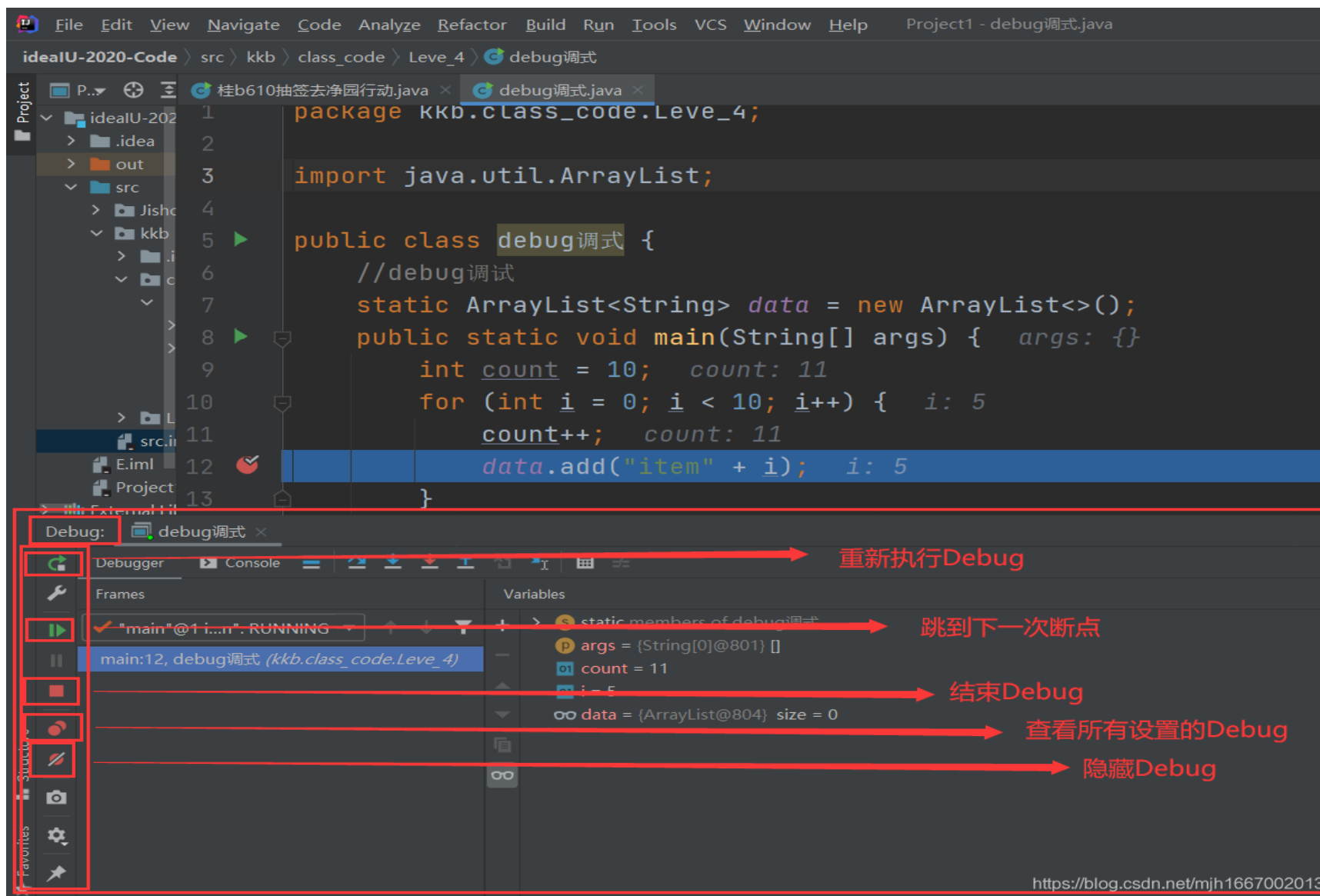


# 第2章 Java编程基础






老师：李沁洳



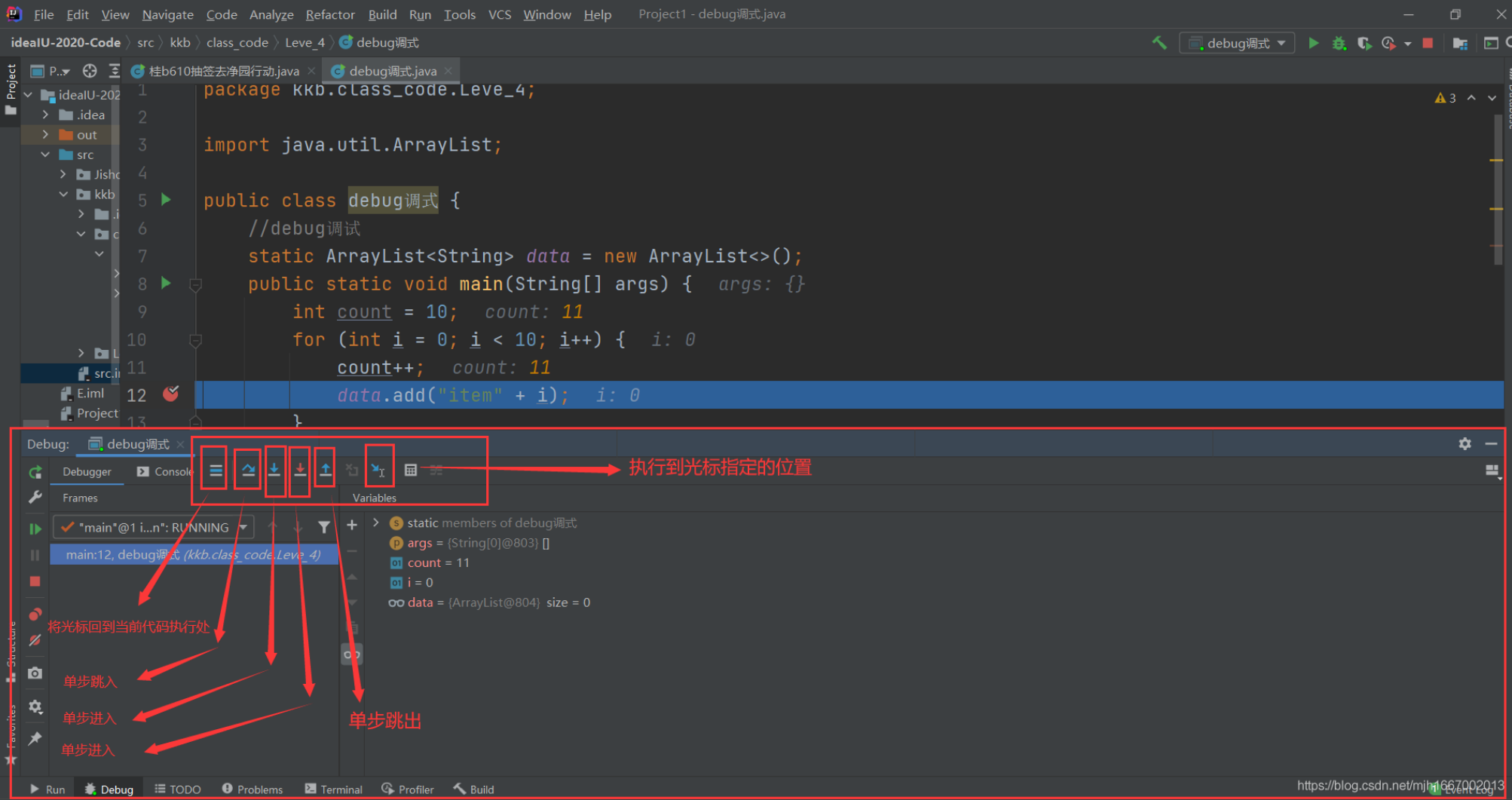
# 断点的使用









# 断点的使用

-  有返回箭头的按钮，功能是重新执行Debug，当你在执行Debug一半时，发行并不能解决你的问题，这时你不需要重新关闭并打开Debug，按下此按钮，Debug调试会重新执行。
-  一个竖杠加向右的三角形的按钮，功能是跳到下一次断点执行，两个断点之间的代码都被加载执行过了。但是当一个断点在一个for循环中，如果循环有n ( $n \geq 1$ ) 次，for循环外有一个断点，此时该功能是跳过一次for循环，并不是跳过所有循环而直接到下一个断点，那么如果先要跳过所有循环可以将断点暂时不启用，也就是将enabled去掉，因为enable是启用断点的意思。
-  一个红色的正方块的按钮，功能是结束Debug 的执行。按下之后，整个Debug调试都会将结束并停止执行。
