10. String 类

【本节目标】

1. **认识 String 类**
2. **了解 String 类的基本用法**
3. **熟练掌握 String 类的常见操作**
4. **认识字符串常量池**
5. **认识 StringBuffer 和 StringBuilder**

1. String类的重要性

在C语言中已经涉及到字符串了，但是在C语言中要表示字符串只能使用字符数组或者字符指针，可以使用标准库提供的字符串系列函数完成大部分操作，但是这种将数据和操作数据方法分离开的方式不符合面相对象的思想，而字符串应用又非常广泛，因此Java语言专门提供了String类。

在开发和校招笔试中，字符串也是常客，比如：

字符串转整形数字

字符串相加

而且在面试中也频繁被问到，比如：String、StringBuff和StringBulider之间的区别等。

2. 常用方法

2.1 字符串构造

String类提供的构造方式非常多，常用的就以下三种：

```
public static void main(String[] args) {
    // 使用常量串构造
    String s1 = "hello bit";
    System.out.println(s1);

    // 直接newString对象
    String s2 = new String("hello bit");
    System.out.println(s1);

    // 使用字符数组进行构造
    char[] array = {'h','e','l','l','o','b','i','t'};
    String s3 = new String(array);
    System.out.println(s1);
}
```

其他方法需要用到时，大家参考Java在线文档：String官方文档

【注意】

1. **String是引用类型，内部并不存储字符串本身**，在String类的实现源码中，String类实例变量如下：

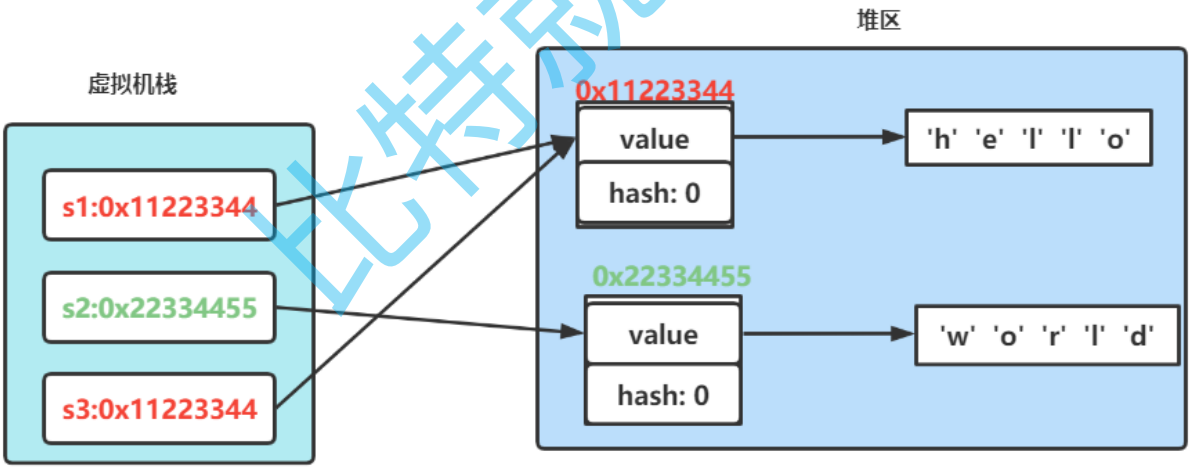
```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {
    /** The value is used for character storage. */
    private final char value[];

    /** Cache the hash code for the string */
    private int hash; // Default to 0
```

在JDK1.8中，字符串实际保存在char类型的数组中

```
public static void main(String[] args) {
    // s1和s2引用的是不同对象  s1和s3引用的是同一对象
    String s1 = new String("hello");
    String s2 = new String("world");
    String s3 = s1;

    System.out.println(s1.length());  // 获取字符串长度---输出5
    System.out.println(s1.isEmpty()); // 如果字符串长度为0，返回true，否则返回false
}
```



2. 在Java中""引起来的也是String类型对象。

```
// 打印"hello"字符串(String对象)的长度
System.out.println("hello".length());
```

2.2 String对象的比较

字符串的比较是常见操作之一，比如：字符串排序。Java中总共提供了4中方式：

1. **==比较是否引用同一个对象**

   注意：对于内置类型，==比较的是变量中的值；对于引用类型==比较的是引用中的地址。

```
public static void main(String[] args) {
    int a = 10;
    int b = 20;
    int c = 10;

    // 对于基本类型变量，==比较两个变量中存储的值是否相同
    System.out.println(a == b);    // false
    System.out.println(a == c);    // true

    // 对于引用类型变量，==比较两个引用变量引用的是否为同一个对象
    String s1 = new String("hello");
    String s2 = new String("hello");
    String s3 = new String("world");
    String s4 = s1;
    System.out.println(s1 == s2);   // false
    System.out.println(s2 == s3);   // false
    System.out.println(s1 == s4);   // true
}
```

