1 Java字符串

Java字符串就是Unicode字符序列。Java没有内置的字符串类型，而是在标准Java类库中提供了一个预定义类`String`。每个用双引号括起来的字符串都是`String`类的一个实例。例如：

```
String e = ""; // 空串
String greeting = "Hello";
```

类似于C和C++，Java字符串中的代码单元和码点从0开始计数。

2 String API

下面介绍`String`类中常用的方法。

```
/* java.lang.String */
char charAt(int index)
    // 返回给定位置的代码单元
int codePointAt(int index)
    // 返回从给定位置开始的码点
int offsetByCodePoints(int startIndex, int cpCount)
    // 返回从 startIndex 码点开始，cpCount 个码点之后的码点索引
int CompareTo(String other)
    // 按照字典序，若字符串位于 other 之前，返回一个负数；若位于 other 之后，返回一个正数；若相等，返回0
IntStream codePoints()
    // Java 8引入。将字符串的码点作为一个流返回。可以调用 toArray 方法将它们放在一个数组中
String(int[] codePoints, int offset, int count)
    // 用数组中从 offset 开始的 count 个码点构造一个字符串
boolean empty()
    // 若字符串为空串，返回 true
boolean blank()
    // Java 11引入。若字符串由空格组成，返回 true
boolean equals(Object other)
    // 若字符串与 other 相等，返回 true，否则返回 false
boolean equalsIgnoreCase(String other)
    // 若字符串与 other 相等（忽略大小写），返回 true
boolean startsWith(String prefix)
    // 若字符串以 prefix 开头，返回 true
boolean endsWith(String suffix)
    // 若字符串以 suffix 结尾，返回 true
```

```
int indexOf(String str)
    // 返回与字符串 str 匹配的第一个子串的开始位置，从0开始从前往后匹配。若不存在 str，返回-1
int indexOf(String str, int fromIndex)
    // 返回与字符串 str 匹配的第一个子串的开始位置，从 fromIndex 开始从前往后匹配。若不存在
str，返回-1
int indexOf(int cp)
    // 返回码点为 cp 的字符第一次出现的位置，从0开始从前往后匹配。若不存在该字符，返回-1
int idexOf(int cp, int fromIndex)
    // 返回码点为 cp 的字符第一次出现的位置，从 fromIndex 开始从前往后匹配。若不存在该字符，
返回-1
int lastIndexOf(String str)
    // 返回与字符串 str 匹配的最后一个子串的开始位置，从字符串末尾开始从后往前匹配。若不存在
str ，返回-1
int lastIndexOf(String str, int fromIndex)
    // 返回与字符串 str 匹配的最后一个子串的开始位置，从 fromIndex 开始从后往前匹配。若不存
在 str，返回-1
int lastIndexOf(int cp)
    // 返回码点为 cp 的字符最后一次出现的位置，从字符串末尾开始从后往前匹配。若不存在该字符，返
回-1
int lastIndexOf(int cp, int fromIndex)
    // 返回码点为 cp 的字符最后一次出现的位置，从 fromIndex 开始从后往前匹配。若不存在该字
符，返回-1
int length()
    // 返回字符串代码单元的个数
int codePointCount(int startIndex, int endIndex)
    // 返回 startIndex 和 endIndex-1 之间的码点个数
String replace(CharSequence oldString, CharSequence newString)
    // 返回一个新字符串，用 newString 代替原始字符串中所有的 oldString
String substring(int beginIndex)
    // 返回一个新字符串，这个字符串包含原始字符串中从 beginIndex 到末尾的所有代码单元
String substring(int beginIndex, int endIndex)
    // 返回一个新字符串，这个字符串包含原始字符串中从 beginIndex 到 endIndex-1 的所有代码单
元
String toLowerCase()
    // 返回一个新字符串，将原始字符串中的大写字母改为小写
String toUpperCase()
    // 返回一个新字符串，将原始字符串中的小写字母改为大写
String trim()
    // 返回一个新字符串，删除原始字符串头部和尾部小于等于 U+0020 的字符
String strip()
    // Java 11引入。返回一个新字符串，删除原始字符串头部和尾部的空格
static String join(CharSequence delimiter, CharSequence... elements)
    // 静态方法，Java 8引入。返回一个字符串，用给定的定界符连接所有元素
String repeat(int count)
    // Java 11引入。返回一个新字符串，将当前字符串重复 count 次
```

在上面的API注释中，有一些`CharSequence`类型的参数。这是一种接口类型，所有字符串都属于这个接口。关于接口的内容将在后面学习，现在只需要知道，当看到一个`CharSequence`形参时，可以传入`String`类型的实参。

使用这些API就可以实现各种功能，下面做简要介绍。

3 子串

可以用String类的`substring`方法从一个字符串中截取子串。例如：

```
String greeting = "Hello";
String s = greeting.substring(0, 3); // 截取位置为0、1、2的字符，s为"Hel"
```

