Java核心_方法和数组

课程概要

学习目标

理解方法的概念

熟记方法的定义格式

能够正确定义方法并调用

能够描述出方法重载的概念

能够正确定义重载方法并调用

理解数组的概念和作用

能够按照格式定义数组并使用

熟记数组使用中可能出现的两个异常及其产生原因

能够使用循环遍历数组并查找到指定的数据

方法概述







什么是方法?

方法，也叫函数，是完成特定功能的代码块。 简单来说，一个方法就是一个功能、一个动作或一种行为

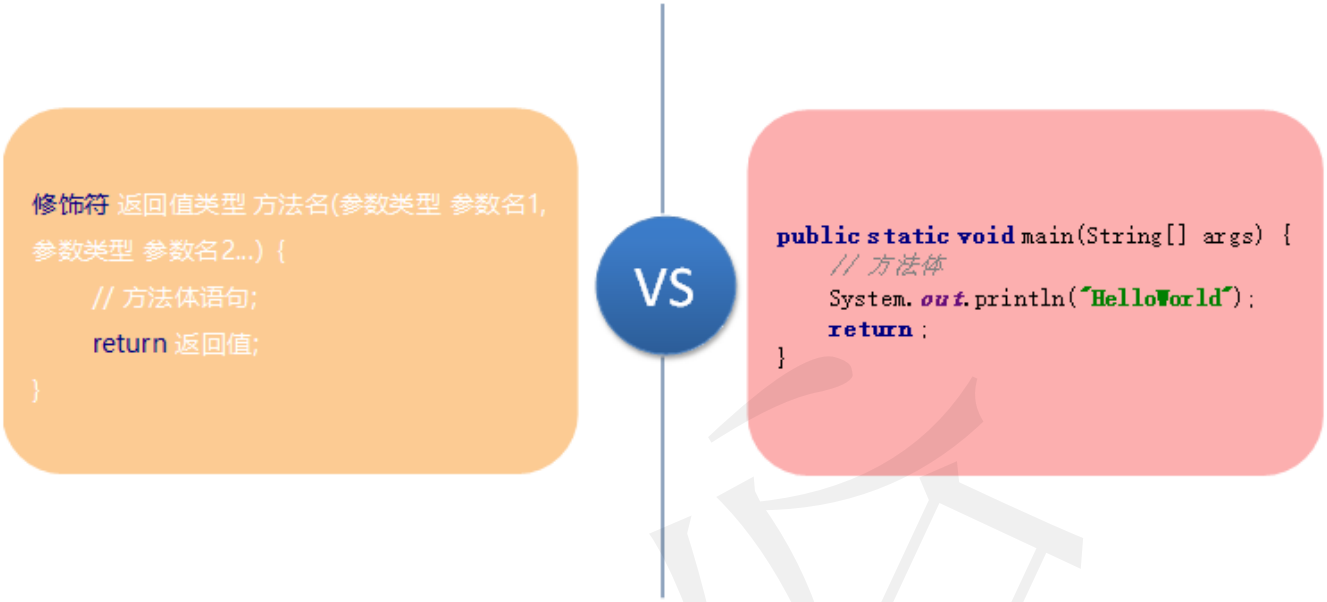
为什么需要方法?

当一些代码被反复使用时，可以把它们提取出来，放到一个方法里，以方法的形式来使用这些代码。

方法的好处

大大提高了代码的复用性，方便维护

方法的格式



方法的格式说明

- 修饰符：public static
- 返回值类型：方法执行结果的数据类型
- 方法名：方法的名称，符合标识符命名规则即可
- 参数列表：方法执行需要的条件。
  参数类型：可以是基本类型，也可以是引用类型
  参数名：即变量名
- 方法体语句：完成特定功能的代码
- return：意思是返回，用于结束方法。
- 返回值：方法执行之后的结果，返回给方法的使用者。若方法没有返回值，则它的返回值类型为void，比如main方法

方法的定义和调用

```
public static int sum(int a, int b) {

}
```

如何定义一个方法

1. **确定方法名**
2. **确定返回值类型**
3. **确定参数列表**

需求：求两个数之和

分析：

A：方法名：sum B：返回值类型：int C：参数列表：int a, int b

方法的调用

概念：

使用方法的过程称为方法的调用

调用方式

A：通过方法名调用方法 B：根据形式参数列表将实际参数传递给方法 ***形式参数***：在定义方法时需明确方法参数的形式，比如参数的**类型和个数**，故**方法定义时的参数列表称为形式参数** ***实际参数***：调用方法时传递给方法的数据，必须符合形式参数的**个数和对应位置的数据类型**

```
public static int sum(int a, int b) {
    int c = a + b;
    return c;
}
```

```
public static void main(String[] args) {
    int x = sum(3, 4);
    System.out.println(x);
}
```

