day05-方法

今日学习内容：

方法的定义和调用

方法的设计练习

方法的重载

方法的值传递机制

今日学习目标：

了解方法的作用

掌握方法的定义

掌握方法的调用

通过多个案例掌握方法定义和调用

了解方法的可变参数

了解方法重载的作用和判断规则

了解参数的值传递机制

数组

5.5.二维数组（了解）

在之前，数组的每一个元素就是一个个的值，这种数组我们称之为一维数组。如：

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

这个数组中存储的是int类型的元素，该数组就是一个一维数组。

二维数组，就是数组中的每一个元素都是一个一维数组。

二维数组就是用在装一维数组的数组。

5.5.1 二维数组的定义和初始化

定义和静态初始化一维数组的语法：

数组元素类型[] 数组名 = new 数组元素类型[]{值1，值2，值3，...}；如：

`int[] nums = new int[]{1,3,5,7};`

定义和静态初始化二维数组的语法：

```
数组元素类型[][] 数组名 = new 数组元素类型[][]{数组1，数组2，数组3，...};
```

注意，二维数组中的元素类型是一维数组，把数组元素类型[]看成一个整体，表示数据类型。

```
public class ArrayInArrayDemo1 {
    public static void main(String[] args) {
        //定义三个一维数组
        int[] arr1 = { 1, 2, 3 };
        int[] arr2 = { 4, 5 };
        int[] arr3 = { 6 };
        //把三个一维数组存储到另一个数组中,那么该数组就是二维数组
        int[][] arr = new int[][] { arr1, arr2, arr3 };
    }
}
```

更简单的写法(字面量写法)

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

定义和动态初始化二维数组的语法：

```
// 表示声明一个二维数组，这个二维数组有x行，y列
数组元素类型[][] 数组名 = new 数组元素类型[x][y];
```

x表示二维数组中有几个一维数组

y表示每一个一维数组中有几个元素。

`int[][] arr = new int[3][5];`

System.out.println(arr.length);　　//输出3

5.5.2 获取二维数组的元素

因为二维数组表示数组的中的数组，如果要获取数组的每一个元素值，则需要两个循环嵌套。

```
//二维数组
int[][] arr = new int[][] {
    { 1, 2, 3 },
    { 4, 5 },
    { 6 }
};
```

使用for循环：

```
for (int index = 0; index < arr.length; index++) {
    //取出每一个一维数组
    int[] arr2= arr[index];
    //迭代一维数组
    for (int j = 0; j < arr2.length; j++) {
        int ele = arr2[j];
        System.out.println(ele);
    }
    System.out.println("-----");
}
```

使用for-each：

```
for (int[] arr2 : arr) {
    //arr2为每次遍历出来的一维数组
    for (int ele : arr2) {
        //ele为从arr2一维数组中遍历出来的元素
        System.out.println(ele);
    }
    System.out.println("-----");
}
```

方法

思考：之前讲解的循环操作，可以解决的是代码重复的问题，但是此时的重复的代码必须是有规律的。那循环操作，能解决所有的代码重复吗?答案肯定是不行的，比如针对于某一种功能的重复操作，循环解决不了，终于方法就该登场了。举个例子。

编写一个飞机大战游戏，程序在运行过程中，需要不断地发射子弹。假设发射子弹功能需要编写200行代码，那么，每次发射子弹都需要重复地编写这200行代码，这样的程序太low了。**在开发中我们要遵循DRY原则（Don't Repeat Yourself）——不要重复你自己的代码**，因为重复意味着**维护成本很大**，如果要修改功能代码，则每一个重复的地方都要修改一次，你敢确保每个重复的地方都能改到吗？何况，你不感到厌烦吗？不无聊吗？

为了解决功能代码重复编写的问题，可以把发射子弹的代码提取出来专门放在一个代码块（一对{}）中，并为这段代码起个唯一的名字，如此，每次发射子弹的时候直接通过这个名字就可以调用发射子弹的功能代码了。

