22 括号生成

Label：回溯、动态规划

数字 n 代表生成括号的对数，请你设计一个函数，用于能够生成所有可能的并且 有效的 括号组合。

输入：n = 3
输出：["((()))","(()())","(())()","()(())","()()()"]

输入：n = 1
输出：["()"]

- 动态规划

```
class Solution {
    public List<String> generateParenthesis(int n) {
        List<List<String>> result = new ArrayList<List<String>>();
        // 初始化
        result.add(new ArrayList<String>(Arrays.asList("")));
        result.add(new ArrayList<String>(Arrays.asList("()")));

        for (int i = 2; i <= n; i++) {
            List<String> temp = new ArrayList<String>();  // 用来存储当前结果
            for (int j = 0; j < i; j++) {
                List<String> str1 = result.get(j);
                List<String> str2 = result.get(i - 1 - j);
                for (String s1 : str1) {    // s1 与 s2 组合起来的括号对数是 n-1 对
                    for (String s2 : str2) {
                        String el = "(" + s1 + ")" + s2;
                        temp.add(el);
                    }
                }
            }
            result.add(temp);
        }
        return result.get(n);
    }
}
```

- 深度优先

```
public class Solution {
    public List<String> generateParenthesis(int n) {
        List<String> res = new ArrayList<>();
        // 特判
        if (n == 0) {
            return res;
        }
        dfs("", 0, 0, n, res);
        return res;
    }
    /**
     * @param curStr 当前递归得到的结果
     * @param left   左括号已经用了几个
     * @param right  右括号已经用了几个
     * @param n      左括号、右括号一共得用几个
     * @param res    结果集
     */
    private void dfs(String curStr, int left, int right, int n, List<String>
res) {
        if (left == n && right == n) {
            res.add(curStr);  // 左右括号都用完了
            return;
        }
        // 剪枝  左括号用的比右括号的少，就立刻剪枝
        if (left < right) {
            return;
        }
        // 添加一个左括号
        if (left < n) {
            dfs(curStr + "(", left + 1, right, n, res);
        }
        // 添加一个右括号
        if (right < n) {
            dfs(curStr + ")", left, right + 1, n, res);
        }
    }
}
```

- 广度优先

```
public class Solution {

    class Node {
        private String res;  // 当前得到的字符串
        private int left;  // 剩余左括号数量
        private int right; // 剩余右括号数量

        public Node(String res, int left, int right) {
            this.res = res;
            this.left = left;
            this.right = right;
        }
    }

    public List<String> generateParenthesis(int n) {
        List<String> res = new ArrayList<>();
        if (n == 0) {
            return res;
        }
        Queue<Node> queue = new LinkedList<>();
        queue.offer(new Node("", n, n));
        // 总共需要拼凑的字符总数是 2 * n
        n = 2 * n;
        while (n > 0) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                Node curNode = queue.poll();
                if (curNode.left > 0) {
                    queue.offer(new Node(curNode.res + "(", curNode.left - 1,
curNode.right));
                }
                if (curNode.right > 0 && curNode.left < curNode.right) {  // 注意
这里有个剪枝
                    queue.offer(new Node(curNode.res + ")", curNode.left,
curNode.right - 1));
                }
            }
            n--;
        }

        // 最后一层就是题目要求的结果集
        while (!queue.isEmpty()) {
            res.add(queue.poll().res);
        }
        return res;
    }
}
```