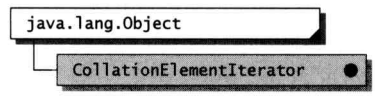
java.text

CollationElementIterator



语法

public final class CollationElementIterator extends Object

描述

CollationElementIterator类是一个迭代器，它依据一个特殊的Collator对象中的规则检查在对地区敏感的字符串中被分解的字符。

分解是把先前合并的字符，如一个字母“ a ”与其发音符对(a-grave)“ \u00E0 ”分解成一个统一码组合字符序列，如“ a\u0300 ”这种方式。该类基本上用于查找字符串。迭代器返回确定位置上的字符的转换元素。转换元素包括了一些属性，这些属性用来定义一个字符是如何被所给的Collator对象转换的。

这些属性包括基于音调和字符大写/小写的排序规则等。

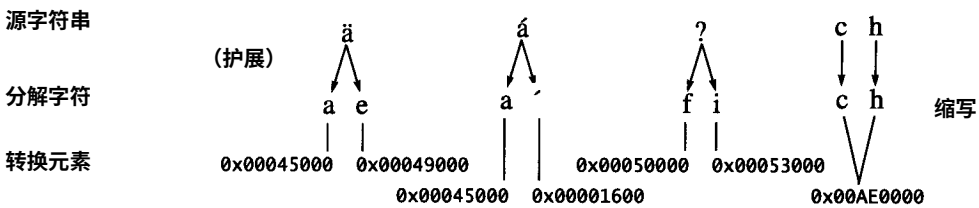
详细内容请参见《The Java Class Libraries，Second Edition，Volume 1》。

版本1.2中所作的修改

该类包含了五个方法：getOffset()、previous()、setOffset()，setText()和get-MaxExpansion()。下面的部分对转换元素进行了详细的说明。

转换元素

当一个Collator实例被创建，例如从RuleBasedCollator中被创建时，它把在源字符串中的用户字符翻译成一系列转换元素，如下图所示(用户字符在BreakIterator中说明)。



每一个单一的没有音调的字符，例如，英语中的大写字母表 A-Z，被映射到一个单一的转换元素。更复杂的用户字符则需要一个以上的转换元素。这些复杂的用户字符包括音调字符(如“ á ”)，连字符(如，？)和扩展字符(如，“ ä ”，扩展为“ ae ”用于在德文中排序)。缩写字符(如，传统西班牙语中的“ ch ”)把两个字符映射到一个转换元素上。这些转换元素包括每个被分解的字符的排序数据，用于确定长度的数值，如，大写/小写和音调。一个CollationElementIterator实例对这些转换元素进行迭代。

next()和previous()方法用于对转换元素进行迭代，而setOffset()把迭代器设置到一个指定的用户字符上。第一次调用next()把迭代器指向第一个转换元素并返回这个转换元素。因为在用户字符和转换元素之间没有一一对应的映射，所以，调用next()并非总是能增加getOffset()的返回值(在是扩展字符的情况下)。当偏移量增加时，其增加值可能比1大(在是缩写字符的情况下)。

调用setOffset()重设迭代器，调用next()返回该偏移量处的转换元素。例如，如果当前的源字符扩展成了三个转换元素，而使用next()只返回了两个，调用setOffset()将重新设置迭代器，使它指向在源文本某个偏移量处的字符并且将会从新设置每一个内部的分解/扩展缓冲区。

一个转换元素的前16位(前4个十六进制数)是它的基本序号，序号通常以拉丁字母作为基本字符。接下来的8位(两个十六进制数)是二级序号，通常是音调，最后8位(两个十六进制数)是最后一级序号，通常是用来区别大小写。

