2、具体内容

所有的项目开发之中都一定会存在有数组的使用，但是本次所讲解的知识数组的基本概念，而基本形式的数组出现的几率会有，但是不高，并且也不会涉及到过多复杂的操作，这只是针对你自己编写的程序代码而言。

■数组的基本概念

如果说现在要定义100个整型变量，那么按照传统的做法，现在的实现如下：

```
int i1,i2,i3,i4,.....,i100;
```

这种方式的确是可以进行定义，但是如果说这100个变量属于关联的一组变量，则按照此种模式定义出来的变量就不适合程序维护了（没有任何的参考规律），所以在程序开发之中考虑到一组变量的整体维护，专门提供有数组的概念，数组的本质在于：一组相关变量的集合，但是需要注意的一点的是：在Java里面将数组定义为了引用数据类型，所以数组的使用一定要牵扯到内存的分配，那么首先就一定可以想到使用关键字new来处理，数组的定义格式：

- 数组的动态初始化，初始化之后数组每一个元素的保存内容为其对应数据类型的默认值：

|- 声明并初始化数组：

|- 数据类型 数组名称 [] = new 数据类型[长度];

|- 数据类型 [] 数组名称 = new 数据类型[长度];

- 数组的静态初始化：在数组定义的时候就为其设置好里面的内容：

|- 简化格式：数据类型 数组名称 [] ={数据1,数据2,数据3,……};

|- 完整格式：数据类型 数组名称 [] = new 数据类型 [] {数据1,数据2,数据3,……};

当创建了一个数组之后就可以按照如下的方式进行使用：

• 数组里面可以通过角标进行每一个元素的访问，角标从0开始定义，所以可以使用的角标范围：“0~数组长度-1”，同时一定要注意，如果使用的时候超过了数组角标范围则会出现“java.lang.ArrayIndexOutOfBoundsException:”（数组越界）异常。

• 使用数组是为了其可以进行方便的变量管理，所以在进行数组操作的时候往往利用for循环来完成；

• 对于数组的长度也可以使用“数组名称.length”属性进行获得。

范例：定义数组

```
public class ArrayDemo{
    public static void main(String[] args) {
        //使用数组的动态初始化实现了数组的定义
        int data [] = new int [3];
        data[0] = 11;//为数组设置内容
        data[1] = 23;//为数组设置内容
        data[2] = 56;//为数组设置内容
        for(int x=0;x<data.length;x++){
            System.out.println(data[x]);
        }
    }
}
```

在以后进行项目的开发过程之中，见到最多的数组使用形式：进行数组的循环处理。

数组本身分为动态初始化与静态初始化，上面使用的是动态初始化，动态初始化之后会发现数组之中的每一个元素的内容都是其对应数据类型的默认值，随后可以通过下标为数组进行内容的设置，如果现在不希望这么复杂，而是希望数组定义的时候就已经可以提供内容，则可以采用静态初始化的方式完成。

范例：使用静态初始化定义数组

```
public class ArrayDemo{
    public static void main(String[] args) {
        //使用数组的静态初始化
        int data [] = new int []{11,23,56};
        for(int x=0;x<data.length;x++){
            System.out.println(data[x]);
        }
    }
}
```

对于数组的操作而言，基本上都是拿到数据后循环控制。

■数组的引用传递

通过数组的基本定义可以发现，在数组使用过程之中依然需要关键字new进行内存空间的开辟，同理，那么这里面也一定存在有内存的关系匹配。

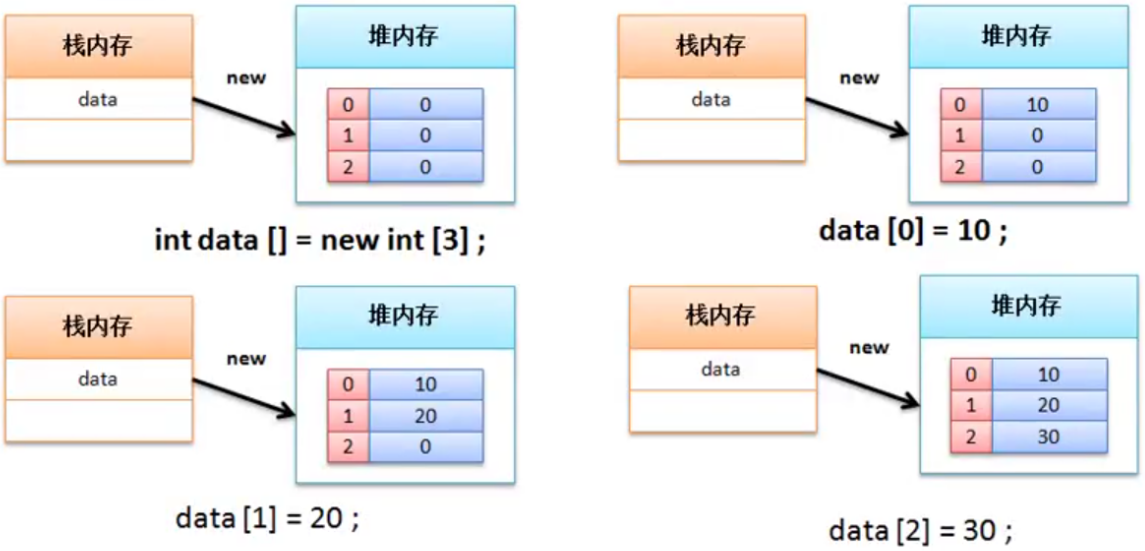
范例：定义一个简单代码

```
public class ArrayDemo{
```

```
    public static void main(String[] args) {
        //使用数组的静态初始化
        int data [] = new int [3];
        data[0] = 10;//为数组设置内容
        data[1] = 20;//为数组设置内容
        data[2] = 30;//为数组设置内容
        for(int x=0;x<data.length;x++){
            System.out.println(data[x]);
        }
    }
}
```

