day05——方法

各位同学，今天我们学习的内容是方法。方法也是Java语言中一个很重要的组成部分，在实际开发中几乎每时每刻都在使用方法。所以对于今天的课程一定要搞清楚。

我们先来学习一下方法是什么

一、方法概述

1.1 方法是什么

方法是一种语法结构，它可以把一段代码封装成一个功能，以便重复调用。这句话什么意思呢？意思是，把一段功能代码围在一起，别人都可以来调用它。

下图是方法的完整格式

```
修饰符  返回值类型  方法名( 形参列表 ){
      方法体代码(需要执行的功能代码)
      return 返回值;
}
```

我们看一个需求，比如现在张工、李工两个人都需要求两个整数的和。不使用方法，代码如下。

```
// 1、李工。
int a = 10;
int b = 20;
int c = a+b;
System.out.println("和是：" + c);


// 2、张工。
int a1 = 10;
int b1 = 20;
int c1 = a1+b1;
System.out.println("和是：" + c1);
```

阅读上面的代码，我们不难发现。两次求和的代码中，除了求和的数据不一样，代码的组织结构完全一样。

像这种做相同事情的代码，就可以用方法进行封装。需要用到这段代码功能时，让别人调用方法就行。代码如下

```
//目标：掌握定义方法的完整格式，搞清楚使用方法的好处。
public class MethodDemo1 {
    public static void main(String[] args) {
        // 需求：假如现在很多程序员都要进行2个整数求和的操作。
        // 1、李工。
        int rs = sum(10, 20);
        System.out.println("和是：" + rs);
```

```
        // 2、张工。
        int rs2 = sum(30, 20);
        System.out.println("和是：" + rs2);
    }

