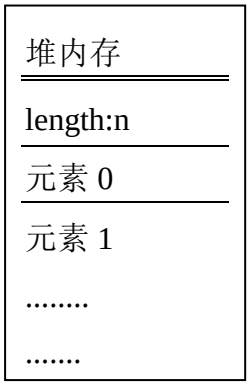
纲要

a) 数组概要
b) 一维数组的声明和使用
c) 二维数据的声明和使用
d) 数组的排序
e) 数组的查找
f) Arrays 工具类

1.内容

1.1 数组概要

数组是一种引用数据类型，在内存中存储示意图如下：



1. 数组是一组数据的集合
2. 数组作为一种引用类型
3. 数组元素的类型可以是基本类型，也可以是引用类型，但同一个数组只能是同一种类型
4. 数组作为对象，数组中的元素作为对象的属性，除此之外数组还包括一个成员属性 length，length 表示数组的长度
5. 数组的长度在数组对象创建后就确定了，就无法再修改了
6. 数组元素是有下标的，下标从 0 开始，也就是第一个元素的下标为 0，依次类推最后一个

元素的下标为 n-1，我们可以通过数组的下标来访问数组的元素

1.2 一维数组的声明和使用

1.2.1 数组的声明

一维数组的声明格式有以下两种：

1. 数组元素的类型[] 变量名称
2. 数组元素的类型 变量名称[]

数组元素的类型，可以是 java 中的任意类型，变量名称可以是任意合法的标识符，上面两种格式较常用的是第一种，例如：

int [] a;

Student[] stu

在一行中也可以声明多个数组，例如：

int[] a, b, c

1.2.2 数组的创建

数组有两种创建方式

- 第一种，使用 new 操作符来创建数组，格式为：new 数组元素的数据类型[数组元素的个数]

1. 基本类型的数组

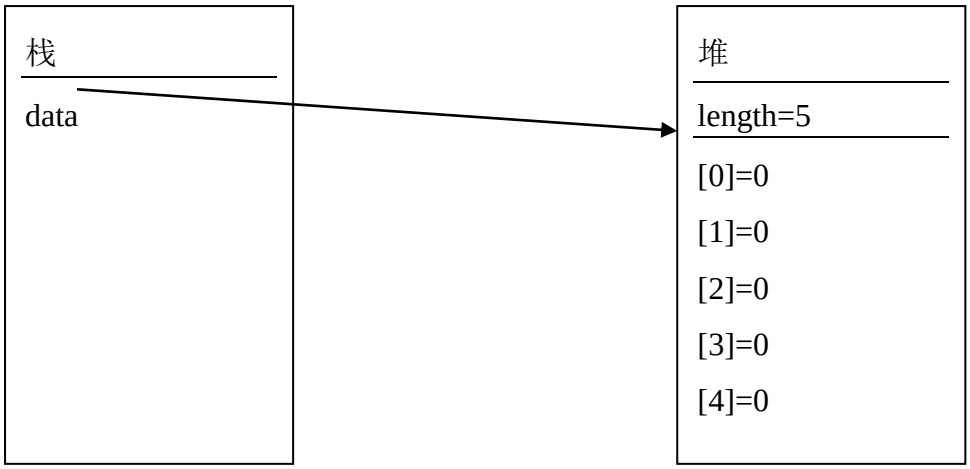
```
public class ArrayTest01 {


    public static void main(String[] args) {


        //声明 int 类型的数组，长度为 5
        //数组中的元素必须为 int 类型
        int[] data = new int[5];


        //对数组中的元素进行赋值，如果不赋值默认为该类型的默认值，以上数组默认为 0
        //如何赋值？变量名[下标],下标从 0 开始
```