-  两个重叠的红色圆圈的按钮，功能是查看所有的断点。快捷键是 (shift + ctrl + F8) ，至于它的用法在上面将给断点添加条件时已讲述。
-  一个红色的圆圈中有一个灰色的斜杠的按钮，功能是隐藏所有的Debug断点。用途就是，当你在Debug调试时，你觉得此时已经没有问题了，这时可以暂时隐藏所有Debug断点，无障碍运行一次，如果有问题还要取消隐藏，不至于重新打开Debug。

# 断点的使用



# 断点的使用

-  功能是将你的光标移动到当前代码所执行处，不管此刻你的鼠标光标的位置在哪里。
-  功能是单步跳入。Debug调试是一行一行的执行下去，但是如果遇到调用方法时，是不会进入方法里面的。
-  按钮 与  按钮，因为功能相似，放在一起讲。两个的功能都是单步进入的执行，但唯一的区别是，蓝色向下箭头的按钮遇到方法时，只有当经过的方法时用户自定义的方法才会跳进去，如果是系统自定的方法则不会跳进去。而红色向下箭头的按钮，不管是系统自定义的方法还是用户自定义的方法都会跳入到方法里面去执行。
-  功能是单步跳出。可以跳出进入的方法。
-  功能是将当前执行的位置直接执行到用户指定鼠标光标的位置。

