河南中医药大学智能医学工程专业《计算机程序设计》课程

第07章：数组与字符串

黄子杰

河南中医药大学信息技术学院（智能医疗行业学院）与河南方和信息科技有限公司 联合建设
河南中医药大学信息技术学院互联网技术教学团队
https://webdev.hactcm.edu.cn
2024/9/13

本章概要

- 数组：

相同类型的元素的集合。数组的地址空间是连续的，数组变量存放的是数组首元素的地址。数组便于查找元素，不利于元素的增删。只能存放小规模的数据，因为内存中很难找到一大块连续的空间来存放。

- 字符串：

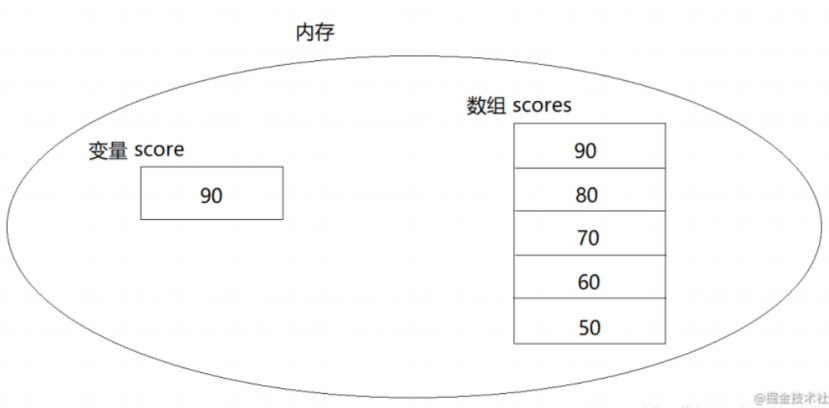
Java中的字符串是引用数据类型，存储在方法区的常量池当中，一旦被创建就不能被修改。事实上，Java利用了char数组来完成字符串的创建，且该数组被final修饰。

7.1 一维数组

Java语言的数组分为一维数组、二维数组以及多维数组。

数组中每个元素都只带一个下标，这样的数组称为一维数组。

注：使用Java数组前，应声明数组并为该数组分配存储空间。



7.1.1 为什么要使用数组

- 为什么需要数组?

使用数组可以:

（1）大规模存储临时需要处理的数据;

（2）采用循环语句简化对大量变量的操作。

- 概述:

数组是一种数据结构，用于存储同一类型的数据集合。使用数组可以有效地管理和操作这些数据。

- 详细说明:

数据集中存储：数组允许将多个数据值存储在一个变量中，简化数据的管理。

随机访问：数组可以通过索引快速访问元素，时间复杂度为O(1)。

节省内存：比起创建多个单独的变量，数组可以节省内存和提高程序的效率。

7.1.1 为什么要使用数组

- 数组有何特点?

（1）在一个数组中，数组元素的类型是唯一的，即一个数组里只能存储一种数据类型的数据，而不能存储多种数据类型的数据；

（2）对数组初始化完成后，为数组在内存中分配的存储空间已经确定，故不能改变数组的长度；

（3）数组中不但能存储基本类型的数据，还能存储引用类型的数据；

（4）数组本身是引用数据类型。

7.1.1 为什么要使用数组

- 生活举例:

数组：就像一个文件夹，用于存储多个文件。我们可以通过文件夹的索引（名称）快速找到某个文件。

- 面试题示例:

问：为什么使用数组？数组的优点是什么？

答：使用数组可以集中存储相同类型的数据，通过索引快速访问元素，并节省内存空间。

7.1.2 什么是数组

- 概述：

数组是一组具有相同数据类型的元素的集合。每个元素通过索引进行访问。

- 详细说明：

数组声明：定义数组时需要指定数组的类型和名称。
数组初始化：可以在声明时初始化数组，或先声明后赋值。
索引：数组的索引从0开始，最后一个元素的索引是数组长度-1。