2. **`boolean equals(Object anObject)` 方法：按照字典序比较**

   字典序：字符大小的顺序

   String类重写了父类Object中equals方法，Object中equals默认按照==比较，String重写equals方法后，按照如下规则进行比较，比如：`s1.equals(s2)`

```
public boolean equals(Object anObject) {
    // 1. 先检测this 和 anObject 是否为同一个对象比较，如果是返回true
    if (this == anObject) {
        return true;
    }

    // 2. 检测anObject是否为String类型的对象，如果是继续比较，否则返回false
    if (anObject instanceof String) {
        // 将anObject向下转型为String类型对象
        String anotherString = (String)anObject;
        int n = value.length;

        // 3. this和anObject两个字符串的长度是否相同，是继续比较，否则返回false
        if (n == anotherString.value.length) {
            char v1[] = value;
            char v2[] = anotherString.value;
            int i = 0;

            // 4. 按照字典序，从前往后逐个字符进行比较
            while (n-- != 0) {
                if (v1[i] != v2[i])
                    return false;
                i++;
            }
            return true;
        }
    }
```

```
    return false;
}
```

```
public static void main(String[] args) {
    String s1 = new String("hello");
    String s2 = new String("hello");
    String s3 = new String("Hello");

    // s1、s2、s3引用的是三个不同对象，因此==比较结果全部为false
    System.out.println(s1 == s2);      // false
    System.out.println(s1 == s3);      // false

    // equals比较：String对象中的逐个字符
    // 虽然s1与s2引用的不是同一个对象，但是两个对象中放置的内容相同，因此输出true
    // s1与s3引用的不是同一个对象，而且两个对象中内容也不同，因此输出false
    System.out.println(s1.equals(s2)); // true
    System.out.println(s1.equals(s3)); // false
}
```

3. **`int compareTo(String s)` 方法: 按照字典序进行比较**

   与equals不同的是，equals返回的是boolean类型，而compareTo返回的是int类型。具体比较方式：

   1. 先按照字典次序大小比较，如果出现不等的字符，直接返回这两个字符的大小差值
   2. 如果前k个字符相等(k为两个字符长度最小值)，返回值两个字符串长度差值

```
public static void main(String[] args) {
    String s1 = new String("abc");
    String s2 = new String("ac");
    String s3 = new String("abc");
    String s4 = new String("abcdef");
    System.out.println(s1.compareTo(s2));  // 不同输出字符差值-1
    System.out.println(s1.compareTo(s3));  // 相同输出 0
    System.out.println(s1.compareTo(s4));  // 前k个字符完全相同，输出长度差值 -3
}
```

4. `int compareToIgnoreCase(String str)` 方法：与compareTo方式相同，但是忽略大小写比较

```
public static void main(String[] args) {
    String s1 = new String("abc");
    String s2 = new String("ac");
    String s3 = new String("ABc");
    String s4 = new String("abcdef");
    System.out.println(s1.compareToIgnoreCase(s2));  // 不同输出字符差值-1
    System.out.println(s1.compareToIgnoreCase(s3));  // 相同输出 0
    System.out.println(s1.compareToIgnoreCase(s4));  // 前k个字符完全相同，输出长度差值 -3
}
```

2.3 字符串查找

字符串查找也是字符串中非常常见的操作，String类提供的常用查找的方法：

方法	功能
char charAt(int index)	返回index位置上字符，如果index为负数或者越界，抛出IndexOutOfBoundsException异常
int indexOf(int ch)	返回ch第一次出现的位置，没有返回-1
int indexOf(int ch, int fromIndex)	从fromIndex位置开始找ch第一次出现的位置，没有返回-1
int indexOf(String str)	返回str第一次出现的位置，没有返回-1
int indexOf(String str, int fromIndex)	从fromIndex位置开始找str第一次出现的位置，没有返回-1
int lastIndexOf(int ch)	从后往前找，返回ch第一次出现的位置，没有返回-1
int lastIndexOf(int ch, int fromIndex)	从fromIndex位置开始找，从后往前找ch第一次出现的位置，没有返回-1
int lastIndexOf(String str)	从后往前找，返回str第一次出现的位置，没有返回-1
int lastIndexOf(String str, int fromIndex)	从fromIndex位置开始找，从后往前找str第一次出现的位置，没有返回-1


```
public static void main(String[] args) {
    String s = "aaabbbcccaaabbbccc";
    System.out.println(s.charAt(3));                // 'b'
    System.out.println(s.indexOf('c'));             // 6
    System.out.println(s.indexOf('c', 10));         // 15
    System.out.println(s.indexOf("bbb"));           // 3
    System.out.println(s.indexOf("bbb", 10));       // 12
    System.out.println(s.lastIndexOf('c'));         // 17
    System.out.println(s.lastIndexOf('c', 10));     // 8
    System.out.println(s.lastIndexOf("bbb"));       // 12
    System.out.println(s.lastIndexOf("bbb", 10));   // 3
```

