# Day06

### 数据结构与算法

* 任何一门编程语言有的内容
* 数据结构和算法仅仅只是一个思想层面的内容，是可以用任何的编程语言来进行实现的

### 数据结构

* 数据存放的一种格式
* 一维的数据结构：数组、链表

### 算法

* 在不同的结构中间各种转换
* 多维数据结构（图）
* 二维数据结构中的特殊情况 （二叉树）

### 数组

* 一维数据结构（线性数据结构）：存储的顺序
* 1.数组是定长的
* 2.数组的存储是连续的
* 3.数组的偏移
* 4.arr[3] 方括号的内容叫偏移量（存储地址的偏移）
* 5.数组的偏移查询是性能最好的一种查询方式
* 数组的优缺点
* 优点：依靠偏移的特性，查询的性能（查询每一个指定位置）是最好的
* 缺点：
* 1.数组是定长的，导致数组中的元素很难被删除和添加
* 2.因为数据占的空间必须是连续的，如果数组比较大的时候，当系统空间碎片很多的时候，导致数组存不下

### 链表

* 链表：本质就是一个对象
* 一堆的数据结构（链表）多个对象形成一个关系
* 1.空间是不连续的
* 2.每存放的一个值，都要开辟一个新的引用空间
* 优点：
* 1.内存只要足够大，就可以存，完全不需要担心空间碎片的问题
* 2.删除和添加是非常方便的
* 缺点：
* 1.里面每个对象都有一个next属性，就会创建一个存储空间，消耗资源（next属性属于微乎其微）
* 2.查询速度会受到影响（指定位置的查询会受到影响）
* 每一个节点，都认为自己是根节点

### 一维数据结构算法 - 循环遍历

* 一维数据结构（线性数据结构）的遍历
* 遍历：将几个中的每一个元素进行获取并查看
* for
* while do while 做
* For in 遍历对象
* for of ES6中新增的，遍历可迭代对象
* for each ES5中新增的，for的简写

1. // 数组的遍历
2. **var** arr=[1,2,3,4,5,6,7,8]
3. **function** bianArr() {
4. **if** (arr==**null**) **return** ;
5. **for** (**var** i = 0; i < arr.length;i++) {
6. console.log(arr[i])
7. }
8. }
9. bianArr(arr)
10. // 链表的遍历
11. **function** Node(value) {// 构造函数
12. **this**.value=value
13. **this**.next=**null**
14. }
15. **var** node1=**new** Node(1)
16. **var** node2=**new** Node(2)
17. **var** node3=**new** Node(3)
18. **var** node4=**new** Node(4)
19. **var** node5=**new** Node(5)
20. node1.next=node2
21. node2.next=node3
22. node3.next=node4
23. node4.next=node5
25. **function** bianLian(root) {
26. **var** temp=root
27. **while** (**true**){
28. **if** (temp!=**null**){
29. console.log(temp.value)
30. } **else** {
31. **break**
32. }
33. temp=temp.next
34. }
35. }
36. bianLian(node1)

### 一维数组算法 - 递归

* 如何写递归
* 1.找公式 5! = 5 \* 4! n \* n- 1!
* 2.找出口1 = 1 0 = 1
* 优点：
* 1.节省代码
* 2.可以看得懂（这是递归的唯一优点）
* 缺点：
* 1.存储空间释放不了

### 斐波那契数列

* 传递具体的位数 看具体的位数是什么
* 1 1 2 3 5 8 13 21 34 55 ... 斐波那契数列
* 递归的效率最低的，但是可读性非常高，非常容易理解

1. **function** feiBo(n) {
2. **if** (n==1||n==2) **return** 1
3. **return** feiBo(n-1)+feiBo(n-2)
4. }
5. console.log(feiBo(10))  //55

### 链表的逆置

1. **function** Node(value) {
2. **this**.value=value
3. **this**.next=**null**
4. }
5. **var** node1=**new** Node(1)
6. **var** node2=**new** Node(2)
7. **var** node3=**new** Node(3)
8. **var** node4=**new** Node(4)
9. **var** node5=**new** Node(5)
10. node1.next=node2
11. node2.next=node3
12. node3.next=node4
13. node4.next=node5
15. **function** nizhi(root) {
16. **if** (root.next.next==**null**){ // 找到倒数第二个节点
17. // 让倒数第一个节点指向倒数第二个节点
18. root.next.next=root
19. **return** root.next
20. } **else** { //  如果是倒数第三个 以此类推 如果是第一个，就指向null
21. **var** abc=nizhi(root.next)
22. root.next.next=root
23. root.next=**null**
24. **return** abc
25. }
26. }
27. console.log(nizhi(node1))