方法调用过程图解

1：通过方法名调用方法

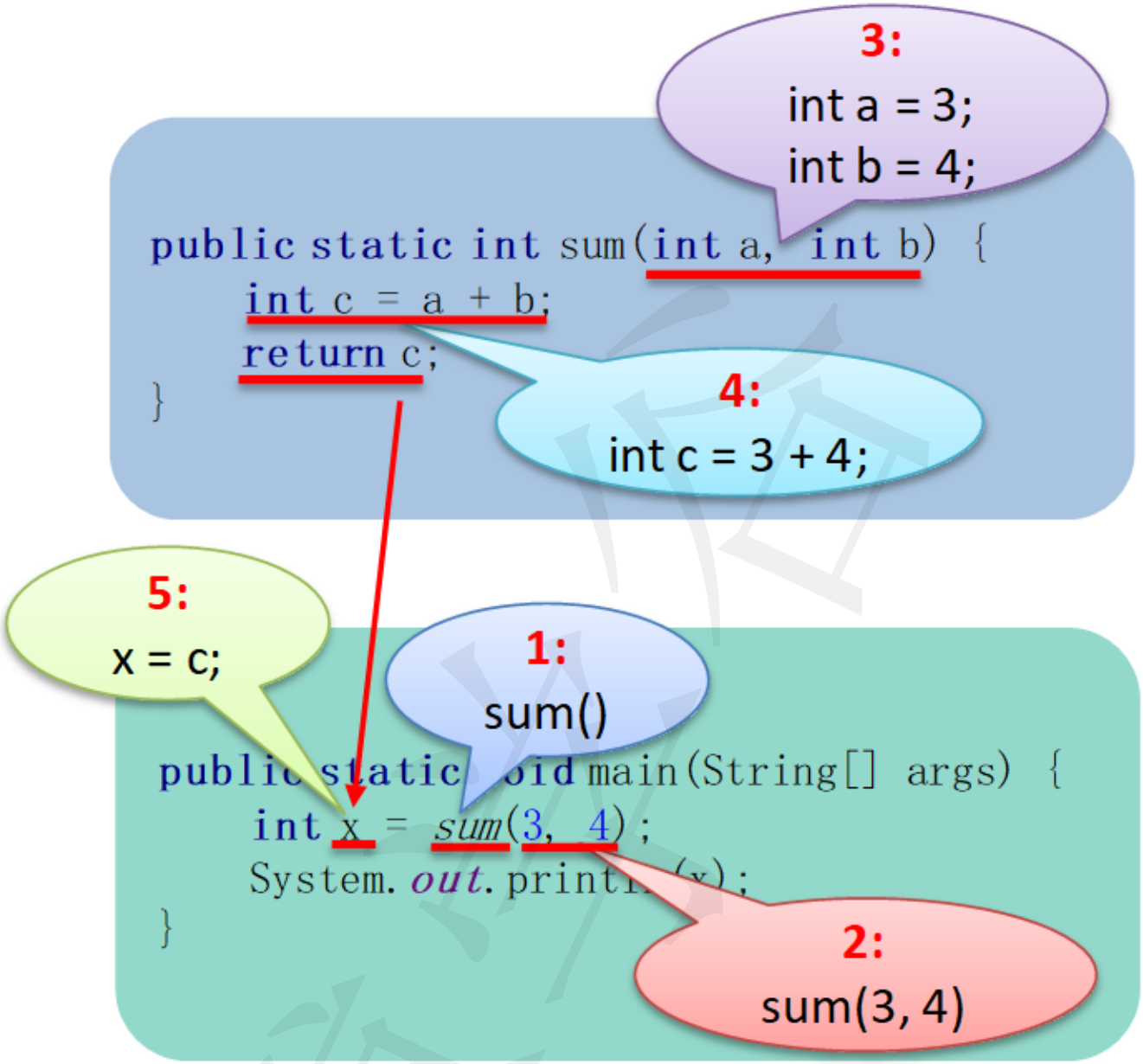
2：传递实际参数

3：方法执行时，实际参数值赋值给形式参数

4：在方法中进行运算，并将结果赋值给变量c

5：方法的返回值c赋值给接收者x

如果方法没有返回值，则不需要接收



案例：比较两个整数是否相同

需求：

键盘录入两个整数，比较它们是否相同

分析：

A：定义方法实现功能，要确定三部分内容：

方法名；返回值类型；参数列表；

B：方法名：该方法要实现“比较”功能：compare

C：返回值类型：比较的结果为是否相同，所以是boolean类型

D：参数列表：要实现比较功能，需要“两个整数”作为条件，所以参数为：int a, int b

步骤：

1.定义方法，实现比较两个int型整数是否相同的功能

2.实现键盘录入两个int型整数的功能

3.调用方法，输出比较结果

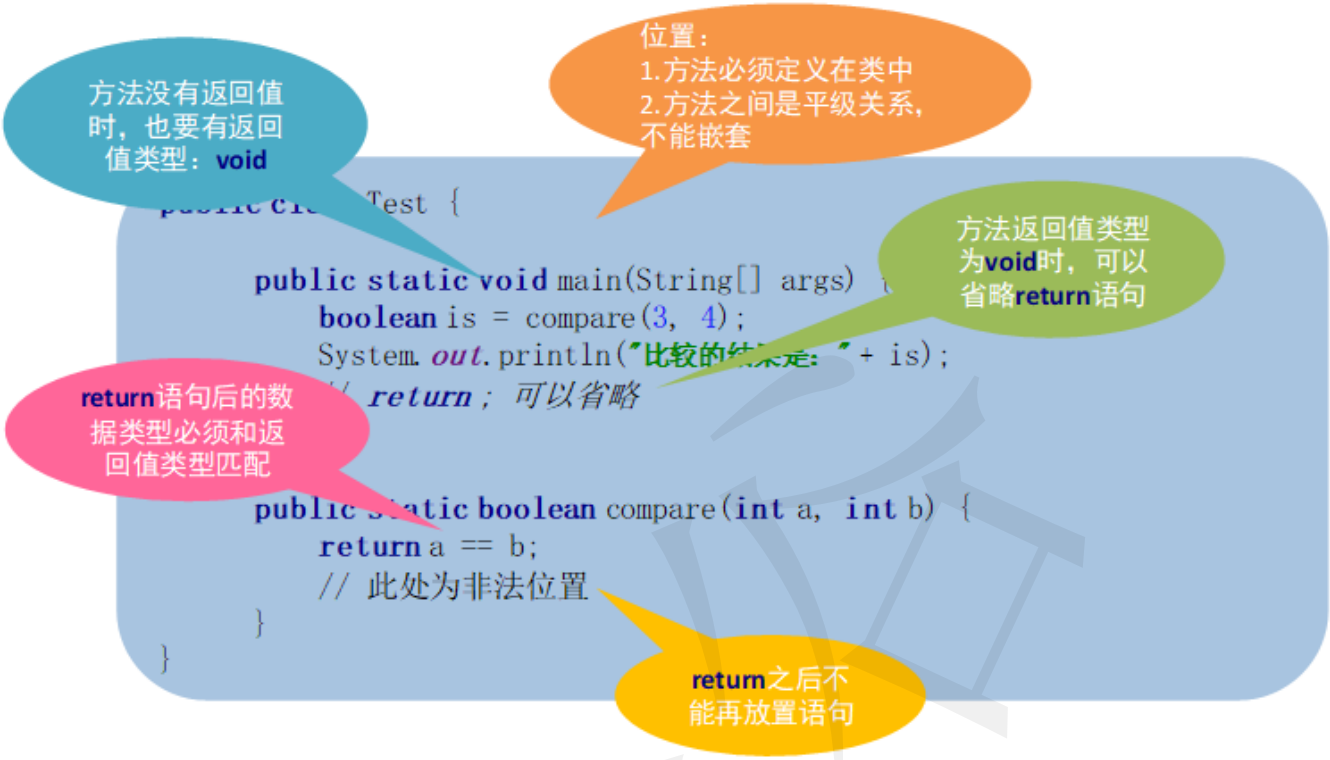
```
public static boolean compare(int a, int b) {
    return a == b;
}
```

定义方法的注意事项

```
public class Test {

    public static void main(String[] args) {
        boolean is = compare(3, 4);
        System.out.println("比较的结果是：" + is);
        // return；可以省略
    }

    public static boolean compare(int a, int b) {
        return a == b;
        // 此处为非法位置
    }
}
```



小贴士： 方法调用时，若不关心方法的返回结果，可以直接调用，省略接收返回值的动作

方法重载

什么是方法重载?

