# Day08

### 栈和队列

* 一维：栈和队列
* 栈：先入后出 相当于箱子
* 堆：先入先出 相当于管道

1. // 用数组去封装一个栈结构
2. **function** Stack(){
3. **this**.arr = [];
4. **this**.push = **function**(value){
5. **this**.arr.push(value)
6. }
7. **this**.pop = **function**(){
8. **return** **this**.arr.pop()
9. }
10. }
11. **var** stack = **new** Stack();
12. stack.push(1);
13. stack.push(2);
14. stack.push(3);
15. console.log(stack.arr)
16. stack.pop()
17. console.log(stack.arr)
18. // 用数组去封装一个队列结构
19. **function** Queue(){
20. **this**.arr = [];
21. **this**.push = **function**(value){
22. **this**.arr.push(value)
23. }
24. **this**.pop = **function**(){
25. **return** **this**.arr.shift()
26. }
27. }
28. **var** queue = **new** Queue();
29. queue.push(1)
30. queue.push(2)
31. queue.push(3)
32. console.log(queue.arr)
33. queue.pop()
34. console.log(queue.arr)

### 二维数组的数据结构

* 一维的数据结构：数组和链表
* 二维的数据结构：

1. 二维数组 一维数组进化
2. 拓扑学 只关心关系，不会去关心距离的
3. 二维拓扑结构 链表的进化

### 树形结构

* 树是图的一种
* 树 -- 有向五环图 --- 类似电脑文件夹
* 根节点
* 叶子节点：下边没有其他节点的时候
* 节点：不是根节点，也不是叶子节点，其他的都是节点
* 树的度：这个数有最多叉的节点，就管这个树叫几叉树
* 树的深度：树有几层，深度就为几

### 二叉树

* 满二叉树
* 1.所有的叶子节点必须在树的最底层
* 2.所有的非叶子节点都必须含有两个子节点
* 完全二叉树
* 国内的定义
* 1.叶子节点在最下面一层或者在倒数第二层
* 2.叶子节点都向左靠拢
* 国际定义
* 1.叶子节点在最下面一层或者在倒数第二层
* 2.如果有叶子节点，就必须有两个叶子节点
* 所有的节点都认为自己是根节点
* 左子树，右子树

### 二叉树算法-遍历

1. // 定义二叉树
2. **function** Node(value){
3. **this**.value = value;
4. **this**.left = **null**;
5. **this**.right = **null**;
6. }
7. **var** a = **new** Node("a");
8. **var** b = **new** Node("b");
9. **var** c = **new** Node("c");
10. **var** d = **new** Node("d");
11. **var** e = **new** Node("e");
12. **var** f = **new** Node("f");
13. **var** g = **new** Node("g");

16. a.left = c;
17. a.right = b;
19. c.left = f;
20. c.right = g;
22. b.left = d;
23. b.right = e;
24. // 前序遍历
25. **function** fun(root){
26. **if**(root == **null**){
27. **return**
28. };
29. console.log(root.value);   //先输出自己
30. fun(root.left)   // 输出左边
31. fun(root.right)   // 输出右边
32. }
33. fun(a)
34. // 中序遍历
35. **function** fun(root){
36. **if**(root == **null**){
37. **return**
38. };
39. fun(root.left)   // 输出左边
40. console.log(root.value);   //先输出自己
41. fun(root.right)   // 输出右边
42. }
43. fun(a)
44. // 后序遍历
45. **function** fun(root){
46. **if**(root == **null**){
47. **return**
48. };
49. fun(root.left)   // 输出左边
50. fun(root.right)   // 输出右边
51. console.log(root.value);   //先输出自己
52. }
53. fun(a)