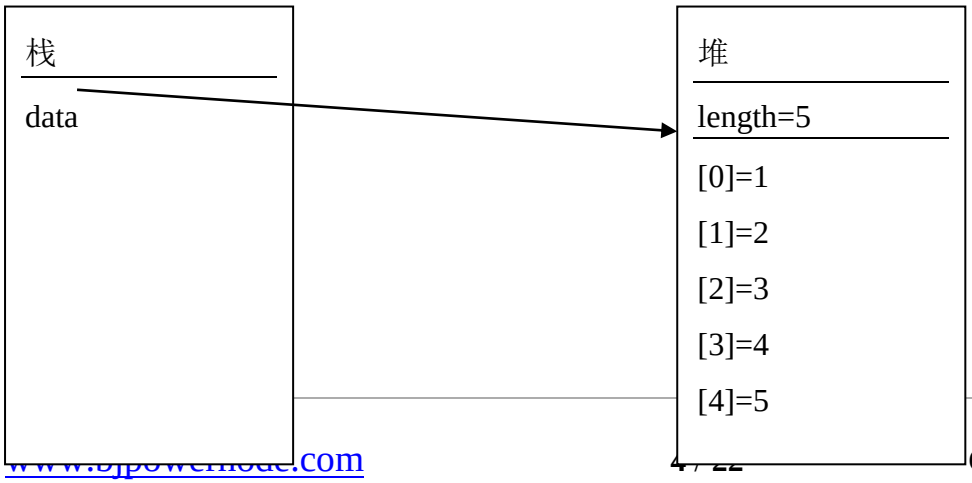
```
        data[0] = 1;
        data[1] = 2;
        data[2] = 3;
        data[3] = 4;
        data[4] = 5;

        //输出数组中的元素，变量名[下标]
        System.out.println(data[0]);
        System.out.println(data[1]);
        System.out.println(data[2]);
        System.out.println(data[3]);
        System.out.println(data[4]);

        System.out.println("----------------------");
        //采用 length 属性可以取得数组的长度
        for (int i=0; i<data.length; i++) {
            System.out.println(data[i]);
        }

        //输出指定的数组元素
        System.out.println("data[3]=" + data[3]);

        //会抛出 ArrayIndexOutOfBoundsException 异常
        //数组下标越界
        System.out.println("data[10]=" + data[10]);

        //不能成功赋值，数组中的类型必须是一种类型
        //data[0] = "iiii";
    }
}
```

内存结构

```
int[] data = new int[5];
```




```
data[0] = 1;
data[1] = 2;
data[2] = 3;
data[3] = 4;
data[4] = 5;
```



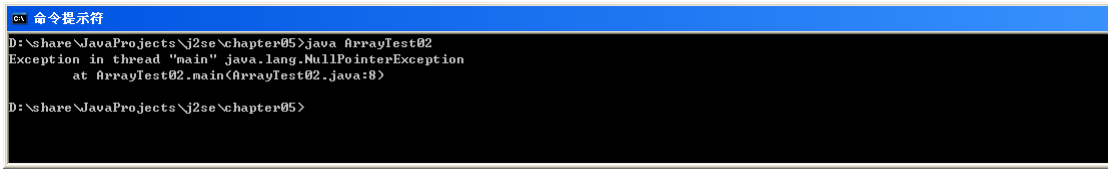
必须清楚数组为引用类型，它在堆中分配

2. 引用类型的数组

【示例代码】

```
public class ArrayTest02 {

    public static void main(String[] args) {

        //声明引用类型的数组
        Student[] student = new Student[2];

        //出现空指针
        //因为引用类型的数组，它采用 null 作为默认的初始化值
        student[0].id = 1001;
        student[0].name = "张三";

        student[1].id = 1002;
        student[1].name = "李四";


    }
}

class Student {

    int id;

    String name;
```

```
}
```



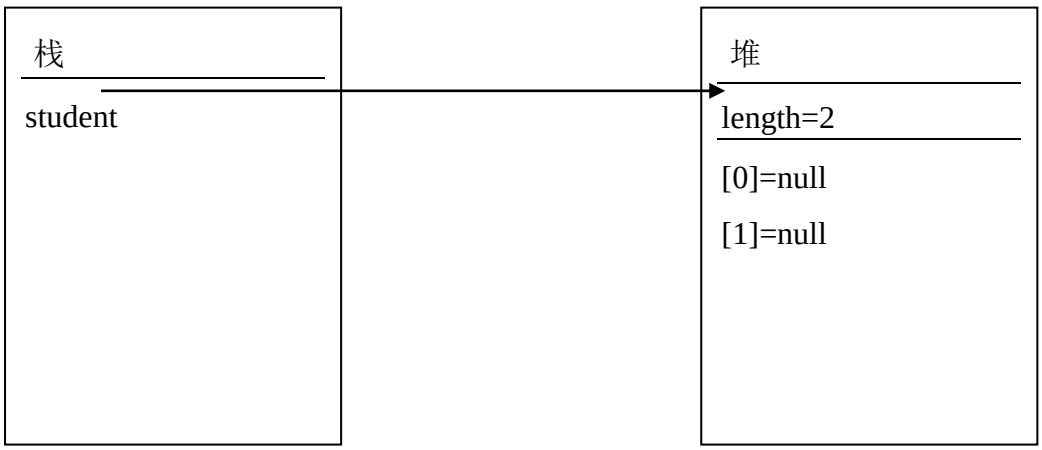
修正空指针

```
public class ArrayTest03 {

    public static void main(String[] args) {

        //声明引用类型的数组
        Student[] student = new Student[2];

        //初始数组元素为 Student 对象
        /*
        student[0] = new Student();
        student[0].id = 1001;
        student[0].name = "张三";

        student[1] = new Student();
```

