Cover

常用 Function

in list ⏳ Now & 📌 Plan & 😄 API

Description Edit

Integer

- 转数字：***Interger.***parseInt(string str)/ ***Double.***parseDouble(string str)
- 转字符串：***Interger/Double.***toString(int n)
  Arrays.toString(array)
  其他 Class(Integer /Collections /StringBuilder)直接在名称后面.toString();
- 转二进制：***Interger.***toBinaryString(int n)
- 转 long 型：1）**直接 *1L** 2）(long)n

Character

- 转数字： **char - '0'**

Character.isDigit(char c) → return true(是数字)

Character.isLetter() → return true(是字母)

Character.isLetterOrDigit() → return true(是字母或数字)

Character.toLowerCase()

Character.toUpperCase()

String

String.join("", list<String>) -> return 把list里的string连在一起

删除开头的 0 或者制定位置的字符：deleteCharAt(0/index)

charAt(int index) → return char

toCharArray() → return char[]

length() 记得加()

equals() 这个逻辑才是正确的

substring(index start) / subString(index start, index end) 时间复杂度 O(n)

startsWith(string str) → return true(如果字符串是以 str 开头的) 时间复杂度 O(m)

indexOf(string str) 找出现 str 的第一个开头位置index 时间复杂度 O(n*m)

lastIndexOf() 找出现 str 最后一个的开头位置 index

trim() 把字符串的空格 trim 掉

toLowerCase()

toUpperCase()

split("正则表达式")

```
+: 匹配一次或者多次
?: 匹配 0 次或者 1 次 do(es)?
\ 反斜杠+字母表示特殊字符

" " 代表一个空格 "\\s+" 代表一个或者多个空格
"/+"代表一个或者多个/
"\\.+"代表一个或者多个.
","代表一个,
```

StringBuilder

增：append(string str) / insert(offset, ..)

append((char) (count[i] + 'a')) // 把 countingsort 转回 char 加入字符串

删：delete(int start, int end) / deleteCharAt(index)

大小：length()

判空： stringBuilder.length() >0 ?

Array

Bucket Sort 里的实例化： List<Integer>[] bucket = new List[nums.length];

复制：arr.clone() → return newarray

int[] newarray = Arrays.copyOfRange(oldarray, from, to) 左闭右开

从一个数组到另一个数组：***System.***arraycopy(copysourse, scrPos, nums, copyPos, copylength)

填充：Arrays.fill(res, 0/1) !只能填一维，不过用 for效率是一样的

打印：***Arrays.***toString(array)

返回空 array : return new int[]{}; / return new int[0]; 不是[]

实例化二维数组： int[][] matrix = {{1,0,1,0,0},{1,0,1,1,1},{1,1,1,1,1},{1,0,0,1,0}}; // 实例化一位数组也是直接写 {1，2，4}就可以了

实例化 List：List<Integer> list = ***Arrays.***asList(1,3,4);

array转 list： Arrays.asList(res) -> return List<Integer> 注意 res 不能是`int[]` 而是 `Integer[]`

list 转 array：res.toArray() → return int[] 、res.toArray(new int[][]{}) → return int[][]

ArrayList

增：add(Integer integer)/ add(int index, Integer integer)

addAll(List another) 把其他的**list** 全部加入 不能是 int[]

删：remove (int index / Object o) —> return oldvalue/boolean

改：set(int index, Integer integer) —> return oldvalue

查：get (int index) （只 index） —> return value

包含：contains(object o)

等概率抽取：list.get(random.nextInt(list.size()))

排序：Collections.sort(List<T>) （ Arrays.sort(int[]) 一样）都是升序，加上（a,b) →(b-a) 变降序

res.toArray() → return int[]

res.toArray(new int[][]{}) → return int[][]

Stack

增：push(Object value)

删：pop() → return Object value /remove(o/index) 但没用

查：peek() → return Object value

判空：isEmpty()

转列表: queue.toArray(); （只能用于打印 因为是 Object 类型）
转 ArrayList : List<Integer> list = new ArrayList<>(stack);

包含: contains(Object o)

Queue/Deque：LinkList

```
(Deque)Queue<Integer> queue = new LinkedList<>(); queue有的 deque 也有   // D

//增
queue.offer(4)//其实queue 一般是用offer增 因为增加不了只会返回false 不会报错
queue.add(0,5);//按位置增加
queue.addAll(Collections);
deque.addFirst(6);//增加到前面

//删
queue.poll();//从前面删除
deque.pollLast(); //从后面删除
queue.remove(Object); /()
deque.removeLast();

//查

queue.peek(); //查看前面的元素
deque.peekLast();//查看后面的元素
deque.getLast();

//转列表
queue.toArray();
//转 ArrayList
List<Integer> list = new ArrayList<>(queue);
//判空
queue.isEmpty(); //Arraylist 也可以用 T[]不行
```

```
//包含
queue.contains(Object o)
```

