学习笔记

1.数据结构&算法

1.2.数据结构

- 一维：
  - 基础：数组array(string),链表linked list
  - 高级：栈stack，队列queue，双端队列deque，集合set，映射map（hash or map），etc
- 二维：

    - 基础：树tree，图graph
    - 高级：二叉搜索树binary search tree（red-black tree，AVL），堆heap，并查集disjoint set，字典树Trie，etc
- 特殊：
    - 位运算 Bitwise，布隆过滤器 BloomFilter
    - LRU Cache

1.3.算法

- if-else，switch --> branch
- for,while loop --> Iteration
- 递归 Recursion（Divide & Conquer,Backtrace)
- 搜索 Search:深度优先搜索 Depth first search，广度优先搜索Br eadth first search，A*，etc
- 动态规划 Dynamic Programming
- 二分查找 Binary Search
- 贪心 Greedy
- 数学 Math，几何 Geometry

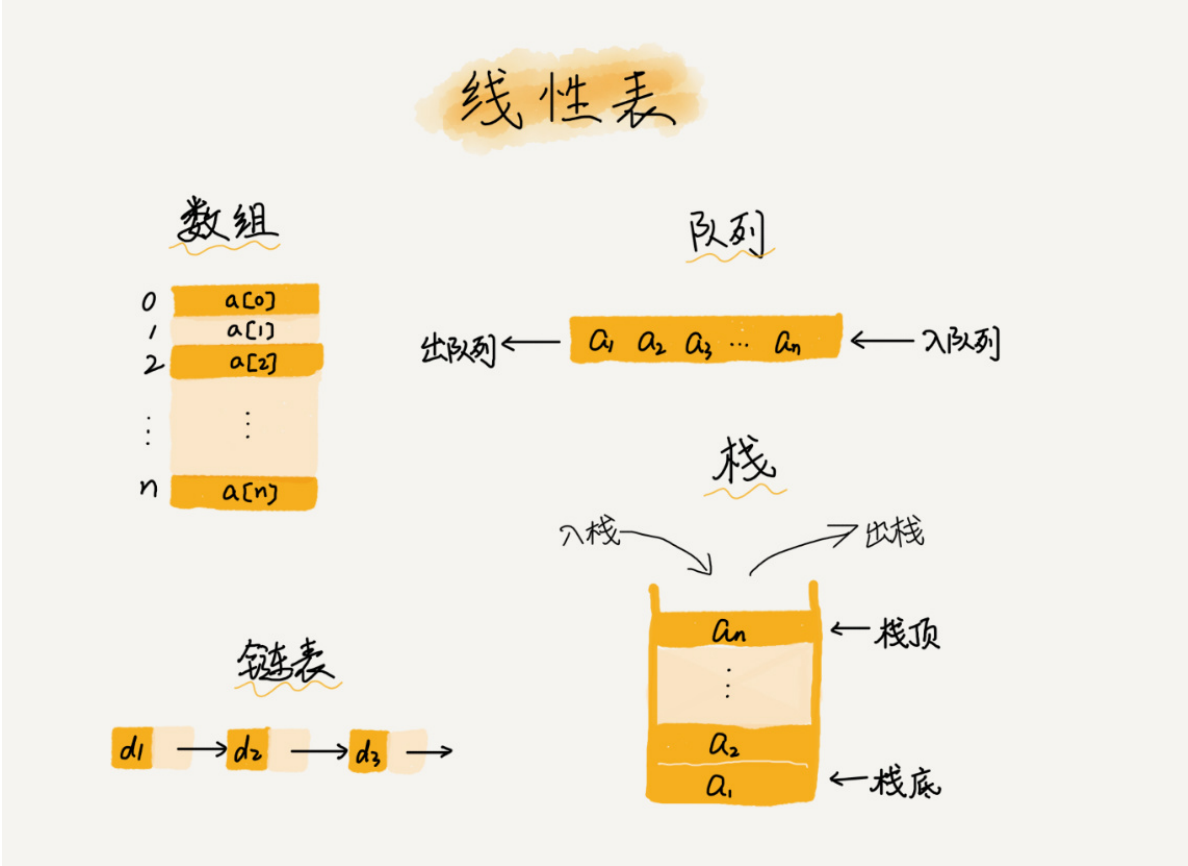
2.数组、链表、跳表

2.1.数组(Array)、链表(Linked List)、跳表(skip List)