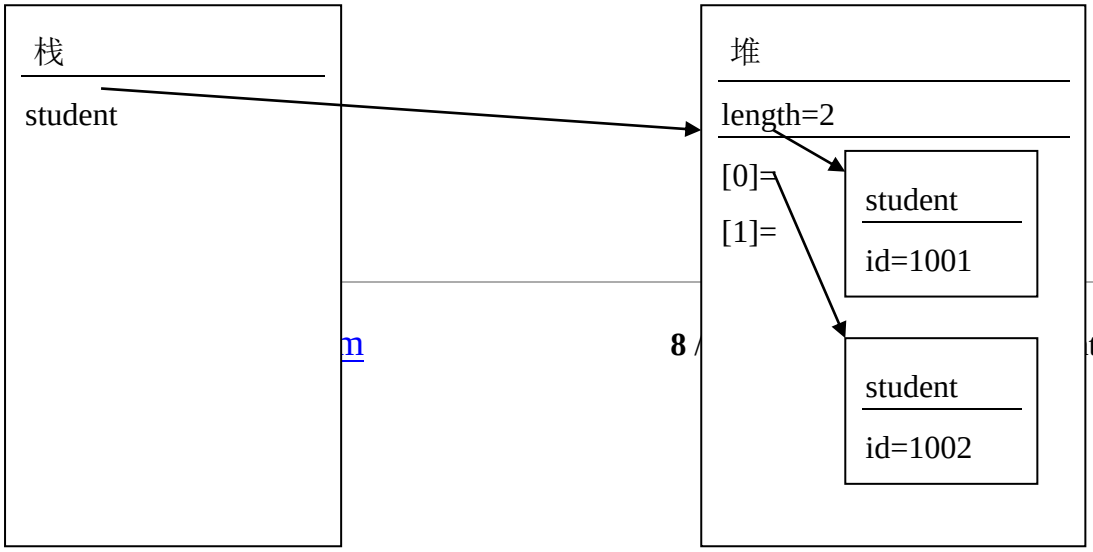
```
        student[1].id = 1002;
        student[1].name = "李四";
        */

        //可以采用如下方式赋值
        Student zhangsan = new Student();
        zhangsan.id = 1001;
        zhangsan.name = "张三";
        student[0] = zhangsan;

        Student lisi = new Student();
        lisi.id = 1002;
        lisi.name = "李四";
        student[1] = lisi;

        for (int i=0; i<student.length; i++) {
            System.out.println("id=" + student[i].id + ", name=" + student[i].name);
        }
    }
}

class Student {

    int id;

    String name;

}
```



```
Student[] student = new Student[2];
```




```
Student zhangsan = new Student();
zhangsan.id = 1001;
zhangsan.name = "张三";
student[0] = zhangsan;

Student lisi = new Student();
lisi.id = 1002;
lisi.name = "李四";
student[1] = lisi;
```



- 第二种，使用数组的初始化语句，格式为：数组元素的类型[] 变量名称 = {数组元素 1，数组元素 2,.....数组元素 n}或数组元素的类型 变量名称[] = {数组元素 1，数组元素 2,.....数组元素 n}

```
public class ArrayTest04 {

    public static void main(String[] args) {
        //静态初始化
        int[] data = {1, 2, 3, 4, 5};

        for (int i=0; i<data.length; i++) {
            System.out.println(data[i]);
        }

        Student zhangsan = new Student();
        zhangsan.id = 1001;
        zhangsan.name = "张三";

        Student lisi = new Student();
        lisi.id = 1002;
        lisi.name = "李四";

        //静态初始化
```



```
        Student[] students = {zhangsan, lisi};
        for (int i=0; i<students.length; i++) {
            System.out.println("id=" + students[i].id + ", name=" + students[i].name);
        }

    }
}

class Student {

    int id;

    String name;

}
```

