### 4.1.1 方法的定义

Java中方法的定义语法如下：

|  |
| --- |
| 修饰符 返回值类型 方法名称（参数类型1 参数名1，参数类型2 参数名2，…）{  方法体;  return 返回值;  } |

* 修饰符：包含权限修饰符、静态修饰符static以及final修饰符，这些修饰符会在后面的章节讲到，暂时固定使用public static。
* 返回值类型：就是功能结果的数据类型。
* 方法名称：方法的标识符，自定义，命名规则与变量相同。
* 参数列表：指方法名称后面的()号里面内容，如果把方法当作工厂，参数就是工厂的原材料，可以定义多个参数(类比工厂的多种原材料)，每个参数都要声明数据类型及参数名，相当于变量的定义，定义好的参数（变量）将在方法内部使用，进行特定的处理。方法也可以没有参数，称为无参方法，只用来完成特定的功能，表现为()号里面是空的。
  + - 参数类型：就是参数的数据类型。
    - 参数名：就是变量名。
* 方法体：是一对{}里面的内容，就是方法要完成的功能的代码集合。
* return返回值：返回值是方法功能的最终结果，如果说参数是工厂的原材料，返回值是工厂的成品，由return带给调用者并结束方法, return下面不应该再出现语句,return 后面的返回值的数据类型必须与方法名称前面的返回值类型一致。
* 也可以没有return语句，即没有返回值，只执行特定的功能代码就结束，此时返回值类型处要写成void。
* 以上的（）中可以为空，表示这是个不带参数的方法（无参方法）。
* 一个类里面可以有多个方法，但方法跟方法之间，是平级关系 定义时不能存在嵌套（所以方法要定义在main方法外面,这时我们对类的认知就提高了一个层次，即类不但有main方法，还有自定义方法）。

**示例**：自定义一个方法，返回两个数中较大的一个：

|  |
| --- |
| **public** **static** **int** max(**int** x,**int** y){  **int** m;  **if**(x>y) {  m=x;  }**else** {  m=y;  }  **return** m;  } |

这个方法中public static是修改符，int是方法的返回值类型，max是自定义的方法名称，（int x，int y）表示这个方法定义了两个int类型的参数，return m表示该方法的返回值，注意这个返回值m的数据类型必须与方法名称前的类型相同，这里都是int类型。

在main方法中调用自定义方法：

|  |
| --- |
| **public** **static** **void** main(String[] args) {  System.***out***.println(*max*(5, 6));//输出5,6之间的最大值  System.***out***.println(*max*(8, 18));//输出8,18之间的最大值  } |

运行结果:

|  |
| --- |
| 6  18 |

### 4.1.2 方法的调用

已经定义的方法不调用不执行，通过对方法的调用来执行方法中的方法体。

方法调用过程中要分清楚形式参数与实际参数，方法定义时的参数称为形式参数（简称形参），方法调用时实际传入的值称为实际参数（简称实参），形参前面要有数据类型，实参前面无须数据类型。如上面示例中定义的方法max(int x,int y)里面的int x和int y就是形参，调用时max(5,6)中的5和6就是实参。

方法调用规则：如果是无参方法，直接调用方法的名字加()号即可；如果是有参方法，还须填写实参，且实参的参数列表(即参数的个数，类型，顺序)必须和形参一致，否则无法编译通过。

**示例**：定义一个带参的方法（注意在main方法以外）

|  |
| --- |
| public static int add(int m,int n){ // m,n是形参  int result= m+n;  return result;  } |

在main方法中调用：

|  |
| --- |
| int sum=add(100,50); //100,50是实参  System.***out***.print(sum); |

调用过程：

(1)main方法中调用add方法，把实参100，50分别传递给形参m，n；赋值m=100,n=50；

(2)执行方法语句int result= m+n;把m和n的实际值进行相加得到结果150 赋值给result；(3)执行return语句，将result的结果返回；