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

1.2 方法的执行流程

当调用一个方法时，执行流程，按照下图中标注的序号执行。

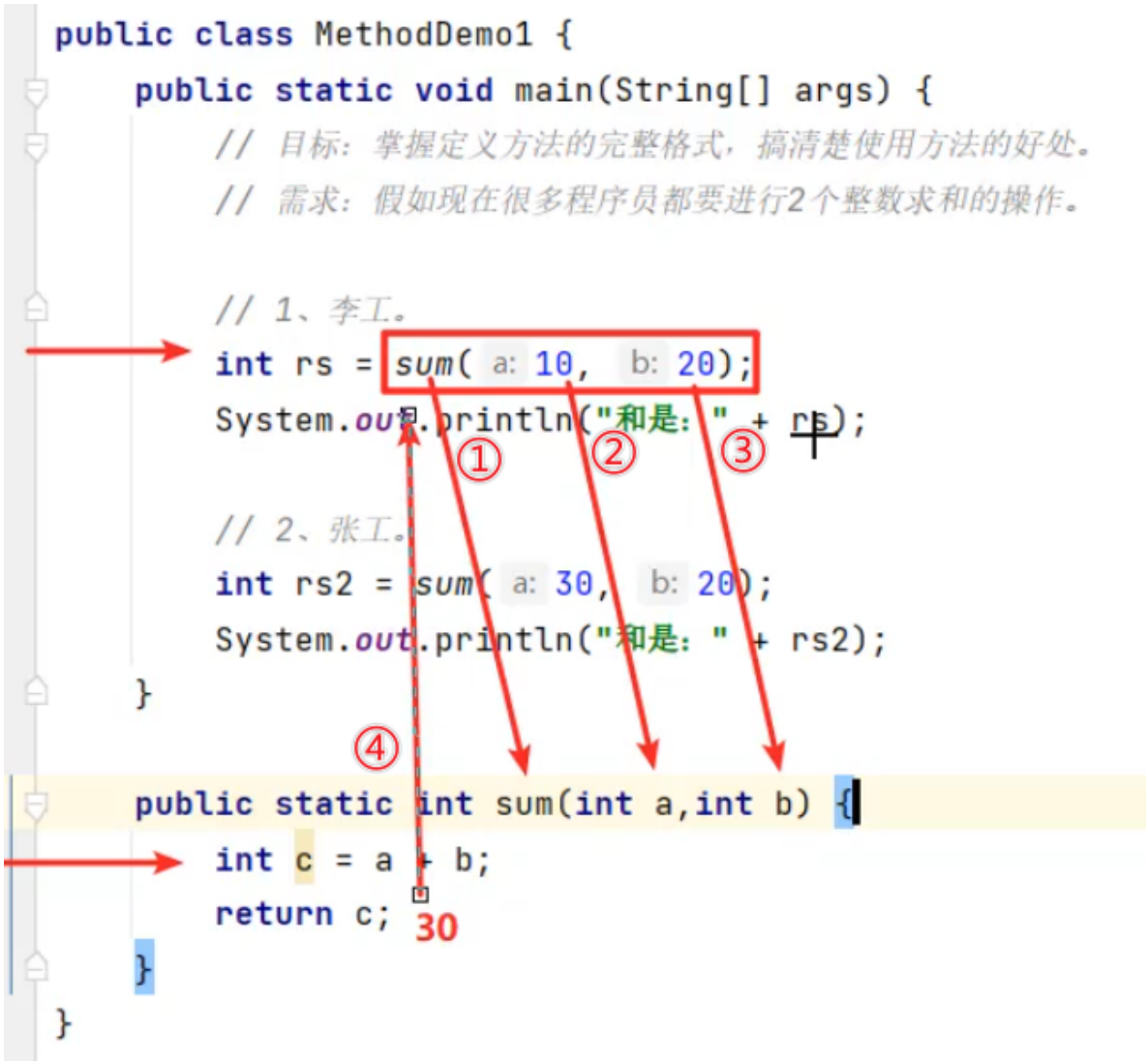
① 通过sum方法名找到sum方法

② 把10传递给方法中的参数a

③ 把20传递给方法中的参数b；

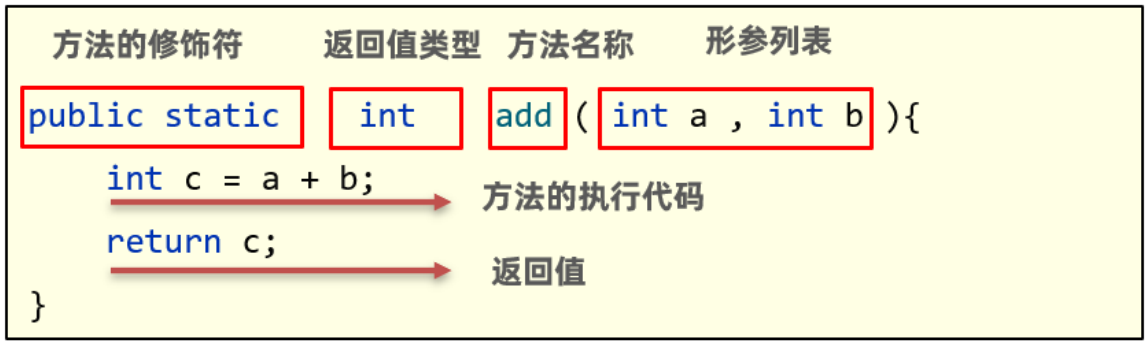
④ 执行方法中的代码，此时`int c=a+b;`；相当于`int c = 10+20`；c的值为30

`return c`的含义是，把c的结果返回给调用处。 也就是调用sum方法的结果为30，



学习完方法的执行流程之后，下面有几个注意事项需要我们写代码时注意一下。

1.3 定义方法的注意点



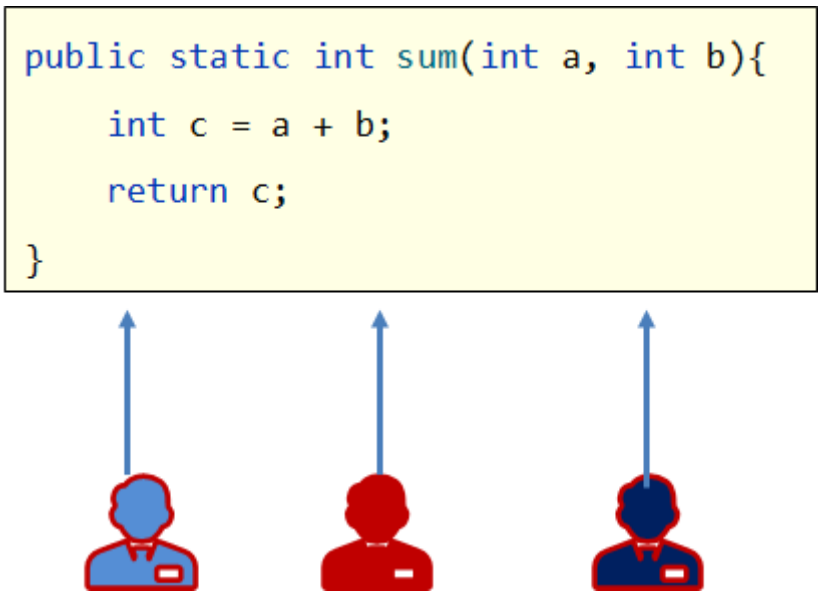
1. 方法的修饰符：暂时都使用public static 修饰。（目前看做是固定写法，后面是可以改动的）
2. 方法申明了具体的返回值类型，内部必须使用return返回对应类型的数据。
3. 形参列表可以有多个，甚至可以没有； 如果有多个形参，多个形参必须用“，”隔开，且不能给初始化值。

1.4 使用方法的好处

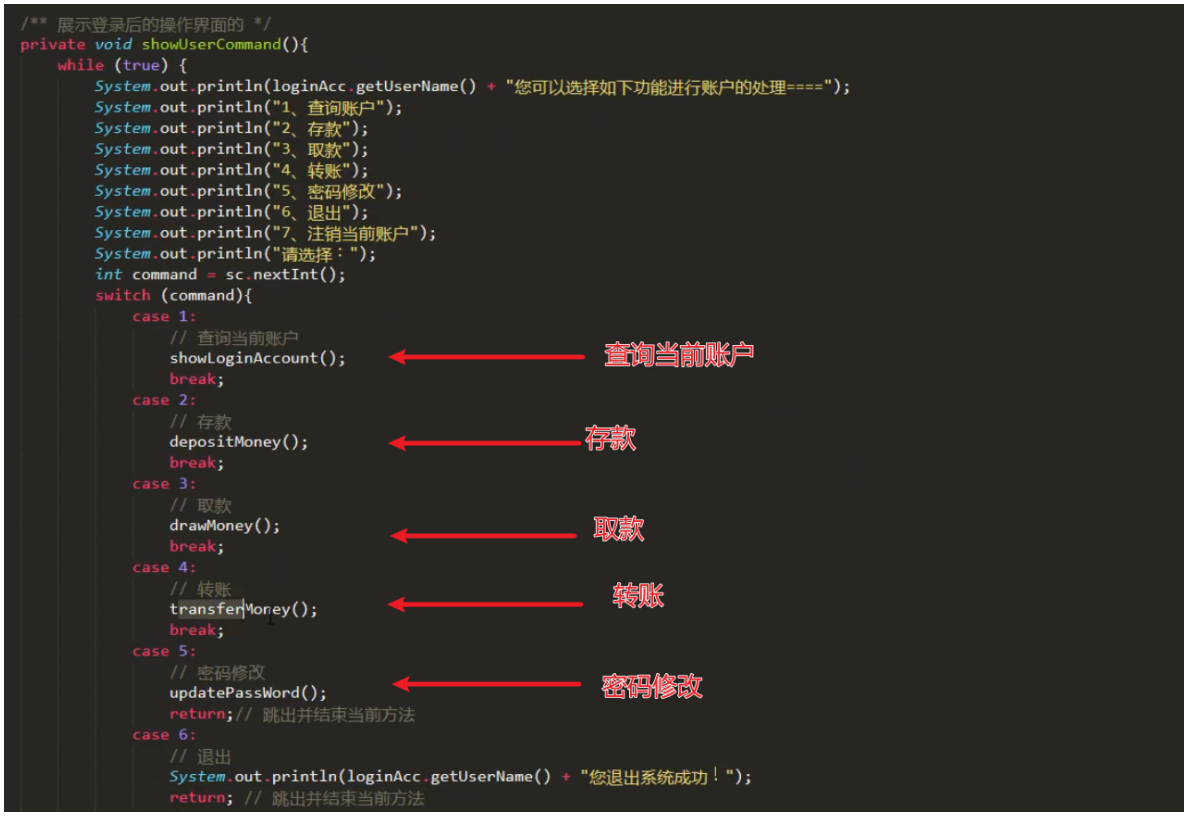
最好，我们总结一下，用方法有什么好处，可以归纳为下面2点：

1. 提高了代码的复用性，提高了开发效率。
2. 让程序的逻辑更清晰。

如下图所示：写好一个方法之后，每一个人都可以直接调用，而不用再重复写相同的代码。所以是提高了代码的复用性，不用写重复代码，自然也提高了开发效率。



那么让程序的逻辑更加清晰，是如何体现的呢？ 比如，我们后期会用所学习的技术，做一个ATM系统，ATM系统中有查看账户、存钱、取钱、修改密码等功能，到时候我们可以把每一个功能都写成一个方法。如下图所示，这样程序的逻辑就更加清晰了。



好了，关于方法是什么，以及方法的基本使用就学习到这里。

总结一下

