Java基础5 数组

数组动态初始化

范例： 声明并开辟空间

```
public class ArrayDemo {
    public static void main(String[] args) {
        int[] data = new int[3]; /*开辟了一个长度为3的数组*/
        data[0] = 10; // 第一个元素
        data[1] = 20; // 第二个元素
        data[2] = 30; // 第三个元素
        for(int x = 0; x < data.length; x++) {
            System.out.println(data[x]); //通过循环控制索引
        }
    }
}
```

范例：分布模式开辟

```
public class ArrayDemo {
    public static void main(String[] args) {
        int[] data = null;
        data = new int[3]; /*开辟了一个长度为3的数组*/
        data[0] = 10; // 第一个元素
        data[1] = 20; // 第二个元素
        data[2] = 30; // 第三个元素
        for(int x = 0; x < data.length; x++) {
            System.out.println(data[x]); //通过循环控制索引
        }
    }
}
```

但是千万要记住，数组属于引用数据类型，所以在数组使用之前一定要开辟控件（实例化），如果使用了没有开辟空间的数组，则一定会出现 `NullPointerException` 异常信息：

```
public class ArrayDemo {
    public static void main(String[] args) {
        int []data = null;
        System.out.println(data[x]);
    }
}
```

数组静态初始化

在之前所进行的数组定义都有一个明显特点：数组先开辟内存空间，而后再使用索引进行内容的设置，实际上这种做法都叫做动态初始化，而如果希望数组在定义的时候可以同时出现设置内容，那么就可以采用静态初始化完成。

```
public class ArrayDemo {
    public static void main(String args[]) {
        int data[] = {1, 2, 4, 545, 11, 32, 13131, 4444};
        for(int i = 0; i < data.length; i++) {
            System.out.println(data[i]);
        }
    }
}
```

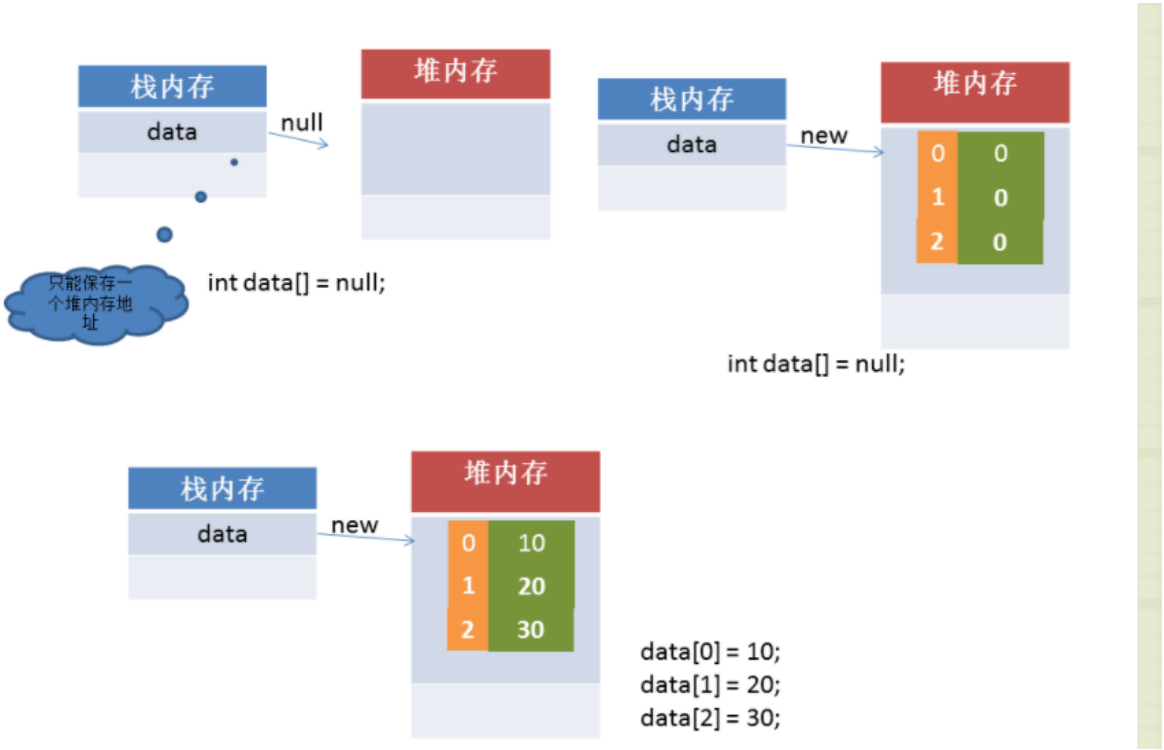
在开发之中，对于静态数组的初始化强烈建议使用完整语法模式，这样可以轻松地使用匿名数组这一概念。

```
public class ArrayDemo {
    public static void main(String[] args) {
        System.out.println(new int[] {1, 2, 4, 545, 11, 32, 13131,
4444}.length);
    }
}
```

