

数组的定义与使用

问题

1. 引用作为参数传递
2. 引用和对象再重点介绍
3. 对象放哪里，是不是往后放一放

本节目标

1. 理解数组基本概念
2. 掌握数组的基本用法
3. 数组与方法互操作
4. 熟练掌握数组相关的常见问题和代码

1. 数组基本用法

1.1 什么是数组

数组本质上就是让我们能 "批量" 创建相同类型的变量。

例如：

如果需要表示两个数据，那么直接创建两个变量即可 `int a; int b`

如果需要表示五个数据，那么可以创建五个变量 `int a1; int a2; int a3; int a4; int a5;`

但是如果需要表示一万个数据，那么就不能创建一万个变量了。这时候就需要使用数组，帮我们批量创建。

注意事项：在 Java 中，数组中包含的变量必须是 **相同类型**。

1.2 创建数组

基本语法

```
// 动态初始化
数据类型[] 数组名称 = new 数据类型 [] { 初始化数据 };

// 静态初始化
数据类型[] 数组名称 = { 初始化数据 };
```

代码示例

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

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

注意事项: 静态初始化的时候, 数组元素个数和初始化数据的格式是一致的.

其实数组也可以写成

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

这样就和 C 语言更相似了. 但是我们还是更推荐写成 `int[] arr` 的形式. `int` 和 `[]` 是一个整体.

1.3 数组的使用

代码示例: 获取长度 & 访问元素

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

// 获取数组长度
System.out.println("length: " + arr.length); // 执行结果: 3

// 访问数组中的元素
System.out.println(arr[1]); // 执行结果: 2
System.out.println(arr[0]); // 执行结果: 1
arr[2] = 100;
System.out.println(arr[2]); // 执行结果: 100
```

注意事项

1. 使用 `arr.length` 能够获取到数组的长度. 这个操作为成员访问操作符. 后面在面向对象中会经常用到.
2. 使用 `[]` 按下标取数组元素. 需要注意, 下标从 0 开始计数
3. 使用 `[]` 操作既能读取数据, 也能修改数据.
4. 下标访问操作不能超出有效范围 `[0, length - 1]`, 如果超出有效范围, 会出现下标越界异常

代码示例: 下标越界

```
int[] arr = {1, 2, 3};
System.out.println(arr[100]);

// 执行结果
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 100
    at Test.main(Test.java:4)
```

抛出了 `java.lang.ArrayIndexOutOfBoundsException` 异常. 使用数组一定要下标谨防越界.

代码示例: 遍历数组

所谓 "遍历" 是指将数组中的所有元素都访问一遍, 不重不漏. 通常需要搭配循环语句.

```
int[] arr = {1, 2, 3};
for (int i = 0; i < arr.length; i++) {
    System.out.println(arr[i]);
}

// 执行结果
1
2
3
```

代码示例: 使用 for-each 遍历数组

```
int[] arr = {1, 2, 3};
for (int x : arr) {
    System.out.println(x);
}

// 执行结果
1
2
3
```

for-each 是 for 循环的另外一种使用方式. 能够更方便的完成对数组的遍历. 可以避免循环条件和更新语句写错.

2. 数组作为方法的参数

2.1 基本用法

代码示例: 打印数组内容

```
public static void main(String[] args) {
    int[] arr = {1, 2, 3};
    printArray(arr);
}

public static void printArray(int[] a) {
    for (int x : a) {
        System.out.println(x);
    }
}

// 执行结果
1
2
3
```

在这个代码中

- `int[] a` 是函数的形参, `int[] arr` 是函数实参.

- 如果需要获取到数组长度, 同样可以使用 `a.length`

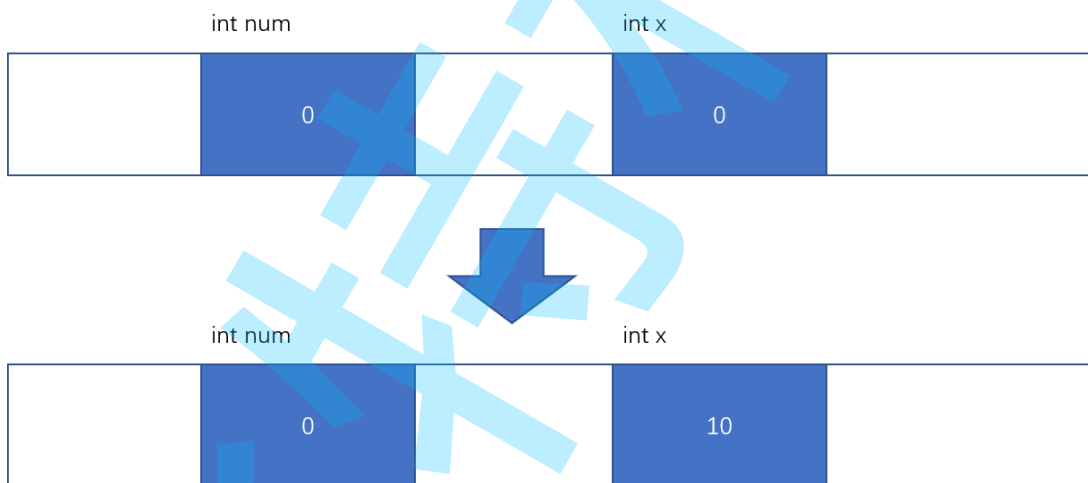
2.2 理解引用类型(重点/难点)

我们尝试以下代码

代码示例1 参数传内置类型