```
1.什么是方法？
    答：方法是一种语法结构，它可以把一段代码封装成一个功能，以便重复调用
2.方法的完整格式是什么样的？
    //格式如下：
    修饰符   返回值类型   方法名( 形参列表 ){
        方法体代码(需要执行的功能代码)
        return 返回值;
    }
3.方法要执行必须怎么办？
    必须调用才执行；
    //调用格式：
    方法名(...);

4.使用方法有什么好处？
    答：提高代码的复用性，提高开发效率，使程序逻辑更清晰。
```

二、方法的其他形式

各位同学，刚才我们学习了定义完整格式的方法。但是实际开发中，需要按照方法解决的实际业务需求，设计出合理的方法形式来解决问题。

实际上设计一个合理的方法，需要重点关注下面两点

```
修饰符 返回值类型 方法名(形参列表){
    方法体代码(需要执行的功能代码)
    return 返回值;
}
```

设计一个合理的方法的原则如下：

- 如果方法不需要返回数据，返回值类型必须申明成void（无返回值申明），此时方法内部不可以使用return返回数据。
- 方法如果不需要接收外部传递进来的数据，则不需要定义形参，且调用方法时也不可以传数据给方法。
- 没有参数，且没有返回值类型（void）的方法，称为值无参数、无返回值方法。此时调用方法时不能传递数据给方法。

接下来我们看几个案例代码，练习根据实际需求定义出合理的方法

需求1：写一个方法，打印3个"Hello World"

分析：需求已经非常明确，打印的是3个HelloWorld，在方法中直接循环3次就可以完成需求。不需要外部给方法传递数据，所以不需要参数。

```
//打印3个"Hello World"
public static void printHelloWorld1(){
    for (int i = 1; i <= 3; i++) {
        System.out.println("Hello World");
    }
}
```

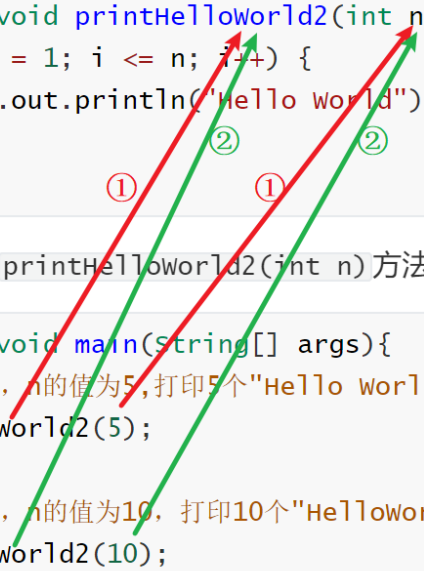
调用方法时，使用`方法名()`就可以了。比如，接下来，在main方法中调用`printHelloWorld1()`方法

