

极客学院
jikexueyuan.com

名企数据结构面试题之字符串（上）

名企数据结构面试题之字符串（上） — 课程概要

- 揭开JDK源变量的神秘面纱
- 深入理解引用与引用传递
- 透彻分析String、StringBuilder和StringBuffer
- C语言风格字符串
- 最后一个单词的长度

揭开JDK源变量的神秘面纱

揭开JDK源变量的神秘面纱

- 普通JDK的缺陷
- fastdebug的安装
- OpenJDK简介
- C语言环境的测试
- 环境安装失败的解决方案

揭开JDK源变量的神秘面纱 — 普通JDK的缺陷

普通JDK，或者MyEclipse自带的JRE，无法查看JDK源码的局部变量。

```
public synchronized StringBuffer append(String str) {  
    super.append(str);  
    return this;  
}
```

```
/**
```

```
 * Appends the specified StringBuffer to the end of this StringBuffer.
```

```
 * <p>
```

```
 * The characters of the StringBuffer are appended to the end of this StringBuffer in order, to the contents of this StringBuffer.
```

```
 * @param str the StringBuffer to be appended to this StringBuffer.
```

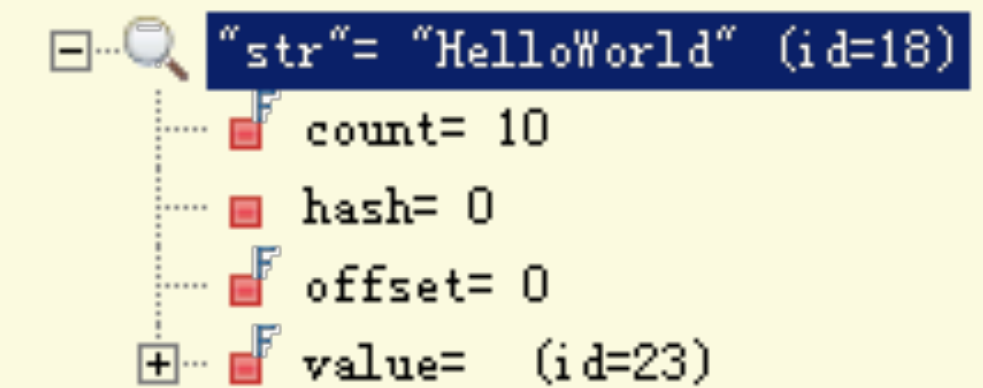
“str”
str cannot be resolved to a variable

揭开JDK源变量的神秘面纱 — fastdebug的安装

安装了fastdebug之后，就能随心所欲地调试JDK源码了！

```
public synchronized StringBuffer append(String str) {  
    super.append(str);  
    return this;  
}
```

```
/**  
 * Appends the specified <tt>StringBuffer</tt> t  
 * <p>  
 * The characters of the <tt>StringBuffer</tt> a  
 * in order, to the contents of this <tt>StringB
```



A screenshot of the Fastdebug variable view for the variable 'str'. The view shows a tree structure with the following fields: 'count' (value 10), 'hash' (value 0), 'offset' (value 0), and 'value' (value (id=23)). The 'value' field is expanded, showing the string 'HelloWorld'.

```
"str" = "HelloWorld" (id=18)  
├── count= 10  
├── hash= 0  
├── offset= 0  
└── value= (id=23)  
    HelloWorld
```

揭开JDK源变量的神秘面纱 — OpenJDK简介

很不幸，fastdebug的官网下载链接已经关闭！

Linux、Mac系统上，推荐使用OpenJDK。

<http://openjdk.java.net/>

OpenJDK比fastdebug更为强大，可以查看到native方法的源码。

揭开JDK源变量的神秘面纱 — C语言环境的测试

Windows: <http://www.jikexueyuan.com/course/424.html>

Linux: <http://www.jikexueyuan.com/course/423.html>

Mac: <http://www.jikexueyuan.com/course/466.html>

一共两处用到C语言：

- C语言风格字符串
- 内存中的栈与堆，数据结构中的栈与堆详解

揭开JDK源变量的神秘面纱 — **环境安装失败的解决方案**

初学者最忌讳自己摸索环境配置、插件安装问题！

在校生，如果“Java环境变量”之类的简单问题在你手上停留了超过2小时，果断问老师、问同学。

因为你需要的是**迅速建立**“Hello World”的成就感！

大学期间，不建议摸索**过多的**负载均衡配置、分布式系统、集群服务器等等问题。

因为名企面试官偏重考查应聘者的编程能力！

揭开JDK源变量的神秘面纱 — 环境安装失败的解决方案

工作后，“环境配置”类问题的一般解决方案：

- 大公司：自己摸索；问同事、问架构师
- 小公司：自己摸索；尝试其它解决方案

| 曾经遇到过的问题 | 最终解决方案 |
|---------------------|---------------------------------|
| Spring MVC表单标签有乱码 | 换成Struts2标签或者JSTL标签 |
| JFreeChart无法在前台显示图片 | 去掉JFreeChart相关jar包，换成纯前端的jqplot |
| 更新maven项目之后，发现依赖冲突 | 暂时注释掉同事新加的maven依赖，先写好自己的模块 |

不能在一棵树上吊死，要灵活机智！

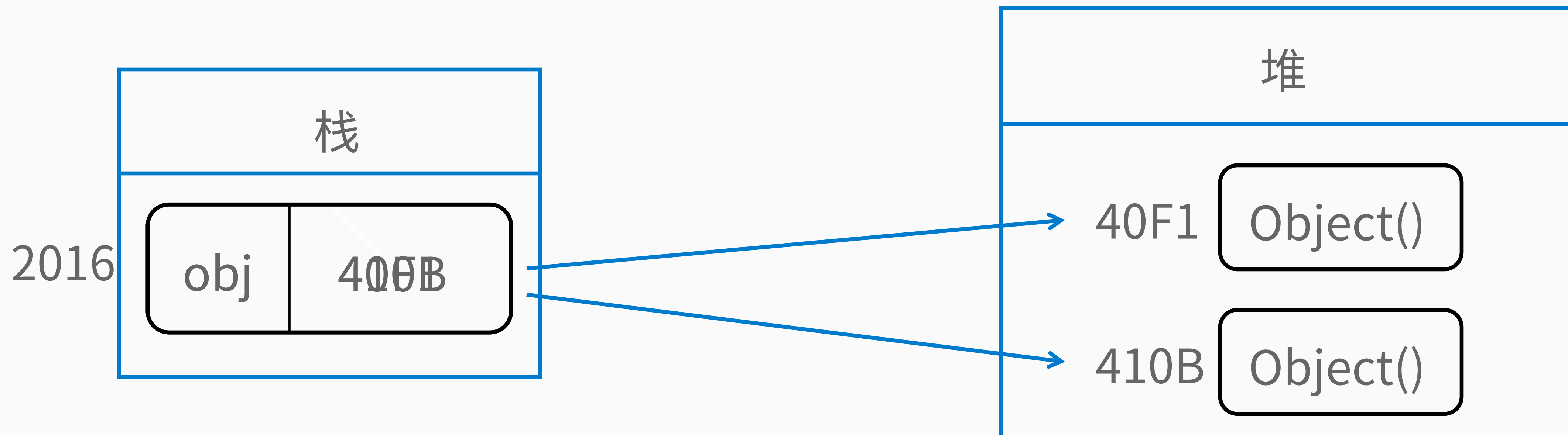
深入理解引用与引用传递

深入理解引用与引用传递

- 引用的本质
- 引用传递的本质
- 改变对象的值

深入理解引用与引用传递 — 引用的本质

- 对象存在于堆内存中
- 引用是变量，存在于栈内存中；也可能存在于堆内存中
- 引用的值，就是堆内存对象的起始地址
- 通常称为“引用指向对象”



