14.字符串-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向对象的思想，而字符串应用又非常广泛，因此Java语言专门提供了String类。

2. String原理介绍

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：

```
1   public static void main(String[] args) {
2       // 使用字符串常量进行赋值
3       String s1 = "hello bit";
4       System.out.println(s1);
5
6       // 直接new String对象
7       String s2 = new String("hello bit");
8       System.out.println(s1);
9
10      // 使用字符数组进行构造
11      char[] array = {'h','e','l','l','o','b','i','t'};
12      String s3 = new String(array);
13      System.out.println(s1);
14
15      //使用字节数组 构造对象
16      byte[] bytes = {97,98,99,100};
17      String s4 = new String(bytes);
```

```
18        System.out.println(str);
19    }
```

其他构造方法需要用到时，大家参考Java在线文档：

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html

查看String源码，我们观察到：**String是引用类型，内部并不存储字符串本身。**

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence,
               Constable, ConstantDesc {

    The value is used for character storage.
    Implementation Note: This field is trusted by the VM, and is a subject to constant folding if String instance is constant. Overwriting this field after construction will cause problems. Additionally, it is marked with Stable to trust the contents of the array. No other facility in JDK provides this functionality (yet). Stable is safe here, because value is never null.
    @Stable
    private final byte[] value;

    The identifier of the encoding used to encode the bytes in value. The supported values in this implementation are LATIN1 UTF16
    Implementation Note: This field is trusted by the VM, and is a subject to constant folding if String instance is constant. Overwriting this field after construction will cause problems.
    private final byte coder;

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

    Cache if the hash has been calculated as actually being zero, enabling us to avoid recalculating this.
    private boolean hashIsZero; // Default to false;

    use serialVersionUID from JDK 1.0.2 for interoperability
    @java.io.Serial
    private static final long serialVersionUID = -6849794470754667710L;
```

2.2 字符串常量池（StringTable）

想要了解字符串的布局，我们首先得了解一个新的内存叫做：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，后续给大家详细介绍)，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，后续JVM中会给大家详细介绍。

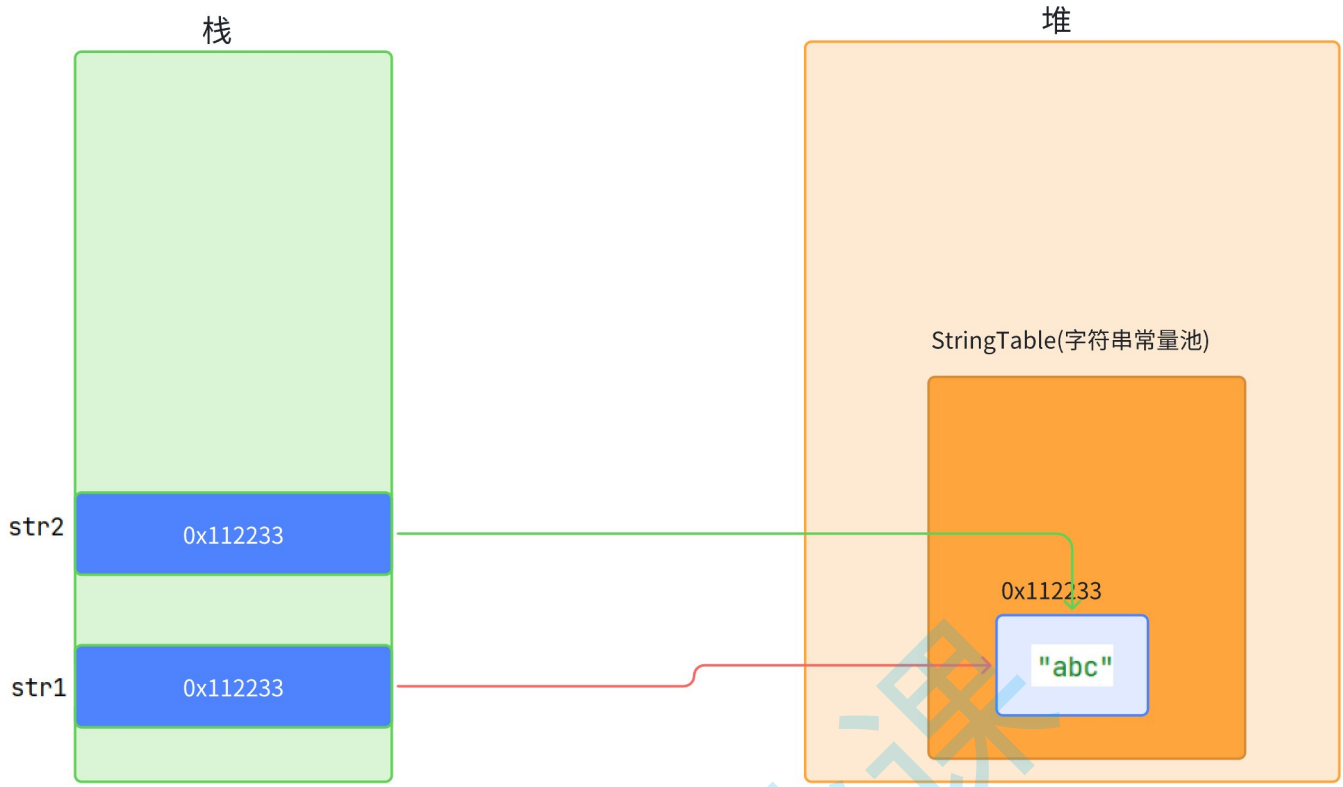
关于“池”的理解：

> "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。

2.3 字符串内存存储

示例1：

