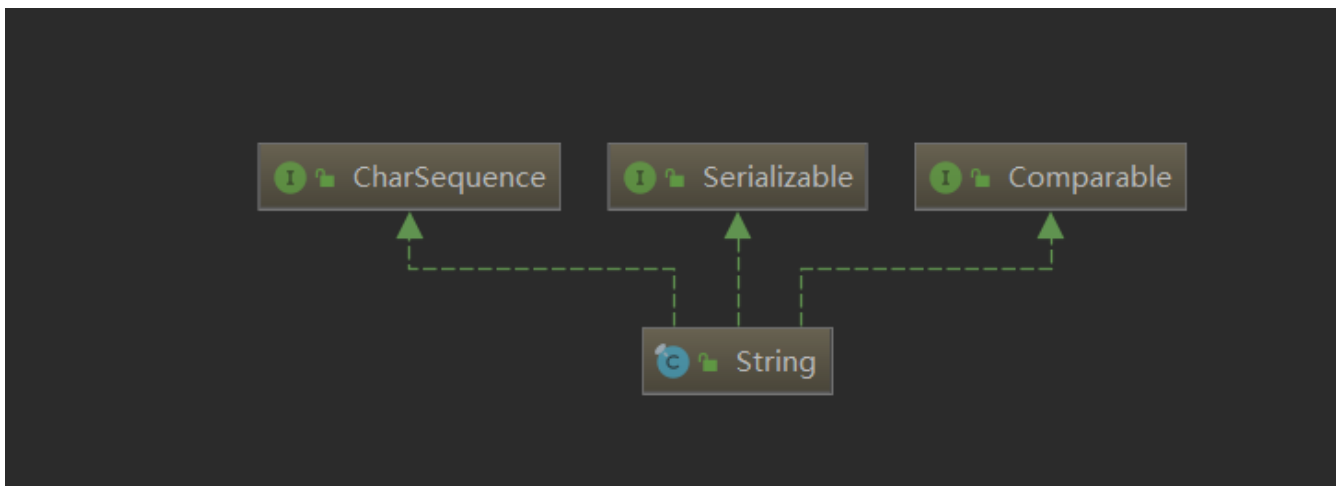


- 1. String类
  - 1.1. 构造方法
  - 1.2. String的不可变形
  - 1.3. String.format
- 2. String的常用方法
- 3. String与正则表达式
  - 3.1. 正则表达式简介
  - 3.2. 常用的正则表达式词法
  - 3.3. 正则示例
    - 3.3.1. String.matches()匹配正则表达式
    - 3.3.2. String.split()从正则表达式匹配的地方分离
    - 3.3.3. 替换与正则表达式匹配的地方
- 4. StringBuilder类
  - 4.1. StringBuilder的构造方法
  - 4.2. StringBuilder的常用方法
- 5. 基本数据类型与字符串的转换
  - 5.1. 基本类型转换为字符串
  - 5.2. 字符串转化为基本类型

# 字符串

## 1. String类



这里介绍一下 `String` 类的常用方法

注意 `String` 是 `CharSequence` 子类。

### 1.1. 构造方法

```

1 String() //初始化空字符
2 String(char[] value) //将字符数组的内容转化成字符串
3 String(char[] value, int offset, int count) //将字符数组指定区间的内容转化为字符串,offset指定字符数组的起始转化位置, count指定转化的字符数组数目, 如果offset+count超过字符数组长度会抛出异常
4 Sting(String original) //使用指定字符串初始化
5 String(StringBuilde builder) //使用StringBuilde对象进行初始化
6 static String valueOf(int i) //返回int参数的字符串形式, valueOf的参数可以是任意的基本数据类型

```

## 1.2. String的不可变形

String对象是不可变的。查看JDK文档你就会发现，String类中每一个看起来会修改String值的方法，实际上都是**创建了一个全新的String对象**，以包含修改后的字符串内容。而最初的String对象则丝毫未动。

因此当我们使用"+"、"+="运算符进行字符串拼接时，实际上也会产生新的String对象，效率低下。当需要进行大量的字符串拼接时，我们可以使用StringBuilder对象，调用append()方法提高效率。（StringBuilder是可变字符串类型）

```

1 public static void main(String[] args)
2 {
3     StringBuilder builder=new StringBuilder();
4     for(int i=1;i<=5;i++){
5         builder.append(i);
6         builder.append(" ");
7     }
8     System.out.println(builder);
9 }
10 /*Output
11 1 2 3 4 5
12 */