在同一个类中的多个方法，它们的方法名相同，参数列表不同，这样的情况，称为方法重载。方法重载与修饰符和返回值类型无关。

参数列表不同:

参数的个数不同 对应位置的参数类型不同

方法签名:

方法名 + 参数列表

为什么需要方法重载?

当实现的功能相同，但具体的实现方式不同时，我们可以通过定义名称相同，参数（条件）不同的方法，来更好的识别和管理类中的方法。

```
public static int sum(int a, int b) {
    return a + b;
}

public static long sum(long a, long b) {
    return a + b;
}

public static double sum(double a, float b, int c) {
    return a + b + c;
}
```

案例：比较两个数是否相同

需求：

分别比较两个int/long/double型的数是否相同

分析：

A：定义重载方法，分别实现比较两个int型数据、两个long型数据、两个double型数据是否相等的功能 B：分别定义两个int型数据、两个long型数据、两个double型数据 C：调用三次compare方法，分别传入两个int型数据、两个long型数据、两个double型数据 D：分别输出三次调用的结果

```
public static boolean compare(int a, int b) {
    System.out.println("比较int型数据");
    return a == b;
}

public static boolean compare(long a, long b) {
    System.out.println("比较long型数据");
    return a == b;
}

public static boolean compare(double a, double b) {
    System.out.println("比较double型数据");
    return a == b;
}
```

案例：判断哪些方法是重载关系

```
1. public static void open() {}

2. public static void Open(int a) {}

3. static void open(int a, int b) {}

4. public static void Open(double a, int b) {}

5. public static void OPEN(int a, double b) {}

6. public void open(int i, double d) {}

7. public static void OPEN() {}

8. public static void Open(int i, int j, byte b) {}
```

答案：

1,3,6

2,4,8

5,7

数组概述

为什么需要数组?

为了存储多个数据值

什么是数组?

数组是用来存储同一种数据类型多个元素的容器。 数据类型：可以是基本类型，也可以是引用类型 容器：比如教室、衣柜、纸箱等，可以存放多个事物

数组的定义和访问

数组的定义格式一：

数据类型[ ] 数组名 = new 数据类型[长度];

```
// 定义一个长度为3的整型数组
int[] arr = new int[3];
```

定义格式详解：

- 数据类型：即数组中存储元素的数据类型，可以是基本数据类型，也可以是引用数据类型
- [ ]：表示数组
- 数组名：数组的变量名，遵循标识符命名规范
- new：创建数组的关键字，通过new开辟内存空间
- 长度：即数组长度，数组最多能够存放元素的个数。数组长度在定义时指定，不可更改

数组的定义格式二：

数据类型[ ] 数组名 = new 数据类型[]{元素1, 元素2, 元素3...};

格式二的好处：

定义时元素是确定的，避免内存空间的浪费

数组的定义格式三：

数据类型[ ] 数组名 = {元素1, 元素2, 元素3...};

格式三是格式二的变形，简化了代码编写

```
// 格式二：定义一个长度为3的整型数组
int[] arr2 = new int[]{1, 2, 3};

// 格式三：定义一个长度为3的整型数组
int[] arr3 = {1, 2, 3};
```

数组的访问

通过数组的索引访问数组的元素

- **索引**：也叫下标、脚标，是数组元素距离数组起始位置的偏移量

  第一个元素的偏移量为0，所以数组的索引从0开始

- **格式**：数组名[索引]

  取值：数组名[索引]

  赋值：数组名[索引] = 值;

```
// 格式三：定义一个长度为3的整型数组
int[] arr3 = {1, 2, 3};
System.out.println(arr3[0]); // 输出：1
```

```
// 格式三：定义一个长度为3的整型数组
int[] arr3 = {1, 2, 3};
arr3[1] = 4; // 给第2个元素赋值为4
System.out.println(arr3[1]); // 输出：4
```

数组的遍历

需求：

给定一个**int**型数组，输出数组中的每一个元素

分析：

A：使用格式一定义一个长度为5的**int**型数组

B：为数组的前三个元素分别赋值为1，2，3

C：使用循环遍历数组

数组的长度：数组名.**length**

结论：

1：数组的最大索引为数组长度-1

2：数组中**未手动赋值**的元素，有默认值0

3：直接输出数组变量名，得到的是数组的内存地址值

```
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe"...
[I@1e643faf
1
2
3

Process finished with exit code 0
```