# /2.5

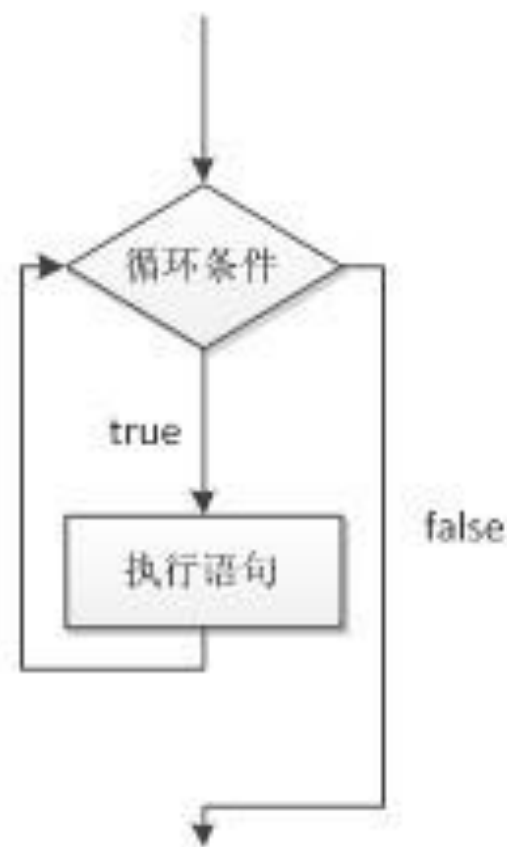
## 循环结构语句



## 2.5.1 While循环语句

```
while (循环条件) {  
    执行语句  
    .....  
}
```

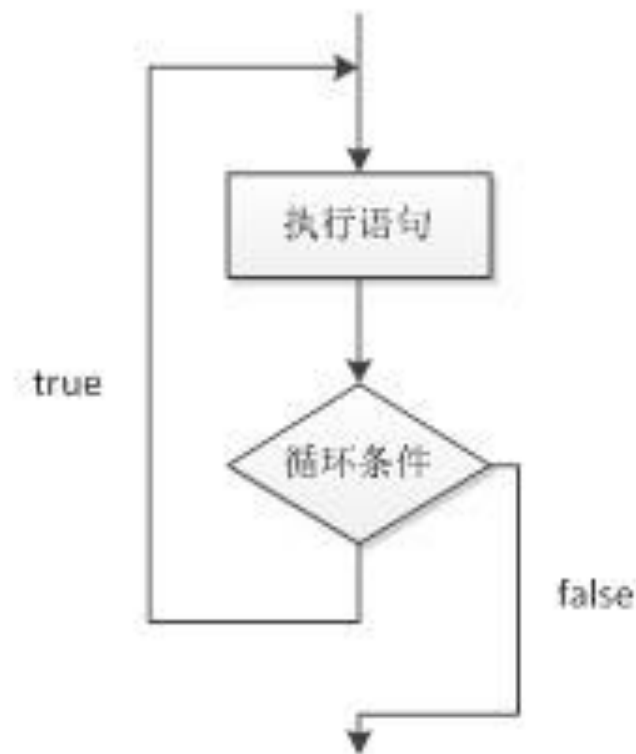
```
int x = 1;  
while (x <= 4) {  
    System.out.println("x = " + x);  
    x++;  
}
```



## 2.5.2 do...while语句

```
do {  
    执行语句  
    .....  
} while(循环条件);
```

```
int x=1;  
do {  
    System.out.println("x = " + x);  
    x++;  
}while(x>4);
```





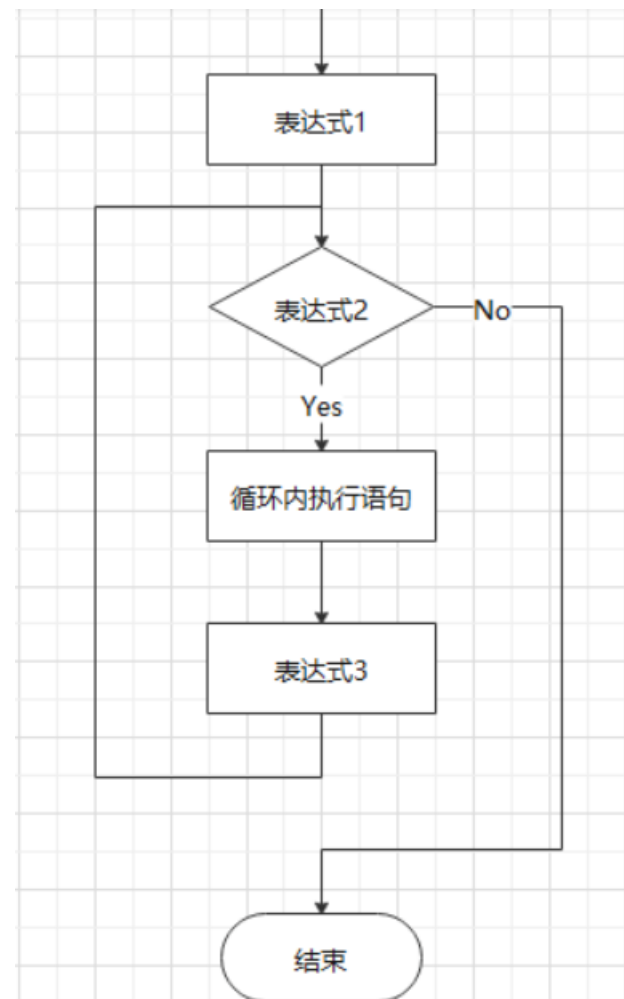
## 2.5.3 for语句

```
for(表达式1;表达式2;表达式3) {  
    语句序列  
}
```

```
for(;;) {}
```

```
int sum = 0;  
int i = 0  
for( ; i <= 10; i++) ;  
sum+=i;
```

