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";
```

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
/* java.lang.String */

/**
 * 用 codePoints 数组中从 offset 开始的 count 个码点构造一个字符串
 * @param codePoints 码点数组
 * @param offset 数组的起始索引
 * @param count 码点数量
 * @since 1.5
 */
public String(int[] codePoints, int offset, int count)

/**
 * 获得字符串的长度
 * @return 字符串中代码单元的个数
 */
public int length()

/**
 * 计算指定范围内的码点个数
 * @param startIndex 起始索引（按代码单元计算）
 * @param endIndex 结束索引（按代码单元计算，不包含）
 * @return 从 startIndex 到 endIndex - 1 的码点个数
 * @throws IndexOutOfBoundsException 当 startIndex < 0，
 *                                   或 endIndex 超过字符串的长度，
 *                                   或 startIndex > endIndex
 * @since 1.5
 */
public int codePointCount(int startIndex, int endIndex)

/**
 * 返回给定位置的代码单元
 * @param index 代码单元的索引
 * @return 给定索引处的代码单元
 * @exception IndexOutOfBoundsException 当 index 越界
 */
public char charAt(int index)

/**
 * 返回从给定位置开始的码点
 * @param index 代码单元的索引
 * @return 如果给定索引处的代码单元是辅助字符的第一个代码单元，
 *         并且下一个代码单元是辅助字符的第二个代码单元，则返回辅助字符的码点；
 *         否则，返回给定索引处的代码单元
 * @throws IndexOutOfBoundsException 当 index 越界
 * @since 1.5
 */
public int codePointAt(int index)

/**
 * 返回从 startIndex 码点开始，cpCount 个码点之后的码点索引
 * @param startIndex 起始码点的索引
 * @param cpCount 偏移量（按码点的数量计算）
```

```
     * @return 从 startIndex 偏移 cpCount 后，找到的码点的索引
     * @throws IndexOutOfBoundsException 当 startIndex 或 startIndex + cpCount 越界
     * @since 1.5
     */
    public int offsetByCodePoints(int startIndex, int cpCount)

    /**
     * 按照字典序比较字符串
     * @param other 待比较的字符串
     * @return 如果当前字符串位于 other 之前，则返回一个负数；
     *         如果当前字符串位于 other 之后，则返回一个正数；
     *         如果当前字符串与 other 相等，则返回 0。
     *         对两个字符串从前向后逐位比较，如果第一个不相同字符的索引为 k，则返回
     *             this.charAt(k) - other.charAt(k)
     *         如果没有不相同字符，但二者长度不同，则返回
     *             this.length() - other.length()
     */
    public int compareTo(String other)

    /**
     * 将字符串的码点作为一个流返回
     * @return 一个 IntStream 对象，包含字符串中的码点
     * @since 9
     */
    public IntStream codePoints()

    /**
     * 判断字符串是否为空串
     * @return 如果字符串为空串，则返回 true，否则返回 false
     * @since 1.6
     */
    public boolean isEmpty()

    /**
     * 判断字符串是否为空串或者只包含空格
     * @return 如果字符串为空串或者只包含空格，则返回 true，否则返回 false
     * @since 11
     */
    public boolean isBlank()

    /**
     * 判断当前字符串与给定的对象是否相等
     * @param other 待比较的对象
     * @return 如果 other 是字符串，并且与当前字符串相等，则返回 true，否则返回 false
     */
    public boolean equals(Object other)

    /**
     * 判断当前字符串与给定的字符串是否相等，忽略大小写
     * @param anotherString 待比较的字符串
     * @return 如果 anotherString 与当前字符串相等（忽略大小写），则返回 true，否则返回 false
     */
```

```
public boolean equalsIgnoreCase(String anotherString)

/**
 * 判断当前字符串是否以给定的字符串开头
 * @param prefix 待判断的前缀
 * @return 如果 prefix 是当前字符串的前缀，则返回 true，否则返回 false。
 *         如果 prefix 为空串，或者与当前字符串相等，也返回 true
 */
public boolean startsWith(String prefix)

/**
 * 判断当前字符串是否以给定的字符串结尾
 * @param suffix 待判断的后缀
 * @return 如果 suffix 是当前字符串的后缀，则返回 true，否则返回 false。
 *         如果 suffix 为空串，或者与当前字符串相等，也返回 true
 */
public boolean endsWith(String suffix)

/**
 * 在当前字符串中从前向后查找子串
 * @param str 待查找的子串
 * @return str 在当前字符串中第一次出现的索引，如果没有该子串则返回 -1
 */
public int indexOf(String str)

/**
 * 在当前字符串中，从指定位置开始从前向后查找子串
 * @param str 待查找的子串
 * @param fromIndex 起始索引
 * @return str 在当前字符串中第一次出现的索引，如果没有该子串则返回 -1
 */
public int indexOf(String str, int fromIndex)

/**
 * 在当前字符串中从前向后查找指定字符
 * @param cp 待查找字符的码点
 * @return cp 在字符串中第一次出现的索引（按代码单元计算），若不存在该字符则返回 -1
 */
public int indexOf(int cp)

/**
 * 在当前字符串中，从指定位置开始从前向后查找指定字符
 * @param cp 待查找字符的码点
 * @param fromIndex 起始索引
 * @return cp 在字符串中第一次出现的索引（按代码单元计算），若不存在该字符则返回 -1
 */
public int idexOf(int cp, int fromIndex)

/**
 * 在当前字符串中从后向前查找子串
 * @param str 待查找的子串
 * @return str 在当前字符串中最后一次出现的索引，如果没有该子串则返回 -1
```

```
 */
public int lastIndexOf(String str)

/**
 * 在当前字符串中，从指定位置开始从后向前查找子串
 * @param str 待查找的子串
 * @param fromIndex 起始索引
 * @return str 在当前字符串中最后一次出现的索引，如果没有该子串则返回 -1
 */
public int lastIndexOf(String str, int fromIndex)

/**
 * 在当前字符串中从后向前查找指定字符
 * @param cp 待查找字符的码点
 * @return cp 在字符串中最后一次出现的索引（按代码单元计算），若不存在该字符则返回 -1
 */
public int lastIndexOf(int cp)

/**
 * 在当前字符串中，从指定位置开始从后向前查找指定字符
 * @param cp 待查找字符的码点
 * @param fromIndex 起始索引
 * @return cp 在字符串中最后一次出现的索引（按代码单元计算），若不存在该字符则返回 -1
 */
public int lastIndexOf(int cp, int fromIndex)

/**
 * 用新的字符串替换指定的子串
 * @param oldString 被替换的子串
 * @param newString 用于替换的字符串
 * @return 替换后的新字符串
 * @since 1.5
 */
public String replace(CharSequence oldString, CharSequence newString)

/**
 * 获得子串
 * @param beginIndex 起始索引
 * @return 从 beginIndex 到末尾的子串
 * @throws IndexOutOfBoundsException 当 beginIndex 越界
 */
public String substring(int beginIndex)

/**
 * 获得子串
 * @param beginIndex 起始索引（包括）
 * @param endIndex 结束索引（不包括）
 * @return 从 beginIndex 到 endIndex - 1 的子串
 * @throws IndexOutOfBoundsException 当 beginIndex < 0,
 *                                   或 endIndex 超过字符串的长度,
 *                                   或 beginIndex > endIndex
 */
```

```
public String substring(int beginIndex, int endIndex)

/**
 * 将大写字母改为小写
 * @return 转换后的新字符串
 */
public String toLowerCase()

/**
 * 将小写字母改为大写
 * @return 转换后的新字符串
 */
public String toUpperCase()

/**
 * 删除字符串头部和尾部的空格。这里的空格指 Unicode 编码小于等于 U+0020 的字符
 * @return 去除空格后的新字符串
 */
public String trim()

/**
 * 删除字符串头部和尾部的空白字符。空白字符包括 '\t'、'\n'、'\f'、'\r' 等
 * @return 去除空白字符后的新字符串
 * @since 11
 */
public String strip()

/**
 * 用给定的分隔符连接所有元素
 * @param delimiter 分隔符
 * @param elements 要连接在一起的元素
 * @return 连接得到的字符串
 * @throws NullPointerException 当 delimiter 或 elements 为 null
 * @since 1.8
 */
public static String join(CharSequence delimiter, CharSequence... elements)

/**
 * 将当前字符串重复若干次
 * @param count 重复的次数
 * @return 将当前字符串重复 count 次所得到的新字符串
 * @throws IllegalArgumentException 当 count < 0
 * @since 11
 */
public String repeat(int count)
```

