方法

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

方法是什么

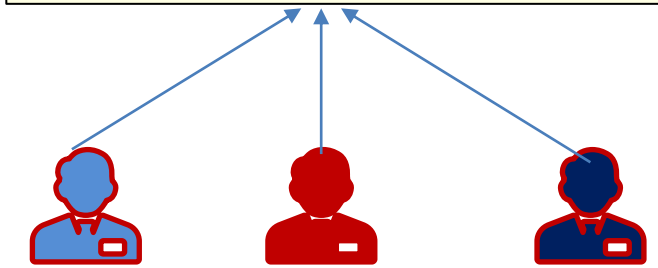
- 方法是一种语法结构，它可以把一段代码封装成一个功能，以方便重复调用。

```
public class Test {
    public static void main(String[] args) {


    }

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

使用方法的好处是？

- 提高了代码的复用性。
- 让程序的逻辑更清晰。

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



关于方法同学们需要学会什么

怎么定义方法	怎么调用方法	方法的内存图	方法的参数传递机制	方法其他常见形式、技术
方法有很多不同形式的写法，同学们需要掌握在不同的业务场景下写出合适的方法形式	方法定义出来是拿来调用的，只能调用才能让方法跑起来	方法在内存中具体是怎么去工作的	方法的参数传递原理是什么样的，需要注意什么问题(面试热点)	如方法在开发中常见的开发形式： 方法重载 方法递归

目录

Contents

- 方法定义、调用
  - 方法完整的定义形式、调用
  - 方法的其他定义形式、调用
- 方法使用的常见问题
- 方法案例
- 方法调用的内存图
- 方法的参数传递机制
- 方法的参数传递案例
- 方法重载
- 补充知识：单独使用return关键字

方法定义的完整格式

```
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;
}
```

示例：使用方法对2个整数求和并返回。

```
方法的修饰符      返回值类型  方法名称   形参列表
public static      int       add ( int a , int b ){
    int c = a + b;                          方法的执行代码
    return c;                               返回值
}
```

方法的调用格式

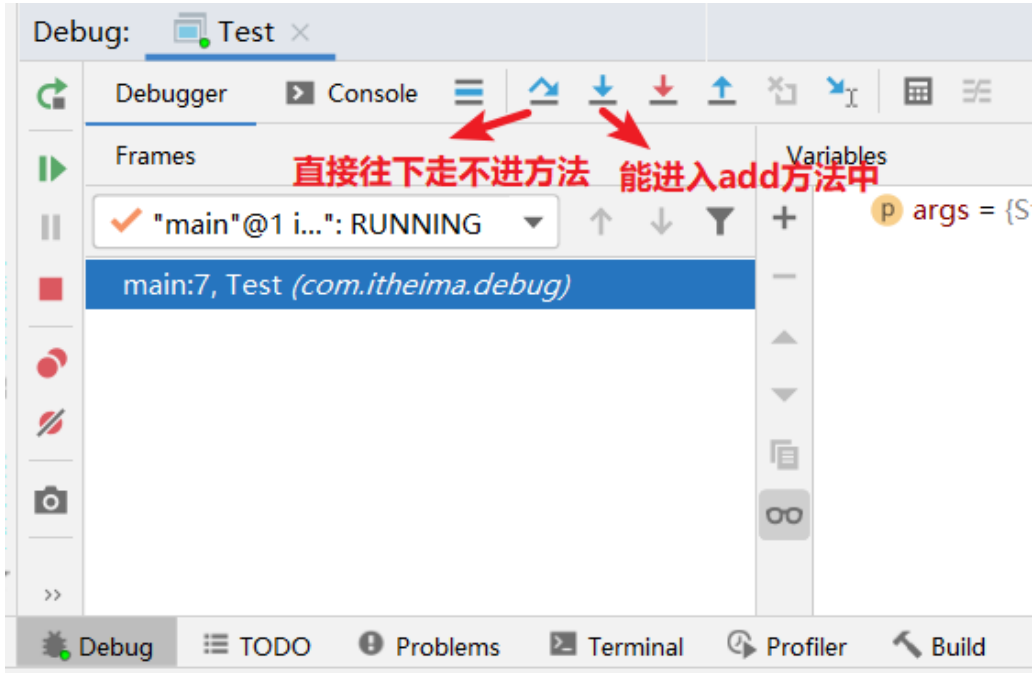
- 方法必须调用才可以跑起来，调用格式：

```
方法名(...);
```

```
int sum = add(10, 20);
System.out.println(sum);
```

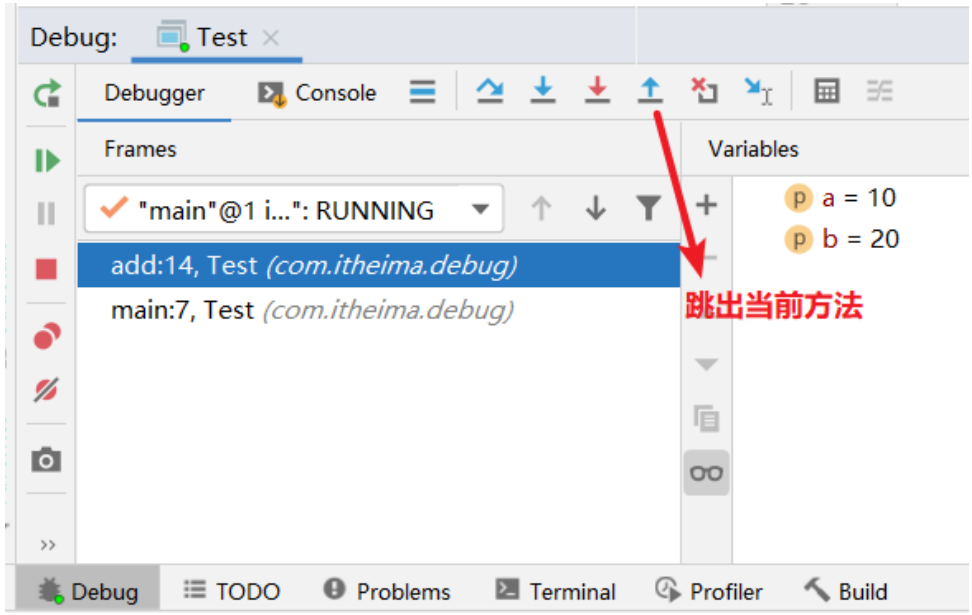
方法的调用流程 - Debug