```
1   public static void main(String[] args) {
2       String str1 = "abc";
3       String str2 = "abc";
4       System.out.println(str1 == str2);
5    }
```

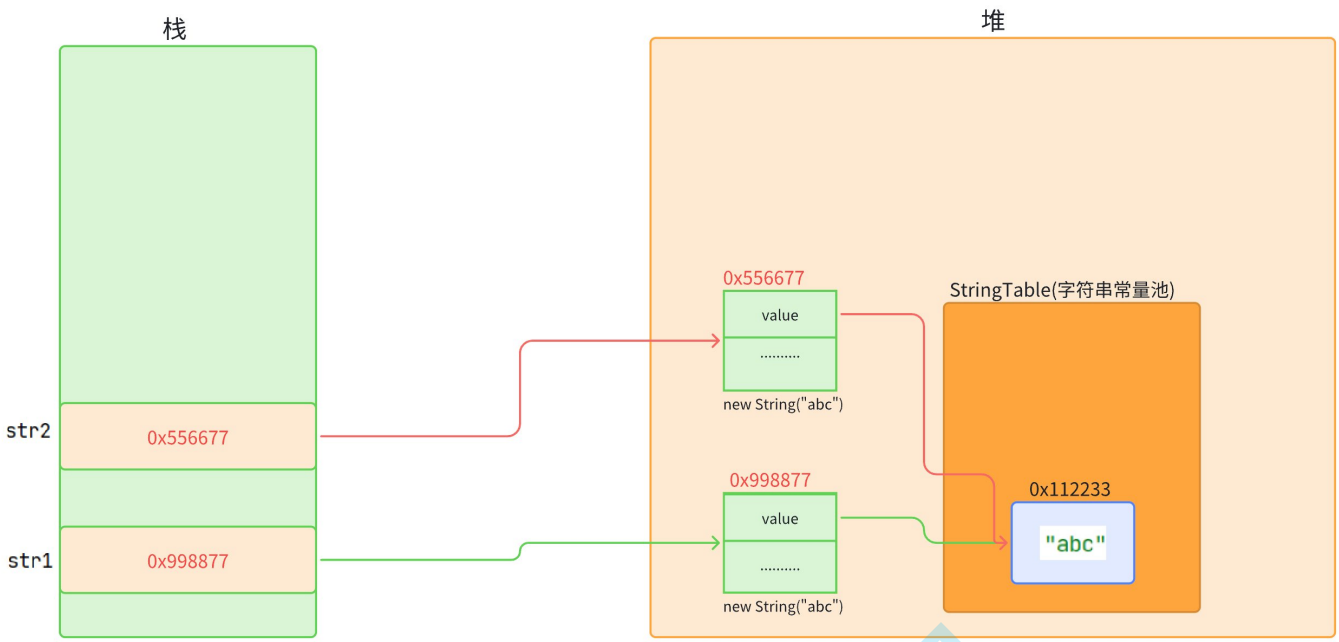


说明：

- str1在进行存储的时候，"abc"会先存储到字符串常量池当中
- 当str2再次存储的时候，先会检查字符串常量池当中是否存在"abc"常量，如果存在，则不再重复存储。

示例2：