深入理解引用与引用传递 — 引用的本质

- 引用是**变量**，变量的值是对象的起始地址
- 地址的值是**无符号整形**
- 引用本身也有地址

很像C语言里边的指针；所以，“Java中有指针”与“Java中没有指针，只有引用”的观点其实并不冲突。

注意：Java中的引用（指针）不能指向基本类型！

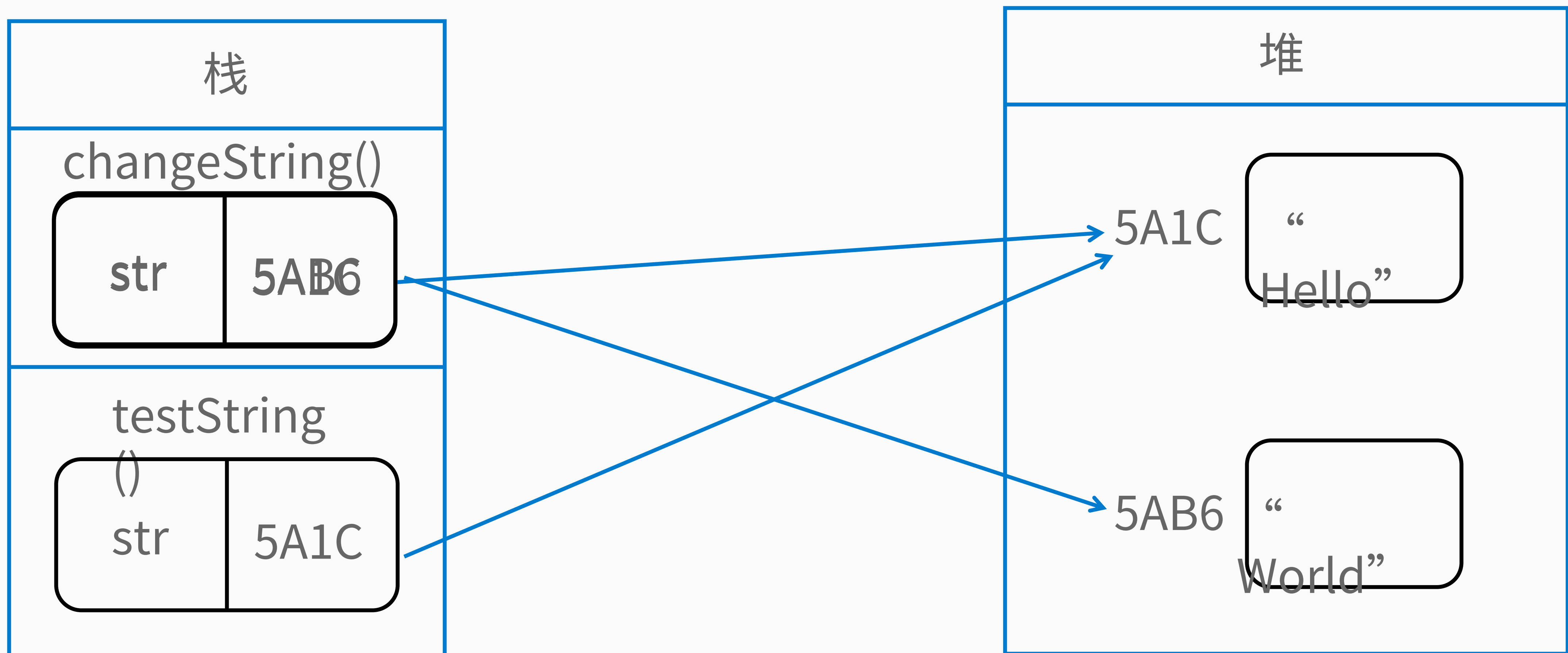
深入理解引用与引用传递 — 引用传递的本质

引用传递的本质，还是**传值**，具体步骤：

- 在栈中开辟形参引用
- 把实参引用的值传递给形参引用

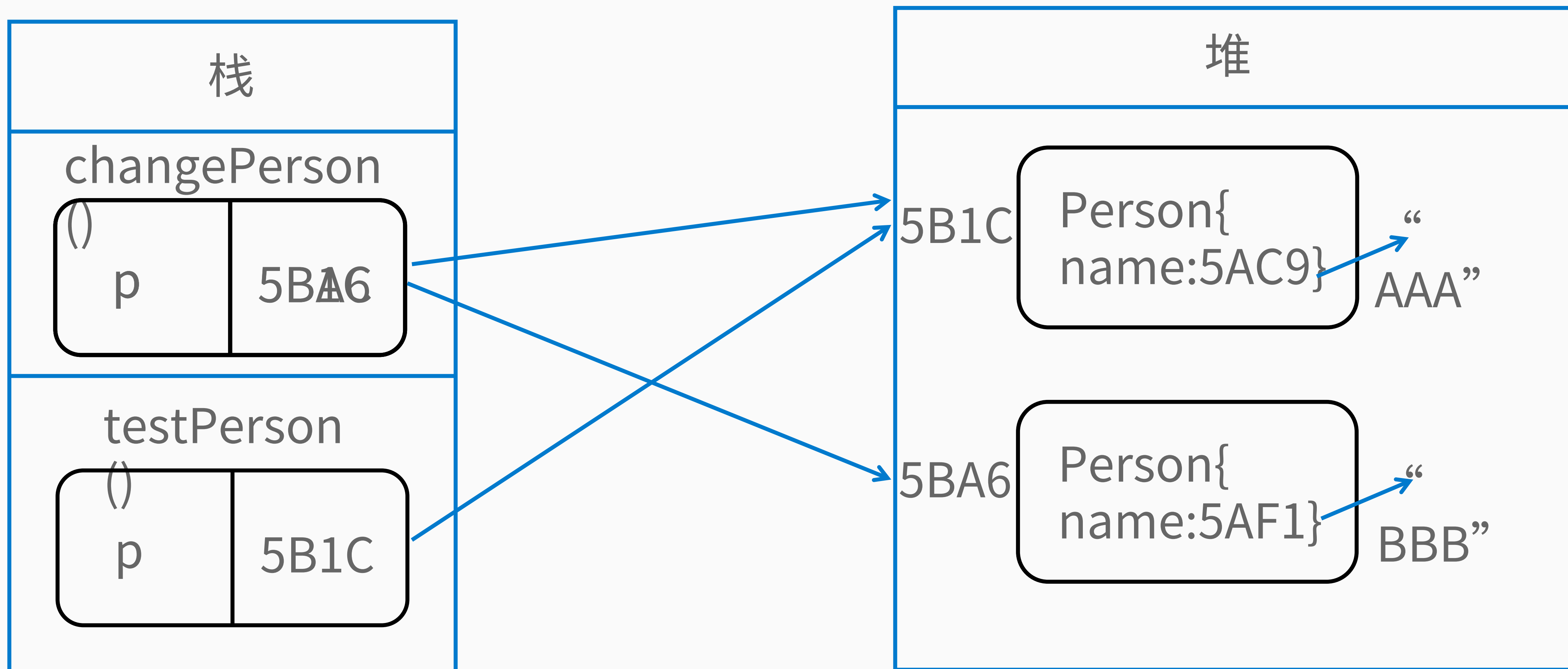