```
public class Test {
    public static void main(String[] args) {   args: []
        System.out.println("开始");

        int sum = add( a: 10,  b: 20);
        System.out.println(sum);

        System.out.println("结束");
    }

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

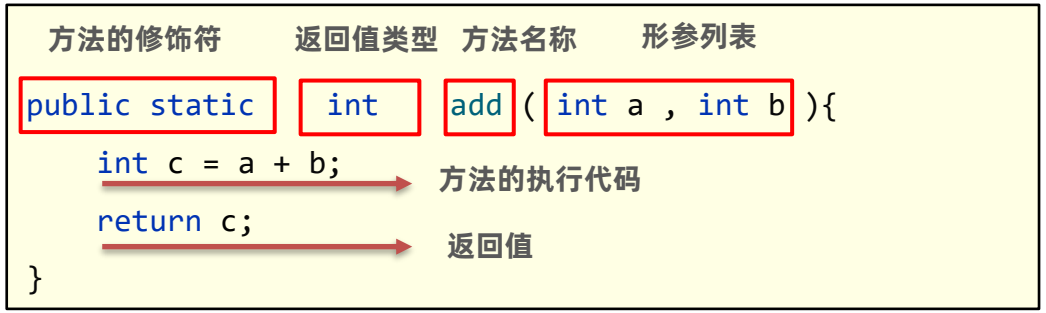


方法的调用流程 - Debug

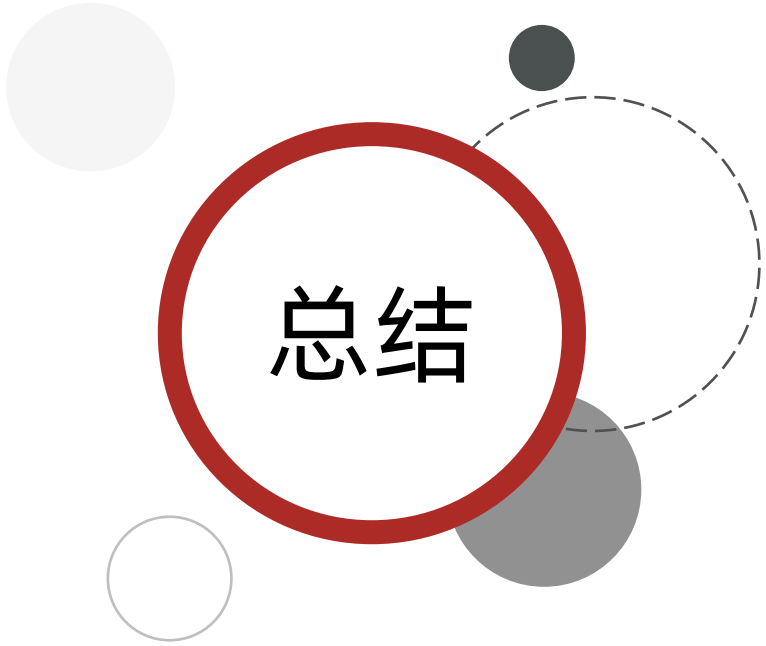
```
public class Test {
    public static void main(String[] args) {
        System.out.println("开始");

        int sum = add( a: 10,  b: 20);
        System.out.println(sum);

        System.out.println("结束");
    }

    public static int add(int a, int b){   a: 10    b: 20
        int c = a + b;   a: 10    b: 20
        return c;
    }
}
```



方法格式的注意点



- 方法的修饰符：暂时都使用public static 修饰。
- 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。
- 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．方法的完整格式是什么样的？

```
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;
}
```

2.方法要执行必须怎么办，如何进行？

- **必须进行调用；调用格式：方法名称(...)。**

目录

Contents

- 方法定义、调用
  - 方法完整的定义形式、调用
  - 方法的其他定义形式、调用
- 方法使用的常见问题
- 方法案例
- 方法调用的内存图
- 方法的参数传递机制
- 方法的参数传递案例
- 方法重载
- 补充知识：单独使用return关键字

方法的其他写法

- 方法定义时：返回值类型、形参列表可以按照需求进行填写。

```
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;
}
```

示例

```
// 打印3行Hello World（使用方法）
public static void print(){
    System.out.println("Hello World");
    System.out.println("Hello World");
    System.out.println("Hello World");
}
```

注意事项

- 如果方法不需要返回结果，返回值类型必须申明成void（无返回值），此时方法内部不可以使用return返回数据。
- 方法如果没有参数，或者返回值类型申明为void可以称为无参数、无返回值的方法，依次类推。

总结

1.如果方法不需要返回结果，不需要接收参数，应该怎么办，要注意什么？

- **方法不需要返回结果，则申明返回值类型为void；方法不需要参数，则形参列表可以不写。**
- **方法没有申明返回值类型，内部不能使用return返回数据。**
- **方法如果没有形参列表，调用的时候则不能传入参数值，否则报错。**

目录

Contents

- 方法定义、调用
- 方法使用的常见问题
- 方法案例
- 方法调用的内存图
- 方法的参数传递机制
- 方法的参数传递案例
- 方法重载
- 补充知识：单独使用return关键字

方法常见问题

- 方法的编写顺序无所谓。
- 方法与方法之间是平级关系，不能嵌套定义。
- 方法的返回值类型为void（无返回值），方法内则不能使用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。
- return语句下面，不能编写代码，因为永远执行不到，属于无效的代码。
- 方法不调用就不执行，调用时必须严格匹配方法的参数情况。
- 有返回值的方法调用时可以选择定义变量接收结果，或者直接输出调用，甚至直接调用；无返回值方法的调用只能直接调用。

目录

Contents

- 方法定义、调用
- 方法使用的常见问题
- 方法案例
  - 定义方法的技巧、计算 1- n的和返回
  - 判断整数是奇数还是偶数
  - 数组求最值案例改方法实现
- 方法调用的内存图
- 方法的参数传递机制
- 方法的参数传递案例
- 方法重载
- 补充知识：单独使用return关键字