```
1   public static void main(String[] args) {
2       String str1 = new String("abc");
3       String str2 = new String("abc");
4       System.out.println(str1 == str2);
5    }
```



说明：

- 第一次存储的时候，会将"abc"存储到常量池当中
- 每次new都会在堆中实例化新的对象
- 存储str2的时候，会使用常量池的"abc"对象进行存储

3. 常用方法介绍

3.1 String对象的比较

字符串的比较是常见操作之一，比如：当你登录一个网站的时候，输入了用户名和密码之后，需要后台进行比较和对比。

3.1.1 ==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。

```
1   public static void main(String[] args) {
2       int a = 10;
3       int b = 20;
4       int c = 10;
5
6       // 对于基本类型变量，==比较两个变量中存储的值是否相同
7       System.out.println(a == b);    // false
8       System.out.println(a == c);    // true
9
10      // 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
11      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
    }
```

3.1.2 `equals` 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)`

JDK17当中的源码部分：

```
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    return (anObject instanceof String aString)
            && (!COMPACT_STRINGS || this.coder == aString.coder)
            && StringLatin1.equals(value, aString.value);
}


@IntrinsicCandidate
public static boolean equals(byte[] value, byte[] other) {
    if (value.length == other.length) {
        for (int i = 0; i < value.length; i++) {
            if (value[i] != other[i]) {
                return false;
            }
        }
        return true;
    }
    return false;
}
```

使用示例：

```
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
    }
```

3.1.3 `compareTo` 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
    }
```

3.1.4 `compareToIgnoreCase` 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
    }
```

3.2 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
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

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

3.3 转换

3.3.1 数值和字符串转化

```java
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);
}
```

3.3.2 大小写转换

```java
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());
}
```

3.3.3 字符串转数组

```java
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();
```

```
9       // 数组转字符串
10      String s2 = new String(ch);
11      System.out.println(s2);
12  }
```

3.3.4 格式化

```
1   public static void main(String[] args) {
2       String s = String.format("%d-%d-%d", 2019, 9,14);
3       System.out.println(s);
4   }
```

3.4 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

```
1   String str = "helloworld" ;
2   System.out.println(str.replaceAll("l", "_"));
3   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.

3.5 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

```java
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

```java
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

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

- 其中第一个 `\` 告诉编译器：“小伙子，我接下来要用一个特殊的字符表示法"。第二个 `\` 是我们实际想要在字符串中表示的那个反斜杠。”
- 然后这个 `\.` 被传递给正则表达式引擎，在那里它被表示为一个字面上的点号。

注意事项:

1. 字符"|","*","+"都得加上转义字符，前面加上 `"\\"`.

> 同理上述示例的点号

2. 而如果是 `"\"` ，那么就得写成 `"\\\\"`.

> - 正则表达式需要 `\\` 来匹配一个反斜杠。
> - 在Java字符串中表示 `\\` ，我们需要 `\\\\`。

3. 如果一个字符串中有多个分隔符，可以用"|"作为连字符.

> - `|` 在正则表达式中表示"或"，意味着匹配它左边或右边的表达式。

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]);
}
```

3.6 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

