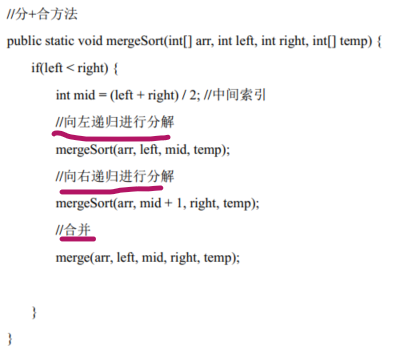
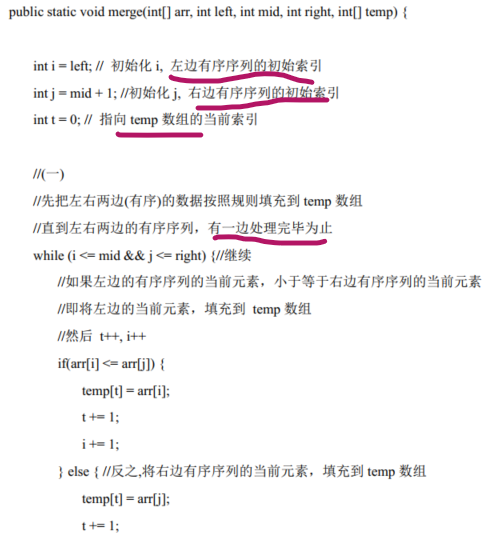
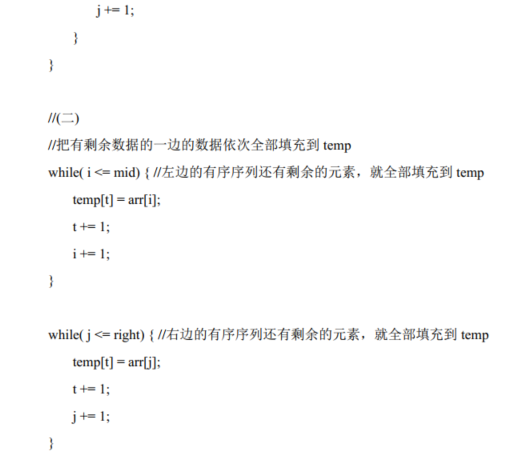
**和尚刷题总结：https://github.com/heshangcode/LeetCodeAndJianZhiOffer**

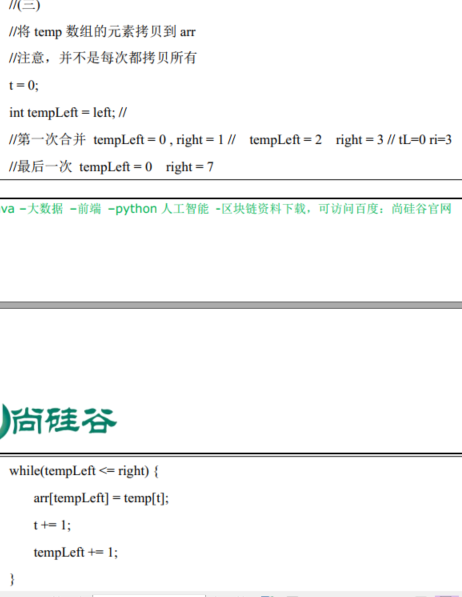
**一、排序查找**

**1.归并排序**：第4题（二分查找）









**2.堆排序：**第23题（合并多个有序链表）（大顶堆、小顶堆）

|  |
| --- |
| //创建一个堆，并设置元素的排序方式（小顶堆）          PriorityQueue<ListNode> queue = new PriorityQueue(new Comparator<ListNode>() {              public int compare(ListNode o1, ListNode o2) {                  return (o1.val - o2.val);              }          });  public Comparator<? super E> comparator()返回用来对此队列中的元素进行排序的比较器；如果此队列根据其元素的自然顺序进行排序，则返回 null。 |

