常用API

什么是API?

```
public class Demo {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        if(a > b){
            System.out.println(a);
        }else{
            System.out.println(b);
        }
    }
}
```

什么是API?

```
public class Demo {
    public static void main(String[] args) {
        int max = getMax(10,20);
        System.out.println(max);
    }

    public static int getMax(int a, int b){
        if(a > b){
            return a;
        }else{
            return b;
        }
    }
```

什么是API?

API(Application Programming interface) 应用程序接口

简单来说：就是Java帮我们已经写好的一些方法，我们直接拿过来用就可以了。

目录 Contents

Math 类概述

Math 包含执行基本数字运算的方法

没有构造方法，如何使用类中的成员呢?

看类的成员是否都是静态的，如果是，通过类名就可以直接调用

Math

Math 类的常用方法

方法名	说明
public static int abs(int a)	返回参数的绝对值
public static double ceil(double a)	向上取整
public static double floor(double a)	向下取整
public static int round(float a)	四舍五入
public static int max(int a,int b)	返回两个int值中的较大值
public static int min(int a,int b)	返回两个int值中的较小值
public static double pow(double a,double b)	返回a的b次幂的值
public static double random()	返回值为double的正值，[0.0,1.0)

目录 Contents

System 类概述

System 不能被实例化

System 类的常用方法

方法名	说明
public static void exit(int status)	终止当前运行的 Java 虚拟机，非零表示异常终止
public static long currentTimeMillis()	返回当前时间(以毫秒为单位)
arraycopy(数据源数组, 起始索引, 目的地数组, 起始索引, 拷贝个数)	数组copy

目录 Contents

Object 类的概述

每个类都可以将 Object 作为父类。所有类都直接或者间接的继承自该类

构造方法：public Object()

回想面向对象中，为什么说子类的构造方法默认访问的是父类的无参构造方法?
因为它们的顶级父类只有无参构造方法

结论

1，Object类是所有类的直接或者间接父类
2，直接打印一个对象就是打印这个对象的toString方法的返回值
3，Object类的toString方法得到的是对象的地址值
4，我们一般会对toString方法进行重写

Object 类的常用方法

方法名	说明
public String toString()	返回对象的字符串表示形式。建议所有子类重写该方法，自动生成
public boolean equals(另一个对象)	比较对象是否相等。默认比较地址，重写可以比较内容，自动生成

Object

面试题

String s = “abc”;

StringBuilder sb = new StringBuilder(“abc”);

s.equals(sb);

sb.equals(s);

目录 Contents

Objects

Objects 类的常用方法

方法名	说明
public static String toString(对象)	返回参数中对象的字符串表示形式。
public static String toString(对象, 默认字符串)	返回对象的字符串表示形式。
public static Boolean isNull(对象)	判断对象是否为空
public static Boolean nonNull(对象)	判断对象是否不为空

目录 Contents

BigDecimal 类的构造方法

方法名	说明
BigDecimal(double val)	参数为double
BigDecimal(String val)	参数为String

BigDecimal 类的常用方法

作用：可以用来精确计算

方法名	说明
public BigDecimal add(另一个BigDecimal对象)	加法
public BigDecimal subtract (另一个BigDecimal对象)	减法
public BigDecimal multiply (另一个BigDecimal对象)	乘法
public BigDecimal divide (另一个BigDecimal对象)	除法
public BigDecimal divide (另一个BigDecimal对象，精确几位，舍入模式)	除法

结论

1， BigDecimal是用来进行精确计算的

2，创建BigDecimal的对象，构造方法使用参数类型为字符串的。

3，四则运算中的除法，如果除不尽请使用divide的三个参数的方法。

代码示例：

BigDecimal divide = bd1.divide(参与运算的对象, 小数点后精确到多少位, 舍入模式);

参数1，表示参与运算的BigDecimal 对象。

参数2，表示小数点后面精确到多少位

参数3，舍入模式

BigDecimal.ROUND_UP 进一法

BigDecimal.ROUND_FLOOR 去尾法

BigDecimal.ROUND_HALF_UP 四舍五入

目录 Contents

基本类型包装类

基本类型包装类概述

将基本数据类型封装成对象的好处在于可以在对象中定义更多的功能方法操作该数据

常用的操作之一：用于基本数据类型与字符串之间的转换

“123”→123

基本数据类型	包装类
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

基本类型包装类

Integer 类的概述和使用

Integer：该对象中包装了一个基本数据类型int的值

方法名	说明
public Integer(int value)	根据 int 值创建 Integer 对象**(过时)**
public Integer(String s)	根据 String 值创建 Integer 对象**(过时)**
public static Integer valueOf(int i)	返回表示指定的 int 值的 Integer 实例
public static Integer valueOf(String s)	返回一个保存指定值的 Integer 对象 String

基本类型包装类

自动装箱和自动拆箱

- 装箱：把基本数据类型转换为对应的包装类类型
- 拆箱：把包装类类型转换为对应的基本数据类型

```
Integer i = 100;  // 自动装箱
i += 200;         // i = i + 200;  i + 200 自动拆箱; i = i + 200; 是自动装箱
```