```
public static void main(String[] args) {  
    int num = 0;  
    func(num);  
    System.out.println("num = " + num);  
}  
  
public static void func(int x) {  
    x = 10;  
    System.out.println("x = " + x);  
}  
  
// 执行结果  
x = 10  
num = 0
```

我们发现, 修改形参 `x` 的值, 不影响实参的 `num` 值.



代码示例2 参数传数组类型

```
public static void main(String[] args) {  
    int[] arr = {1, 2, 3};  
    func(arr);  
    System.out.println("arr[0] = " + arr[0]);  
}  
  
public static void func(int[] a) {  
    a[0] = 10;  
    System.out.println("a[0] = " + a[0]);  
}  
  
// 执行结果
```

```
a[0] = 10  
arr[0] = 10
```

我们发现, 在函数内部修改数组内容, 函数外部也发生改变.

此时数组名 `arr` 是一个 "引用". 当传参的时候, 是按照引用传参.

3 初步认识引用 (reference) 和对象 (object)

首先, 我们要认识到, 所谓的引用或者对象, 都是内存中数据的一种规定的组织方式.

然后, 用一个不是特别严格的比喻来讲. 对象是军队, 引用就是一直军队的军党委. 我们做为要使用对象的人, 类比成要给一直军队发号施令的司令员, 我们不能直接越级指挥军队, 而应该通过军党委来指挥军队.

所以, 所谓对象, 是一个数据的组织单位. 所谓引用, 是对象的军党委, 当然, 引用也是由数据组成的.

我们这节课的学习中, 所谓的数组, 其实准确的表述都是数组对象, 还有我们直接学习过的字符串, 其实也是对象, 只是被 `String` 类型的引用指向的对象. 以后我们还会学习很多其他的对象的.


备注: 上面的理解, 只是从一个层面理解引用和对象, 以后我们会从其他角度, 再次去理解这两个 java 中至关重要的概念.

如果, 通过一个引用, 管理着一个对象, 我们通常将其表述为: **该引用指向该对象.**

3.1 关于引用和对象在本文中的绘制形式

为了严格区分引用和对象, 我们在本文中, 尤其在前期的课程中, 将会严格地用不同的图形进行描述引用和对象.

虽然我们在大部分的图形描述时, 引用和对象的大小都是类似的, 但实际上, 就和军队与军党委的对比一样. 对象往往是远大于引用的.



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



3.2 关于引用指向对象的一些规则


只能引用指向对象, 不能对象指向引用, 也不能有引用指向引用, 更不会对象指向对象.

只有军党委可以指挥军队, 没有军队指挥军党委.

唯一正确表示:



下面的全部是错误的:



引用同一时刻只能指向一个对象; 对象同一时刻可以被多个引用指向.

这里, 再去硬贴合我们军队的例子就不合适了, 大家把这个规则牢记即可.

正确表示:



下面是错误的：

image-20200915105005831

引用可以不指向任何对象，null 的理解。

可以认为一个军党委暂时处于赋闲状态。用 java 代码表述为：

```
int[] a = null;    // a 不指向任何的对象
```

我们通常把这种情况称为：**a 等于空，或者 a 是空。**

image-20200915105109807

引用的赋值运算，是让被赋值的引用指向该引用当前指向的对象。

