

09 方法

- 方法(method)是程序中最小的执行单元
- 1 什么是方法?

1. 什么是方法?

方法是程序中最小的执行单元。

2. 实际开发中，什么时候用到方法?

重复的代码、具有独立功能的代码可以抽取到方法中。

3. 实际开发中，方法有什么好处?

- 可以提高代码的复用性
- 可以提高代码的可维护性

- 2 方法的定义格式

- 2.1 最简单的方法定义与调用

格式:

```
public static void 方法名 () {  
    方法体 (就是打包起来的代码) ;  
}
```

范例:

```
public static void playGame () {  
    七个打印语句;  
}
```

格式:

```
方法名 ();
```

范例:

```
playGame ();
```

注意：方法必须先定义后调用，否则程序将报错

- 2.2 带参数的方法定义与调用

单个 参数

格 式

```
public static void 方法名 ( 参数 ) { ... }
```

范 例

```
public static void method(int number) { ... }
```

多个 参数

格 式

```
public static void 方法名 ( 参数1, 参数2, ..... )  
{ ... }
```

范 例

```
public static void getSum(int number1, int number2 )  
{ ... }
```

带参数方法 调用

单个参数	多个参数
单个参数： 方法名 (参数);	多个参数： 方法名 (参数1, 参数2,);
范例1： method(10) ;	范例1： getSum(10,20);
范例2： method(变量) ;	范例2： getSum(变量1, 变量2);

注意：方法调用时，参数的数量与类型必须与方法定义中小括号里面的变量一一对应，否则程序将报错。

• 形参和实参

■ 形参：全称形式参数，是指方法定义中的参数

■ 实参：全称实际参数，方法调用中的参数

注意：方法调用时，形参和实参必须一一对应，否则程序将报错。

• 2.3 带返回值的方法定义与调用（最完整）

方法的返回值其实就是方法运行的最终结果。

- 如果在调用处要根据方法的结果，去编写另外一段代码逻辑
- 为了在调用处拿到方法产生的结果，就需要定义带有返回值的方法

带返回值方法定义

格式

```
public static 返回值类型 方法名 (参数) {  
    方法体;  
    return 返回值;  
}
```

范例

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

• 带返回值方法的调用

01

直接调用：
方法名 (实参);






02

赋值调用：
整数类型 变量名 = 方法名 (实参);



03

输出调用：
System.out.println(方法名 (实参));

• 2.4 方法的注意点

-  方法不调用就不执行
-  方法与方法之间是平级关系，不能互相嵌套定义
-  方法的编写顺序和执行顺序无关
-  方法的返回值类型为void，表示该方法没有返回值，没有返回值的方法可以省略return语句不写。如果要编写return，后面不能跟具体的数据。
-  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

• 2.5 return关键字

-  方法没有返回值：可以省略不写。如果书写，表示**结束方法**
-  方法有返回值：必须要写。表示**结束方法**和**返回结果**

• 3 方法的重载

- 在同一个类中，定义了多个**同名的方法**，这些同名的方法具有同种的功能。
- 每个方法具有**不同的参数类型或参数个数**，这些同名的方法，就构成了重载关系
- **简单记：**同一个类中，方法名相同，参数不同的方法。与返回值无关。
参数不同：个数不同、类型不同、顺序不同

```
public class MethodDemo {
    public static void fn(int a, double b) {
        //方法体
    }
    public static void fn(double a, int b) {
        //方法体
    }
}
```



- 以上是顺序不同可以构成重载，但是不建议这样写
- **Java虚拟机通过参数的不同来区分同名的方法**
- 会把相同功能的方法名起成一样的名字
 - 好处：定义方法时不用那么多单词
 - 好处：调用方法时不用那么麻烦

• 4 return 和 break 关键字的区别

- return：和循环没有什么关系，跟方法有关。表示：结束方法和返回结果。方法执行到

return, 那么方法全部结束, 里面的循环也随之结束了。

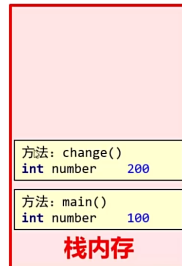
- break: 和方法没有什么关系, 和结束循环和结束switch有关。

5 方法调用的基本内存原理

- 基本数据类型

传递基本数据类型时, 传递的是真实的数据, 形参的改变, 不影响实际参数的值

```
public class ArgsDemo01 {
    public static void main(String[] args) {
        int number = 100;
        sout("调用change方法前:" + number);
        change(number);
        sout("调用change方法后:" + number);
    }
    public static void change(int number) {
        number = 200;
    }
}
```

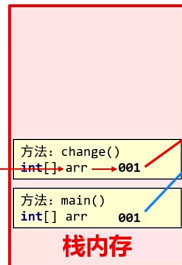


调用change方法前: 100

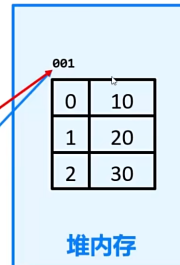
- 引用数据类型

传递引用数据类型时, 传递的是地址值, 形参的改变, 影响实际参数的值

```
public class ArgsDemo02 {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30};
        sout("调用change方法前:" + arr[1]);
        change(arr);
        sout("调用change方法后:" + arr[1]);
    }
    public static void change(int[] arr) {
        arr[1] = 200;
    }
}
```



调用change方法前: 20

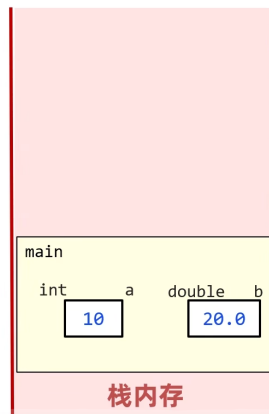


6 基本数据类型和引用数据类型

基本数据类型

```
public class Test {
    public static void main(String[] args){
        int a = 10;
        double b = 20.0;
    }
}
```

变量中存储的是真实的数据

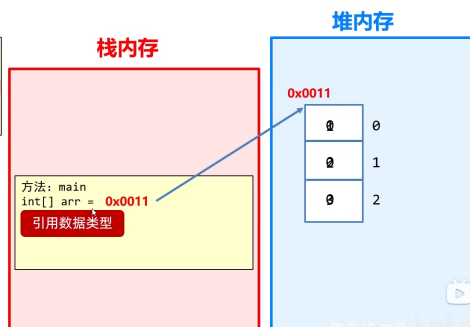


引用数据类型

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

变量中存储的是地址值

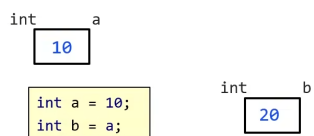
引用: 使用了其他空间中的数据



从内存的角度去解释：

基本数据类型：数据值是存储在自己的空间中

特点：赋值给其他变量，也是赋的真实的值。



引用数据类型：数据值是存储在其他空间中，自己空间中存储的是地址值。

特点：赋值给其他变量，赋的地址值。

```
int[] arr1 = {1,2,3};
int[] arr2 = arr1;
```