方法定义的技巧说明

```
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;
}
```

返回值类型　　形参列表

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

- 修饰符：public static (暂时固定的)
- 方法名称：自己取名，有意义，英文小写，驼峰模式。（有意义的名字即可）
- 方法体代码：完成自己需要写的功能代码即可。（具体需求具体实现）
- 真正需要关注的就两点：1、分析方法是否需要申明返回值类型；2、分析方法是否需要接收参数。

案例

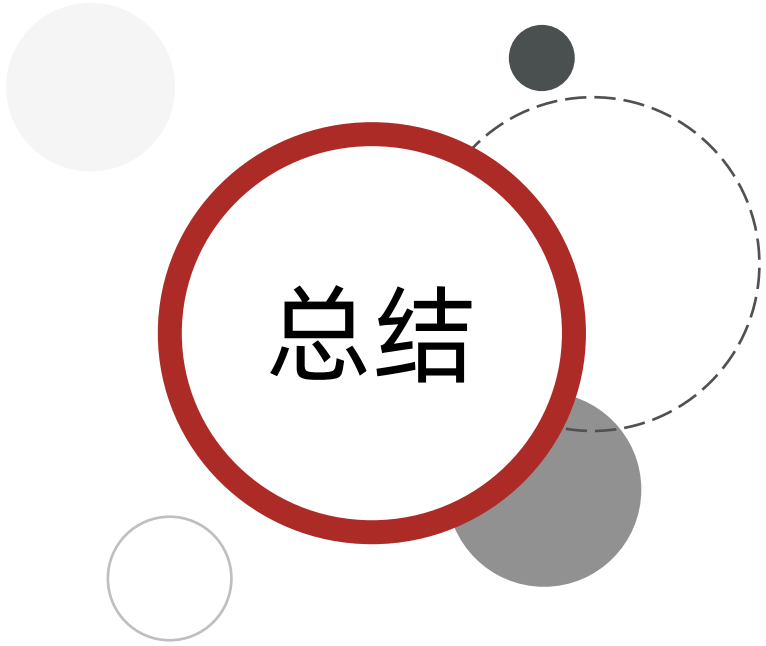
计算1-n的和返回

需求：定义一个方法，方法中计算出 1-n的和并返回。

分析：

1.根据格式编写方法 ----> 因n不固定，故方法需要声明形参接收；要返回结果，还需申明返回值类型。

2.方法内部使用 for 循环计算出 1-n 的和并返回。



1.定义方法重点关注的是哪两点？

- **方法是否需要申明返回值类型。**
- **方法是否需要定义形参列表。**

2.如何使用方法完成1-n的求和？

```
public static int sun(int n){
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}
```

目录

Contents

- 方法定义、调用
- 方法使用的常见问题
- 方法案例
  - 定义方法的技巧、计算 1- n的和返回
  - 判断整数是奇数还是偶数
  - 数组求最值案例改方法形式
- 方法调用的内存图
- 方法的参数传递机制
- 方法的参数传递案例
- 方法重载
- 补充知识：单独使用return关键字

案例

判断整数是奇数还是偶数

需求：拿一个整数，然后调用方法，把整数交给方法，在方法中输出该数为奇数还是偶数

分析：

1.根据格式编写方法 ----> 因要传入数据给方法，方法需要声明形参接收。

2.方法内部使用if语句判断，并输出对应的结论。

目录

Contents

- 方法定义、调用
- 方法使用的常见问题
- 方法案例
  - 定义方法的技巧、计算 1- n的和返回
  - 判断整数是奇数还是偶数
  - 数组求最值案例改方法形式
- 方法调用的内存图
- 方法的参数传递机制
- 方法的参数传递案例
- 方法重载
- 补充知识：单独使用return关键字

案例 数组求最值改方法实现

```
public class Test {
    public static void main(String[] args) {
        // 1、定义一个静态初始化的数组存储这些数据
        int[] faceScores = {15, 9000, 10000, 20000, 9500, -5};

        // 2、定义一个变量用于保存最大值数据
        int max = faceScores[0];
        // 3、遍历数组中的每个元素
        for (int i = 1; i < faceScores.length; i++) {
            // 4、判断这个元素值是否大于最大值变量中存储的数据，若大，则替换。
            if(faceScores[i] > max){
                max = faceScores[i];
            }
        }
        // 5、输出最大值变量即可
        System.out.println("数组的元素最大值是：" + max);
    }
}
```

需求：

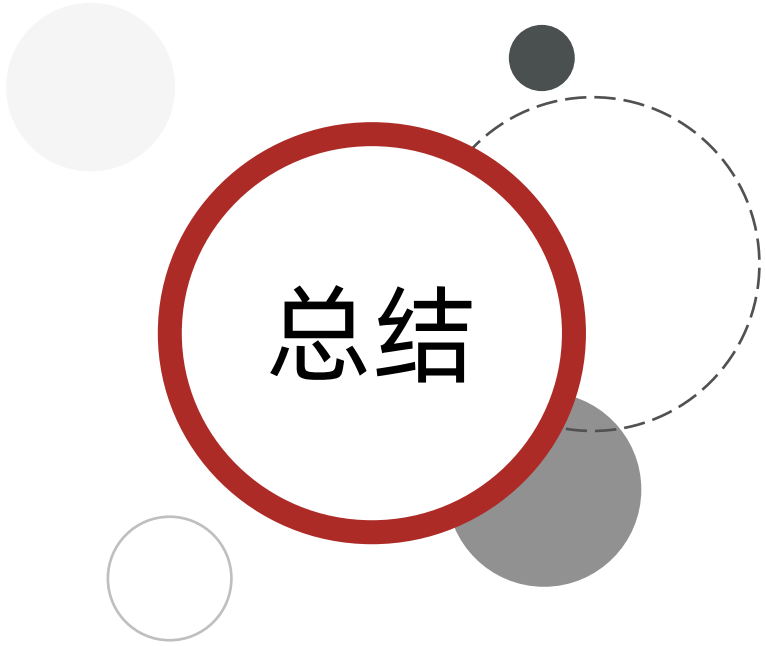
把找出数组的最大值案例，改造成方法，可以支持返回任意整型数组的最大值数据。

分析：

1.根据格式编写方法

- **要返回最大值，需要申明返回值类型。**
- **需要接收数组，需要申明形参列表。**

2．方法内部找出数组的最大值并返回。



1.如何使用方法返回一个整型数组的最大值的?