1、静态初始化

通过直接指定初值来初始化数组，

例如：

```
int list[] = {5,3,4,6,1,2};

int[] list = {5,3,4,6,1,2};
```

2、利用new关键字初始化

- 利用关键字new初始化数组，不仅能为数组分配需要的存储空间，还能按照Java的默认初始化原则为数组元素赋值。

有两种方法：

（1）先声明数组，然后初始化。

例如：

```
int list[] = null;//声明list数组
list = new int[6];//为list数组分配存储6个整数元素的存储空间
```

（2）声明数组并同时初始化。

例如：

```
int list[] = new int[6];//声明一维数组的同时分配6个整数元素的存储空间
```


7.1.2 什么是数组

例如：我要存储5名学生成绩。

- // 声明数组：在内存空间中会开辟一串连续的空间

```
double[] scores = new double[5];
```

- // 给数组赋值：将数组元素存储到指定的数组空间中

```
scores[0] = 80;
scores[1] = 90;
```

- // 取出数组的值

```
scores[2]
```

7.1.2 什么是数组

- 生活举例：

数组：就像一个装有多个小盒子的抽屉，每个小盒子有一个编号，可以用来存放不同的物品。

- 面试题示例：

问：什么是数组？如何声明和初始化一个数组？

答：数组是一组相同类型的数据集合。声明数组时指定类型和名称，初始化时可以指定长度或直接赋值。

- 代码示例：

eg7.1.2.txt

7.1.3 如何使用数组

- 概述：

数组使用时需要声明、初始化、赋值和访问。常见操作包括遍历、查找、插入和删除元素。

- 详细说明：

声明和初始化：声明数组后，可以通过索引访问和修改元素。

遍历数组：使用循环遍历数组中的每个元素。

数组长度：使用array.length获取数组的长度。

7.1.3 如何使用数组

- 数据的操作，逃脱不了及格本质：CRUD （增删改查）。

增加数据：数组名[下标] = 元素值;
修改数据：数组名[下标] = 新元素值;
删除数据：数组名[下标] = 数组对应元素类型的默认值;
查询数据：
查询单个的：数组名[下标];
查询所有的：利用循环将数组中的每一个空间的元素取出来（遍历）

7.1.3 如何使用数组

- 生活举例：

数组操作：就像在一排抽屉中存取物品，我们可以依次检查每个抽屉的内容。

- 面试题示例：

问：如何遍历一个数组?
答：使用for循环遍历数组，通过索引访问每个元素。

- 代码示例：

eg7.1.3.txt

7.1.4 数组常见错误

- 概述：

数组常见错误包括数组下标越界、空指针异常、数组类型不匹配等。

详细说明：

1、数组下标越界：尝试访问数组中不存在的索引位置，会抛出ArrayIndexOutOfBoundsException。

2、空指针异常：如果数组声明了但未初始化，尝试访问元素会抛出NullPointerException。

3、数组类型不匹配：将不匹配的数据类型赋值给数组元素，编译时会报错。

数组下标越界

数组越界异常也是常见的Java致命错误之一。

当使用一个索引超过数组长度或小于0的值时，将会抛出数组越界异常。以下代码演示了一个引起数组越界异常的例子：

```
int[] arr = {1, 2, 3};

int element = arr[5]; // ArrayIndexOutOfBoundsException
```

为避免数组越界异常，开发人员应该在访问数组元素之前，始终确保索引值在合法范围内。

空指针异常

空指针异常是Java开发中常见的致命错误之一。
在代码中，如果没有对空引用进行合适的判空处理，就有可能导致空指针异常的发生。

```
int[] arr1 = new int[]{1,2,3};
arr1 = null;
System.out.println(arr1[0]);// NullPointerException
```


7.1.4 数组常见错误

- 生活举例：

数组错误：就像试图打开一个不存在的抽屉，或者试图将不适合的物品放入某个抽屉。

- 面试题示例：

问：如何处理数组下标越界的问题？
答：确保索引在合法范围内，通常使用循环遍历时需要检查索引是否超出数组长度。

7.2 常用算法

概述：

常用的数组算法包括计算平均值、最大值和最小值。

- 详细说明：

平均值：数组所有元素之和除以元素个数。
最大值：数组中所有元素的最大值。
最小值：数组中所有元素的最小值。

7.2.1 平均值, 最大值, 最小值

- 生活举例：

平均值、最大值、最小值：就像计算一个班级所有学生的平均成绩，找出最高分和最低分。

- 面试题示例：

问：如何计算数组的平均值、最大值和最小值？
答：通过遍历数组累加元素和找出最大最小值，然后计算平均值。

7.2.1 平均值, 最大值, 最小值

- 代码示例:

eg7.2.1.txt

7.2.2 数组排序

- 概述:

排序算法用于将数组中的元素按某种顺序排列。常见的排序算法有冒泡排序、选择排序和快速排序等。

- 详细说明:

冒泡排序：重复交换相邻元素，直到所有元素有序。

选择排序：每次选择最小（或最大）的元素放到已排序部分的末尾。

快速排序：通过分治法选择基准元素，将数组分成左右两个部分，递归排序。

冒泡排序

- 概述：

冒泡排序通过重复地遍历待排序的数列，比较相邻的两个元素，并交换位置来完成排序。每次遍历都会将未排序部分中的最大（或最小）元素移动到数列的一端，就像气泡一样逐渐浮出水面。

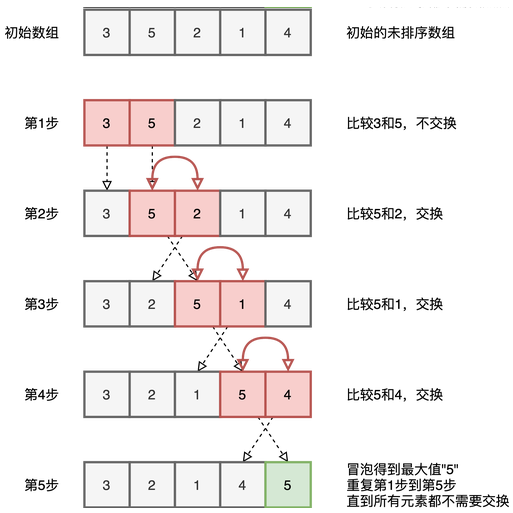
- 算法步骤：

从数列的开头开始，依次比较相邻的两个元素。

如果前一个元素比后一个元素大（或小），则交换它们的位置。

对每一对相邻元素重复上述步骤，直至整个数列有序。

冒泡排序



- 代码示例：

eg冒泡排序.txt

选择排序

- 概述

选择排序是一种简单的排序算法，它的基本思想是每次从待排序的元素中选出最小（或最大）的一个元素，存放在序列的起始位置，直到全部待排序的元素排完。

- 算法步骤

初始状态：无序区为整个数组。

从无序区选择最小（或最大）的元素：将其与无序区的第一个元素交换位置。

无序区减1：有序区加1。

重复上述步骤：直到所有元素均排序完成。

选择排序

String 类

原数组	4	6	8	7	9	2	10	1
第一次排序：	1	6	8	7	9	2	10	4
第二次排序：	1	2	8	7	9	6	10	4
第三次排序：	1	2	4	7	9	6	10	8
第四次排序：	1	2	4	6	9	7	10	8
第五次排序：	1	2	4	6	7	9	10	8
第六次排序：	1	2	4	6	7	8	10	9
第七次排序：	1	2	4	6	7	8	9	10