数组的初始化

在内存中为数组开辟连续空间并为每个元素赋值的过程

内存：计算机的重要组件，用于程序运行中临时存储数据。

连续空间：数组元素在内存空间中的存放位置是连续的

动态初始化：

只指定长度，由系统给出默认值 整数型： 0 浮点型： 0.0 字符型： '\u0000' （空字符） 布尔型： false 引用类型：null

静态初始化：

给出初始化值，由系统决定数组长度

格式一：

```
int[] arr = new int[5];
```

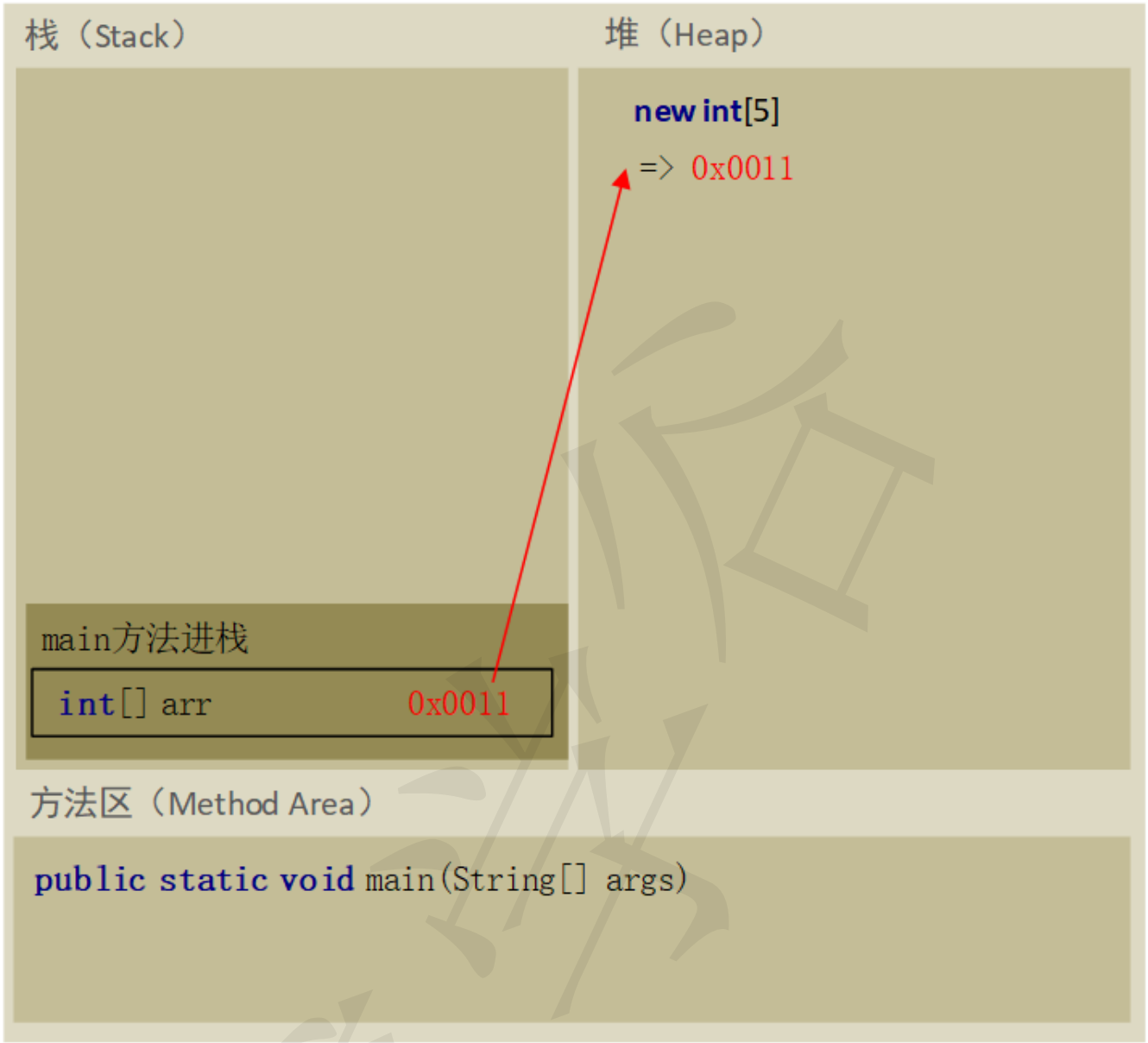
格式二：`int[] arr = new int[]{1, 2, 3, 4, 5};`

格式三：`int[] arr = {1, 2, 3, 4, 5};`

Java程序的内存分配

- 方法区：存储可运行的class文件，包含方法，静态成员，常量等（面向对象部分详解）
- 栈：方法运行时使用的内存，特点是“后进先出”，即最先进入栈区的方法最后出栈，比如main方法
- 堆：存储new出来的数组或对象（面向对象部分详解）
- 本地方法栈：JVM在调用操作系统功能时使用，与开发无关
- 寄存器：CPU使用，与开发无关