```
public static void main(String[] args){
    //调用printHelloWorld1()方法
    printHelloWorld1();
}
```

需求2：写一个方法，打印若干个"Hello World"，具体多少个，有调用者指定

分析：需求不明确打印HelloWorld的个数，而是需要调用者指定。也就是说，调用者调用方法时需要给方法传递打印HelloWorld的个数。那么定义方法时，就需要写一个参数，来接收调用者传递过来的个数。

```java
//打印n个"Hello World", n的值有调用者指定。
public static void printHelloWorld2(int n){第一次调用：n=5
    for (int i = 1; i <= n; i++) {          第二次调用：n=10
        System.out.println("Hello World");
    }
}
```



在main方法中调用`printHelloWorld2(int n)`方法，此时需要给n传递实际值。

```java
public static void main(String[] args){
    //第一次调用，n的值为5,打印5个"Hello World"
    printHelloWorld2(5);

    //第二次调用，n的值为10，打印10个"HelloWorld"
    printHelloWorld2(10);
}
```

三、方法使用常见的问题

各位同学，自己第一次写方法时，或多或少会可能会出现一些问题。下面把使用方法时，常见的问题整理一下。

目的是让同学们，以后写方法时避免出现这些问题。一旦出现这些问题，要知道是什么原因。

- 1. 方法在内种没有先后顺序，但是不能把一个方法定义在另一个方法中。

- 2. 方法的返回值类型写void（无返回申明）时，方法内不能使用return返回数据，
  如果方法的返回值类型写了具体类型，方法内部则必须使用return返回对应类型的数据。

- 3. return语句的下面，不能编写代码，属于无效的代码，执行不到这儿。

- 4. 方法不调用就不会执行， 调用方法时，传给方法的数据，必须严格匹配方法的参数情况。

- 5. 调用有返回值的方法，有3种方式：
  ① 可以定义变量接收结果
  ② 或者直接输出调用，
  ③ 甚至直接调用；

- 6. 调用无返回值的方法，只有1种方式： 只能直接调用。

四、方法的案例

4.1 方法案例1

案例 计算1-n的和

需求：

- 求 1-n的和。

分析：

1．方法是否需要接收数据进行处理？ 需要接收n具体的值，因此形参声明为：int n.

2．方法是否需要返回数据？ 需要返回1-n的求和结果，因此返回值类型声明为int.

3．方法内部的业务：完成求1-n的和并返回。

按照需求：定义方法如下

```
/*
分析：
    需要求1~n的和，由于n不确定是多少，所以就把n写成形式参数，n的具体值由调用者指定。
    在方法中把n当做一个确定的数据来使用就行。
*/
public static int add(int n){
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        // i = 1 2 3 ... n
        sum += i;
    }
    return sum;
}
```

定义好方法之后，在main方法中调用

```
public static void main(String[] args) {
    int rs = add(5);
    System.out.println("1-5的和是：" + rs); //15

    int rs = add(6);
    System.out.println("1-6的和是：" + rs); //21
}
```

4.2 方法案例2

案例 判断一个整数是奇数还是偶数

需求：

- 判断一个整数是奇数还是偶数，并把判断的结果输出出来。

分析：

1. 方法是否需要接收数据进行处理？ 需要接收一个整数来判断，因此形参声明为：int number.

2. 方法是否需要返回数据？ 方法内部判断完后直接输出结果即可，无需返回，因此返回值类型声明为：void

3. 方法内部的业务：通过if语句判断number是奇数还是偶数，并输出结果。

按照需求：定义方法如下

```
/*
分析：
    需求中，是要判断一个数是奇数还是偶数，但是并没有明确说，是哪一个数。
    也就是说这个数可能是奇数，也可以能是偶数，是一个能够变化的数。
    把这个数写成方法的形式参数，就可以达到这个目的。因为调用方法时，调用者可以给传递      奇数，也可以传递偶数。
*/
public static void judge(int number){
    if(number % 2 == 0){
        System.out.println(number + "是一个偶数！");
    }else {
        System.out.println(number + "是一个奇数！");
    }
}
```

定义好方法之后，在main方法中调用