```
int[] a = { 1, 2, 3, 4, 5 };
int[] b = null;

b = a;    // 让 b 指向 a 目前指向的对象
```

TODO: 图形表述

```
int[] a = null;    // a 不指向任何的对象
int[] b = { 1, 2, 3, 4, 5 };
b = a;             // 让 b 指向 a 目前指向的对象
                  // 由于 a 目前不指向任何的对象。所以 b 也不再指向任何的对象
```

引用的相等与不等比较，是比较两个引用是否指向同一个对象。

```
int[] a = { 1, 2, 3, 4, 5 };
int[] b = { 1, 2, 3, 4, 5 };

a == b;    // 由于 a 和 b 指向不同的对象，所以结果是 false
a != b;    // 由于 a 和 b 指向不同的对象，所以结果是 true

b = a;     // 让 b 指向 a 目前指向的对象
a == b;    // 由于 a 和 b 指向相同的对象，所以结果是 true
a != b;    // 由于 a 和 b 指向相同的对象，所以结果是 false
```

TODO: 图形表述

3.3 解引用

解引用 (dereference)，动词；对一个引用进行解引用，就是通过引用，找到该引用指向的对象。用我们军队的例子，所谓的解引用，就是要通过军党委要对军党委指挥的军队发布命令了。

接下来，我们通过代码，看 java 中有哪些解引用的形式。

通过引用.某某某，进行解引用操作

```
int[] arr = { 1, 2, 3, 4 5 };  
  
System.out.println(arr.length);
```

这里就是要对 arr 这个引用做解引用动作，也就是找到 arr 指向的数组对象，然后发布获取其长度的命令。

通过引用[下标]，进行解引用操作

```
int[] arr = { 1, 2, 3, 4, 5 };  
System.out.println(arr[0]);
```

这里就是要对 arr 这个引用做解引用动作时，也就是找到 arr 指向的数组对象，然后发布获取其第一个元素的命令。

3.4 引用类型支持的操作符

1. 赋值操作符: `=`
2. 比较操作符: `==` 和 `!=`
3. 解引用操作符: `.` 和 `[]`

其余的操作符一律不支持。

```
int[] a = { 1, 2, 3, 4, 5 };  
int[] b = null;  
  
b = a;           // 赋值操作符，支持  
b == a;          // 比较操作符，支持  
b != a;          // 比较操作符，支持  
b[0];            // 解引用操作符，支持  
b.length;        // 解引用操作符，支持  
  
// 以下统统是错误的  
b = b + 1;  
b = b - 1;  
b = b * 1;  
b = b / 1;
```

小Tips： 通常来说 `.` 操作符都，可以直接读作“的”。arr.length 可以读作 arr 的长度。完整表述为

```
arr 引用指向的对象中的长度。  
// 师傅，不要再念了！ ^_^
```

3.5 通过引用操作对象表现出的“共享”特性

```
int[] a = { 1, 2, 3, 4, 5 };  
System.out.println(Arrays.toString(a)); // [1, 2, 3, 4, 5]  
int[] b = a;  
System.out.println(Arrays.toString(b)); // [1, 2, 3, 4, 5]  
b[0] = 100;  
System.out.println(Arrays.toString(b)); // [100, 2, 3, 4, 5]  
System.out.println(Arrays.toString(a)); // [100, 2, 3, 4, 5];
```

由于通过引用修改的对象中的数据，而 a 和 b 指向同一个对象。所以，通过 b 修改了数组对象中的数据之后，通过 a 可以看到修改后的变化。

TODO: 图形表述

```
public static void method(int[] b) {
    System.out.println(Arrays.toString(b)); // [1, 2, 3, 4, 5]
    b[0] = 100;
    System.out.println(Arrays.toString(b)); // [100, 2, 3, 4, 5]
}

public static void main(String[] args) {
    int[] a = { 1, 2, 3, 4, 5 };
    System.out.println(Arrays.toString(a)); // [1, 2, 3, 4, 5]
    method(a);
    System.out.println(Arrays.toString(a)); // [100, 2, 3, 4, 5]
}
```

通过方法传递参数，其实也是让 b 指向 a 目前指向的对象，然后做修改。所以同样表现出“共享”特性

TODO: 图形表述

3.6 NullPointerException 异常

常被翻译为**空指针异常**，但大家要注意，这里的指针和我们 C 语言解答学过的指针毫无关系。

在对一个引用做解引用的动作时，如果引用 == null，表示这个引用不指向任何的对象。也就相当于对一个没有指挥军队的军党委发布命令。军党委当然会说，司令员，这个我们做不到的。表现在 java 中，就是会抛出 `NullPointerException` 了。

接下来，我们通过代码，看 java 出现 `NullPointerException` 的常见形式。

通过引用.某某某，进行解引用操作