### 双链表

* 优点： 不管你给出哪一个节点，都可以对整个链表进行遍历
* 缺点： 存储结构变得稍微复杂，思想简单，但是代码量会增大

1. **function** Node(value) {
2. **this**.value=value
3. **this**.next=**null**
4. }
5. **var** node1=**new** Node(1)
6. **var** node2=**new** Node(2)
7. **var** node3=**new** Node(3)
8. **var** node4=**new** Node(4)
9. **var** node5=**new** Node(5)
10. node1.next=node2
12. node2.next=node3
13. node2.prev=node1
15. node3.next=node4
16. node3.prev=node2
18. node4.next=node5
19. node4.prev=node3
21. node5.prev=node4
23. console.log(node3)

### 数组排序算法

* 冒泡
* 选择
* 快排

### 数组算法-冒泡

1. **var** arr=[5,9,1,3,6,7,2,8]
2. // 面试题 在进行第四轮冒泡排序以后，出现的结果是什么
3. **function** maopao(arr) {
4. // 第一次循环
5. // for (var i = 0; i < arr.length - 1; i++) {
6. //     if (arr[i]>arr[i+1]){
7. //         var num=arr[i+1]
8. //         arr[i+1]=arr[i]
9. //         arr[i]=num
10. //     }
11. // }
12. // 第二次循环
13. // for (var i = 0; i < arr.length - 2; i++) {
14. //     if (arr[i]>arr[i+1]){
15. //         var num=arr[i+1]
16. //         arr[i+1]=arr[i]
17. //         arr[i]=num
18. //     }
19. // }
20. **for** (**var** i = 0; i < arr.length; i++) {
21. **for** (**var** j = 0; j < arr.length-1-i; j++) {
22. **if** (arr[j]>arr[j+1]){
23. **var** num=arr[j+1]
24. arr[j+1]=arr[j]
25. arr[j]=num
26. }
27. }
28. }
29. **return** arr
30. }
31. console.log(maopao(arr))
32. // 扩展到原型
33. Array.prototype.mySort=**function** () {
34. **var** arr=**this**
35. **for** (**var** i = 0; i < arr.length; i++) {
36. **for** (**var** j = 0; j < arr.length-1-i; j++) {
37. **if** (arr[j]>arr[j+1]){
38. **var** num=arr[j+1]
39. arr[j+1]=arr[j]
40. arr[j]=num
41. }
42. }
43. }
44. **return** arr
45. }
46. console.log(arr.mySort())
47. **var** arr=[5,9,1,3,6,7,2,8]
48. // 1.比较
49. **function** compare(a,b) {
50. **if** (a>b){
51. **return** **true**
52. } **else** {
53. **return** **false**
54. }
55. }
56. // 2.交换
57. **function** exchange(arr,a,b) {
58. **var** temp=arr[a]
59. arr[a]=arr[b]
60. arr[b]=temp
61. }
63. **function** sort(arr) {
64. **if** (arr==**null** || arr.length==0) **return** []
65. **for** (**var** i = 0; i < arr.length; i++) {
66. **for** (**var** j = 0; j < arr.length-1-i; j++) {
67. **if** (compare(arr[j],arr[j+1])){
68. exchange(arr,j,j+1)
69. }
70. }
71. }
72. **return** arr
73. }
74. console.log(sort(arr))

### 数组算法-选择

1. // 选择排序：每一圈选择最大的数，放在最后面
2. **var** arr=[5,9,1,3,6,7,2,8]
3. // 1.比较
4. **function** compare(a,b) {
5. **if** (a<b){
6. **return** **true**
7. } **else** {
8. **return** **false**
9. }
10. }
11. // 2.交换
12. **function** exchange(arr,a,b) {
13. **var** temp=arr[a]
14. arr[a]=arr[b]
15. arr[b]=temp
16. }
18. **function** sort(arr) {
19. **if** (arr==**null** || arr.length==0) **return** []
20. **for** (**var** i = 0; i < arr.length; i++) {
21. **var** maxIndex=0 // 初始化一个索引
22. **for** (**var** j = 0; j < arr.length-i; j++) {
23. **if** (compare(arr[maxIndex],arr[j])){
24. maxIndex=j
25. }
26. }
27. exchange(arr,maxIndex,arr.length-1-i)
28. }
29. **return** arr
30. }
32. console.log(sort(arr))

### 数组算法-快排

1. // 简单快排
2. // 1.选择第一个数，小的站左边，大的站右边，再递归
4. **var** arr=[5,9,1,3,6,7,2,8,4];
5. **function** quickSort(arr) {
6. **if** (arr==**null**||arr.length==0) **return** [];

9. **var** leader=arr[0]
10. **var** left=[]
11. **var** right=[]
12. **for** (**var** i= 1; i < arr.length; i++) {
13. **if** (arr[i]<leader){
14. left.push(arr[i])
15. } **else** {
16. right.push(arr[i])
17. }
18. }
19. // 依次递归
20. left=quickSort(left)
21. right=quickSort(right)
22. left.push(leader)
23. **return** left.concat(right)
24. }
26. console.log(quickSort(arr))