数组类型：变量arr存储的是数组在堆内存中的地址值，而不是数组元素的值，变量arr通过内存地址引用堆内存中的数组，所以数组是***引用类型。***



数组初始化过程图解

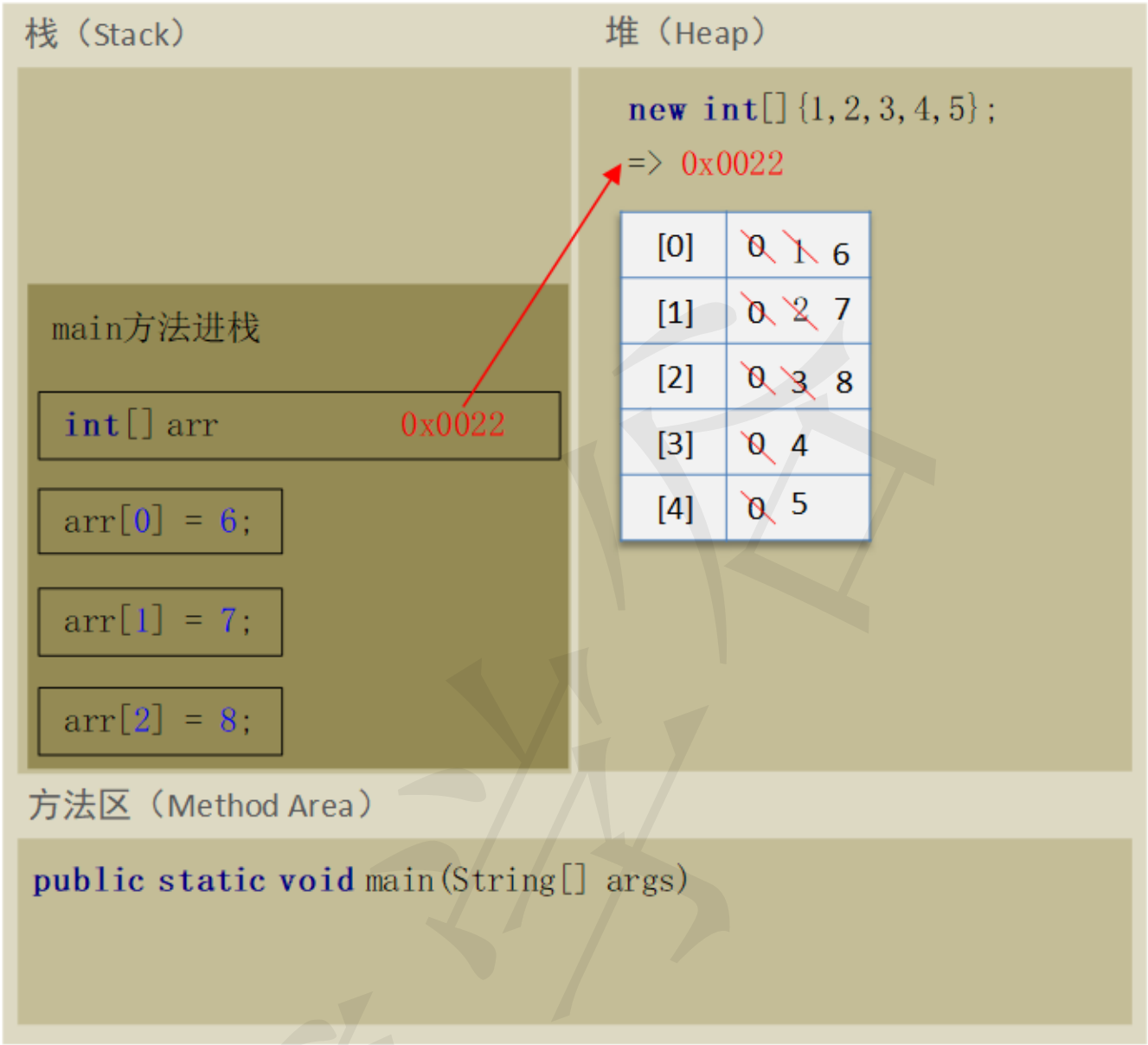
动态初始化图解：

```
public static void main(String[] args) {
        // 定义一个长度为5的整型数组
        int[] arr = new int[5];
        arr[0] = 1;
        arr[1] = 2;
        arr[2] = 3;
        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i]);
        }
}
```


栈（Stack）
堆（Heap）
new int[5];
=> 0x0011
[0] 0 1
[1] 0 2
[2] 0 3
[3] 0
[4] 0
main方法进栈
int[] arr 0x0011
arr[0] = 1;
arr[1] = 2;
arr[2] = 3;
方法区（Method Area）
public static void main(String[] args)

静态初始化图解：

```
public static void main(String[] args) {
        // 定义一个长度为5的整型数组
        int[] arr = new int[]{1, 2, 3, 4, 5};
        arr[0] = 6;
        arr[1] = 7;