注意：上述方法都是实例方法。

2.4 转化

1. **数值和字符串转化**

```
public static void main(String[] args) {
    // 数字转字符串
    String s1 = String.valueOf(1234);
    String s2 = String.valueOf(12.34);
    String s3 = String.valueOf(true);
    String s4 = String.valueOf(new Student("Hanmeimei", 18));
```

```
    System.out.println(s1);
    System.out.println(s2);
    System.out.println(s3);
    System.out.println(s4);
    System.out.println("================================");
    // 字符串转数字
    // 注意：Integer、Double等是Java中的包装类型，这个后面会讲到
    int data1 = Integer.parseInt("1234");
    double data2 = Double.parseDouble("12.34");
    System.out.println(data1);
    System.out.println(data2);
}
```

2. **大小写转换**

```
public static void main(String[] args) {
    String s1 = "hello";
    String s2 = "HELLO";
    // 小写转大写
    System.out.println(s1.toUpperCase());
    // 大写转小写
    System.out.println(s2.toLowerCase());
}
```

3. **字符串转数组**

```
public static void main(String[] args) {
    String s = "hello";
    // 字符串转数组
    char[] ch = s.toCharArray();
    for (int i = 0; i < ch.length; i++) {
        System.out.print(ch[i]);
    }
    System.out.println();
    // 数组转字符串
    String s2 = new String(ch);
    System.out.println(s2);
}
```

4. **格式化**

```
public static void main(String[] args) {
    String s = String.format("%d-%d-%d", 2019, 9,14);
    System.out.println(s);
}
```

2.5 字符串替换

使用一个指定的新的字符串替换掉已有的字符串数据，可用的方法如下：

方法	功能
String replaceAll(String regex, String replacement)	替换所有的指定内容
String replaceFirst(String regex, String replacement)	替换收个内容

代码示例: 字符串的替换处理

```
String str = "helloworld" ;
System.out.println(str.replaceAll("l", "_"));
System.out.println(str.replaceFirst("l", "_"));
```

注意事项: 由于字符串是不可变对象, 替换不修改当前字符串, 而是产生一个新的字符串.

2.6 字符串拆分

可以将一个完整的字符串按照指定的分隔符划分为若干个子字符串。

可用方法如下:

方法	功能
String[] split(String regex)	将字符串全部拆分
String[] split(String regex, int limit)	将字符串以指定的格式，拆分为limit组

代码示例: 实现字符串的拆分处理

```
String str = "hello world hello bit" ;
String[] result = str.split(" ") ; // 按照空格拆分
for(String s: result) {
    System.out.println(s);
}
```

代码示例: 字符串的部分拆分

```
String str = "hello world hello bit" ;
String[] result = str.split(" ",2) ;
for(String s: result) {
    System.out.println(s);
}
```

拆分是特别常用的操作. 一定要重点掌握. 另外有些特殊字符作为分割符可能无法正确切分, 需要加上转义.

代码示例: 拆分IP地址

```
String str = "192.168.1.1" ;
String[] result = str.split("\\.") ;
for(String s: result) {
   System.out.println(s);
}
```

注意事项:

1. 字符"|","*","+"都得加上转义字符，前面加上`"\\"`.
2. 而如果是`"\"`，那么就得写成`"\\\\"`.
3. 如果一个字符串中有多个分隔符，可以用"|"作为连字符.

