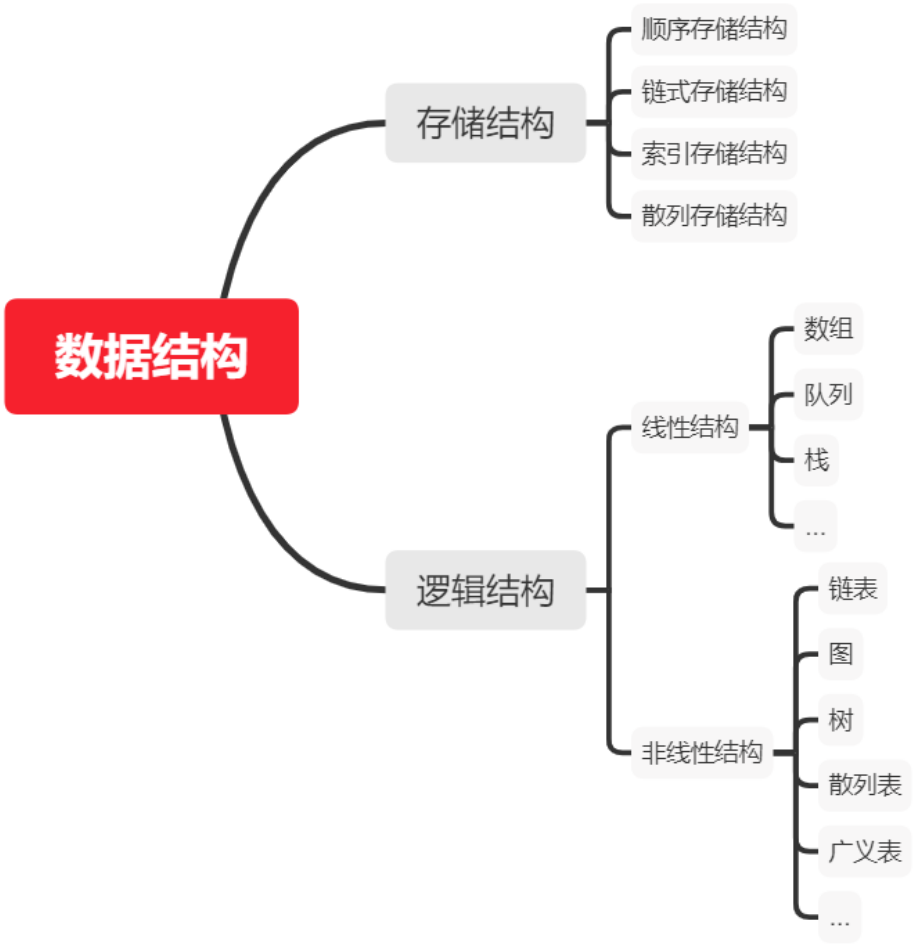
day05 数组

数据结构

数据结构主要是数据的一个存储和逻辑结构的体现。只要能**存储数据**的一个结构我们就称为数据结构。



相关知识点

- 数组 又被称为顺序表主要通过索引下标来进行访问（排序）
- 队列 先进先出的容器
- 栈　先进后出
- 树　二叉树
- 图　对应的指针的指向
- 链表 通过对应的指针来进行指向
- 散列表 hash表（hashcode来进行编码的）
- ...

数组

概述

数组是一种线性的数据结构，他可以存储对应的数据，一般通过对应的索引下标进行数据的访问。在一般情况下我们在数组里面存储的数据类型是一致的。（数组也可以存储不同数据类型的数据）

数组的定义

使用 [] 赋值来定义

```
var arr = []
//里面的数据使用,隔开
var arr = [1,2,3]
```

使用new关键词来定义

```
var arr = new Array();//没有数据的数组定义
//传入一个参数 表示定义的长度
var arr = new Array(10);//表示当前长度为10
//传入多个参数 表示赋值填入
var arr = new Array(1,2,3)//表示这个数组里面有三个值 分别为1,2,3
```

数组的特性

- 数组具备下标 可以通过下标来访问和赋值操作
- 数组具备length属性 length重新修改是可行（改大会进行扩容操作 改小会进行删除操作）

```
var arr = [1,2,3,4,5]
//通过下标访问 下标从0开始
console.log(arr[0])//访问第一个元素
arr[3] = 18
console.log(arr)
//可以做到 自动扩容
arr[10] = 20
console.log(arr)
//通过length来访问对应的长度
console.log(arr.length)
//length可以赋值 大于会进行扩容操作 小于的时候会进行删除操作
arr.length = 2
console.log(arr)
console.log(arr.length)
```

数组的遍历

使用普通的循环

```
var arr = [1,2,3,4]
for(var i=0;i<arr.length;i++){
    console.log(arr[i])
}
```