注意：在使用包装类类型的时候，如果做操作，最好先判断是否为 null

我们推荐的是，**只要是对象，在使用前就必须进行不为 null 的判断**

基本类型包装类

Integer的成员方法

方法名	说明
static int parseInt(String s)	将字符串类型的整数变成int类型的整数

基本类型包装类

int 和 String 的相互转换

基本类型包装类的最常见操作就是：用于基本类型和字符串之间的相互转换

1. int 转换为 String

方式一：加双引号即可

方式二：public static String **valueOf(int i)**：返回 int 参数的字符串表示形式。该方法是 String 类中的方法

2. String 转换为 int

public static int **parseInt(String s)**：将字符串解析为 int 类型。该方法是 Integer 类中的方法

基本类型包装类

案例：字符串中数据的处理

需求：有一个字符串：“91 27 46 38 50”，把其中的每一个数存到int类型的数组中

思路：

① 定义一个字符串

② 把字符串中的数字数据存储到一个int类型的数组中

③ 遍历数组输出结果

基本类型包装类

案例：字符串中数据排序

需求：有一个字符串：“91 27 46 38 50”，把其中的每一个数存到int类型的数组中

思路：

① 定义一个字符串

② **把字符串中的数字数据存储到一个int类型的数组中**

③ 遍历数组输出结果

基本类型包装类

案例：字符串中数据排序

需求：有一个字符串：“91 27 46 38 50”，把其中的每一个数存到int类型的数组中

思路：

① 定义一个字符串

② 把字符串中的数字数据存储到一个int类型的数组中

两个难点：如何获取到里面的每一个值

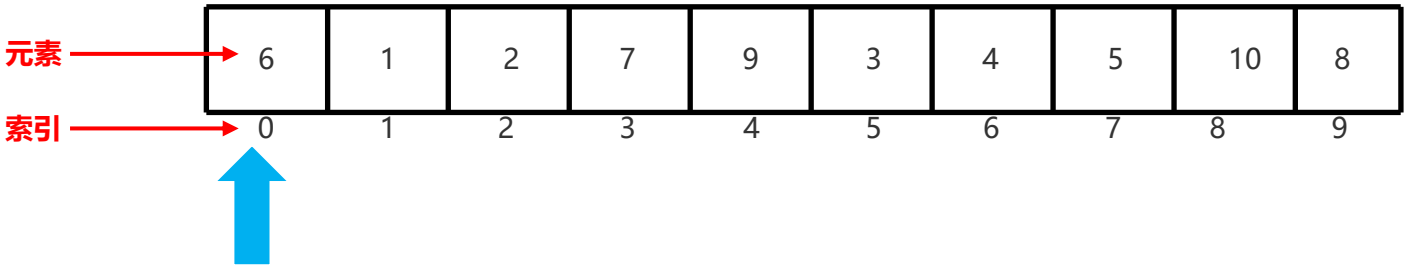
如何把每一值变成int类型再存入数组

③ 遍历数组输出结果

目录 Contents

数组的高级操作

基本查找

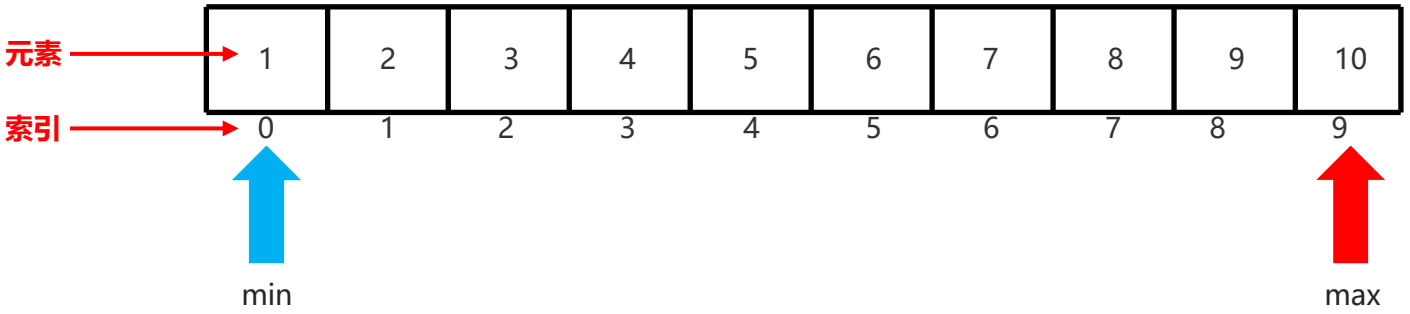


需求：我要查找数组中的3在哪个索引?

数组的高级操作

二分查找

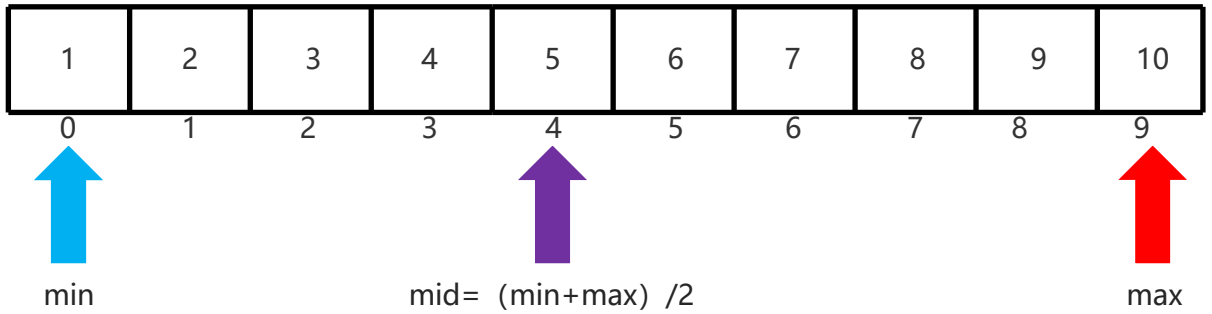
min和max表示查找的范围



需求：我要查找数组中的3在哪个索引?