以上面的程序为例下面要进行一次内存的简单分析。

数组内存分析



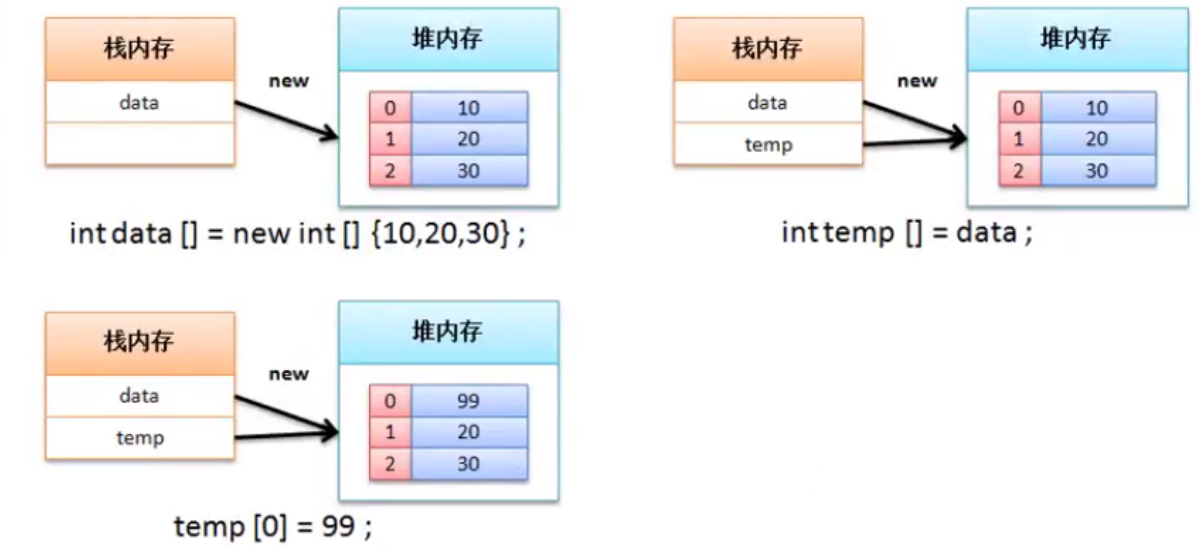
但是数组本身逼近是属于引用数据类型，那么既然是引用数据类型，就一定会发生引用传递，引用传递应该还是按照传统的方式那样：一个堆内存可以被多个栈内存所指向。

范例：观察数组引用

```
public class ArrayDemo{
    public static void main(String[] args) {
        int data[] = new int [3]{10,20,30};//静态初始化
        int temp[] = data;   //引用传递
        temp[0] = 99;
        for(int x=0;x<data.length;x++){
            System.out.println(data[x]);
        }
    }
}
```

下面通过此程序进行内存分析。

数组内存分析



由于数组属于引用类型，所以一定要为其开辟堆内存空间之后才可以使用，如果现在使用了为开辟堆内存空间的数组，则一定会出现“NullPointerException”异常。

```
public class ArrayDemo{
	public static void main(String[] args) {
		int data[] = null;
		System.out.println(data[0]);
	}
}
```

必须提供有实例化对象才可以使用数组的操作形式进行数组的操作。

■foreach迭代输出

对于数组而言，一般都会使用for循环进行输出，但是在使用传统for循环输出的时候往往都采用了下标的形式访问。

```
public class ArrayDemo {
	public static void main(String[] args) {
		int data[] = new int [3]{10,20,30};
		for(int x=0;x<data.length;x++){
			System.out.println(data[x]);
		}
	}
}
```