```
public static void main(String[] args) {
    judge(7); //调用后打印：7是一个奇数
    judge(8); //调用后打印：8是一个偶数
}
```

五、方法在计算机中的执行原理

各位同学，刚才我们已经写了好几个方法并成功调用了。但是不知道同学们奇不奇怪一个问题。方法在计算机的内存中到底是怎么干的呢？

为了让大家更加深刻的理解方法的执行过程，接下来，给同学们讲一下方法在计算机中的执行原理。理解方法的执行原理，对我们以后知识的学习也是有帮助的。

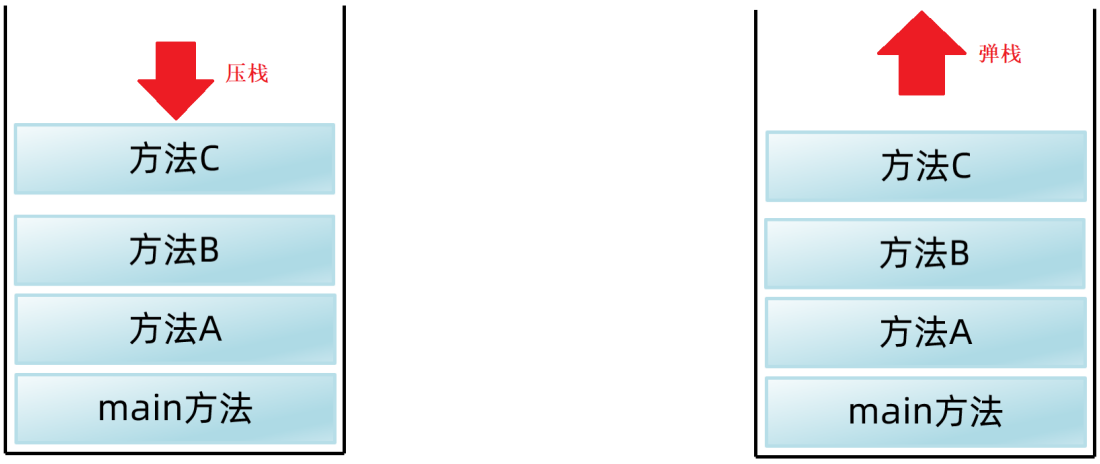
我们知道Java程序的运行，都是在内存中执行的，而内存区域又分为栈、堆和方法区。那Java的方法是在哪个内存区域中执行呢？

答案是栈内存。 **每次调用方法，方法都会进栈执行；执行完后，又会弹栈出去。**

方法进栈和弹栈的过程，就类似于手枪子弹夹，上子弹和击发子弹的过程。最后上的一颗子弹是，第一个打出来的；第一颗上的子弹，是最后一个打出来的。

假设在main方法中依次调用A方法、B方法、C方法，在内存中的执行流程如下：

- 每次调用方法，方法都会从栈顶压栈执行没执行
- 每个方法执行完后，会从栈顶弹栈出去

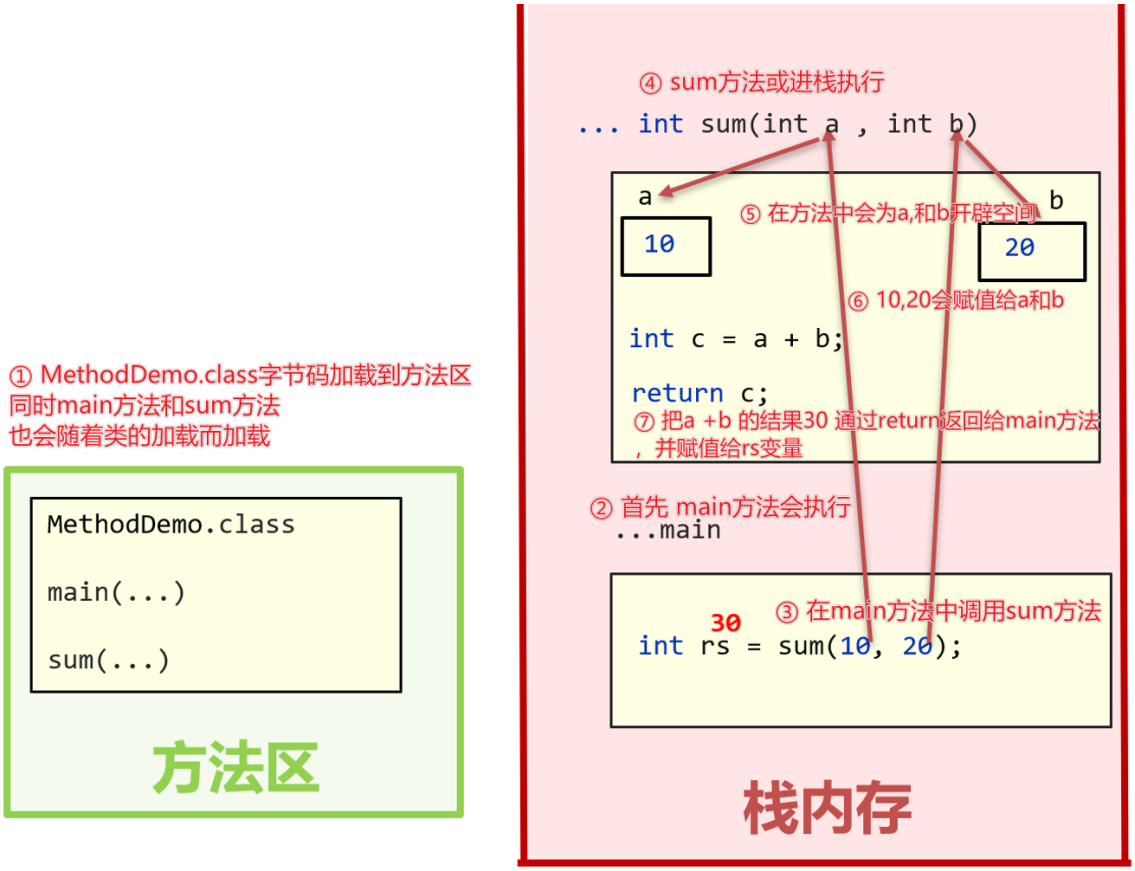


5.1 有返回值的方法，内存分析

下面我们分析一下，求两个整数和的代码，在内存中的执行原理。