        arr[2] = 8;
        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i]);
        }
}
```


栈（Stack）
堆（Heap）
new int[]{1,2,3,4,5};
=> 0x0022
[0] 0 1 6
[1] 0 2 7
[2] 0 3 8
[3] 0 4
[4] 0 5
main方法进栈
int[] arr 0x0022
arr[0] = 6;
arr[1] = 7;
arr[2] = 8;
方法区（Method Area）
public static void main(String[] args)

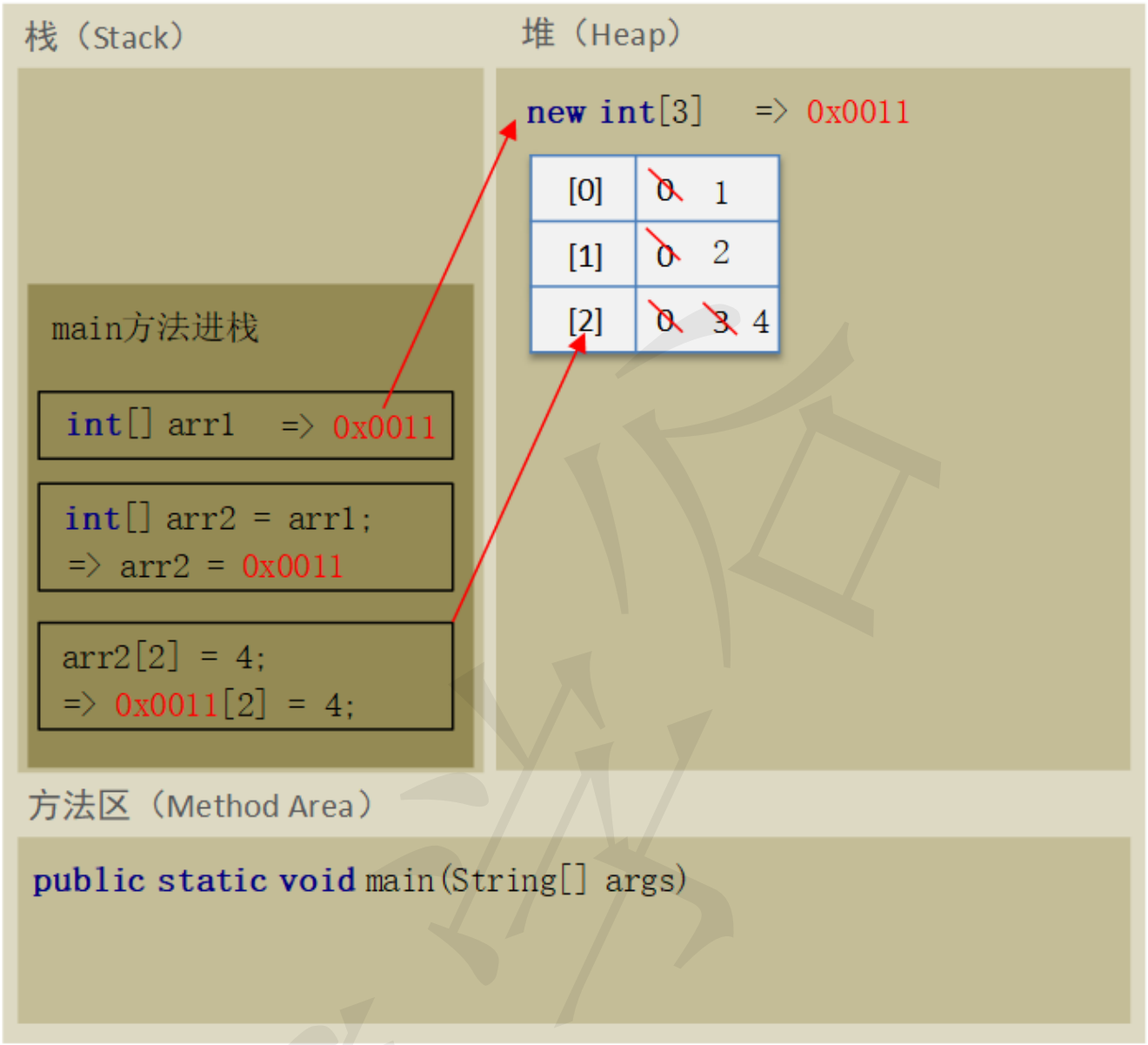
两个数组的内存图解

```
public static void main(String[] args) {
    int[] arr1 = new int[3];
    arr1[0] = 1;
    arr1[1] = 2;
    arr1[2] = 3;
    System.out.println(arr1[0]); // 1
    System.out.println(arr1[1]); // 2
    System.out.println(arr1[2]); //3

    int[] arr2 = new int[2];
    arr2[0] = 4;
    arr2[1] = 5;
    System.out.println(arr2[0]); // 4
    System.out.println(arr2[1]); // 5
}
```