数组引用传递

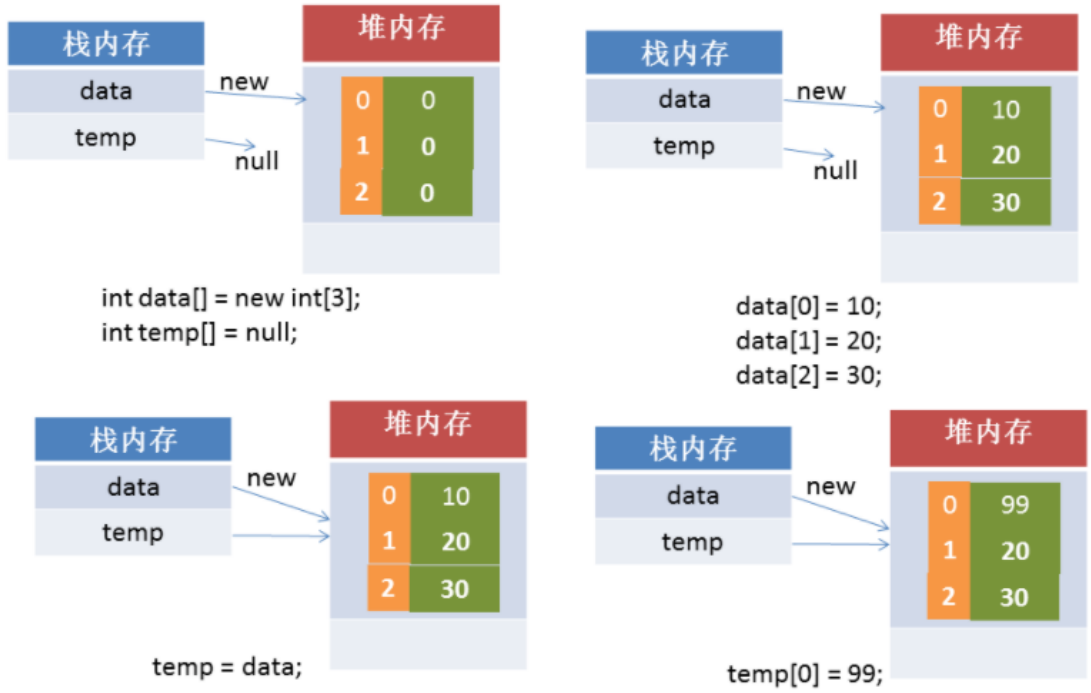
既然数组属于引用数据类型，那么也一定可以发生引用传递。在这之前首先来研究一下数组的空间开辟。

```
public class ArrayDemo {
    public static void main(String[] args) {
        int[] data = null;
        data = new int[3]; //开辟一个长度为3的数组
        data[0] = 10;
        data[1] = 20;
        data[2] = 30;
    }
}
```



那么既然说到了引用数据类型了，就一定可以发生引用传递，而现在的引用传递的本质也一定是：同一块堆内存空间可以被不同的栈内存所指向。

```
public class ArrayDemo {
    public static void main(String args[]) {
        int data[] = null;
        data = new int[3]; //开辟一个长度为3的数组
        int temp[] = null; //声明对象
        data[0] = 10;
        data[1] = 20;
        data[2] = 30;
        temp = data;  //int temp[] = data;
        temp[0] = 99;
        for(int i = 0; i < temp.length; i++) {
            System.out.println(data[i]);
        }
    }
}
```



引用传递分析都是一个套路。同一块堆内存被不同的栈内存所指向。

二维数组

```
public class ArrayDemo {
    public static void main(String args[]) {
        //此时的数组并不是一个等列数组
        int data[][] = new int[][] {
            {1, 2, 3}, {4, 5}, {6, 7, 8, 9}};
        //如果在进行输出的时候一定要使用双重循环，
        //外部的循环控制输出的行数，而内部的循环控制输出列数
        for(int i = 0; i < data.length; i++) {
            for(int j = 0; j < data[i].length; j++) {
                System.out.print("data[" + i + "][" + j + "]=" + data[i][j] +
"、");
            }
            System.out.println();
        }
    }
}
```