```

## 1.3. String.format

String有一个静态方法，可以使用格式化字符串返回特定格式的字符串

```

1 public static String format(String format,Object... args)

```

示例：

```

1 //将输入的整数转换为16进制字符串
2 public static void main(String[] args)
3 {
4     Scanner cin=new Scanner(new BufferedInputStream(System.in));
5     StringBuilder builder=new StringBuilder();
6     for(int i=0;i<3;i++)
7     {
8         builder.append(String.format("%h",cin.nextInt()));
9         builder.append("\n");
10    }
11    System.out.println(builder);
12 }
13 /*Output
14 12
15 16
16 32
17 c
18 10
19 20
20 */

```

## 2. String的常用方法

```

1 //查询
2 char charAt(int index) //获取指定索引位置的字符
3 int contains(String str) //判断是否包含str
4 int indexOf(int ch) //返回指定字符在此字符串中第一次出现处的索引。
5 int indexOf(String str) //返回指定字符串在此字符串中第一次出现处的索引。
6 int indexOf(int ch,int fromIndex) //返回指定字符在此字符串中从指定位置后第一次出现处的索引。
7 int indexOf(String str,int fromIndex) //返回指定字符串在此字符串中从指定位置后第一次出现处的索引。
8 int lastIndexOf(String str) //返回指定字符串最后一次出现的索引
9
10 //字符串比较
11 int compareTo(String anotherString) //按字典序比较两个字符串
12 int compareToIgnoreCase(String anotherString) //比较两个字符串忽略大小写差异
13 boolean equals(Object anObject) //判断此字符串是否与指定对象相同
14 boolean equalsIgnoreCase(String anString) //判断此字符串是否与指定字符串相同，忽略大小写
15
16 //字符串替换

```

```

17 String replace(CharSequence target,CharSequence replacement) //用
    replacement替换所有target
18 String replaceAll(String regex, String replacement) //用replacement
    替换与给定的正则表达式regex匹配的每个子字符串。
19 String replaceFirst(String regex, String replacement) //用replacement
    替换与给定的正则表达式regex匹配的第一个子字符串。
20
21 //字符串分割
22 String[] split(String regex) //返回一个字符串数组,
    将原字符串从正则表达式regex匹配的地方分隔
23 String[] split(String regex,int limit) //返回一个字符串数组,
    将原字符串从正则表达式regex匹配的地方分隔,limit限定匹配个数
24
25 //字符串子串
26 String substring(int start) //从指定位置开始截取字符串,默认到末尾。
27 String substring(int start,int end) //从指定位置开始到指定位置结束截取字符串
    (如果含有起点和终点,Java中一般是左闭右开区间,即end取不到)。
28 CharSequence subSequence(int from,int to) //返回[from,to)的CharSequence
29
30 //字符串拼接
31 String concat(String str) //将指定字符串拼接到当前字符串末尾,可以有
32 static String join(CharSequence link,CharSequence... elements)//用link将多个
    elements相连
33
34 //字符串转换
35 char[] toCharArray() //转化为字符数组
36 String toLowerCase() //将所有字符串转化为小写字母
37 String toUpperCase() //将所有字符串转化为大写字母
38 String trim() //删除字符串的所有前导,尾随空格
39
40 //字符串匹配
41 boolean matches(String regex) //返回这个字符串是否与正则表达式regex
    匹配
42
43 int length() //获取字符串的长度。
44 boolean isEmpty() //判断是否为空串

```

### 3. String与正则表达式

String 和正则表达式 `regex expression` 配合可以增强对字符串的操作能力。

String 类提供了如下和正则表达式相关的操作

```

1 String replaceAll(String regex, String replacement)           //用replacement替
  换与给定的正则表达式regex匹配的每个子字符串。
2 String replaceFirst(String regex, String replacement)         //用replacement替
  换与给定的正则表达式regex匹配的第一个子字符串。
3 String[] split(String regex)                                   //返回一个字符串数组，将
  原字符串从正则表达式regex匹配的地方分隔
4 String[] split(String regex,int limit)                         //返回一个字符串数组，将
  原字符串从正则表达式regex匹配的地方分隔,limit限定匹配个数
5 boolean matches(String regex)                                  //返回这个字符串是否与正则表达式regex匹
  配

