1021. 删除最外层的括号

- 如果最外层的两个括号匹配上必然满足一个条件：**左括号'('和右括号')'的数量必然相等**
- 此时分隔最外层括号内字符串即可

```
class Solution {
    public String removeOuterParentheses(String S) {
       char[] arr = S.toCharArray();
        HashMap<Character, Character> map = new HashMap<>();
        map.put(')', '(');
        int left = 0, right = 0;
        int start = 0, end = 0;
        StringBuilder result = new StringBuilder();
        for(int i = 0; i < arr.length; i++){
            if(arr[i] == '(')
                left++;
            else if(arr[i] == ')')
                right++;
            if(left == right && arr[i] == ')'){
                end = i;
                String subString = S.substring(start + 1 , end);
//S[start+1]~S[end-1]
                result.append(subString);
                start = i + 1;
                left = 0;   //计数器重置
                right = 0;
            }
        }
        return result.toString();
    }

    public static String Solution(String S){
        StringBuilder sb = new StringBuilder();
        int level = 0;
        for (char c : S.toCharArray()) {
            if (c == ')')
                --level;
            if (level >= 1)  //第二个if和第三个if不能换位置
                sb.append(c);   //此时level不为0， 表示肯定不是最外层，因为最外层时，
'('和')'数量相同，level为0
            if (c == '(')
                ++level;
        }
        return sb.toString();
    }

    public static String Solution(String S){
        StringBuilder result = new StringBuilder();
        Stack<Character> stack = new Stack<>();
        int i = 0, j = 0;
        for(char c : S.toCharArray()){
```

```
            if(c == '(')
                stack.push('(');
            else
                stack.pop();
            if(stack.size() == 0){  //此时，最外层括号匹配上
                result.append(S.substring(j+1, i));
                j = i + 1;
            }
            i++;
        }
        return result.toString();
    }
}
```

剑指 Offer 09. 用两个栈实现队列

子烁

```
class CQueue {
    Stack<Integer> stack1, stack2;
    public CQueue() {
        stack1 = new Stack<>();
        stack2 = new Stack<>();
    }

    public void appendTail(int value) {
        stack1.push(value);
    }

    public int deleteHead() {
        if(stack2.isEmpty()){
            while(stack1.isEmpty()){
                stack2.push(stack1.pop());
            }
        }
        return stack2.pop();
    }
}

/**
 * Your CQueue object will be instantiated and called as such:
 * CQueue obj = new CQueue();
 * obj.appendTail(value);
 * int param_2 = obj.deleteHead();
 */
```

剑指 Offer 30. 包含min函数的栈

子烁

```
class MinStack {

    private Stack<Integer> stack;
    private Stack<Integer> miniStack;
    /** initialize your data structure here. */
    public MinStack() {
        stack = new Stack<>();
        miniStack = new Stack<>();
    }

    public void push(int x) {
        if(stack.isEmpty()){
            stack.push(x);
            miniStack.push(x);
            return;
        }
        int topVal = miniStack.peek();
        if(x > topVal)
            miniStack.push(topVal);
        else
            miniStack.push(x);
        stack.push(x);
    }

    public void pop() {
        stack.pop();
        miniStack.pop();
    }

    public int top() {
        return stack.peek();

    }

    public int min() {
        return miniStack.peek();

    }
}
```