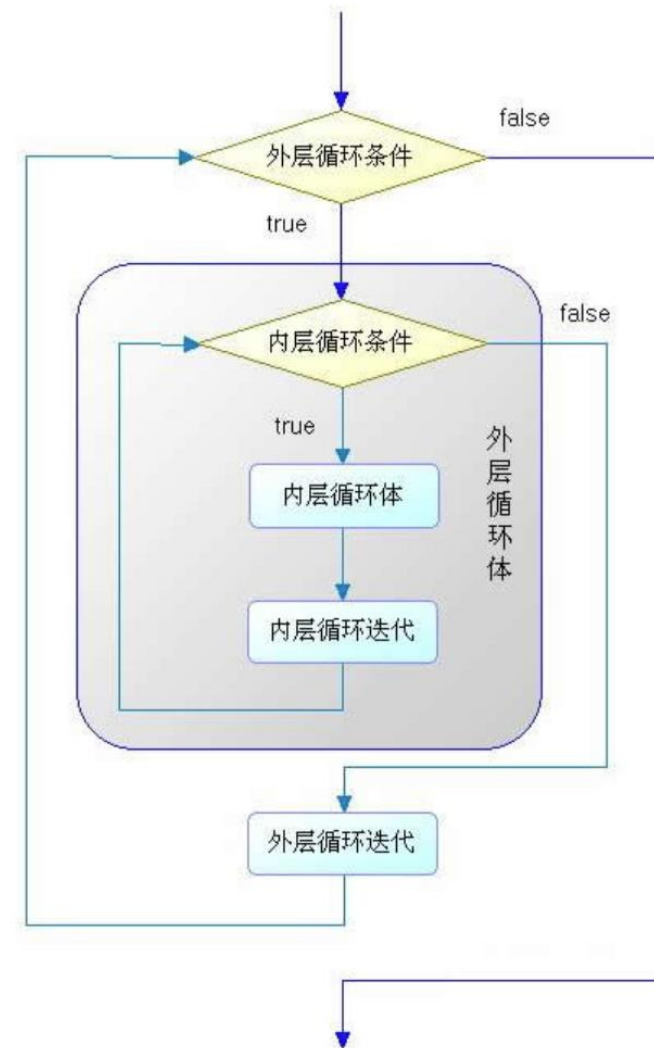
```
for(int i=0; i < 5; i++){  
    System.out.println("循环次数 =" + i);  
}
```



## 2.5.4 循环嵌套

```
for(初始化表达式;循环条件;操作表达式) {  
    ...  
    for (初始化表达式;循环条件;操作表达式) {  
        执行语句  
    }  
    ...  
}
```

```
for(int i = 1; i <= 9; i++){  
    for(int j =1; j <= i; j++){  
        System.out.print("*");  
    }  
    System.out.print("\n");  
}
```



## 2.5.5 跳转语句

- break语句：用在switch条件语句和循环语句中，它的作用是终止某个case并跳出switch结构。

```
public class Example16 {  
    public static void main(String[] args) {  
        int x = 1;                // 定义变量x，初始值为1  
        while (x <= 4) {          // 循环条件  
            System.out.println("x = " + x); // 条件成立，打印x的值  
            if (x == 3) {  
                break;  
            }  
            x++;                  // x进行自增  
        }  
    }  
}
```

## 2.5.5 跳转语句

- continue语句：用在循环语句中，它的作用是终止本次循环，执行下一次循环

```
public class Example18 {  
    public static void main(String[] args) {  
        int sum = 0;                // 定义变量sum，用于记住和  
        for (int i = 1; i <= 100; i++) {  
            if (i % 2 == 0) {        // i是一个偶数，不累加  
                continue;           // 结束本次循环  
            }  
            sum += i;                // 实现sum和i的累加  
        }  
        System.out.println("sum = " + sum);  
    }  
}
```

# /2.6

## 方法



## 2.6.1 什么是方法

方法就是一段可以重复调用的代码。

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

- 修饰符：是对访问权限的限定，例如，public、static都是修饰符
- 返回值类型：用于限定方法返回值的数据类型
- 参数类型：用于限定调用方法时传入参数的数据类型
- 参数名：是一个变量，用于接收调用方法时传入的数据
- return关键字：用于结束方法以及返回方法指定类型的值
- 返回值：被return语句返回的值，该值会返回给调用者

