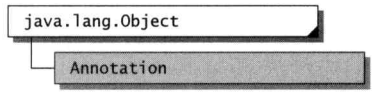
java.text

Annotation



语法

public class Annotation extends Object

描述

一个Annotation对象是一个特殊类型的属性值（对术语属性的定义请参阅AttributedCharacterIterator类的描述）。一个注解可用来作为一文本属性值的包容器，以给它一种“注解”的特征，这样其行为看起来就像是一个“块”属性。属性值的这种包容器可以防止把属性值看作是更大的属性场的一部分，或认为它还可以分解为更小的属性场。以下是注解特征。

- 属性关键字/值对所适用的文本域中的域的语义是严格的。也就是说属性不适用于文本域的子域。另外，即使两相邻的文本域对一个属性具有相同的值，属性仍然不适用于由两文本域合并而成的文本域。
- 如果底层文本被修改，那么文本域上的属性关键字或其值便不再适用于此文本。

这至少对以下三个方面产生影响。

- AttributedCharacterIterator类的getRunStart()方法和getRunLimit()方法把两相邻的注解看作是彼此分开的属性场。
- 如果所创建的迭代器没有完全跨越整个注解实例，那么 getAttribute()方法所返回的值为null。
- 当构造函数AttributedString()从一个已存在的属性文本对象构造一个新的属性文本对象时，如果指定的子区域没有完全包含注解的范围，那么任何一个其值被一个 Annotation对象包装起来的属性都会被抛弃。

不是所有被包装进一个Annotation对象的属性都是有意义的。我们看一个注解能发挥作用的示例，在这个示例中注解的作用是存储“读”信息（在日语中被称为“ yomi”），所谓“读”信息是指文本的发音方式。在下面这个示例中，法语的动词“ aimez”对应于英语中的“ like”，它的发音为“ eme”（用法语音标）。这个发音只适用于整个单词，因此只有当迭代器跨越了整个“ aimez”单词时，getValue()方法的返回值才是“ eme”。同时我们注意到“ aimez”是以一种相邻的注解形式出现的，要注意这时把它们合在一起是没有任何意义的。（这些句子翻译成英语是“ You like it? Do you like the automobile? ”。）

注解的另一个示例是附于一个句子上的语法信息。在上面的第一个句子中，可以认为，“ vous ”是主语，“ l'aimez ”是谓语，但是不能认为“ l ”，“ aimez ”或“ aim ”是谓语。在文本被修改后，通常所附加的语法信息便不再有效。

把属性值包装进一个Annotation对象中，可以保证迭代器不把相邻的属性场看成一个单一的属性场，即使属性值是相等的（例如示例中的“ eme ”）。这样还可以通知文本存储器对象，在底层文本被修改后应当抛弃属性关键字/值对。

目前允许在 AttributedCharacterIterator.Attribute类中的两个预定义常量INPUT_METHOD_SEGMENT和READING的值为Annotation对象。

使用

如果把一个属性值“ 提升 ”为一个注解值，那么便创建了一个注解对象并为它提供了属性值。然后，可以把注解当作一个属性值添加到属性文本上。具体做法如下：

1) 根据所给字符串，首先创建一个AttributedString类的实例或自定义的属性文本对象：

```
AttributedString attrStr = new AttributedString(
    "A grown man has seen a million things schemed.");
```

2) 调用Annotation()构造函数并传递一个属性值给它，创建一个预定义注解。接着为在23～30之间的区域中具有关键字READING的单词“ million ”添加发音文本。

```
attrStr.addAttribute(AttributedCharacterIterator.Attribute.READING,
    new Annotation("mil-yen"), 23, 30);
```

对每个需要注解的子区域重复上述步骤。

3) 创建Attribute类的一个子类并定义自己的静态域，这样就可以创建一个用户自定义注解关键字。在下面这个示例中创建了一个称为 GRAMMAR的静态域，这个静态域可以标识一个句子的主语和谓语，如下所示：

```
public class MyAttr extends AttributedCharacterIterator.Attribute {
    public static final MyAttr GRAMMAR = new Attribute("grammar");
}
attrStr.addAttribute(MyAttr.GRAMMAR, new Annotation("subject"), 0, 12);
attrStr.addAttribute(MyAttr.GRAMMAR, new Annotation("predicate"), 12, 46);
import java.text.AttributedString;
import java.text.AttributedCharacterIterator;
import java.text.Annotation;
import java.util.Locale;
```

(4) 调用AttributedString类中的getIterator()方法以获得一个迭代器，此迭代器可以用来获取那些关键字跨越了整个区域或某可选区域的注解。

AttributedCharacterIterator iter = attrStr.getIterator();

然后，可以利用AttributedCharacterIterator类中的任何一个方法或从CharacterIterator类继承来的方法移动到任何的字符或注解，并取得一个或多个注解场的开始和结束字符的位置。