代码示例：

eg选择排序.txt

插入排序

- 概述：

通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。

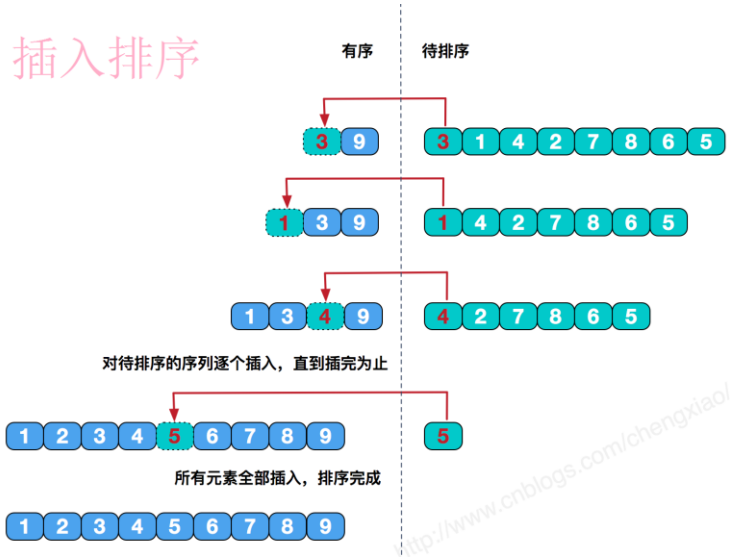
- 算法步骤：

初始状态：从第二个元素开始，将每个元素插入到前面已经排序的子数组中。

从后向前扫描：对已排序序列，从后向前扫描，找到合适的位置插入当前元素。

重复上述步骤：直到所有元素均排序完成。

插入排序



代码示例：

eg插入排序.txt

使用Java8 Stream API排序

- 在Java 8中，可以使用Stream API来对集合进行排序。以下是一个使用Comparator和Stream的例子，演示了如何对List进行排序：

代码示例:

eg7Stream.txt

7.2.3 数组复制

- 概述：

数组复制用于将一个数组的内容复制到另一个数组中。可以使用循环或系统提供的工具方法来实现。

- 详细说明：

手动复制：使用循环逐个复制元素。

系统方法：使用System.arraycopy或Arrays.copyOf方法进行数组复制。

代码示例（使用System.arraycopy）：

7.2.3 数组复制

- 生活举例：

数组复制：就像从一份文档中复印一份副本，确保两个文档内容完全相同。

- 面试题示例：

问：如何复制一个数组？

答：可以使用循环逐个复制，或使用System.arraycopy和Arrays.copyOf等系统方法。

7.2.3 数组复制

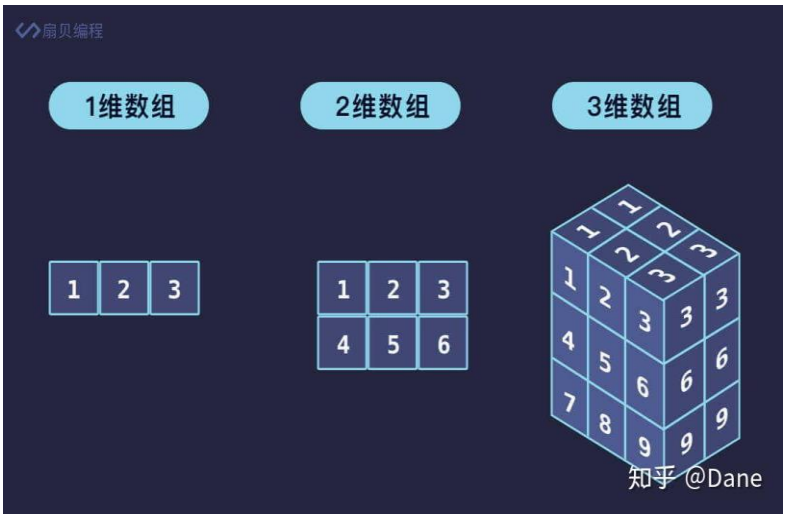
- 代码示例：

eg7.2.3.txt

7.3 多维数组

- 概述：

Java中的多维数组是指具有多个维度的数组，也就是数组中的每一个元素本身是一个数组。Java中可以定义二维数组、三维数组或者更高维度的数组。多维数组可以更直观地把数据组织成表格状或者立体状的数据结构，方便进行数据的操作和管理。