堆（完全二叉树）的定义如下：具有n个元素的序列（h1,h2,...,hn),当且仅当满足（hi>=h2i,hi>=2i+1）或（hi<=h2i,hi<=2i+1）(i=1,2,...,n/2)时称之为堆。在这里只讨论满足前者条件的堆。由堆的定义可以看出，堆顶元素（即第一个元素）必为最大项（大顶堆）。完全二叉树可以很直观地表示堆的结构。堆顶为根，其它为左子树、右子树。初始时把要排序的数的序列看作是一棵顺序存储的二叉树，调整它们的存储序，使之成为一个堆，这时堆的根节点的数最大。然后将根节点与堆的最后一个节点交换。然后对前面(n-1)个数重新调整使之成为堆。依此类推，直到只有两个节点的堆，并对它们作交换，最后得到有n个节点的有序序列。从算法描述来看，堆排序需要两个过程，一是建立堆，二是堆顶与堆的最后一个元素交换位置。所以堆排序有两个函数组成。一是建堆的渗透函数，二是反复调用渗透函数实现排序的函数。

（2）采用PriorityQueue实现大小顶堆 解决topK问题

讲的挺好，参考链接：<https://blog.csdn.net/hefenglian/article/details/81807527>

①选择最大的K个数

用PriorityQueue默认是自然顺序排序，要选择最大的k个数，构造小顶堆，每次取数组中剩余数与堆顶的元素进行比较，如果新数比堆顶元素大，则删除堆顶元素，并添加这个新数到堆中。

②选择最小的K个数

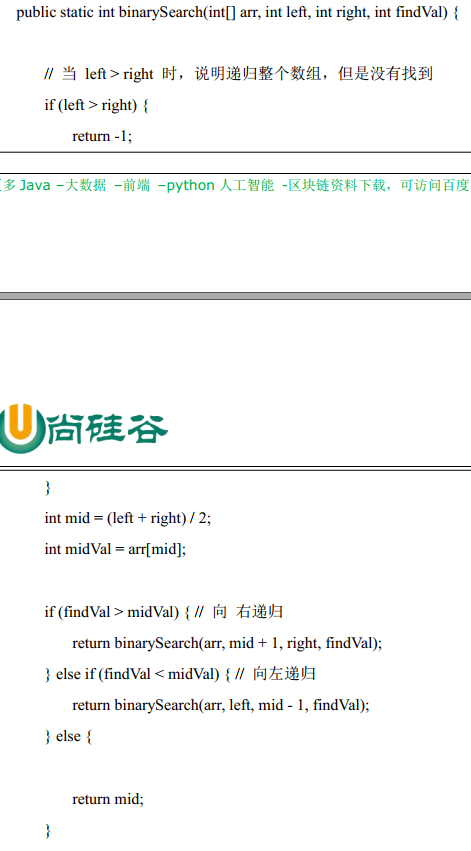
选择最小的k个数可以用冒泡排序，复杂度为O(n\*k)，有点高。

要选择最小的K个数使用大顶堆，每次取数组中剩余数与堆顶的元素进行比较，如果新数比堆顶元素小，则删除堆顶元素，并添加这个新数到堆中。

利用PriorityQueue构建大顶堆与小顶堆

|  |
| --- |
| /\*\*  \* Definition for singly-linked list.  \* public class ListNode {  \* int val;  \* ListNode next;  \* ListNode(int x) { val = x; }  \* }  \*/  class Solution {  public ListNode mergeKLists(ListNode[] lists) {  if(lists==null || lists.length==0) {  return null;  }  //创建一个堆，并设置元素的排序方式  PriorityQueue<ListNode> queue = new PriorityQueue(new Comparator<ListNode>() {  public int compare(ListNode o1, ListNode o2) {  return (o1.val - o2.val);//构建的小顶堆  return (o2.val-o1.val);//构建的大顶堆  }  });  //遍历链表数组，然后将每个链表的每个节点都放入堆中  for(int i=0;i<lists.length;i++) {  while(lists[i] != null) {  queue.add(lists[i]);    lists[i] = lists[i].next;  }  }  ListNode dummy = new ListNode(-1);  ListNode head = dummy;  //从堆中不断取出元素，并将取出的元素串联起来  while( !queue.isEmpty() ) {  dummy.next = queue.poll();  dummy = dummy.next;  }  dummy.next = null;  return head.next;    }  } |