```

了解这些方法的使用之前，我们先了解一下正则表达式。

### 3.1. 正则表达式简介

可以将正则表达式理解为描述字符串的一种语法。这里只介绍一些常用、简单的正则表达式用法。

- 例如，要找一个数字，它**可能有一个负号**在最前面，那你就写一个负号加上一个问号，就像这样："-?"
- 在正则表达式中，用"\d"表示一位 数字。**注意Java对反斜线\'\'的特殊对待。**
  - 在Java中，"\\\"的意思是：我要插入一个反斜线。
  - 如果你想表示一位数字，那么在Java中正则表达式应该是"\\d"。
  - 你想在Java中的正则表达式插入一个普通的反斜线："\\\"
  - 不过换行和制表符之类的东西只需使用单反斜线："\\n\\te"
  - 不过换行和制表符之类的东西只需使用单反斜线："\\n\\te"

### 3.2. 常用的正则表达式词法

在以下表格中出现的字符有特殊含义，因此若想表示它的本来含义要在前面用\\转义。

由于Java中的\\也具有转义含义，因此在下列正则表达式中出现的\\在Java字符串中数目加倍

使用()可以将正则表达式分组

字符	说明
\	将下一字符标记为特殊字符、文本、反向引用或八进制转义符。例如, "n"匹配字符"n"。"\n"匹配换行符。序列"\\\\"匹配"\\", "\"匹配"("。
^	匹配输入字符串开始的位置。
\$	匹配输入字符串结尾的位置。
*	<b>零次或多次匹配前面的字符或子表达式。</b> 例如, zo* 匹配"z"和"zoo"。
+	<b>一次或多次匹配前面的字符或子表达式。</b> 例如, "zo+"与"zo"和"zoo"匹配, 但与"z"不匹配。
?	<b>零次或一次匹配前面的字符或子表达式。</b> 例如, "do(es)?"匹配"do"或"does"中的"do"。
.	<b>匹配除"\r\n"之外的任何单个字符。</b>
x y	<b>匹配 x 或 y。</b> 例如, 'z food' 匹配"z"或"food"。'(z f)ood' 匹配"zood"或"food"。
[xyz]	字符集。匹配包含的任一字符。例如, "[abc]"匹配"plain"中的"a"。
[^xyz]	反向字符集。匹配未包含的任何字符。例如, "[^abc]"匹配"plain"中"p", "l", "i", "n"。
[a-z]	字符范围。匹配指定范围内的任何字符。例如, "[a-z]"匹配"a"到"z"范围内的任何小写字母。
[^a-z]	反向范围字符。匹配不在指定的范围内的任何字符。例如, "[^a-z]"匹配任何不在"a"到"z"范围内的任何字符。
\n	换行符匹配。
\b	匹配一个字边界, 即字与空格间的位置。例如, "er\b"匹配"never"中的"er", 但不匹配"verb"中的"er"。
\B	非字边界匹配。"er\B"匹配"verb"中的"er", 但不匹配"never"中的"er"。
\d	<b>数字字符匹配。</b> 等效于 [0-9]。
\D	<b>非数字字符匹配。</b> 等效于 [^0-9]。
\s	<b>匹配任何空白字符</b> , 包括空格、制表符、换页符等。与 [\f\n\r\t\v] 等效。
\S	<b>匹配任何非空白字符。</b> 与 [^\f\n\r\t\v] 等效。
\w	<b>匹配任何字类字符</b> , 包括下划线。与 "[A-Za-z0-9_]"等效。
\W	<b>与任何非单词字符匹配。</b> 与 "[^A-Za-z0-9_]"等效。

更多关于正则表达式的内容请查阅相关资料。

## 3.3. 正则示例

### 3.3.1. String.matches()匹配正则表达式

检查一个 `String` 是否匹配如上所述的正则表达式：

```
1  //: strings/IntegerMatch.java
2  public class IntegerMatch {
3      public static void main(String[] args) {
4          System.out.println("-1234".matches("-?\\d+"));
5          System.out.println("5678".matches("-?\\d+"));
6          System.out.println("+911".matches("-?\\d+"));
7          System.out.println("+911".matches("(+|-)\\d+"));
8      }
9      /* Output;
10     true
11     true
12     false
13     true
14     *///:~
```

前三个都好理解，第四个正则表达式的描述为：“可能以一个加号或减号开头的一位或多位数字”