而从JDK1.5以后为了减轻下标对程序的影响（如果下标处理不当则会出现数组越界异常），所以参考.NET中的设计引用了一个增强型for循环（foreach），利用foreach的语法结构可以直接自动获取数组中的每一个元素，避免下标访问。

for(数据类型 变量 : 数组|集合){}

最大的特点在于可以自动将数组中的每一个元素的内容取出保存在变量里面，这样就可以直接通过变量获取数组内容，而避免了下标的方式来获取了。

范例：使用foreach语法形式输出

```
public class ArrayDemo {
	public static void main(String[] args) {
		int data[] = new int [3]{10,20,30};
		for(int temp : data){//自动循环，将data数组每一个内容交给temp
			System.out.println(temp[x]);
		}
	}
}
```

这种语法的好处是可以避免下标的操作。

■二维数组

在之前所定义的数组里面会发现只有一个"[]"，所以这个时候的数组就好像一行数据一样，可以利用下标进行行数据的访问。

- 传统的数组就好比一行数据：

下标:	0	1	2	3	4	5
数据:	890	90	91	789	123	89

- 如果说现在你需要的是一个多行多列的结构（表），则就需要通过两个下标才可以描述出一个数据，那么就需要行下标与列下标共同定义才可以找到，所以这样的数组形式就称为二维数组。

下标	0	1	2	3	4	5
0	890	90	91	789	123	89
1	1	23	12	6	8	0
2	11	2	5	7	32	246

对于二维数组可以使用的定义语法如下：

- 数组的动态初始化：

|- 数据类型 数组名称[][] = new 数据类型[col][row];

- 数组的静态初始化：

|- 数据类型 数组名称[][] = new 数据类型[][]{{数据,数据,……},{数据,数据,……},……};

范例：定义二维数组

```
public class ArrayDemo {
```

```
    public static void main(String[] args) {
        int data[][] = new int      [][]{
            {1,2,3,4,5},{1,2,3},{5,6,7,8}
        };
        for(int x = 0;x<data.length;x++){
            for(int y=0;x<data[x].length;y++){
                System.out.println("data["+x+"]["+y+"]"+data[x][y]);
            }
            System.out.println();
        }
    }
}
```

既然二维数组的每一行都属于一个数组，那么这种情况下就可以通过每一行的数组求出数组长度。

索引(下标)	***0***	***1***	***2***	***3***	***4***
0	1	2	3	4	5
1	1	2	3		
2	5	6	7	8	

如果这个时候要求使用foreach来进行输出呢？

范例：使用foreach输出二维数组

```
public class ArrayDemo {
    public static void main(String[] args) {
        int data[][] = new int      [][]{
            {1,2,3,4,5},{1,2,3},{5,6,7,8}
        };
        for(int temp[]: data){
            for(int num:temp){
                System.out.println(num+"、");
            }
            System.out.println();
        }
    }
}
```

通过foreach的输出格式可以清楚的观察到，二维数组就是数组的嵌套使用。随着开发技术的发展，如果要进行一些应用层程序开发，那么很少会设计到二维数组，更不用说更高级的多维数组。

■数组与方法

对于引用数据类型而言，主要的特点是可以与方法进行引用传递，而数组本身也属于引用数据类型，所以自然也可以通过方法实现引用传递的操作。

范例：实现一个数组的引用传递

```
public class ArrayDemo {
	public static void main(String[] args) {
		int data[] = new int[]{1,2,3,4,5};
		printArray(data);//传递数组

	}
	public static void printArray(int temp[]){
		for(int x=0;x<data.length;x++){
			System.out.println(data[x]);
		}
	}
}
```

对于此时的引用传递具体的内存关系如下：

数组与方法

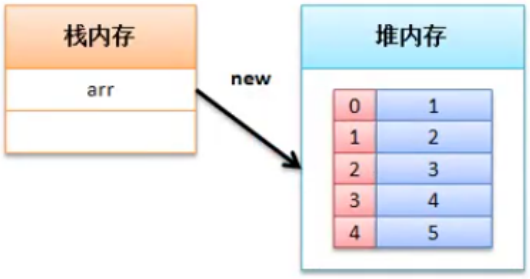
既然可以通过方法来接收一个数组，那么也就可以通过方法返回一个数组对象，那么此时只需要在方法的返回值类型上进行控制即可。

范例：定义方法返回数组

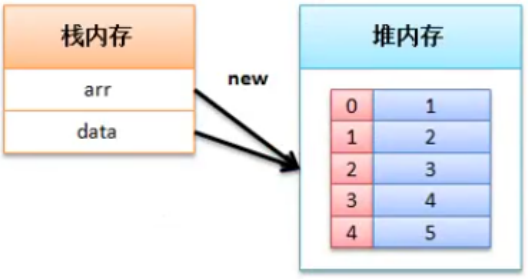
```
public class ArrayDemo {
	public static void main(String[] args) {
		int data[] = initArray();//通过方法可以获得数组
		printArray(data);//传递数组

	}
	public static int[] initArray(){
		int arr[] = new int[]{1,2,3,4,5};
		return arr;	//return a array
	}
	//要求接收一个int型的数组
	public static void printArray(int temp[]){
		for(int x=0;x<data.length;x++){
			System.out.println(data[x]);
		}
	}
}
```