**3.二分查找算法**（前提数组有序）（递归实现）时间复杂度（O(log(n)）



**4.DFS算法（图的深度与广度优先遍历）**

java代码使用BFS和DFS遍历二叉树

https://blog.csdn.net/qq\_37638061/article/details/89598413

**二、字符串：**

**1.找出字符串的数字字符并转成数字**（第8题）

①判断字符是否为数字Character.isDigit(chars[idx])

②将字符串转为字符数组str.toCharArray()

③取出字符串的第i个字符str.chatAt(i)

**2.最长公共前缀**（第14题）

①截取字符串中的子字符串s.substring(0,j)

public String substring(int beginIndex)

public String substring(int beginIndex, int endIndex)

参数的意思

* beginIndex -- 起始索引（包括）。
* endIndex -- 结束索引（不包括）。

第一种：返回一个新的字符串，它是此字符串的一个子字符串。该子字符串从指定索引处的字符开始，直到此字符串末尾。第二种：返回一个新字符串，它是此字符串的一个子字符串。该子字符串从指定的 beginIndex 处开始，直到索引 endIndex- 1 处的字符。因此，该子字符串的长度为 endIndex-beginIndex。

②将字符转化为字符串Character.toString(char)

**三、数组：**

**1.数组遍历，**双指针法（第15题三数之和）

Arrays此类包含用来操作数组（比如排序和搜索）的各种方法

①static <T> List<T> asList(T... a) 返回一个受指定数组支持的固定大小的列表。

②static char[] copyOf(char[] original, int newLength) 复制指定的数组，截取或用 null 字符填充（如有必要），以使副本具有指定的长度。

③static char[] copyOfRange(char[] original, int from, int to) 将指定数组的指定范围复制到一个新数组。

④static boolean equals(char[] a, char[] a2) 如果两个指定的 char 型数组彼此相等，则返回 true。

⑤static void sort(char[] a) 对指定的 char 型数组按数字升序进行排序。

⑥static String toString(char[] a) 返回指定数组内容的字符串表示形式

**2.删除排序数组的重复项**（第26题）双指针法

**3.数组中第k大元素（第215题）**

改进排序算法即可，排到第k个元素就找到了

①a=Integer.MIN\_VALUE最小值表示

②b=Integer.MAX\_VALUE最大值表示

**4.全排列（第46题）**

用到的是回溯算法（路径+选择表）

**四、链表**

**1.合并两个有序链表（第21题）**

**2.合并多个有序链表（第23题）**

**五、二叉树**

**1.** **二叉树的最近公共祖先（第236题）DFS、剪枝**

**2.二叉树的前序中序后序遍历**

**六、回溯算法**

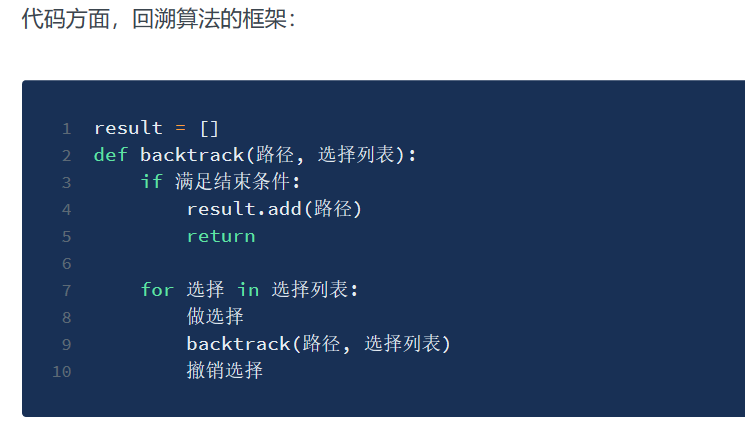
废话不多说，直接上回溯算法框架。解决一个回溯问题，实际上就是一个决策树的遍历过程。你只需要思考 3 个问题：

