关于java自动装箱与拆箱的分析 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/284/2021_2022__E5_85_B3_E4_BA_8Ejava_c104_284974.htm 自动装箱与拆箱的功能事实上是编译器来帮您的忙，编译器在编译时期依您所编写的语法，决定是否进行装箱或拆箱动作。例如： Integer i = 100.相当于编译器自动为您作以下的语法编译： Integer i = new Integer(100).所以自动装箱与拆箱的功能是所谓的“ 编译器蜜糖 ”(Compiler Sugar)，虽然使用这个功能很方便，但在程序运行阶段您得了解Java的语义。例如下面的程序是可以通过编译的： Integer i = null.int j = i.这样的语法在编译时期是合法的，但是在运行时期会有错误，因为这种写法相当于： Integer i = null.int j = i.intValue().null表示i没有参考至任何的对象实体，它可以合法地指定给对象参考名称。由于实际上i并没有参考至任何的对象，所以也就不可能操作intValue()方法，这样上面的写法在运行时会出现NullPointerException错误。 自动装箱、拆箱的功能提供了方便性，但隐藏了一些细节，所以必须小心。再来看范例4.6，您认为结果是什么呢？ Uuml. 范例4.7 AutoBoxDemo3.java public class AutoBoxDemo3 { public static void main(String[] args) { Integer i1 = 200. Integer i2 = 200. if (i1 == i2) System.out.println("i1 == i2"). else System.out.println("i1 != i2"). } } 结果是显示i1 != i2，这有些令人惊讶，两个范例语法完全一样，只不过改个数值而已，结果却相反。 其实这与==运算符的比较有关，在第3章中介绍过==是用来比较两个基本数据类型的变量值是否相等，事实上==也用于判断两个对象引用名称是否参考至同一个对象。 在自动装箱时对于

值从128到127之间的值，它们被装箱为Integer对象后，会存在内存中被重用，所以范例4.6中使用==进行比较时，i1 与 i2实际上参考至同一个对象。如果超过了从128到127之间的值，被装箱后的Integer对象并不会被重用，即相当于每次装箱时都新建一个Integer对象，所以范例4.7使用==进行比较时，i1与i2参考的是不同的对象。 所以不要过分依赖自动装箱与拆箱，您还是必须知道基本数据类型与对象的差异。范例4.7最好还是依正规的方式来写，而不是依赖编译器蜜糖(Compiler Sugar)。例如范例4.7必须改写为范例4.8才是正确的。 &Uuml. 范例4.8 AutoBoxDemo4.java public class AutoBoxDemo4 { public static void main(String[] args) { Integer i1 = 200. Integer i2 = 200. if (i1.equals(i2)) System.out.println("i1 == i2"). else System.out.println("i1 != i2"). } } 结果这次是显示i1 == i2。使用这样的写法，相信也会比较放心一些，对于这些方便但隐藏细节的功能到底要不要用呢？基本上只有一个原则：如果您不确定就不要用。 