代码示例: 多次拆分

```
String str = "name=zhangsan&age=18" ;
String[] result = str.split("&") ;
for (int i = 0; i < result.length; i++) {
   String[] temp = result[i].split("=") ;
   System.out.println(temp[0]+" = "+temp[1]);
}
```

这种代码在以后的开发之中会经常出现

2.7 字符串截取

从一个完整的字符串之中截取出部分内容。可用方法如下:

方法	功能
String substring(int beginIndex)	从指定索引截取到结尾
String substring(int beginIndex, int endIndex)	截取部分内容

代码示例: 观察字符串截取

```
String str = "helloworld" ;
System.out.println(str.substring(5));
System.out.println(str.substring(0, 5));
```

注意事项:

1. 索引从0开始
2. 注意前闭后开区间的写法, substring(0, 5) 表示包含 0 号下标的字符, 不包含 5 号下标

2.8 其他操作方法

方法	功能
String trim()	去掉字符串中的左右空格,保留中间空格
String toUpperCase()	字符串转大写
String toLowerCase()	字符串转小写

代码示例: 观察trim()方法的使用

```
String str = "   hello  world   " ;
System.out.println("["+str+"]");
System.out.println("["+str.trim()+"]");
```

trim 会去掉字符串开头和结尾的空白字符(空格, 换行, 制表符等).

代码示例: 大小写转换

```
String str = "   hello%$$%@#$%world 哈哈哈  " ;
System.out.println(str.toUpperCase());
System.out.println(str.toLowerCase());
```

这两个函数只转换字母。

2.9 字符串常量池

2.9.1 创建对象的思考

下面两种创建String对象的方式相同吗?

```
public static void main(String[] args) {
    String s1 = "hello";
    String s2 = "hello";
    String s3 = new String("hello");
    String s4 = new String("hello");
    System.out.println(s1 == s2);    // true
    System.out.println(s1 == s3);    // false
    System.out.println(s3 == s4);    // false
}
```

上述程序创建方式类似，为什么s1和s2引用的是同一个对象，而s3和s4不是呢?

在Java程序中，类似于：1， 2， 3，3.14，“hello”等字面类型的常量经常频繁使用，**为了使程序的运行速度更快、更节省内存**，Java为8种基本数据类型和String类都提供了常量池。

> "池" 是编程中的一种常见的, 重要的**提升效率**的方式, 我们会在未来的学习中遇到各种 "内存池", "线程池", "数据库连接池" ....

> 比如：家里给大家打生活费的方式
>
> 1. 家里经济拮据，每月定时打生活费，有时可能会晚，最差情况下可能需要向家里张口要，速度慢
> 2. 家里有矿，一次性打一年的生活费放到银行卡中，自己随用随取，速度非常快
>
> 方式2，就是池化技术的一种示例，钱放在卡上，随用随取，效率非常高。常见的池化技术比如：数据库连接池、线程池等。

为了节省存储空间以及程序的运行效率，Java中引入了：

1. **Class文件常量池**：每个.Java源文件编译后生成.Class文件中会保存当前类中的字面常量以及符号信息
2. **运行时常量池**：在.Class文件被加载时，.Class文件中的常量池被加载到内存中称为运行时常量池，运行时常量池每个类都有一份
3. **字符串常量池**

此处简单了解下，后序在讲JVM时会给同学们详细阐释。

2.9.2 字符串常量池(StringTable)

字符串常量池在JVM中是StringTable类，实际**是一个固定大小的HashTable**(一种高效用来进行查找的数据结构，后序给大家详细介绍)，不同JDK版本下字符串常量池的位置以及默认大小是不同的：

JDK版本	字符串常量池位置	大小设置
Java6	(方法区)永久代	固定大小：1009
Java7	堆中	可设置，没有大小限制，默认大小：60013
Java8	堆中	可设置，有范围限制，最小是1009