下面来针对与此程序进行内存关系分析：

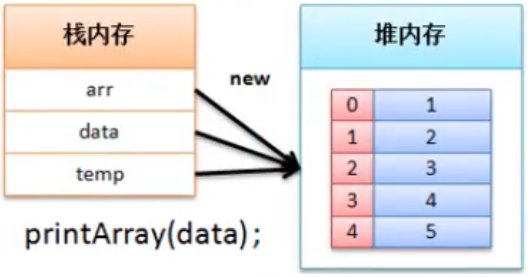
数组与方法



initArray()



int data [] = initArray();

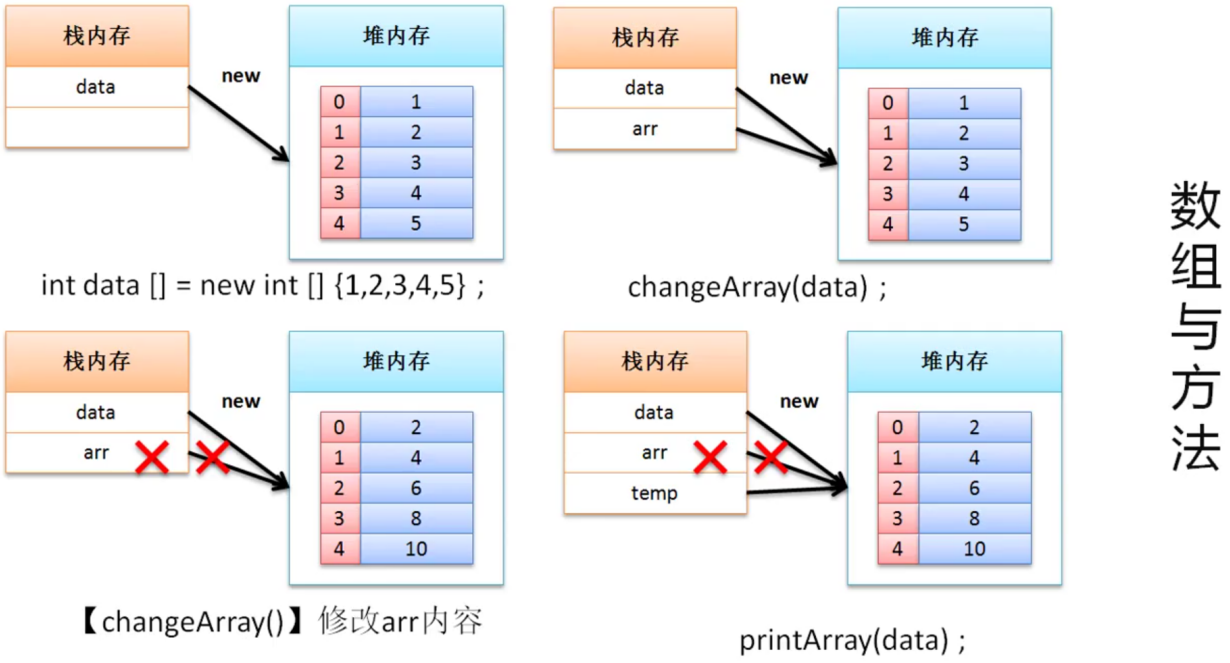


printArray(data);

范例：通过方法修改数组内容

```
public class ArrayDemo {
	public static void main(String[] args) {
		int data[] = new int[]{1,2,3,4,5};//通过方法可以获得数组
		changArray(data);
		printArray(data);//传递数组

	}
	public static void changArray(int arr[]){
			for(int x=0;x<data.length;x++){
				arr[x]*=2;//每个元素的内容乘2保存
		}
	}
	//要求接收一个int型的数组
	public static void printArray(int temp[]){
		for(int x=0;x<data.length;x++){
			System.out.println(data[x]);
		}
	}
}
```

本程序的内存关系如下：



案例：随意定义一个int数组，要求可以计算出这个数组元素的总和、最大值、最小值、平均值

对于此程序的最基本的实现如下：

```
public class ArrayDemo{
	public static void main(String[] args) {
		int data[] = new int[]{1,2,3,4,5};//通过方法可以获得数组
		int sum = 0;
		double avg = 0.0;
		int max = data[0];//假定第一个是最大值
		int min = data[0];//假定第一个是最小值
		for(int x=0;x<data.length;x++){
			if(data[x]>max)		//max地址改变
				max = data[x];
			if(data[x]<min)
				min = data[x];
			sum+=data[x];//每个元素的内容乘2保存
		}
		avg = sum/data.length;
		System.out.println("内容sum："+sum);
		System.out.println("数组avg："+avg);
		System.out.println("数组max："+max);
		System.out.println("数组min："+min);
	}
}
```