深入理解引用与引用传递 — 引用传递的本质

testString()的调用过程：



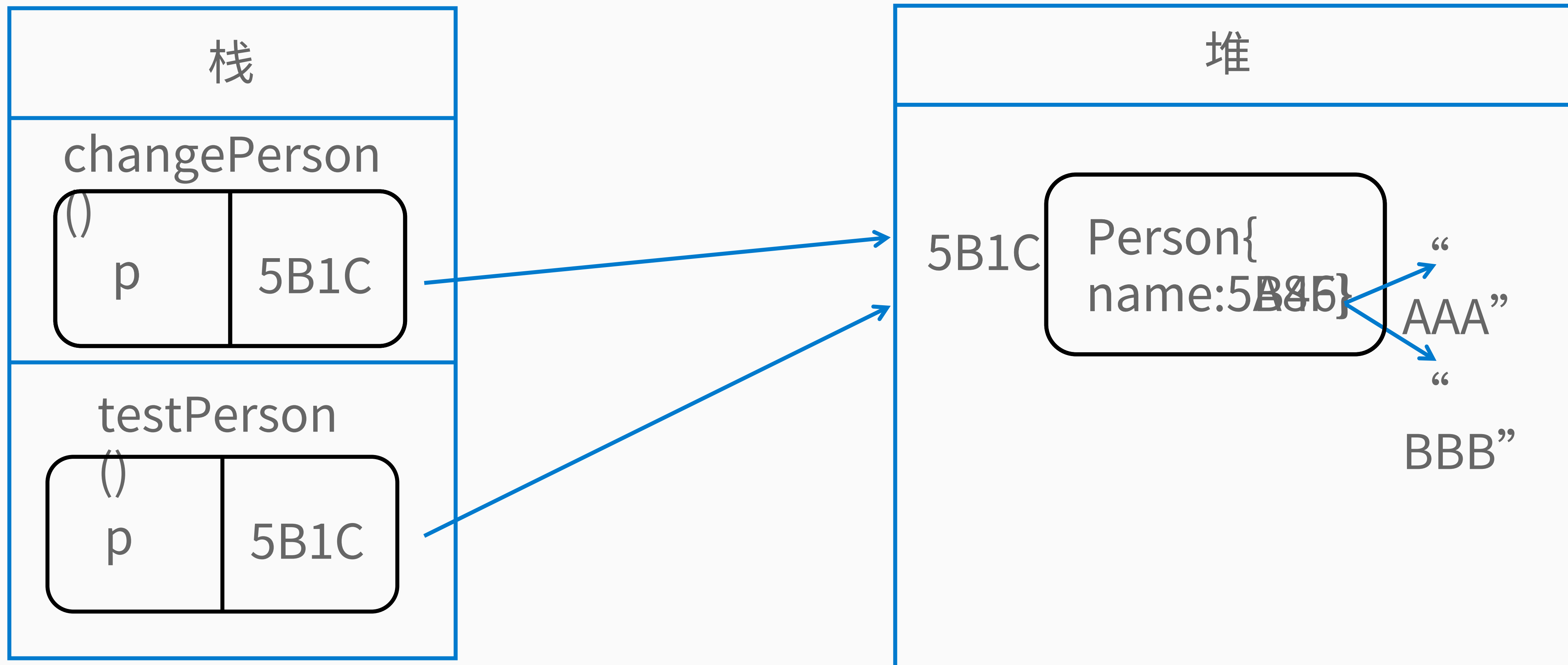
深入理解引用与引用传递 — 引用传递的本质

testPerson()的调用过程：



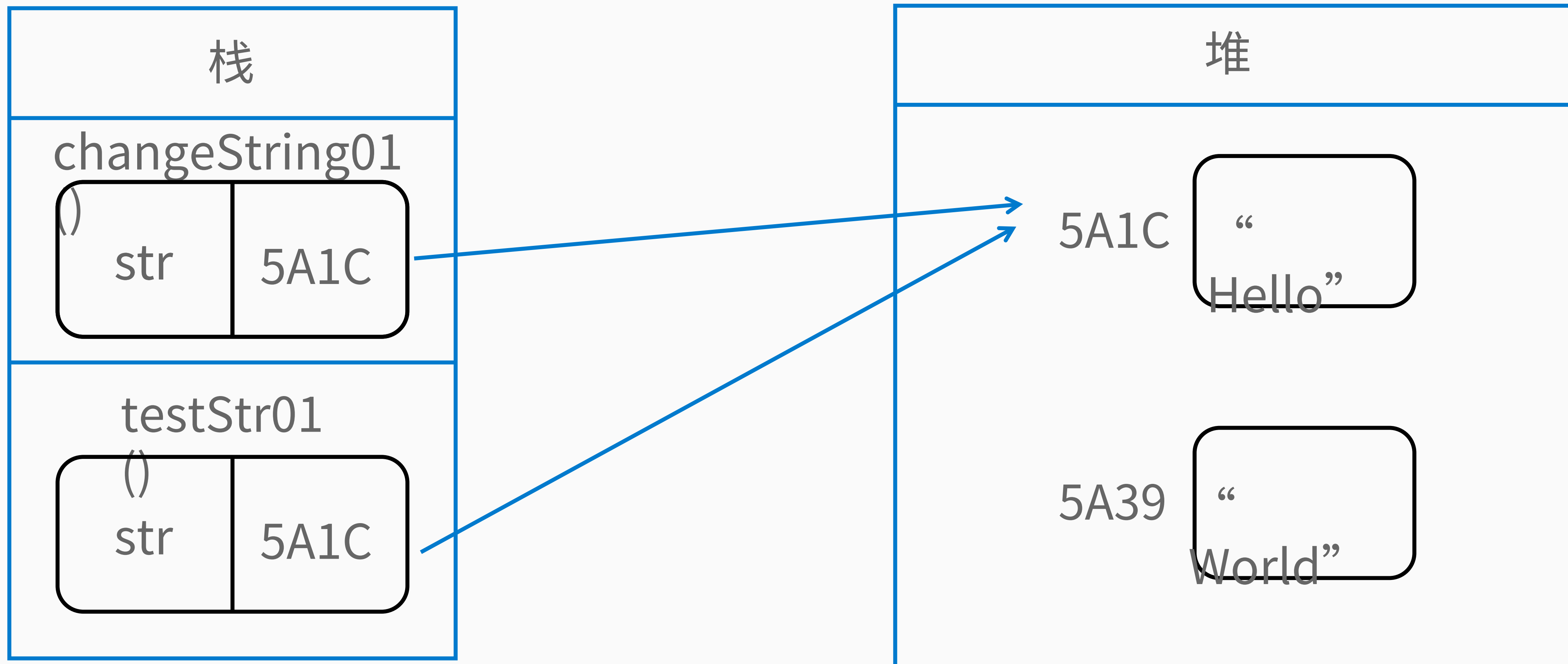
深入理解引用与引用传递 — 改变对象的值

testPerson()的调用过程：



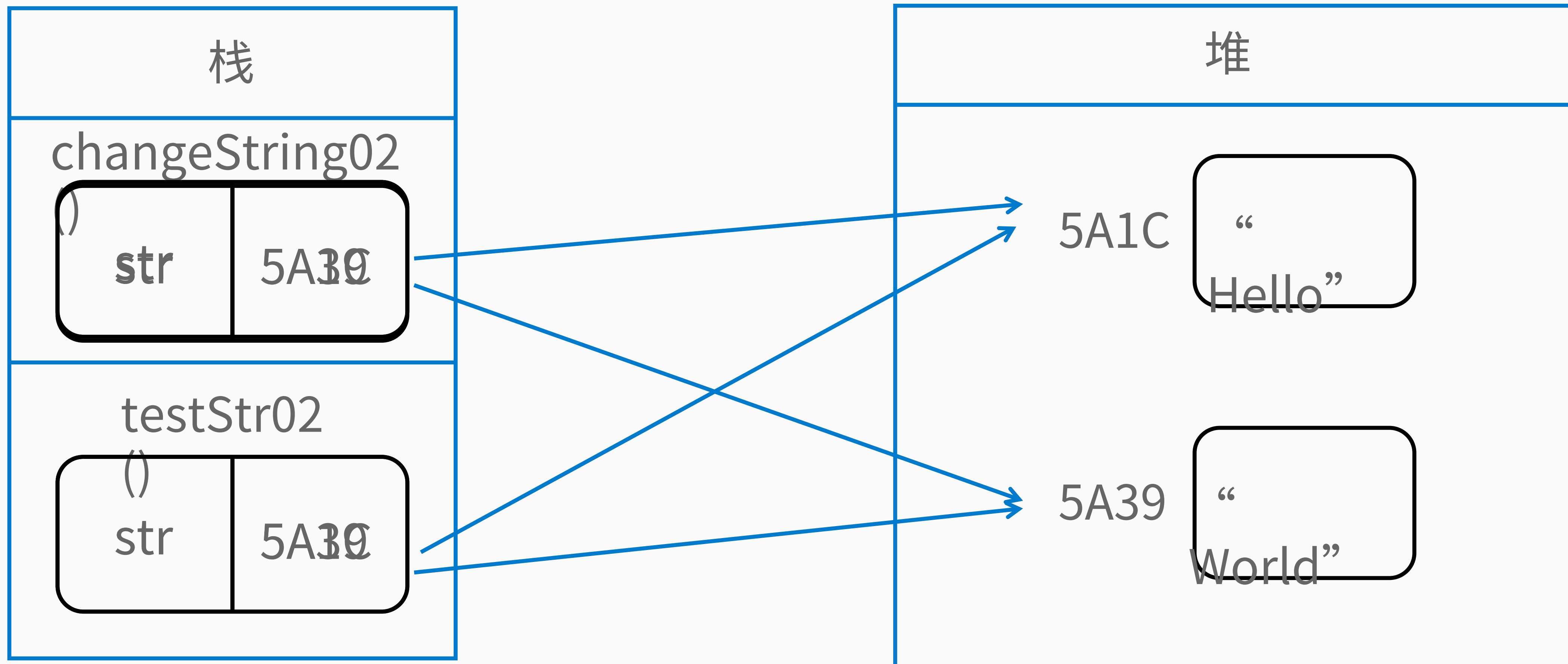
深入理解引用与引用传递 — 改变对象的值

testStr01()的调用过程:



深入理解引用与引用传递 — 改变对象的值

testStr02()的调用过程:



透彻分析String、StringBuilder和StringBuffer

透彻分析String、StringBuilder和StringBuffer

StringBuilder与String的性能对比

在名企面试官面前，如果仅仅回答“String不可变、StringBuilder可变、String不可被继承”等等**卖萌**的答案，只能等着被pass！

StringTest.java

当进行大量的字符串拼接操作时，StringBuilder的append()方法比String的“+”快50倍以上！

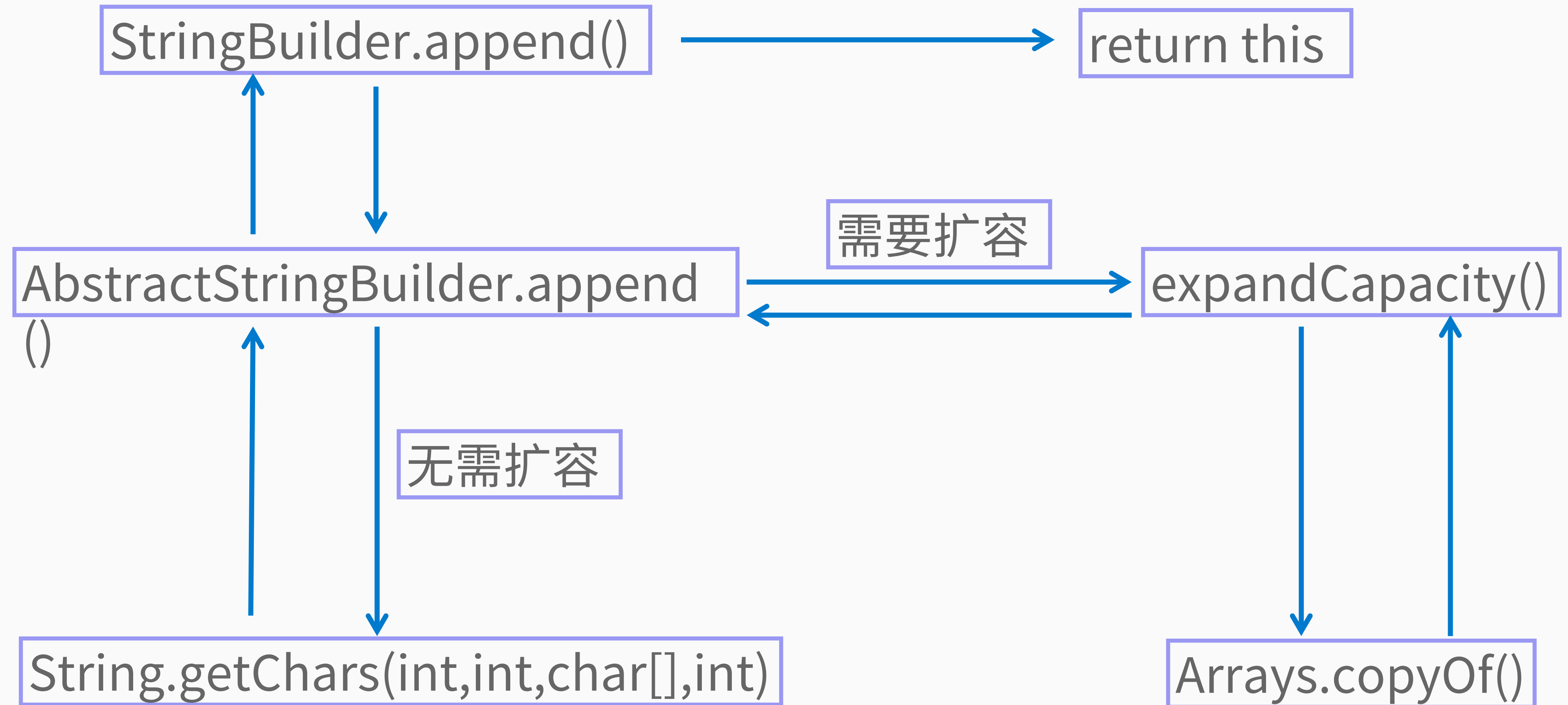
透彻分析String、StringBuilder和StringBuffer

StringBuilder关键源码剖析

以append(String str)为例，会涉及如下关键源码：

| 类名 | 方法名、属性名 | 作用 |
|-----------------------|---------------------------------|----------|
| StringBuilder | append(String) | 在末尾追加字符串 |
| AbstractStringBuilder | append(String) | 在末尾追加字符串 |
| AbstractStringBuilder | char value[] | 存储字符数组 |
| String | getChars(int, int, char[],int) | 复制字符数组 |
| AbstractStringBuilder | expandCapacity(int) | 扩充容量 |
| Arrays | copyOf(char[], int) | 复制字符数组 |

透彻分析String、StringBuilder和StringBuffer

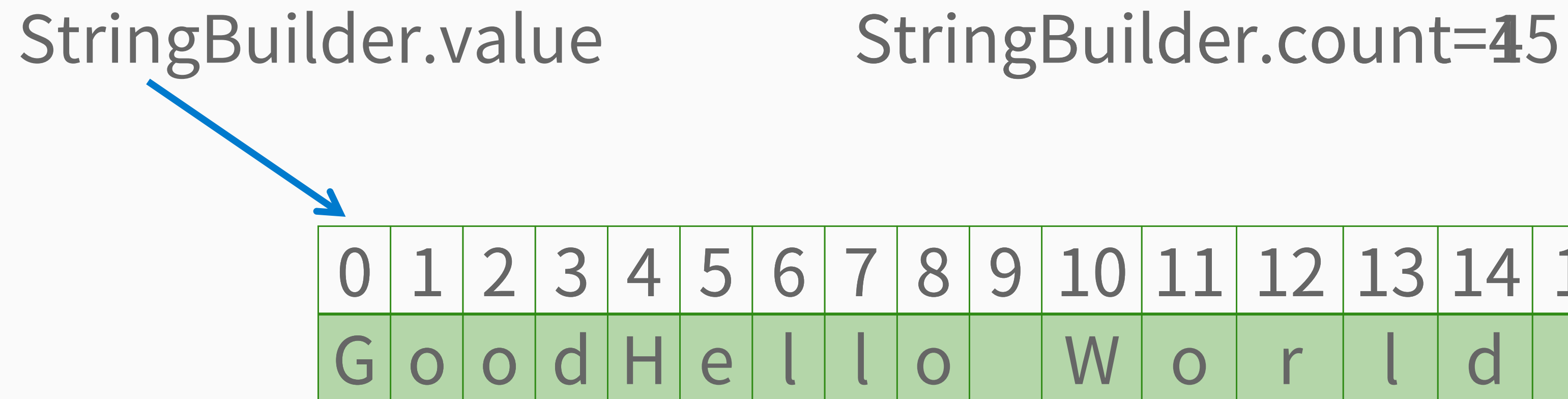
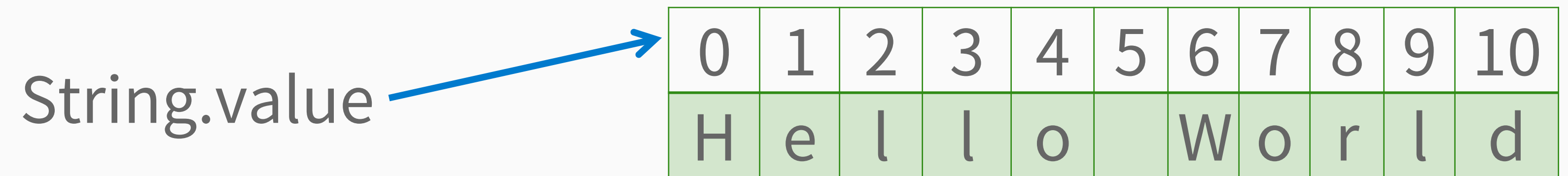


