栈和队列
stack & queue

无隅

栈和队列

- 栈： LIFO(Last in first out)　后进先出
- 队列： FIFO(First in first out) 先进先出

栈的初始化与基本操作

- Java 类库 : Stack<String> stack = new Stack<>();
- 基本操作：

方法	参数	返回值	时间复杂度
push	Element	Element/void	O(1)
pop	void	Element	O(1)
peek	void	Element	O(1)
isEmpty	void	boolean	O(1)

栈的实现

- 定义属性字段

1. 在数组的基础上实现
2. 属性： elements, size, capacity
3. 构造器实现

- 定义方法

1. push, pop, peek, isEmpty

栈的实现 – 属性及构造器

```java
public class Stack<T> {
    private int size;
    private int capacity;
    private T[] elementData;

    public Stack(int capacity) {
        this.size = 0;
        this.capacity = capacity;
        this.elementData = new T[capacity];
    }
}
```

栈的实现 – 方法、函数

```
public T push(T element) {
    if (size == capacity) {
        resize();
    }
    elementData[size++] = element;
    return element;
}
```

```
public T peek() {
    if (this.size == 0) {
        // throw Exception
        throw new IllegalStateException();
    }
    return elementData[size - 1];
}
```

```
public T pop() {
    if (size == 0) {
        // throw Exception
        throw new EmptyStackException();
    }

    T element = elementData[--this.size];
    return element;
}
```

```
public boolean isEmpty() {
    return this.size == 0;
}
```

什么时候考虑使用栈

- 调用函数
- 递归
- 深度优先搜素 DFS(Depth-first Search)

栈和堆在计算机操作系统上的概念

- 栈区 stack

1. 存储 primitive variables function call
2. 栈区的读取速度更快

- 堆区 heap

1. 堆区存放引用类型变量
2. 堆区可以动态地分配内存空间

队列的初始化与基本操作

- Java 类库 : Queue<String> queue = new LinkedList<>();
- 基本操作：

方法	参数	返回值	时间复杂度
offer (add)	Element	boolean	O(1)
poll (remove)	void	Element	O(1)
Peek (element)	void	Element	O(1)

队列的实现

- 定义属性字段

1. 在链表的基础上实现
2. 属性：　elements, size, capacity
3. 构造器实现

- 定义方法

1. offer, poll, peek, isEmpty

队列的实现 – 属性及构造器

什么时候考虑使用队列

- 广度优先搜索 BFS (Breadth-first Search)
- 优先队列 Priority Queue (Heap)
- 多任务调度

栈与队列典型题目

用栈实现队列

使用栈实现队列的下列操作：
push(x) -- 将一个元素放入队列的尾部。
pop() -- 从队列首部移除元素。
peek() -- 返回队列首部的元素。
empty() -- 返回队列是否为空。
注意：
你只能使用标准的栈操作。
假设所有操作都是有效的 （例如，一个空的队列不会调用 pop 或者 peek 操作）。

(https://leetcode-cn.com/problems/implement-queue-using-stacks/description/)

```java
class MyQueue {

    /** Initialize your data structure here. */
    private Stack<Integer> newStack;
    private Stack<Integer> oldStack;
    public MyQueue() {
        newStack = new Stack<Integer>();
        oldStack = new Stack<Integer>();
    }
    /** Push element x to the back of queue. */
    public void push(int x) {
        newStack.push(x);
    }
    /** Removes the element from in front of queue and returns that element. */
    public int pop() {
        if(oldStack.isEmpty()) {
            while(!newStack.isEmpty()) {
                oldStack.push(newStack.pop());
            }
        }
        return oldStack.pop();
    }
    /** Get the front element. */
    public int peek() {
        if(oldStack.isEmpty()) {
            while(!newStack.isEmpty()) {
                oldStack.push(newStack.pop());
            }
        }
        return oldStack.peek();
    }
    /** Returns whether the queue is empty. */
    public boolean empty() {
        return newStack.isEmpty() && oldStack.isEmpty();
    }
}
```

```java
private void shiftStacks() {
    if(oldStack.isEmpty()) {
        while(!newStack.isEmpty()) {
            oldStack.push(newStack.pop());
        }
    }
}
```

最小栈

设计一个支持 push ， pop ， top 操作，并能在常量时间内检索最小元素的栈。

push(x) -- 将元素 x 推入栈中。

pop() -- 删除栈顶的元素。

top() -- 获取栈顶元素。

getMin() -- 检索栈中的最小元素。

```java
class MinStack {
    private Stack<Integer> stack = new Stack<>();
    private Stack<Integer> minStack = new Stack<>();

    public void push(int x) {
        stack.push(x);
        if (minStack.isEmpty() || x <= minStack.peek()){
            minStack.push(x);
        }
    }

    public void pop() {
        if (stack.pop().equals(minStack.peek())){
            minStack.pop();
        }
    }

    public int top() {
        return stack.peek();
    }

    public int getMin() {
        return minStack.peek();
    }
}
```

有效的括号

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串，判断字符串是否有效。

有效字符串需满足：

左括号必须用相同类型的右括号闭合。

左括号必须以正确的顺序闭合。

注意：空字符串可被认为是有效字符串

示例 1: 输入 : "()" 输出 : true

示例 2: 输入 : "()[]{}" 输出 : true

示例 3: 输入 : "(]" 输出 : false

示例 4: 输入 : "([)]" 输出 : false

示例 5: 输入 : "{[]}" 输出 : true

```java
public boolean isValid(String s) {
    if (s == null || s.length() == 0) {
        return true;
    }
    int length = s.length();
    Stack<Character> stack = new Stack<>();
    for (int i = 0; i < length; i++) {
        char ch = s.charAt(i);
        if (isLeft(ch)) {
            stack.push(ch);
        } else {
            if (!stack.isEmpty()) {
                char c = stack.pop();
                if (!isMatch(c, ch)) {
                    return false;
                }
            } else {
                return false;
            }
        }
    }
    return stack.isEmpty();
}
private boolean isLeft(char c) {
    return (c == '(')
        || (c == '[')
        || (c == '{');
}
private boolean isMatch(char left, char right) {
    if (left == '(') {
        return ')' == right;
    } else if (left == '[') {
        return ']' == right;
    } else if (left == '{') {
        return '}' == right;
    } else {
        return false;
    }
}
```

```java
public boolean isValid(String s) {
    if (s == null || s.length() == 0) {
        return true;
    }
    Stack<Character> stack = new Stack<>();
    int n = s.length();
    for (int i = 0; i < n; i ++) {
        if (s.charAt(i) == '(' || s.charAt(i) == '[' || s.charAt(i) == '{')  {
            stack.push(s.charAt(i));
        } else if (s.charAt(i) == ')') {
            if (stack.isEmpty() || stack.pop() != '(') return false;
        } else if (s.charAt(i) == ']') {
            if (stack.isEmpty() || stack.pop() != '[') return false;
        } else if(s.charAt(i) == '}') {
            if (stack.isEmpty() || stack.pop() != '{') return false;
        }
    }
    return stack.isEmpty();
}
```

总结

- 栈与队列的概念与基本实现
- 使用栈和队列的场景
- 典型面试题