使用`substring`便于计算子串的长度。字符串`str.substring(a, b)`的长度为$b - a$。

4 拼接

Java允许使用`+`号拼接两个字符串。例如：

```
String expletive = "Expletive";
String PG13 = "deleted";
String message = expletive + PG13; // message is "Expletivedeleted"
```

当将一个字符串与一个非字符串的值进行拼接时，后者会转换成字符串。例如：

```
int age = 13;
String rating = "PG" + age; // rating is "PG13"
```

这种特性通常用于输出语句。例如：

```
int answer = 3;
System.out.println("The answer is " + answer);
```

如果需要把多个字符串放在一起，用一个界定符分隔，可以使用静态`join`方法。例如：

```
String all = String.join(" / ", "S", "M", "L", "XL");
    // all is "S / M / L / XL"
```

在Java 11 中，还提供了一个`repeat`方法：

```
String repeated = "Java".repeat(3); // repeated is "JavaJavaJava"
```

5 不可变字符串

String类没有提供修改字符串中某个字符的方法，这意味着不能修改Java字符串中的单个字符，所以将String类对象称为不可变的。要想修改一个字符串，可以提取想要保留的子串，再与希望替换的字符拼接。例如：

```
String greeting = "Hello";
greeting = greeting.substring(0, 3) + "p!"; // greeting被修改为"Help!"
```

在字符串不可变的条件下，编译器可以让字符串共享。各种字符串存放在公共的存储池中，字符串变量指向存储池中相应的位置。如果复制一个字符串，原始字符串与复制的字符串共享相同的地址。

6 检测字符串是否相等

可以使用`equals`方法检测两个字符串是否相等。例如：

```
"Hello".equals(str);
```

要想检测两个字符串是否相等，而不区分大小写，可以使用`equalsIgnoreCase`方法：

```
"Hello".equalsIgnoreCase("hello");
```

不能使用`==`运算符判断两个字符串是否相等，这个运算符只能确定两个字符串的地址是否相等。

7 空串与Null串

空串`""`是一个String对象，长度为0，内容为空。可以用以下两种方法之一判断字符串是否为空：

```
if(str.length() == 0)
if(str.equals(""))
```

`null`是String变量的一个特殊值，表示目前没有任何对象与该变量关联。判断字符串是否为`null`可以使用如下条件：

```
if(str == null)
```

要判断一个字符串既不是空串也不是`null`，可以使用如下条件：

```
if(str != null && str.length() != 0)
```

注意两个条件不可交换顺序。如果在一个`null`值上调用方法会出现错误，因此必须先判断是否为`null`。

8 字符串长度

`length`方法返回代码单元数量。要想得到实际长度，可以用如下方法：

```
int cpCount = str.codePointCount(0, str.length()); // 返回两个位置之间的码点个数
```

调用`str.charAt(n)`返回位置n的代码单元。要想得到第n个码点，可以用如下方法：

```
int index = str.offsetByCodePoints(0, n); // 返回从0开始，n 个码点之后的码点索引
int cp = str.codePointAt(index); // 返回给定位置的码点
```

9 字符串遍历

可以使用`codePoints`方法生成一个`int`值的流，每个`int`值对应一个码点，用`toArray`方法将它转换为数组，再完成遍历。例如：

```
int[] codePoints = str.codePoints().toArray();
```

反之，要把一个码点数组转换为一个字符串，可以使用构造器。例如：

```
String str = new String(codePoints, 0, codePoints.length);
```

10 StringBuilder类

`StringBuilder`类在Java 5中引入，可以用较短的字符串构建字符串。这个类的前身是`StringBuffer`，它的效率稍有些低，但允许采用多线程的方式添加或删除字符。如果所有字符串编辑操作都在单个线程中执行，则应该使用`StringBuilder`。这两个类的API是一样的。

下面介绍`StringBuilder`类中的重要方法。

```
/* java.lang.StringBuilder */
StringBuilder()
    // 构造一个空的字符串构建器
int length()
    // 返回构建器或缓冲器中的代码单元数量
StringBuilder append(String str)
    // 追加一个字符串并返回 this
StringBuilder append(char c)
    // 追加一个代码单元并返回 this
StringBuilder appendCodePoint(int cp)
    // 追加一个码点并返回 this
void setCharAt(int i, char c)
    // 将第 i 个代码单元设置为 c
StringBuilder insert(int offset, String str)
    // 在 offset 位置插入一个字符串并返回 this
StringBuilder insert(int offset, char c)
    // 在 offset 位置插入一个代码单元并返回 this
StringBuilder delete(int startIndex, int endIndex)
    // 删除偏移量从 startIndex 到 endIndex-1 的代码单元并返回 this
String toString()
    // 返回一个与构建器或缓冲器内容相同的字符串
```

如果需要用许多小段的字符串来构建一个字符串，可以用如下方法：

```
/* 构建空的字符串构建器 */
StringBuilder builder = new StringBuilder();
/* 每次需要添加一部分内容时，调用 append 方法 */
builder.append(ch); // 追加一个代码单元
builder.append(str); // 追加一个字符串
/* 字符串构建完成时调用 toString 方法，得到一个 String 对象 */
String completedString = builder.toString();
```