```
int[] arr = null;

System.out.println(arr.length);

// 运行结果，会出现运行时异常
Exception in thread "main" java.lang.NullPointerException
```

这里就是要对 arr 这个引用做解引用动作时，由于 arr 是空，所以出现了异常。

通过引用[下标]，进行解引用操作

```
int[] arr = null;
System.out.println(arr[0]);

// 执行结果
Exception in thread "main" java.lang.NullPointerException
```

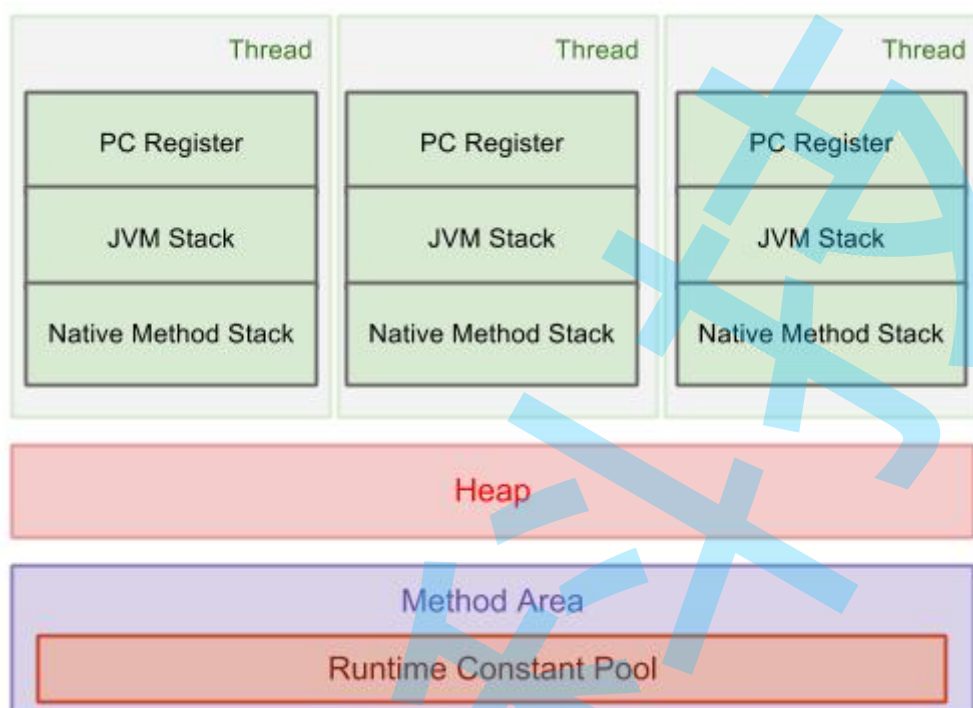
这里就是要对 arr 这个引用做解引用动作时，由于 arr 是空，所以出现了异常。

3.7 初识 JVM 内存区域划分(第一次认识)

一个宿舍楼会划分成几个不同的区域: 大一学生, 大二学生... 计算机专业学生, 通信专业学生....

内存也是类似, 这个大走廊被分成很多部分, 每个区域存放不同的数据.

JVM 的内存被划分成了几个区域, 如图所示:



- 程序计数器 (PC Register): 只是一个很小的空间, 保存下一条执行的指令的地址.
- 虚拟机栈(JVM Stack): 重点是存储**局部变量表**(当然也有其他信息). 我们刚才创建的 `int[] arr` 这样的存储地址的引用就是在这里保存.
- 本地方法栈(Native Method Stack): 本地方法栈与虚拟机栈的作用类似. 只不过保存的内容是Native方法的局部变量. 在有些版本的 JVM 实现中(例如HotSpot), 本地方法栈和虚拟机栈是一起的.
- 堆(Heap): JVM所管理的最大内存区域. 使用 `new` 创建的对象都是在堆上保存 (例如前面的 `new int[]{1, 2, 3}`).
- 方法区(Method Area): 用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据. 方法编译出的字节码就是保存在这个区域.