1、路径：也就是已经做出的选择。

2、选择列表：也就是你当前可以做的选择。

3、结束条件：也就是到达决策树底层，无法再做选择的条件。

如果你不理解这三个词语的解释，没关系，我们后面会用「全排列」和「N 皇后问题」这两个经典的回溯算法问题来帮你理解这些词语是什么意思，现在你先留着印象。

****

其核心就是 for 循环里面的递归，在递归调用之前「做选择」，在递归调用之后「撤销选择」，特别简单。

**七、动态规划（用空间换时间）**

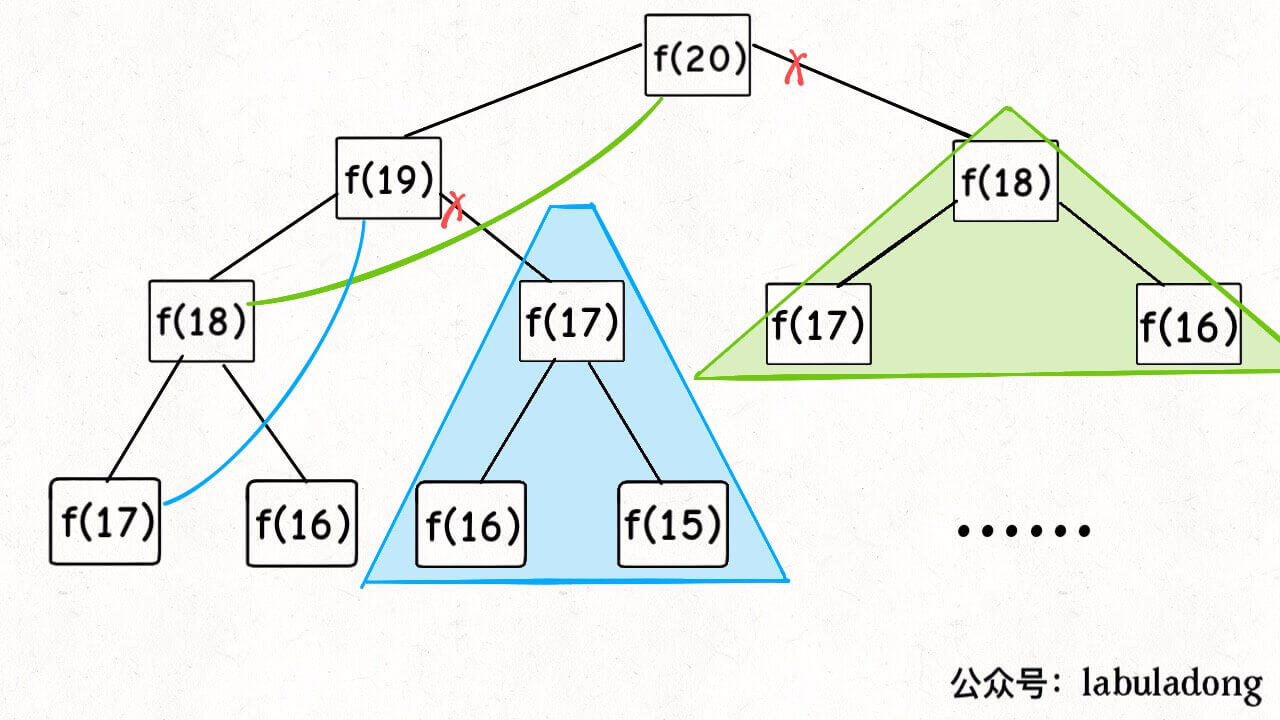
1.动态规划的通用技巧：数学归纳思想。

2.动态规划问题的一般形式就是求最值。动态规划其实是运筹学的一种最优化方法，只不过在计算机问题上应用比较多，比如说让你求最长递增子序列呀，最小编辑距离呀等等。

3. 第一个斐波那契数列的问题，解释了如何通过「备忘录」或者「dp table」的方法来优化递归树，并且明确了这两种方法本质上是一样的，只是自顶向下和自底向上的不同而已。

