笔记总链接：http://bbs.itheima.com/thread-200600-1-1.html

一、Java语言基础组成-Part 4

1.9 数组

1.9.4 数组操作常见操作

对数组操作最基本的动作就是存和取。

核心思想：就是对角标的操作。

示例：遍历并打印数组元素

```
class ArrayDemo{
        public static void main(String[] args) {
             int[] arr = {89,34,270,17};

             for(int x = 0; x < arr.length; x += 1){
                  System.out.println("arr[" + x + "] = " + arr[x] + ";");
             }
        }
}
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe
D:\code\day05>javac ArrayDemo.java

D:\code\day05>java ArrayDemo
arr[0] = 89;
arr[1] = 34;
arr[2] = 270;
arr[3] = 17;
```

常见操作一：获取最值（最大值，最小值）

思路：

1、需要进行比较，并定义变量记录住每次比较后较大的值。

2、对数组中的元素进行遍历取出，和变量中记录的元素进行比较。

如果遍历到的元素大于变量中记录的元素，就用变量该记录住大的值。

3、遍历结果，该变量记录就是最大值。

两个明确：

明确一：结果。是数组中的元素：int类型。

明确二：未知内容。数组。

示例1：通过定义变量记录较大的值的方式实现。

```
class ArrayDemo{
        public static void main(String[] args) {
              int[] arr= {89,34,-270,17,3,100};
              int max = getMax(arr);
              System.out.println("max = " + max);
        }

        public static int getMax(int[] arr){
              int maxElement = arr[0];
              for(int x = 1; x < arr.length; x++){
                    if(arr[x] > maxElement)
                         maxElement = arr[x];
             }
             return maxElement;
        }
}
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe
D:\code\day05>javac ArrayDemo.java

D:\code\day05>java ArrayDemo
max = 100
```

示例2：通过定义变量记录较大的值的索引方式实现。

```
class ArrayDemo{
      public static void main(String[] args) {
           int[] arr= {89,34,-270,17,3,100};
           int max = getMax(arr);
           System.out.println("max = " + max);
      }

      public static int getMax(int[] arr){
            int maxIndex = 0;
            for(int x = 1; x < arr.length; x++){
                  if(arr[x] > arr[maxIndex])
                       maxIndex = x;
            }
           return arr[maxIndex];
      }
}
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe
D:\code\day05>javac ArrayDemo.java

D:\code\day05>java ArrayDemo
max = 100
```

常见操作二：排序(选择排序，冒泡排序)

选择排序

思路：

1、首先拿数组第一个元素依次与除其自身外的其他每个元素顺序比较，如果第一个元素大于剩下的某个元素，就互换内容。

2、经过第一轮比较之后，此时，第一个元素就是数组中最小的元素。然后再拿第二个元素与除第一个元素和其自身的元素进行比较，如果第二个元素大于剩下的某个元素，就互换内容。此时，第二个元素就是数组中倒数第二小的元素。

3、依次类推，直到最后一个元素。

代码：

```
class ArrayDemo{
        public static void main(String[] args) {
             int[] arr= {89,34,-270,17,3,100};
             System.out.print("排序前数组：" );
             printArray(arr);
             selectSort(arr);
             System.out.print("排序后数组：" );
             printArray(arr);
        }

        public static void selectSort(int[] arr){
              for(int x = 0; x < arr.length - 1; x++){
                    for(int y = x + 1; y < arr.length; y++){
                          if(arr[x] > arr[y]){
                               int temp = arr[x];
                               arr[x] = arr[y];
                               arr[y] = temp;
                         }
                    }
             }
        }

        public static void printArray(int[] arr){
             System.out.print("[" );
             for(int x = 0; x < arr.length; x++){
                    if(x != arr.length - 1)
                         System.out.print(arr[x] + "," );
                    else
                         System.out.println(arr[x] + "]" );
             }
        }
}
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe
D:\code\day05>javac ArrayDemo.java

D:\code\day05>java ArrayDemo
排序前数组：[89,34,-270,17,3,100]
排序后数组：[-270,3,17,34,89,100]
```

P.S.

