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。

```
// 我的答案
// 层序遍历 + 双端队列（奇偶层逻辑分离）

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        if (root == NULL) return {};
        deque <TreeNode*> value;
        vector<vector<int>> output;
        TreeNode* q = root;
        value.push_back(q);
        int flip = 1;
        while (!value.empty()) {
            int linelen = value.size();
            output.push_back({});
            if (flip == 1) {
                for (int i = 0; i < linelen; i++) {
                    q = value.back();
                    output.back().push_back(q->val);
                    value.pop_back();
                    if (q->left)
                        value.push_front(q->left);
                    if (q->right)
                        value.push_front(q->right);
                }
            }
            else {
                for (int i = 0; i < linelen; i++) {
                    q = value.front();
                    output.back().push_back(q->val);
                    value.pop_front();
                    if (q->right)
                        value.push_back(q->right);
                    if (q->left)
                        value.push_back(q->left);
                }
            }
            flip = -flip;
        }
        return output;
    }
};
```

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)

```
// 层序遍历 + 倒序(c++ reverse函数)
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        if (root == NULL) return {};
        deque <TreeNode*> value;
        vector<vector<int>> output;
        TreeNode* q = root;
        value.push_back(q);
        int flip = -1;
        while (!value.empty()) {
            int linelen = value.size();
            output.push_back({});
            for (int i = 0; i < linelen; i++) {
                q = value.front();
                output.back().push_back(q->val);
                value.pop_front();
                if (q->left)
                    value.push_back(q->left);
                if (q->right)
                    value.push_back(q->right);
            }
            if (flip == 1) {
                reverse(output.back().begin(), output.back().end());
            }
            flip = -flip;
        }
        return output;
    }
};

// 这个更好！！
// 用vector<int>(temp.rbegin(), temp.rend())取代了reverse
// vector的初始化的运用
vector<vector<int>> levelOrder(TreeNode* root) {
    vector<vector<int>> ans;
    if(root == NULL){
        return ans;
    }
    queue<TreeNode*> q;
    q.push(root);
    bool isLeft = false;
    while(!q.empty()){
        int rowLen = q.size();
        vector<int> temp;
        for(int i = 0; i < rowLen; ++i){
            TreeNode* curNode = q.front();
            q.pop();
            if(curNode != NULL){
                temp.push_back(curNode->val);
                if(curNode->left)q.push(curNode->left);
                if(curNode->right)q.push(curNode->right);
            }
        }
        isLeft = !isLeft;
        if(!isLeft){
            ans.push_back(vector<int>(temp.rbegin(), temp.rend()));
        }else{
            ans.push_back(temp);
        }
    }
    return ans;
}
```

```
// 直接确定放置的位置
// 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。
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        if(root==nullptr) return {};
        vector<vector<int>> ans;
        queue<TreeNode*> q;
        q.push(root);
        bool rev = false;
        while(!q.empty()){
            int node_num = q.size();
            vector<int> cur_level(node_num);
            for(int i=0;i<node_num;++i){
                auto cur = q.front(); q.pop();
                if(cur->left!=nullptr) q.push(cur->left);
                if(cur->right!=nullptr) q.push(cur->right);
                cur_level[rev?node_num-1-i:i] = cur->val;
            }
            rev = !rev;
            ans.push_back(cur_level);
        }
        return ans;
    }
};

```