上述过程中，被提取出来的代码可以就是类中定义的一个方法。

假设需要发送子弹N次：
发射子弹代码量 = 200 * N 行

发射子弹代码量 = 200 + 1 * N 行

```
// 需求 ： 实现输出 3 X 3 乘法表功能
1 * 1 = 1
1 * 2 = 2   2 * 2 = 4
1 * 3 = 3   2 * 3 = 6   3 * 3 = 9

// 3 X 3 乘法表功能实现
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(j + " * " + i + " = " + i * j + " ");
    }
    System.out.println();
}
```

那么，我现在需要一个 4 x 4 的乘法表?

> 一句话：重复性的功能代码需要定义成方法，通过方法调用完成重复性的执行。

6.1.定义和调用（重点）

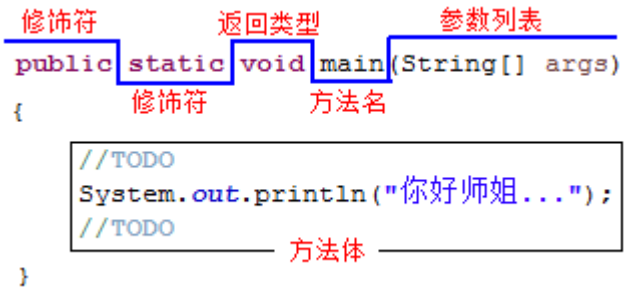
6.1.1.方法(Method)的定义（重点）

方法：**完成某一特定功能（如：求和，统计数量等）且可以被重复调用的代码块**

定义方法，语法格式：

```
[修饰符] 返回值类型 方法名称(参数类型 参数名1,参数类型 参数名2, ...) {
    // 方法体
    [return 返回值;]
}
```

看看最熟悉的main方法的定义。



格式分析：

- 修饰符：public、static等,static修饰的方法直接使用类名调用即可，目前都使用static修饰
- 返回值类型：限定返回值的类型，方法在完成一个功能后，是否需要给调用者返回一个结果?
  - 如果需要给调用者返回结果，就写上返回数据的类型
  - 如果不需要给调用者返回结果，此时使用关键字void，表示无返回值
- 方法名称：用于调用方法，遵循标识符规范，使用动词表示，首字母小写，采用用驼峰表示法
- 形式参数：方法圆括号中的变量，可以有多个，多个参数使用，号分割。
- 方法体：方法的{}中的代码，编写如何完成该功能的代码
- return关键字：在方法体中使用return关键字
  - 功能1：把值返回给该方法调用者，此时该方法不能使用void修饰
  - 功能2：结束当前方法
  - 注意：方法体没有return时，方法的返回类型声明为void，表示无返回。
- 实际参数：在调用某一个具体方法时，实际传递的参数值

注意事项：

- 方法必须定义在类中，在Java中最小的程序单元是类，必须先有类
- 一个类中可以定义多个方法
- 方法和方法是平行的，不能在方法中定义另一个方法
- 方法的定义没有先后顺序

6.1.2.方法调用（重点）

需求：在MethodDemo类中，定义一个求两个整数之和的方法

方法定义分析：

- 求两个数之和，到底是哪两个整数？说明有两个未知的因素，使用两个int类型变量表示。
- 该方法要求求两个数之和，必定需要给调用者返回一个结果，否则，毫无意义。

```
public static int getSum(int a, int b) {
    int c = a + b;
    return c;   //返回 a + b之和
}
```

注意：

- 方法中的int a和int b就是形式参数，即使参数名不叫a和b，也不影响，仅仅是变量名唯一。
- 方法定义出来，必须要调用才能生效。

方法调用：

- 方法调用格式：因为方法是static修饰的，可以直接用方法所在类的名称调用。
- 如果方法有返回类型，此时需要返回类型定义变量，接受方法返回的结果，并打印，这才有意义。
- 调用方法的地方，可以称之为调用者/调用处