## 2.6.2 什么是方法（无返回值）

```
public class Example19 {  
    public static void main(String[] args) {  
        printRectangle(3, 5);  
        printRectangle(2, 4);  
        printRectangle(6, 10);  
    }  
    // 下面定义了一个打印矩形的方法，接收两个参数，其中height为高，width为宽  
    public static void printRectangle(int height, int width) {  
        // 下面是使用嵌套for循环实现*打印矩形  
        for (int i = 0; i < height; i++) {  
            for (int j = 0; j < width; j++) {  
                System.out.print("*");  
            }  
            System.out.print("\n");  
        }  
        System.out.print("\n");  
    }  
}
```

## 2.6.2 什么是方法（有返回值）

---

```
public class Example20 {  
    public static void main(String[] args) {  
        int area = getArea(3, 5);           // 调用 getArea方法  
        System.out.println(" The area is " + area);  
    }  
    // 下面定义了一个求矩形面积的方法，接收两个参数，其中x为高，y为宽  
    public static int getArea(int x, int y) {  
        int temp = x * y;                   // 使用变量temp记住运算结果  
        return temp;                       // 将变量temp的值返回  
    }  
}
```



## 2.6.2 方法的重载

Java允许在一个程序中定义多个名称相同的方法，但是参数的类型或个数必须不同，这就是方法的重载。

```
public class Example21 {  
    public static void main(String[] args) {  
        // 下面是针对求和方法的调用  
        int sum1 = add(1, 2);  
        int sum2 = add(1, 2, 3);  
        double sum3 = add(1.2, 2.3);  
        // 下面的代码是打印求和的结果  
        System.out.println("sum1=" + sum1);  
        System.out.println("sum2=" + sum2);  
        System.out.println("sum3=" + sum3);  
    }  
    // 下面的方法实现了两个整数相加  
    public static int add(int x, int y) {  
        return x + y;  
    }  
    // 下面的方法实现了三个整数相加  
    public static int add(int x, int y, int z) {  
        return x + y + z;  
    }  
    // 下面的方法实现了两个小数相加  
    public static double add(double x, double y) {  
        return x + y;  
    }  
}
```

# /2.7

## 数组



## 2.7.1 数组的定义

---

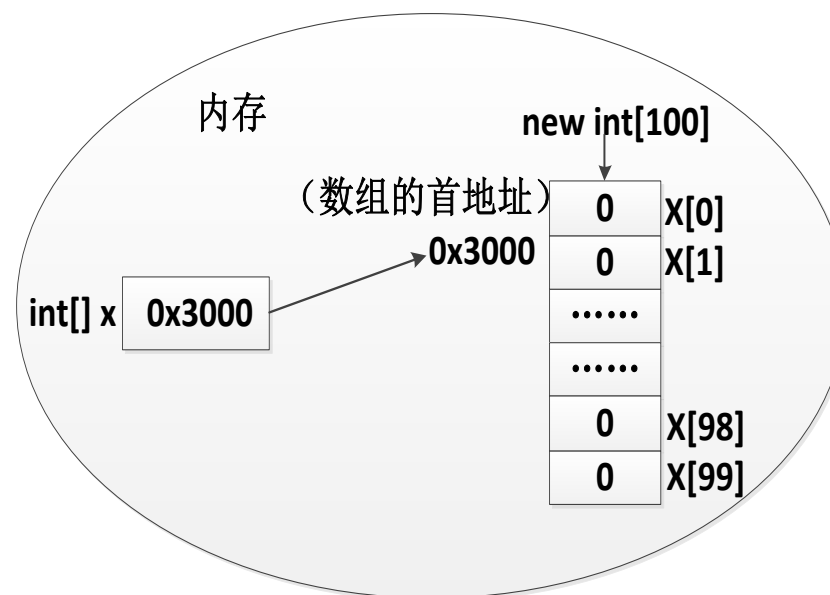
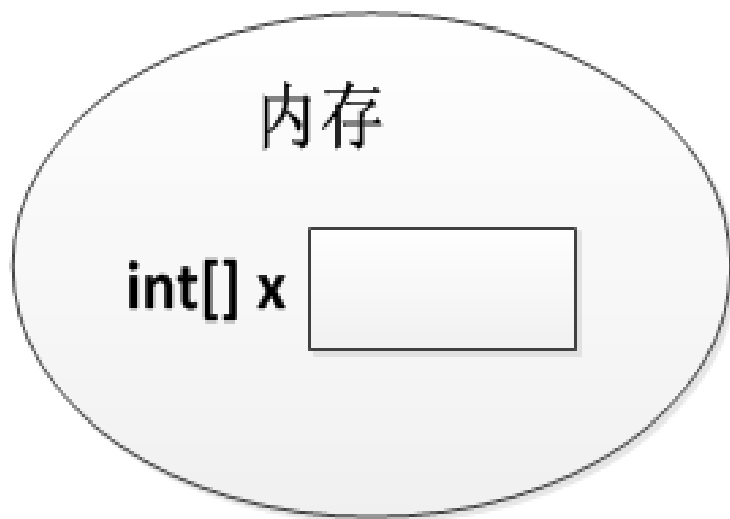
- 数组，是指一组类型相同的数据集合，数组中的每个数据称为元素。
- 数组可以存放任意类型的元素，但同一个数组里存放的元素类型必须一致。