第二个凑零钱的问题，展示了如何流程化确定「状态转移方程」，只要通过状态转移方程写出暴力递归解，剩下的也就是优化递归树，消除重叠子问题而已。（备忘录也可以理解为子问题，动态规划的最后结果就是由这一个个子问题叠加得出来的，我们要保证我们规定的子问题相互独立，无重叠）

4.最本质方法就是穷举，然后对穷举进行优化，找出子问题，叠加得出结果



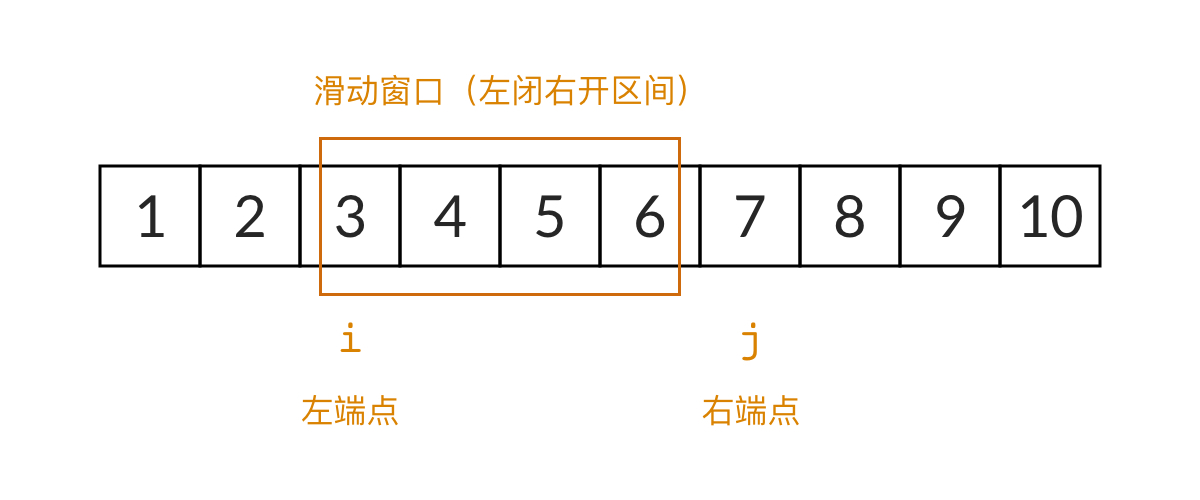
斐波那契数列

**八、滑动窗口**

什么是滑动窗口

**滑动窗口可以看成数组中框起来的一个部分**。在一些数组类题目中，我们可以用滑动窗口来观察可能的候选结果。当滑动窗口从数组的左边滑到了右边，我们就可以从所有的候选结果中找到最优的结果。

对于这道题来说，数组就是正整数序列 [1,2,3,…,n]。我们设滑动窗口的左边界为 i，右边界为 j，则滑动窗口框起来的是一个左闭右开区间 [i,j)。注意，为了编程的方便，滑动窗口一般表示成一个左闭右开区间。在一开始，i=1,j=1，滑动窗口位于序列的最左侧，窗口大小为零。



滑动窗口的重要性质是：窗口的左边界和右边界永远只能向右移动，而不能向左移动。这是为了保证滑动窗口的时间复杂度是 O(n)。如果左右边界向左移动的话，这叫做“回溯”，算法的时间复杂度就可能不止 O(n)。