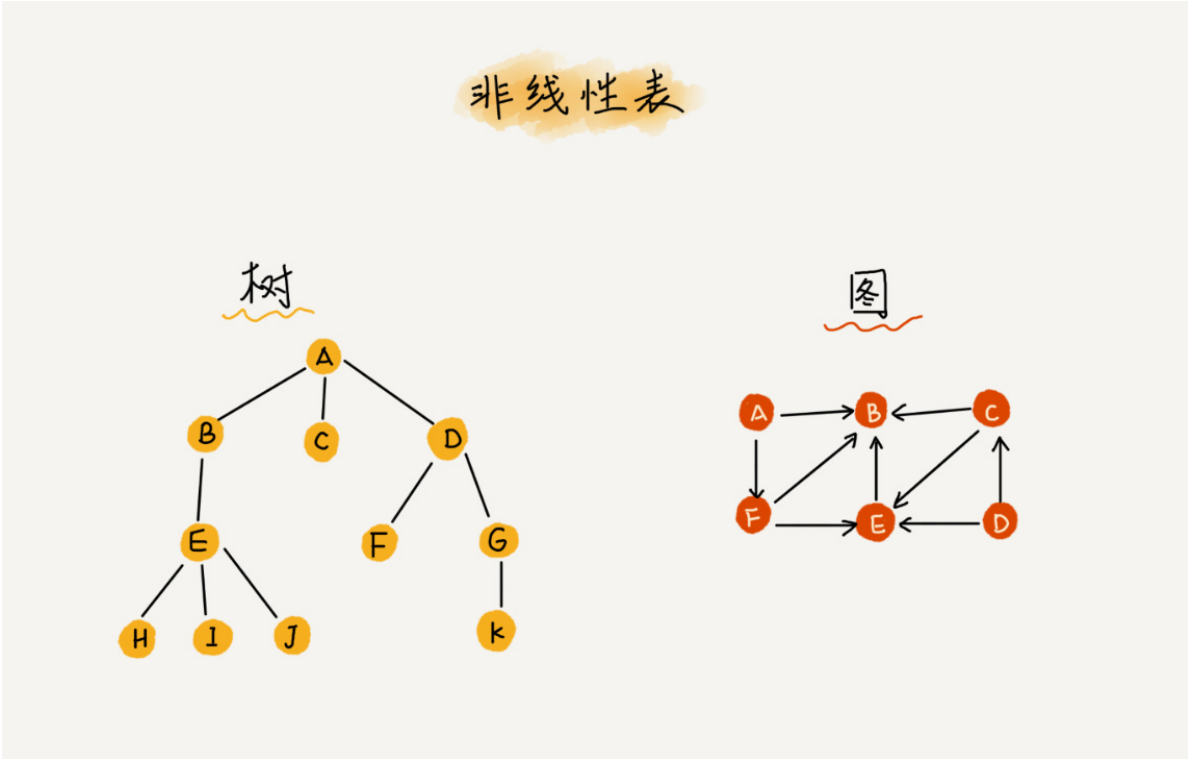
2.1.1.数组（Array)

数组(Array)是一种**线性表**数据结构。使用一组**连续的内存空间**，来存储一组具有**相同类型**的数据。[1]