声明数组的方法：

- `数据类型[] 数组名 = null;`(尽量避免使用这个方法)
- `数据类型[] 数组名;`  
`数组名 = new 数据类型[长度];`
- `数据类型[] 数组名 = new 数据类型[长度];`

## 2.7.1 数组的定义

```
int[] x;           // 声明一个 int[] 类型的变量  
x = new int[100];  // 创建一个长度为 100 的数组
```



## 2.7.1 数组的定义

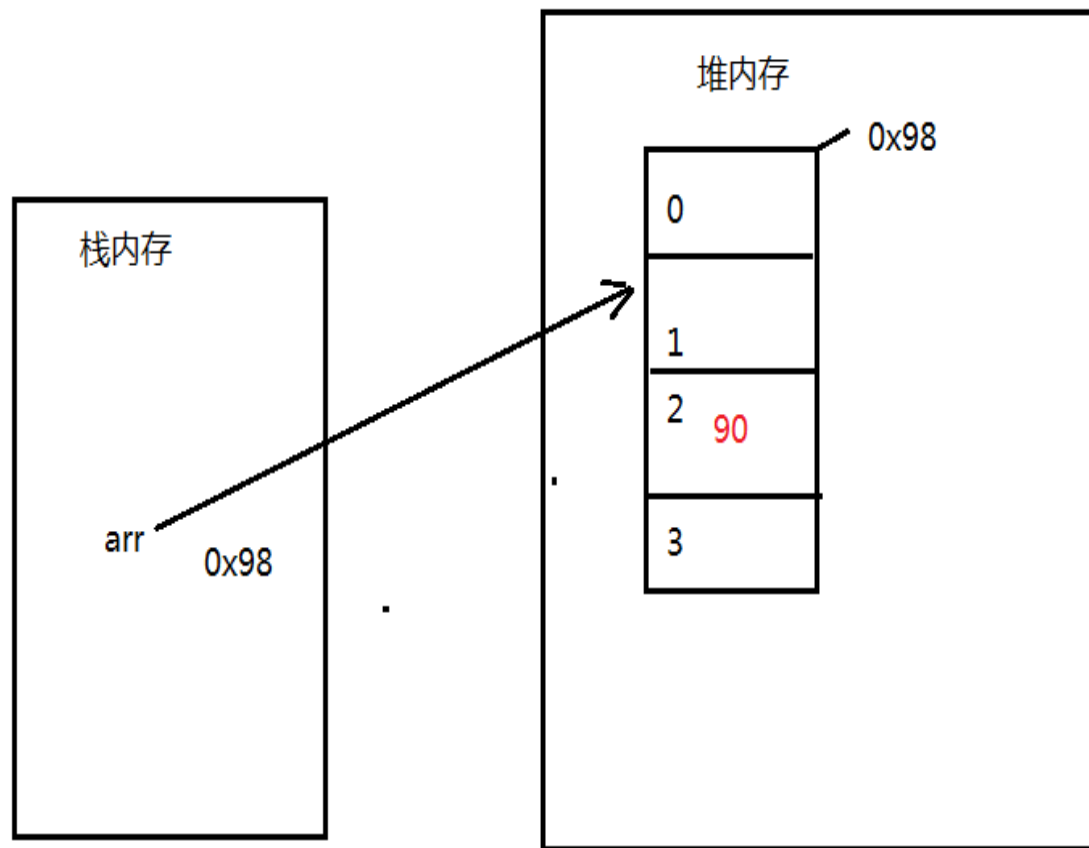
```
int[] arr = new int[4];
```

= 赋值运算符:

这时候的=号是把数组对象内存地址赋予给arr变量。

栈内存的特点：栈内存存储的都是局部变量，变量一旦出了自己的作用域，那么马上会从内存中消失，释放内存空间。

凡是以new关键字创建的对象，jvm都会在堆内存中开辟一个新的空间，创建一个新的对象。



arr[2] = 90

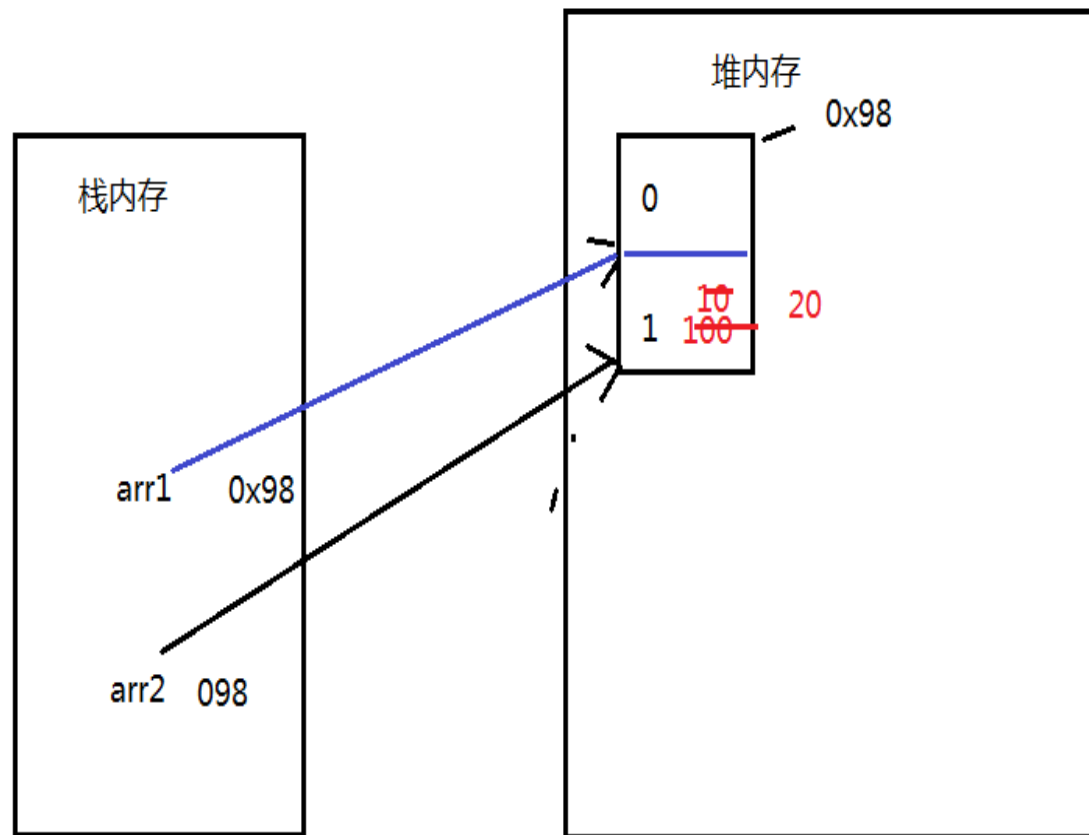
堆内存的特点：

堆内存存储的都是对象数据，对象一旦被使用完，并不会马上从内存中消失，而是等待垃圾回收器不定期把垃圾对象回收，这时候该对象才会消失，释放内存。

对象如果没有变量引用了，那么该对象就是一个垃圾对象了。

## 2.7.1 数组的定义

```
int[] arr1 = new int[2];  
arr1[1] = 100;  
int[] arr2 = arr1;  
arr1[1] = 10;  
arr2[1] = 20;  
System.out.println(arr1[1]);
```



## 2.7.1 数组的定义

- 在Java中，可通过“数组名.length”获得数组的长度。
- 数组变量和数组元素在内存中是分开存放的。
- 当数组被成功创建后，数组中元素会被自动赋予一个默认值。

数据类型↵	默认初始化值↵
byte、short、int、long↵	0↵
float、double↵	0.0↵
char↵	一个空字符，即'\u0000'↵
boolean↵	false↵
引用数据类型↵	null，表示变量不引用任何对象↵

## 2.7.1 数组的定义

---

```
public class Example22 {  
    public static void main(String[] args) {  
        int[] arr;                // 声明变量  
        arr = new int[3];         // 创建数组对象  
        System.out.println("arr[0]=" + arr[0]);    // 访问数组中的第一个元素  
        System.out.println("arr[1]=" + arr[1]);    // 访问数组中的第二个元素  
        System.out.println("arr[2]=" + arr[2]);    // 访问数组中的第三个元素  
        System.out.println("数组的长度是: " + arr.length); // 打印数组长度  
    }  
}
```



## 2.7.1 数组的定义

---

### 数组初始化（分配空间并赋初值）

- 在定义数组时只指定数组的长度，由系统自动为元素赋初值的方式称为动态初始化。
- 在定义数组的同时就为数组的每个元素赋值成为静态初始化。

- 数据类型[] 数组名 = new 类型[] {元素, 元素, ...}
- 数据类型[] 数组名 = {元素, 元素, 元素, ...}

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

```
int[] arr1 = {2,3,4,5};
```

## 2.7.2 数组的常见操作

---

### 1. 数组的遍历

```
int [] arr3 = new int[4];
for (int i = 0; i < arr3.length; i++) {
    arr3[ i ] = i*i;
}
for (int i : arr3) { //Java 5以上版本支持
    System.out.println(i);
}
```

## 2.7.2 数组的常见操作

## 2. 最值的获取

[illegible]