HashMap

增：put(Object key, Object value） → return null/ oldValue

加入但优先保留之前的: putIfAbsent(key, value) // 没有时候才增加 否则保留之前的 -> return oldvalue/null

加入但优先保留之前的且返回最新：computeIfAbsent(key, lambda) -> return new // 可以在后面直接加.add 啥的

删：remove(Object key) → return Object value

改：**put**(Object key, Object value)

查：get(Object key) → return Object value

key数组：.keySet()

value数组：.values()

是否存在key: containsKey(Object key)

是否存在value: containsValue(Object value)

默认值：put(key, [mapname].getOrDefault(key, [defaultValue]) + 1)

HashSet

增：add(Object o)

删：remove(Object o) -> **return boolean** 是否有且删了

是否存在: contains(Object o) 时间复杂度是 O1

是否为空：isEmpty()

转 ArrayList : List<String> list = new ArrayList<>(hashset);
转 Array: toArray();

TreeMap (按照 key 排序)

增：put(Object key, Object value） → return null/ oldValue

putIfAbsent(key, value) // 没有时候才增加 否则保留之前的 -> return old/null

删：remove(Object key) → return Object value

treeMap.pollFirstEntry(); -> return key 最小的的Map.Entry<>
treeMap.pollLastEntry(); -> return key 最大的的Map.Entry<>
TreeMap 从大到小排序不用重写（Collections.reverseOrder()）

改：**put**(Object key, Object value)

查：get(Object key) → return Object value

查最小/大Key ： firstKey() / lastKey()

key数组：.keySet() 按 key 的顺序

value数组：.values() 按 key 的顺序 (return new ArrayList<>(map.values()))

子 map：subMap(int from, [boolean inclusive], int to, [boolean inclusive]

ceiling：ceilingKey(Object key) & ceilingEntry(Object key)

floor：floorKey(Object key) & floorEntry(Object key)

higher：higherKey(Object o) & higherEntry(Object o)

lower：lowerKey(Object o) & lowerEntry(Object o)

是否存在key: containsKey(Object key)

是否存在value: containsValue(Object value)

默认值：put(key, [mapname].getOrDefault(key, [defaultValue]) + 1) **defaultvalue 一般写 0** 因为后面会加一

Map : TreeMap & HashMap 's Interface

entrySet() -> return Set<Map.Entry<o1, o2>>

Map.Entry entry : map.entrySet()

实例化: new AbstractMap.SimpleEntry<>(key, value);

getKey() -> return **Object** key

getValue() -> return **Object** Value
上面这两个 Object 是真的 Object 如果放到 int 数组还需要**加(int)转换**, 所以下面的 map 遍历推荐第1种

如何遍历Map

```
        Map<Integer, Integer> map = new TreeMap<>();
        map.put(2,3);
        map.put(4,242);
       // 1
        for (int key: map.keySet()){
            System.out.println(key);
            System.out.println(map.get(key));
        }
       // 2
        for(Map.Entry entry : map.entrySet()){
            System.out.println(entry.getKey());
            System.out.println(entry.getValue());
        }
```

TreeSet

增：add(Object o)

删：pollLast() / pollFirst() / remove(Object o) -> return boolean

查：Last()/ First()

子set：subSet(int from, [boolean inclusive], int to, [boolean inclusive])

ceiling：ceiling(Object o) 大于等于 o 的最小的数

floor：floor(Object o) 小于等于 o 的最大的数

higher：higher(Object o) 大于 o 的最小的数

lower：lower(Object o) 小于 o 的最大的数

是否存在: contains(Object o) 时间复杂度是 O1

是否为空：isEmpty()

PriorityQueue

增：offer(Object o)

addAll(map.entrySet())

删：poll() / remove() 从小的删 -> return value/ remove(Object o) ->return boolean

查：peek()

包含：contains(Object o)->return boolean

是否为空：isEmpty()

其他

remove(index)/(key)/() -> return value
remove(Object o) -> return true/false

向上取整 (int)Math.ceil(double) 向下取整 Math.floor()

Custom Fields

LAST VIEW　TIMES　OTHER SORT

+ Add date...

Attachments

ceiling ↗
Added Apr 27 at 4:48 AM - Comment - Delete - Edit

Remove cover

Add an attachment

Activity

Show details