主方法所在的类往往被称为主类，那么既然是主类肯定不希望涉及到过于复杂的功能。在进行开发的过程中，主方法就相当于一个客户端，而对于客户端的代码应该尽量简单，所以这个时候最好的做法是将一系列的计算过程交给单独的程序类去完成。

范例：改善操作设计

```
class ArrayUtil {//是一个操作工具的类
	private int sum;//保存总和
```

```
	private double avg;//保存平均值
	private int max;//保存最大值
	private int min;//保存最小值
	public ArrayUtil(int data[]) {//进行数组计算
		int max = data[0];//假定第一个是最大值
		int min = data[0];//假定第一个是最小值
		for(int x=0;x<data.length;x++){
			if(data[x]>max)		//max地址改变
				this.max = data[x];
			if(data[x]<min)
				this.min = data[x];
			this.sum+=data[x];//每个元素的内容乘2保存
		}
		this.avg = this.sum/data.length;
	}
	public int getSum() {
		return sum;
	}
	public void setSum(int sum) {
		this.sum = sum;
	}
	public double getAvg() {
		return avg;
	}
	public void setAvg(double avg) {
		this.avg = avg;
	}
	public int getMax() {
		return max;
	}
	public void setMax(int max) {
		this.max = max;
	}
	public int getMin() {
		return min;
	}
	public void setMin(int min) {
		this.min = min;
	}

}
public class ArrayDemo{
	public static void main(String[] args) {
		int data[] = new int[]{1,2,3,4,5};//通过方法可以获得数组
		ArrayUtil util = new ArrayUtil(data);//data calculate
		System.out.println("内容sum："+util.getSum());
		System.out.println("数组avg："+util.getAvg());
		System.out.println("数组max："+util.getMax());
		System.out.println("数组min："+util.getMin());
	}
}
```

此时的主类就好比我们使用电脑一样，只关心如何操作，而具体的操作过程被类进行包装了。

■数组操作案例：数组排序

数组排序是指可以将一个杂乱无章的数组按照顺序进行进行排序，但是对于数组排序总是通过一个基础的模型来完成的，例如：本次先通过一个升序排序的方式来观察排序的处理。

数组排序

原始数组	8	9	0	2	3	5	10	7	6	1
第1次排序	8	0	2	3	5	9	7	6	1	10
第2次排序	0	2	3	5	8	7	6	1	9	10
第3次排序	0	2	3	5	7	6	1	8	9	10
第4次排序	0	2	3	5	6	1	7	8	9	10
第5次排序	0	2	3	5	1	6	7	8	9	10
第6次排序	0	2	3	1	5	6	7	8	9	10
第7次排序	0	2	1	3	5	6	7	8	9	10
第8次排序	0	1	2	3	5	6	7	8	9	10

范例：数组排序分析

```
public class ArrayDemo{
    public static void main(String[] args) {
        int data[] = new int[]{8,9,0,2,3,5,10,7,6,1};//通过方法可以获得数组
        for(int x=0;x<data.length;x++){
            for(int y=0;y<data.length-x-1;y++){
                if(data[y]>data[y+1]){    //交换数据
                    int temp = data[y];
                    data[y] = data[y+1];
                    data[y+1] = temp;
                }
            }
        }
        printArray(data);
    }
    public static void printArray(int data[]){
        for(int x=0;x<data.length;x++){
            System.out.println(data[x]);
        }
    }
```

```
}
```

以上的程序代码都是通过主方法完成的，不符合于面向对象的设计结构，那么最好的做法是将这个排序处理的操作交给一个类进行处理完成。

```
class ArrayUtil {//是一个操作工具的类
	public static void sort(int data[]){   //
		for(int x=0;x<data.length;x++){
			for(int y=0;y<data.length-x-1;y++){
				if(data[y]>data[y+1]){     //交换数据
					int temp = data[y];
					data[y] = data[y+1];
					data[y+1] = temp;
				}
			}
		}
	}
	public static void printArray(int data[]){
		for(int x=0;x<data.length;x++){
			System.out.print(data[x]+" ");
		}
		System.out.println();
	}
}
public class JavaDemo {
	public static void main(String[] args) {
		int data[] = new int[]{8,9,0,2,3,5,10,7,6,1};//通过方法可以获得数组
		ArrayUtil.sort(data);
		ArrayUtil.printArray(data);
	}

}
```