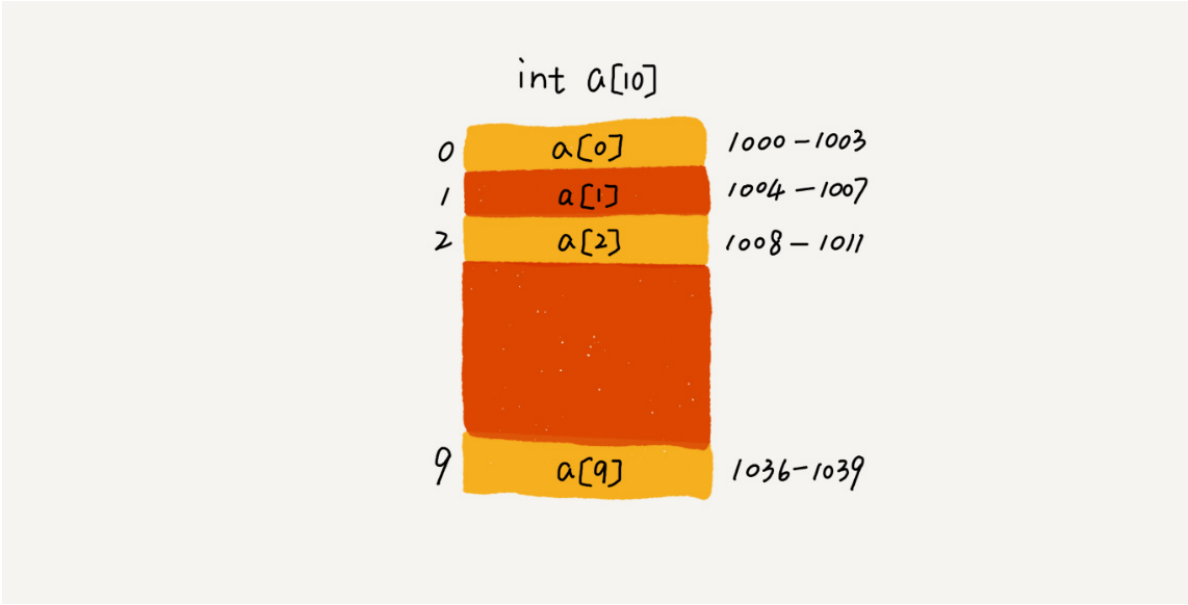
线性表(Linear List)就是数据排成像一条线一样的结构。每个线性表上的数据最多只有两个方向。除了数组，链表、队列、栈也是线性表结构。



非线性表(Nonlinear List)中的数据之间不是简单的前后关系，比如二叉树、堆、图等。



连续的内存空间和相同的数据类型：



数组的操作函数的复杂度

函数	功能描述	时间复杂度
prepend	在数组前端插入元素	O(1)
append	在数组后端插入元素	O(1)
lookup	查找	O(1)
insert	插入	O(n)
delete	删除	O(n)
size	求元素个数	
isEmpty	判空	

数组的创建

```
ArrayList<Integer> array = new ArrayList<Integer>();
```

Source for java.util.ArrayList

函数	返回值	作用
add(Object element)	boolean	
add(int index, Object element)	void	
addAll(Collection c)	boolean	
addAll(int index, Collection c)	boolean	
clear()	void	
clone()	Object	
contains(Object o)	boolean	
ensureCapacity(int minCapacity)	void	
equals(Object o)	boolean	
forEach(Consumer action)	void	
get(int index)	Object	
hashCode()	int	
indexOf(Object o)	int	
isEmpty()	boolean	
iterator()	Iterator	
lastIndexOf(Object o)	int	
listIterator()	ListInterator	
listIterator(int index)	ListInterator	
remove(Object o)	boolean	
remove(int index)	Object	
removeAll(Collection<?> c	boolean	
removeIf(Predicate filter)	boolean	
replaceAll(Collection<?> c)	boolean	
retainAll(Collection<?> c)	boolean	
set(int index, Object element)	Object	
size()	int	
sort(Comparator c)	Spliterator	
spliterator()	Spliterator	
subList(int fromIndex,int toIndex)	List	
toArray()	Object[]	