一个数组两个引用的内存图解

```
public static void main(String[] args) {
    int[] arr1 = new int[3];
    arr1[0] = 1;
    arr1[1] = 2;
    arr1[2] = 3;
    System.out.println(arr1[0]); // 1
    System.out.println(arr1[1]); // 2
    System.out.println(arr1[2]); // 3
    int[] arr2 = arr1;
    arr2[2] = 4;
    System.out.println(arr1[2]); // 4
    System.out.println(arr2[2]); // 4
}
```



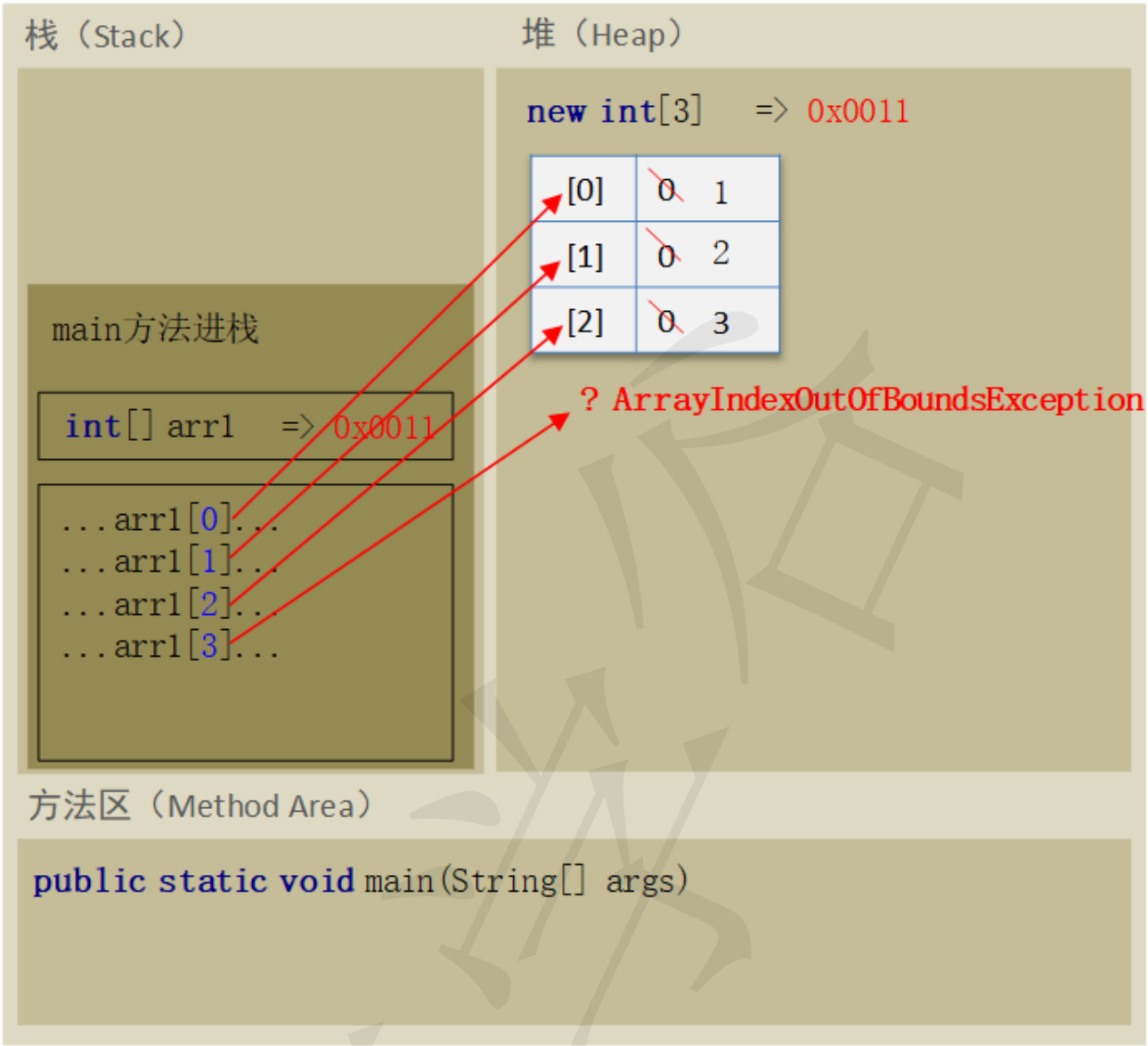
数组的常见操作

数组使用中两个常见问题

数组索引越界异常： ArrayIndexOutOfBoundsException

当访问了不存在的索引时 **异常**：即非正常情况，可以简单理解为程序运行过程中出现错误。

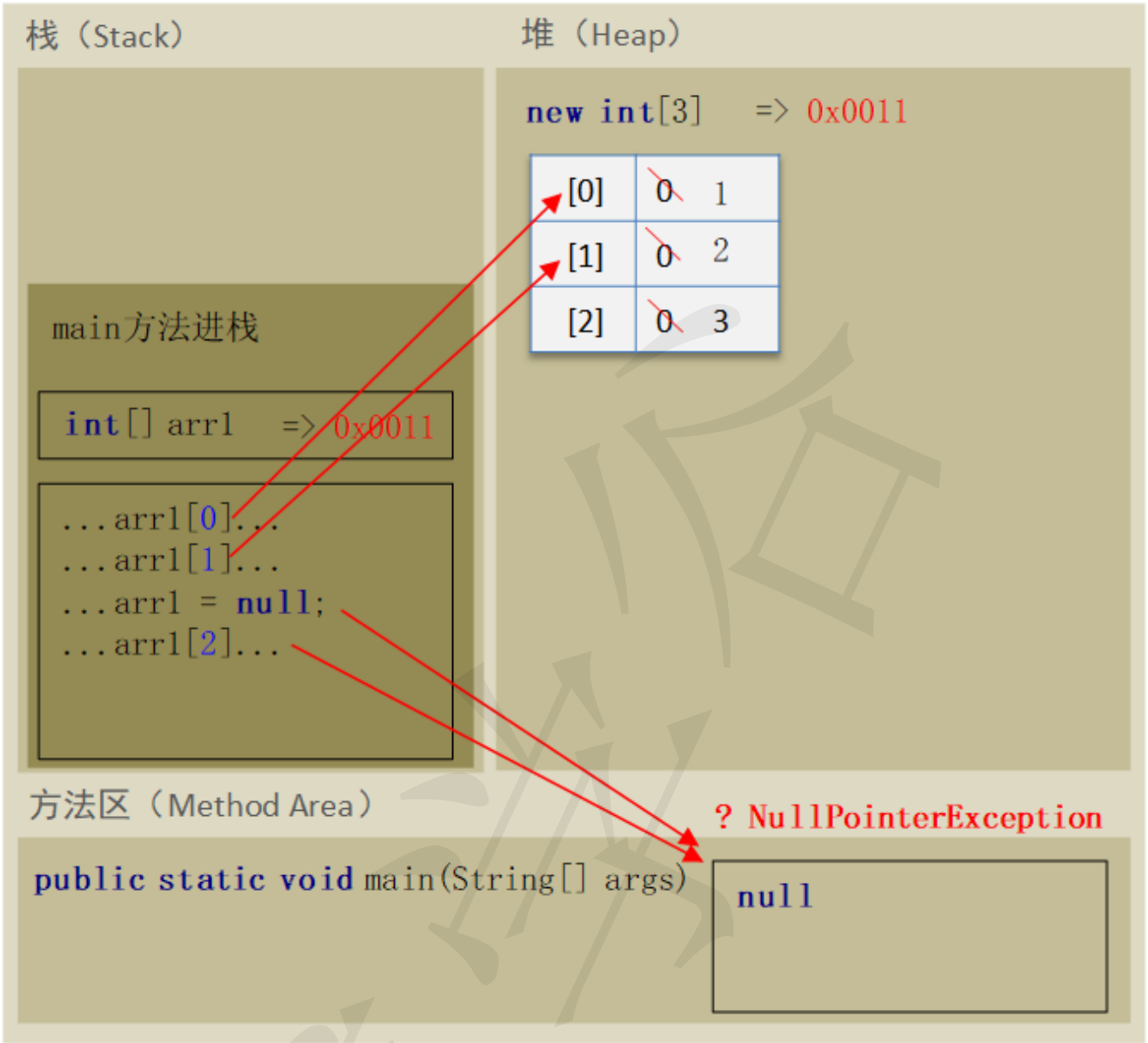
```
public static void main(String[] args) {
    int[] arr1 = new int[3];
    arr1[0] = 1;
    arr1[1] = 2;
    arr1[2] = 3;
    System.out.println(arr1[0]); // 1
    System.out.println(arr1[1]); // 2
    System.out.println(arr1[2]); // 3
    System.out.println(arr1[3]); // 报错
}
```



空指针异常：NullPointerException

数组引用存储的值为null而非数组的地址值时

```
public static void main(String[] args) {
    int[] arr1 = new int[3];
    arr1[0] = 1;
    arr1[1] = 2;
    arr1[2] = 3;
    System.out.println(arr1[0]); // 1
    System.out.println(arr1[1]); // 2
    arr1 = null;
    System.out.println(arr1[2]); // 报错
}
```