- **方法需要申明返回值类型：int**
- **方法需要定义形参列表: (int[] arr)**

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

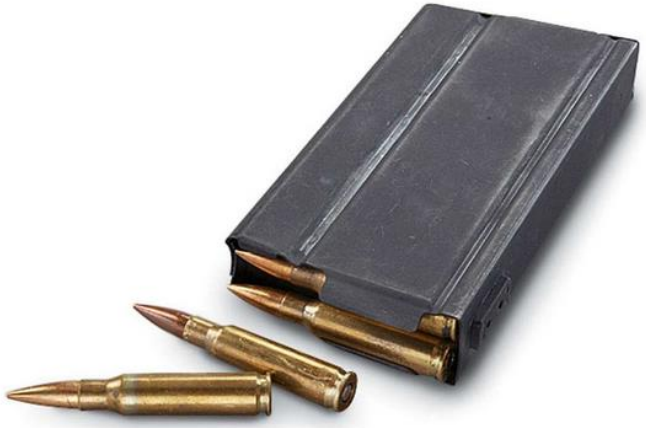
目录

Contents

- 方法定义、调用
- 方法使用的常见问题
- 方法案例
- **方法调用的内存图**
- 方法的参数传递机制
- 方法的参数传递案例
- 方法重载
- 补充知识：单独使用return关键字

方法的调用流程 – 内存图解

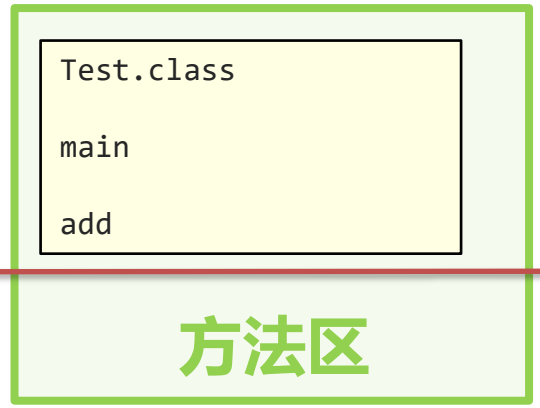
- 方法没有被调用的时候，在方法区中的字节码文件中存放
- 方法被调用的时候，需要进入到栈内存中运行

