

1. 基本介绍

1.1 基本介绍

编程的真相就是对数据的处理

- 那么在计算机中对于数据的组织和存储结构也会影响我们的效率。
- 常见的数据结构较多
 - 每一种都有其对应的应用场景，不同的数据结构 的不同操作 性能是不同的。
 - 有的查询性能很快，有的插入速度很快，有的是插入头和尾速度很快。
 - 有的做范围查找很快，有的允许元素重复，有的不允许重复等等。
 - 在开发中如何选择，要根据具体的需求来选择。
- 注意：数据结构和语言无关，常见的编程语言都有 直接或者间接 的使用上述常见的数据结构



线性结构

- 线性结构（英语：Linear List）是由 n ($n \geq 0$) 个数据元素（结点） $a[0]$, $a[1]$, $a[2]$..., $a[n-1]$ 组成的有限序列。
- 其中：
 - 数据元素的个数 n 定义为表的长度 = `"list".length()` (`"list".length() = 0` (表里没有一个元素) 时称为空表)。
 - 将非空的线性表 ($n \geq 1$) 记作：($a[0]$, $a[1]$, $a[2]$, ..., $a[n-1]$)。
 - 数据元素 $a[i]$ ($0 \leq i \leq n-1$) 只是个抽象符号，其具体含义在不同情况下可以不同。
- 上面是维基百科对于线性结构的定义，有一点点抽象，其实我们只需要记住几个常见的线性结构即可。



2. 数组

- 1、早期的 js 中的数组是按照链表来创建的，而不是直接创建连续的内存空间
- 2、链表对于 增删 的效率比较高，但是对于查找的效率不是很高。反而是数组对查找的效率比较高，因为直接根据下标值来查找即可

■ 数组 (Array) 结构是一种重要的数据结构:

- 几乎是每种编程语言都会提供的一种**原生数据结构** (语言自带的) ;
- 并且我们可以借助于数组结构来实现其他的数据结构, 比如栈 (Stack)、队列 (Queue)、堆 (Heap) ;

■ 通常数组的内存是连续的, 所以数组在知道下标值的情况下, 访问效率是非常高的

数组	X	I	A	O	F	U	G	E
下标	0	1	2	3	4	5	6	7
内存地址	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07

连续

■ 这里我们不再详细讲解TypeScript中数组的各种用法, 和JavaScript是一致的。

- https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array

3. 栈

3.1 基本介绍

基本介绍

■ 栈也是一种**非常常见**的数据结构, 并且在程序中的**应用非常广泛**。

■ 数组

- 我们知道数组是一种**线性结构**, 并且可以在数组的**任意位置**插入和删除数据。
- 但是有时候, 我们为了实现某些功能, 必须对这种**任意性**加以**限制**。
- 而**栈和队列**就是比较常见的**受限的线性结构**, 我们先来学习栈结构。

■ 栈结构示意图



■ 栈 (stack), 它是一种受限的线性结构, **后进先出(LIFO)**

- 其限制是仅允许在**表的一端**进行插入和删除运算。这一端被称为**栈顶**, 相对地, 把另一端称为**栈底**。
- LIFO(last in first out)表示就是后进入的元素, 第一个弹出栈空间。类似于自助餐托盘, 最后放上的托盘, 往往先把拿出去使用。
- 向一个栈插入新元素又称作**进栈**、**入栈**或**压栈**, 它是把新元素放到栈顶元素的上面, 使之成为新的栈顶元素;
- 从一个栈删除元素又称作**出栈**或**退栈**, 它是把栈顶元素删除掉, 使其相邻的元素成为新的栈顶元素。

■ 生活中类似于栈的

- 自助餐的托盘, 最新放上去的, 最先被客人拿走使用。
- 收到很多的邮件(实体的), 从上往下依次处理这些邮件。(最新到的邮件, 最先处理)
- 注意: 不允许改变邮件的次序, 比如从最小开始, 或者处于最紧急的邮件, 否则就不再是栈结构了。而是队列或者优先级队列结构。

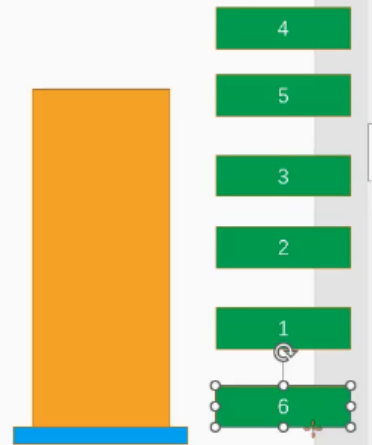
■ 面试题目:

🔍 题目

有六个元素6,5,4,3,2,1 的顺序进栈,问下列哪一个不是合法的出栈序列? ()
A. 5 4 3 6 1 2 D. 4 5 3 2 1 6 C. 3 4 6 5 2 1 D. 2 3 4 1 5 6

■ 题目答案: C

- A答案: 65进栈, 5出栈, 4进栈出栈, 3进栈出栈, 6出栈, 21进栈, 1出栈, 2出栈
- B答案: 654进栈, 4出栈, 5出栈, 3进栈出栈, 2进栈出栈, 1进栈出栈, 6出栈
- D答案: 65432进栈, 2出栈, 3出栈, 4出栈, 1进栈出栈, 5出栈, 6出栈



3.2 栈封装

■ 我们先来创建一个栈的类, 用于封装栈相关的操作

```
class Stack<T> {  
    private data: T[] = [];  
}
```

■ 代码解析:

- 我们创建了一个Stack, 用户创建栈的类, 可以定义一个泛型类。
- 在构造函数中, 定义了一个变量, 这个变量可以用于保存当前栈对象中所有的元素。
- 这个变量是一个数组类型。
- 我们之后无论是压栈操作还是出栈操作, 都是从数组中添加和删除元素。
- 栈有一些相关的操作方法, 通常无论是什么语言, 操作都是比较类似的。

■ 栈常见有哪些操作呢?

- **push(element):** 添加一个新元素到栈顶位置。
- **pop():** 移除栈顶的元素, 同时返回被移除的元素。
- **peek():** 返回栈顶的元素, 不对栈做任何修改 (这个方法不会移除栈顶的元素, 仅仅返回它)。
- **isEmpty():** 如果栈里没有任何元素就返回true, 否则返回false。
- **size():** 返回栈里的元素个数。这个方法和数组的length属性很类似。

3.2.1 数组封装

```
1 class ArrayStack<T = any> {
2     private data: T[] = [];
3
4     push(ele: T): void { this.data.push(ele); }
5     pop(): T | undefined { return this.data.pop(); }
6     peek(): T | undefined { return this.data[this.size() - 1]; }
7     isEmpty(): boolean { return this.data.length === 0; }
8     size(): number { return this.data.length; }
9 }
10
11 let arrayStack = new ArrayStack<string>();
12 arrayStack.push("aaa");
13 arrayStack.push("bbb");
14
15 console.log(arrayStack.peek());
16 console.log(arrayStack.pop(), arrayStack.pop(), arrayStack.pop());
17
```