函数	返回值	作用
toArray(Object[] a)	Object[]	
trimToSize()	void	

顺序表的优缺点

优点

- 取数据元素操作的时间效率较高(即时间复杂度低)，内存空间利用效率高

缺点

- 插入和删除操作时间复杂度高

2.1.2.链表(Linked List)

链表(Linked List)整体看上去就像一条链子，每一个元素都有两个属性，即**该元素的值item**和**下一个元素next。**

链表操作函数的复杂度

函数	功能描述	时间复杂度
prepend	在链表前端插入元素	O(1)
append	在链表后端插入元素	O(1)
lookup	查找	O(n)
insert	在链表的某个位置插入元素	O(1)
getElement	获取链表对应位置的元素	
indexOf	获取某元素在链表中的索引	
update	修改链表中某个位置上的元素的值	
removeAt	移除链表中某位置上的元素	
remove	移除链表中的某元素	O(1)
size	链表长度	
isEmpty	是否为空	
toString	以字符串的形式展示链表内的所有元素	

链表的实现

要设计单链表类，需要先设计结点类。一个结点类的成员变量有两个，一是结点的数据元素，另一个是表示下一个结点的next。

链表有：单链表、双向链表、循环链表

单链表结点类设计如下

```
public class Node {
    Object data;//结点的数据元素
    Node next;//表示下一个结点的对象引用
    Node(Object obj, Node next){
        this.data = obj;
        this.next = next;
    }
    Node(Object obj){
        this(obj,null);
    }
}
```

链表有：单链表、双向链表、循环链表

链表的创建

```
LinkedList<Integer> linkedList = new LinkedList<>();
```

Source for java.util.LinkedList

函数	返回值	作用
add(Object element)	boolean	
add(int index, Object element)	void	
addAll(Collection c)	boolean	
addAll(int index, Collection c)	boolean	
addFirst(Object e)	void	
addLast(Object e	void	
clear()	void	
clone()	Object	
contains(Object o)	boolean	
descendingIterator()	Iterator	
element()	Object	
get(int index)	Object	
getFirst()	Object	
getLast()	Object	
IndexOf(Object o)	int	
lastIndexOf(Object o)	int	
listIterator(int index)	ListIterator	
offer(Object e)	boolean	
offerFirst(Object e)	boolean	
offerLast(Object e)	boolean	
peek()	Object	
peekFirst()	Object	
peekLast()	Object	
poll()	Object	
pollFirast()	Object	
pollLast()	Object	
pop()	Object	
push(Object e)	void	
remove()	Object	
remove(Object o	boolean	

函数	返回值	作用
remove(int index)	Object	
removeFirst()	Object	
removeFirstOccurrence(Object o)	Object	
removeLast()	Object	
removeLastOccurrence(Object o)	Object	
equals(Object o)	boolean	
hashCode()	int	
isEmpty()	boolean	
iterator()	Iterator	
listIterator()	ListInterator	
set(int index, Object element)	Object	
size()	int	
sort(Comparator c)	Spliterator	
spliterator()	Spliterator	
toArray()	Object[]	
toArray(Object[] a)	Object[]	
subList(int fromIndex,int toIndex)	List	
toArray(IntFunction generator)	Object[]	
toString()	String	
forEach(Consumer action)	void	

2.3.跳表

注意：只能用于元素有序的情况

所以，跳表（skip list）对标的是平衡树（AVL Tree）和二分查找，是一种 插入/删除/搜索 都是O(logn)的数据结构，1989年出现。

最大的优势是原理简单，容易实现，方便扩展，效率更高，因此在一些热门的项目里用来替代平衡树，如Redis，levelDB等。

升维思想+空间换时间

2.4.参考链接

- Java 源码分析（ArrayList）
- Linked List 的标准实现代码
- Linked List 示例代码
- Java 源码分析（LinkedList）
- LRU Cache - Linked list： LRU 缓存机制
- Redis - Skip List：跳跃表、为啥 Redis 使用跳表（Skip List）而不是使用 Red-Black?