(4)最后将方法的结果result赋值给sum。

如图4.1所示，留意实参与形参的值传递。

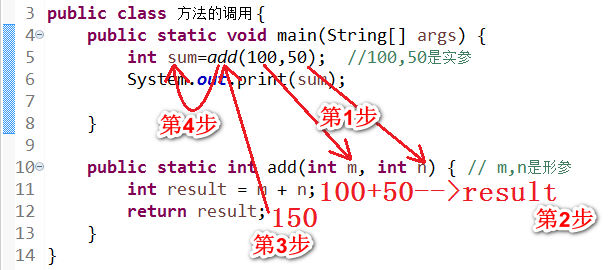


图4.1 方法调用图示

方法的调用形式：

(1)直接调用，用方法名(实参列表)单独构成一条语句。适用于方法中本身有输出语句的情况。

(2)输出调用，用System.out.println(方法名(实参列表))这种基本格式，输出方法的返回值。只适用于方法有返回值的情况。

(3)赋值调用，用变量=方法名(实参列表)这种格式，将方法的返回值赋值给一个变量。

(4)运算调用，将有返回值的方法当作一个操作数参与算术运算，比较运算，逻辑运算等，赋值调用也可归于此类。

另外，不但main方法可以调用普通方法，普通方法之间也可互相调用。

**示例**：方法的调用。

|  |
| --- |
| **public** **static** **void** main(String[] args) {  *sum1*(10, 20); // 1.直接调用;  System.***out***.println(*sum3*(10, 20, 30));//2输出调用  **int** sum = *sum2*(20, 30);// 3.赋值调用  System.***out***.println("sum=" + sum);  **int** total=*sum2*(10, 20)+*sum3*(20,30,40);//4.运算调用  System.***out***.println("total=" + total);  }  // 方法：求两个数的和,无返回值  **public** **static** **void** sum1(**int** num1, **int** num2) {  System.***out***.println("两个数的和是" + (num1 + num2));  }  // 方法：求两个数的和,有返回值  **public** **static** **int** sum2(**int** num1, **int** num2) {  **return** num1 + num2;  }  // 方法：求三个数的和,有返回值  **public** **static** **int** sum3(**int** num1, **int** num2, **int** num3) {// 方法之间也能互相调用  **return** *sum2*(num1, num2) + num3; // 这里调用了sum2方法  } |

运行结果：

|  |
| --- |
| 两个数的和是30  60  sum=50  total=120 |

**示例**：使用方法，输出1-100之间的所有素数。

**分析**：这需要创建一个方法，能判断每一个传递进来的参数是否是素数，是就返回true，否则返回false。然后对1-100进行循环，对每一个迭代变量都调用方法进行判断，如果是true就输出。

|  |
| --- |
| **public** **static** **void** main(String[] args) {// 主方法  **for** (**int** i = 1; i <= 100; i++) {  **if** (*sushu*(i) == **true**) {  System.***out***.print(i + ",");  }  }  }  **public** **static** **boolean** sushu(**int** num) {  **boolean** flag = **true**;  **for** (**int** i = 2; i <= num - 1; i++) {  **if** (num % i == 0) {  flag = **false**;  **break**;  }  }  **return** flag;  } |

运行结果：

|  |
| --- |
| 1,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97, |

**示例**：设计一个方法，输入圆柱形的半径，高度，输出体积。

|  |
| --- |
| **public** **static** **void** main(String[] args) {// 主方法  Scanner sc=**new** Scanner(System.***in***);  System.***out***.print("请输入圆柱体的半径：");  **double** r=sc.nextDouble();  System.***out***.print("请输入圆柱体的高度：");  **int** h=sc.nextInt();  System.***out***.println("该圆柱体的体积是:"+*capacity*(r, h));  }  // 定义一个方法，输入圆柱形的半径，高度，输出体积  **public** **static** **double** capacity(**double** radius, **int** height) {  **return** 3.14159\*radius\*radius\*height;  } |

运行测试：

|  |
| --- |
| 请输入圆柱体的半径：10  请输入圆柱体的高度：10  该圆柱体的体积是:3141.59 |

### 4.1.3 方法的重载

如果想定义方法求两个数字的最大值，但这个两个数字可能都是整数，也可能都是浮点数，也可能既有整数又有浮点数，按我们前面学习的知识，不得不针对这么多种状况定义多个不同名称，不同参数类型的方法来解决，如定义以下四个方法求最大值：max1(int x,int y)、max2(int x,double y)、max3(double x,int y)、max4(double x,double y)，调用时分别根据实参的情况调用不同的方法。这样，同是求最大值，但有多个不同名称的方法，调用时很难分清楚哪种情况该去调用哪个方法，极易出错。我们需要这样的机制：如果多个不同的方法完成相似的功能，最好方法名称可以相同，只是参数列表不同。为了解决解决这个问题，Java引入了方法的重载。