```
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

3.7 去除左右两边的空格

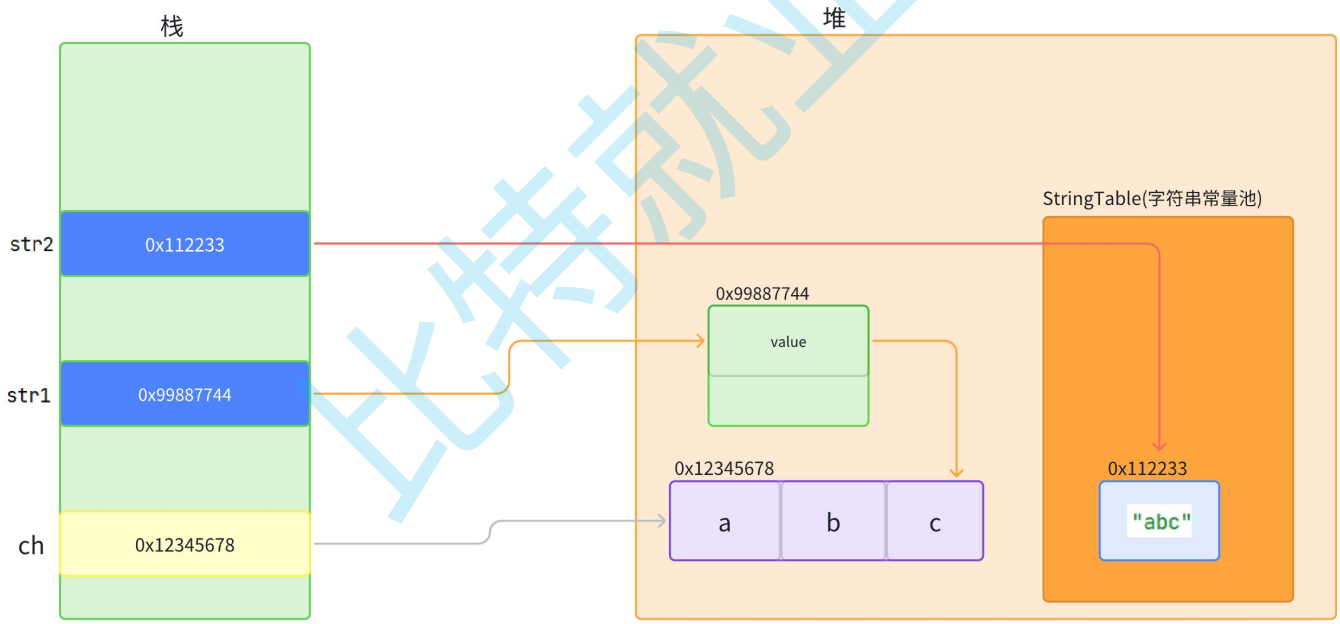
方法	功能
String trim()	去掉字符串中的左右空格,保留中间空格

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

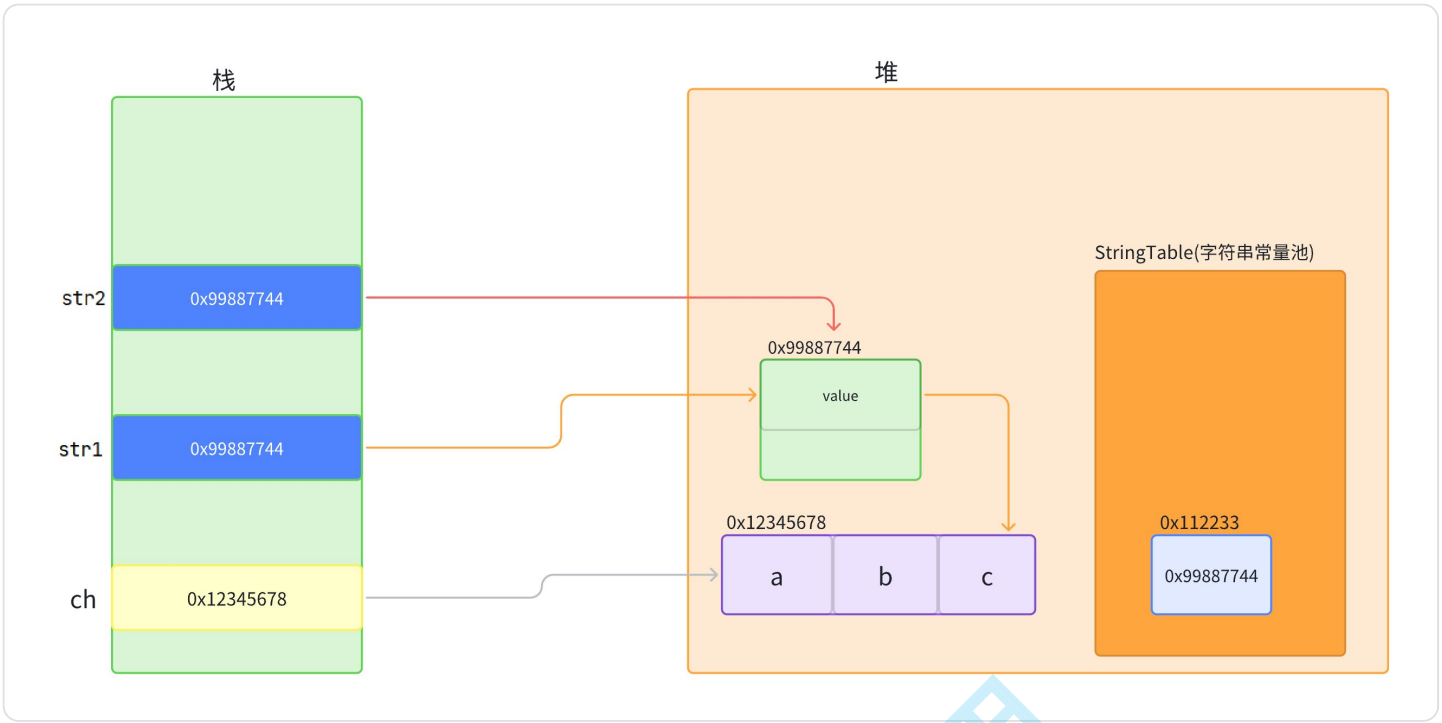
3.8 intern方法

- 当调用 `intern()` 方法时，如果字符串常量池中已经包含一个等于此String对象的字符串（由equals(Object)方法确定），则返回常量池中的字符串。
- 否则，将此String对象添加到常量池中，并返回此String对象的引用。

```
1   public static void main(String[] args) {
2       char[] ch = new char[]{'a', 'b', 'c'};
3       String s1 = new String(ch);      // s1对象并不在常量池中
4       //s1.intern();                   // s1.intern(); 调用之后，会将s1对象的引用放入
    到常量池中
5       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
6       System.out.println(s1 == s2);
7   }
8
9   // 输出false
10  // 将上述方法打开之后，就会输出true
```



如果将代码注释放开，此时示例图如下：



4. 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**

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

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:

```
String str = "abc";
```

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**

```java
public static String toUpperCase(String str, byte[] value, Locale locale) {
    if (locale == null) {
        throw new NullPointerException();
    }
    int first;
    final int len = value.length;

    // Now check if there are any characters that need to be changed, or are surrogate
    for (first = 0 ; first < len; first++ ) {
        int cp = value[first] & 0xff;
        if (cp != CharacterDataLatin1.instance.toUpperCaseEx(cp)) {   // no need to check Cha
            break;
        }
    }
    if (first == len) {
        return str;
    }
    String lang = locale.getLanguage();
    if (lang == "tr" || lang == "az" || lang == "lt") {
        return toUpperCaseEx(str, value, first, locale, localeDependent: true);
    }
    byte[] result = new byte[len];
    System.arraycopy(value, srcPos: 0, result, destPos: 0, first);  // Just copy the first few
                                                                    // upperCase characters.
    for (int i = first; i < len; i++) {
        int cp = value[i] & 0xff;
        cp = CharacterDataLatin1.instance.toUpperCaseEx(cp);
        if (!canEncode(cp)) {                                       // not a latin1 character
            return toUpperCaseEx(str, value, first, locale, localeDependent: false);
        }
        result[i] = (byte)cp;
    }
    return new String(result, LATIN1);
}
```

3. String之所以不可以被修改的原因是内部数组因为为私有成员

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence,
               Constable, ConstantDesc {
```

> The value is used for character storage.
>
> Implementation Note: This field is trusted by the VM, and is a subject to constant folding if String instance is constant. Overwriting this field after construction will cause problems. Additionally, it is marked with `Stable` to trust the contents of the array. No other facility in JDK provides this functionality (yet). `Stable` is safe here, because value is never null.