数组的高级操作

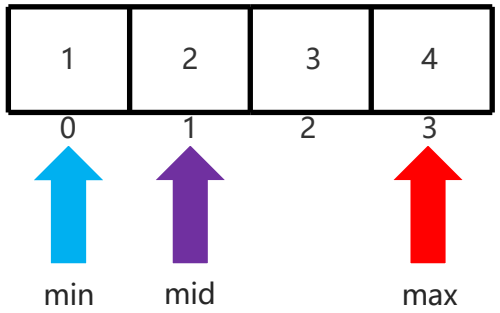
二分查找



数组的高级操作

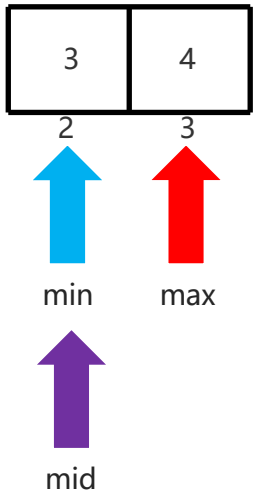
二分查找

二分查找相当于每次去掉一半的查找范围



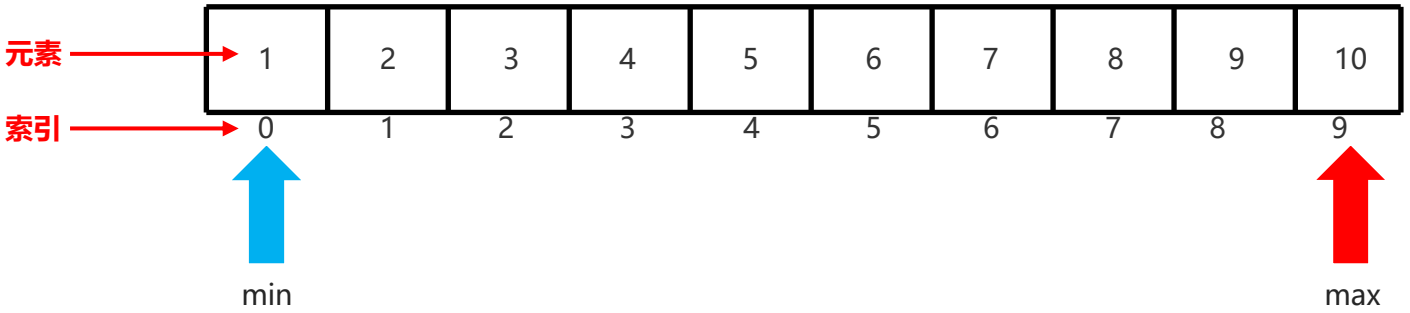
数组的高级操作

二分查找



数组的高级操作

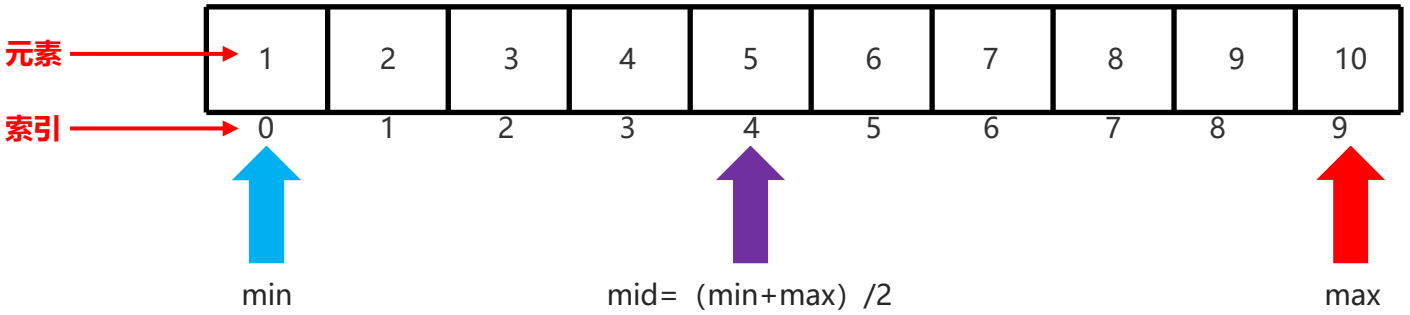
二分查找 --- 元素不存在



需求：我要查找数组中的11在哪个索引?

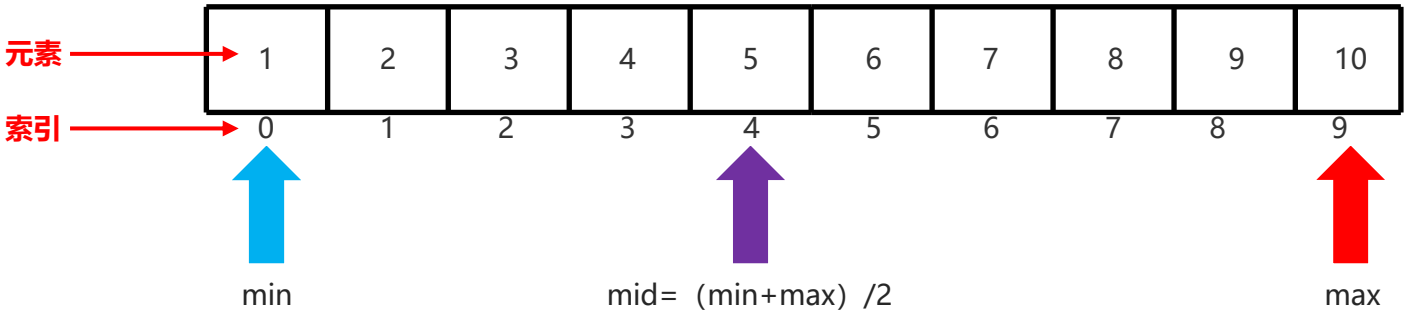
数组的高级操作

二分查找 --- 元素不存在



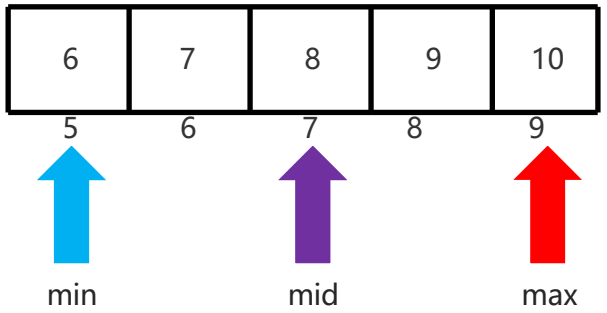
数组的高级操作

二分查找 --- 元素不存在



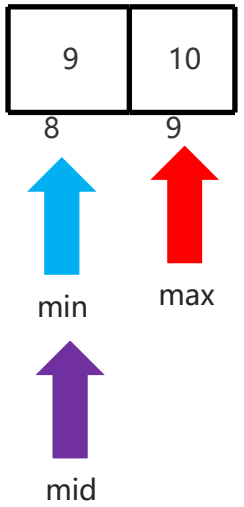
数组的高级操作

二分查找 --- 元素不存在



数组的高级操作

二分查找 --- 元素不存在

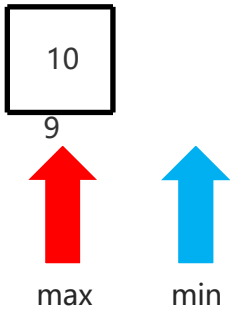


数组的高级操作

二分查找 --- 元素不存在

数组的高级操作

二分查找 --- 元素不存在



数组的高级操作

数组的二分查找步骤

1，定义两个变量，表示要查找的范围。默认min = 0， max = 最大索引
2，循环查找，但是min <= max
3，计算出mid的值
4，判断mid位置的元素是否为要查找的元素，如果是直接返回对应索引
5，如果要查找的值在mid的左半边，那么min值不变，max = mid -1.继续下次循环查找
6，如果要查找的值在mid的右半边，那么max值不变，min = mid + 1.继续下次循环查找
7，当 min > max 时，表示要查找的元素在数组中不存在，返回-1.