在以后进行类设计的时候，如果发现类中没有属性存在的意义，那么定义的方法就没有必要使用普通方法了，因为普通方法需要在有实例化对象产生的情况下才可以调用。

■数组操作案例：数组反转

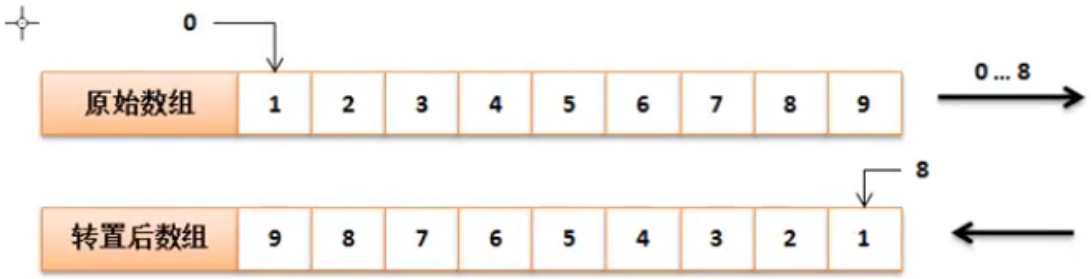
数组的反转操作指的是前后转置处理，即：首尾交换，例如，现在有一个数组，其内容如下：

- 数组内容：1,2,3,4,5,6,7,8,9；
- 交换后的内容：9,8,7,6,5,4,3,2,1；

对于数组的前后交换有两种做法：

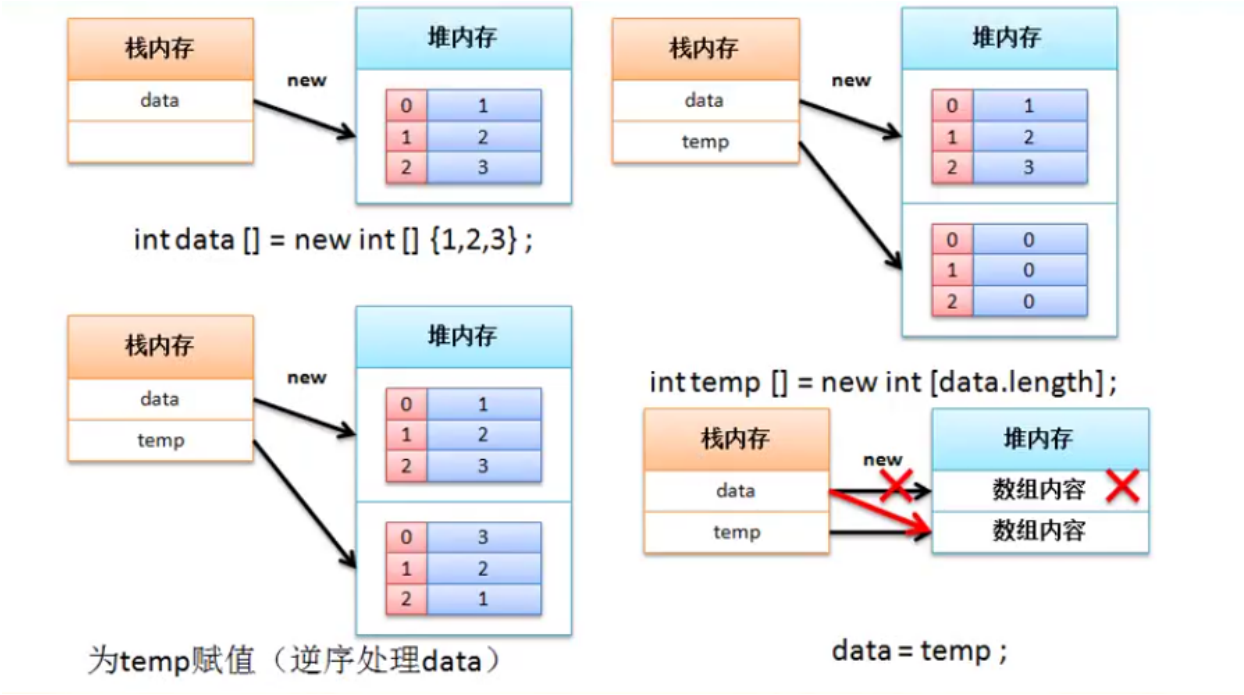
做法一：定义一个新的数组而后按照逆序的方式保存(会产生无用的垃圾空间)

数组转置




```
class ArrayUtil {//是一个操作工具的类
	public static void printArray(int data[]){
		for(int x=0;x<data.length;x++){
			System.out.print(data[x]+" ");
		}
		System.out.println();
	}
}
public class JavaDemo {
	public static void main(String[] args) {
		int data[] = new int[]{1,2,3,4,5,6,7,8,9};//通过方法可以获得数组
		int temp[] = new int[data.length];//第二个数组
		int foot = temp.length-1;	//第二个数组角标
		for(int x=0;x<data.length;x++){
			temp[foot--] = data[x];
		}
		data = temp;
		ArrayUtil.printArray(data);
	}
}
```