## 2.7.2 数组的常见操作

---

### 3. 数组的排序

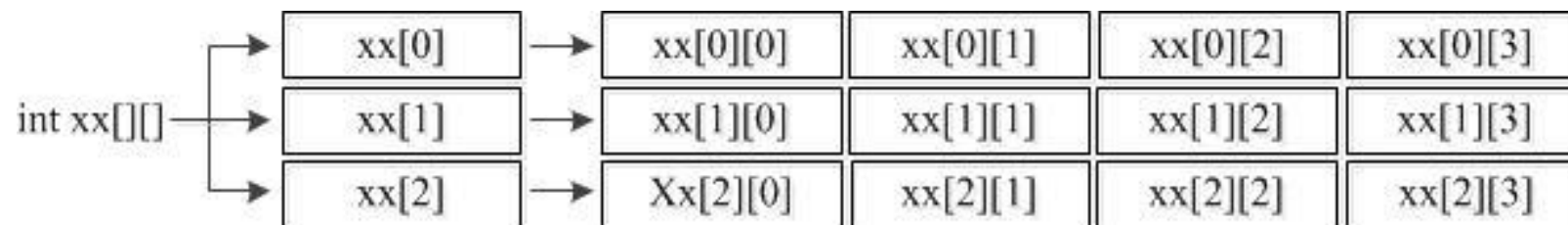
例子：冒泡排序法

## 2.7.3 二维数组

- 多维数组可以简单地理解为在数组中嵌套数组。在程序中比较常见的就是二维数组。
- 二维数组的定义方式
- 方式一：

```
int[][] arr = new int[3][4];
```

上面的代码定义了一个3行4列的二维数组



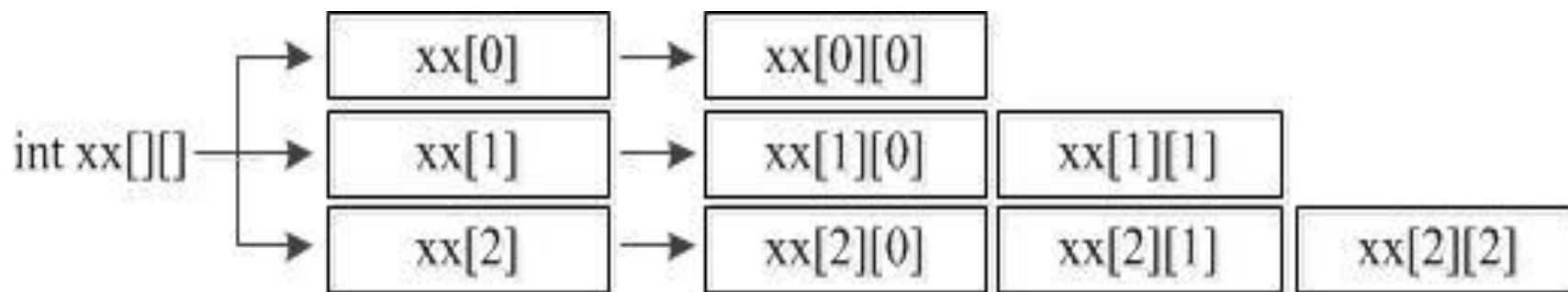
## 2.7.3 二维数组

- 方式二:

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

- 上述方式与第一种方式类似，只是数组中每个元素的长度不确定，采用第二种方式常见的数组结构如下图所示。

上面的代码定义了一个3行4列的二维数组

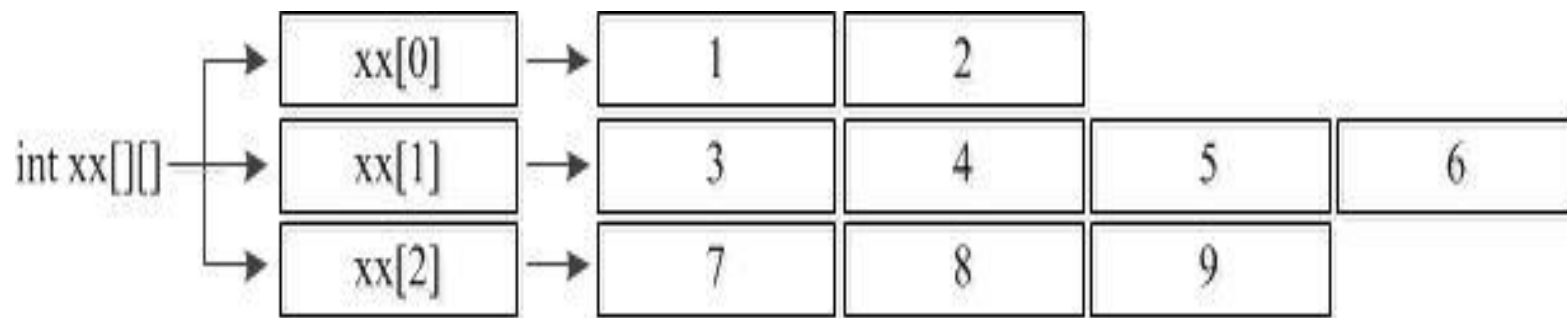


## 2.7.3 二维数组

- 方式三：

```
int[][] arr = {{1,2},{3,4,5,6},{7,8,9}};
```

采用上述方式定义的二维数组有三个元素，这三个元素都是数组，分别是{1,2}、{3,4,5,6}、{7,8,9}。



## 2.7.3 二维数组

---

多维数组的访问也是通过索引的方式。

```
arr[0][1];
```

二维数组的例子