成员概述	
构造函数	
Annotation()	构造一个包含了一个属性值的 Annotation类的实例。
属性方法	
getValue()	检索Annotation类的实例的值。
调试方法	
toString()	建立Annotation类的实例的字符串表达式。

参见

AttributedCharacterIterator，AttributedCharacterIterator.Attribute。

示例

此例首先创建了一个属性字符串，该字符串包含了 READING属性关键字，且给字符串中的每个单词都添加了发音字符串。每个发音字符串都被一个 Annotaion类的实例封装起来，以表明一个发音字符串只适用于整个单词。然后，创建了一个迭代器，该迭代器跨越了第一个单调的全部，但只跨越了第二个单词的一部分（以演示它是如何处理部分单词）。最后打印输出属性场。注意，打印输出的是头两个单词而不是部分单词“ aim ”（法语单词“ aimez ”的一部分）的发音字符串（注解的值）。

```
import java.text.AttributedString;
import java.text.AttributedCharacterIterator;
import java.text.Annotation;
import java.util.Locale;

class Main {
    // Assign READING constant to a variable (to save typing).
    static AttributedCharacterIterator.Attribute attrREAD =
        AttributedCharacterIterator.Attribute.READING;

    public static void main(String[] args) {
        // Create an attributed string.
        AttributedString attrStr = new AttributedString("Vous l'aimez.");

        // Add reading attribute key/value pairs for each subrange.
        attrStr.addAttribute(attrREAD, new Annotation("vu"), 0, 4);   // Vous
        attrStr.addAttribute(attrREAD, new Annotation("l"), 5, 7);    // l'
        attrStr.addAttribute(attrREAD, new Annotation("eme"), 7, 12); // aimez

        AttributedCharacterIterator.Attribute[] matchAttr =
            new AttributedCharacterIterator.Attribute[] {
                attrREAD
            };

        // Create an iterator for the attributed string.
        AttributedCharacterIterator iter =
           attrStr.getIterator(matchAttr, 0, 10);

        // Print each run of the READING attribute.
        while (iter.current() != AttributedCharacterIterator.DONE) {
            printToRunLimit(iter);
        }
    }

    // Print the run start, run limit, string and attribute.
    public static void printToRunLimit(AttributedCharacterIterator aci) {
        // Get the run limit of the READING attribute.
        int runStart = aci.getRunStart(attrREAD);
        printInColumn("" + runStart, 5);
        int runLimit = aci.getRunLimit(attrREAD);
        printInColumn("" + runLimit, 5);
        Object attr = aci.getAttribute(attrREAD);

        // Print characters to the run limit
        System.out.print("\"");
        while (aci.getIndex() < runLimit) {
            System.out.print(aci.current());
            aci.next();
        }
```

```
            System.out.print("\"    ");
            printInColumn("", 10 - (runLimit - runStart));
            if (attr != null) {
                if (attr.getClass() == Annotation.class) {
                    System.out.println(((Annotation)attr).getValue());
                }
            } else {
                System.out.println(attr);
            }
        }

        // Print string, then pad spaces to 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(" ");

                 }
             }
        }
```

输出

```
> java Main
0    4    "Vous"          vu
4    5    " "             null
5    7    "l'"            l
7    10   "aim"           null
```

Annotation()

目的	构造一个包含了属性值的Annotation类的实例。
语法	public Annotaion(Object val)
描述	此方法构造一个包含了val的Annotation类的实例。可以用此Annotation类的实例替换一个属性关键字/值对中的属性值。一个Annotation类的实例封装了属性值。 值有可能为null。INPUT_METHOD_SEGMENT属性（在AttributedCharacterIterator.Attribute中）取值为null的注解。
参数	
val	注解的值，可能为null。
示例	见类的示例。

getValue()

目的	检索Annotation类的实例的值。
语法	public Object getValue()
描述	Annotation类的对象值包一个单一的值，就是在构造函数Annotation()创建它时所接收到的值。此方法返回的便是该值。
返回	返回Annotation类的实例的值，如果实例在创建时所接收到的值为 null，那么这个值可以是null。
示例	见类的示例。

toString()

目的	建立Annotation类的实例的字符串表达式。

语法 public String toString()

描述 此方法建立Annotation类的实例的字符串表达式。这个字符串表达式包括类名及注解的值。

返回 返回Annotation类的实例的非空字符串表达式。

重载 Object.toString。

示例 此例为一个地区值创建一个 Annotation类的实例，并打印出它的字符串表达式。

```
import java.text.Annotation;
import java.util.Locale;

class Main {
    public static void main(String[] args) {

        // Create an annotation.
        Annotation annot = new Annotation(java.util.Locale.FRENCH);

        System.out.println(annot.toString());
    }
}
```

输出

```
> java Main
java.text.Annotation[value=fr]
```