栈：删除最外层的括号

题目来源：Leetcode 1021 https://leetcode-cn.com/problems/remove-outermost-parentheses/

暴力解法：拆解原语后删外层括号

Java代码

```
/**
 * 解法一：暴力解法思路分析
 * 1.定义容器存储原语子串
 *  new ArrayList<String>();
 * 2.定义左括号、右括号计数器：
 *  int left = 0, right = 0;
 * 3.遍历字符串，读取到括号时对应计数器自增
 * 4.检查是否到达原语结尾，截取原语子串并添加到容器中
 * 5.遍历容器，删除最外层括号后合并成新串
 *
 * 边界问题：
 *  遍历字符串，注意索引越界：i < S.length()
 *  截取原语字符串时，注意起止索引：[start, end)
 * 细节问题：
 *  需要记录上一次截取原语子串之后的位置
 *  删除原语子串的最外层括号，其实就是重新截取
 * @param S
 * @return
 */
public String removeOuterParentheses(String S) {
    int len = S.length();
    // 1.定义容器存储原语子串
    List<String> list = new ArrayList<>();
    // 2.定义左括号、右括号计数器
    int left = 0, right = 0, lastOpr = 0;
    // 3.遍历字符串，读取到括号时对应计数器自增
    for (int i = 0; i < len; i++) {
        char c = S.charAt(i);
        if (c == '(') {
            left++;
        } else if (c == ')') {
            right++;
        }
        // 4.检查是否到达某个原语结尾，截取原语子串添加到容器
        if (left == right) {
            list.add(S.substring(lastOpr, i + 1));
            lastOpr = i + 1;
        }
    }
    // 5.遍历容器中的原语子串，删除最外层后合并成新串
    StringBuilder sb = new StringBuilder();
    for (String s : list) {
        sb.append(s.substring(1, s.length() - 1));
    }
```

```
        return sb.toString();
    }
```

优化解法：直接定位原语内层子串

java代码

```
    /**
     * 解法二：优化解法思路分析
     * 1.定义容器存储删除外层括号后的原语子串
     *  new StringBuilder();
     * 2.定义左括号、右括号计数器：
     *  int left = 0, right = 0;
     * 3.遍历字符串，读取到括号时对应计数器自增
     * 4.检查是否到达原语结尾，截取不包含最外层的原语子串并拼接到容器中
     *
     * @param S
     * @return
     */
    public String removeOuterParentheses(String S) {
        int len = S.length();
        // 1.定义容器存储删除外层括号后的原语子串
        StringBuilder sb = new StringBuilder();
        // 2.定义左括号、右括号计数器
        int left = 0, right = 0, lastOpr = 0;
        // 3.遍历字符串，读取到括号时对应计数器自增
        for (int i = 0; i < len; i++) {
            char c = S.charAt(i);
            if (c == '(') {
                left++;
            } else if (c == ')') {
                right++;
            }
            // 4.检查是否到达某个原语结尾，截取不包含最外层的原语子串添加到容器
            if (left == right) {
                sb.append(S.substring(++lastOpr, i));
                lastOpr = i + 1;
            }
        }
        return sb.toString();
    }
```

最优解：栈解法

java代码

```
    /**
     * 最优解：栈解法
     * 1.使用数组模拟一个栈，临时存储左括号字符
     *  push(Character) ; pop(); isEmpty()
     * 2.遍历字符串，根据情况进行入栈/出栈操作
```

```
     *  读取到左括号，左括号入栈
     *  读取到右括号，左括号出栈
     * 3.判断栈是否为空，若为空，找到了一个完整的原语
     * 4.截取不含最外层括号的原语子串并进行拼接
     *
     * @param S
     * @return
     */
    public String removeOuterParentheses(String S) {
        StringBuilder result = new StringBuilder();
        // 1.使用数组模拟一个栈，临时存储字符，替代计数器
        Stack stack = new Stack();
        int lastOpr = 0;
        // 2.遍历字符串，根据情况进行入栈 / 出栈操作
        for (int i = 0; i < S.length(); i++) {
            char ch = S.charAt(i);
            if (ch == '(') { // 遇到左括号，左括号入栈
                stack.push(ch);
            } else if (ch == ')') { // 遇到右括号，左括号出栈
                stack.pop(); // 栈顶的左括号出栈
            }
            // 3.判断栈是否为空，若为空，找到了一个完整的原语
            if (stack.isEmpty()) {
                // 4.截取不含最外层括号的原语子串并进行拼接
                result.append(S.substring(lastOpr + 1, i));// 去掉原语的最外层括号并
追加到结果
                lastOpr = i + 1;// 往后找，再次初始化原语开始位置
            }
        }
        return result.toString();
    }
```