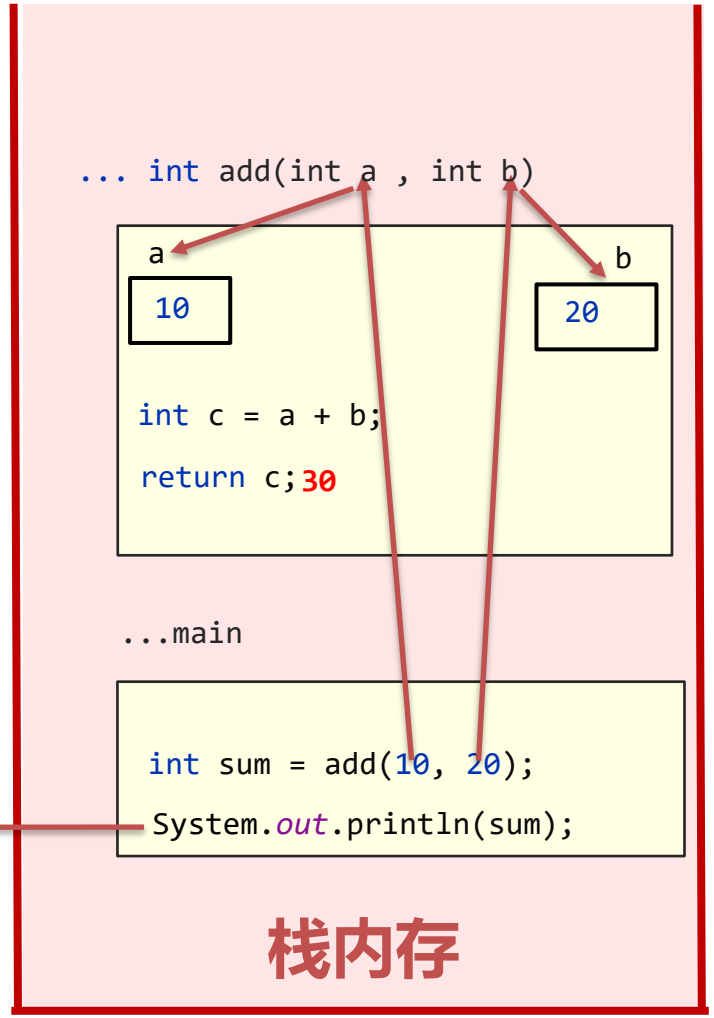


方法的调用流程 - 内存图解

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

30

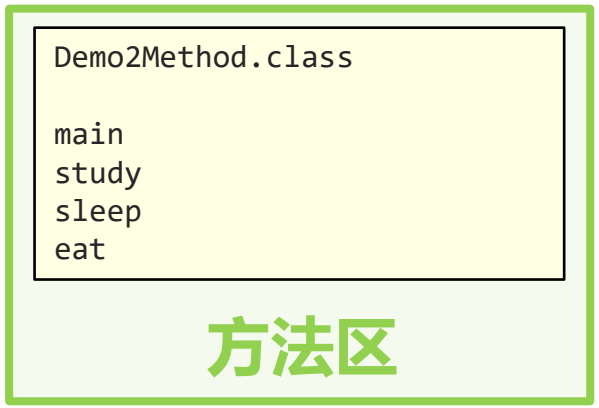


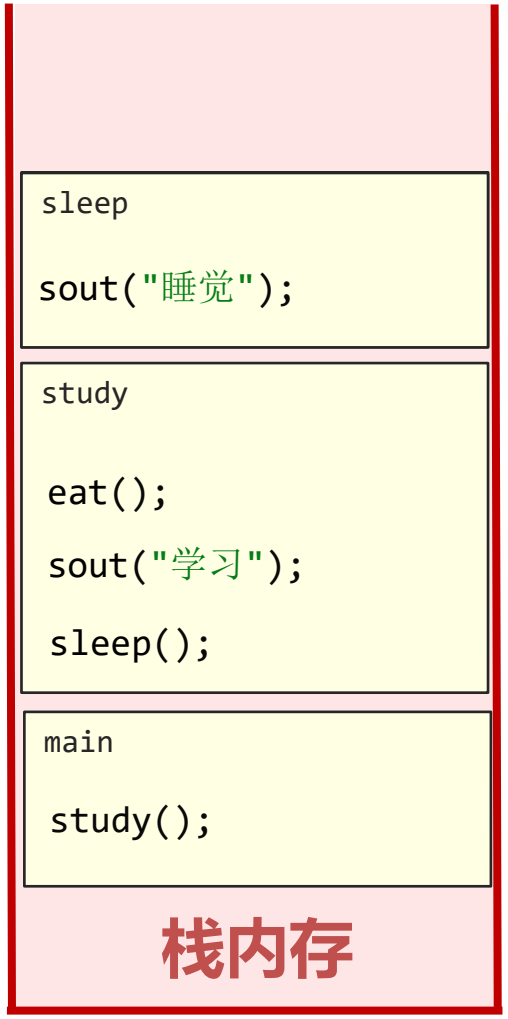


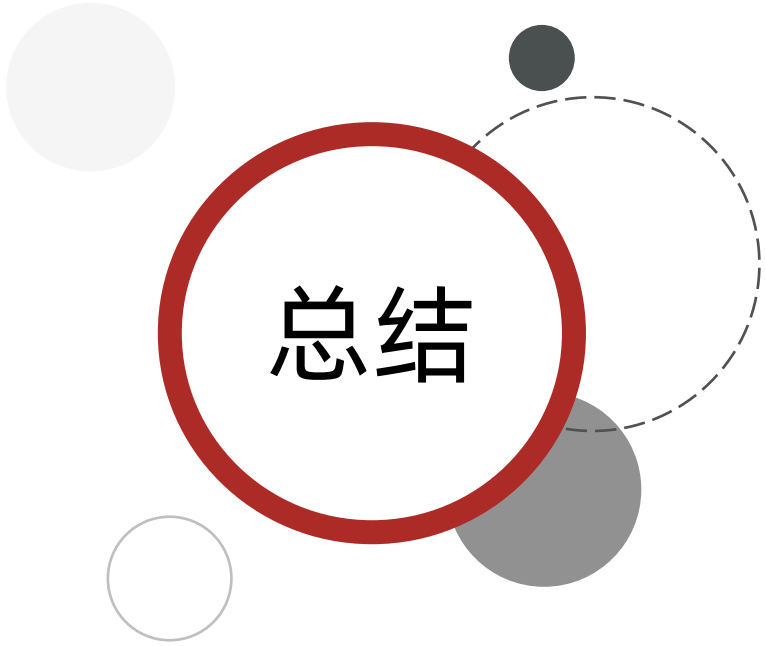
方法的调用流程 - 内存图解

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

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

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

    public static void study(){

        eat();

        System.out.println("学习");

        sleep();

    }
}
```

```
吃饭
学习
睡觉
```







1．方法的运行区域在哪里？

- **栈内存。**

目录

Contents

- 方法定义、调用
- 方法使用的常见问题
- 方法案例
- 方法调用的内存图
- 方法的参数传递机制
  - 基本类型的参数传递
  - 引用类型的参数传递
- 方法的参数传递案例
- 方法重载
- 补充知识：单独使用return关键字

Java的参数传递机制：值传递

- 在传输实参给方法的形参的时候，并不是传输实参变量本身， 而是传输实参变量中存储的值，这就是值传递。

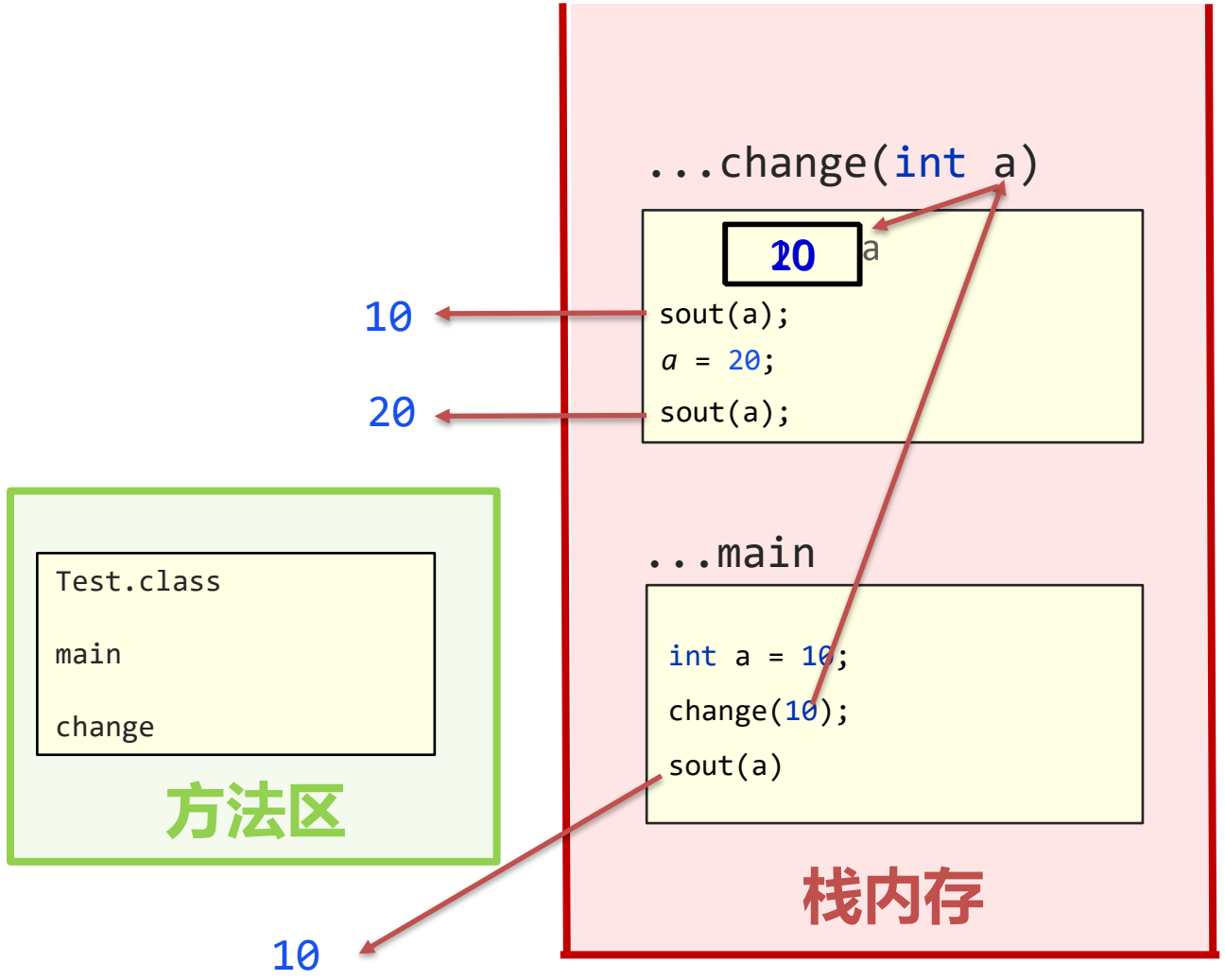
注意：

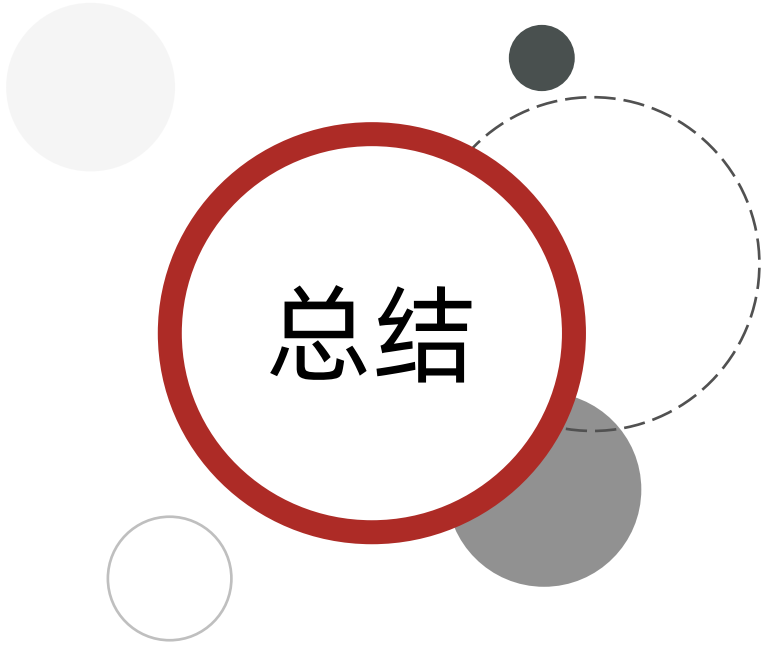
- 实参：如在方法内部定义的变量。
- 形参：如在定义方法时， “()”中所声明的参数。