下面进行一下内存的分析处理，观察程序存在的问题。



做法二：在一个数组上转置

一个数组转置

现在如果要想实现这种转置最需要确定的是数组转换的次数，次数的计算"数组长度/2"，实际上不用去考虑数组是奇数个数还是偶数个数。

```
class ArrayUtil {//是一个操作工具的类
	public static void printArray(int data[]){
		for(int x=0;x<data.length;x++){
			System.out.print(data[x]+" ");
		}
		System.out.println();
	}
}
public class JavaDemo {
	public static void main(String[] args) {
		int data[] = new int[]{1,2,3,4,5,6,7,8,9};//通过方法可以获得数组
		int center = data.length/2;//确定转换的次数
		int head = 0;  //操作角标
		int tail = data.length - 1;//操作角标
		for(int x = 0;x<center;x++){
			int temp = data[head];
			data[head] = data[tail];
```

```
                data[tail] = temp;
                head++;
                tail--;
            }

            ArrayUtil.printArray(data);
        }
}
```

两种实现如果要进行比较可以发现，第一种处理方式循环次数较多，并且还会产生垃圾；第二种实现循环次数降低，但是存在if判断增加了时间复杂度，可是可以减少无用对象的产生，以提升性能。

范例：将转换功能变为类定义

```
class ArrayUtil {//是一个操作工具的类
        public static void reverse(int data[]){
                int center = data.length/2;//确定转换的次数
                int head = 0;   //操作角标
                int tail = data.length - 1;//操作角标
                for(int x = 0;x<center;x++){
                        int temp = data[head];
                        data[head] = data[tail];
                        data[tail] = temp;
                        head++;
                        tail--;
                }
        }
        public static void printArray(int data[]){
                for(int x=0;x<data.length;x++){
                        System.out.print(data[x]+" ");
                }
                System.out.println();
        }
}
public class JavaDemo {
        public static void main(String[] args) {
                int data[] = new int[]{1,2,3,4,5,6,7,8,9};//通过方法可以获得数组
                ArrayUtil.reverse(data);//转置处理
                ArrayUtil.printArray(data);
        }
}
```

可以发现数组由于可以通过角标进行元素的控制，所以相应的循环逻辑使用的会比较多。

■数组相关操作方法

由于数组是一个重要的概念，所以Java语言本身也提供有数组的相关处理，这些处理是在开发中使用的。

1、数组排序：java.util.Arrays.sort(数组名称)

```
import java.util.*;
class ArrayUtil {//是一个操作工具的类
	public static void printArray(int data[]){
		for(int x=0;x<data.length;x++){
			System.out.print(data[x]+" ");
		}
		System.out.println();
	}
}
public class JavaDemo {
	public static void main(String[] args) {
		int data[] = new int[]{1,2,3,4,5,6,7,8,9};//通过方法可以获得数组
		java.util.Arrays.sort(data);//排序
		ArrayUtil.printArray(data);
	}
}
```

2、数组拷贝(把方法做了一些变性)：

- System.arraycopy(源数组,源数组开始点,目标数组,目标数组开始点,拷贝长度)；

范例：实现数组拷贝

- 现在假设有两个数组
  - |- 数组一：1、2、3、4、5、6、7、8、9；
  - |- 数组而：11、22、33、44、55、66、77、88、99；
- 要求拷贝之后的数组而内容为：11、22、33、6、7、8、77、88、99；

```
class ArrayUtil {//是一个操作工具的类
	public static void printArray(int data[]){
		for(int x=0;x<data.length;x++){
			System.out.print(data[x]+" ");
		}
		System.out.println();
	}
}
public class JavaDemo {
	public static void main(String[] args) {
		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(dataA,5,dataB,3,3);
		ArrayUtil.printArray(dataB);
	}
}
```

这些操作的支持都是系统本身提供的，即：你都是可以在开发中使用的操作，实际上如要要自己实现拷贝也可以完成，直接定义一个方法操作即可。

```
class ArrayUtil {//是一个操作工具的类
```

```
    public static void printArray(int data[]){
        for(int x=0;x<data.length;x++){
            System.out.print(data[x]+" ");
        }
        System.out.println();
    }
    public static void arraycopy(int[] src,int sindex,int des[],int dindex,int len){
        for(int x=0;x<len;x++){
            des[dindex++] = src[sindex++];
        }
    }
}
public class JavaDemo {
    public static void main(String[] args) {
        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};
        ArrayUtil.arraycopy(dataA,5,dataB,3,3);
        ArrayUtil.printArray(dataB);
    }
}
```

如果要是自己去定义这种拷贝或者排序的方法就需要考虑所有的数据类型。

■方法可变参数

如果说现在要求定义一个方法，这个方法可以实现任意多个整型数据的相加处理。这样的情况下最早的时候通过数组来进行处理。

范例：传统实现

```
class ArrayUtil {//是一个操作工具的类
    public static int sum(int[] data){
        int sum = 0;
        for(int temp:data){
            sum+=temp;
        }
        return sum;
    }
}
public class JavaDemo {
    public static void main(String[] args) {
        System.out.println(ArrayUtil.sum(new int[]{1,2,3}));
    }
}
```