```
public class MethodDemo {
    public static void main(String[] args) {
        int rs = sum(10, 20);
        System.out.println(rs);
    }
    public static int sum(int a, int b ){
        int c = a + b;
        return c;
    }
}
```

如下图所示：以上代码在内存中的执行过程，按照①②③④⑤⑥⑦的步骤执行.



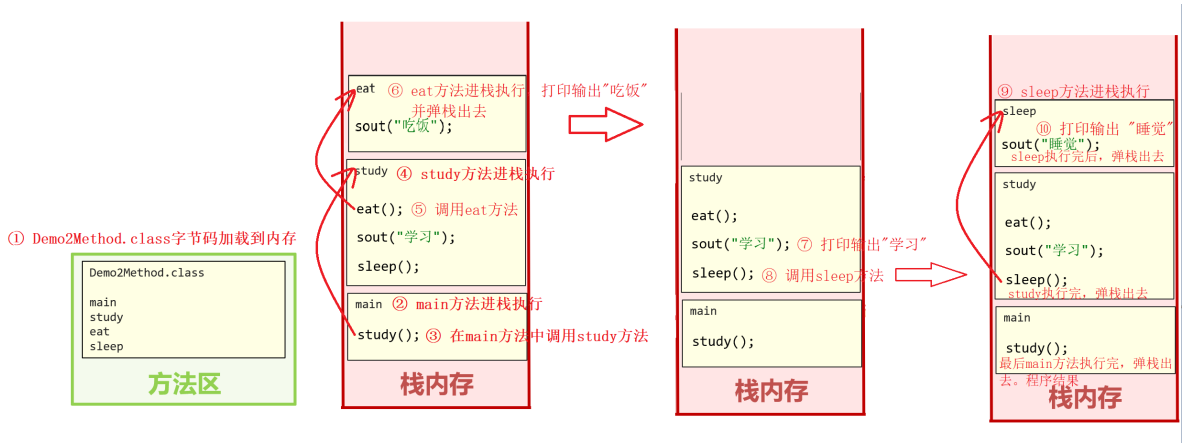
5.2 无返回值的方法，内存分析

刚才我们分析的是有有参数有返回值的方法内存原理。下面再分析一个无返回值、无参数的内存原理。

```
public class Demo2Method {
    public static void main(String[] args) {
        study();
    }

    public static void study(){
        eat();
        System.out.println("学习");
        sleep();
    }
    public static void eat(){
        System.out.println("吃饭");
    }

    public static void sleep(){
        System.out.println("睡觉");
    }
}
```



总结一下

1.方法的运行区域在哪里？
　　答：栈内存。

2.栈有什么特点？方法为什么要在栈中运行自己？
　　答：先进后出。保证一个方法调用完另一个方法后，可以回来继续执行。

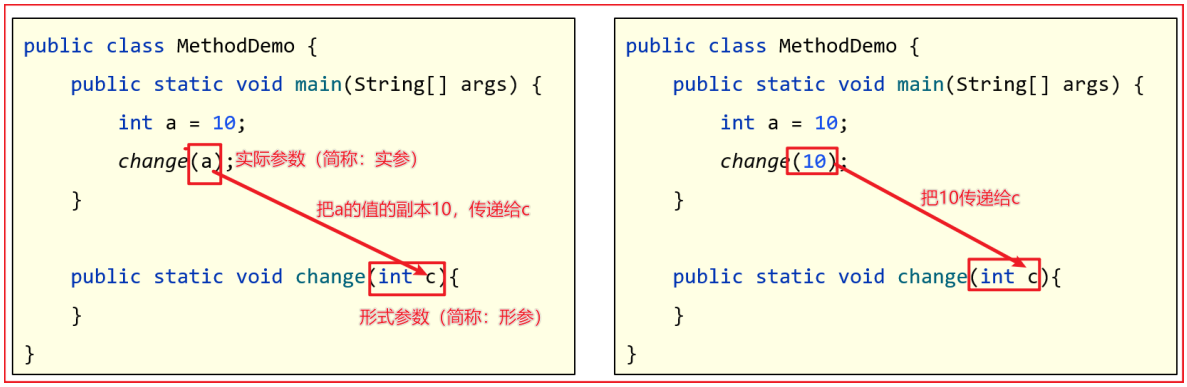
六、方法参数的传递机制

各位同学，刚才我们学习了方法运行的原理，相信大家对方法的运行过程有更加深刻的认识。但是方法参数的传递过程还需要，还需要进一步学习一下。

因为我们刚才演示的一些方法中传递的参数都是基本类型，实际上参数还可以是传递引用类型。接下来，学习一下当参数是基本类型时、和参数是引用类型时的区别。

先记住一个结论：**Java的参数传递机制都是：值传递**

所谓值传递：指的是在传递实参给方法的形参的时候，传递的是实参变量中存储的值的副本。 同学们肯定想知道，形参是什么？实参又是什么呢？ 请看下面这个张图

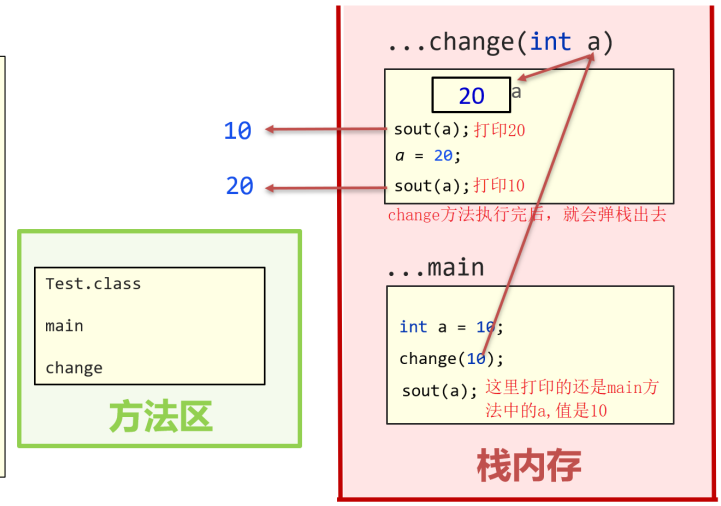


6.1 参数传递的基本类型数据

接下来，看一下方法参数传递是基本类型数据时，内存中是怎么执行的。