	成员概述	
	串尾常量	
	UNLLlORDER	当到达串尾时通过next()返回。
	迭代器方法	
1.2	getMaxExpansion()	获取以指定转换元素结尾的一个扩展序列中转换元素的最大数量。
1.2	getOffset()	获取源文本中的当前字符的偏移量。
	next()	把迭代器指向字符串中下一个转换元素且返回所指向的转换元素。
1.2	previous()	把迭代器指向字符串中前一个转换元素且返回所指向的转换元 素。
	reset()	把迭代器重新设置到字符串的开始。
1.2	setOffset()	在源文本中设置迭代器的位置。
1.2	setText()	设置进行迭代一的新字符串。
	获取序号的方法	
	primaryOrder()	从一个转换元素中获取基本序号值。
	secondaryOrder()	从一个转换元素中获取二级序号值。
	tertiaryOrder()	从一个转换元素中获取末级序号值。

参见

Collator.RuleBasedCollator。

《The Java Class Libraries，Second Edition，Volume 1》中CollationElementIterator。

1.2 getMaxExpansion()

目的	获取以指定转换元素结尾的一个扩展序列中转换元素的最大数量。
语法	public int getMaxExpansion(int element)
描述	只有当查询时使用该方法。该方法返回以指定转换元素 element结尾的一个扩展序列中转换元素的最大数量。一个扩展序列是一个字符被扩展成多个字符所形成的序列。

例如，德语中的字符“ ？”被扩展为“ ae”，即转换元素 0x00045000、0x00049000。在这种情况下，getMaxExpansion(0x00049000)返回2。因为用户字符扩展成了两个转换元素。简单的英语字符“ a”被翻译成一个转换元素0x00450000。在这种情况下，getMaxExpansion(0x00450000)返回1。

参数

element 从previous()或next()返回的转换元素，一个整数。

返回 一个扩展序列中转换元素的最大数量，一个整数。

示例 下面的示例使用了next()对一个字符串进行了迭代，并打印出每个转换元素的getMaxExpansion()的值。其中，“GERMAN”代表当前德语地区，所以，音调化的字符“？”扩展为“a”，后接“¨”而不是过去的“ae”。

```
import java.text.CollationElementIterator;
import java.text.Collator;
import java.text.RuleBasedCollator;
import java.util.Locale;

public class Main {
    public static void main(String[] args) {
        Locale loc = Locale.GERMAN;
        int element;

        // Set up string to iterate over.
        String str = "äabcABCàÀ";
        System.out.println("String: " + str + "\n");

        // Create an instance of a subclass of collator.
        Collator collator = Collator.getInstance(loc);

        if (collator instanceof RuleBasedCollator) {
            // Cast to the subclass.
            RuleBasedCollator rbc = (RuleBasedCollator)collator;

            // Try changing the decomposition mode.
            // rbc.setDecomposition(Collator.NO_DECOMPOSITION);

            // Get the first key of the string.
            CollationElementIterator cei =
                rbc.getCollationElementIterator(str);

            System.out.print("collation  primary  secondary  tertiary");
            System.out.println("   getMaxExpansion");
            System.out.println(" element    order     order      order");
            System.out.println("  (hex)");

            // Iterate to next character and get collation element.
            while ((element = cei.next()) !=
                    CollationElementIterator.NULLORDER) {

                System.out.print(" ");
                printInColumn(Integer.toHexString(element), 12);

                // Print the primary, secondary and tertiary orders.
                printInColumn(
                    CollationElementIterator.primaryOrder(element), 11
                );

                printInColumn(
                    CollationElementIterator.secondaryOrder(element), 11
                );

                printInColumn(
                    CollationElementIterator.tertiaryOrder(element), 11
                );
```

```
                    printInColumn(
                        cei.getMaxExpansion(element), 0
                    );

                    System.out.println();
                }
            }
        }

        // Print string in a particular vertical column
        static void printInColumn(String str, int col) {
            System.out.print(str);
            for (int p = str.length(); p < col; ++p) {
                System.out.print(" ");
            }
        }

        // Print integer in a particular vertical column
        static void printInColumn(int integer, int col) {
            System.out.print(integer);
            for (int p = Integer.toString(integer).length(); p < col; ++p) {
                System.out.print(" ");
            }
        }
    }
```