7.3.1 二重循环

- 概述： 多维数组通常需要使用二重循环进行遍历。内层循环负责处理每一行或列的元素。

- 详细说明：

外层循环：遍历多维数组的行。

内层循环：遍历每一行中的列。

7.3.1 二重循环

- 生活举例：

二重循环：就像在一个表格中遍历每一行和每一列的单元格。

- 面试题示例：

问：如何遍历一个二维数组？
答：使用嵌套的for循环，外层循环遍历行，内层循环遍历列。

7.3.1 二重循环

- 代码示例：

eg7.3.1.txt

7.3.2 控制流程进阶

- 概述：

多维数组的控制流程涉及对复杂数据结构的访问和操作，需要了解数组的行列关系。

- 详细说明：

动态维度：可以动态创建不同维度的数组。
不规则数组：多维数组不一定是规则矩阵，行的长度可以不同。

7.3.2 控制流程进阶

- 生活举例：

不规则数组：就像一个不规则的书架，每层书架上的书本数量不同。

- 面试题示例：

问：如何处理不规则的二维数组？
答：可以使用嵌套循环遍历每一行，每行的长度可能不同。

7.3.2 控制流程进阶

- 代码示例：

eg7.3.2.txt

7.3.3 二维数组

- 创建二维数组

在 Java 中二维数组被看作数组的数组，即二维数组为一个特殊的一维数组，其每个元素又是一个一维数组。Java 并不直接支持二维数组，但是允许定义数组元素是一维数组的一维数组，以达到同样的效果。

- 二维数组的定义：

```
//第一种方式：
int a[ ][ ] = {{1,2,3},{4,5,6}};

//第二种方式；
int[ ][ ] ints = new int[2][3];

//第三种方式：第二维的长度可以动态申请
int[ ][ ] arr3 = new int[5][ ];//五行的长度
```

7.3.3 二维数组

- 二维数组的遍历：

```
for(int x=0; x<a.length; x++) {
    for(int y=0; y<a[x].length; y++) {
        System.out.print(a[x][y]+" ");
    }
    System.out.println();
}
for(int[] cells : a) {
    for(int cell : cells) {
        System.out.print(cell+" ");
    }
     System.out.println();
```


7.3.3 二维数组

- 生活举例：

二维数组：就像一个棋盘，每个格子可以存放棋子的状态。

- 面试题示例：

问：如何声明和初始化一个二维数组？
答：声明时指定两个维度，初始化时可以使用嵌套数组的方式填充数据。

7.3.3 二维数组

- 提问：

- * 使用二维数组打印一个 10 行杨辉三角。

【提示】

1. 第一行有 1 个元素，第 n 行有 n 个元素
2. 每一行的第一个元素和最后一个元素都是 1
3. 从第三行开始，对于非第一个元素和最后一个元素的元素。

即： yanghui[i][j] = yanghui[i-1][j-1] + yanghui[i-1][j];