所谓方法的重载指的是在一个类中可以有多个名字相同的方法，但这些方法要么参数的类型不同，要么参数的个数不同，要么参数的顺序不同，否则就不是重载，而是重复，程序会报错的。简单来说，方法重载就是方法名称相同，但参数列表不同。需要注意的是重载与返回值无关，就是说两个方法如果参数的类型，个数及顺序相同，即使返回值类型不同，它们也不是重载方法，而被视为重复，是不允许的。例如：

|  |
| --- |
| int fun(int a,int b);  double fun(int a,int b); |

这两个方法不是重载方法，而是重复，程序报错。可以这样理解：假设你调用fun(2,3),那么编译器无法识别要调用哪一个方法。

Java根据实际参数的参数列表（参数的个数，类型，顺序）来判定调用同名方法中的哪一个，可以通过向重载方法传递不同的参数来选择想要调用的方法。

**示例**：定义多个求最大值的同名不同参（重载）方法max。

|  |
| --- |
| **static** **int** max(**int** a, **int** b) {  **return** a > b ? a : b;  }  **static** **double** max(**double** a, **int** b) {  **return** a > b ? a : b;  }  **static** **int** max(**int** a, **int** b, **int** c) {  **int** x = *max*(a, b);  **return** *max*(x, c);  } |

调用重载方法：

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **int** max1 = *max*(2, 3); // 结果为3，显然调用了 int max(int a,int b)这个方法  **double** max2 = *max*(3.1, 4); // 结果为4.0，显然调用了 int max(double a,int b)这个方法  **int** max3 = *max*(5, 8, 12); // 结果为12，显然调用了 int max(int a,int b,int c)这个方法  System.***out***.println("max1:"+max1+",max2:"+max2+",max3:"+max3);  } |

运行结果：

|  |
| --- |
| max1:3,max2:4.0,max3:12 |

### 4.1.4 方法的递归

一个方法在方法体内调用自身被称为递归。执行递归方法会不断调用自身，每调用一次进入新的一层。

**示例**：下面的方法就是一个递归方法：

|  |
| --- |
| public static void fun(int a){  int b=a+1;  fun(b);  } |

这就是一个递归方法，该方法会不断地调用自身，使得程序的运行无法中止。由此可见，必须在方法体中设置判断条件，使得程序在满足判断条件后就不再调用自身，然后再逐层返回。

下面举一个计算n的阶乘n!的例子来说明递归的执行过程。

|  |
| --- |
| public static int f(int n){  if(n>1)  return n\*f(n-1);  else if(n==1 || n==0)  return 1;  else  return -1;  } |

输入4来分析程序的执行过程：f(4)返回4\*f(3)，于是调用f(3)。f(3)返回3\*f(2)，于是调f(2)，f(2)返回2\*f(1)，于是调用f(1)，f(1)返回1，于是层层返回，最后f(4)返回的值为4\*3\*2\*1,即4！。

上一章的斐波那契数列也可以用递归来实现。注：斐波那契数列指的是这样一个数列 1, 1, 2, 3, 5, 8, 13, 21,… 这个数列前两项都为1，从第3项开始，每一项都等于前两项之和。使用递归输出斐波那契数列前20项代码如下。

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **for**(**int** i=1;i<=20;i++) {  System.***out***.print(*f*(i)+"\t");  **if**(i%5==0) {  System.***out***.println("");  }  }  }  **public** **static** **int** f(**int** n) {  **if** (n == 1 || n == 2) {  **return** 1;  } **else** {  **return** *f*(n - 1) + *f*(n - 2);  }  } |

**示例**：使用递归实现计算1-100之间的和.

|  |
| --- |
| **Public** **static** **void** main(String[] args) {  System.***out***.println(“1-100的和是:”+*sum*(100));  }  **public** **static** **int** sum(**int** n) {  **if**(n==1) {  **return** 1;  }**else** {  **return** n+*sum*(n-1);  }  } |