1、上面的selectSort方法之所以不用返回int数组的原因是因为：arr引用变量是对传入selectSort方法中作为参数的数组的引用，selectSort方法执行完毕之后，我们依然可以通过arr引用变量操作传入的数组。所以，没有必要再通过返回值获取。

2、上面的选择排序算法效率比较低，因为数组每一个元素与剩下的元素比较就是为了获得最小的元素并且与之互换。例如：{89,34,-270,17,3,100}这个数组，第一轮就要互换4次才能使第一个元素存储的是这个数组中最小的元素。如果是这样，那么更高效率的方式则是只需要通过两个变量，一个记录最小值，一个记录最小值所在的角标即可。等当前元素与余下的所有元素比较完，直接互换，这样只需互换一次就能达到目标，效率自然就会提高。

代码：

```
class ArrayDemo{
        public static void main(String[] args) {
              int[] arr= {89,34,-270,17,3,100};
              System.out.print("排序前数组：" );
              printArray(arr);
              selectSort(arr);
              System.out.print("排序后数组：" );
              printArray(arr);
        }

        public static void selectSort(int[] arr){
              for(int x = 0; x < arr.length - 1; x++){
                    int num = arr[x];
                    int index = x;
                    for(int y = x + 1; y < arr.length; y++){
                          if(num > arr[y]){
                               num = arr[y];
                               index = y;
                         }
                   }
                   //如果最小的就是自己，就没有必要执行swap操作
                   if(index != x)
                         swap(arr,x,index);
             }
        }

       public static void swap(int[] arr, int a,int b){
             int temp = arr[a];
             arr[a] = arr[b];
             arr[b] = temp;
        }

        public static void printArray(int[] arr){
             System.out.print("[" );
             for(int x = 0; x < arr.length; x++){
                    if(x != arr.length - 1)
                         System.out.print(arr[x] + "," );
                    else
                         System.out.println(arr[x] + "]" );
             }
        }
}
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe
D:\code\day05>javac ArrayDemo.java

D:\code\day05>java ArrayDemo
排序前数组：[89,34,-270,17,3,100]
排序后数组：[-270,3,17,34,89,100]
```

冒泡排序

思路：

1、首先在第一轮排序中，数组从第一个元素到倒数第二个元素依次与其右边的元素进行比较，如果左边的元素大于右边的元素，那么两个元素就互换。

2、经过第一轮比较，最大的元素就已经存储到数组最右边的结点中了。

3、第二轮排序则是从第一个元素到倒数第三个元素依次与其右边的元素进行比较，如果左边的元素大于右边的元素，那么两个元素就互换。

4、依照此方式，一直到只有第一和第二个元素互相比较而结束。

代码：

```
class ArrayDemo{
        public static void main(String[] args) {
         int[] arr= {89,34,-270,17,3,100};
         System.out.print("排序前数组：" );
         printArray(arr);
         bubbleSort(arr);
         System.out.print("排序后数组：" );
         printArray(arr);
        }

       public static void bubbleSort(int[] arr){
              for(int x = 0; x < arr.length - 1; x++){
                    for(int y = 0; y < arr.length - 1 -x; y++){
                          if(arr[y] > arr[y+1]){
                               int temp = arr[y];
                               arr[y] = arr[y+1];
                               arr[y+1] = temp;
                         }
                    }
             }
        }

       public static void printArray(int[] arr){
          System.out.print("[" );
          for(int x = 0; x < arr.length; x++){
             if(x != arr.length - 1)
               System.out.print(arr[x] + "," );
            else
               System.out.println(arr[x] + "]" );
          }
      }
}
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe
D:\code\day05>javac ArrayDemo.java

D:\code\day05>java ArrayDemo
排序前数组：[89,34,-270,17,3,100]
排序后数组：[-270,3,17,34,89,100]
```

在真实开发中，是不可能让我们自己去写这些排序算法的，因为JDK中已经提供好了API可以直接供我们调用。

示例：

```
import java.util.Arrays;

class ArrayDemo{
        public static void main(String[] args) {
              int[] arr= {89,34,-270,17,3,100};
              System.out.print("排序前数组：" );
              printArray(arr);
              Arrays.sort(arr);
              System.out.print("排序后数组：" );
              printArray(arr);
        }

        public static void printArray(int[] arr){
             System.out.print("[" );
              for(int x = 0; x < arr.length; x++){
                    if(x != arr.length - 1)
                         System.out.print(arr[x] + "," );
                    else
                         System.out.println(arr[x] + "]" );
             }
        }
}
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe
D:\code\day05>javac ArrayDemo.java

D:\code\day05>java ArrayDemo
排序前数组：[89,34,-270,17,3,100]
排序后数组：[-270,3,17,34,89,100]
```

常见操作三：折半查找(二分查找)

示例：简单遍历查找方式

```
class ArrayDemo{
        public static void main(String[] args) {
              int[] arr= {4,1,5,7,8,4,2};
              int index = getIndex(arr,2);
              System.out.println("index = " + index);
        }

        public static int getIndex(int[] arr, int key){
              for(int x = 0; x < arr.length; x++){
                    if(arr[x] == key)
                          return x;
              }
              return -1;
        }
}
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe
D:\code\day05>javac ArrayDemo.java

D:\code\day05>java ArrayDemo
index = 6
```

P.S.