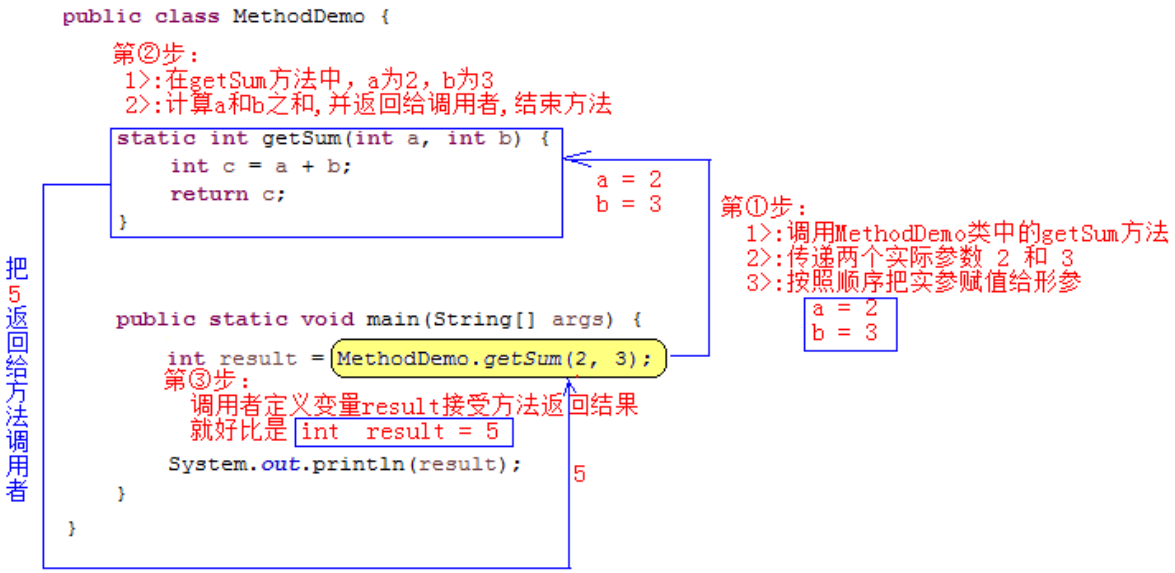
```
语法格式：返回值类型 变量 = 方法所在类名.方法名（实际参数）；
int  result = MethodDemo.getSum(2 , 3);
System.out.println(result);      //输出5
```

- 调用方法时，传递的参数2和3就是实际参数，简称实参，和顺序，类型有关。

完整代码如下：

```
public class MethodDemo {
    //定义求两个整数之和的方法
    public static int getSum(int a, int b) {
        int c = a + b;
        return c;
    }
    public static void main(String[] args) {
        //调用MethodDemo类中的getSum方法,传入2和3两个参数值,并接受方法返回的结果
        int result = MethodDemo.getSum(2, 3);
        System.out.println(result);
    }
}
```

画图分析：



6.2.设计练习（掌握）

如何定义一个方法，主要是确定有没有参数，是什么参数，有没有返回值，返回什么，这得结合具体需求来确定。

- 形式参数：完成一个功能，存在哪些未知的因素，把它们作为方法定义时的参数(形式参数)
- 返回类型：完成一个功能，要不要给调用者返回一个结果?
    - 如果需要给调用者返回结果，就写上返回数据的类型
    - 如果不需要给调用者返回结果，此时使用关键字void，表示无返回

```
static ? 方法名(?) {
    //TODO
}
```

6.2.1.方法设计练习（掌握）

需求1：定义一个方法，打印指定行数的指定字符串

```
//返回值类型：因为只要求打印指定的字符串， 所以，该方法不需要返回值，void
//参数：打印的行数和字符串都是未知的的，所以设置两个形参来接收调用者传递进来的实参
static void print(int line, String output) {
    for (int i = 0; i < line; i++) {
        System.out.println(output);
    }
}
```