透彻分析String、StringBuilder和StringBuffer

附加以下“面向对象”的回答，会更加出彩：

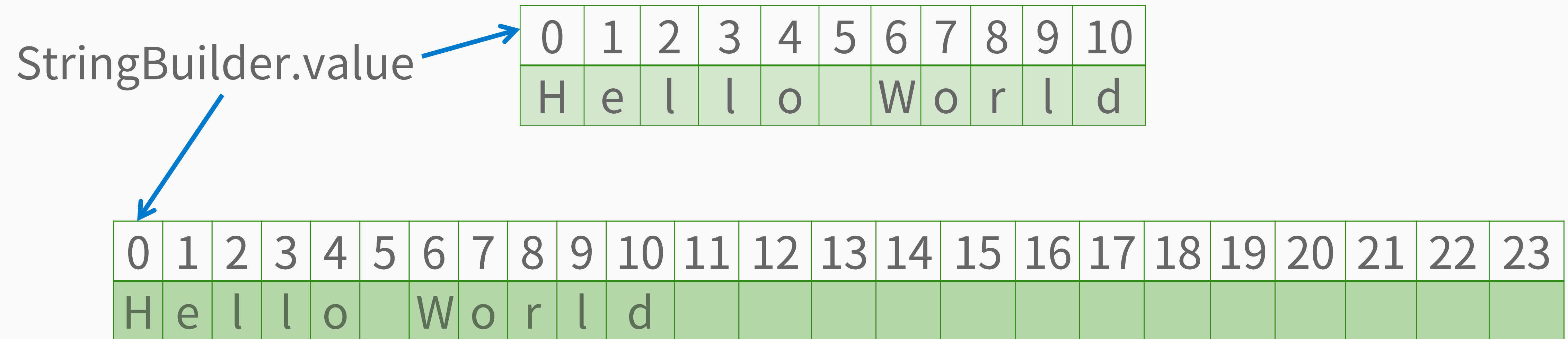
- StringBuilder是**抽象类**AbstractStringBuilder的一个具体实现
- StringBuilder与AbstractStringBuilder**重载**了不同的append()方法
- 所有的append()方法都会返回this，这样就实现了**链式编程**

透彻分析String、StringBuilder和StringBuffer



将数组的容量扩大至原来的 $2n+2$;

其中，`expandCapacity()`又调用了Arrays的`copyOf()`方法，目的是把原来数组的元素拷贝至新的数组。



透彻分析String、StringBuilder和StringBuffer

假设执行了65535次append(“H”), 即: $n=65535$; 那么, 一共进行了多少次新数组内存的开辟, 以及旧数组内存的释放?

为了方便, 进行一些简化:

- 数组初始容量为1
- 每次扩容, 容量扩大至原来的2倍

$1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 32 \rightarrow 64 \rightarrow \cdots \rightarrow 65536$

$65536=2^{16}$, 故而, 进行了 $\log_2 N$ 次开辟和释放。

透彻分析String、StringBuilder和StringBuffer

同样的道理，n=65535，复制了多少个字符？

首先，65535次复制无法避免。

其次，计算数组扩容所复制字符的个数。

1、2、4、8、16 … 32768

根据等比数列求和公式：

$$S_n = \frac{a_1(1 - q^n)}{1 - q}$$

$a_1=1, q=2, n=16$ 代入可得 $s_n=65535$

所以，一共复制2n个字符。

透彻分析String、StringBuilder和StringBuffer

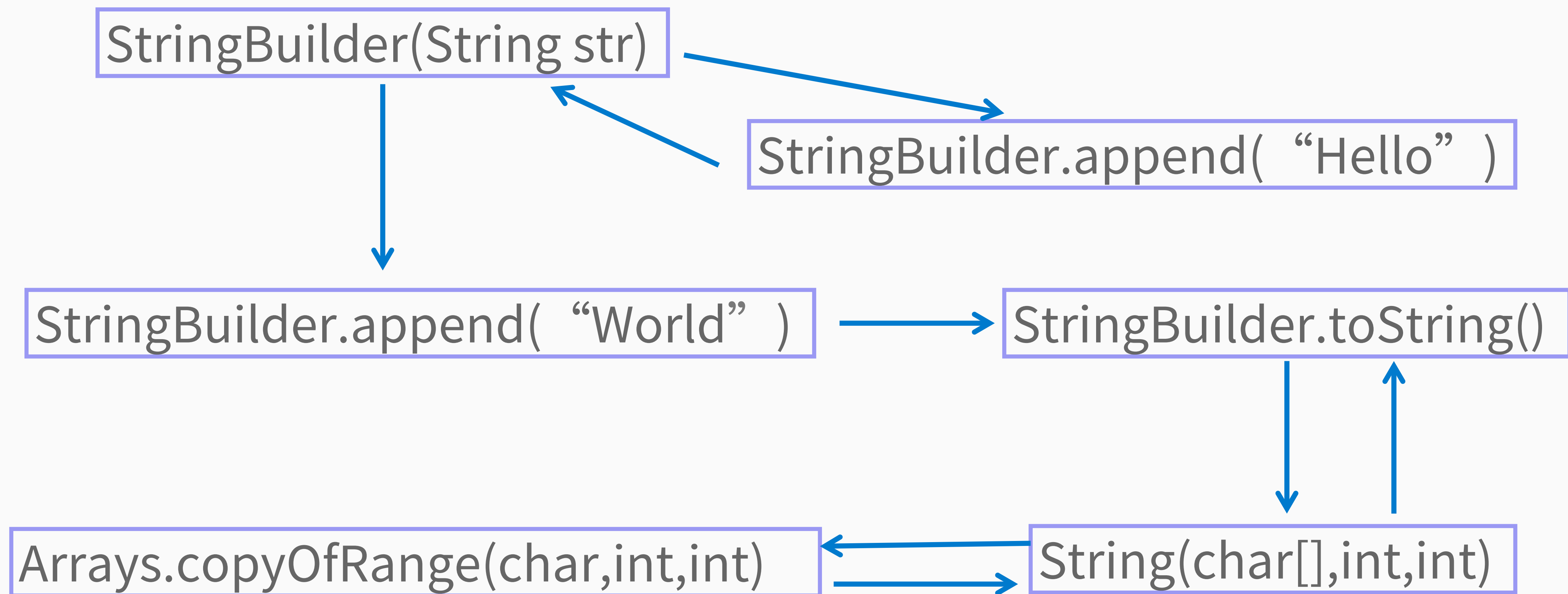
String关键源码剖析

String的“+”，会涉及如下关键源码：

| 类名 | 方法名、属性名 | 作用 |
|---------------|-----------------------------|------------------------|
| StringBuilder | StringBuilder(String) | StringBuilder的构造函数 |
| StringBuilder | append(String) | 在末尾追加字符串 |
| StringBuilder | toString() | StringBuilder转换为String |
| String | String(char[], int,int) | String的构造函数 |
| Arrays | copyOfRange(char[],int,int) | 复制字符数组 |

透彻分析String、StringBuilder和StringBuffer

比如str= “Hello” ，那么str+= “World” 的执行步骤:



100

概括一下整个过程：

String.value

[illegible][illegible]

透彻分析String、StringBuilder和StringBuffer

同StringBuilder的append()，假设执行了65535次“+”，即： $n=65535$ ；那么，一共进行了多少次新对象、新数组的开辟，以及旧对象、旧数组的释放？

- 每次“+”，要new StringBuilder()，一共n次
- 每次“+”，要new char[str.length()+1]，一共n次

故而，进行了 $2n$ 次的开辟和释放。

透彻分析String、StringBuilder和StringBuffer

同样的道理，n=65535，复制了多少个字符？

1、2、3、4、5、6...65535

根据等差数列求和公式：

$$S_n = \frac{n(a_1 + a_n)}{2}$$

$a_1=1, a_n=65535, n=65535$ 代入可得：

$$S_n = \frac{65535 * 65536}{2}$$

透彻分析String、StringBuilder和StringBuffer

| 方法 | 操作次数 | 次数 |
|----------------------|---------|---------------|
| StringBuilder的append | 开辟、释放内存 | $O(\log_2 N)$ |
| String的+ | 开辟、释放内存 | $O(N)$ |
| StringBuilder的append | 字符复制 | $O(N)$ |
| String的+ | 字符复制 | $O(N^2)$ |