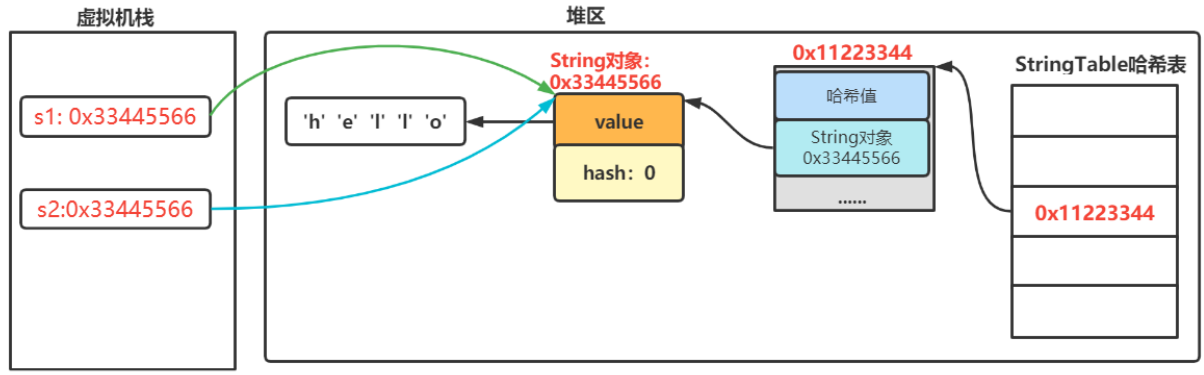
关于方法区、堆等内存结构的具体局部，后序JVM中会给大家详细介绍。

2.9.3 再谈String对象创建

由于不同JDK版本对字符串常量池的处理方式不同，此处在Java8 HotSpot上分析

1. **直接使用字符串常量进行赋值**

```
public static void main(String[] args) {
    String s1 = "hello";
    String s2 = "hello";
    System.out.println(s1 == s2);    // true
}
```

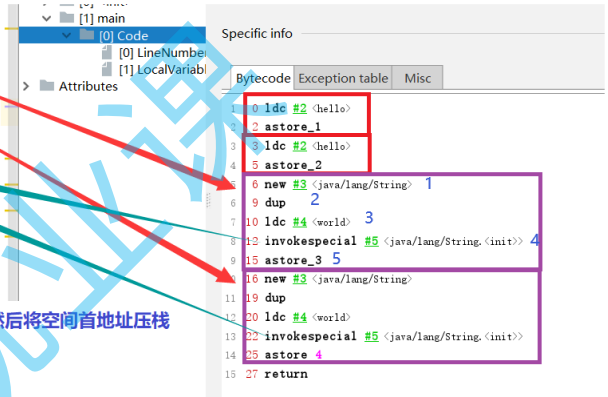


说明：
1. 在字节码文件加载时，"hello"和"world"常量串已经创建好了，并保存在字符串常量池中
2. 当使用 String s1 = "hello"; 创建对象时，先在字符串常量池中找，找到了，将该字符串引用赋值给s1

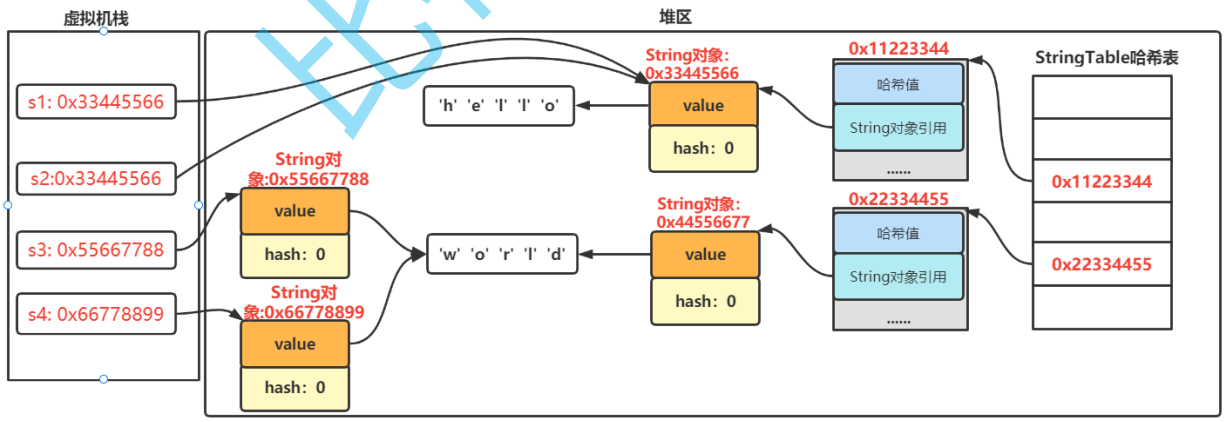
2. 通过new创建String类对象

```
public static void main(String[] args) {
    String s1 = "hello";
    String s2 = "hello";
    String s3 = new String("world");
    String s4 = new String("world");
}
```

```
public String( @NotNull String original) {
    this.value = original.value;
    this.hash = original.hash;
}
```



1. new: 在堆上开辟String对象大小的空间，并将对象中成员初始化成0，然后将空间首地址压栈
2. dup：将栈顶元素，即String对象空间的首地址拷贝一份到栈顶备用
3. ldc：将常量池中"world"对象引用拷贝到栈顶
4. invokespecial：调用String类构造方法，此时：要取走栈顶2个元素
   即："world"对象的引用 以及 第一步new的空间String空间的地址
5. astore2：用栈顶元素给s2赋值