使用es5新增的for in关键词

```
//使用es5新增的for in循环 （for in循环是为了遍历对象 数组也是一个对象）
//for in遍历的是下标（对象的key）
for(var index in arr){
    // console.log(index)
    console.log(arr[index])
}
```

使用es6新增的for of关键词

```
//在for in基础上进行改版的 for of （专门遍历数组的）es6
//for of不能遍历对象 只能遍历数组（伪数组）
for(var value of arr){
    console.log(value)//值
}
```

for in 和 for of的区别

- for in是用于遍历对象的 他遍历的是对象的key（es5）
- for of是用于遍历数组的 他遍历的是数组的值 （es6）

练习

将一个数组中的第二个元素和第三个元素进行位置互换

```
var arr = [1,2,3,4,5]
//传入的对应的需要互换的位置
function fn(pos1,pos2,arr){
    //保存第一个位置的值
    var temp = arr[pos1-1]
    //将第一个位置的值进行覆盖
    arr[pos1-1] = arr[pos2-1]
    //将保存的第一个位置的值赋给第二个位置
    arr[pos2-1] = temp
}
fn(2,3,arr)
```

将数组 [2,3,1,5,6,7,3] 中的最大值和最小值的位置进行互换

```
//得到最大值和最小值的下标
function fn(arr){
    //假设第一个是最大值 同时也是最小值
    var max = 0 //下标
    var min = 0 //下标
    //将对应的最大值及最小值和其他的比较 如果比最大值还大那么你就是最大值 如果比最小值还小那么
就是最小值
    for(var i=1;i<arr.length;i++){
        if(arr[max]<arr[i]){ // 如果比最大值还大
            max = i //记录最大下标
        }
        if(arr[min]>arr[i]){// 如果比最小值还小
            min = i //记录最小下标
        }
    }
    //换位置
    var temp = arr[min]
    arr[min] = arr[max]
```

```
    arr[max] = temp
    return arr
}
console.log(fn([2,3,1,5,6,7,3]))
```

传入某个数组 返回第二大的数

```
function fn1(arr) {
    if(arr.length<2){
        return arr[0]
    }
    //得到最大值
    var max = arr[0] > arr[1] ? arr[0] : arr[1]
    //得到第二大的值
    var twoMax = arr[0] < arr[1] ? arr[0] : arr[1]
    //再去比对
    for (var i = 2; i < arr.length; i++) {
        //如果你比最大值大 将你的值记录变成最大值 当前的最大值变成第二大的值
        if (arr[i] > max) {
            twoMax = max
            max = arr[i]
        } else {
            //如果你比最大值小 那么比较你是否大于第二大的值 赋值操作
            if (arr[i] > twoMax) {
                twoMax = arr[i]
            }
        }
    }
    return twoMax
}
console.log(fn1([9,2,3,10,8,1]))
```

数组的相关方法

所有的存储空间都具备增删改查的方法

添加（add push set...）删除 (delete (删除直接回收) remove（删除完还在内存中） pop...)

- push 添加到后面（返回新的长度）

```
var arr = [1,2]
var newLength = arr.push(3)
console.log(newLength) //3
console.log(arr)//[1,2,3]
```

- pop 删除最后一个 （返回删除的元素）

```
var arr = [1,2]
var deleteItem = arr.pop()
console.log(deleteItem)
console.log(arr)//[1]
```

- shift 删除第一个 （返回删除的元素）

```
//shift和unshift
var arr = [1, 2]
var deleteItem =  arr.shift()
console.log(deleteItem)
console.log(arr)
```

- unshift 添加到第一个 （返回的新的长度）

```
var length = arr.unshift(3) //2
console.log(length)
console.log(arr)
```

修改 （replace set...）

覆盖

```
arr[0] = 新的值
```

先删再加 splice

```
var arr = [1,2,3]
//从下标为1的位置删除 删除一个 插入一个4和5
arr.splice(1,1,4,5)//新的数组 这个数组其实就原本的数组
console.log(arr)
```

splice的删除的操作

```
var arr = [1, 2, 3, 4]
//省略个数 删到最后（包含下标1）
arr.splice(1)
console.log(arr)
var arr = [1, 2, 3, 4]
//从下标1开始 删除俩个（包含下标1）
arr.splice(1,2)
console.log(arr)
```

splice的添加

```
//添加
var arr = [1,2,3,4]
arr.splice(2,0,5)
console.log(arr)
```

splice返回的是删除的数据组成的数组

查询（query select get...）

indexOf 根据传入的值查询第一次出现的下标

查询索引indexOf 传入对应的元素 返回的是index （如果没有返回-1）

```
var arr = [2, 4, 6, 8, 4]
console.log(arr.indexOf(4))
//从第三个开始数 从前往后数 第二个参数为查找坐标（从左到右）
console.log(arr.indexOf(4,2))//4
```