```java
public class Test {
    public static void main(String[] args) {
        int a = 10;
        change(a);
        System.out.println(a); // 10
    }

    public static void change(int a){
        System.out.println(a); // 10
        a = 20;
        System.out.println(a); // 20
    }
}
```



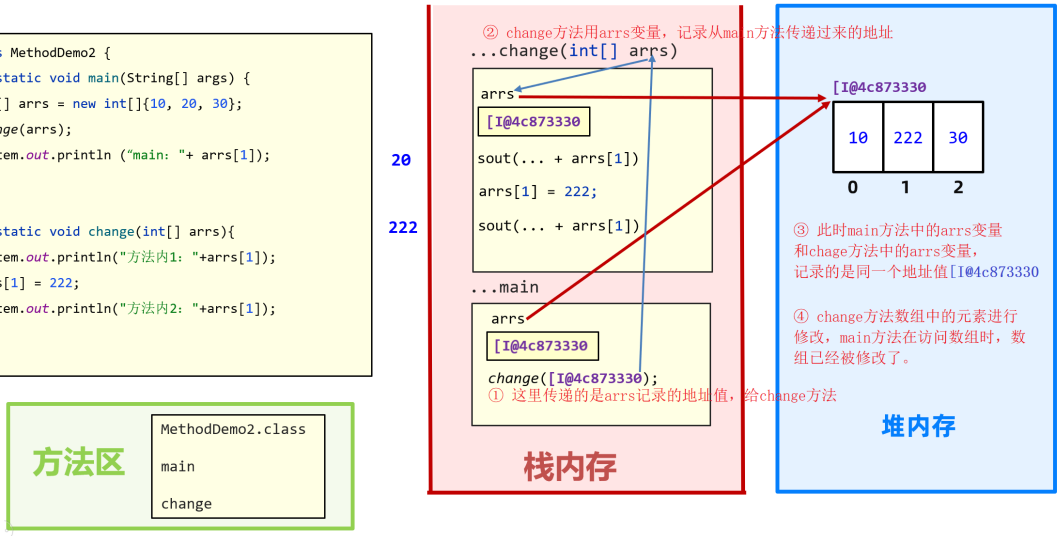
我们把参数传递的结论再复习一下：**Java的参数传递机制都是：值传递，传递的是实参存储的值的副本。**

6.3 参数传递的是引用数据类型

接下来，看一下方法的参数是引用类型的数据时，内存中是怎么执行的。

```java
public class MethodDemo2 {
    public static void main(String[] args) {
        int[] arrs = new int[]{10, 20, 30};
        change(arrs);
        System.out.println ("main: "+ arrs[1]);
    }

    public static void change(int[] arrs){
        System.out.println("方法内1: "+arrs[1]);
        arrs[1] = 222;
        System.out.println("方法内2: "+arrs[1]);
    }
}
```



我们发现调用change方法时参数是引用类型，**实际上也是值传递，只不过参数传递存储的地址值。**此时change方法和main方法中两个方法中各自有一个变量arrs，这两个变量记录的是同一个地址值[I@4c873330，change方法把数组中的元素改了，main方法在访问时，元素已经被修改了。

总结一下：

1. 基本类型和引用类型的参数在传递的时候有什么不同？
    - 都是值传递
    - 基本类型的参数传递存储的数据值。
    - 引用类型的参数传递存储的地址值。

七、方法参数传递案例

7.1 方法参数传递案例1

需求：输出一个int类型的数组内容，要求输出格式为：[11, 22, 33, 44, 55]。

分析：

1.方法是否需要接收数据进行处理？

方法要打印int类型数组中的元素，打印哪一个数组需求并不明确；

所以可以把int数组写成参数，让调用者指定

2.方法是否需要返回数据？

方法最终的目的知识打印数组中的元素。

不需要给调用者返回什么，所以不需要返回值，返回值类型写void

3.方法内部的业务：遍历数组，并输出相应的内容

代码如下

```
public class MethodTest3 {
    public static void main(String[] args) {
        // 目标：完成打印int类型的数组内容。
        int[] arr = {10, 30, 50, 70};
        printArray(arr);

        int[] arr2 = null;
        printArray(arr2);

        int[] arr3 = {};
        printArray(arr3);
    }

    /*
        参数：int[] arr表示要被打印元素的数组，需要调用者传递
    */
    public static void printArray(int[] arr){
        if(arr == null){
            System.out.println(arr); // null
            return; // 跳出当前方法
        }