虽然以上的程序可以实现任意多个数字的参数内容传递，但是与实际的要求并不符合，实际要求的是可以传递任意多个参数，而不是一个数组。从JDK1.5开始为了方便开发者进行可变参数的定义，对于方法的参数提供有新的支持了。

范例：采用可变参数

```
class ArrayUtil {//是一个操作工具的类
```

```
	public static int sum(int... data){
		int sum = 0;
		for(int temp:data){
			sum+=temp;
		}
		return sum;
	}
}
public class JavaDemo {
	public static void main(String[] args) {
		System.out.println(ArrayUtil.sum(1,2,3));
		System.out.println(ArrayUtil.sum(new int[]{1,2,3}));
	}
}
```

可变参数的最大作用在于，在以后进行一些程序类设计或者开发者调用的时候，利用此种形式就可以避免数组的传递操作了，可变参数的本质需要清楚的是：依然是数组。

■对象数组

在之前所接触到的都是基本数据类型定义的数组，但是在Java程序本身各种数据类型都可以成为数组，所以类也可以成为数组类型，而这样的数组就称为对象数组，对象数组的定义格式如下：

- 动态初始化：类 对象数组名称[] = new 类[长度];每一个元素的内容都是：null
- 静态初始化：类 对象数组名称[] = new 类[]{实例化对象,实例化对象,……};

范例：使用动态初始化定义对象数组

```
class Person
{
	private String name;
	private int age;
	public Person(String name, int age) {
		super();
		this.name = name;
		this.age = age;
	}

	public String getInfo() {
		return "person [name=" + name + ", age=" + age + "]";
	}
	//setter、getter略
}
public class JavaDemo {
	public static void main(String[] args) {
		Person per[] = new Person[3];//对象数组
		per[0] = new Person("sanzhang",20);
		per[1] = new Person("sili",10);
		per[2] = new Person("wuwang",18);
```

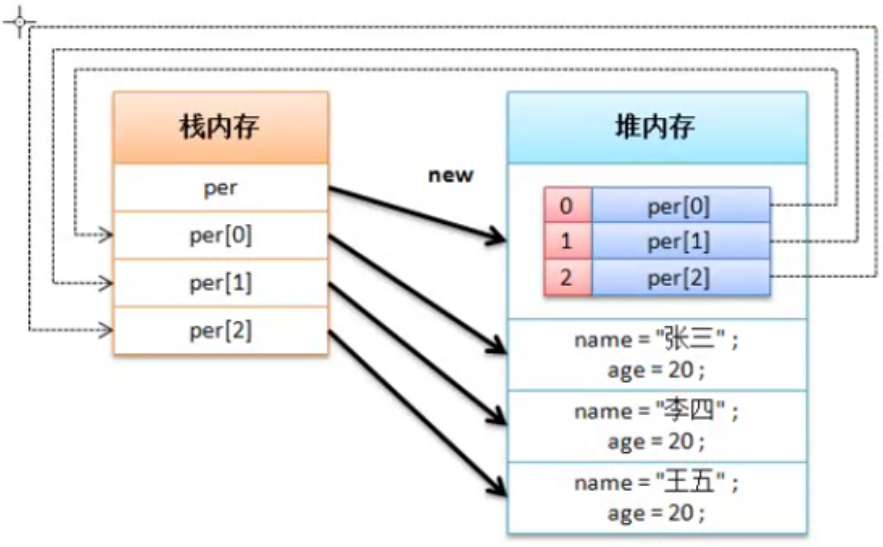
```
            for(int x=0;x<per.length;x++) {
                System.out.println(per[x].getInfo());
            }
        }
}
```

范例：对象数组静态初始化

```
class Person
{
        private String name;
        private int age;
        public Person(String name, int age) {
                super();
                this.name = name;
                this.age = age;
        }

        public String getInfo() {
                return "person [name=" + name + ", age=" + age + "]";
        }
        //setter、getter略
}
public class JavaDemo {
        public static void main(String[] args) {
                Person per[] = new Person[]{
                 new Person("sanzhang",20),new Person("sili",10),new Person("wuwang",18),
                };//对象数组

                for(int x=0;x<per.length;x++) {
                        System.out.println(per[x].getInfo());
                }
        }
}
```

对于对象数组而言，本事只是更换了一种所谓的数组定义的类型。

对象数组



所有的开发都不可能离开对象数组，但是通过一系列的分析也应该知道一个数组中最大的缺陷：长度是固定的。优势：数组线性保存，根据索引访问，速度较快。（时间复杂度为“1”）