简单实现indexOf

```
var arr = [2, 4, 6, 8,4]
//模拟实现indexOf
function myIndexOf(value,start) {
    if (value == undefined) {
        throw new Error('参数未传递')
    }
    //如果没有传递start 那么应该为默认值0
    if(start == undefined){
        start = 0
    }
    //开始查询
    for (var i=start;i<arr.length;i++) {
        if (arr[i] == value) {
            return i
        }
    }
    return -1
}
console.log( myIndexOf(4))
```

lastIndexOf (从右到左)

```
var arr = [2, 4, 6, 8, 4]
console.log(arr.lastIndexOf(4))//4
console.log(arr.lastIndexOf(4,2))//1
```

简单实现

```
//模拟实现indexOf
function myLastIndexOf(value, start) {
    if (value == undefined) {
        throw new Error('参数未传递')
    }
    //如果没有传递start 那么应该为默认值0
    if (start == undefined) {
        start = arr.length
    }else if(typeof start != 'number'){
        throw new Error('参数类型不正确')
    }

    //开始查询
    for (var i = start; i >= 0; i--) {
        if (arr[i] == value) {
            return i
        }
    }
    return -1
}
```

```
console.log(myLastIndexOf(4))
console.log(myLastIndexOf(4,3))
```

以上方法都会对于原本的数组影响

不影响原本数组的方法（一定有返回值）

数组的拼接 concat方法

```
var arr1 = [1, 2]
var arr2 = [3, 4]
//数组的concat方法传入的参数为数组
var concatArr = arr1.concat(arr2)
console.log(arr1) //[1,2]
console.log(arr2) //[3,4]
console.log(concatArr) //[1,2,3,4]
//可以传入数组也可以传入对应的值
var concatArr = arr1.concat(3,4)
console.log(concatArr)
```

slice 截取 返回新的数组

```
var arr = [1, 2, 3, 4, 5, 6]
//传入一个开始下标 结束的下标（默认没有传入结束的下标截取到末尾）不包含结束的下标
var sliceArr = arr.slice(4,5)//[5]
console.log(sliceArr, arr)
```

join 将对应的数组转为字符串

```
var arr = [1, 2, 3, 4, 5]
//join如果不传参默认使用,来进行分割每个元素 如果传入参数那么使用参数来分割
var str = arr.join('a')//传入的参数一定是字符串
console.log(str)
```

位置变换的相关方法

sort 排序

```
//排序的相关方法
var arr = [2,3,5,10,4,6]
//默认情况下 sort方法是按照ascii码排序
// arr.sort()
//高阶函数用法 将函数作为参数的函数叫做高阶函数
var newArr = arr.sort(function(a,b){
    return a-b //正序 b-a就是倒序
})
console.log(arr)
console.log(newArr)
console.log(newArr == arr)
```

reverse 反转

```
//反转方法
var arr1 = arr.reverse()
console.log(arr)
console.log(arr1 == arr)
```

排序算法

常用的排序算法 (On2)

冒泡排序 (逐层冒泡)

选择排序 （选择一个数跟其他的进行比较）

插入排序 （在插入数据的时候的排序法）

希尔排序 （快速插入排序）

快速排序 （快速冒泡排序 （数据量不大的情况下最快））

归并排序 （大数据处理中最快的排序）

堆排序

桶排序

...

冒泡排序(On^2)

前一个跟后一个比较 俩俩相比 比较完就交换位置 直到所有的内容比较完

```
function bubleSort(arr){
    //轮数
    for(var i=0;i<arr.length-1;i++){
        //比较的次数
        for(var j=1;j<arr.length-i;j++){
            //俩俩比较
            if(arr[j]<arr[j-1]){
                var temp = arr[j]
                arr[j] = arr[j-1]
                arr[j-1] = temp
            }
        }
    }
    return arr
}
```

选择排序(On^2)

选择一个下标和所有的数进行比较 如果比较完发现这个下标不是之前的下标就换位置

```
function selectorSort(arr){
    for(var i=0;i<arr.length-1;i++){
        //指定max为当前遍历的下标
        var max = i
        //max要跟其他的所有未排序比较
        for(var j=i+1;j<arr.length;j++){
            if(arr[max]<arr[j]){
                max=j
```

```
            }
        }
        //如果最大值不是原本的下标位置就需要交换位置了
        if(max!=i){
            var temp = arr[max]
            arr[max] = arr[i]
            arr[i] = temp
        }
    }
    return arr
}
```

快速排序（二分排序）Onlogn

```
function quikSort(arr) {
    //如果你的数组长度只有一个 或者没有就直接返回这个数组
    if (arr.length <= 1) {
        return arr
    }
    //中间值取第一个
    var mid = arr[0]
    //左右的容器
    var left = [],
        right = []
    //循环比较
    for (var i = 1; i < arr.length; i++) {
        arr[i] > mid ? right.push(arr[i]) : left.push(arr[i])
    }
    //递归调用无限取中间值
    return quikSort(left).concat(mid).concat(quikSort(right))
}
```