        System.out.print("[");
        // 直接遍历接到的数组元素
        for (int i = 0; i < arr.length; i++) {
            if(i == arr.length - 1){
                System.out.print(arr[i]);
            }else {
                System.out.print(arr[i] + ", ");
            }
        }
        System.out.println("]");
    }
}
```

7.2 方法参数传递案例2

需求：比较两个int类型的数组是否一样，返回true或者false

分析：

1.方法是否需要接收数据进行处理？

因为，方法中需要两个int数组比较，但是需求并不明确是哪两个数组；

所以，需要接收两个int类型的数组，形参声明为：int[] arr1，int[] arr2

2.方法是否需要返回数据？

因为,方法最终的结果需要true或者false;

所以，返回值类型是boolean

3. 方法内部的业务：判断两个数组内容是否一样。

代码如下

```
public class MethodTest4 {
    public static void main(String[] args) {
        // 目标：完成判断两个int类型的数组是否一样。
        int[] arr1 = {10, 20, 30};
        int[] arr2 = {10, 20, 30};
        System.out.println(equals(arr1, arr2));
    }

    /*
        参数：
            int[] arr1，参与比较的第一个int数组
            int[] arr2  参与比较的第二个int数组
        返回值：
            返回比较的结果true或者false
    */
    public static boolean equals(int[] arr1, int[] arr2){
        // 1、判断arr1和arr2是否都是null.
        if(arr1 == null && arr2 == null){
            return true; // 相等的
        }

        // 2、判断arr1是null，或者arr2是null.
        if(arr1 == null || arr2 == null) {
            return false; // 不相等
        }

        // 3、判断2个数组的长度是否一样，如果长度不一样，直接返回false.
        if(arr1.length != arr2.length){
            return false; // 不相等
        }

        // 4、两个数组的长度是一样的，接着比较它们的内容是否一样。
        // arr1 = [10, 20, 30]
        // arr2 = [10, 20, 30]
        for (int i = 0; i < arr1.length; i++) {
            // 判断当前位置2个数组的元素是否不一样，不一样直接返回false
            if(arr1[i] != arr2[i]){
                return false; // 不相等的
            }
        }
        return true; // 两个数组是一样的。
```

```
    }
}
```

八、方法重载

接下来，我们学习一个开发中很重要的一个方法的形式——叫方法重载。

所谓方法重载指的是：一个类中，出现多个相同的方法名，但是它们的形参列表是不同的，那么这些方法就称为方法重载了。

我们在这里要能够认识，哪些是重载的方法。

下面案例中有多个test方法，但是参数列表都不一样，它们都是重载的方法。调用时只需要通过参数来区分即可。

```
public class MethodOverLoadDemo1 {
    public static void main(String[] args) {
        // 目标：认识方法重载，并掌握其应用场景。
        test();
        test(100);
    }

    public static void test(){
        System.out.println("===test1===");
    }

    public static void test(int a){
        System.out.println("===test2===" + a);
    }

    void test(double a){

    }

    void test(double a, int b){
    }

    void test(int b, double a){
    }

    int test(int a, int b){
        return a + b;
    }
}
```

我们认识了方法重载，那么方法重载有哪些应用场景呢?

一般在开发中，我们经常需要为处理一类业务，提供多种解决方案，此时用方法重载来设计是很专业的。

比如，我们现在看一个案例

```
需求：开发武器系统，功能需求如下：
    可以默认发一枚武器。
    可以指定地区发射一枚武器。
    可以指定地区发射多枚武器。
```

上面的几个需求中，不管以什么样的方式发武器，其实最终的目的都是发武器。

所以我们可以设计几个名称相同的方法，这样调用者调用起来就不用记那么多名字了

代码如下：

```java
public class MethodTest2 {
    public static void main(String[] args) {
        // 目标：掌握方法重载的应用场景。
        fire();
        fire("岛国2");
        fire("米国", 999);
    }

    public static void fire(){
        fire("岛国");
    }

    public static void fire(String country){
        fire(country, 1);
    }

    public static void fire(String country, int number){
        System.out.println("发射了" + number + "枚武器给" + country);
    }
}
```

总结一下：

```
1.什么是方法重载？
    答：一个类中，多个方法的名称相同，但它们形参列表不同。
2.方法重载需要注意什么？
    - 一个类中，只要一些方法的名称相同、形参列表不同，那么它们就是方法重载了，
      其它的都不管（如：修饰符，返回值类型是否一样都无所谓）。

    - 形参列表不同指的是：形参的个数、类型、顺序不同，不关心形参的名称。

3、方法重载有啥应用场景？
    答：开发中我们经常需要为处理一类业务，提供多种解决方案，此时用方法重载来设计是很      专业
的。
```

九、return单独使用

各位同学，关于方法的定义，我们还剩下最后一种特殊用法，就是在方法中单独使用return语句，可以用来提前结束方法的执行。

如，下面的chu方法中，当除数为0时，就提前结束方法的执行。

```
public class Test {
    public static void main(String[] args) {
        System.out.println("开始");
        chu(10 , 0);
        System.out.println("结束");
    }

    public static void chu(int a , int b){
        if(b == 0){
            System.err.println(“您的数据有误！！不执行！！”);
            return; // 直接跳出并结束当前chu方法的执行
        }
        int c = a / b;
        System.out.println("除法结果是："+c);
    }
}
```

今天的课程就到此结束了