孰优孰劣，高下立判！

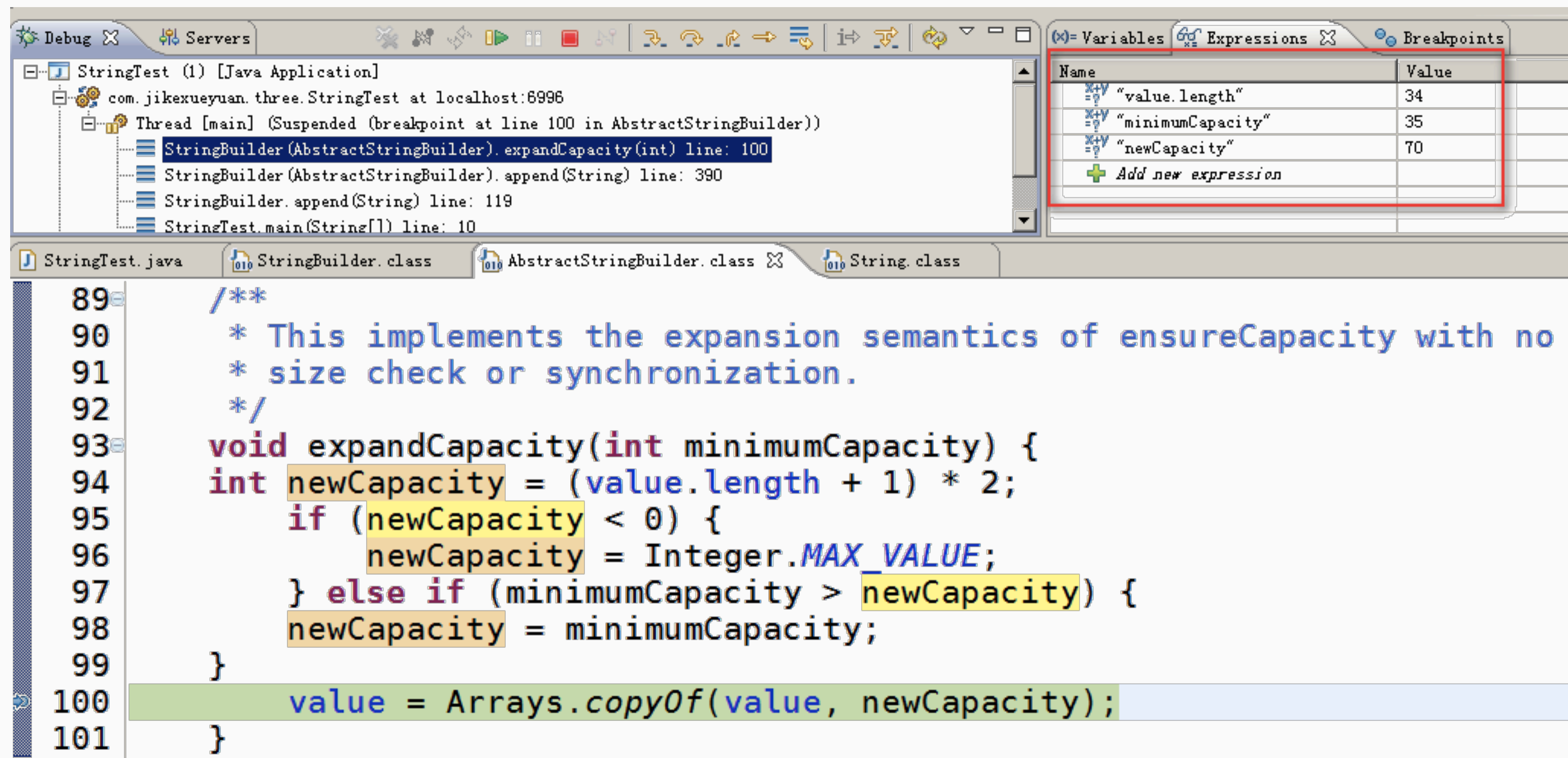
StringBuffer的方法只是比StringBuilder多了同步关键字，两者功能一样。

透彻分析String、StringBuilder和StringBuffer

StringBuilder关键源码调试

注意：进入main()函数的for循环之后再源码里边下断点！

重点观察expandCapacity()方法的变量变化情况：



Debug console shows the execution flow:

- StringTest (1) [Java Application]
- com.jikexueyuan.three.StringTest at localhost:8996
- Thread [main] (Suspended (breakpoint at line 100 in AbstractStringBuilder))
- StringBuilder (AbstractStringBuilder). expandCapacity(int) line: 100
- StringBuilder (AbstractStringBuilder). append(String) line: 390
- StringBuilder.append(String) line: 119
- StringTest.main(String[]) line: 10

Source code (StringBuilder.expandCapacity):

```
89 /**
90  * This implements the expansion semantics of ensureCapacity with no
91  * size check or synchronization.
92  */
93 void expandCapacity(int minimumCapacity) {
94     int newCapacity = (value.length + 1) * 2;
95     if (newCapacity < 0) {
96         newCapacity = Integer.MAX_VALUE;
97     } else if (minimumCapacity > newCapacity) {
98         newCapacity = minimumCapacity;
99     }
100     value = Arrays.copyOf(value, newCapacity);
101 }
```

Variables table:

| Name | Value |
|----------------------|-------|
| "value.length" | 34 |
| "minimumCapacity" | 35 |
| "newCapacity" | 70 |
| + Add new expression | |

透彻分析String、StringBuilder和StringBuffer

String关键源码调试

重点观察Arrays.copyOfRange()方法的变量变化情况：

The screenshot displays a Java IDE in a debug state. The top pane shows the call stack for 'StringTest (1) [Java Application]'. The current frame is 'Thread [main] (Suspended (breakpoint at line 3212 in Arrays))', with the call to 'Arrays.copyOfRange(char[], int, int) line: 3212' selected. The bottom pane shows the source code of 'Arrays.copyOfRange' in 'Arrays.class'. The code is as follows:

```
3204 */
3205 public static char[] copyOfRange(char[] original, int from, int to) {
3206     int newLength = to - from;
3207     if (newLength < 0)
3208         throw new IllegalArgumentException(from + " > " + to);
3209     char[] copy = new char[newLength];
3210     System.arraycopy(original, from, copy, 0,
3211                     Math.min(original.length - from, newLength));
3212     return copy;
3213 }
3214
3215 /**
```

The right pane shows the 'Variables' tab with the following data:

| Name | Value |
|----------------------|-------|
| "to-from" | 25 |
| "newLength" | 25 |
| + Add new expression | |

透彻分析String、StringBuilder和StringBuffer

面试的时候：

- 写代码展示效率的差异
- 借助control键剖析源代码的调用过程
- 分析时间复杂度、空间复杂度
- 调试验证

不卑不亢，秒杀Offer!

透彻分析String、StringBuilder和StringBuffer

其它需要了解的源码

| 类名 | 对应数据结构 | 难度系数 |
|---------------|--------|------|
| ArrayList | 顺序表 | 1 |
| LinkedList | 双向链表 | 2 |
| Stack | 栈 | 1 |
| Queue | 队列 | 1 |
| HashMap | 哈希表 | 3 |
| LinkedHashMap | 哈希表+链表 | 4 |
| PriorityQueue | 优先队列 | 4 |
| TreeMap | 红黑树 | 5 |