结论：只要是new的对象，都是唯一的。

通过上面例子可以看出：使用常量串创建String类型对象的效率更高，而且更节省空间。用户也可以将创建的字符串对象通过 intern 方式添加进字符串常量池中。

3. intern方法

intern 是一个native方法(Native方法指：底层使用C++实现的，看不到其实现的源代码)，**该方法的作用是手动将创建的String对象添加到常量池中。**

```
public static void main(String[] args) {
    char[] ch = new char[]{'a', 'b', 'c'};
    String s1 = new String(ch);    // s1对象并不在常量池中
    //s1.intern();              // s1.intern()；调用之后，会将s1对象的引用放入到常量池中
    String s2 = "abc";       // "abc" 在常量池中存在了，s2创建时直接用常量池中"abc"的引用
    System.out.println(s1 == s2);
}

// 输出false
// 将上述方法打开之后，就会输出true
```

注意：在Java6 和 Java7、8中Intern的实现会有些许的差别。

> **面试题：请解释String类中两种对象实例化的区别**
>
> JDK1.8中
>
> 1. **String str = "hello"**
>
>   只会开辟一块堆内存空间，保存在字符串常量池中，然后str共享常量池中的String对象
>
> 2. **String str = new String("hello")**
>
>   会开辟两块堆内存空间，字符串"hello"保存在字符串常量池中，然后用常量池中的String对象给新开辟的String对象赋值。
>
> 3. **String str = new String(new char[]{'h', 'e', 'l', 'l', 'o'})**
>
>   现在堆上创建一个String对象，然后利用copyof将重新开辟数组空间，将参数字符串数组中内容拷贝到String对象中

2.10 字符串的不可变性

String是一种不可变对象. 字符串中的内容是不可改变。字符串不可被修改，是因为：

1. **String类在设计时就是不可改变的，String类实现描述中已经说明了**

   以下来自JDK1.8中String类的部分实现：

```
/**
 * The {@code String} class represents character strings. All
 * string literals in Java programs, such as {@code "abc"}, are
 * implemented as instances of this class.
 * <p>      Strings是不可改变的，它们的内容在创建好之后就不能被修改
 * Strings are constant; their values cannot be changed after they
 * are created. String buffers support mutable strings.
 * Because String objects are immutable they can be shared. For example:
 * <blockquote><pre>
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {
    /** The value is used for character storage. */
    private final char value[];

    /** Cache the hash code for the string */
    private int hash; // Default to 0
```

String类中的字符实际保存在内部维护的value字符数组中，该图还可以看出：

1. String类被final修饰，表明该类不能被继承
2. value被修饰被final修饰，表明value自身的值不能改变，即不能引用其它字符数组，但是其引用空间中的内容可以修改。

2. **所有涉及到可能修改字符串内容的操作都是创建一个新对象，改变的是新对象**

比如`replace`方法：

```
@NotNull public String replace(char oldChar, char newChar) {
    if (oldChar != newChar) {
        int len = value.length;
        int i = -1;
        char[] val = value; /* avoid getfield opcode */

        while (++i < len) {
            if (val[i] == oldChar) {
                break;
            }
        }
        if (i < len) {          在新空见上进行修改
            char buf[] = new char[len];
            for (int j = 0; j < i; j++) {
                buf[j] = val[j];
            }
            while (i < len) {
                char c = val[i];
                buf[i] = (c == oldChar) ? newChar : c;
                i++;            如果要修改实际是创建了一个新的对象
            }
            return new String(buf,  share: true);
        }
```

【纠正】网上有些人说：字符串不可变是因为其内部保存字符的数组被final修饰了，因此不能改变。

这种说法是错误的，不是因为String类自身，或者其内部value被final修饰而不能被修改。

final修饰类表明该类不想被继承，final修饰引用类型表明该引用变量不能引用其他对象，但是其引用对象中的内容是可以修改的。

```
public static void main(String[] args) {
   final int array[] = {1,2,3,4,5};
   array[0] = 100;
   System.out.println(Arrays.toString(array));
   // array = new int[]{4,5,6};   // 编译报错：Error:(19, 9) java: 无法为最终变量array分配值
}
```

> **为什么 String 要设计成不可变的?**(不可变对象的好处是什么?) (选学)
>
> 1. 方便实现字符串对象池. 如果 String 可变, 那么对象池就需要考虑写时拷贝的问题了.
> 2. 不可变对象是线程安全的.
> 3. 不可变对象更方便缓存 hash code, 作为 key 时可以更高效的保存到 HashMap 中.