输出

```
> java Main
String: äabcABCàÀ

collation  primary  secondary  tertiary   getMaxExpansion
 element    order      order      order
  (hex)
 520000       82         0          0          1
 1c00         0         28          0          3
 520000       82         0          0          1
 530000       83         0          0          1
 540000       84         0          0          1
 520001       82         0          1          1
 530001       83         0          1          1
 540001       84         0          1          1
 520000       82         0          0          1
 1600         0         22          0          3
 520001       82         0          1          1
 1600         0         22          0          3
```

1.2 getOffset()

目的 获取在源文本中当前字符的偏移量。

语法 public int getOffset()

描述 此方法能获取源文本中当前字符的偏移量。偏移量是指从零开始计算的当前用户字符的位置（在没有被分解的情况下）。也就是说，如果接着调用next()，则可以返回这个字符的Collation元素(根据语言的需要可以包括不止一个字符)。例如，如果当前的字符扩展成三个字符，而利用 next()已经返回了其中的两个，那么调用getOffset()将获取的值与调用next()前的值相同。

返回 当前字符的偏移量，一个非负整数。

示例 此方法在下面的内容以及在previous()中都有说明。

```
import java.text.CollationElementIterator;
import java.text.Collator;
import java.text.RuleBasedCollator;
```

```
import java.util.Locale;

public class Main {
    public static void main(String[] args) {
        Locale loc = Locale.GERMAN;
        int element;

        // Set up string to iterate over.
        String str = "äb";
        System.out.println("String: " + str + "\n");

        // Create an instance of a subclass of collator.
        Collator collator = Collator.getInstance(loc);

        if (collator instanceof RuleBasedCollator) {
            // Cast to the subclass.
            RuleBasedCollator rbc = (RuleBasedCollator)collator;

            // Get the first collation key of the string.
            CollationElementIterator cei =
                rbc.getCollationElementIterator(str);

            element = cei.next();

            // Get the decomposed 'a'
            System.out.println( Integer.toHexString(element) );   // 52000
            System.out.println( cei.getOffset() );                // 1

            element = cei.next();

            // Get the decomposed accent for the 'a'
            System.out.println( Integer.toHexString(element) );   // 1c00
            System.out.println( cei.getOffset() );                // 1

            element = cei.next();

            // Get the character 'b'
            System.out.println( Integer.toHexString(element) );   // 53000
            System.out.println( cei.getOffset() );                // 2
        }
    }
}
```

1.2 previous()

目的 把迭代器指向字符串中前一个转换元素，且返回所指向的转换元素。

语法 public int previous()

描述 每调用一次该方法就把迭代器指向字符串中前一个转换元素，且返回所指向的转换元素。重复使用该方法在转换元素上移动迭代器，直到遇到字符串开始并返回nullORDER为止。

返回 前一个转换元素，到达字符串开始处时返回nullORDER。

示例 该示例对一个字符串从后向前进行迭代。注意它是如何先迭代音调字符而后才迭代基本字符的，例如对字符“ à ”和“ à ”，因为上述原因，所以会有偏移量被重复打印(7，7和6，6)。

```
import java.text.CollationElementIterator;
import java.text.Collator;
import java.text.RuleBasedCollator;
import java.util.Locale;
```

```
public class Main {
    public static void main(String[] args) {
        Locale loc = Locale.FRENCH;
        int element;

        // Set up string to iterate over.
        String str = "abcABCàÀ";

        // Create an instance of a subclass of collator.
        Collator collator = Collator.getInstance(loc);

        if (collator instanceof RuleBasedCollator) {
            // Cast to the subclass.
            RuleBasedCollator rbc = (RuleBasedCollator)collator;

            // Get the collation element of the string.
            CollationElementIterator cei =
                rbc.getCollationElementIterator(str);

            // Set the text. Set offset to the end of the string.
            cei.setText(str);
            cei.setOffset(str.length());
            System.out.println(str);                    // abcABCàÀ

            while ((element = cei.previous()) !=
                    CollationElementIterator.NULLORDER) {
                System.out.print(cei.getOffset());
            }                                           // 7766543210
            System.out.println("");
        }
    }
}
```