- 运行时常量池(Runtime Constant Pool): 是方法区的一部分, 存放字面量(字符串常量)与符号引用. (注意从JDK 1.7开始, 运行时常量池在堆上).

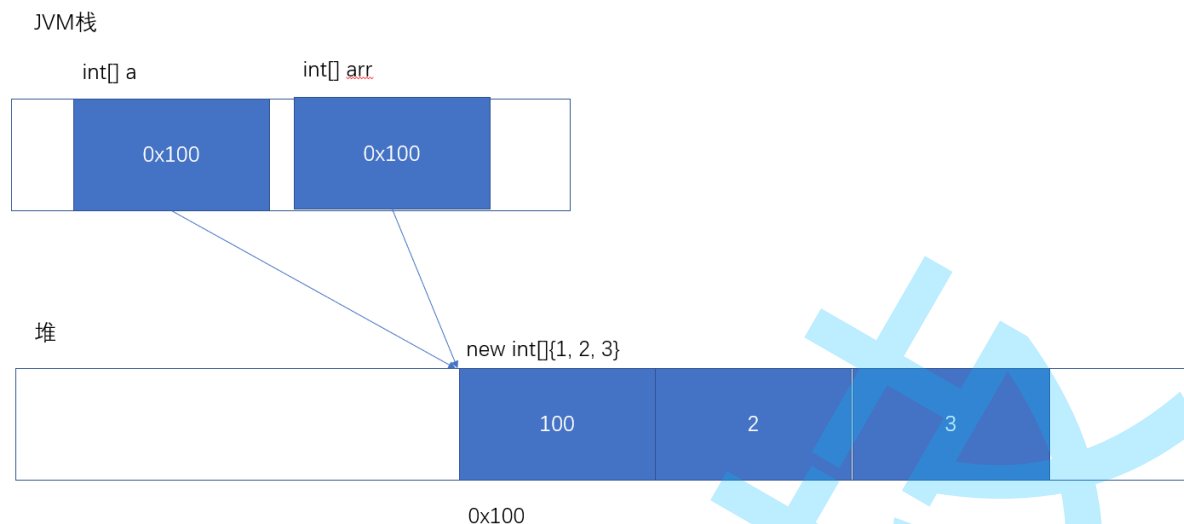
Native 方法:

JVM 是一个基于 C++ 实现的程序. 在 Java 程序执行过程中, 本质上也需要调用 C++ 提供的一些函数进行和操作系统底层进行一些交互. 因此在 Java 开发中也会调用到一些 C++ 实现的函数.

这里的 Native 方法就是指这些 C++ 实现的, 再由 Java 来调用的函数.

我们发现, 在上面的图中, 程序计数器, 虚拟机栈, 本地方法栈被很多个原谅色的, 名叫 Thread(线程) 的方框圈起来了, 并且存在很多份. 而堆, 方法区, 运行时常量池, 只有一份. (关于线程, 这是我们后面重点讲解的内容).

关于上面的划分方式, 我们随着后面的学习慢慢理解. 此处我们重点理解 虚拟机栈 和 堆.



TODO: 图不合适

- 局部变量和引用保存在栈上, new 出的对象保存在堆上.
- 堆的空间非常大, 栈的空间比较小.
- 堆是整个 JVM 共享一个, 而栈每个线程具有一份(一个 Java 程序中可能存在多个栈).

3.8 小结

1. 引用和对象都是一种数据组织形式, 所以都是存放在内存中的。
2. 对象一定保存在堆上。但引用则不一定。
3. 引用是一类只能用来指向对象的数据类型。
4. 引用可以指向对象, 也可以不指向对象。
5. 解引用操作就是通过引用找到对象。
6. java 中我们无法直接接触到对象, 都是通过引用接触到对象。
7. 对一个空的引用做解引用操作, 会出现 `NullPointerException`
8. JVM 为了便于管理, 讲它管理的内存人为的划分成 6 个区域:
 1. PC 区
 2. 虚拟机栈
 3. 本地方法栈
 4. 堆
 5. 方法区
 6. 运行时常量池

最后, 关于引用和对象的理解是整个 java 学习过程中的重点和难点, 我们会在以后的课程中多次反复地从不同的角度带着大家理解这两个概念, 切勿盲人摸象。

3. 数组作为方法的返回值

代码示例: 写一个方法, 将数组中的每个元素都 * 2