```
    @Stable
    private final byte[] value;
```

此时final修饰该数组表示，该数组不能指向其他的对象，但是可以修改对象内部的数据。其次String类没有提供任何方法来修改 `value` 数组的内容。

> `@Stable`：这是一个JVM优化注解，提示JVM这个字段的值在初始化后不会改变，可以进行某些优化。

5. 字符串修改

对于String类本身来说，是不可以修改的。形如如下代码：

```
1   public static void main(String[] args) {
2       String s = "hello";
3       s = s + " world";
4       System.out.println(s);  // 输出: hello world
5   }
```

在这个例子中，`s + " world"` 创建了一个新的String对象，而原来的"Hello"对象保持不变. 对于String的拼接来说，如果是在循环当中会产生很多的临时对象。此时Java提供了**StringBuffer 和 StringBuilder**。

我们通过下面的例子直观感受一下运行的效率：

```
1   public static void main(String[] args) {
2       long start = System.currentTimeMillis();
3       String s = "";
4       for(int i = 0; i < 10000; ++i){
5           s += i;
6       }
7       long end = System.currentTimeMillis();
8       System.out.println(end - start);
```

```
9
10      start = System.currentTimeMillis();
11      StringBuffer sbf = new StringBuffer("");
12      for(int i = 0; i < 10000; ++i){
13          sbf.append(i);
14      }
15      end = System.currentTimeMillis();
16      System.out.println(end - start);
17
18      start = System.currentTimeMillis();
19      StringBuilder sbd = new StringBuilder();
20      for(int i = 0; i < 10000; ++i){
21          sbd.append(i);
22      }
23      end = System.currentTimeMillis();
24      System.out.println(end - start);
25  }
```

可以看到在对String类进行拼接时，效率是非常慢，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。

6. StringBuilder和StringBuffer

6.1 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 https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/StringBuilder.html

方法	说明
StringBuff append(String str)	在尾部追加，相当于String的+=，可以追加：bool double、float、int、long、Object、String、
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
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
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
StringBuff insert(int offset, String str)	在offset位置插入：八种基类类型 & String类型
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
String substring(int start)	从start开始一直到末尾的字符以String的
String substring(int start，int end)	将[start, end)范围内的字符以String的
StringBuffer reverse()	反转字符串
String toString()	将所有字符按照String的方式返

```
1   public static void main(String[] args) {
2       StringBuilder sb1 = new StringBuilder("hello");
3       StringBuilder sb2 = sb1;
4       // 追加：即尾插-->字符、字符串、整形数字
5       sb1.append(' ');                    // hello
6       sb1.append("world");                // hello world
7       sb1.append(123);                    // hello world123
8       System.out.println(sb1);            // hello world123
9       System.out.println(sb1 == sb2);     // true
10      System.out.println(sb1.charAt(0));    // 获取0号位上的字符  h
11      System.out.println(sb1.length());     // 获取字符串的有效长度14
12      System.out.println(sb1.capacity());   // 获取底层数组的总大小
13      sb1.setCharAt(0, 'H');      // 设置任意位置的字符 Hello world123
14      sb1.insert(0, "Hello world!!!");          // Hello world!!!Hello world123
```

```
15        System.out.println(sb1);
16        System.out.println(sb1.indexOf("Hello"));          // 获取Hello第一次出现的
      位置
17        System.out.println(sb1.lastIndexOf("hello"));  // 获取hello最后一次出现的位置
18        sb1.deleteCharAt(0);                                // 删除首字符
19        sb1.delete(0,5);                                    // 删除[0, 5)范围内的字符
20
21        String str = sb1.substring(0, 5);                   // 截取[0, 5)区间中的字符
      以String的方式返回
22        System.out.println(str);
23        sb1.reverse();                          // 字符串逆转
24        str = sb1.toString();                   // 将StringBuffer以String的方式返回
25        System.out.println(str);
26    }
```

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。

6.2 StringBuffer的介绍

StringBuffer和StringBuilder从方法的功能上来说没有太大区别，这里不做单独介绍，可以查看官方文档：https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/StringBuffer.html

唯一的区别在于方法的声明上会存在一下区别：

```
@Override
@IntrinsicCandidate
public StringBuilder append(String str) {
    super.append(str);
    return this;
}
```

```
@Override
@IntrinsicCandidate
public synchronized StringBuffer append(String str) {
    toStringCache = null;
    super.append(str);
    return this;
}
```

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

7. 小试牛刀-String类oj

1. 第一个只出现一次的字符

387. 字符串中的第一个唯一字符 - 力扣（LeetCode）

```
1   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. 字符串最后一个单词的长度

字符串最后一个单词的长度_牛客题霸_牛客网

```
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. 验证回文串

125. 验证回文串 - 力扣（LeetCode）

```java
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;
    }
}
```

完