1.3 二维数组的声明和使用

1	2	3
4	5	6

二维数组属于多维数组，那么什么是多维数组呢，当数组元素的类型是数组时就成了多维数组，二维数组的声明格式如下：

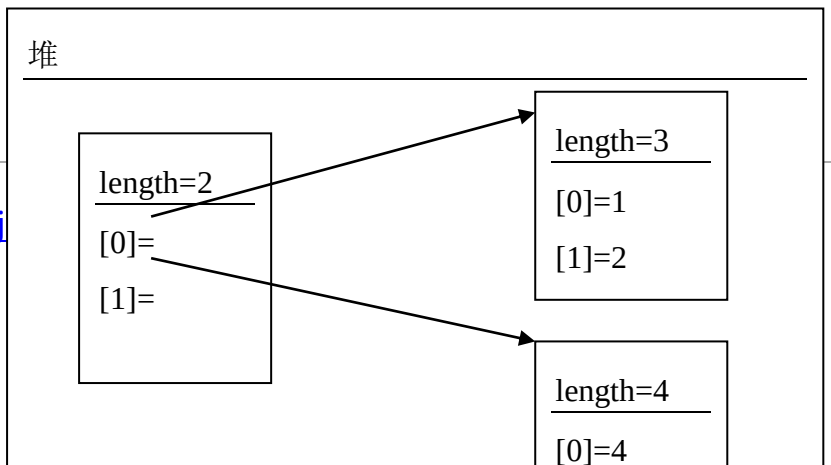
1. 数组元素的数据类型[][] 变量名；
2. 数组元素的数据类型 变量名[][]；

其中方括号的个数就是数组的维数，声明二维数组如下：

```
int [][] data;
```

在这里介绍三种二维数组的创建方式

1. **采用 new 关键字直接创建**

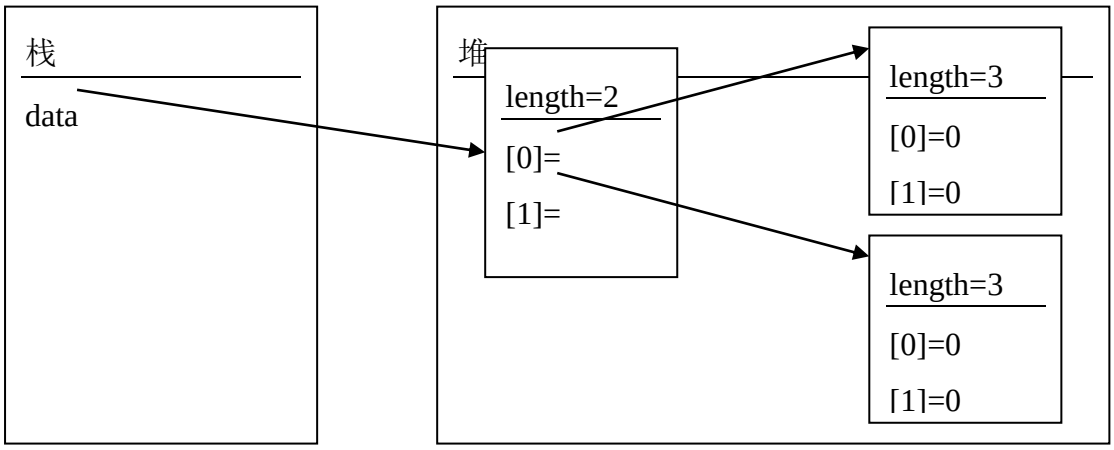


```
public class ArrayTest05 {

    public static void main(String[] args) {

        //声明二维数组
        int[][] data = new int[2][3];

        //对二维数组赋值
        data[0][0] = 1;
        data[0][1] = 2;
        data[0][2] = 3;

        data[1][0] = 4;
        data[1][1] = 5;
        data[1][2] = 6;

        //输出二维数组
```