问题 输出 调试控制台 终端

张嘉辉@LAPTOP-RVH1UC43 MINGW64 /d/CodeBuild/数据结构
\$ ts-node index.ts
bbb
bbb aaa undefined
张嘉辉@LAPTOP-RVH1UC43 MINGW64 /d/CodeBuild/数据结构
\$

```
class ArrayStack<T = any> {
    private data: T[] = [];

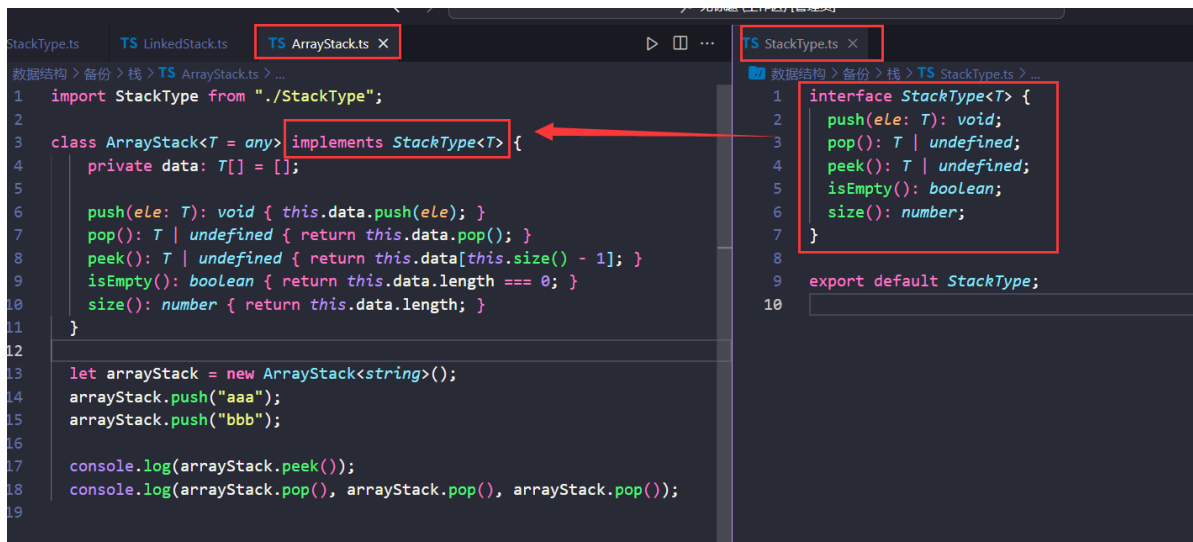
    push(ele: T): void { this.data.push(ele); }
    pop(): T | undefined { return this.data.pop(); }
    peek(): T | undefined { return this.data[this.size() - 1]; }
    isEmpty(): boolean { return this.data.length === 0; }
    size(): number { return this.data.length; }
}

let arrayStack = new ArrayStack<string>();
arrayStack.push("aaa");
arrayStack.push("bbb");

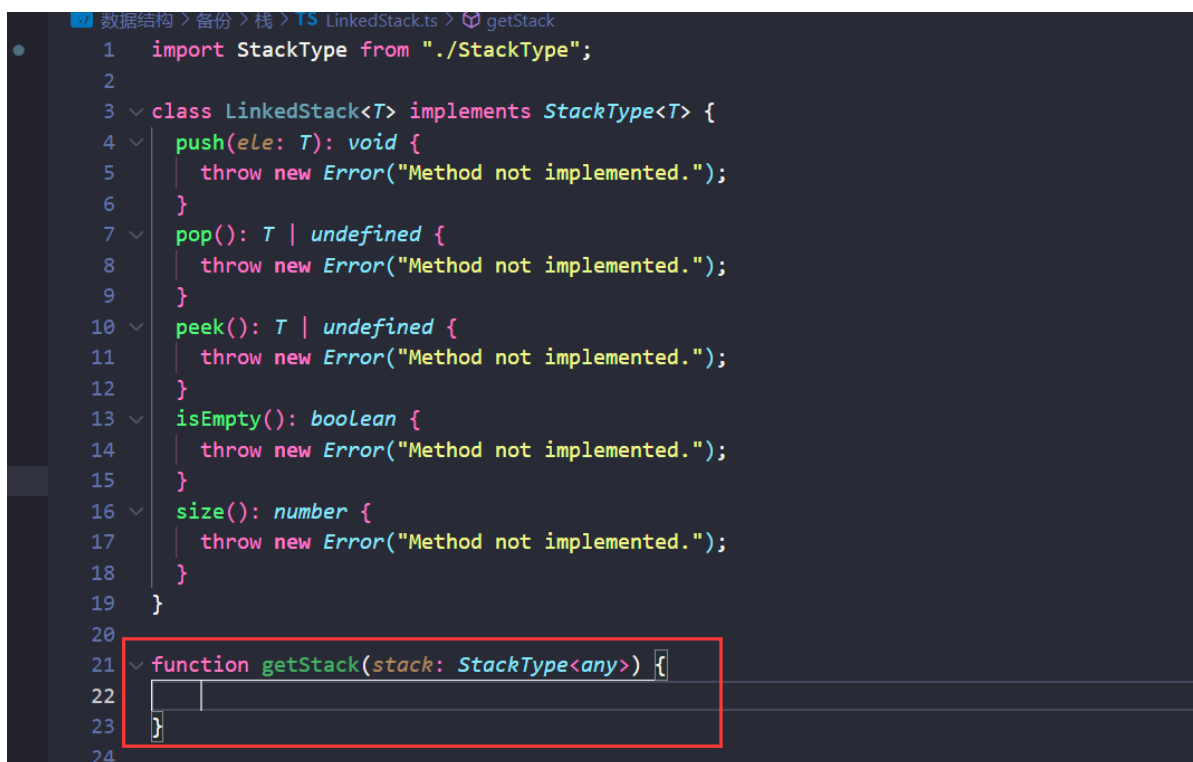
console.log(arrayStack.peek());
console.log(arrayStack.pop(), arrayStack.pop(), arrayStack.pop());
```