数组的高级操作

冒泡排序

排序：将一组数据按照固定的规则进行排列

冒泡排序：相邻的数据两两比较，小的放前面，大的放后面。

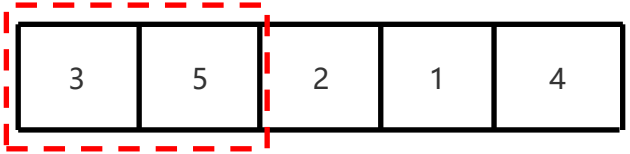
数组的高级操作

冒泡排序

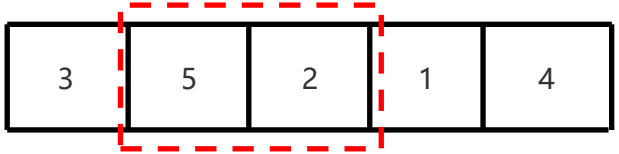
初始数据：

最终数据：

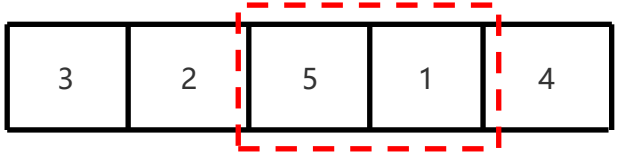
数组的高级操作



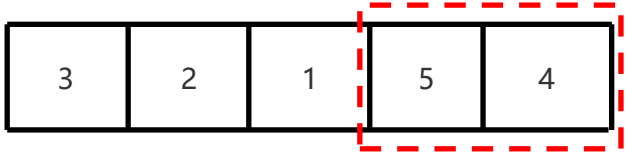
数组的高级操作



数组的高级操作



数组的高级操作



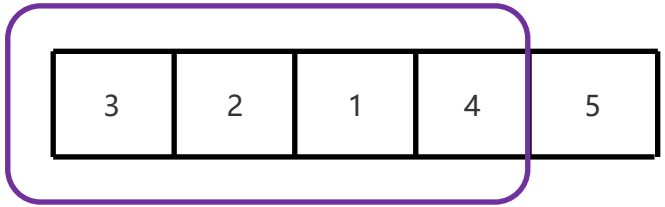
1，相邻的元素两两比较，大的放右边，小的放左边，找到最大值。

数组的高级操作

1，相邻的元素两两比较，大的放右边，小的放左边，找到最大值。

2，第一次循环结束，最大值已经找到，在数组的最右边。

数组的高级操作

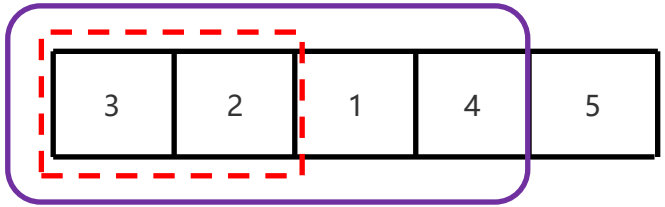


1，相邻的元素两两比较，大的放右边，小的放左边，找到最大值。

2，第一次循环结束，最大值已经找到，在数组的最右边。

3，下一次只要在剩余的元素找最大值就可以了。

数组的高级操作

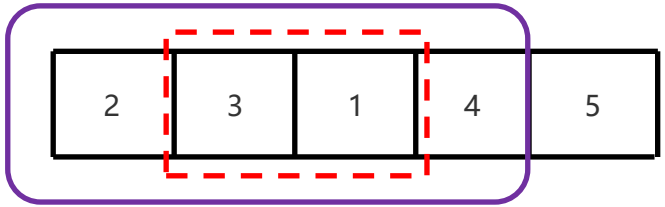


1，相邻的元素两两比较，大的放右边，小的放左边，找到最大值。

2，第一次循环结束，最大值已经找到，在数组的最右边。

3，下一次只要在剩余的元素找最大值就可以了。

数组的高级操作

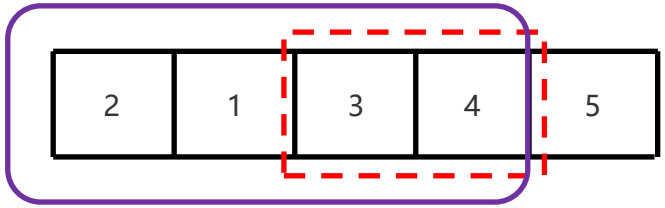


1，相邻的元素两两比较，大的放右边，小的放左边，找到最大值。

2，第一次循环结束，最大值已经找到，在数组的最右边。

3，下一次只要在剩余的元素找最大值就可以了。

数组的高级操作



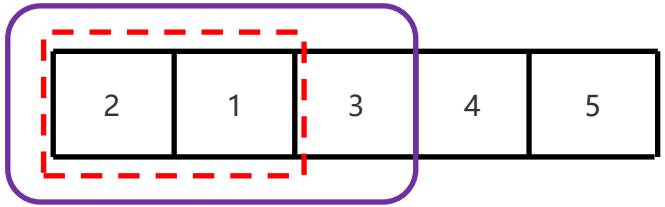
1，相邻的元素两两比较，大的放右边，小的放左边，找到最大值。

2，第一次循环结束，最大值已经找到，在数组的最右边。

3，下一次只要在剩余的元素找最大值就可以了。

4，因为已经确定了5是最大值，所以4跟5无须再进行比较了。

数组的高级操作



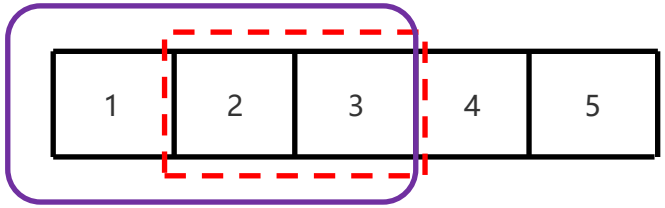
1，相邻的元素两两比较，大的放右边，小的放左边，找到最大值。

2，第一次循环结束，最大值已经找到，在数组的最右边。

3，下一次只要在剩余的元素找最大值就可以了。

4，因为已经确定了5是最大值，所以4跟5无须再进行比较了。

数组的高级操作



1，相邻的元素两两比较，大的放右边，小的放左边，找到最大值。

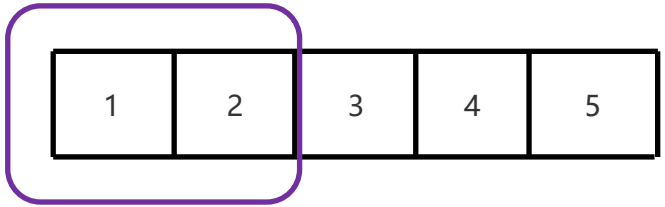
2，第一次循环结束，最大值已经找到，在数组的最右边。

3，下一次只要在剩余的元素找最大值就可以了。

4，因为已经确定了5是最大值，所以4跟5无须再进行比较了。

5，因为已经确定了5是最大值，4是次大值。所以3无须跟4和5再进行比较了。

数组的高级操作



1，相邻的元素两两比较，大的放右边，小的放左边，找到最大值。

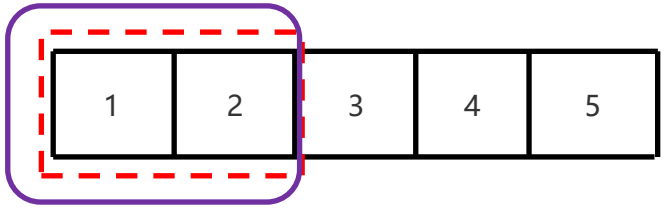
2，第一次循环结束，最大值已经找到，在数组的最右边。

3，下一次只要在剩余的元素找最大值就可以了。

4，因为已经确定了5是最大值，所以4跟5无须再进行比较了。

5，因为已经确定了5是最大值，4是次大值。所以3无须跟4和5再进行比较了。

数组的高级操作



1，相邻的元素两两比较，大的放右边，小的放左边，找到最大值。

2，第一次循环结束，最大值已经找到，在数组的最右边。

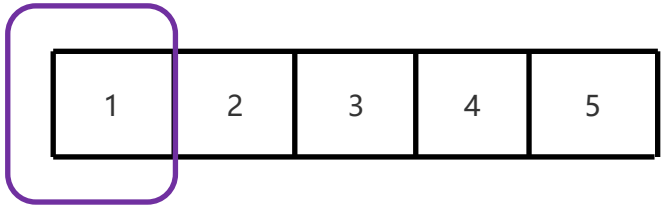
3，下一次只要在剩余的元素找最大值就可以了。