1.2 setOffset()

目的 在源文本中设置迭代器的位置。

语法 public void setOffset(int newOffset)

描述 该方法用于在源文本中设置迭代器的位置。注意，该方法是以未分解字符为偏移量单位对迭代器进行设置的，而不像 next()，它是以分解后的字符作为偏移量单位对迭代器进行设置的。在进行字符比较的时候，可以通过不断地增加newOffset的值而后调用 setOffset()来对一个字符串进行迭代。先调用setOffset()重设迭代器的位置，再调用 next()将返回位于指定偏移量上的字符的第一个转换元素。例如，当前的源字符扩展成三个转换元素，而调用next()已经返回了其中的两个，再调用 setOffset()将会重设迭代器，这样迭代器就会指向原文本中一个指定的偏移量上的字符，并且将会重设每一个内部的分解/扩展缓冲区。

参数

newOffset 用于指定偏移量。该偏移量的值必须在 0到源字符串长度值之间(包括0和字符串长度值)。

示例 下面的示例为一个字符串创建了一个 CollationElementIterator和一个模式，并且对字符串进行迭代，看一看是否从该字符串中可以找到这种模式。当找到一匹配，就打印这个匹配和其相应的偏移量。为了能找到匹配，该程序对模式进行迭代，并与每一个转换元素进行比较，直到到达字符串尾。

```
import java.text.CollationElementIterator;
import java.text.RuleBasedCollator;
import java.text.Collator;
import java.util.Locale;

public class Main {
    public static void main(String[] args) {
        Locale loc = Locale.FRENCH;

        String str = "abcàÀABCÀADE";
        String pat = "ÀA";

        System.out.println("String:  " + str);
        System.out.println("Pattern: " + pat + "\n");

        // Create an instance of a subclass of collator.
        Collator collator = Collator.getInstance(loc);

        boolean match;
        int patElement, strElement;

        if (collator instanceof RuleBasedCollator) {
            // Cast to the subclass.
            RuleBasedCollator rbc = (RuleBasedCollator)collator;

            CollationElementIterator strIter =
                      rbc.getCollationElementIterator(str);

            CollationElementIterator patIter =
                      rbc.getCollationElementIterator(pat);

            int END_OF_STRING = CollationElementIterator.NULLORDER;
            System.out.println("Searching... ");

            if (str.length() < pat.length()) {
               return;
        }

            // Step through the string.
            for (int i = 0; i <= (str.length() - pat.length()); i++) {
                // Print diagnostic.
                System.out.println(str.substring(i, i + pat.length()));

                // See if the pattern's collation elements match
                // the ones at this offset in the string.
                patIter.reset();
                strIter.setOffset(i);
                match = true;
                while (((patElement = patIter.next()) != END_OF_STRING)) {
                    strElement = strIter.next();
                    if (patElement != strElement) {
                        match = false;
                        break;
                    }
                }
                if (match) {
                    System.out.println(
                       "Found a match " + pat + " at offset: " + i);
                }
            }
        }
    }
}
```

输出

```
> java Main
```

```
String:  abcàÀABCÀADE
Pattern: ÀA

Searching...
ab
bc
cà
àÀ
ÀA
Found a match ÀA at offset: 4
AB
BC
CÀ
ÀA
Found a match ÀA at offset: 8
AD
DE
```

1.2 setText()

目的 设置一个新的进行迭代的字符串。

语法 public void setText(String newText)

public void setText(CharacterIterator newCharIter)

描述 此方法用于设置一个新的进行迭代的字符串。它的参数可以是一个 String实例或是一个CharacterIterator实例。偏移量设为zero。

参数

newCharIter 一个新的非空字符串迭代器，用于迭代。

newText 需要进行迭代的一个新的非空的字符串。

示例 见previous()。