**一旦数组被创建，它的大小是固定的。使用一个数组引用变量，通过下标来访问数组中的元素。**

**为了在程序中使用数组，必须声明一个引用数组的变量，并指明数组的元素类型。例如**

**//声明数组变量myList**

**double[] myList;**

**不同于基本数据类型变量的声明，声明一个数组变量时并不在内存中给数组分配任何空间。它只是创建一个对数组的引用的存储位置。如果变量不包含对数组的引用，那么这个变量的值为null**

**可以使用new操作符创建数组，并且将它的引用赋给一个变量：**

**arrayRefVar=new elementType[arraySize];**

**这句语句做了两件事情：**

**1）使用new elementType[arraySize]创建了一个数组；**

**2）把这个新创建的数组的引用赋值给变量arrayRefVar。**

**声明一个数组变量、创建数组、然后将数组引用赋值给变量这三个步骤可以合并在一条语句中，**

**double[] myList=new double[10];**

**一个数组变量看起来似乎是存储了一个数组，但实际上它存储的是指向数组的引用。严格来讲，一个数组变量和一个数组是不同的，但多数情况下它们的差别是可以忽略的。**

**当创建数组后，它的元素被赋予默认值，数值型基本数据类型的默认值为0，char型的默认值为'\u0000'，boolean型的默认值为false。**

**数组初始化语法中不适用操作符new。使用数组初始化语法时，必须将声明、创建和初始化数组都放在一条语句中。将它们分开会产生语法错误。因此下面是错误的**

**//错误语句**

**double[] myList;**

**myList={1.9,2.9,3.5};**

**处理数组元素时，经常会用到for循环，理由如下**

**1）数组中所有元素都是同一类型的。可以使用循环以同样的方式反复处理这些元素。**

**2）由于数组的大小是已知的，所以很自然地使用for循环。**

**foreach循环**

**foreach循环不使用下标变量就可以顺序地遍历整个数组。例如**

**//显示数组myList的所有元素**

**for(double e:myList){**

**System.out.println(e);**

**}**

**此代码可以读作“对myList中每个元素e进行以下操作”。注意，变量e必须声明为myList中元素相同的数据类型。**

**foreach语法：**

**for(elementType element:arrayRefVar){**

**//Process the element**

**}**

**数组的复制**

**要将一个数组中的内容复制到另外一个中，你需要将数组的每个元素复制到另外一个数组中。不能简单地使用=号，list2=list1，只是将list1的引用值复制给了ilst2.在这条语句之后，list1和list2都指向同一个数组。**

**复制数组有三种方法：**

**1）使用循环语句逐个地复制数组的元素。**

**2）使用System类中的静态方法arraycopy。**

**3）使用clone方法复制数组。**

**arraycopy的语法：**

**arraycopy(sourceArray,srcPos,tragetArray,tarPos,length);**

**srcPos和tarPos分别表示在源数组sourceArray和目标数组targetArray中的起始位置。从sourceArray复制到targetArray中的元素个数由参数length指定。**

**改写成如下就可以完整复制数组：**

**System.arraycopy(sourceArray,0,targetArray,0,sourceArray.length);**

**arraycopy方法没有给目标数组分配内存空间。复制前必须创建目标数组以及分配给它的内存空间。复制完成后，sourceArray和targetArray具有相同的内容，但占有独立的内存空间。**

**arraycopy方法违反了java命名习惯。**

**当将一个数组传递给方法时，数组的引用被传给方法。**

**java使用按值传递的方式将实参传递给方法。传递基本数据类型变量的值与传递数组值有很大的不同：**

**1.对于基本数据类型参数，传递的是实参的值。**

**2.对于数组类型参数，参数值是数组的引用，给方法传递的是这个引用。从语义上来讲，最好的描述就是参数传递的是共享信息（pass-by-sharing），即方法中的数组和传递的数组是一样的。**

**数组在java中是对象。JVM将对象存储在一个称作堆（heap）的内存区域中，堆用于动态内存分配。**

**当从方法中返回一个数组时，数组的引用被返回。**

**可变长参数列表**

**具有同样类型的可变长度的参数可以传递给方法，并将作为数组对待。方法中的参数声明如下：**

**（类型名...参数名）**

**在方法声明中，指定类型后紧跟省略号（...）。只能给方法中指定一个可变长参数，同时该参数必须是最后一个参数。任何常规参数必须在它之前。**

**java将可变长参数当做数组对待。可以将一个数组或数目可变的参数传递给可变长参数。当用数目可变的参数调用方法时，java会创建一个数组并把参数传给它。例如**

**public class VarArgsDemo {**

**public static void main(String args[]) {**

**printMax(34, 3, 3, 2, 56.5);**

**printMax(new double[]{1, 2, 3});**

**}**

**public static void printMax(double... numbers) {**

**if (numbers.length == 0) {**

**System.out.println("No argument passed");**

**return;**

**}**

**double result = numbers[0];**

**for (int i = 1; i < numbers.length; i++)**

**if (numbers[i] > result)**

**result = numbers[i];**

**System.out.println("The max value is " + result);**

**}**

**}**

**Arrays类**

**java.util.Arrays类包含一些实用的方法用于常见的数组操作，比如排序和查找。**

**java.util.Arrays类包括各种各样的静态方法，用于实现数组的排序和查找、数组的比较和填充数组元素，以及返回数组的字符串表示。这些方法都有对所有基本类型的重载方法。**

**可以使用sort或者parallelSort方法对整个数组或部分数组进行排序。**

**double[] numbers={6.0,4.4,2.5,4.6};**

**java.util.Arrays.sort(numbers);**

**java.util.Arrays.parallelSort(numbers);**

**java.util.Arrays.sort(numbers,1,3);**

**java.util.Arrays.parallelSort(numbers,1,3);**

**sort(numbers)对整个数组排序。sort(numbers,1,3)对从numbers[1]到numbers[3-1]的部分进行排序。如果你的计算机有多个处理器，那么parallelSort将更加高效。**

**可以采用二分查找法（binarySerach方法）在数组中查找关键字。数组必须提前排好序。如果数组中不存在关键字，方法返回-（插入点下标+1）**

**java.util.Arrays.binarySearch(numbers,0);**

**可以采用equals方法检测两个数组是否相等。如果它们的内容相同，那么这两个数组相等。**

**可以使用fill方法填充整个数组或部分数组。**

**//将5填充到list1中**

**java.util.Arrays.fill(list1,5);**

**//将8填充到元素list2[1]到list2[5-1]中**

**java.util.Arrays.fill(list2,1,5,8);**

**可以使用toString方法返回一个字符串，该字符串代表了数组中的所有元素。这是一个显示数组中所有元素的快捷和简便的方法。**

**命令行参数**

**main方法可以从命令行接收字符串参数。**

**main方法具有String[]类型参数args。参数args是一个字符串数组。main方法就像一个带参数的普通方法。可以通过传递实参来调用一个普通方法。**

**如果运行程序时没有传递字符串，那么使用new String[0]创建数组。在这种情况下，该数组是长度为0的空数组。args是对这个空数组的引用。因此，args不是null，但是args.length是0.**