调用方法：

```
public static void main(String[] args) {
    MethodTest.print(3,"你好师姐");//static修饰的方法调用：类型.方法名（实参）
    MethodTest.print(5,"你好龙哥");
}
```

需求2：定义一个方法，传入一个int数组，按照指定格式(格式:[x,x,x])打印int类型数组

```
//返回值类型：因为只要求打印指定的字符串， 所以，该方法不需要返回值，void
//参数：打印的数据所在的数组是未知的，所以定义一个形参来接收调用者传递进来的实参
static void printArray(int[] arr) {
    String str = "[";
    for (int i = 0; i < arr.length; i++) {
        str = str + arr[i];
        if (i == arr.length - 1) {
            str = str + "]";
        } else {
            str = str + ", ";
        }
    }
    System.out.println(str);
}
```

调用方法：

```
public static void main(String[] args) {
    int[] arr = new int[] { 1, 3, 5, 7, 9 };
    MethodTest.printArray(arr);
}
```

需求3：定义一个方法，传入一个int数组，返回指定元素在数组中第一次出现的索引

```
//返回值类型：因为要求返回元素的索引，所以返回值的类型应该为int
//参数：数组和要找的元素是未知的，所以定义两个形参来接收调用者传递的实参
static int indexOf(int[] arr, int key) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == key) {
            return i;
        }
    }
    return -1;//如果没找到，  返回-1
}
```

调用方法：

```
public static void main(String[] args) {
    int[] arr = new int[] { 1, 2, 3, 5, 3, 7, 9 };
    int index = indexOf(arr, 3);//main和indexOf两个方法在同一个类中，也可以直接使用方法名来调用
    System.out.println(index);
}
```

需求4：定义一个方法，传入一个int数组，返回该数组所有元素之和的平均数

```
public static double getAvg(int[] arr){
    double avg = 0.0;
    int sum = 0;
    for(int item:arr){
        sum += item;
    }
    avg = sum * 1.0 / arr.length;
    return avg;
}
```

调用方法：

```
public static void main(String[] args) {
    int[] arr = new int[] { 1, 2, 3, 5, 3, 7, 9 };
    double avg = getAvg(arr);
    System.out.println(avg);
}
```

需求5：定义一个方法，传入两个参数，一个double数组表示多个货品的价格，一个double类型的折扣，返回货品总价格

```
static double getTotalPrice(double[] arr, double cutOff) {
    double total = 0;
    for (double ele : arr) {
        total = total + ele;//求和
    }
    return total * cutOff;//返回折后总价
}
```

调用方法：

```
public static void main(String[] args) {
    double[] arr = new double[] { 10.0, 20.0, 30.0, 50.0, 30.0, 70.0, 90.0 };
    double totalPrice = getTotalPrice(arr,0.8);
    System.out.println(totalPrice);
}
```

6.2.2.方法可变参数（掌握）

在方法中传递数组有一种更简单的方式——方法的可变参数，其本质是一个**语法糖**，目的是让开发者写代码更简单。

语法：

```
[修饰符] 返回值类型 方法名称(参数类型 参数1,xx,xx,可变参数类型...参数名)
{
    方法体;
    [return 返回值;]
}
```

- 方法的可变参数其底层是就是一个一维数组类型
- 可变参数必须作为方法的最后一个参数，避免多个参数的歧义性
- 推论：方法最多只有一个可变参数

需求6：定义一个方法，传入两个参数，一个double数组表示多个货品的价格，一个double类型的折扣，返回货品总价格（使用可变参数）

```
static double getTotalPrice(double cutOff, double... arr) {
    double total = 0;
    for (double ele : arr) {
        total = total + ele;
    }
    return total * cutOff;
}
```