如果一个数组是无序的，那么可以通过简单遍历查找的方式查找到某个元素所在的角标。但是如果一个数组是有序的，那么就可以通过一种更高效的方式达到相同的目的，也就是二分查找。

思路：

1、设置三个变量记录角标：min、max、mid。min初始值为0，max为数组最大角标，mid为（max+min）/2。

2、查看mid角标的元素是否与待查找的值相等，如果相等，则直接返回角标值，程序终止执行。

3、如果待查找的值小于角标为mid的元素值，那么说明待查找的元素的位置可能在min与mid角标之间。设置max = mid - 1，mid = (max + min)/2，重复第1、2步的操作。

4、如果待查找的值大于角标为mid的元素值，那么说明待查找的元素的位置可能在mid与max角标之间。设置min = mid + 1，mid = (max + min)/2，重复第1、2步的操作。

5、如果数组中不存在待查找的元素，那么按照如上流程，最终min角标值会大于max角标值，此时返回-1。

代码1：

```
class ArrayDemo{
        public static void main(String[] args) {
          int[] arr= {13,15,19,28,33,45,78,106};
          int index = binarySearch(arr,78);
          System.out.println("index = " + index);
       }

       public static int binarySearch(int[] arr, int key){
            int max,min,mid;
            min = 0;
            max =arr. length - 1;
            mid = (max + min)/2;

             while(arr[mid] !=key){
                  if(key > arr[mid])
                       min = mid + 1;
                  else if (key < arr[mid])
                       max = mid - 1;
                  if(max < min)
                        return -1;
                 mid = (max + min)/2;
            }
            return mid;
      }
}
```

复制代码

运行结果：

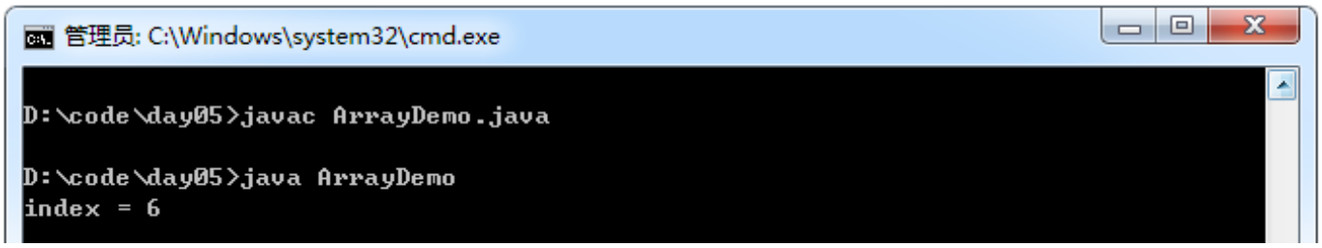
```
管理员: C:\Windows\system32\cmd.exe
D:\code\day05>javac ArrayDemo.java

D:\code\day05>java ArrayDemo
index = 6
```

代码2：

```
class ArrayDemo{
        public static void main(String[] args) {
            int[] arr= {13,15,19,28,33,45,78,106};
            int index = binarySearch(arr,78);
            System.out.println("index = " + index);
         }

        public static int binarySearch(int[] arr, int key){
             int max,min,mid;
             min = 0;
             max = arr. length - 1;

              while(min <= max){
                   mid = (max + min) >> 1;

                    if(key > arr[mid])
                         min = mid + 1;
                    else if (key < arr[mid])
                         max = mid - 1;
                    else
                          return mid;
             }
              return -1;
       }
}
```

复制代码

运行结果：



P.S.

给定一个有序的数组，如果往该数组中存储一个元素，并保证这个数组还是有序的，那么这个元素的存储的角标如何获取？

可以先通过二分查找，返回min的值，然后将待插入元素存在角标为min的数组位置，数组中角标为min以及比min大的角标所在的数组元素全部往后顺延一个位置。

代码：

```
class ArrayDemo{
        public static void main(String[] args) {
          int[] arr= {13,15,19,28,33,45,78,106};
          int index = binarySearch(arr,44);
          System.out.println("index = " + index);
        }

        public static int binarySearch(int[] arr, int key){
             int max,min,mid;
             min = 0;
             max = arr. length - 1;

              while(min <= max){
                   mid = (max + min) >> 1;

                    if(key > arr[mid])
                         min = mid + 1;
                    else if (key < arr[mid])
                         max = mid - 1;
                    else
                          return mid;
             }
             return min;
        }
}
```

复制代码

运行结果：

说明：由上面的结果可以看到，如果要向数组{13,15,19,28,33,45,78,106}中插入值为44的元素，那么应该插入的位置角标是5，角标为5以及其后的元素都应该往后顺延一个位置。

P.S.