4，因为已经确定了5是最大值，所以4跟5无须再进行比较了。

5，因为已经确定了5是最大值，4是次大值。所以3无须跟4和5再进行比较了。

6，同理3，4，5的位置已经确定了，2也无须这三个值进行比较了

数组的高级操作



1，相邻的元素两两比较，大的放右边，小的放左边，找到最大值。

2，第一次循环结束，最大值已经找到，在数组的最右边。

3，下一次只要在剩余的元素找最大值就可以了。

4，因为已经确定了5是最大值，所以4跟5无须再进行比较了。

5，因为已经确定了5是最大值，4是次大值。所以3无须跟4和5再进行比较了。

6，同理3，4，5的位置已经确定了，2也无须这三个值进行比较了

7，最后只剩下一个值1了，肯定就放在最后一个格子中

数组的高级操作

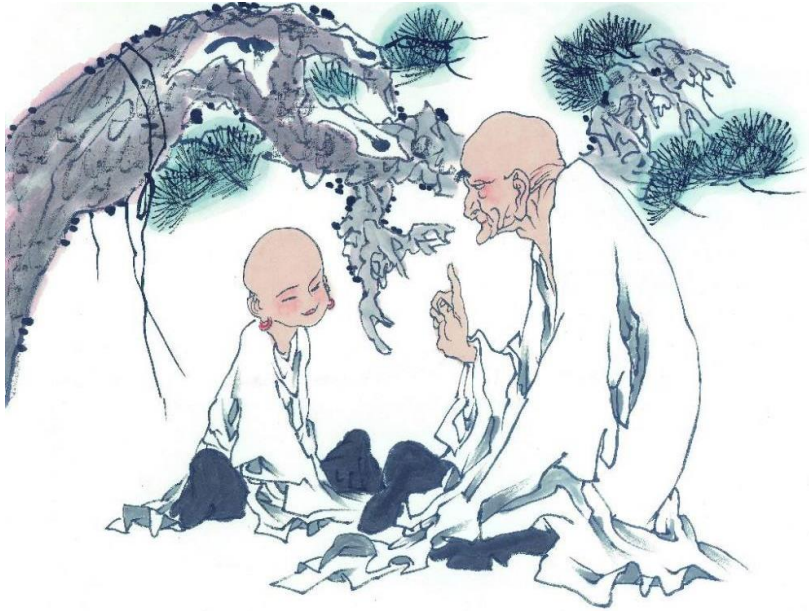
冒泡排序

- 如果有n个数据进行排序，总共需要比较n-1次
- 每一次比较完毕，下一次的比较就会少一个数据参与

数组的高级操作

递归

老和尚给小和尚讲故事，故事是：从前有座山，山里有个庙，庙里有个老和尚，老和尚给小和尚讲故事，故事是……



数组的高级操作

递归

递归概述：以编程的角度来看，递归指的是方法定义中调用方法本身的现象

做一个小案例：

求1-100之间的和

数组的高级操作

递归

递归概述：以编程的角度来看，递归指的是方法定义中调用方法本身的现象

递归解决问题的思路：

把一个复杂的问题层层转化为一个**与原问题相似的规模较小**的问题来求解

递归策略只需**少量的程序**就可描述出解题过程所需要的多次重复计算

递归解决问题要找到两个内容：

- 递归出口：否则会出现内存溢出
- 递归规则：与原问题相似的规模较小的问题

数组的高级操作

案例：递归求阶乘

需求：用递归求5的阶乘，并把结果在控制台输出

分析：

① 阶乘：一个正整数的阶乘是所有小于及等于该数的正整数的积，自然数n的阶乘写作n!

5! = 5*4*3*2*1

② 递归出口：1! = 1

③ 递归规则：n!=n*(n-1)!

5! = 5*4!

数组的高级操作

案例：递归求阶乘

需求：用递归求5的阶乘，并把结果在控制台输出

思路：

① 定义一个方法，用于递归求阶乘，参数为一个int类型的变量

② 在方法内部判断该变量的值是否是1

是：返回1

不是：返回n*(n-1)!

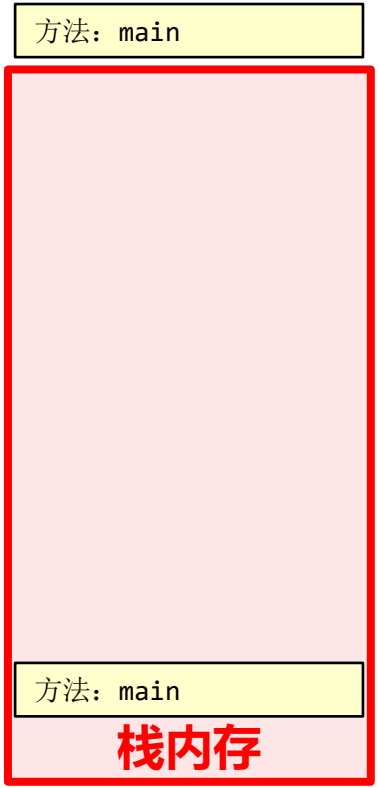
③ 调用方法

④ 输出结果

数组的高级操作

案例：递归求阶乘

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = jc(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int jc(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * jc(n - 1);
        }
    }
}
```

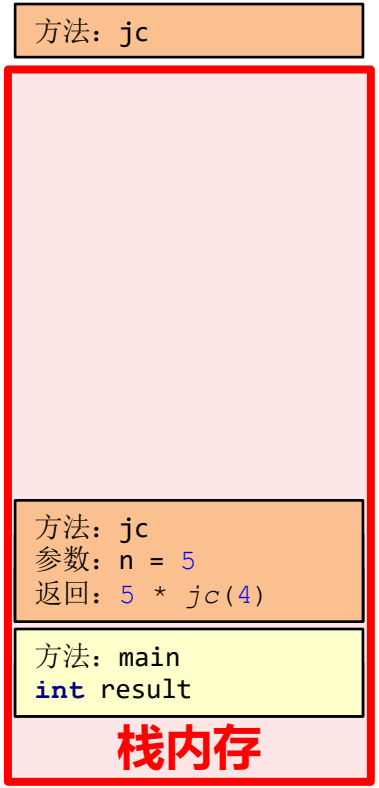


数组的高级操作

案例：递归求阶乘

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = jc(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int jc(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * jc(n - 1);
        }
    }
```

```
public static int jc(int n) {
    if (n == 1) {
        return 1;
    } else {
        return n * jc(n - 1);
    }
}
```