### 3.3.2. String.split()从正则表达式匹配的地方分离

`String.split()` 方法，其功能是“将字符串从 正则表达式匹配的地方切开。”

```
1  //: strings/SpUtting.java
2  import java.util.*;
3
4  public class Splitting {
5      public static String knights = "Then, when you have found the
6      shrubbery, you must " +
7      "cut down the mightiest tree in the forest... " +
8      "with... a herring!";
9
10     public static void split(String regex) {
11         System.out.println(Arrays.toString(knights.split(regex)));
12     }
13
14     public static void main(String[] args) {
15         split(" "); // 匹配空格
16         split("\\W+"); // 非单词字符
```

```

16         split("n\\w+"); // 'n'后面跟随非单词字符
17     }
18 } /* Output:
19 [Then,, when, you, have, found, the, shrubbery,, you, must, cut, down, the,
20 mightiest, tree, in, the, forest..., with..., a, herring!]
21 [Then, when, you, have, found, the, shrubbery, you, must, cut, down, the,
22 mightiest, tree, in, the, forest, with, a, herring]
23 [The, whe, you have found the shrubbery, you must cut dow, the mightiest
24 tree i, the forest... with... a herring!]
25 *///:~

```

首先看第一个语句，注意这里用的是空白的字符作为正则表达式，其中并不包含任何特殊的字符。因此第一个 `split()`，只是按空格来划分字符串。

第二个和第三个 `split()` 都用到了 `\\w`，它的意思是“非单词字符”(如果 `w` 小写 `\\w`，则表示一个“单词字符”)。通过第二个例子可以看到，它将标点字符删除了(标点符号也是非单词字符)。第三个 `split()` 表示“字母 `n` 后面跟着一个或多个非单词字符。”可以看到，在原始字符串中，与正则表达式匹配的部分，在最终结果中都不存在了。

### 3.3.3. 替换与正则表达式匹配的地方

你可以只替换正则表达式第一个匹配的子串，或是替换所有匹配的地方。

```

1  //: strings/Replacing.java
2  public class Replacing {
3      static String s = "Then, when you have found the shrubbery, you must "
4      +
5          "cut down the mightiest tree in the forest... " +
6          "with... a herring!";
7      public static void main(String[] args) {
8          System.out.println(s.replaceFirst("f\\w+", "located")); //只替换第一个
9                          匹配的地方
10
11         System.out.println(s.replaceAll("shrubbery|tree|herring", "banana")); //替换
12         所有匹配的地方
13     }
14 } /* Output:
15 Then, when you have located the shrubbery, you must cut down the mightiest
16 tree in the forest... with... a herring!
17 Then, when you have found the banana, you must cut down the mightiest
18 banana in the forest... with... a banana!
19 *///:~

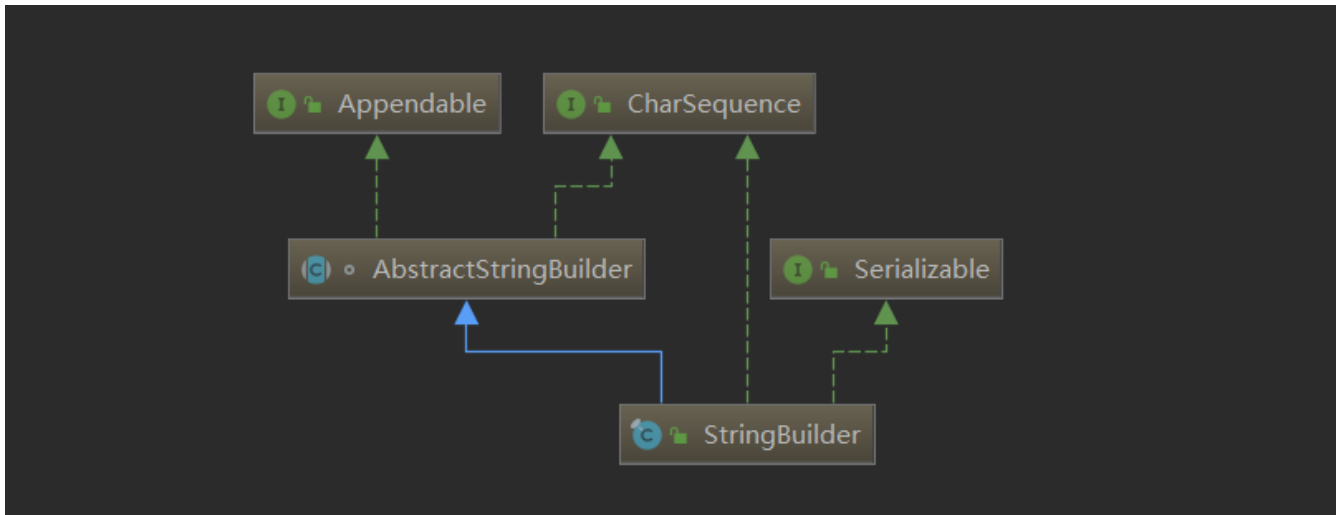
```