在实际开发中，二分查找也不需要我们自己去写，JDK中已经提供了相关的API供调用。

示例：

```
import java.util.Arrays;

class ArrayDemo{
        public static void main(String[] args) {
          int[] arr= {13,15,19,28,33,45,78,106};
          //如果存在返回的具体的角标位置，不存在返回的是-插入点-1
          int index = Arrays.binarySearch(arr,44);
          System.out.println("index = " + index );
        }
}
```

复制代码

运行结果：

说明：返回的是-6而不是5的原因可以通过API文档查找到.

binarySearch

```
public static int binarySearch(int[] a,
                               int key)
```

使用二分搜索法来搜索指定的 int 型数组，以获得指定的值。必须在进行此调用之前对数组进行排序（通过 sort(int[]) 方法）。如果没有对数组进行排序，则结果是不确定的。如果数组包含多个带有指定值的元素，则无法保证找到的是哪一个。

参数：

a - 要搜索的数组
key - 要搜索的值

返回：

如果它包含在数组中，则返回搜索键的索引；否则返回 (-(*插入点*) - 1)。*插入点* 被定义为将键插入数组的那一点：即第一个大于此键的元素索引，如果数组中的所有元素都小于指定的键，则为 a.length。注意，这保证了当且仅当此键被找到时，返回的值将 >= 0。

练习：获取一个十进制整数的2、8、16进制表现形式。

注意：

1、什么时候使用数组呢？

2、如果数据出现了对应关系，而且对应关系的一方是有序的数字编号，并作为角标使用，这时候必须要想到数组的应用。

3、将这些数据存储到数组中，根据运算的结果作为角标直接去查数组中对应的元素即可，这种方式称为查表法。

思路：

1、首先判断如果传入的十进制数为0，那么它的2、8、16进制都是0，直接返回0，不需要再执行余下的程序。

2、如下面的示意图中所示，以将十进制数转换成十六进制数为例：

将60与15进行与操作，其值就是60的十六进制的最低位。

再将60无符号右移4位，再与15进行与操作，其值就是60的十六进制的倒数第二位。

3、由上面的例子可以总结出，将一个十进制数转换为十六进制的步骤就是：

将十进制数与15相与，将结果存储到一个数组的最低位。

然后将十进制数右移4位，再与15进行与操作，其值就是该数对应的十六进制的倒数第二位。

再右移4位，与15相与...直到相与结果为0为止。

4、进而可以推理得到，10进制转换为2和8进制的规律与转换为16进制很相似，只是偏移量和相与的数字不同而已。

10进制转换为2进制的偏移量为1，相与数字为1。

10进制转换为8进制的偏移量为3，相与数字为7。

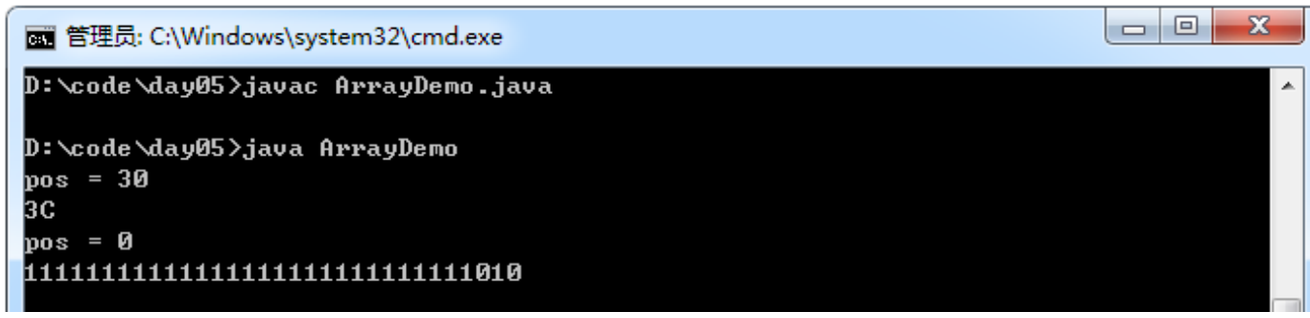
答案：

```
import java.util.Arrays;

class ArrayDemo{
        public static void main(String[] args) {
          toHex(60);
          toBin(-6);
        }

        //十进制-->二进制
        public static void toBin(int num){
          trans(num,1,1);
        }

        //十进制-->十六进制
        public static void toHex(int num){
          trans(num,15,4);
        }

        //十进制-->八进制
        public static void toOctal(int num){
          trans(num,7,3);
        }

        //进制转换的通用方法
        public static void trans(int num, int base,int offset){
            if(num == 0){
                 System.out.println("0" );
                 return;
            }
            char[] chs = {'0','1' ,'2' ,'3' ,'4' ,'5' ,'6' ,'7' ,'8' ,'9' ,'A' ,'B' ,'C' ,'D' ,'E' ,'F' };

            char[] arr = new char[32];
            int pos = arr.length ;
            while(num != 0){
                  int temp = num & base;
                  arr[--pos] = chs[temp];
                  num = num >>> offset;
            }

            System.out.println("pos = " + pos);
            for(int x = pos; x < arr.length; x++){
                  System.out.print(arr[x]);
            }
            System.out.println();
        }
```

```
46.    }
```

