其他对象的上下文中实例化一个非静态内部类时，才需要使用修改过的语法。 但是，非静态内部类具有一些限制。尤其是，它们不能声明静态初始化列表和静态

成员，除非是在常量字段中。此外，方法内部声明的内部类不能访问方法的局部变量和参数，除非它们被初始化成final。 匿名类 匿名类是不能有名称的类，所以没办法引用它们。必须在创建时，作为new语句的一部分来声明它们。 这就要采用另一种形式的new语句，如下所示： new 这种形式的new语句声明一个新的匿名类，它对一个给定的类进行扩展，或者实现一个给定的接口。它还创建那个类的一个新实例，并把它作为语句的结果而返回。要扩展的类和要实现的接口是new语句的操作数，后跟匿名类的主体。 如果匿名类对另一个类进行扩展，它的主体可以访问类的成员、覆盖它的方法等等，这和其他任何标准的类都是一样的。如果匿名类实现了一个接口，它的主体必须实现接口的方法。 注意匿名类的声明是在编译时进行的，实例化在运行时进行。这意味着for循环中的一个new语句会创建相同匿名类的几个实例，而不是创建几个不同匿名类的一个实例。 从技术上说，匿名类可被视为非静态的内部类，所以它们具有和方法内部声明的非静态内部类一样的权限和限制。 如果要执行的任务需要一个对象，但却不值得创建全新的对象（原因可能是所需的类过于简单，或者是由于它只在一个方法内部使用），匿名类就显得非常有用。匿名类尤其适合在Swing应用程序中快速创建事件处理程序。 清单D就是一个非常简单的Swing应用程序，它展示了与匿名类有关的几个概念。这个例子创建了两个匿名类。第一个对java.awt.event.WindowAdapter进行扩展，并在应用程序窗口关闭时调用应用程序的onClose方法。 即使onClose声明为private，匿名类也能调用它，因为匿名类本质上是应用程序类的一个内部类。第二个匿名类实现

了java.awt.ActionListener接口，它在一个按钮被按下后关闭应用程序窗口。注意匿名类可以访问本地变量frame。这是由于匿名类在与frame相同的方法内部声明。然而，frame要被声明为final，否则会生成编译错误。 更优化的代码 内部和匿名类是Java 1.1为我们提供的两个出色的工具。它们提供了更好的封装，结果就是使代码更容易理解和维护，使相关的类都能存在于同一个源代码文件中（这要归功于内部类），并能避免一个程序产生大量非常小的类（这要归功于匿名类）。