Java库谜题61：日期游戏 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/138/2021_2022_Java_E5_BA_93_E8_B0_9C_c97_138638.htm 下面的程序演练了Date和Calendar类的某些基本特性，它会打印出什么呢？ import java.util.*. public class DatingGame { public static void main(String[] args) { Calendar cal = Calendar.getInstance(). cal.set(1999, 12, 31). // Year, Month, Day System.out.print(cal.get(Calendar.YEAR) " "). Date d = cal.getTime(). System.out.println(d.getDay()). } } 该程序创建了一个Calendar实例，它应该表示的是1999年的除夕夜，然后该程序打印年份和日。看起来该程序应该打印1999 31，但是它没有；它打印的是2000 1。难道这是致命的Y2K(千年虫)问题吗？ 不，事情比我们想象的要糟糕得多：这是致命的Date/Calendar问题。在Java平台首次发布时，它唯一支持日历计算类的就是Date类。这个类在能力方面是受限的，特别是当需要支持国际化时，它就暴露出了一个基本的设计缺陷：Date实例是易变的。在1.1版中，Calendar类被添加到了Java平台中，以矫正Date的缺点，由此大部分的Date方法就都被弃用了。遗憾的是，这么做只能使情况更糟。我们的程序说明Date和Calendar API有许多问题。 该程序的第一个bug就位于方法调用cal.set(1999,12,31)中。当月份以数字来表示时，习惯上我们将第一个月被赋值为1。遗憾的是，Date将一月表示为0，而Calendar延续了这个错误。因此，这个方法调用将日历设置到了1999年第13个月的第31天。但是标准的（西历）日历只有12个月，该方法调用肯定应该抛出一个IllegalArgumentException异常，对吗？它是应该这么做，但

是它并没有这么做。Calendar类直接将其替换为下一年，在本例中即2000年的第一个月。这也就解释了我们的程序为什么打印出的第一个数字是2000。 有两种方法可以订正这个问题。你可以将cal.set调用的第二个参数由12改为11，但是这么做容易引起混淆，因为数字11会让读者误以为是11月。更好的方式是使用Calendar专为此目的而定义的常量，即Calendar.DECEMBER。 该程序打印出的第二个数字又是怎么回事呢？cal.set调用很明显是要把日历设置到这个月的第31天，Date实例d表示的是与Calendar相同的时间点，因此它的getDay方法应该返回31，但是程序打印的却是1，这是怎么搞得呢？ 为了找出原因，你必须先阅读一下文档，它叙述道Date.getDay返回的是Date实例所表示的星期日期，而不是月份日期。这个返回值是基于0的，从星期天开始计算。因此程序所打印的1表示2000年1月31日是星期一。请注意，相应的Calendar方法get(Calendar.DAY_OF_WEEK) 不知为什么返回的是基于1的星期日期值，而不是像Date的对应方法那样返回基于0的星期日期值。 有两种方法可以订正这个问题。你可以调用Date.date这一名字极易让人混淆的方法，它返回的是月份日期。然而，与大多数Date方法一样，它已经被弃用了，因此你最好是将Date彻底抛弃，直接调用Calendar的get(Calendar.DAY_OF_MONTH)方法。用这两种方法，该程序都可以打印出我们想要的1999 31： public class DatingGame { public static void main(String[] args) { Calendar cal = Calendar.getInstance(). cal.set(1999, Calendar.DECEMBER, 31). System.out.print(cal.get(Calendar.YEAR) " "). System.out.println(cal.get(Calendar.DAY_OF_MONTH)). } } 本

谜题只是掀开了Calendar和Date缺陷的冰山一角。这些API简直就是雷区。Calendar其他的严重问题包括弱类型（几乎每样事物都是一个int）、过于复杂的状态空间、拙劣的结构、不一致的命名以及不一致的雨衣等。在使用Calendar和Date的时候一定要当心，千万要记着查阅API文档。 对API设计者来说，其教训是：如果你不能在第一次设计时就使它正确，那么至少应该在第二次设计时应该使它正确，绝对不能留到第三次设计时去处理。如果你对某个API的首次尝试出现了严重问题，那么你的客户可能会原谅你，并且会再给你一次机会。如果你第二次尝试又有问题，你可能会永远坚持这些错误了。 