leedcode题目：
146. LRU缓存机制

- 使用java语音中自带的数据结构LinkedHashMap实现

```
/*
java自带的数据结构 LinkedHashMap,可以完成本题
 */
class LRUCache extends LinkedHashMap<Integer, Integer>{
    private int capacity;

    public LRUCache(int capacity){
        super(capacity, 0.75F, true);
        this.capacity = capacity;
    }

    public int get(int key){
        return super.getOrDefault(key, -1);
    }

    public void put(int key, int value){
        super.put(key, value);
    }

    protected boolean removeEldesEntry(Map.Entry<Integer, Integer> eldest){
        return size() > capacity;
    }
}
```

- 使用哈希表和双向链表实现

哈希表用于定位，便于找到缓存项在双向链表中的位置。

(1) 对于get操作，首先判断ke是否存在：
-- 如果key不存在，则返回-1；
-- 如果key存在，则key对应的节点是最近被使用的节点。通过哈希表定位到该节点在双向表中的位置，并将其移动到双向表的头部，最后返回该节点的值。
(2) 对于put操作，首先判断key是否存在：
-- 如果key不存在，使用key和value创建一个新的节点，在双向链表的头部添加该节点，并将key和该节点添加进哈希表中。然后判断双向链表的节点数是否超出容量，如果超出容量，则删除双向链表的尾部节点，并删除哈希表中对应的项；
-- 如果key存在，则与get操作类似，先通过哈希表定位，再将对应的节点的值更新为value，并将该节点移到双向链表的头部。

```
/*
```

```
使用哈希表和双向链表实现
 */
public class LRUCache {
    //建立一个双向链表类
    class DLinkedNode{
        int key;
        int value;
        DLinkedNode prev;
        DLinkedNode next;
        public DLinkedNode(){}
        public DLinkedNode(int _key, int _value){
            key = _key;
            value = _value;

        }
    }

    private Map<Integer, DLinkedNode> cache = new HashMap<>();
    private int size;
    private int capacity;
    private DLinkedNode head, tail;

    public LRUCache(int capacity) {
        this.size = 0;
        this.capacity = capacity;
        //使用伪头部和伪尾部节点
        head = new DLinkedNode();
        tail = new DLinkedNode();
        head.next = tail;
        tail.prev = head;
    }

    public int get(int key) {
        DLinkedNode node = cache.get(key);
        if (node == null){
            return -1;
        }
        //如果key存在，先通过哈希表定位，再移动到头部
        moveToHead(node);
        return node.value;
    }

    public void put(int key, int value) {
        DLinkedNode node = cache.get(key);
        if (node == null){
            //如果key不存在，创建一个新节点
            DLinkedNode newNode = new DLinkedNode(key,value);
            //添加进哈希表
            cache.put(key,newNode);
            //添加至双向链表的头部
            addToHead(newNode);
            size++;
            if (size > capacity){
                //如果超出容量，删除双向链表的尾部节点
                DLinkedNode tail = removeTail();
                //删除哈希表中对应的项
                cache.remove(tail.key);
                size--;
```

```
            }
        } else {
            //如果key存在，先通过哈希表定位，在修改value,并移到头部
            node.value = value;
            moveToHead(node);
        }
    }

    private void addToHead(DLinkedNode node) {
        node.prev = head;
        node.next = head.next;
        head.next.prev = node;
        head.next = node;
    }

    private void removeNode(DLinkedNode node){
        node.prev.next = node.next;
        node.next.prev = node.prev;
    }

    private DLinkedNode removeTail() {
        DLinkedNode res = tail.prev;
        removeNode(res);
        return res;
    }

    private void moveToHead(DLinkedNode node) {
        removeNode(node);
        addToHead(node);
    }
}
```

2.5.实战题目

Array 实战题目

2.5.1.leedcode题目：11.盛水最多的容器

- 第一种解法：双指针法

```
//双指针法
public static int maxArea1(int[] height) {
    int left = 0, right = height.length - 1;
    int ans = 0;
    while (left < right){
        int area = Math.min(height[left],height[right]) * (right - left);
        ans = Math.max( ans , area);
        if (height[left] <= height[right]){
            left++;
        } else {
            right--;
        }
    }
    return ans;
}
```