3.2.2 接口抽取

1、下面就是进行接口抽取的操作，这个操作主要是因为，栈的操作普遍存在下面的方法，如果我们要实现链表封装的话就可以继承该接口，实现代码编写的提示



2、继承可以实现多态的操作，这样我们就可以为下面的 `getStack` 传入不同实现的栈，只要能继承 `StackType` 就可以传入参数



3.3 面试题

3.3.1 十进制转二进制

```
1 import ArrayStack from "../实现/ArrayStack";
2
3 let arrayStack = new ArrayStack<number>();
4
5 function BinaryToDecimal(num: number): string {
6   while (num > 0) {
7     let BinaryData = num % 2;
8     arrayStack.push(BinaryData);
9     num = (num - BinaryData) / 2;
10  }
11
12  let str = "";
13  while (!arrayStack.isEmpty()) {
14    str += arrayStack.pop();
15  }
16
17  return str;
18 }
19
20 console.log(BinaryToDecimal(40));
21
```

```
1 import StackType from "../StackType";
2
3 class ArrayStack<T = any> implements StackType<T> {
4   private data: T[] = [];
5
6   push(ele: T): void {
7     this.data.push(ele);
8   }
9   pop(): T | undefined {
10    return this.data.pop();
11  }
12   peek(): T | undefined {
13    return this.data[this.size() - 1];
14  }
15   isEmpty(): boolean {
16    return this.data.length === 0;
17  }
18   size(): number {
19    return this.data.length;
20  }
21 }
```

张嘉辉@LAPTOP-RVH1UC43 MINGM64 /d/CodeBuild/数据结构/备份/栈/面试题

```
$ ts-node 十进制转二进制.ts
101000
```

张嘉辉@LAPTOP-RVH1UC43 MINGM64 /d/CodeBuild/数据结构/备份/栈/面试题

```
$
```

```
import ArrayStack from "../实现/ArrayStack";

let arrayStack = new ArrayStack<number>();

function BinaryToDecimal(num: number): string {
  while (num > 0) {
    let BinaryData = num % 2;
    arrayStack.push(BinaryData);
    num = (num - BinaryData) / 2;
  }

  let str = "";
  while (!arrayStack.isEmpty()) {
    str += arrayStack.pop();
  }

  return str;
}

console.log(BinaryToDecimal(40));
```

3.3.2 有效的括号

力扣官网: [20. 有效的括号 - 力扣 \(LeetCode\)](https://leetcode.cn/problems/valid-parentheses/)

```
数据结构 > 备份 > 栈 > 面试题 > TS 有效的括号.ts > matchParenthese
1  import ArrayStack from "../实现/ArrayStack";
2
3  let arrayStack = new ArrayStack<string>();
4  function matchParenthese(str: string) {
5      if (str.length % 2 !== 0) return false;
6
7      for (let i = 0; i < str.length; i++) {
8          const par = str[i];
9          if (par === "(") {
10             arrayStack.push("(");
11         } else if (par === "[" ) {
12             arrayStack.push("[");
13         } else if (par === "{") {
14             arrayStack.push("{");
15         } else {
16             let c = arrayStack.pop();
17             if (par !== c) {
18                 return false;
19             }
20         }
21     }
22
23     return arrayStack.isEmpty();
24 }
25 console.log(matchParenthese("{[]}()"));

问题 输出 调试控制台 终端
张嘉辉@LAPTOP-RVH1UC43 MINGW64 /d/codeBuild/数据结构/备份/栈/面试题
$ ts-node 有效的括号.ts
true
张嘉辉@LAPTOP-RVH1UC43 MINGW64 /d/codeBuild/数据结构/备份/栈/面试题
$
```

```
import ArrayStack from "../实现/ArrayStack";

let arrayStack = new ArrayStack<string>();
function matchParenthese(str: string) {
    if (str.length % 2 !== 0) return false;

    for (let i = 0; i < str.length; i++) {
        const par = str[i];
        if (par === "(") {
            arrayStack.push("(");
        } else if (par === "[" ) {
            arrayStack.push("[");
        } else if (par === "{") {
            arrayStack.push("{");
        } else {
            let c = arrayStack.pop();
            if (par !== c) {
                return false;
            }
        }
    }

    return arrayStack.isEmpty();
}

console.log(matchParenthese("{[]}()"));
// console.log(matchParenthese("("));
```