C语言风格字符串

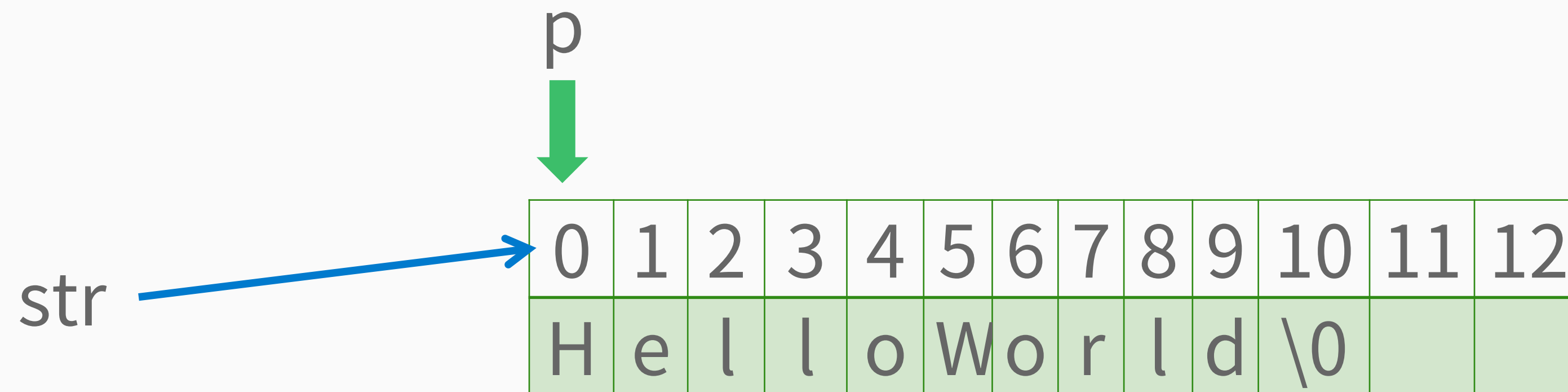
C语言风格字符串

- 字符串的长度
- 字符串的拷贝
- 字符串的连接
- 字符串的比较
- 字符串的大小写转换

C语言风格字符串 — 字符串的长度

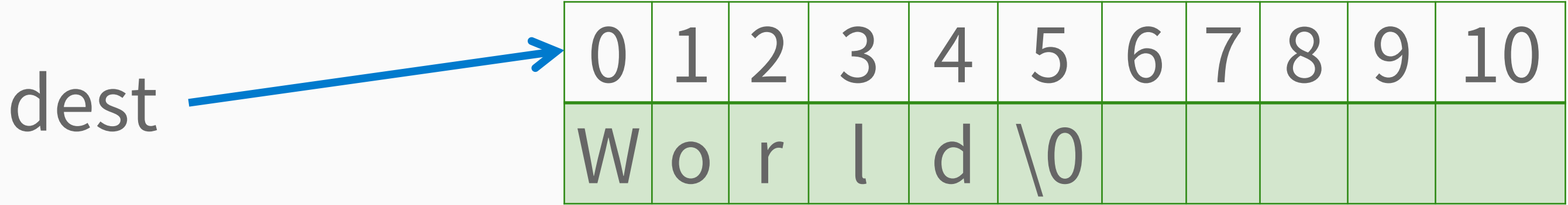
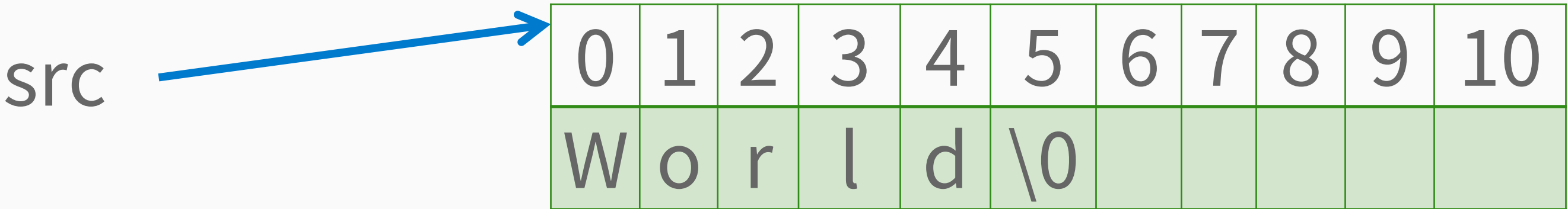
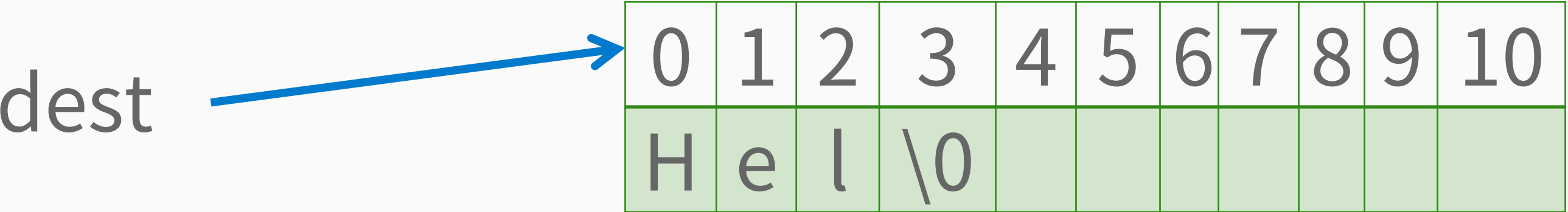
C语言风格的字符串末尾有一个 ‘\0’ 。

```
char* str= "HelloWorld" ;
```



C语言风格字符串 — 字符串的拷贝

目标字符串dest比源字符串src短



C语言风格字符串 — 字符串的拷贝

目标字符串dest比源字符串src长

dest

| | | | | | | | | | | |
|---|---|---|---|---|----|---|---|---|---|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| W | o | r | l | d | \0 | | | | | |

src

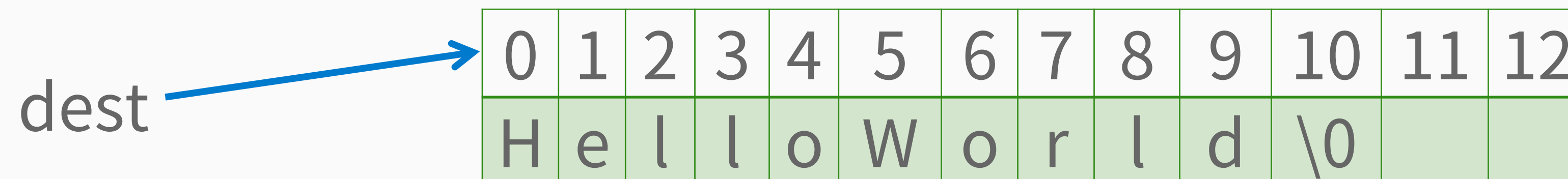
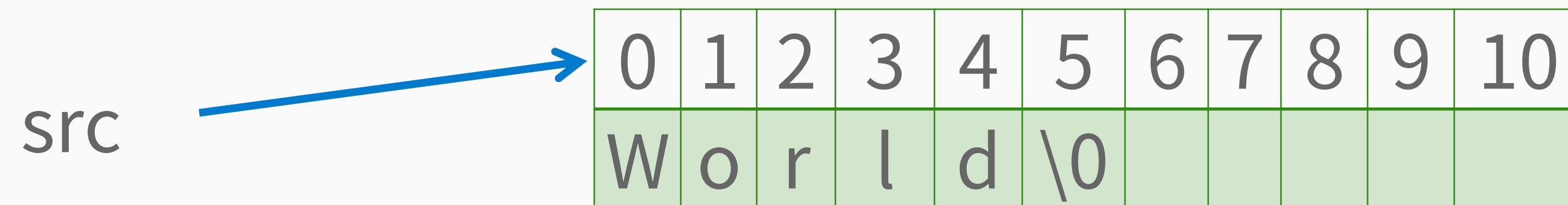
| | | | | | | | | | | |
|---|---|---|----|---|---|---|---|---|---|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| H | e | l | \0 | | | | | | | |

dest

| | | | | | | | | | | |
|---|---|---|----|---|---|---|---|---|---|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| H | e | l | \0 | | | | | | | |

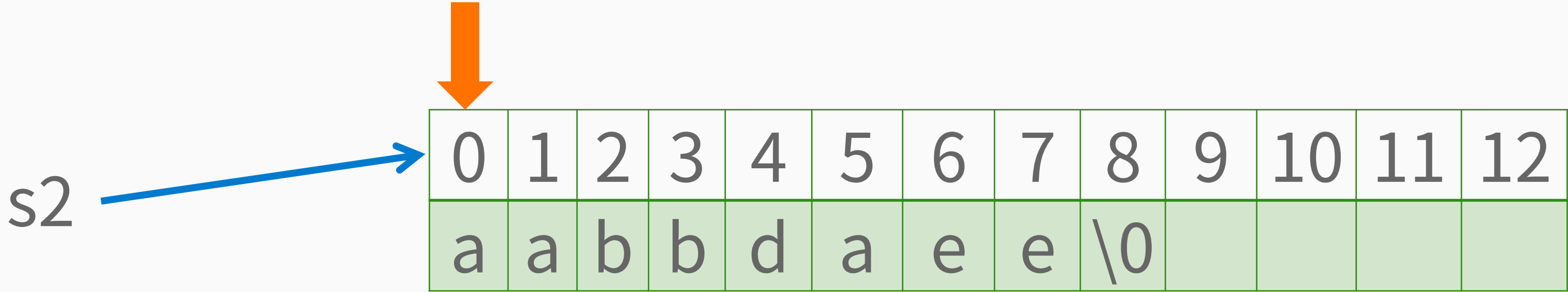
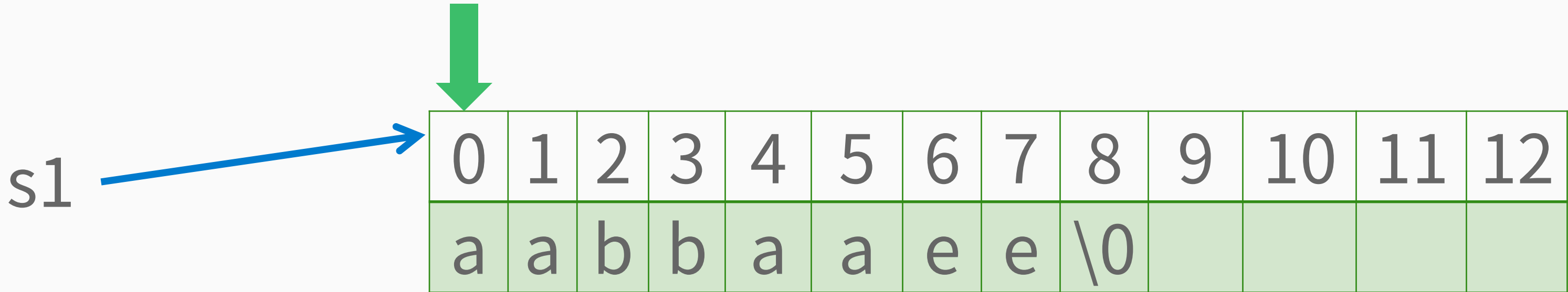
C语言风格字符串 — 字符串的连接