在这道题中，我们关注的是滑动窗口中所有数的和。当滑动窗口的右边界向右移动时，也就是 j = j + 1，窗口中多了一个数字 j，窗口的和也就要加上 j。当滑动窗口的左边界向右移动时，也就是 i = i + 1，窗口中少了一个数字 i，窗口的和也就要减去 i。滑动窗口只有 右边界向右移动（扩大窗口） 和 左边界向右移动（缩小窗口） 两个操作，所以实际上非常简单。

**作者：nettee**

**链接：https://leetcode-cn.com/problems/he-wei-sde-lian-xu-zheng-shu-xu-lie-lcof/solution/shi-yao-shi-hua-dong-chuang-kou-yi-ji-ru-he-yong-h/**

**来源：力扣（LeetCode）**

**著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。**

**最后、常用的函数、知识点总结**

**1. 通常情况下**，我们通常直接使用此方法对数组等进行排序，其实还有一种自定义比较器制定比较规则的方法：

sort(T[] a, Comparator<? super T> c)

继承此方法的时候，要自定义比较器，conpareTo方法返回值为1(升序),0，-1(降序)。

Arrays.sort(str,new Comparator<String>(){

@Override

public int compare(String s1, String s2) {

String c1 = s1 + s2;

String c2 = s2 + s1;

return c1.compareTo(c2);

}

});

//此句可进行优化,lamada表达式

Arrays.sort(str,(s1,s2)->(s1+s2).compareTo(s2+s1));

**2.类Arrays此类包含用来操作数组（比如排序和搜索）的各种方法。**

|  |
| --- |
| **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。** |

|  |
| --- |
| **copyOf(int[] original, int newLength)**  **复制指定的数组，截取或用 0 填充（如有必要），以使副本具有指定的长度。** |

|  |
| --- |
| **copyOfRange(char[] original, int from, int to)**  **将指定数组的指定范围复制到一个新数组。** |

|  |
| --- |
| **equals(int[] a, int[] a2)**  **如果两个指定的 int 型数组彼此相等，则返回 true。** |

|  |
| --- |
| **sort(int[] a)**  **对指定的 int 型数组按数字升序进行排序。** |

|  |
| --- |
| **toString(char[] a)**  **返回指定数组内容的字符串表示形式。** |

|  |
| --- |
| char[] c=s.toCharArray();  **字符串转成字符数组** |

**3.变量的作用域**

一个变量的作用域，在定义它的时刻起，到离它最近的{}里面，出了这个{}作用域就结束了

**4.定义可变长的二维数组**

利用类ArrayList中的方法toArray()来实现

1.List<int []> list=new ArrayList<>();

2.list.toArray(new int[list.size()][]);

**5.字符串相关**

⑴“trim()的作用是去掉字符串两端的多余的空格,注意,是两端的空格,且无论两端的空格有多少个都会去掉,当然 中间的那些空格不会被去掉,如: String s = " a s f g "; String s1 = s.trim(); 那么s1就是"a s f g",可见,这和上面所说的是..

|  |
| --- |
| StringBuffer sb=new StringBuffer();          while(!stack.empty()){              sb.append(stack.pop());              sb.append(" ");          }            return sb.toString().trim();  **toString()**  返回表示该序列中的数据的字符串。分配并初始化一个新的字符串对象，以包含该对象当前表示的字符序列。然后返回该字符串。此序列的后续更改不影响字符串的内容 |

|  |
| --- |
| **StringBuffer类**  线程安全的可变字符序列。一个类似于 [String](mk:@MSITStore:C:\Users\fzhen\Desktop\资料库\javaapi\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/String.html) 的字符串缓冲区，但不能修改。虽然在任意时间点上它都包含某种特定的字符序列，但通过某些方法调用可以改变该序列的长度和内容。 |

|  |
| --- |
| res.append(s.substring(i + 1, j + 1) + " "); // 添加单词  substring(i,j)截取字符串中的指定位置子字符串从i到j-1 |