```
public class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        change(a);
    }

    public static void change(int c){

    }
}
```

基本类型的参数传递

```
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
    }
}
```





1．形参和实参各指什么？

- **形参：以方法为例，就是方法定义时的变量。**
- **实参：在方法内部定义的变量。**

2．Java的参数传递机制是什么样的？

- **值传递，传输的是实参存储的值。**

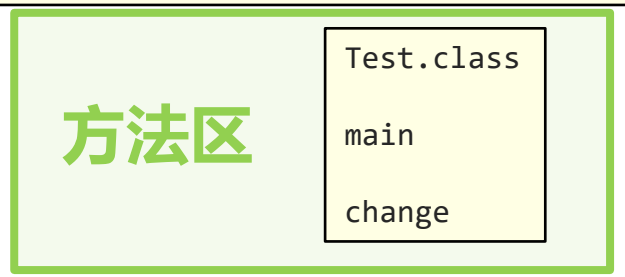
目录

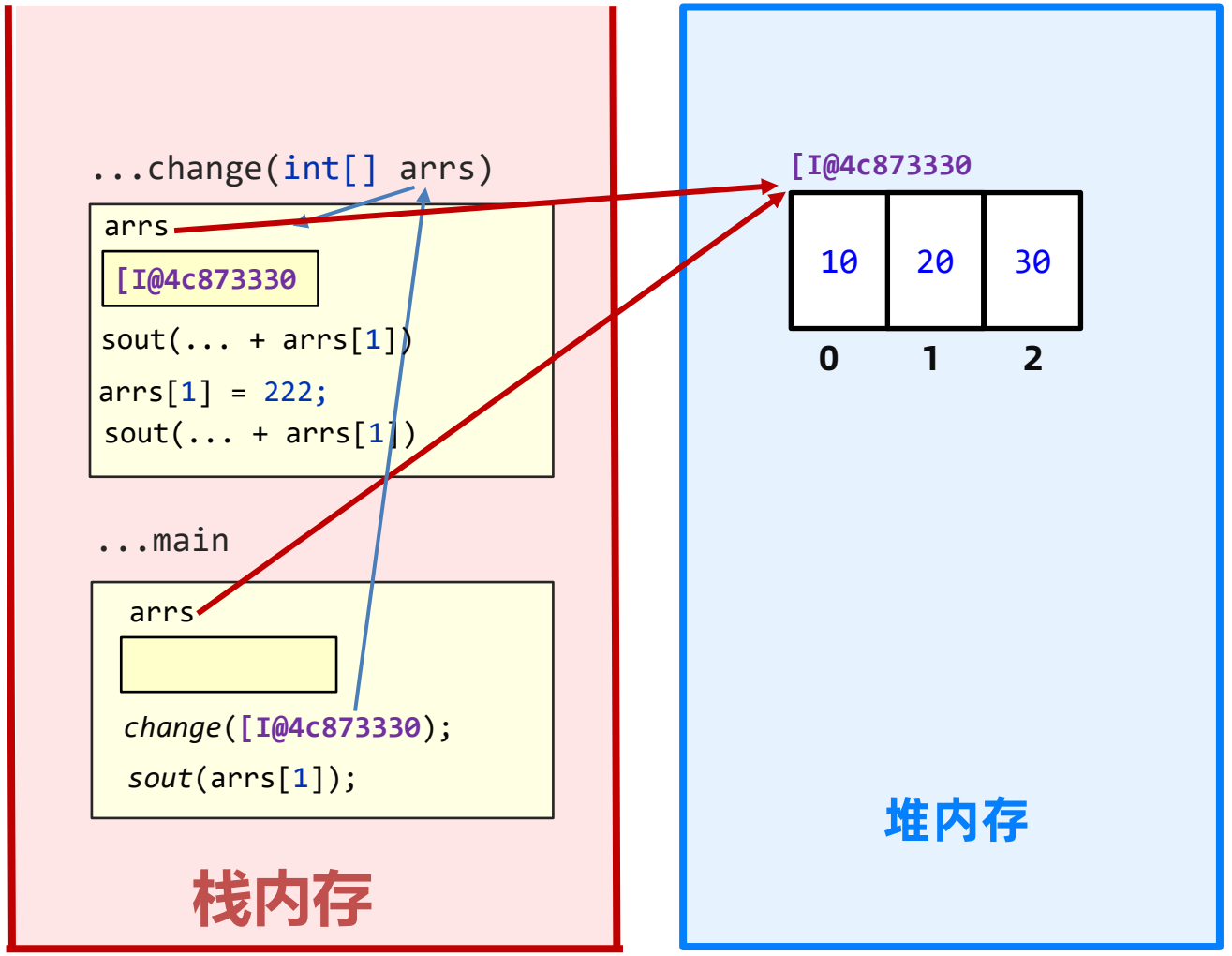
Contents

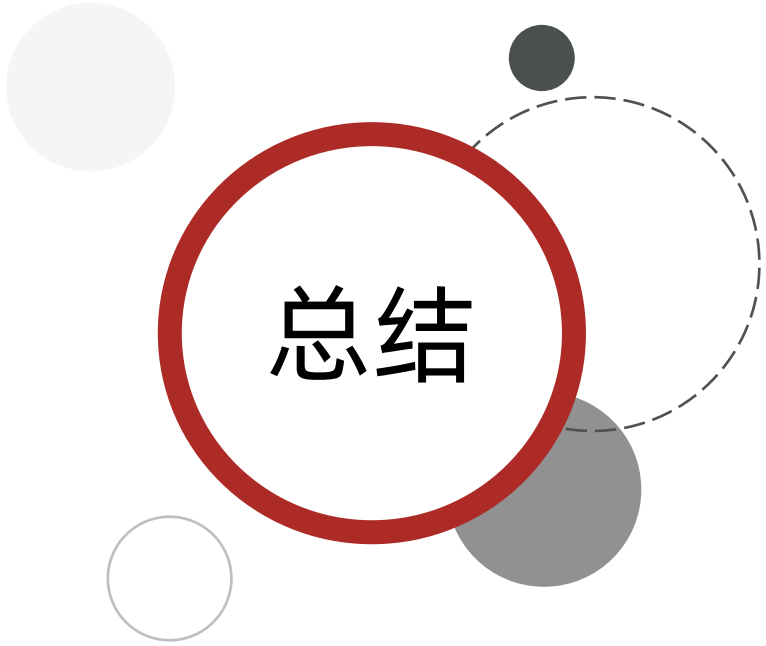
- 方法定义、调用
- 方法使用的常见问题
- 方法案例
- 方法调用的内存图
- 方法的参数传递机制
  - 基本类型的参数传递
  - 引用类型的参数传递
- 方法的参数传递案例
- 方法重载
- 补充知识：单独使用return关键字

引用类型的参数传递