第一个表达式要匹配的是，以字母 `f` 开头，后面跟一个或多个字母(注意这里的 `w` 是小写的)。并且只替换掉第一个匹配的部分，所以“found”被替换成“located”。



第二个表达式要匹配的是三个单词中的任意一个，因为它们以竖直线分隔表示“或”，并且替换所有匹配的部分。

## 4. StringBuilder类



注意 `StringBuilder` 是 `CharSequence` 子类，是可以修改的字符串，可以理解为支持动态扩容的字符数组。

### 4.1. StringBuilder的构造方法

```
1  StringBuilder()                // 初始化为空串，并设置容量为16个字节；
2  StringBuilder(CharSequence seq) // 使用seq初始化，容量在此基础上加16；
3  StringBuilder(int capacity)    // 指定容量
4  StringBuilder(String str)      // 使用str初始化，容量str大小的基础上加16；
```

注意扩容过程效率低下，因此调用构造方法时，可以指定容量大小，避免大量的扩容过程。

### 4.2. StringBuilder的常用方法

这里介绍一些字符串更新的方法，用于替换 `String` 类相应的方法，提高效率。

```
1  //字符串添加
2  StringBuilder append(boolean b)                //将boolean类型参数添加到
   末尾。append参数可以是任意的基本参数类型
3  StringBuilder append(char c)
4  StringBuilder append(char[] str)
5  StringBuilder append(char[] str,int start,int end)
6  StringBuilder append(CharSequence s)            //将指定字符序列添加到末
   尾
```

```

7  StringBuilder append(CharSequence s,int start,int end) //将指定字符序列s指定
    范围[start,end)添加到末尾
8
9  //字符串插入, insert的参数类型与append完全一致不再赘述
10 StringBuilder insert(int offset, boolean b) //将 boolean参数的字
    符串表示插入到此序列指定位置
11
12 //字符串查询
13 char charAt(int index) //返回指定索引处的字符
14 void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) //字符从该
    序列复制到目标字符数组 dst 。
15 int indexOf(String str) //返回指定子字符串第一次
    出现的字符串内的索引, 不存在返回-1
16 int indexOf(String str, int fromIndex) //返回指定子串的第一次
    出现在字符串中的索引, 从指定的索引开始搜索
17 int lastIndexOf(String str) //返回指定子字符串最后一
    次出现的字符串内的索引。
18
19 //字符串删除
20 StringBuilder delete(int start,int end) //删除[start,end)的字
    符串
21 StringBuilder delete(int index) //删除指定索引处的的字符
22
23 //反转字符串
24 StringBuilder reverse() //反转字符串
25
26
27 //字符串设置
28 void setCharAt(int index, char ch) //指定索引处的字符设置
    为 ch
29 void setLength(int newLength) //设置字符序列的长度
30
31 //字符串子串
32 CharSequence subSequence(int start, int end) //返回当前字符串的
    [start,end)的子串序列
33 String substring(int start) //返回从start开始到最
    后的子串
34 String substring(int start, int end) //返回一个新的 String
    , 其中包含此序列中当前包含的字符的子序列。
35
36 //改变容量
37 void ensureCapacity(int minimumCapacity) //确保容量至少等于规定
    的最小值。
38 void trimToSize() //调整当前
    StringBuilder的容量, 去除不必要的内存占用。
39 int length()

```

---

## 5. 基本数据类型与字符串的转换

---

### 5.1. 基本类型转换为字符串

1. 使用数值型基本类型的包装类的 `toString()` 方法，转化为十进制形式的字符串

`static String toString(int i, int radix)` 根据指定基数 `radix` 将 `i` 转化为字符串，  
 $2 \leq radix \leq 36$

`static String toHexString(int i)` 将无符号整数 `i` 转化为16进制的字符串形式

`static String toOctalString(int i)` 将无符号整数 `i` 转化为8进制的字符串形式

`static String toBinaryString(int i)` 将无符号整数 `i` 转化为2进制的字符串形式

注意： `Float`、`Double` 没有 `toString(int i, int radix)`、`toOctalString(int i)`、`toBinaryString(int i)`

2. 使用 `String` 类的 `valueOf()` 方法
3. 用一个空字符串加上基本类型——通过重载的 `+` 运算符，得到的就是基本类型数据对应的字符串

### 5.2. 字符串转化为基本类型

1. 使用数值型基本类型包装类的 `valueOf(String s)` 将 `s` 看作十进制整数转化为相应基本数据类型。
2. 重载的 `valueOf(String s, int radix)` 将 `s` 看作 `radix` 为基数转化为相应基本数据类型（适用于：`Integer`、`Long`、`Byte`）。