复杂度：时间复杂度O(n)，空间复杂度O(1)

2.5.2.leedcode题目：283.移动零

- 第一种解法：暴力求解

```
public void moveZeroes2(int[] nums){
    int[] ans = new int[nums.length];
    int j = 0;
    for (int i = 0; i < nums.length; i++){
        if (nums[i] != 0){
            ans[j] = nums[i];
            j++;
        }
    }
    for (int i = 0 ; i < nums.length ; i++){
        nums[i] = ans[i];
    }
}
```

复杂度：时间复杂度O(n)；空间复杂度O(n)。

- 第一种解法：快慢指针法

```
public void moveZeroes3(int[] nums){
    if (nums.length == 0) return;
    int j = 0;
    for (int i = 0; i < nums.length; i++){
        if (nums[i] != 0){
            int temp = nums[j];
            nums[j] = nums[i];
            nums[i] = temp;
            j++;
        }
    }
}
```

复杂度：时间复杂度O(n)；空间复杂度O(1)

2.5.3.leedcode题目：70.爬楼梯

- 第一种解法：递归

```
//递归法
public int climbStairs(int n) {
    if (n <=2 ) return n;
    return climbStairs(n-1)+climbStairs(n-2);
}
```

- 第二种解法：动态规划

```
//动态规划
public int climbStairs2(int n){
    int a = 0, b = 0;
    int ans = 1;
    for (int i = 0; i < n; i++){
        a = b;
        b = ans;
        ans = a + b;
    }
    return ans;
}
```

- 第三种解法：数学矩阵

```

```

- 第四种解法：通项公式

```

```

2.5.4.leedcode题目：15.三数之和

- 第一种解法：

```
public class Solution15 {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> ans = new ArrayList<>();
        int len = nums.length;
        if (nums == null || len < 3) return ans;
        Arrays.sort(nums);
        for (int i = 0; i < len; i++){
            if (nums[i] > 0) break;//若当前数大于0，则三数之和一定大于0 ，结束循环
            if (i > 0 && nums[i] == nums[i - 1]) continue;//去重
            int L = i + 1;
            int R = len - 1;
            while (L < R){
                int sum = nums[i] + nums[L] + nums[R];
                if (sum == 0){
                    ans.add(Arrays.asList(nums[i],nums[L],nums[R]));
                    while (L < R && nums[L] == nums[L+1]) L++;//去重
                    while (L < R && nums[R] == nums[R-1]) R--;//去重
                    L++;
                    R--;
                } else if (sum < 0) L++;
                else R--;
            }
        }
        return ans;
    }
}
```

2.5.5.leedcode题目：26.删除数组中的重复项

Linked List 实战题目

2.5.6.leedcode题目：206. Reverse Linked List

2.5.7.leedcode题目：24. Swap Nodes in Pairs

2.5.8.leedcode题目：141. Linked List Cycle

2.5.9.leedcode题目：142. Linked List Cycle II

2.5.10.leedcode题目：25. Reverse Nodes in k-Group

3.栈、队列、优先队列、双端队列

3.1.栈(Stack)、队列(queue)、双端队列(Deque)

- *Stack关键点：* 先入后出；添加、删除皆为O（1）
- *Queue关键点：* 先入先出；添加、删除皆为O（1）
- *双端队列：* 两端可以进出的Queue--double ended queue；添加、删除皆为O（1）

高性能的 container datatyps 库

栈（Stack） java实现

代码示例：

Java

```
Stack<Integer> stack = new Stack<>();
stack.push(1);
stack.push(2);
stack.push(3);
stack.push(4);
stack.push(5);
System.out.println(stack);
System.out.println(stack.search(4));

stack.pop();
stack.pop();
Integer topElement = stack.peek();
System.out.println(topElement);
System.out.println(" 3的位置 " + stack.search(3));
```