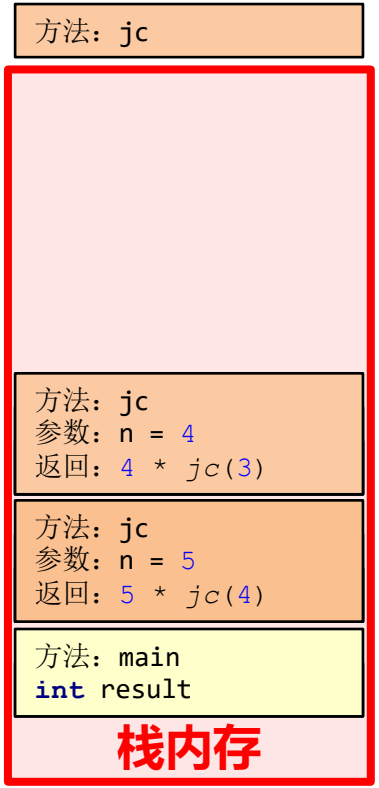
数组的高级操作

案例：递归求阶乘

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = jc(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int jc(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * jc(n - 1);
        }
```

```
public static int jc(int n) {
    if (n == 1) {
        return 1;
    } else {
        return n * jc(n - 1);
    }
}
```

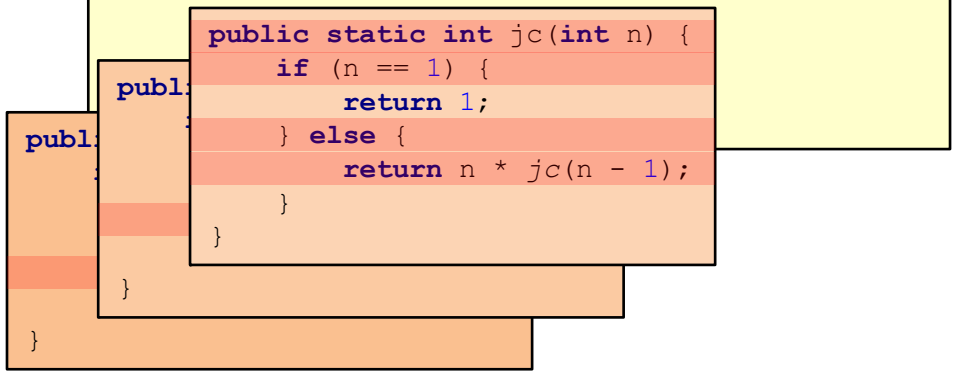
```
}
```



数组的高级操作

案例：递归求阶乘

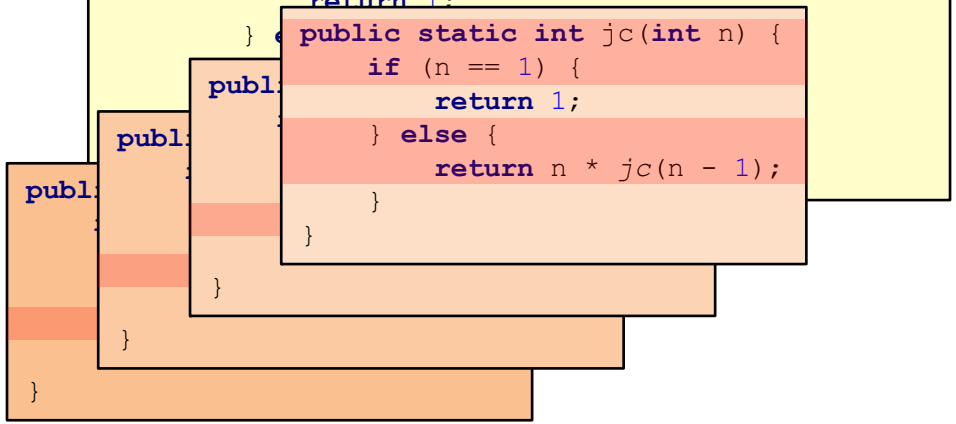
```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = jc(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int jc(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * jc(n - 1);
        }
    }
```



数组的高级操作

案例：递归求阶乘

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = jc(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int jc(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * jc(n - 1);
        }
    }
```



数组的高级操作

案例：递归求阶乘

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = jc(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int jc(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * jc(n - 1);
        }
    }
```

方法：jc
参数：n = 1
返回：1

方法：jc
参数：n = 2
返回：2 * jc(1)

方法：jc
参数：n = 3
返回：3 * jc(2)

方法：jc
参数：n = 4
返回：4 * jc(3)

方法：jc
参数：n = 5
返回：5 * jc(4)

方法：main
int result

栈内存

数组的高级操作

案例：递归求阶乘

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = jc(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int jc(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * jc(n - 1);
        }
    }
```