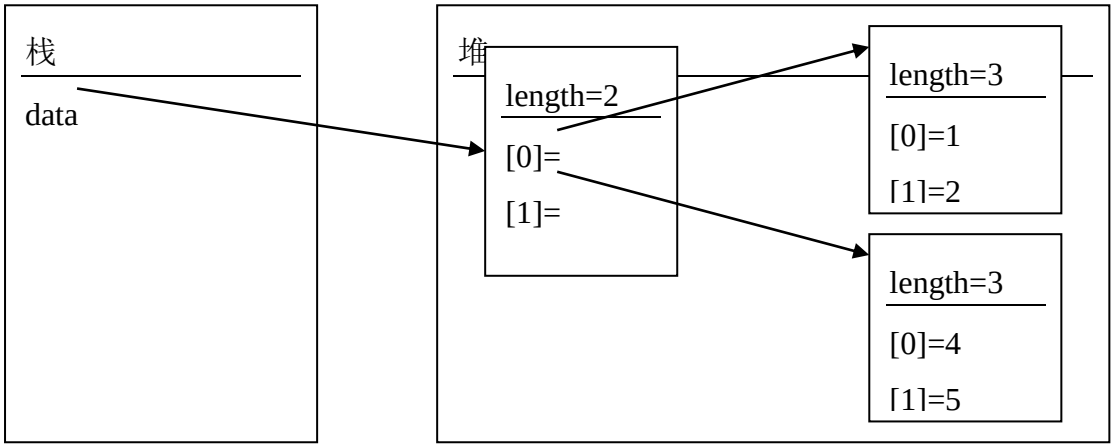
```
        for (int i=0; i<data.length; i++) {
            for (int j=0; j<data[i].length; j++) {
                System.out.println(data[i][j]);
            }
        }

    }
}
```

```
int[][] data = new int[2][3];
```




```
        //对二维数组赋值
        data[0][0] = 1;
        data[0][1] = 2;
        data[0][2] = 3;

        data[1][0] = 4;
```

```
        data[1][1] = 5;
        data[1][2] = 6;
```



2. 从高维开始逐维创建

```
public class ArrayTest06 {

    public static void main(String[] args) {

        //从高维开始逐维创建
        int[][] data = new int[2][];
        data[0] = new int[2];
        data[1] = new int[4];

        data[0][0] = 1;
        data[0][1] = 2;

        data[1][0] = 1;
        data[1][1] = 2;
        data[1][2] = 3;
```



```
        data[1][3] = 4;

        //输出二维数组
        for (int i=0; i<data.length; i++) {
            for (int j=0; j<data[i].length; j++) {
                System.out.println(data[i][j]);
            }
        }

    }
}
```

3. 采用初始化语句块创建数组对象

```
public class ArrayTest07 {

    public static void main(String[] args) {

        //静态初始化
         // 多个数组之间用逗号隔开
        int[][] data = {{1,2},{1,2,3,4}};

        for (int i=0; i<data.length; i++) {
            for (int j=0; j<data[i].length; j++) {
                System.out.println(data[i][j]);
            }
        }

    }
}
```