- 目标串的尾部，有足够的空间容纳源字符串
- 求目标串的**长度**
- 将源字符串**复制**到目标串的尾部



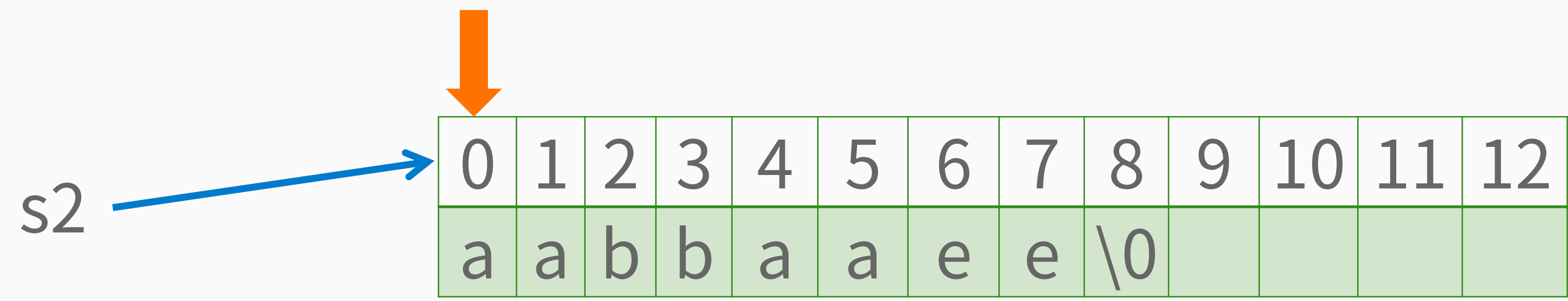
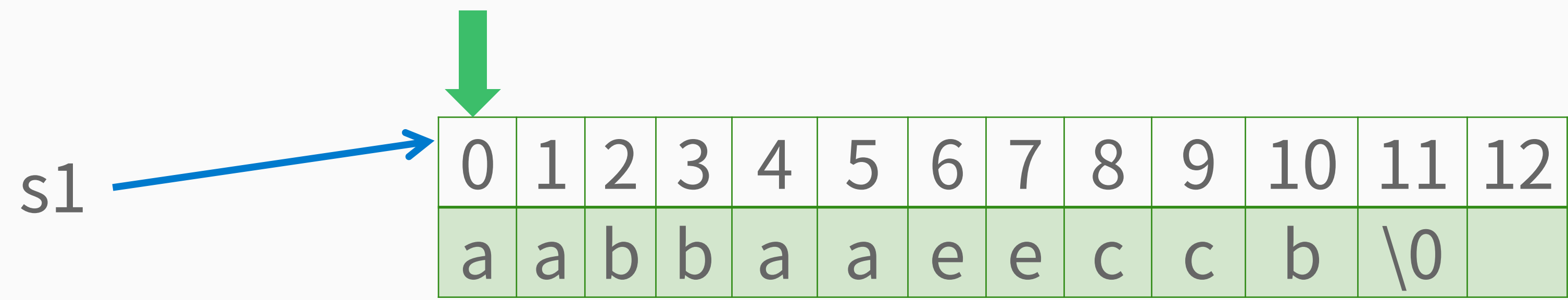
C语言风格字符串 — 字符串的比较

相等返回0，小于返回-1，大于返回1



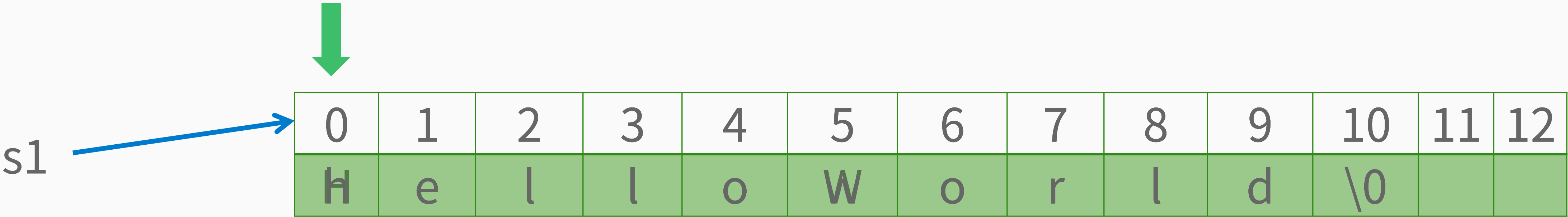
C语言风格字符串 — 字符串的比较

长度不相等，同理



C语言风格字符串 — 字符串的大小写转换

以“转换为小写”为例



最后一个单词的长度

最后一个单词的长度

- 问题描述
- 思路分析
- Java版的代码
- C语言版的代码

最后一个单词的长度 — 问题描述

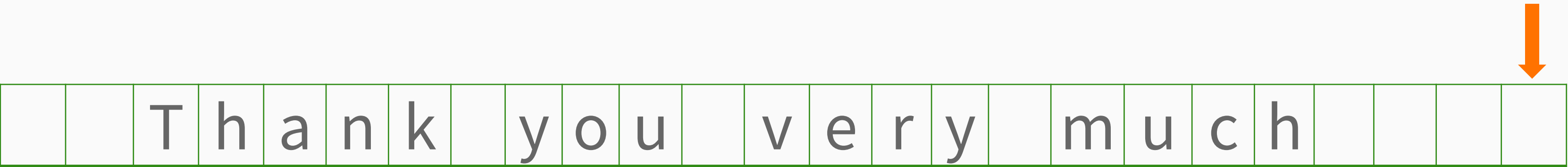
leetcode 58: Length of Last Word

给定句子，只由字母、空格组成，返回最后一个单词的长度；
所谓**单词**，就是不含空格的字符序列。

“ Thank you very much ” 最后一个单词的长度为4。

注意：禁用split

最后一个单词的长度 — 思路分析



count=0

最后一个单词的长度 — Java版的代码

| Key | Value |
|-------|-------------------------|
| 类名 | _058LengthOfLastWord |
| 方法名 | lengthOfLastWord |
| 测试输入 | " Thank you very much " |
| 测试输出 | 4 |
| 时间复杂度 | O(N) |
| 空间复杂度 | O(1) |

最后一个单词的长度 — C语言版的代码

| Key | Value |
|-------|-------------------------|
| 文件名 | _058LengthOfLastWord.c |
| 方法名 | lengthOfLastWord |
| 测试输入 | " Thank you very much " |
| 测试输出 | 4 |
| 时间复杂度 | O(N) |
| 空间复杂度 | O(1) |

名企数据结构面试题之字符串（上）

本套课程中我们学习了名企数据结构面试题之字符串（上）。你应当掌握了以下知识：

- 揭开JDK源变量的神秘面纱
- 深入理解引用与引用传递
- 透彻分析String、StringBuilder和StringBuffer
- C语言风格字符串
- 最后一个单词的长度

你可以对JDK源代码进行调试，还可以使用leetcode来测试代码的正确性；如果想进一步提高，你可以继续在极客学院学习**名企数据结构面试题之字符串（下）**课程。

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台

