# 代码点与代码单元

Java字符串由char序列组成。大多数的常用Unicode字符使用一个代码单元就可以表示，而辅助字符需要一对代码单元表示。

length()方法将返回采用UTF-16编码表示的给定字符串所需要的代码单元数量

要想的到实际的长度，即代码点数量

int cpCount = str.codePointCount(0.str.length())

调用str.charAt(n)将返回位置n的代码单元，n介于0-str.length()-1之间

要想得到第i个代码点

int index = str.offsetByCodePoints(0,i);

int cp = str.codePointAt(index);

# 字符串API

char charAt (int index)

返回给定位置的代码单元。除非对底层的代码单元感兴趣，否则不需要调用这个方法

int codePointAt(int index) 5.0

返回从给定位置开始或结束的代码点

int offsetByCodePoints(int startIndex,int cpCount) 5.0

返回从startIndex代码点开始，位移cpCount后的代码点索引

int compareTo(String other)

按照字典顺序，如果字符串位于other之前，返回一个负数；如果字符串位于other之后，返回一个正数；如果两个字符串相等，返回0

boolean endsWith(String suffix)

如果字符串以suffix结尾，返回true

boolean equals(Object other)

如果字符串与other相等，返回true

Boolean equalsIgnoreCase(String other)

如果字符串与other相等（忽略大小写）,返回true

int indexOf(String str)

int indexOf(String str,int fromIndex)

int indexOf(int cp)

int indexOf(int cp,int fromIndex)

返回与字符串str或代码点cp匹配的第一个子串开始位置。这个位置从索引0或fromIndex开始计算。如果原始串中不存在str，返回-1

int lastIndexOf(String str)

int lastIndexOf(String str,int fromIndex)

int lastIndexOf(int cp)

int lastIndexOf(int cp,int fromIndex)

返回与字符串str或代码点cp匹配的最后一个子串的开始位置。这个位置从原始串尾端或fromIndex开始计算。

int length()

返回字符串的长度

int codePointCount(int startIndex,int endIndex) 5.0

返回startIndex和endIndex – 1之间的代码点数量。没有配成对的代用字符将计入代码点

String replace(CharSequence oldString,CharSequence newString)

返回一个新字符串，这个字符串用newString代替原始字符串中所有的oldString。可以用String和StringBuilder对象作为CharSequence参数。

Boolean startsWith(String prefix)

如果字符串以prefix字符串开始，返回true

String substring(int beginIndex)

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到串尾或endIndex-1的所有代码单元。

String toLowerCase()

返回一个新字符串。这个字符串将原始字符串中的所有大写字母改成了小写字母。

String toUpperCase()

返回一个新字符串。这个字符串将原始字符串中的所有小写字母改成了大写字母。

String trim()

返回一个新字符串。这个字符串将删除原始字符串头部和尾部的空格。

阅读联机API文档，下载JDK api 还有很多String API

# 构建字符串

每次连接字符串，都会构建一个新的String对象，既耗时，又浪费空间。使用StringBuilder类就可以避免这个问题的发生。

如果需要用许多小段的字符串构建一个字符串，那么应该按照下列步骤进行。首先，构建一个空的字符串构建器。

StringBuilder builder = new StringBuilder();

当每次需要添加一部分内容时，就调用append()方法

builder.append(ch); //append a single character

builder.append(str); //append a string

在需要构建字符串时就调用ToString方法，将可以得到一个String对象，其中包含了构建器中的字符序列。

String completedString = builder.toString();

StringBuilder的前身是StringBuffer，其效率稍有些低，但允许采用多线程的方式执行添加或删除字符的操作。如果所有字符串在一个单线程中编辑，则应该用StringBuilder替代它。这两个类的API是相同的。

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(Strigng 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)

追加一个代码点，并将其转换为一个或两个代码单元并返回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 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