```
// 直接修改原数组
class Test {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3};
        transform(arr);
        printArray(arr);
    }
}
```

```

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

public static void transform(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        arr[i] = arr[i] * 2;
    }
}
}

```

这个代码固然可行, 但是破坏了原有数组. 有时候我们不希望破坏原数组, 就需要在方法内部创建一个新的数组, 并由方法返回出来.

```

// 返回一个新的数组
class Test {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3};
        int[] output = transform(arr);
        printArray(output);
    }

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

    public static int[] transform(int[] arr) {
        int[] ret = new int[arr.length];
        for (int i = 0; i < arr.length; i++) {
            ret[i] = arr[i] * 2;
        }
        return ret;
    }
}

```

这样的话就不会破坏原有数组了.

另外由于数组是引用类型, 返回的时候只是将这个数组的首地址返回给函数调用者, 没有拷贝数组内容, 从而比较高效.

4. 数组练习

4.1 数组转字符串

代码示例

```
import java.util.Arrays

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

String newArr = Arrays.toString(arr);
System.out.println(newArr);

// 执行结果
[1, 2, 3, 4, 5, 6]
```

使用这个方法后续打印数组就更方便一些.

Java 中提供了 `java.util.Arrays` 包, 其中包含了一些操作数组的常用方法.

什么是包?

例如做一碗油泼面, 需要先和面, 擀面, 扯出面条, 再烧水, 下锅煮熟, 放调料, 泼油.

但是其中的 "和面, 擀面, 扯出面条" 环节难度比较大, 不是所有人都能很容易做好. 于是超市就提供了一些直接已经扯好的面条, 可以直接买回来下锅煮. 从而降低了做油泼面的难度, 也提高了制作效率.

程序开发也不是从零开始, 而是要站在巨人的肩膀上.

像我们很多程序写的过程中不必把所有的细节都自己实现, 已经有大量的标准库(JDK提供好的代码)和海量的第三方库(其他机构组织提供的代码)供我们直接使用. 这些代码就放在一个一个的 "包" 之中. 所谓的包就相当于卖面条的超市. 只不过, 超市的面条只有寥寥几种, 而我们可以使用的 "包", 有成千上万.

我们实现一个自己版本的数组转字符串

```
public static void main(String[] args) {
    int[] arr = {1,2,3,4,5,6};
    System.out.println(toString(arr));
}

public static String toString(int[] arr) {
    String ret = "[";
    for (int i = 0; i < arr.length; i++) {
        // 借助 String += 进行拼接字符串
        ret += arr[i];
        // 除了最后一个元素之外, 其他元素后面都要加上 ", "
        if (i != arr.length - 1) {
            ret += ", ";
        }
    }
    ret += "]";
    return ret;
}
```

4.2 数组拷贝

代码示例

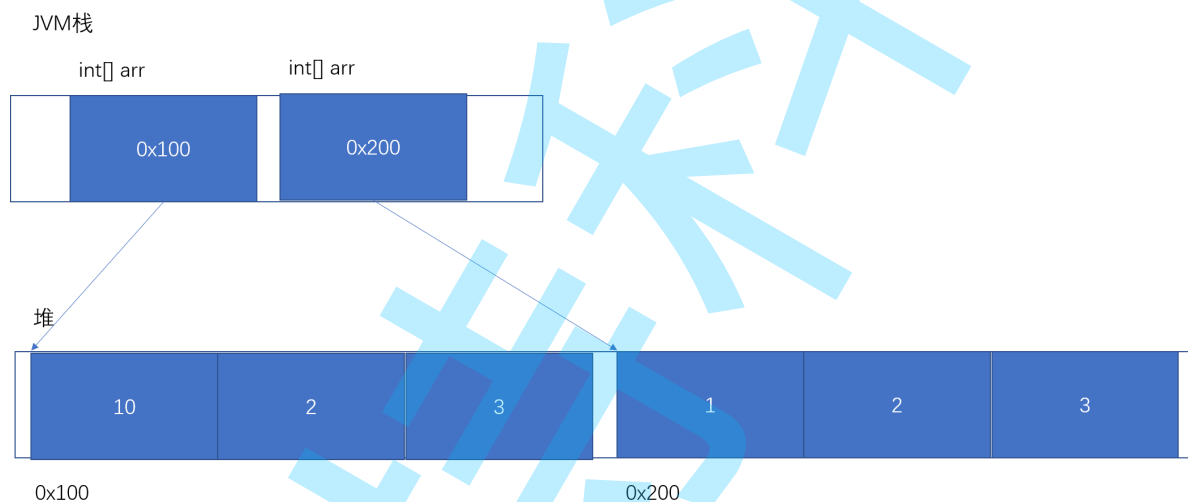
```
import java.util.Arrays

int[] arr = {1,2,3,4,5,6};
int[] newArr = Arrays.copyOf(arr, arr.length);
System.out.println("newArr: " + Arrays.toString(newArr));

arr[0] = 10;
System.out.println("arr: " + Arrays.toString(arr));
System.out.println("newArr: " + Arrays.toString(newArr));