**6.String类学习**

字符串串联是通过 StringBuilder（或 StringBuffer）类及其 append 方法实现的。字符串转换是通过 toString 方法实现的，

|  |
| --- |
| **1.字符数组直接转换成字符串**  char[] a1= {'a','w','q','r'};  String e=new String(a1);  **2.返回字符串中的指定字符**  char charAt(int index)  **3.返回指定数组中表示该字符序列的 String**  String copyValueOf(char[] data)  **4.根据给定正则表达式的匹配拆分此字符串**  String split(String regex)  **5.返回一个新字符串，它是此字符串的一个子字符串**  String substring(int beginIndex, int endIndex)  **6.将此字符串转换为一个新的字符数组。**  Char[] toCharArray()  **7. 返回字符串的副本，忽略前导空白和尾部空白。(将字符串首尾空格去掉)**  String trim()  **8.返回各种字符数组的字符串表示形式**  String valueOf(char[] data) |

**7.StringBuffer类**

线程安全的可变字符序列。一个类似于 String 的字符串缓冲区

|  |
| --- |
| **1.合并字符串，将字符串加到序列中**  append(String str)  **2.将字符序列以字符串的形式返回**  String toString()  **3.插入方法StringBuffer insert(int offset, char c)**  例如，如果 z 引用一个当前内容是“start”的字符串缓冲区对象，则此方法调用 z.append("le") 会使字符串缓冲区包含“startle”，而 z.insert(4, "le") 将更改字符串缓冲区，使之包含“starlet”。  4.字符序列反转**[reverse](mk:@MSITStore:C:\\Users\\fzhen\\Desktop\\资料库\\javaapi\\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/StringBuffer.html" \l "reverse())**()  5.将StringBuffer重新归零**[setLength](mk:@MSITStore:C:\\Users\\fzhen\\Desktop\\资料库\\javaapi\\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/StringBuffer.html" \l "setLength(int))**(0)            Sets the length of the character sequence设置字符序列的长度 |

**8.Arrays类**

此类包含用来操作数组（比如排序和搜索）的各种方法。此类还包含一个允许将数组作为列表来查看的静态工厂。

|  |
| --- |
| **1. 使用二分搜索法来搜索指定的 int 型数组，以获得指定的值。**  static int binarySearch(int[] a, int key)  **2. 对指定的 int 型数组按数字升序进行排序。**  static void sort(int[] a)  **3. 返回指定数组内容的字符串表示形式**  static String toString(int[] a) |

**9. java关于ArrayList中toArray方法的使用**

**toArray()方法中会把形成数组的返回到（）中创建的数组里**

先来看下面这段程序

　　Collection collect= new ArrayList();

　　　　collect.add("小黑");

　　　　collect.add("小白");

　　　　 collect.add("小红");

在上面程序中我创建了一个集合放置了3个String类型的值

接下来我们看官方对toArray(Object array[])这个方法的描述

1.如果array的大小与匹配的元素的个数相等,它们被返回到array

程序解释:

　　String[] arrs = new String[3];

　String[] copys = (String[])collect.toArray(arrs);

值:arrs:[小黑, 小白, 小红]

　　　 copys:[小黑, 小白, 小红]

2.如果array的大小比匹配元素的个数小,将分配并返回一个所需大小的新数组

程序解释:

　　String[] arrs = new String[2];

　String[] copys = (String[])collect.toArray(arrs);

值:arrs:[[null, null]]

　　　 copys:[小黑, 小白, 小红]

3.如果array的大小比匹配元素的个数大,在数组中,在类集元素之后的单元被置为null

程序解释:

　　String[] arrs = new String[5];

　String[] copys = (String[])collect.toArray(arrs);

值:arrs:[小黑, 小白, 小红, null, null]

　　　 copys:[小黑, 小白, 小红, null, null]