输出：5的阶乘是：120

数组的高级操作

案例：递归求阶乘

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = jc(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int jc(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * jc(n - 1);
        }
    }
```

方法：jc
参数：n = 1
返回：1

方法：jc
参数：n = 2
返回：2 * jc(1)

方法：jc
参数：n = 3
返回：3 * jc(2)

方法：jc
参数：n = 4
返回：4 * jc(3)

方法：jc
参数：n = 5
返回：5 * jc(4)

方法：main
int result

栈内存

数组的高级操作

快排

第一位同学
站到你应该站的位置

男孩

孩儿他妈

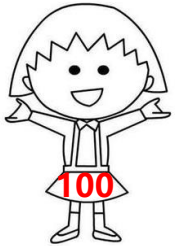
弟弟

姐姐

风间

小新

小丸子



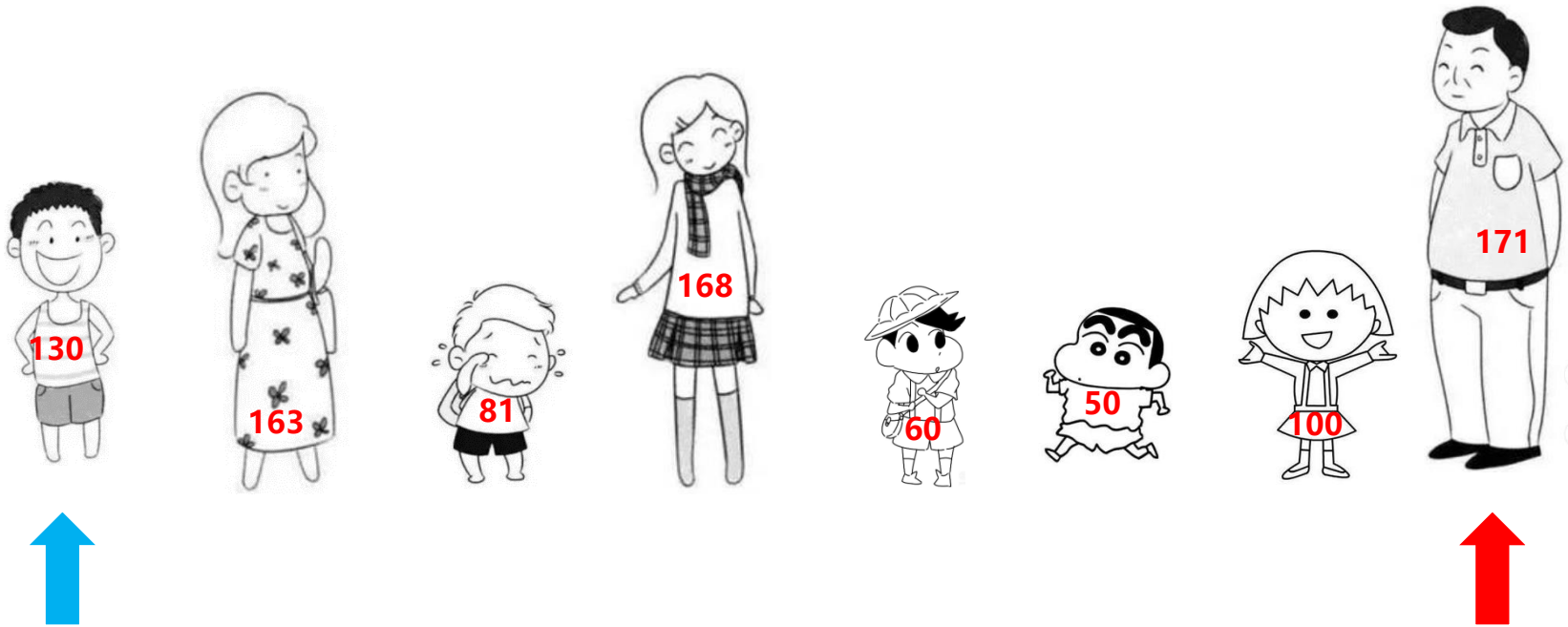
孩儿他爸

找比他高的

找比他矮的

数组的高级操作

快排



数组的高级操作

快排

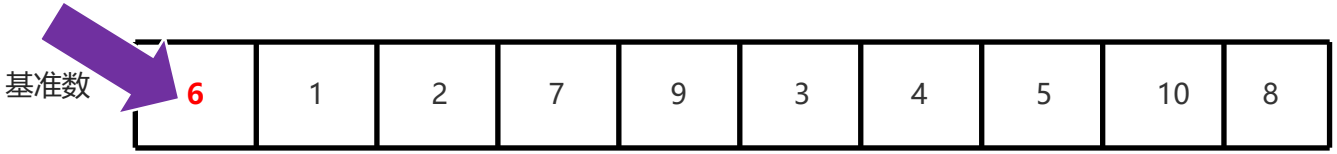
- 冒泡排序算法中，一次循环结束，就相当于确定了当前的最大值，也能确定最大值在数组中应存入的位置。
- 快速排序算法中，每一次递归时以第一个数为基准数，找到数组中所有比基准数小的。再找到所有比基准数大的。小的全部放左边，大的全部放右边，确定基准数的正确位置。

6	1	2	7	9	3	4	5	10	8

数组的高级操作

快排

- 冒泡排序算法中，一次循环结束，就相当于确定了当前的最大值，也能确定最大值在数组中应存入的位置。