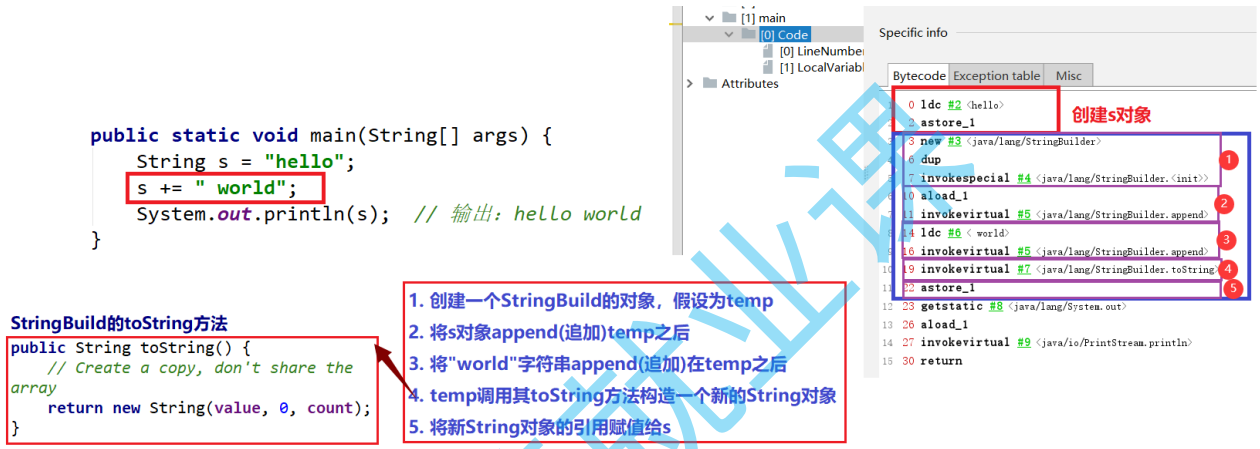
那如果想要修改字符串中内容，该如何操作呢？

2.11 字符串修改

注意：尽量避免直接对String类型对象进行修改，因为String类是不能修改的，所有的修改都会创建新对象，效率非常低下。

```
public static void main(String[] args) {
    String s = "hello";
    s += " world";
    System.out.println(s);  // 输出：hello world
}
```

但是这种方式不推荐使用，因为其效率非常低，中间创建了好多临时对象。




```
public static void main(String[] args) {
    long start = System.currentTimeMillis();
    String s = "";
    for(int i = 0; i < 100000; ++i){
        s += i;
    }
    long end = System.currentTimeMillis();
    System.out.println(end - start);

    start = System.currentTimeMillis();
    StringBuffer sbf = new StringBuffer("");
    for(int i = 0; i < 100000; ++i){
        sbf.append(i);
    }
    end = System.currentTimeMillis();
    System.out.println(end - start);

    start = System.currentTimeMillis();
    StringBuilder sbd = new StringBuilder();
    for(int i = 0; i < 100000; ++i){
        sbd.append(i);
    }

    end = System.currentTimeMillis();
```

```
    System.out.println(end - start);
}
```

输出：

```
27199
3
2
```

可以看待在对String类进行修改时，效率是非常慢的，因此：尽量避免对String的直接需要，如果要修改建议尽量使用StringBuffer或者StringBuilder。

b. **借助StringBuffer 和 StringBuilder**

3. StringBuilder和StringBuffer

3.1 StringBuilder的介绍

由于String的不可更改特性，为了方便字符串的修改，Java中又提供StringBuilder和StringBuffer类。这两个类大部分功能是相同的，这里介绍 StringBuilder常用的一些方法，其它需要用到了大家可参阅 StringBuilder在线文档

