改进JAVA字符串分解的方法 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/273/2021_2022__E6_94_B9_E8_BF_9BJAVA_c104_273479.htm 一、概述 大多数Java程序员都曾经使用过java.util.StringTokenizer类。它是一个很方便的字符串分解器，主要用来根据分隔符把字符串分割成标记（Token），然后按照请求返回各个标记。 这个过程称为Tokenization，实际上就是把字符序列转换成应用程序能够理解的多个标记。 虽然StringTokenizer用起来很方便，但它的功能却很有限。这个类只是简单地在输入字符串中查找分隔符，一旦找到了分隔符就分割字符串。它不会检查分隔符是否在子串之中这类条件，当输入字符串中出现两个连续的分隔符时，它也不会返回""（字符串长度为0）形式的标记。 为了突破这些局限，Java 2平台提供了BreakIterator类，它是在StringTokenizer之上改进的字符串分解器。由于JDK 1.1.x没有提供这个类，为了满足自己的需要，开发者经常花费很多时间从头开始编写分解器。在涉及到数据格式化处理的大型工程中，这类定制的字符串分解器有时随处可见，而且这种情况并不罕见。本文的目标是帮助你利用现有的StringTokenizer类，编写一个高级字符串分解器。 二、StringTokenizer的局限 你可以用以下三种构造函数中的任意一种创建StringTokenizer分解器：StringTokenizer(String sInput)：以空白字符（“ ”，“\t”，“\n”）为分隔符分割字符串。 StringTokenizer(String sInput, String sDelimiter)：以sDelimiter为分隔符分割字符串。 StringTokenizer(String sInput, String sDelimiter, boolean bReturnTokens)：以sDelimiter为分隔符分割字符串，但如

果bReturnTokens为true，则分隔符也作为标记返回。 第一个构造函数不检查输入字符串是否包含子串。例如，如果以空白字符为分隔符分割“ hello. Today \"I am \" going to my home town ”，则字符串分解结果是hello.、Today、"I、am、"、going等，而不是hello.、Today、"I am "、going等。 第二个构造函数不检查两个分隔符连续出现的情况。例如，如果以“ , ”为分隔符分割“ book, author, publication,,,date published ”这个字符串，则StringTokenizer返回book、author、publication和date published这四个标记，而不是book、author、publication、""、""和date published这6个标记（其中""表示0长度字符串）。要得到6个标记的答案，你必须把StringTokenizer的bReturnTokens参数设置为true。 允许设置值为true的bReturnTokens参数是一个重要的功能，因为它考虑到了分隔符连续出现的情况。例如，使用第二个构造函数时，如果数据是动态收集得到而且要用来更新数据库中的表，输入字符串中的标记对应着表里面列的值，那么当我们不能确定哪一个列应该设置为""时，我们就无法把输入串中的标记映射到数据库列。假设我们要把记录插入到一个有6个列的表，而输入数据中包含两个连续的分隔符。此时，StringTokenizer的分解结果是5个标记（两个连续的分隔符代表""标记，它将被StringTokenizer忽略），而我们却有6个字段需要设置。同时，我们也不知道连续分隔符在哪里出现，所以也就不知道哪一个列应该设置成""。 当标记本身等同于分隔符（无论是长度还是值）且位于子串之内时，第三个构造函数无效。例如，如果我们要以“ , ”为分隔符分解字符串“ book, author, publication,\",\",date published ”（这个字符串

包含一个“ , ”标记，它与分隔符一样），结果是book、author、publication、"、"、date published这六个标记，而不是book、author、publication、,（逗号字符）、date published这五个标记。再提醒一下，即使我们把StringTokenizer的bReturnTokens参数设置设置成了true，在这种情况下也没有什么帮助。 三、高级字符串分解器 在编写代码之前，你必须搞清楚一个好的分解器有哪些基本要求。因为Java开发者已经习惯于使用StringTokenizer类，所以一个好的分解器应该提供StringTokenizer类提供的所有实用方法，比如hasMoreTokens()、nextToken()、countTokens()。 本文提供的代码很简单，而且大部分代码足以自我解释。在这里，我主要利用了StringTokenizer类（创建类实例时bReturnTokens参数设置为true），并提供了上面提到的几个方法。大多数时候标记与分隔符不同，有些时候分隔符却要作为标记输出（尽管非常罕见），此时如果出现了对标记的请求，分解器要把分隔符作为标记输出。创建PowerfulTokenizer对象时，你只需要提供输入字符串和分隔符这两个参数，PowerfulTokenizer将在内部使用bReturnTokens设置成true的StringTokenizer。（这么做的原因在于，如果不是用bReturnTokens设置成true的方式创建StringTokenizer，那么它将在解决先前提出的问题时受到限制）。为了正确地控制分解器，代码在几个地方（计算标记的总数量以及nextToken()）检查bReturnTokens是否设置成了true。 你可能已经发现，PowerfulTokenizer实现了Enumeration接口，从而也就实现了hasMoreElements()和nextElement()这两个方法，而这两个方法又分别把调用直接委托给hasMoreTokens()和nextToken()。（由于实现

了Enumeration接口，PowerfulTokenizer实现了与StringTokenizer的向后兼容。） 我们来看一个例子，假设输入字符串是“ hello, Today,,, \"I, am \", going to,,, \"buy, a, book\" ”，分隔符是“ , ”。用分解器分割这个字符串时返回结果如表1所示： 表1：字符串分解结果 输入字符串包含11个逗号（, ）字符，其中3个在子串里面、4个连续出现（“ Today,,, ”中包含两个连续逗号，第一个逗号是Today的分隔符）。下面是PowerfulTokenizer计算标记总数的算法： 如果bReturnTokens=true，把子串中的分隔符数量乘以2，再从实际总数量减去该数字，就得到了标记的总数。理由是，对于子串“ buy, a, book ”，StringTokenizer将返回5个标记（即“ buy:,:a:,:book ”），而PowerfulTokenizer将返回一个标记（即“ buy, a, book ”），两者的差值是4（即，2乘以子串中的分隔符数量）。这个公式对于所有包含分隔符的子串都有效。 类似地，对于bReturnTokens=false的情形，我们从实际总数（19）减去表达式[分隔符总数（11）- 连续分隔符数量（4）子串中的分隔符数量（3）]。由于这时我们不返回分隔符，它们（非连续出现或在子串内部）对我们来说没有用，上面的公式为我们返回了标记的总数量（9）。 请记住这两个公式，它们是PowerfulTokenizer的核心。这两个公式适用于几乎所有它们各自条件下的情形。但是，如果你有更复杂的要求，不能使用这两个公式，那么你应该在编写代码之前分析各种可能出现的情况，并设计出自己的公式。 