- 快速排序算法中，每一次递归时以第一个数为基准数，找到数组中所有比基准数小的。再找到所有比基准数大的。小的全部放左边，大的全部放右边，确定基准数的正确位置。



数组的高级操作

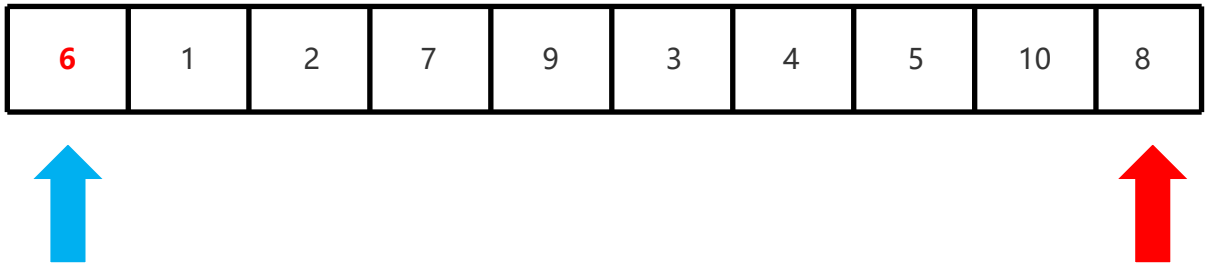
快排

基准数

基准数左边的，都比基准数小

基准数右边的，都比基准数大

相当于已经找到基准数应该在的位置



1，从右开始找比基准数小的

2，从左开始找比基准数大的

3，交换两个值的位置

4，红色继续往左找，蓝色继续往右找，直到两个箭头指向同一个索引为止

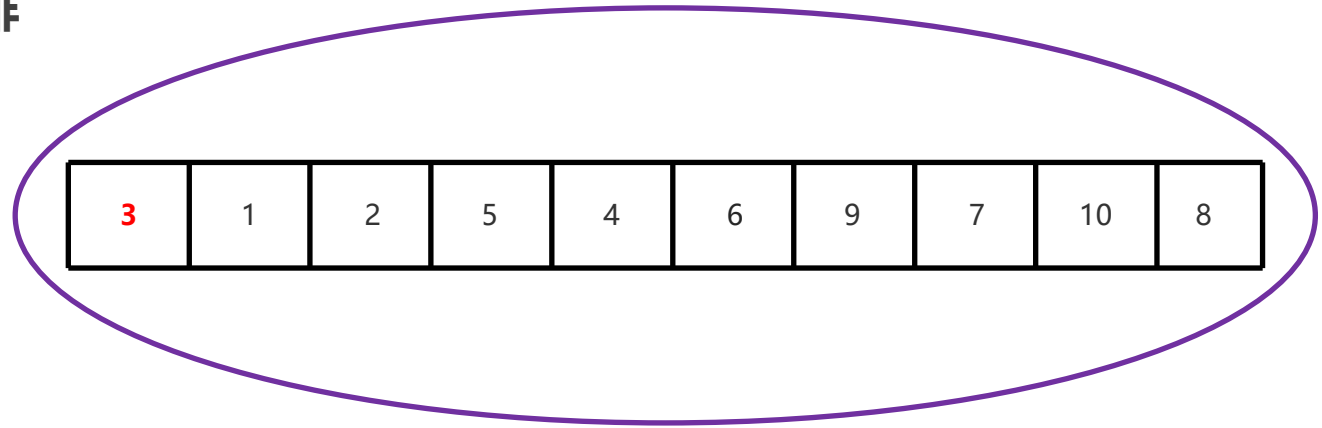
5，基准数归位

数组的高级操作

快排

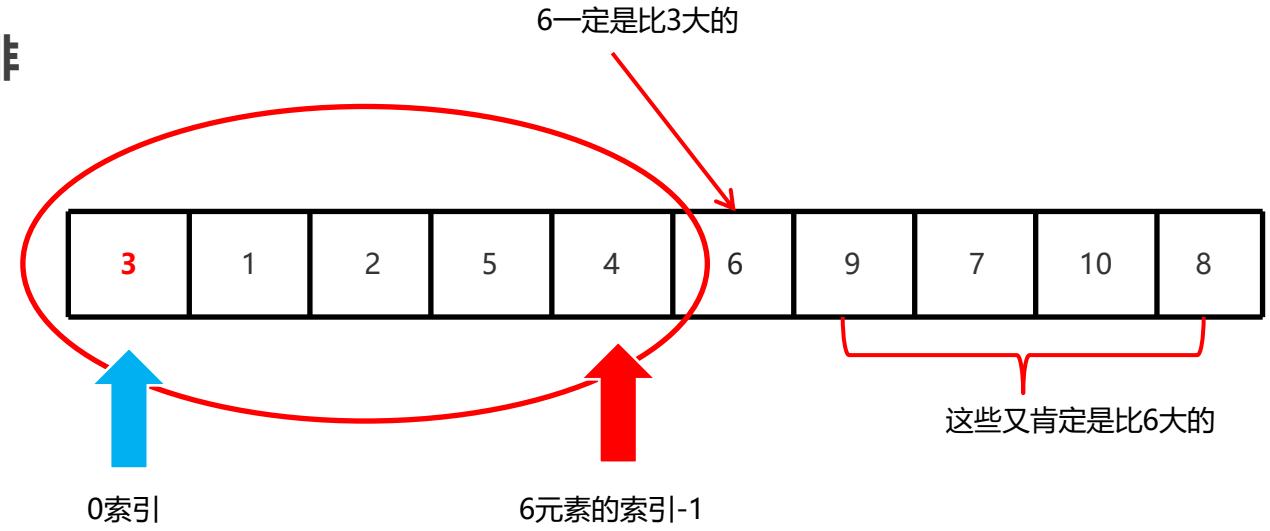
数组的高级操作

快排



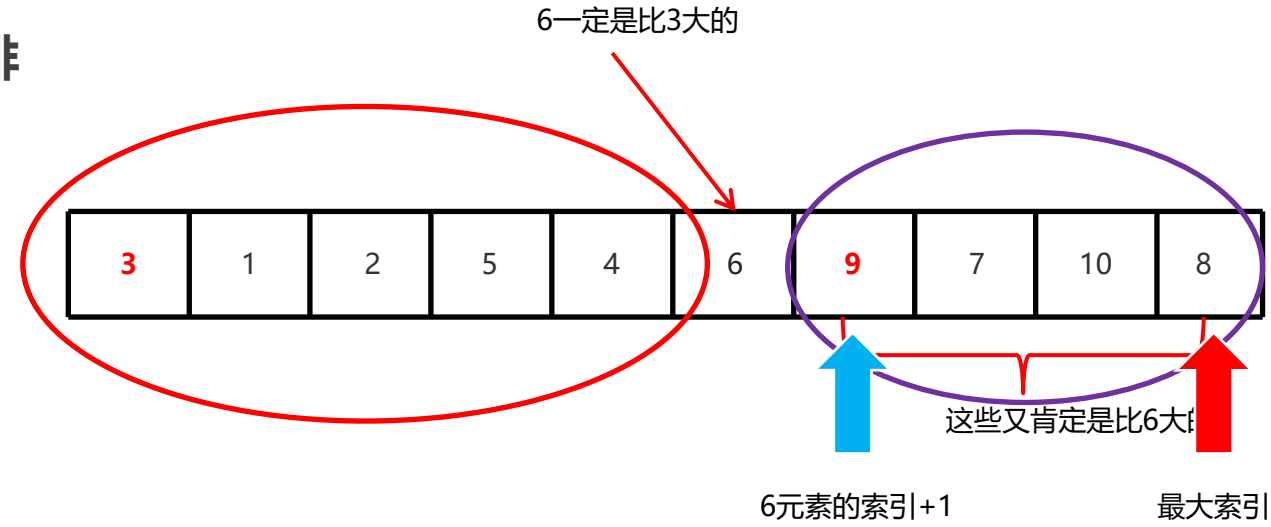
数组的高级操作

快排



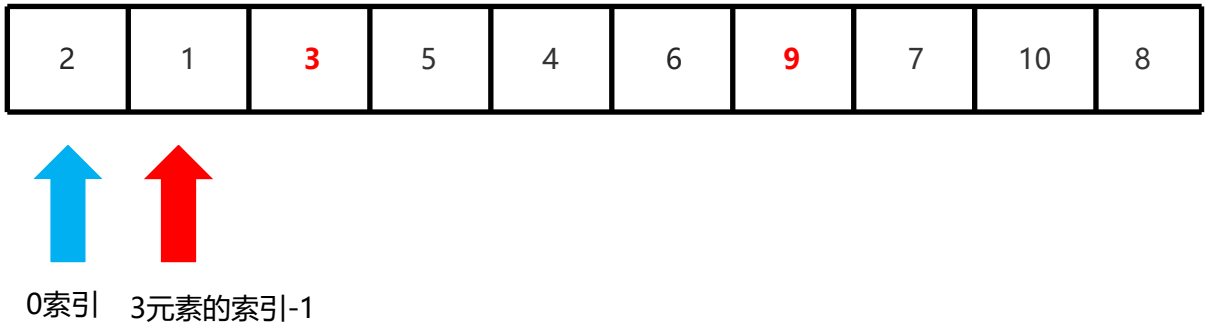
数组的高级操作

快排



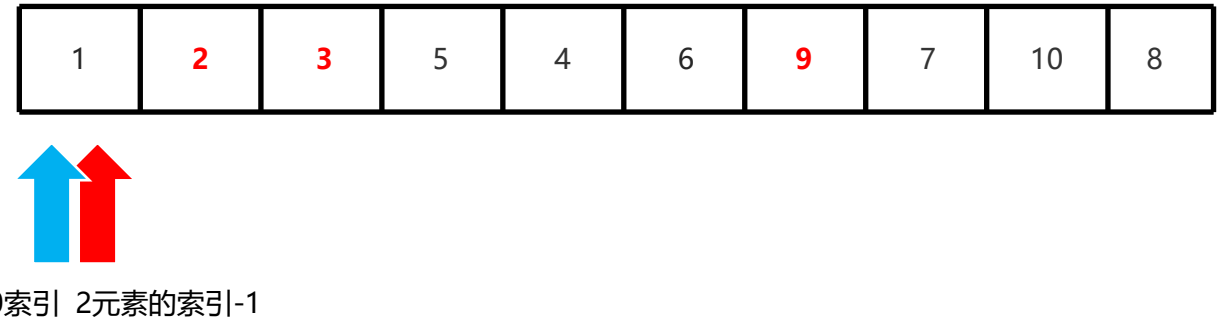
数组的高级操作

快排



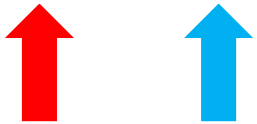
数组的高级操作

快排



数组的高级操作

快排



1元素的索引-1

0索引

此时left = 0

right = -1

left > right 停止递归

目录 Contents

Arrays 类的概述和常用方法

Arrays 类包含用于操作数组的各种方法

方法名	说明
public static String toString(int[] a)	返回指定数组的内容的字符串表示形式
public static void sort(int[] a)	按照数字顺序排列指定的数组
public static int binarySearch(int[] a, int key)	利用二分查找返回指定元素的索引

传智播客旗下高端IT教育品牌