

剑指32 从上到下打印二叉树3

用双端队列，每一行循环后，把队列反过来。

其实不用想这么复杂，直接在值传入vector的阶段控制顺序就行。

```
1 // 我的答案
2 // 层序遍历 + 双端队列 (奇偶层逻辑分离)
3
4 /**
5  * Definition for a binary tree node.
6  * struct TreeNode {
7  *     int val;
8  *     TreeNode *left;
9  *     TreeNode *right;
10  *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
11  * };
12  */
13 class Solution {
14 public:
15     vector<vector<int>> levelOrder(TreeNode* root) {
16         if (root == NULL) return {};
17         deque <TreeNode*> value;
18         vector<vector<int>> output;
19         TreeNode* q = root;
20         value.push_back(q);
21         int flip = 1;
22         while (!value.empty()) {
23             int linelen = value.size();
24             output.push_back({});
25             if (flip == 1) {
26                 for (int i = 0; i < linelen; i++) {
27                     q = value.back();
28                     output.back().push_back(q->val);
29                     value.pop_back();
30                     if (q->left)
31                         value.push_front(q->left);
32                     if (q->right)
33                         value.push_front(q->right);
34                 }
35             }
36             else {
37                 for (int i = 0; i < linelen; i++) {
38                     q = value.front();
39                     output.back().push_back(q->val);
40                     value.pop_front();
41                     if (q->right)
42                         value.push_back(q->right);
43                     if (q->left)
44                         value.push_back(q->left);
45                 }
46             }
47             flip = !flip;
48         }
49         return output;
50     }
51 }
```

```

46     }
47     flip = -flip;
48 }
49 return output;
50 }
51 };

```

其他方法：

用奇数层偶数层判断，`output.size()%2` 判断奇偶层。

不需要双端队列，只需要在list的插入过程中区分前插还是后插就行！（适用于python）

```

1 // 层序遍历 + 倒序(c++ reverse函数)
2 class Solution {
3 public:
4     vector<vector<int>> levelOrder(TreeNode* root) {
5         if (root == NULL) return {};
6         deque <TreeNode*> value;
7         vector<vector<int>> output;
8         TreeNode* q = root;
9         value.push_back(q);
10        int flip = -1;
11        while (!value.empty()) {
12            int linelen = value.size();
13            output.push_back({});
14            for (int i = 0; i < linelen; i++) {
15                q = value.front();
16                output.back().push_back(q->val);
17                value.pop_front();
18                if (q->left)
19                    value.push_back(q->left);
20                if (q->right)
21                    value.push_back(q->right);
22            }
23            if (flip == 1) {
24                reverse(output.back().begin(), output.back().end());
25            }
26            flip = -flip;
27        }
28        return output;
29    }
30 };
31
32 // 这个更好！！
33 // 用vector<int>(temp.rbegin(), temp.rend())取代了reverse
34 // vector的初始化的运用
35 vector<vector<int>> levelOrder(TreeNode* root) {
36     vector<vector<int>> ans;
37     if(root == NULL){
38         return ans;
39     }
40     queue<TreeNode*> q;
41     q.push(root);
42     bool isLeft = false;
43     while(!q.empty()){
44         int rowLen = q.size();

```

```

45     vector<int> temp;
46     for(int i = 0; i < rowLen; ++i){
47         TreeNode* curNode = q.front();
48         q.pop();
49         if(curNode != NULL){
50             temp.push_back(curNode->val);
51             if(curNode->left)q.push(curNode->left);
52             if(curNode->right)q.push(curNode->right);
53         }
54     }
55     isLeft = !isLeft;
56     if(!isLeft){
57         ans.push_back(vector<int>(temp.rbegin(), temp.rend()));
58     }else{
59         ans.push_back(temp);
60     }
61 }
62 return ans;
63 }

```

```

1 // 直接确定放置的位置
2 // 巧妙在于vector提前分配空间大小，然后通过下标访问从后往前放置数据。
3 class Solution {
4 public:
5     vector<vector<int>> levelOrder(TreeNode* root) {
6         if(root==nullptr) return {};
7         vector<vector<int>> ans;
8         queue<TreeNode*> q;
9         q.push(root);
10        bool rev = false;
11        while(!q.empty()){
12            int node_num = q.size();
13            vector<int> cur_level(node_num);
14            for(int i=0;i<node_num;++i){
15                auto cur = q.front(); q.pop();
16                if(cur->left!=nullptr) q.push(cur->left);
17                if(cur->right!=nullptr) q.push(cur->right);
18                cur_level[rev?node_num-1-i:i] = cur->val;
19            }
20            rev = !rev;
21            ans.push_back(cur_level);
22        }
23        return ans;
24    }
25 };
26

```