案例：获取数组中的最大值

需求：

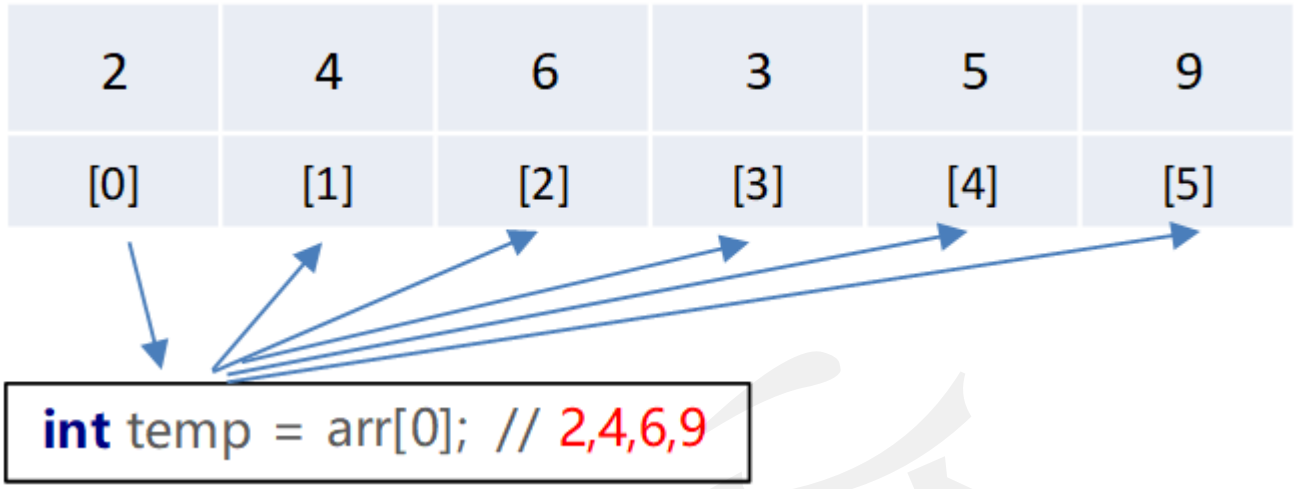
给定一个int型数组，找出它的最大元素

分析：

从第一个元素开始，依次与后面的元素比较，每次都将较大值存储在临时变量中，比较完成后临时变量即为最大值。

步骤：

A：使用数组定义格式三创建一个int型数组： int[] arr = {2, 4, 6, 3, 5, 9}；B：定义临时变量temp，代表最大元素，存储第一个值： int temp = arr[0]；C：使用for循环遍历数组第一个元素后的每一个元素 D：将数组的元素和temp的值比较，将较大值赋值给temp E：输出temp的值




```
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe"...
数组中的最大元素是：9


Process finished with exit code 0
```

案例：定义打印数组元素的方法并调用

需求：

给定一个int型数组，调用自定义方法打印数组的每一个元素

分析：

定义方法的步骤，需要确定三项内容： 方法名：printArray 参数列表：int[] arr 返回值类型：void

步骤：

A：使用数组定义格式三创建一个int型数组： int[] arr = {2, 4, 6, 3, 5, 9}；B：定义printArray方法，用于打印数组的元素： public static void printArray(int[] array) { } C：在main方法中调用该方法： printArray(arr)；D：填写方法的具体实现 E：测试执行代码

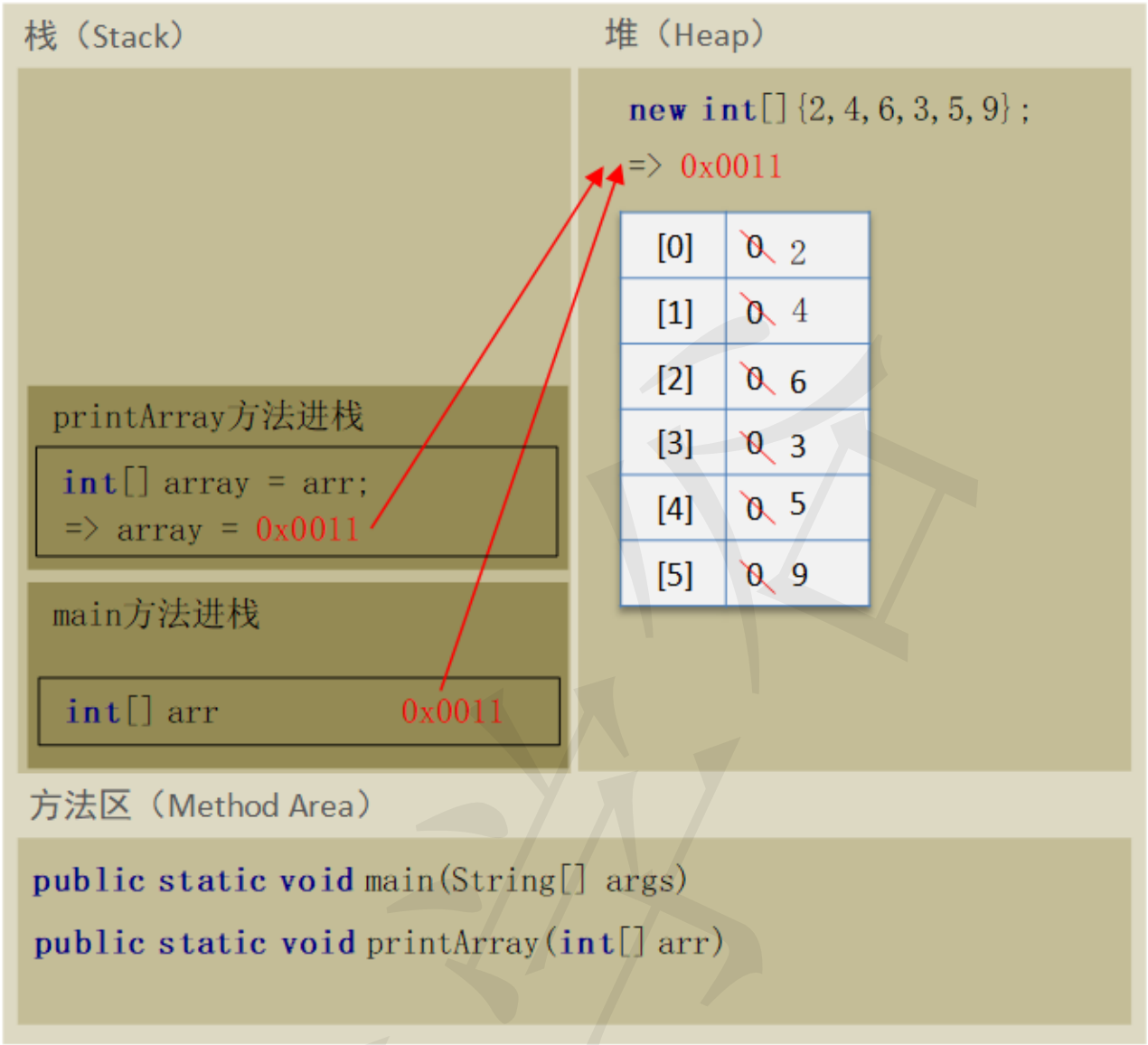
```
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe"...
2
4
6
3
5
9

Process finished with exit code 0
```

案例：定义打印数组元素的方法并调用过程图解

```
public static void main(String[] args) {
    int[] arr = {2, 4, 6, 3, 5, 9};
    printArray(arr);
}

public static void printArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.println(array[i]);
    }
}
```



结论：

引用类型的变量作为参数传递给方法时，传递的是地址值

思考：

基本类型的变量作为参数传递给方法，传过去的是什么呢？

```java
public static void main(String[] args) {
    int x = 3;
    int y = 4;
    int z = sum(x, y);
    System.out.println(x);
    System.out.println(y);
    System.out.println(z);
}
```

```java
public static int sum(int a, int b) {
    int c = a + b;
    a ++;
    b ++;
    return c;
}
```