// 拷贝某个范围.
int[] newArr = Arrays.copyOfRange(arr, 2, 4);
System.out.println("newArr2: " + Arrays.toString(newArr2));
```

注意事项: 相比于 `newArr = arr` 这样的赋值, `copyOf` 是将数组进行了 **深拷贝**, 即又创建了一个数组对象, 拷贝原有数组中的所有元素到新数组中. 因此, 修改原数组, 不会影响到新数组.



TODO: 图不合适

实现自己版本的拷贝数组

```
public static int[] copyOf(int[] arr) {
    int[] ret = new int[arr.length];
    for (int i = 0; i < arr.length; i++) {
        ret[i] = arr[i];
    }
    return ret;
}
```

4.3 找数组中的最大元素

给定一个整型数组, 找到其中的最大元素 (找最小元素同理)

代码示例

```

public static void main(String[] args) {
    int[] arr = {1,2,3,4,5,6};
    System.out.println(max(arr));
}

public static int max(int[] arr) {
    int max = arr[0];
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}

// 执行结果
6

```

类似于 "打擂台" 这样的过程. 其中 max 变量作为 **擂台**, 比擂台上的元素大, 就替换上去, 否则就下一个对手.

4.4 求数组中元素的平均值

给定一个整型数组, 求平均值

代码示例

```

public static void main(String[] args) {
    int[] arr = {1,2,3,4,5,6};
    System.out.println(avg(arr));
}

public static double avg(int[] arr) {
    int sum = 0;
    for (int x : arr) {
        sum += x;
    }
    return (double)sum / (double)arr.length;
}

// 执行结果
3.5

```

注意事项: 结果要用 double 来表示.

4.5 查找数组中指定元素(顺序查找)

给定一个数组, 再给定一个元素, 找出该元素在数组中的位置.

代码示例

```

public static void main(String[] args) {
    int[] arr = {1,2,3,10,5,6};
}

```

```

        System.out.println(find(arr, 10));
    }

    public static int find(int[] arr, int toFind) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == toFind) {
                return i;
            }
        }
        return -1; // 表示没有找到
    }

    // 执行结果
    3

```

4.6 查找数组中指定元素(二分查找)

针对**有序数组**, 可以使用更高效的二分查找.

啥叫有序数组?

有序分为 "升序" 和 "降序"

如 1 2 3 4, 依次递增即为升序.

如 4 3 2 1, 依次递减即为降序.

以升序数组为例, 二分查找的思路是先取中间位置的元素, 看要找的值比中间元素大还是小. 如果小, 就去左边找; 否则就去右边找.

代码示例

```

public static void main(String[] args) {
    int[] arr = {1, 2, 3, 4, 5, 6};
    System.out.println(binarySearch(arr, 6));
}

public static int binarySearch(int[] arr, int toFind) {
    int left = 0;
    int right = arr.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (toFind < arr[mid]) {
            // 去左侧区间找
            right = mid - 1;
        } else if (toFind > arr[mid]) {
            // 去右侧区间找
            left = mid + 1;
        } else {
            // 相等, 说明找到了
            return mid;
        }
    }
    // 循环结束, 说明没找到
    return -1;
}

```

```
}

// 执行结果
5
```

感受二分查找的效率

```
class Test {
    static int count = 0; // 创建一个成员变量，记录二分查找循环次数

    public static void main(String[] args) {
        int[] arr = makeBigArray();
        int ret = binarySearch(arr, 9999);
        System.out.println("ret = " + ret + " count = " + count);
    }

    public static int[] makeBigArray() {
        int[] arr = new int[10000];
        for (int i = 0; i < 10000; i++) {
            arr[i] = i;
        }
        return arr;
    }

    public static int binarySearch(int[] arr, int toFind) {
        int left = 0;
        int right = arr.length - 1;
        while (left <= right) {

            count++; // 使用一个变量记录循环执行次数

            int mid = (left + right) / 2;
            if (toFind < arr[mid]) {
                // 去左侧区间找
                right = mid - 1;
            } else if (toFind > arr[mid]) {
                // 去右侧区间找
                left = mid + 1;
            } else {
                // 相等，说明找到了
                return mid;
            }
        }
        // 循环结束，说明没找到
        return -1;
    }
}

// 执行结果
ret = 9999 count = 14
```

可以看到, 针对一个长度为 10000 个元素的数组查找, 二分查找只需要循环 14 次就能完成查找. 随着数组元素个数越多, 二分的优势就越大.

4.7 检查数组的有序性

给定一个整型数组, 判断是否该数组是有序的(升序)