```
public class Test {
    public static void main(String[] args) {
        int[] arrs = new int[]{10, 20, 30};
        change(arrs);
        System.out.println(arrs[1]); // 222
    }

    public static void change(int[] arrs){
        System.out.println("方法内部2: "+arrs[1]); // 20
        arrs[1] = 222;
        System.out.println("方法内部2: "+arrs[1]); // 222
    }
}
```







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。**
- **基本类型的参数传输存储的数据值。**
- **引用类型的参数传输存储的地址值。**

目录

Contents

- 方法的定义和调用
- 方法使用常见问题
- 方法练习题
- 方法调用内存图
- 方法的参数传递机制
- 方法的参数传递案例
  - 打印数组内容
  - 从数组中查询元素的索引返回
  - 比较两个数组内容是否相等
- 方法重载
- 单独使用return关键字

案例

打印整型数组内容

需求：

设计一个方法用于输出任意整型数组的内容，要求输出成如下格式：

“该数组内容为：[11，22，33，44，55]”

分析：

1、定义一个方法，要求该方法能够接收数组，并输出数组内容。 ---> 需要参数吗？需要返回值类型申明吗？

2、定义一个静态初始化的数组，调用该方法，并传入该数组。

目录

Contents

- 方法的定义和调用
- 方法使用常见问题
- 方法练习题
- 方法调用内存图
- 方法的参数传递机制
- 方法的参数传递案例
  - 打印数组内容
  - 从数组中查询元素的索引返回
  - 比较两个数组内容是否相等
- 方法重载
- 单独使用return关键字

练习

从数组中查询指定元素的索引

需求：

设计一个方法可以接收整型数组，和要查询的元素值；最终要返回元素在该数组中的索引，如果数组中不存在该元素则返回 -1。

例如：[11，22，33，44，55]

输入元素：44。返回索引3

输入元素：88。返回-1

分析：

1、定义方法，接收整型数组，查询的元素值，在方法体中完成元素查询的功能。---> 是否需要参数、返回值类型？

2、定义数组，调用该方法，并指定要搜索的元素值，得到返回的结果输出。

目录

Contents

- 方法的定义和调用
- 方法使用常见问题
- 方法练习题
- 方法调用内存图
- 方法的参数传递机制
- 方法的参数传递案例
  - 打印数组内容
  - 从数组中查询元素的索引返回
  - 比较两个数组内容是否一样
- 方法重载
- 单独使用return关键字

练习

比较2个数组是否一样

需求：

如果两个数组的类型，元素个数，元素顺序和内容是一样的我们就认为这2个数组是一模一样的。

例如：如下2个数组是一样的