方法	说明
StringBuff append(String str)	在尾部追加，相当于String的+=，可以追加：boolean、char、char[]、double、float、int、long、Object、String、StringBuff的变量
char charAt(int index)	获取index位置的字符
int length()	获取字符串的长度
int capacity()	获取底层保存字符串空间总的大小
void ensureCapacity(int mininmumCapacity)	扩容
void setCharAt(int index, char ch)	将index位置的字符设置为ch
int indexOf(String str)	返回str第一次出现的位置
int indexOf(String str, int fromIndex)	从fromIndex位置开始查找str第一次出现的位置
int lastIndexOf(String str)	返回最后一次出现str的位置
int lastIndexOf(String str, int fromIndex)	从fromIndex位置开始找str最后一次出现的位置
StringBuff insert(int offset, String str)	在offset位置插入：八种基类类型 & String类型 & Object类型数据
StringBuffer deleteCharAt(int index)	删除index位置字符
StringBuffer delete(int start, int end)	删除[start, end)区间内的字符
StringBuffer replace(int start, int end, String str)	将[start, end)位置的字符替换为str
String substring(int start)	从start开始一直到末尾的字符以String的方式返回
String substring(int start，int end)	将[start, end)范围内的字符以String的方式返回
StringBuffer reverse()	反转字符串
String toString()	将所有字符按照String的方式返回


```
public static void main(String[] args) {
    StringBuilder sb1 = new StringBuilder("hello");
    StringBuilder sb2 = sb1;
    // 追加：即尾插-->字符、字符串、整形数字
```

```
        sb1.append(' ');              // hello
        sb1.append("world");          // hello world
        sb1.append(123);              // hello world123
        System.out.println(sb1);      // hello world123
        System.out.println(sb1 == sb2);  // true
        System.out.println(sb1.charAt(0));  // 获取0号位上的字符 h
        System.out.println(sb1.length());  // 获取字符串的有效长度14
        System.out.println(sb1.capacity()); // 获取底层数组的总大小
        sb1.setCharAt(0, 'H');    // 设置任意位置的字符 Hello world123
        sb1.insert(0, "Hello world!!!");      // Hello world!!!Hello world123
        System.out.println(sb1);
        System.out.println(sb1.indexOf("Hello"));       // 获取Hello第一次出现的位置
        System.out.println(sb1.lastIndexOf("hello")); // 获取hello最后一次出现的位置
        sb1.deleteCharAt(0);                    // 删除首字符
        sb1.delete(0,5);                        // 删除[0, 5)范围内的字符

        String str = sb1.substring(0, 5);             // 截取[0, 5)区间中的字符以String的方式返回
        System.out.println(str);
        sb1.reverse();                // 字符串逆转
        str = sb1.toString();          // 将StringBuffer以String的方式返回
        System.out.println(str);
    }
```

从上述例子可以看出：String和StringBuilder最大的区别在于**String的内容无法修改，而StringBuilder的内容可以修改。**频繁修改字符串的情况考虑使用StringBuilder。

注意：String和StringBuilder类不能直接转换。如果要想互相转换，可以采用如下原则:

- String变为StringBuilder: 利用StringBuilder的构造方法或append()方法
- StringBuilder变为String: 调用toString()方法。

3.2 面试题：

1. String、StringBuffer、StringBuilder的区别
    - String的内容不可修改，StringBuffer与StringBuilder的内容可以修改.
    - StringBuffer与StringBuilder大部分功能是相似的
    - StringBuffer采用同步处理，属于线程安全操作；而StringBuilder未采用同步处理，属于线程不安全操作
2. 以下总共创建了多少个String对象【前提不考虑常量池之前是否存在】

```
String str = new String("ab");  // 会创建多少个对象
String str = new String("a") + new String("b");  // 会创建多少个对象
```

4. String类oj

1. 第一个只出现一次的字符

```
class Solution {
    public int firstUniqChar(String s) {
        int[] count = new int[256];
```

```
        // 统计每个字符出现的次数
        for(int i = 0; i < s.length(); ++i){
            count[s.charAt(i)]++;
        }

        // 找第一个只出现一次的字符
        for(int i = 0; i < s.length(); ++i){
            if(1 == count[s.charAt(i)]){
                return i;
            }
        }

        return -1;
    }
}
```

2. 最后一个单词的长度

```
import java.util.Scanner;

public class Main{
    public static void main(String[] args){
        // 循环输入
        Scanner sc = new Scanner(System.in);
        while(sc.hasNext()){
            // 获取一行单词
            String s = sc.nextLine();

            // 1. 找到最后一个空格
            // 2. 获取最后一个单词：从最后一个空格+1位置开始，一直截取到末尾
            // 3. 打印最后一个单词长度
            int len = s.substring(s.lastIndexOf(' ')+1, s.length()).length();
            System.out.println(len);
        }
        sc.close();
    }
}
```

3. 检测字符串是否为回文

```
class Solution {
    public static boolean isValidChar(char ch){
        if((ch >= 'a' && ch <= 'z') ||
           (ch >= '0' && ch <= '9')){
            return true;
        }

        return false;
    }


    public boolean isPalindrome(String s) {
```

```
        // 将大小写统一起来
        s = s.toLowerCase();
        int left = 0, right = s.length()-1;
        while(left < right){
            // 1. 从左侧找到一个有效的字符
            while(left < right && !isValidChar(s.charAt(left))){
                left++;
            }

            // 2. 从右侧找一个有效的字符
            while(left < right && !isValidChar(s.charAt(right))){
                right--;
            }

            if(s.charAt(left) != s.charAt(right)){
                return false;
            }else{
                left++;
                right--;
            }
        }

        return true;
    }
}
```