1.4 数组的排序

1.4.1 冒泡排序

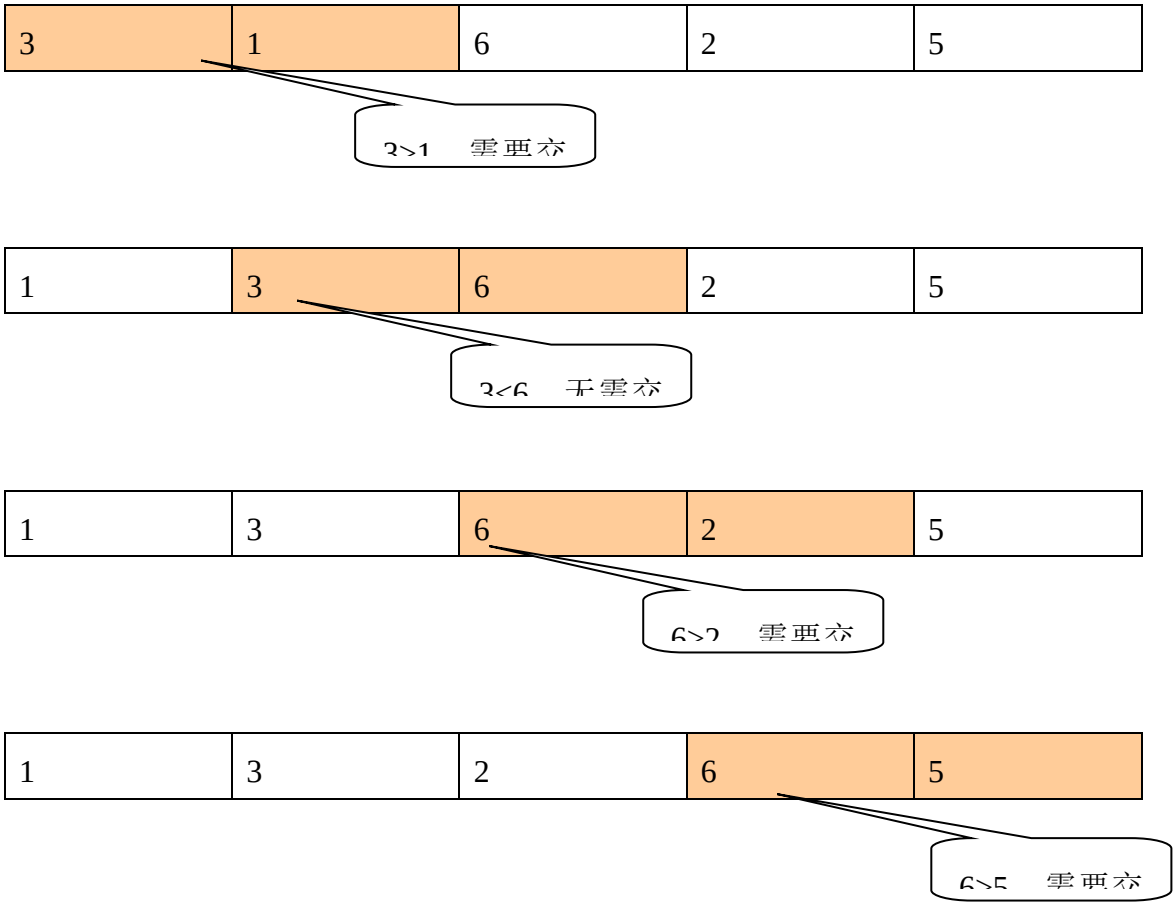
假设有 5 个数字 3，1，6，2，5 在一个 int 数组中，要求按从小到大排序输出

如何采用冒泡排序算法呢？

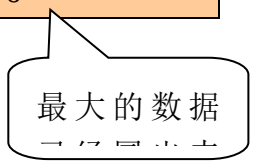
冒泡排序的算法是这样的，首先从数组的最左边开始，取出第 0 号位置（左边）的数据和第 1 号位置（右边）的数据，如果左边的数据大于右边的数据，则进行交换，否而不进行交换。接下来右移一个位置，取出第 1 个位置的数据和第 2 个位置的数据，进行比较，如果左边的数据大于右边的数据，则进行交换，否而不进行交换。沿着这个算法一直排序下去，最大的数就会冒出水面，这就是冒泡排序。

以上示例排序过程如下：

第一遍排序



1	3	2	5	6



从上面我们看到了比较了 N-1 次，那么第二遍就为 N-2 次比较了，如此类推，比较次数的公式如下：

(N-1) + (N-2)+...+1=((N-1)*N)/2

所以以上总共比较次数为((5-1)*5)/2=10

以上就是冒泡排序算法

```
public class ArraySortTest01 {

    public static void main(String[] args) {

        int[] data = {3,1,6,2,5};
        for (int i=data.length-1; i>0; i--) {
            for (int j=0; j<i; j++) {
                if (data[j] > data[j+1]) {
                    int temp = data[j];
                    data[j]   = data[j+1];
                    data[j+1] = temp;
                }
            }
        }
        for (int i=0; i<data.length; i++) {
            System.out.println(data[i]);
        }
    }
}
```