```
int[] arrs = {10, 20, 30};

int[] arrs = {10, 20, 30};
```

请使用方法完成：能够判断任意两个整型数组是否一样，并返回true或者false。

分析：

1、定义方法，接收2个整型数组，---> 是否需要参数、返回值类型？

2、在方法内部完成判断的逻辑，并返回布尔结果。

目录

Contents

- 方法定义、调用
- 方法使用的常见问题
- 方法案例
- 方法调用的内存图
- 方法的参数传递机制
- 方法的参数传递案例
- 方法重载
  - 方法重载的形式、作用
  - 方法重载的识别技巧
- 补充知识：单独使用return关键字

方法重载

- 同一个类中，出现多个方法名称相同，但是形参列表是不同的，那么这些方法就是重载方法。

案例导学

- 开发武器系统，功能需求如下：

  ① 可以默认发一枚武器。

  ② 可以指定地区发射一枚武器。

  ③ 可以指定地区发射多枚武器。

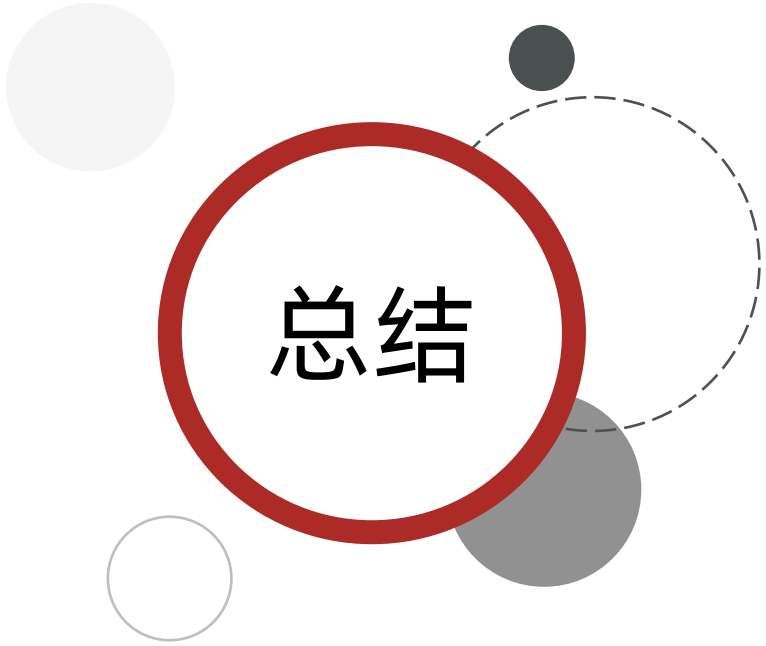
```
public class Test {
    /**（1）默认发一枚武器。*/
    public static void fire(){
            System.out.println("默认发射一枚武器给米国！");
    }
    /** （2）可以指定地区发射一枚武器。 */
    public static void fire(String location){
        System.out.println("给"+location+"发射一枚武器！");
    }
    /**（3）可以指定地区发射多枚武器。*/
    public static void fire(String location , int nums){
        System.out.println("给"+location+"发射"+nums+"枚武器！");
    }
}
```

```
fire();
fire("米国");
fire("岛国" , 1000);
```

调用方法的时候，会通过参数的不同来区分调用的是哪个方法

方法重载的作用

- 可读性好，方法名称相同提示是同一类型的功能，通过形参不同实现功能差异化的选择，这是一种专业的代码设计。



1．方法重载是什么样的？

- **同一个类中，多个方法的名称相同，形参列表不同。**

2．使用方法重载的好处 ？

- **对于相似功能的业务场景：可读性好，方法名称相同提示是同一类型的功能，通过形参不同实现功能差异化的选择，这是一种专业的代码设计。**

目录

Contents

- 方法定义、调用
- 方法使用的常见问题
- 方法案例
- 方法调用的内存图
- 方法的参数传递机制
- 方法的参数传递案例
- 方法重载
  - 方法重载的形式、作用
  - 方法重载的识别技巧
- 补充知识：单独使用return关键字

方法重载的识别技巧

- 只要是同一个类中，方法名称相同、形参列表不同，那么他们就是重载的方法，其他都不管！

  （如：修饰符，返回值类型都无所谓）

- 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。

总结

1．方法重载的关键要求是什么样的？

- **同一个类中，多个方法的名称相同，形参列表不同，其他无所谓。**

2.形参列表不同指的是什么？

- **形参的个数、类型、顺序不同。不关心形参的名称。**

目录

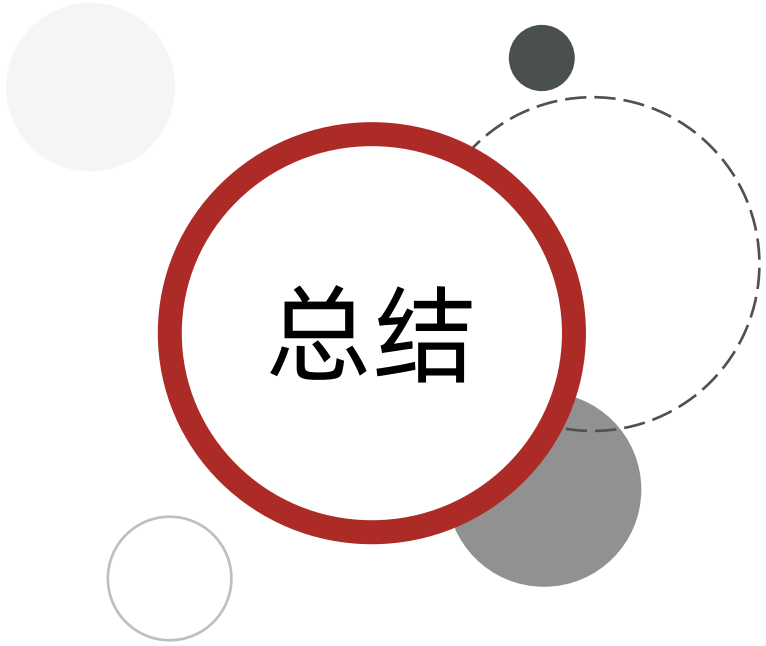
Contents

- 方法定义、调用
- 方法使用的常见问题
- 方法案例
- 方法调用的内存图
- 方法的参数传递机制
- 方法的参数传递案例
- 方法重载
- 补充知识：单独使用return关键字

return关键字单独使用

- return; ---> 可以立即跳出并结束当前方法的执行； return关键字单独使用可以放在任何方法中。

```
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; // 直接结束当前方法chu
        }
        int c = a / b;
        System.out.println("除法结果是："+c);
    }
}
```



1．如果要直接结束当前方法的执行，怎么解决？

- **return; 跳出并立即结束所在方法的执行。**
- **break; 跳出并结束当前所在循环的执行。**
- **continue; 结束当前所在循环的当次继续，进入下一次执行。**

传智教育旗下高端IT教育品牌