- 答案：

eg杨辉三角.txt

7.4 String类

- 特性

String类为final类，不可被继承，代表不可变的字符序列；

实现了Serializable接口，表示字符串支持序列化； 实现了Comparable接口，表示字符串可比较大小；

String对象的字符内容是存储在内部的final char型数组value中的；该数组不可再赋值，其元素也不能再改变；

字符串是常量，用双引号表示，具有不可变性，其值在创建之后不能改变；

体现：

1）当对字符串引用重新赋值时，需重新开辟内存区域进行赋值，不能使用原有的value数组进行赋值；

2）当对现有字符串进行拼接操作时，需重新开辟内存区域进行赋值，不能使用原有的value数组进行赋值；

3）当使用replace方法对字符串内容进行修改时，需重新开辟内存区域进行赋值，不能使用原有的value数组进行赋值；

7.4.1 字符串常量

- 概述：

String类用于表示字符串。Java中的字符串是不可变的，并且字符串常量池用于优化内存使用。

- 详细说明：

字符串常量池：相同内容的字符串常量只会存在一份。

不可变性：String对象一旦创建，其内容不能被修改。

7.4.1 字符串常量

- 生活举例：

字符串常量：就像一个图书馆中的书籍，如果两个人借同一本书，它们都是同一份书。

- 面试题示例：

问：什么是字符串常量池？

答：字符串常量池是一个内存区域，用于存储和复用相同内容的字符串对象，以节省内存。

7.4.1 字符串常量

- 代码示例：

eg7.4.1.txt

7.4.2 字符串对象操作

- 概述：

String类提供了多种方法用于操作字符串，包括查找、替换、分割等。

- 常用方法：length()、charAt()、substring()、indexOf()、replace()、split()等。

- 代码示例：

eg7.4.2.txt

7.4.3 字符串对象修改

- 概述：

String对象不可变，但可以通过创建新的String对象来实现内容的修改。

- 详细说明：

创建新对象：每次对字符串进行修改操作时，都会创建一个新的String对象。

- 代码示例：

eg7.4.3.txt

7.4.4 类型转换

- 概述：

类型转换分为自动类型转换和强制类型转换。

字符串和其他数据类型之间的转换非常常见。String类提供了方法进行这些转换。

- 详细说明：

字符串转基本类型：使用parse方法，如Integer.parseInt()。

基本类型转字符串：使用String.valueOf()或toString()方法。

自动类型转换

自动类型转换是指在表达式中，当两种不同的数据类型组合在一起时，较小的数据类型会自动转换为较大的数据类型，这个过程是自动的，无需编程者手动干预。

在Java中，这种转换遵守以下规则，按照容量从小到大排列，自动类型转换会遵循以下顺序：

byte -> short -> int -> long -> float -> double

比如：

```
int i = 100;
long l = i;  // 自动类型转换从int到long
float f = l; // 自动类型转换从long到float
```

在上面的例子中，int 类型的 i 被自动转换为 long 类型的 l，然后 long 类型的 l 又被自动转换为 float 类型的 f。

由容量大的类型到容量小的类型会导致精度损失而致使报错，由容量小的类型到容量大的类型一般会自动类型转换。

强制类型转换

- 强制类型转换的语法为：

```
type identify = (targetType)value;
```

就是在圆括号中加上我们想要转换成的目标类型，放到变量或数值前面。

例如：

```
double num1 = 2.1;
int num2 = (int)num1
```

这里的 num1 是 double 类型，强制转换为 int 类型会导致它的小数部分丢失。

基本数据类型向 String 类型转换

- 1、使用 + "" 方法

当一个字符串与任何其他类型的数据进行连接时，其他类型的数据会自动转换为String。然后连接成一个新的字符串。

```
int num1 = 10;
float num2 = 1.0f;
double num3 = 3.0;
long num4 = 1000L;
short num5 = 20;
boolean b = true;
String str1 = num1 + "";
String str2 = num2 + "";
String str3 = num3 + "";
String str4 = num4 + "";
String str5 = num5 + "";
String str6 = b + "";
```


基本数据类型向 String 类型转换

- 2、使用 String.valueOf() 方法：

这是一个静态方法，可以接受各种类型的参数，并将其转换为 String 表示。

```
public class Test {
    public static void main(String[] args) {
        int num1 = 10;
        double num2 = 1.1;
        boolean b = true;
        String str1 = String.valueOf(num1);
        String str2 = String.valueOf(num2);
        String str3 = String.valueOf(b);
        System.out.println(str1 + " " + str2 + " " + str3);
    }
}
```

基本数据类型向 String 类型转换

- 使用 toString() 方法：

每个基本数据类型的包装类（如 Integer、Double、Boolean 等）都有一个 toString() 方法，可以将基本类型转换为 String。

```
int i = 100;
double d = 1.2;
String str1 = Integer.toString(i);  // "100"
String str2 = Double.toString(d);
```


String 数据转换到基本数据类型

- 1、使用包装类的 parseXxx() 方法：

每个基本数据类型的包装类都提供了一个 parseXxx() 静态方法，可以将 String 转换为对应的基本类型（Xxx 是数据类型的首字母大写，如 parseInt, parseDouble, parseBoolean 等）。

```
String str1 = "100";
int i = Integer.parseInt(str1);  // 100
String str2 = "false";
boolean b = Boolean.parseBoolean(str2); // false;
```

String 数据转换到基本数据类型

- 2、使用包装类的 valueOf() 方法：

除了 parseXxx() 方法，包装类还提供了 valueOf() 方法将 String 转换为基本类型。与 parseXxx() 方法不同，valueOf() 方法返回的是封装了基本数据类型的对象而不是基本数据类型本身，但在实际使用上，由于自动拆箱的存在，使用起来并无太大差异。

```
String str1 = "100";
String str2 = "true";
String str3 = "1.11";
int num1 = Integer.valueOf(str1);
boolean b = Boolean.valueOf(str2);
double num2 = Double.valueOf(str3);
```


7.5 StringBuffer类的使用

- 概述：

StringBuffer 类在 Java 中用于创建可变的字符串。当我们需要对字符串内容进行修改时，使用 StringBuffer 可以避免生成多余的临时对象，从而提高性能。StringBuffer 是线程安全的，意味着它的方法是同步的，可以在多线程环境下安全地使用。