在上面的 API 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; // 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; // 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); // The answer is 3
```

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"); // S / M / L / XL
```

在 Java 11 中，还提供了一个`repeat()`方法，可以将字符串重复若干次：

```
String repeated = "Java".repeat(3); // 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!"; // 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"); // true
```

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`codePointCount()`方法：

```
int cpCount = str.codePointCount(0, str.length()); // 返回整个字符串的码点个数
```

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
/* java.lang.StringBuilder */

/**
 * 构造一个空的 StringBuilder 对象，默认容量为 16
 */
public StringBuilder()

/**
 * 获得当前 StringBuilder 对象中字符序列的长度
 * @return 构建器或缓冲器中的代码单元数量
 */
public int length()

/**
 * 追加一个字符串
 * @param str 字符串
 * @return this
 */
public StringBuilder append(String str)

/**
 * 追加一个代码单元
 * @param c 一个代码单元
 * @return this
 */
public StringBuilder append(char c)

/**
 * 追加一个码点
 * @param cp 码点
 * @return this
 * @since 1.5
 */
public StringBuilder appendCodePoint(int cp)

/**
 * 修改指定位置的代码单元
 * @param index 要修改的代码单元的索引
 * @param ch 新的代码单元
 * @throws IndexOutOfBoundsException 当 index 越界
 */
public void setCharAt(int index, char ch)

/**
 * 向指定位置插入字符串
 * @param offset 插入位置
 * @param str 待插入的字符串
 * @return this
 * @throws StringIndexOutOfBoundsException 当 offset 越界
 */
public StringBuilder insert(int offset, String str)
```

```
/**
 * 向指定位置插入代码单元
 * @param offset 插入位置
 * @param c 待插入的代码单元
 * @return this
 * @throws IndexOutOfBoundsException 当 offset 越界
 */
public StringBuilder insert(int offset, char c)

/**
 * 删除指定范围的子串
 * @param startIndex 起始索引（包括）
 * @param endIndex 结束索引（不包括）
 * @return this
 * @throws StringIndexOutOfBoundsException 当 startIndex < 0,
 *                                         或 startIndex > this.length(),
 *                                         或 startIndex > endIndex
 */
public StringBuilder delete(int startIndex, int endIndex)

/**
 * 返回表示当前序列中数据的字符串。返回的 String 对象与当前 StringBuilder 对象相互独立
 * @return 当前字符序列的字符串表示
 */
public String toString()
```

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();
```