4.如果任一类集元素的类型都不是array的子类型,则引发一个ArrayStoreException异常

程序解释:

　　 collect.add(1);

　　String[] arrs = new String[5];

　　String[] copys = (String[])collect.toArray(arrs);

抛出异常:java.lang.ArrayStoreException

|  |
| --- |
| **1.返回此列表中首次出现的指定元素的索引，或如果此列表不包含元素，则返回 -1。**  Int indexOf(Object o) |

|  |
| --- |
| List<Integer> list1 = new ArrayList<Integer>();  list1.add(1);  list1.add(6);  list1.add(4);  list1.add(2);  Integer[] t = new Integer[4];  list1.toArray(t);  System.out.println(Arrays.toString(t));  注意toArray()括号里面的类型一定要和返回的类型完全一致要么都是Integer,要么都是int |

**10.HashMap在统计数组里元素时，通过键值对可以用来进行标记，它还可以实现遍历**

|  |
| --- |
| **Map<Integer, Integer> counts = countNums(nums);**  **Map.Entry<Integer, Integer> majorityEntry = null;**  **for (Map.Entry<Integer, Integer> entry : counts.entrySet()) {//返回映射关系中的Set视图，便于遍历**  **if (majorityEntry == null || entry.getValue() > majorityEntry.getValue()) {**  **majorityEntry = entry;**  **}**  **}** |

**11.用位运算判断一个数是奇数还是偶数**

|  |
| --- |
| (nums[i] & 1) == 1 奇数与1相与为1，注意判断时优先级问题一定要将nums[i]&1用括号括起来  (nums[j] & 1) == 0 偶数与1相与为0 |

**12.Character类重要方法**

|  |
| --- |
| [**toLowerCase**](mk:@MSITStore:C:\Users\fzhen\Desktop\资料库\javaapi\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/Character.html#toLowerCase(char))(char ch)            使用取自 UnicodeData 文件的大小写映射信息将字符参数转换为小写。  [**toUpperCase**](mk:@MSITStore:C:\Users\fzhen\Desktop\资料库\javaapi\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/Character.html#toUpperCase(char))(char ch)            使用取自 UnicodeData 文件的大小写映射信息将字符参数转换为大写。  [**toString**](mk:@MSITStore:C:\Users\fzhen\Desktop\资料库\javaapi\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/Character.html#toString())()            返回表示此 Character 值的 String 对象。  [**isLetterOrDigit**](mk:@MSITStore:C:\Users\fzhen\Desktop\资料库\javaapi\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/Character.html#isLetterOrDigit(char))(char ch)            确定指定字符是否为字母或数字。  [**isLowerCase**](mk:@MSITStore:C:\Users\fzhen\Desktop\资料库\javaapi\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/Character.html#isLowerCase(char))(char ch)            确定指定字符是否为小写字母。  [**isUpperCase**](mk:@MSITStore:C:\Users\fzhen\Desktop\资料库\javaapi\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/Character.html#isUpperCase(char))(char ch)            确定指定字符是否为大写字母。  [**isLetter**](mk:@MSITStore:C:\Users\fzhen\Desktop\资料库\javaapi\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/Character.html#isLetter(char))(char ch)            确定指定字符是否为字母。  [**isDigit**](mk:@MSITStore:C:\Users\fzhen\Desktop\资料库\javaapi\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/Character.html#isDigit(char))(char ch)            确定指定字符是否为数字。 |

**大写字母+32=对应的小写字母**

**13.** **String[] ch=s.trim().split(" +");//split(" +")的意思是按照多个空格来拆分数组；**

**14.** 字符序列反转StringBuffer.[**reverse**](mk:@MSITStore:C:\Users\fzhen\Desktop\资料库\javaapi\%5bJava参考文档%5d.JDK_API_1_6_zh_CN.CHM::/java/lang/StringBuffer.html#reverse())()

**15.** **new Integer.valueof()返回的是Integer的对象。**

**Integer.parseInt() 返回的是一个int的值。**