调用方法：

```
public static void main(String[] args) {
    double[] arr = new double[] { 10.0, 20.0, 30.0, 50.0, 30.0, 70.0, 90.0 };
    double totalPrice = getTotalPrice(0.8, arr);
    double totalPrice2 = getTotalPrice(0.8, 10.0, 20.0, 30.0, 50.0, 30.0, 70.0,
90.0);
    System.out.println(totalPrice);
}
```

6.2.3.方法重载（了解）

方法的重载（Overload），在同一类中，方法名称相同，但参数列表不同的多个方法构成方法重载。

方法重载判断原则：“两同一不同”

- 两同：在同一个类中，方法名是相同的

- 一不同：方法参数列表不同（**参数类型、参数个数、参数顺序**）
    - 只要参数类型、参数个数、参数顺序任意一个不同，就叫参数列表不同

方法重载的作用：解决了同一功能的方法由于参数不同所造成方法名称不同。

注意：方法重载和方法的返回值类型无关，只是一般要求返回值类型相同。

实际开发中，需要更简单、快速的识别方法重载。

方法签名

- 方法签名：方法名称 + 参数类型，在同一个类中，方法签名必须是唯一的，否则编译报错。

```
public class Test04Method2 {

    // addNum(int,int)
    public static int addNum(int num1,int num2) {
        int result = num1 + num2;
        return result;
    }

    public static void main(String[] args) {

    }
}
```

有了方法签名，重载：**在同一类中，方法名称相同，但方法签名不同的多个方法构成方法重载。**

以下的方法都在同一个类中，且方法名都为doWork。

```
// dowork(int,char,boolean)
void doWork(int a,char b,boolean c){}
```

三个参数：分别是int类型、char类型、boolean类型.

下列方法哪些是上述doWork方法的重载方法。

```
void doWork(char b, int a, boolean c){} //YES
int doWork(boolean a,char c ,int b){}   //YES
void doWork(int a,char b,double c){}    //YES
void doWork(int x,char y,boolean z){}   //编译报错
int doWork(int x,double y){}            //YES
int doWork(int x, char y,boolean z){}   //编译报错
```

需求：定义一个重载方法，打印方法print

6.3.方法参数的值传递机制（理解）

值传递：方法在调用时，把实参的值**复制一份**给形参的过程称为值传递。java方法调用时的值传递过程就采用值传递。

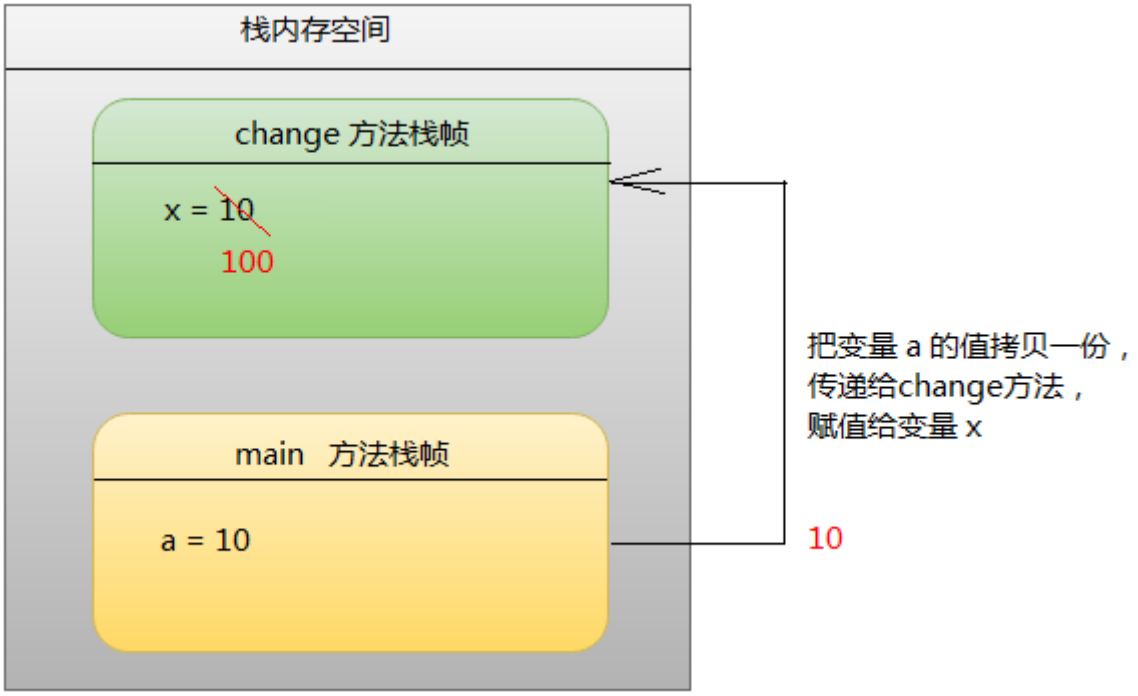
6.3.1.基本类型参数（理解）

```
static void change(int x) {
    System.out.println("change before,x=" + x);//10
    x = 100;// 修改x变量的值
    System.out.println("change after,x=" + x);//100
}

public static void main(String[] args) {
    int a = 10;
    System.out.println("main before,a=" + a);//10
    change(a);
    System.out.println("main after,a=" + a);//100? 10?
}
```