出现并不高

数组方法调用

范例： 定义一个方法，该方法可以实现数组的内容的乘2

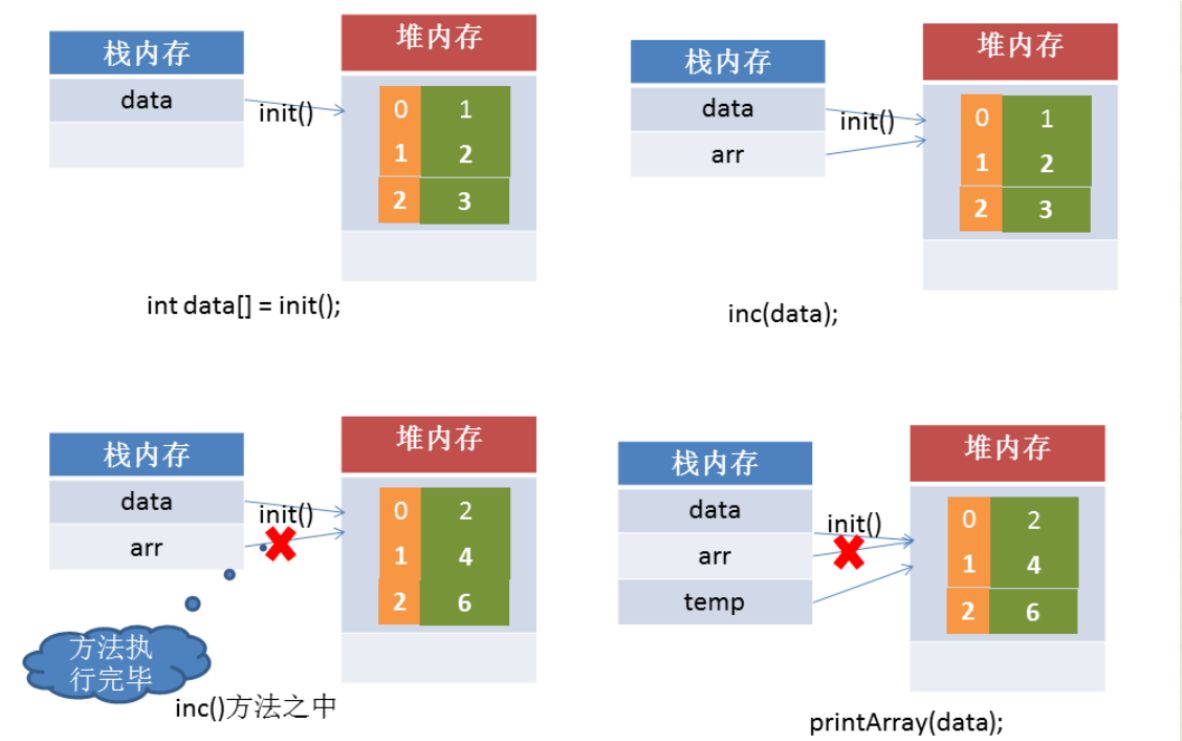
```
public class ArrayDemo {
    public static void main(String args[]) {
            int data[] = init();
            inc(data);
            printArray(data);
```

```
    }

    public static void inc(int arr[]) {
        for(int x = 0; x < arr.length; x++)
            arr[x] *= 2;
    }

    //此时的方法希望可以返回一个数组类型，所以返回值类型定义为整形数组
    public static int[] init() {
        return new int[] {1, 2, 3, 4, 6};
    }

    //定义一个专门进行数组输出的方法
    public static void printArray(int temp[]) {
        for (int i = 0; i < temp.length; i++) {
            System.out.print(temp[i] + "、");
        }
    }
}
```



相关方法

在Java本身给出的类库之中也提供有对于数组的操作的相关支持方法。

范例： 实现数组排序操作

```
int data[] = new int[] {5, 13, 1, 55, 77};
java.until.Arrays.sort(data);
```

范例：数组的拷贝

System.arraycopy(原数组名称, 原数组开始点, 目标数组名称, 目标数组开始点, 拷贝长度)

```
int dataA[] = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9};
        int dataB[] = new int[] {11, 22, 33, 44, 55, 66, 77, 88, 99};
        System.arraycopy(dataB, 4, dataA, 1, 3);
```