常用数据结构+算法在Java中的实现方式

# 数据结构

## （1） 表

### 类1：顺序表

ArrayList

### 类2：链表

#### 《1》 单链表

LinkedList

#### 《2》 双链表

LinkedList

## （2） 栈

<https://juejin.cn/post/6844903838114136071>

### 【1】 法1：Stack

Java8文本

描述已自动生成

#### 《1》 实例化+初始化

Stack<Type> st = new Stack<>();

入栈，出栈，获取栈顶元素 均同步

#### 《2》 相关操作

* 入栈：

文本

描述已自动生成

文本

描述已自动生成

* 出栈：

文本

描述已自动生成

* 判空：

文本

描述已自动生成

* 获取栈顶元素：

文本

描述已自动生成

### 【2】 法2：Deque

Java8

#### 《1》 方式1：ArrayDeque

Deque<Type> dq = new ArrayDeque<>();

入栈，出栈 均非同步

* 入栈：ArrayDeque

文本

描述已自动生成

* 出栈：ArrayDeque

文本

描述已自动生成

* 判空：ArrayDeque

文本

描述已自动生成

#### 《2》 方式2：LinkedList

Deque<Type> dq = new LinkedList<>();

### 实现选择

优先选Deque：

<https://docs.oracle.com/javase/10/docs/api/java/util/Stack.html>

A more complete and consistent set of LIFO stack operations is provided by the [Deque](https://docs.oracle.com/javase/10/docs/api/java/util/Deque.html) interface and its implementations, which should be used in preference to this class. For example:

Deque<Integer> stack = new ArrayDeque<Integer>();

## （3） 队列

<https://blog.csdn.net/securitit/article/details/46919149>

<https://blog.csdn.net/Mcdull__/article/details/116919642>

### 【1】 类1：FIFO队列

#### 《1》 法1：LinkedList

##### 1） 实例化+初始化

Queue<Type> queue = new LinkedList<>();

##### 2） 操作

###### 1》 增

add：LinkedList

文本

描述已自动生成

offer：LinkedList

文本

描述已自动生成

###### 2》 删

remove：LinkedList

文本

描述已自动生成

文本

描述已自动生成

poll：LinkedList

文本

描述已自动生成

##### 3） 规定

刷题时：

插入元素：用add

删除元素：若要返回值进行判断队列是否为空，用poll；否则，

用remove

#### 《2》 法2：ArrayDeque

##### 1） 实例化+初始化

Queue<Type> queue = new ArrayDeque<>();

##### 2） 操作

##### 3） 示例

<https://leetcode.cn/problems/lian-biao-zhong-dao-shu-di-kge-jie-dian-lcof/solution/gong-shui-san-xie-yi-ti-san-jie-zhan-dui-w3rz/>

### 【2】 类2：双端队列

#### 《1》 法1：ArrayDeque

##### 1） 实例化+初始化

1》 默认initial capacity：16

Deque<Type> deque = new ArrayDeque<>();

文本

描述已自动生成

##### 2） 操作

###### 1》 增

* 队首

addFirst：

文本

描述已自动生成

文本

描述已自动生成

offerFirst：

文本

描述已自动生成

* 队尾

addLast：

电脑屏幕的截图

描述已自动生成

文本

描述已自动生成

offerLast：

文本

描述已自动生成

###### 2》 删

* 队首

pollFirst：

电脑屏幕的手机截图

描述已自动生成

removeFirst：

图形用户界面, 文本

描述已自动生成

* 队尾

pollLast：

图形用户界面, 文本

描述已自动生成

removeLast：

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

###### 3》 改

###### 4》 查

* 队首

getFirst：

图形用户界面, 文本

描述已自动生成

peekFirst：

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

* 队尾

getLast:

电脑屏幕的手机截图

描述已自动生成

peekLast:

图形用户界面, 文本, 应用程序, 网站

描述已自动生成

###### 5》 判空

isEmpty:

文本

描述已自动生成

##### 3) 总结

* 增

队首：若要返回true，offerFirst；否则，addFirst

队尾：若要返回true，offerLast；否则，addLast

* 删

队首：若队列为空想抛异常，removeFirst；否则，pollFirst

队尾：若队列为空想抛异常，removeLast；否则，pollLast

* 改
* 查

队首：若队列为空想抛异常，getFirst；否则，peekFirst

队尾：若队列为空想抛异常，getLast；否则，peekLast

#### 《2》 法2：LinkedList

##### 1） 实例化+初始化

1》 空Deque

Deque<Type> deque = new LinkedList<>();

图形用户界面, 文本, 应用程序

描述已自动生成

##### 2） 操作

###### 1》 增

* 队首：

addFirst:

图形用户界面, 文本

描述已自动生成

文本

描述已自动生成

offerFirst

文本

描述已自动生成

* 队尾：

addLast：

文本

描述已自动生成

文本

描述已自动生成

offerLast：

文本

描述已自动生成

###### 2》 删

* 队首：

removeFirst：

文本

描述已自动生成

文本

描述已自动生成

pollFirst：

文本

描述已自动生成

文本

描述已自动生成

* 队尾：

addLast：

文本

描述已自动生成

文本

描述已自动生成

pollLast：

屏幕的截图

描述已自动生成

文本

描述已自动生成

###### 3》 改

###### 4》 查

* 队首：

getFirst:

文本

描述已自动生成

peekFirst:

文本

描述已自动生成

* 队尾：

getLast：

文本

描述已自动生成

peekLast:

屏幕的截图

描述已自动生成

结合 滑动窗口最大值

##### 3) 总结

* 增

队首：若想返回true，offerFirst；否则，addFirst

队尾：若想返回true，offerLast；否则，addLast

* 删

队首：若队列为空想抛异常，removeFirst；否则，pollFirst

队尾：若队列为空想抛异常，removeLast；否则，pollLast

* 改
* 查

队首：若队列为空想抛异常，getFirst；否则，peekFirst

队尾：若队列为空想抛异常，getLast；否则，peekLast

#### 实现选择

1） 优先选ArrayDeque：

<https://docs.oracle.com/javase/tutorial/collections/implementations/deque.html>

* ArrayDeque is more efficient than the LinkedList for add and remove operation at both ends.
* The LinkedList implementation consumes more memory than the ArrayDeque implementation.

## （4） 哈希表

## （5） 集合

### 【1】 法1：HashSet

#### 《1》 场景

对迭代时元素顺序无特定要求时，优先用。三者中大多数操作的耗时最低（O（1））。

#### 《2》 实例化+初始化

1. 默认initial capacity：16，load factor：0.75

Set<Type> set = new HashSet<>();

图形用户界面, 文本

描述已自动生成

#### 《3》 操作

##### 1） 增

add：

文本

描述已自动生成

文本

描述已自动生成

##### 2） 删

remove：

文本

描述已自动生成

电脑萤幕的截图

描述已自动生成

##### 3） 改

##### 4） 查

### 【2】 法2：TreeSet

#### 《1》 场景

需要迭代时元素按value-ordered（值相关顺序，如natural ordring或Comparator/Comparable规定）进行时用。大多数操作的耗时比HashSet高很多（O（logn））。

#### 《2》 实例化+初始化

#### 《3》 操作

### 【3】 法3：LinkedHashSet

#### 《1》 场景

需要迭代时按insertion-ordered iteration (least recently inserted to most recently)时用。LinkedHashSet spares its clients from the unspecified, generally chaotic ordering provided by HashSet at a cost that is only slightly higher（LinkedHashSet让它的使用者免受HashSet提供的未指定的、通常是混乱的排序，其成本仅略高一些）

#### 《2》 实例化+初始化

1. 默认initial capacity（16），load factor（0.75）

Set<Type> set = new LinkedHashSet<>();

图形用户界面, 文本, 聊天或短信

描述已自动生成

#### 《3》 操作

##### 增

add：

文本

描述已自动生成

##### 删

remove：

文本

描述已自动生成

##### 改

##### 查

## （6） 树

## （7） 堆

### 【1】 法1：PriorityQueue（二叉堆）

#### 实例化+初始化

##### 小根堆

Queue<Type> queue = new PriorityQueue<>();

##### 大根堆

<https://leetcode.cn/problems/zui-xiao-de-kge-shu-lcof/solution/3chong-jie-fa-miao-sha-topkkuai-pai-dui-er-cha-sou/>

|  |
| --- |
| // 默认是小根堆，实现大根堆需要重写一下比较器。  Queue<Integer> pq = new PriorityQueue<>((v1, v2) -> v2 - v1); |

#### 《2》 相关操作

参考：

<https://docs.oracle.com/javase/10/docs/api/java/util/Queue.html>

表格

描述已自动生成

覃超：需要视情况决定用哪种，是用exception，还是null来判断。没有优先级。

##### 1） 插入

add：PriorityQueue

文本

描述已自动生成

offer：PriorityQueue

文本

描述已自动生成

##### 删除

remove：AbstractQueue

文本

描述已自动生成

remove（Object）：PriorityQueue

文本

描述已自动生成

poll：PriorityQueue

文本

描述已自动生成

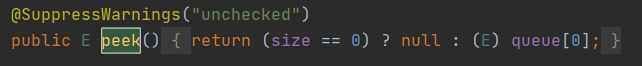
##### 获取堆顶元素

element：AbstractQueue

文本

描述已自动生成

peek：PriorityQueue



#### 《3》 规定

刷题时：

插入元素：用add（）/offer（）

删除头/堆顶元素：用poll（）

删除指定元素：用remove（Object）

获取头/堆顶元素：用peek（）

## （8） 图

# 算法

## （1） 排序

【1】 Arrays.sort方法

【2】 Collections.sort方法（其实调的是List.sort方法）