Python

```
class Stack:
    def __int__(self):
        self.items=['x','y']

    def push(self,item):
        self.items.append(items)

    def pop(self):
        self.items.pop()

    def lengh(self);
        return len(self.items)
```

队列（queue） java实现，python 的 heapq

代码示例：

Java

```
Queue<String> queue = new LinkedList<>();
queue.offer("one");
queue.offer("two");
queue.offer("three");
queue.offer("four");

String polledElement = queue.poll();
System.out.println(polledElement);
System.out.println(queue);

String peekedElement = queue.peek();
System.out.println(peekedElement);
System.out.println(queue);

while (queue.size() > 0){
    System.out.println(queue.poll());
}
```

python

```
class Queue:
    def __int__(self):
        self.queue=[]

    def enqueue(self,item):
        self.queue.append(item)

    def dequeue(self);
        if len(self.queue) < 1:
            return None;
        return self.queue.pop(0)

    def size(self):
        return len(self.queue)
```

双端队列(Deque)

代码示例：

```
Deque<String> deque = new LinkedList<>();

deque.push("a");
deque.push("b");
deque.push("c");
System.out.println(deque);

String str = deque.peek();
System.out.println(str);
System.out.println(deque);

while (deque.size()>0){
    System.out.println(deque.pop());
}

System.out.println(deque);
```

3.2.优先队列(Priority Queue)

- 1.插入操作：O(1)
- 2.取出操作：O(logN) - 按照元素的优先级取出
- 3.底层具体实现的数据额结构较为多样和复杂：heap、bst，treap

Java 的 PriorityQueue

3.3.实战题目

3.3.1.leedcode题目：20.有效的括号

```
public class Solution20 {
    public boolean isValid(String s) {
        //如果是空字符串，可认为是有效字符串，返回true
        if (s == null ) return true;
        //如果长度是奇数，那么肯定不是有效字符串，直接返回false
        if (s.length() % 2 == 1) return false;
        //将括号用哈希表表示，右括号为键值，
        Map<Character , Character> map = new HashMap<>(){{
            put(')','(');
            put(']','[');
            put('}','{');
        }};
        //初始化栈
        Deque<Character> stack = new LinkedList<>();
        for (int i = 0; i < s.length(); i++){
            Character ch = s.charAt(i);
            //如果ch是右括号，就与栈顶元素做匹配，
            if (map.containsKey(ch)){
                //若栈是空的或者栈顶元素不是对应的左括号，返回false
                if (stack.isEmpty() || stack.peek() != map.get(ch)) return
false;//与栈顶元素匹配
                //否则与栈顶元素可匹配，则将栈顶元素出栈
                stack.pop();//出栈
            } else {//ch是左括号，入栈
```

```
                stack.push(ch);//入栈
            }
        }
        //左右括号一一匹配后，栈内元素应该为空，不为空，也说明不是有效字符串
        return stack.isEmpty();
    }
}
```

3.3.2.leedcode题目：155.最小栈

3.3.3.leedcode题目：84.柱状图中最大的矩形

3.3.4.leedcode题目：239.滑动窗口最大值

3.3.5.leedcode题目：641.设计循环双端队列

3.3.6.leedcode题目：42.Trapping Rain Water

3.3.7.课后作业

用 add first 或 add last 这套新的 API 改写 Deque 的代码

3.3.8.课后作业

分析 Queue 和 Priority Queue 的源码

重点学习20个最常用的最基础的数据结构和算法

- 10个数据结构: 数组，链表，栈，队列，散列表，二叉树，堆，跳表，图，Trie树
- 10个算法： 递归，排序，二分查找，搜索，哈希算法，贪心算法，分治算法，回溯算法，动态规划，字符串匹配算法

参考

- [1][数据结构之数组](https://www.cnblogs.com/fengxiaoyuan/p/10934399.html)
- [2]数据结构--Java语音描述，朱战立，清华大学出版社，2015年12月第一版