1.4.2 选择排序

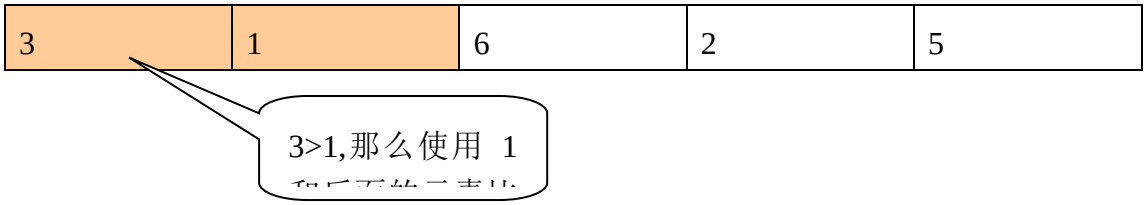
选择排序对冒泡排序进行了改进，使交换次数减少，但比较次数仍然没有减少。

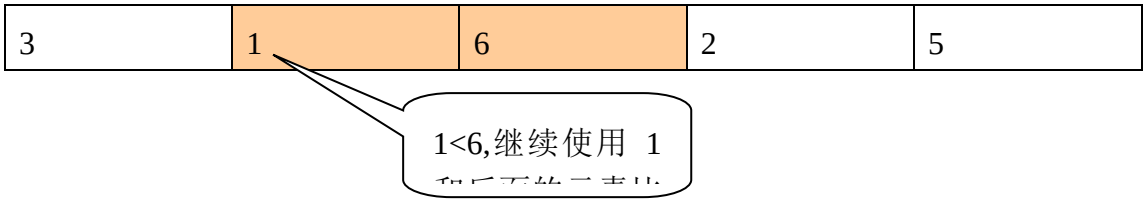
假设有 5 个数字 3，1，6，2，5 在一个 int 数组中，要求按从小到大排序输出

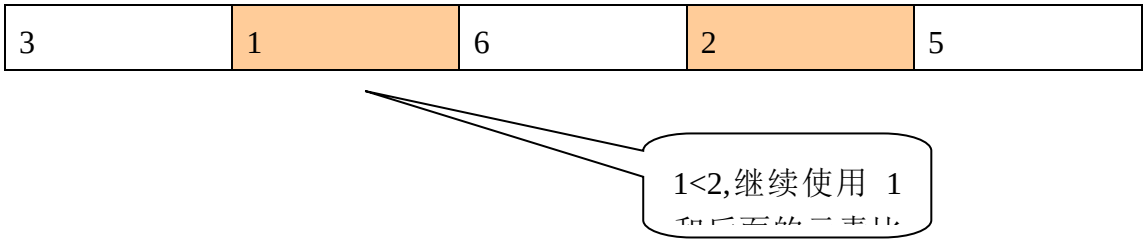
采用选择排序，选择排序是这样的，先从左端开始，找到下标为 0 的元素，然后和后面的元素依次比较，如果找到了比下标 0 小的元素，那么再使用此元素，再接着依次比较，直到比较完成所有的元素，最后把最小的和第 0 个位置交换。