```
public static void main(String[] args) {
    int[] arr = {1,2,3,10,5,6};
    System.out.println(isSorted(arr));
}

public static boolean isSorted(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        if (arr[i] > arr[i + 1]) {
            return false;
        }
    }
    return true;
}
```

4.8 数组排序(冒泡排序)

给定一个数组, 让数组升序(降序) 排序.

算法思路

每次尝试找到当前待排序区间中最小(或最大)的元素, 放到数组最前面(或最后面).

代码示例

```
public static void main(String[] args) {
    int[] arr = {9, 5, 2, 7};
    bubbleSort(arr);
    System.out.println(Arrays.toString(arr));
}

public static void bubbleSort(int[] arr) {
    // [0, bound) 构成了一个前闭后开区间, 表示已排序区间
    // [bound, length) 构成了一个前闭后开区间, 表示待排序区间
    // 每循环一次, 就找到一个合适大小的元素, 已排序区间就增大1.
    for (int bound = 0; bound < arr.length; bound++) {
        for (int cur = arr.length - 1; cur > bound; cur--) {
            if (arr[cur - 1] > arr[cur]) {
                int tmp = arr[cur - 1];
                arr[cur - 1] = arr[cur];
                arr[cur] = tmp;
            }
        }
    } // end for
} // end bubbleSort

// 执行结果
[2, 5, 7, 9]
```

冒泡排序性能较低. Java 中内置了更高效的排序算法

```
public static void main(String[] args) {  
    int[] arr = {9, 5, 2, 7};  
    Arrays.sort(arr);  
    System.out.println(Arrays.toString(arr));  
}
```

关于 Arrays.sort 的具体实现算法, 我们在后面的排序算法课上再详细介绍. 到时候我们会介绍很多种常见排序算法.

4.9 数组逆序

给定一个数组, 将里面的元素逆序排列.

思路

设定两个下标, 分别指向第一个元素和最后一个元素. 交换两个位置的元素.

然后让前一个下标自增, 后一个下标自减, 循环继续即可.

代码示例

```
public static void main(String[] args) {  
    int[] arr = {1, 2, 3, 4};  
    reverse(arr);  
    System.out.println(Arrays.toString(arr));  
}  
  
public static void reverse(int[] arr) {  
    int left = 0;  
    int right = arr.length - 1;  
    while (left < right) {  
        int tmp = arr[left];  
        arr[left] = arr[right];  
        arr[right] = tmp;  
        left++;  
        right--;  
    }  
}
```

4.10 数组数字排列

给定一个整型数组, 将所有的偶数放在前半部分, 将所有的奇数放在数组后半部分

例如

{1, 2, 3, 4}

调整后得到

{4, 2, 3, 1}

基本思路

设定两个下标分别指向第一个元素和最后一个元素。

用前一个下标从左往右找到第一个奇数, 用后一个下标从右往左找到第一个偶数, 然后交换两个位置的元素。

依次循环即可。

代码示例

```
public static void main(String[] args) {
    int[] arr = {1, 2, 3, 4, 5, 6};
    transform(arr);
    System.out.println(Arrays.toString(arr));
}

public static void transform(int[] arr) {
    int left = 0;
    int right = arr.length - 1;
    while (left < right) {
        // 该循环结束, left 就指向了一个奇数
        while (left < right && arr[left] % 2 == 0) {
            left++;
        }
        // 该循环结束, right 就指向了一个偶数
        while (left < right && arr[right] % 2 != 0) {
            right--;
        }
        // 交换两个位置的元素
        int tmp = arr[left];
        arr[left] = arr[right];
        arr[right] = tmp;
    }
}
```

5. 二维数组

二维数组本质上也就是一维数组, 只不过每个元素又是一个一维数组。

基本语法

```
数据类型[][] 数组名称 = new 数据类型 [行数][列数] { 初始化数据 };
```

代码示例

```
int[][] arr = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
```

```
};  
  
for (int row = 0; row < arr.length; row++) {  
    for (int col = 0; col < arr[row].length; col++) {  
        System.out.printf("%d\t", arr[row][col]);  
    }  
    System.out.println("");  
}
```

// 执行结果

1	2	3	4
5	6	7	8
9	10	11	12

二维数组的用法和一维数组并没有明显差别, 因此我们不再赘述.

同理, 还存在 "三维数组", "四维数组" 等更复杂的数组, 只不过出现频率都很低.

作业

自主实现 "数组练习" 章节的所有代码