JAVA中的字符与代码点（2） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/354/2021_2022_JAVA_E4_B8_AD_E7_9A_84_c104_354067.htm 开放的增补字符：基于代码点的 API 新增的低层 API 分为两大类：用于各种char和基于代码点的表示法之间转换的方法和用于分析和映射代码点的方法。 最基本的转换方法是Character.toCodePoint（char high, char low）（用于将两个 UTF-16 代码单元转换为一个代码点）和Character.toChars（int codePoint）（用于将指定的代码点转换为一个或两个 UTF-16 代码单元，然后封装到一个char[]内。不过，由于大多数情况下文本以字符序列的形式出现，因此，另外提供codePointAt和codePointBefore方法，用于将代码点从各种字符序列表示法中提取出来：Character.codePointAt（char[] a， int index）和String.codePointBefore（int index）是两种典型的例子。在将代码点插入字符序列时，大多数情况下均有一些针对StringBuffer和StringBuilder类的appendCodePoint（int codePoint）方法，以及一个用于提取表示代码点的int[]的String构建器。 几种用于分析代码单元和代码点的方法有助于转换过程：Character 类中的isHighSurrogate和isLowSurrogate方法可以识别用于表示增补字符的char值；charCount（int codePoint）方法可以确定是否需要将某个代码点转换为一个或两个char.但是，大多数基于代码点的方法均能够对所有 Unicode 字符实现基于char的旧方法对 BMP 字符所实现的功能。以下是一些典型例子： Character.isLetter（int codePoint）可根据 Unicode 标准识别字母。 Character.isJavaIdentifierStart

（int codePoint）可根据 Java 语言规范确定代码点是否可以启动标识符。 Character.UnicodeBlock.of（int codePoint）可搜索代码点所属的 Unicode 字符子集。 Character.toUpperCase（int codePoint）可将给定的代码点转换为其大写等值字符。尽管此方法能够支持增补字符，但是它仍然不能解决根本的问题，即在某些情况下，逐个字符的转换无法正确完成。例如，德文字符“ "？" ”应该转换为“ SS ”，这需要使用String.toUpperCase方法。 注意大多数接受代码点的方法并不检查给定的int值是否处于有效的 Unicode 代码点范围之内（如上所述，只有 0x0 至 0x10FFFF 之间的范围是有效的）。在大多数情况下，该值是以确保其有效的方法产生的，在这些低层 API 中反复检查其有效性可能会对系统性能造成负面的影响。在无法确保有效性的情况下，应用程序必须使用Character.isValidCodePoint方法确保代码点有效。大多数方法对于无效的代码点采取的行为没有特别加以指定，不同的实现可能会有所不同。 API 包含许多简便的方法，这些方法可使用其他低层的 API 实现，但是专家组觉得，这些方法很常用，将它们添加到 J2SE 平台上很有意义。不过，专家组也排除了一些建议的简便方法，这给我们提供了一次展示自己实现此类方法能力的机会。例如，专家组经过讨论，排除了一种针对String类的新构建器（该构建器可以创建一个保持单个代码点的String）。以下是使应用程序使用现有的 API 提供功能的一种简便方法：/*** 创建仅含有指定代码点的新 String. */ String newString（int codePoint）{ return new String（Character.toChars（codePoint））；} 您会注意到，在这个简单的实现中，toChars方法始终创建一个中间数列，该数列

仅使用一次即立即丢弃。如果该方法在您的性能评估中出现，您可能会希望将其优化为针对最为普通的情况，即该代码点为 BMP 字符：/*** 创建仅含有指定代码点的新 String.* 针对 BMP 字符优化的版本。 */ String newString（int codePoint） { if （Character.charCount（codePoint） == 1） { return String.valueOf（（char） codePoint） ； } else { return new String （Character.toChars（codePoint）） ； }} 或者，如果您需要创建许多个这样的 string，则可能希望编写一个重复使用toChars方法所使用的数列的通用版本：/*** 创建每一个均含有一个指定* 代码点的新 String. * 针对 BMP 字符优化的版本。 */ String[] newStrings（int[] codePoints） { String[] result = new String[codePoints.length] ； char[] codeUnits = new char[2] ； for （int i = 0 ； i int count = Character.toChars（codePoints[i], codeUnits， 0） ； result[i] = new String（codeUnits， 0， count ） ； } return result ；} 不过，最终您可能会发现，您需要的是一个完全不同的解决方案。新的构建器String（int codePoint）实际上建议作为String.valueOf（char）的一个基于代码点的备选方案。在很多情况下，此方法用于消息生成的环境，例如：System.out.println（"Character " String.valueOf（char） " is invalid."） ； 新的格式化 API支持增补文字，提供一种更加简单的备选方案：System.out.printf（"Character %c is invalid.%n" ， codePoint） ； 使用此高层 API 不仅简捷，而它有很多特殊的优点：它可以避免串联（串联会使消息很难本地化），并将需要移进资源包 （resource bundle） 的字符串数量从两个减少到一个。