- 主要方法

append(String str): 追加字符串到当前 StringBuffer 对象的末尾。

insert(int offset, String str): 在指定位置插入字符串。

delete(int start, int end): 删除开始到结束位置的字符。

replace(int start, int end, String str): 使用给定的字符串替换当前字符串序列的子字符串。

reverse(): 将此字符序列用其反转形式取代。

7.6 StringBuilder类的使用

- 概述：

StringBuilder 类与 StringBuffer 类似，也是用于创建可变的字符串对象。不同之处在于，StringBuilder 不是线程安全的，它的方法不是同步的。因此，如果我们的应用不是多线程的，那么使用 StringBuilder 可以获得比 StringBuffer 更好的性能。

StringBuilder 类的方法与 StringBuffer 类相似，不过它们不是同步的。

append(String str): 追加字符串到当前 StringBuilder 对象的末尾。
insert(int offset, String str): 在指定位置插入字符串。
delete(int start, int end): 删除。
replace(int start, int end, String str): 使用给定的字符串替换当前字符串序列的子字符串。
reverse(): 将此字符序列用其反转形式取代。

选择 StringBuffer 还是 StringBuilder?

- 多线程环境：如果我们的应用运行在多线程环境中，并且需要对字符串内容进行修改，那么应该选择 StringBuffer。StringBuffer 中的大多数方法都是同步的，可以确保线程安全。
- 单线程环境：如果我们的应用是单线程的，或者我们确信字符串操作不会在多线程环境中使用，那么 StringBuilder 是更好的选择。因为其方法不是同步的，所以在性能上会比 StringBuffer 更优。

- 总结

StringBuffer 和 StringBuilder 都是为了解决字符串操作中的性能问题而设计的。它们提供了一种可变的字符串，允许进行各种字符串的修改操作，而不会产生大量的临时对象。在选择使用 StringBuffer 还是 StringBuilder 时，主要考虑是不是在多线程环境中使用。理解它们的不同之处和适用场景，可以帮助我们在 Java 编程中更有效地处理字符串。

本章重点

- 一维数组(数组中每个元素都只带一个下标)
- 常用算法(平均值,最大值,最小值,数组排序,数组复制)
- 多维数组(二维)
- 控制流程

本章作业

作业一：数组与算法综合应用

题目描述：

创建一个整型一维数组，包含10个随机生成的整数（范围在1到100之间）。

计算并打印出这个数组的平均值、最大值和最小值。

对数组进行排序（你可以使用冒泡排序、选择排序或任何你熟悉的排序算法）。

编写一个方法，接收一个整型数组和一个目标值作为参数，返回数组中所有等于目标值的元素的索引列表（如果数组中没有该元素，则返回空列表）。

使用第4步中的方法，找出并打印出排序后数组中所有等于中间值的元素的索引。

要求：

注释清晰，说明每一步操作的目的。

排序算法需自行实现，不得直接调用Java内置排序方法。

方法实现需考虑边界情况，如数组为空或目标值不存在。

本章作业

作业二：字符串操作与性能优化

题目描述：

定义一个长字符串（例如，包含一篇短文的文本），并存储在String对象中。

编写一个方法，使用String类的方法对字符串进行以下操作：

将所有大写字母转换为小写字母。

去除字符串中的所有空格。

统计并返回字符串中某个特定字符（例如，'a'）的出现次数。

重复第2步中的操作，但这次使用StringBuffer或StringBuilder对象来优化性能（特别是在处理大量字符串拼接时）。

比较使用String类和StringBuffer/StringBuilder类在性能上的差异（可以通过测量执行时间来估算，但注意在真实环境中差异可能受到多种因素影响）。

要求：

深入理解String、StringBuffer和StringBuilder之间的区别与联系。

体会在不同场景下选择合适的字符串处理类的重要性。

编写代码时考虑代码的可读性和可维护性。

计算机程序设计课程学习平台

面向河南中医药大学智能医学工程专业使用

河南中医药大学信息技术学院（智能医疗行业学院）与河南方和信息科技有限公司 联合建设

河南中医药大学信息技术学院互联网技术教学团队