复制代码

运行结果：



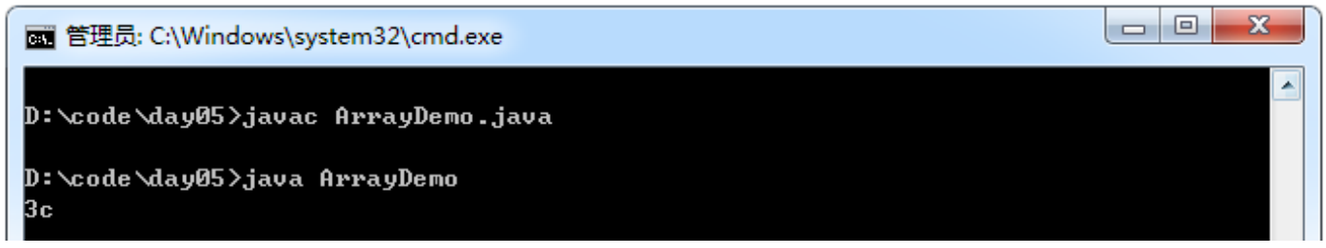
在真实开发中，进制转换也不需要我们写，JDK已经通过API的方式提供给我们，直接调用即可。

示例：

```
01.    class ArrayDemo{
02.            public static void main(String[] args) {
03.                System.out.println(Integer.toHexString(60));
04.            }
05.    }
```

复制代码

运行结果：



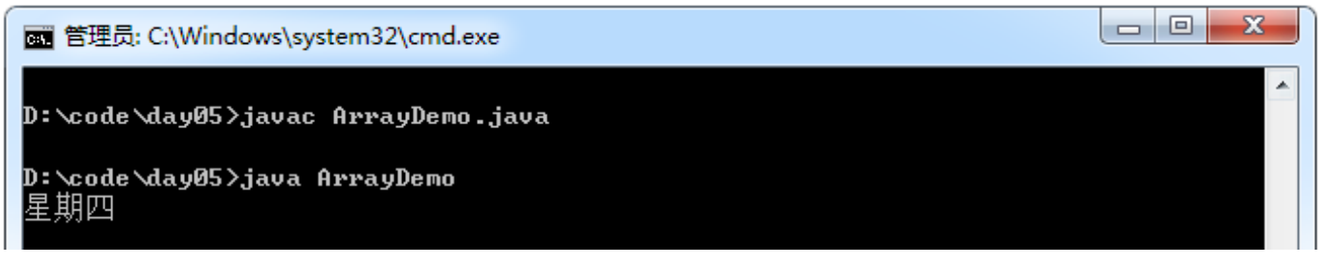
示例（查表法）：

```
01.    class ArrayDemo{
02.            public static void main(String[] args) {
03.                String week = getWeek(4);
04.                System.out.println(week);
05.            }
06.
07.            public static String getWeek(int num){
08.                 if(num > 7 || num < 1){
09.                        return "错误的星期" ;
10.                 }
```

```
        String[] weeks = { "","星期一" ,"星期二" ,"星期三" ,"星期四" ,"星期五" ,"星期六" ,"星期日" };

        return weeks[num];
    }
}
```

复制代码

运行结果：



~END~

~爱上海，爱黑马~