输出结果：

```
main before,a=10
change before,x=10
change after,x=100
main after,a=10
```



当把main方法中a变量的值复制一份，传递给change方法，main方法中的a变量的值不受改变。

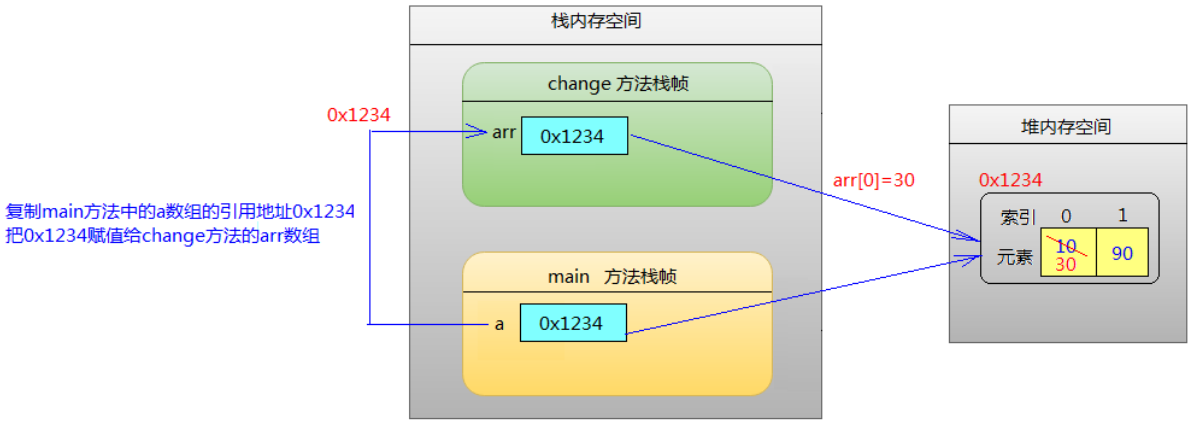
6.3.2.引用类型参数（理解）

```
static void change(int[] arr) {
    System.out.println("change before,arr[0]=" + arr[0]);//10
    arr[0] = 30;//修改数组中第一个元素的值
    System.out.println("change after,x[0]=" + arr[0]);//30
}

public static void main(String[] args) {
    int[] a = new int[] { 10, 90 };
    System.out.println("main before,a[0]=" + a[0]);//10
    change(a);
    System.out.println("main after,a[0]=" + a[0]);//30
}
```

输出结果：

```
main before,a[0]=10
change before,arr[0]=10
change after,x[0]=30
main after,a[0]=30
```



小结：

传递基本类型参数：传递参数**值**

传递引用类型参数：传递参数所引用的堆空间地址值