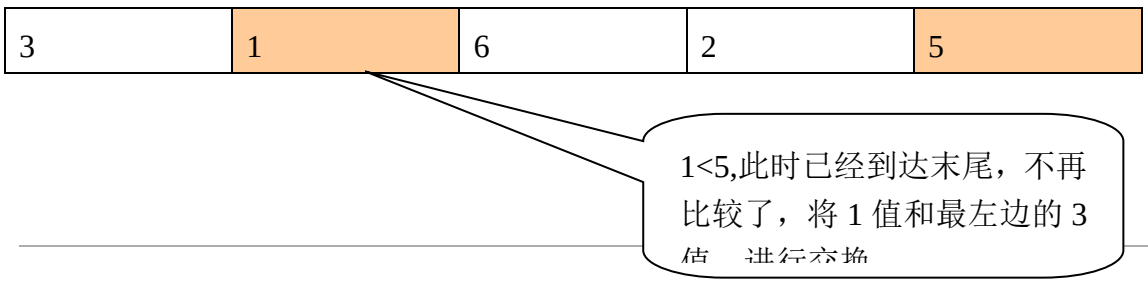
以上示例排序过程如下：

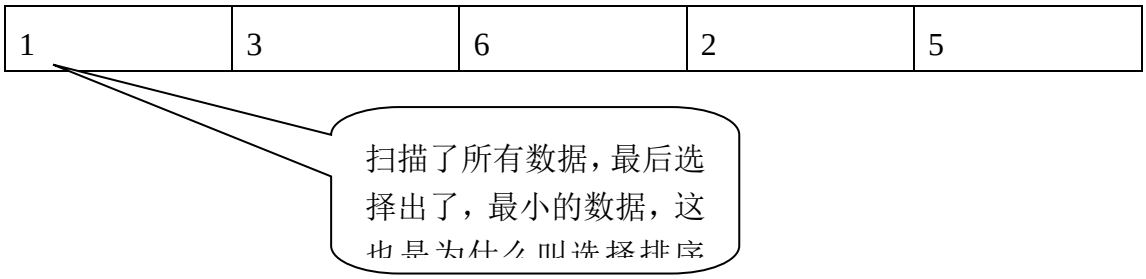
第一遍排序











第二遍排序将从下标为 1 的元素开始，以此类推，经过 N(N-1)/2 次比较，经过 N 次数据交互就完成了所有元素的排序。

【示例代码】

```
public class ArraySortTest02 {

    public static void main(String[] args) {

        int[] data = {3,1,6,2,5};
        for (int i=0; i<data.length; i++) {
            int min = i;
            for (int j=i+1; j<data.length; j++) {
                if (data[j] < data[min]) {
                    min = j;
                }
            }
            //进行位置的交换
            if (min != i) {
                int temp = data[i];
                data[i] = data[min];
```



```
                    data[min] = temp;
                }

            }
            for (int i=0; i<data.length; i++) {
            System.out.println(data[i]);
            }
        }
}
```

1.5 数组的搜索

1.5.1 二分法（折半法）查找

查找数组中的元素我们可以遍历数组中的所有元素，这种方式称为线性查找。线性查找适合与小型数组，大型数组效率太低。如果一个数组已经排好序，那么我们可以采用效率比较高的二分查找或叫折半查找算法。

见示例

数值	11	12	13	14	15	16	17	18	19	20
下标	0	1	2	3	4	5	6	7	8	9

假设，我们准备采用二分法取得 18 在数组中的位置

- 第一步，首先取得数组 0~9 的中间元素

  中间元素的位置为：（开始下标 0 + 结束下标 9）/2=下标 4

  通过下标 4 取得对应的值 15

  18 大于 15，那么我们在后半部分查找

- 第二步，取数组 4~9 的中间元素

  4~9 的中间元素=（下标 4 + 1 +下标 9）/2=下标 7

  下标 7 的值为 18，查找完毕，将下标 7 返回即可

以上就是二分或折半查找法，此种方法必须保证数组事先是排好序的，这一点一定要注意

【示例代码】

```
public class BinarySearchTest01 {

    public static void main(String[] args) {
        int[] data = {11,12,13,14,15,16,17,18,19,20};
        int index = binarySearch(data, 18);
        System.out.println(index);
    }

    //采用折半法查询，必须建立在排序的基础上
    private static int binarySearch(int[] data, int value) {
        //开始下标
        int beginPos = 0;
        //结束下标
        int endPos = data.length - 1;

        while (beginPos <=endPos) {
            int midPos = (beginPos + endPos)/2;
            if (value == data[midPos]) {
                return midPos;
            }else if (value > data[midPos]) {
                beginPos = midPos + 1;
            }else if (value < data[midPos]) {
                endPos = midPos - 1;
            }
        }
        return -1;
    }
}
```

1.6 Arrays 工具类

了解 sort、fill 和 binarySearch

1.6.1 Arrays.sort 的使用

```
import java.util.Arrays;

public class ArraysUtilTest01 {

    public static void main(String[] args) {
        int[] data = {3,1,6,2,5};
        Arrays.sort(data);
        for (int i=0; i<data.length; i++) {
            System.out.println(data[i]);
        }
        System.out.println("----------------");
        for (int i=data.length-1; i>=0; i--) {
            System.out.println(data[i]);
        }
    }
}
```

1.6.2 Arrays.binarySearch 的使用

```
import java.util.Arrays;

public class ArraysUtilTest02 {

    public static void main(String[] args) {
        int[] data = {3,1,6,2,5};
```



```
        Arrays.sort(data);
        for (int i=0; i<data.length; i++) {
            System.out.println(data[i]);
        }
        System.out.println("");

        int index = Arrays.binarySearch(data, 3);
        System.out.println("index=" + index);

    }
}
```