栈实现代码：

```
    /**
     * 栈：后进先出
     * @param <E>
     */
    class Stack<E> {
        Object[] elements = new Object[10000];
        int index = -1; // 栈顶索引
        int size = 0; // 栈中元素个数

        public Stack() {
        }

        /**
         * 往栈顶插入元素
         * @param c
         */
        public void push(E c) {
            elements[++index] = c;
            size++;
        }

        /**
         * 从栈顶获取数据，不移出
```

```
         * @return
         */
        public E peek() {
            if (index < 0) {
                return null;
            }
            return (E)elements[index];
        }

        /**
         * 从栈顶移出元素
         * @return
         */
        public E pop() {
            E e = peek();
            if (e != null) {
                elements[index] = null; // 移出动作
                index--; // 栈顶下移
                size--;
            }
            return e;
        }

        public boolean isEmpty() {
            return size == 0;
        }
    }
```

栈解法优化，使用栈的思想，直接用数组取代栈：

```
    /**
     * 最优解：代码优化：
     * 1.直接用数组取代栈
     *  创建数组、栈顶索引，使用数组操作入栈和出栈
     * 2.将原字符串转为数组进行遍历
     *  char[] s = S.toCharArray();
     * 3.去掉截取子串的操作，将原语字符直接拼接
     *  读取到左括号：此前有数据，当前必属原语
     *  读取到右括号：匹配后不为0，当前必属原语
     * @param S
     * @return
     */
    public String removeOuterParentheses(String S) {
        StringBuilder result = new StringBuilder();
        // 1.直接用数组取代栈
        int index = -1; // 栈顶索引
        int len = S.length();
        char[] stack = new char[len];

        char[] s = S.toCharArray();
        // 2.遍历字符串，根据情况进行入栈 / 出栈操作
        for (int i = 0; i < len; i++) {
            char ch = s[i];
            if (ch == '(') { // 遇到左括号，左括号入栈
                // 3.去掉截取子串的操作，将原语字符直接拼接
```

```
                if (index > -1) { // 此前有数据，当前必属原语
                    result.append(ch);
                }
                stack[++index] = ch;
            } else /*if (ch == ')')*/ { // 遇到右括号，左括号出栈
                stack[index--] = '\u0000'; // 栈顶的左括号出栈
                if (index > -1) {
                    result.append(ch);
                }
            }
        }
        return result.toString();
    }
```

C++代码

栈解法优化，使用栈的思想，直接用数组取代栈：

```
class Solution {
public:
    string removeOuterParentheses(string S) {
        // 1.直接用数组取代栈
        int index=-1; // 栈顶索引
        string res = "";
        char stack[S.size()];
        // 2.遍历字符串，根据情况进行入栈 / 出栈操作
        for (int i = 0; i < S.size(); i++) {
            char ch=S[i];
            if ( ch== '('){ // 遇到左括号，左括号入栈
                // 3.去掉截取子串的操作，将原语字符直接拼接
                if(index>-1) //此前有数据，当前必属原语
                    res+=ch;
                stack[++index]=ch;
            }else{  // 遇到右括号，左括号出栈
                index--;// 栈顶的左括号出栈
                if (index>-1)
                {
                    res+=ch;
                }

            }
        }
        return res;
    }
};
```

Python代码

栈解法优化，使用栈的思想，直接用数组取代栈：

```
class Solution:
    def removeOuterParentheses(self, S: str) -> str:
```

```
        """最优解：代码优化：
            1.直接用数组取代栈
             创建数组、栈顶索引，使用数组操作入栈和出栈
            2.将原字符串转为数组进行遍历
             char[] s = S.toCharArray();
            3.去掉截取子串的操作，将原语字符直接拼接
              读取到左括号：此前有数据，当前必属原语
              读取到右括号：匹配后不为0，当前必属原语"""
        result=[]
        # 1.直接用数组取代栈
        index=-1 # 栈顶索引
        stack = [None] * len(S)
        # 2.遍历原字符串的每个字符
        for ch in S:
            if ch=='(': # 遇到左括号，左括号入栈
                # 3.去掉截取子串的操作，将原语字符直接拼接
                if index>-1: #此前有数据，当前必属原语
                    result.append(ch)
                index+=1
                stack[index]=ch
            else:#遇到右括号，左括号出栈
                index-=1
                stack[index]=None #栈顶的左括号出栈
                if index>-1:
                    result.append(ch)
        return ''.join( result )
```

测试用例

辅助数据结构：栈。代码如下：

```
class Stack<E> {
    Object[] elements = new Object[10000];
    int index = -1; // 栈顶索引
    int size = 0; // 栈中元素个数

    public Stack() {
    }

    /**
     * 往栈顶插入元素
     * @param c
     */
    public void push(E c) {
        elements[++index] = c;
        size++;
    }

    /**
     * 从栈顶获取数据，不移出
     * @return
     */
    public E peek() {
        if (index < 0) {
            return null;
```

```
        }
        return (E)elements[index];
    }

    /**
     * 从栈顶移出元素
     * @return
     */
    public E pop() {
        E e = peek();
        if (e != null) {
            elements[index] = null; // 移出动作
            index--; // 栈顶下移
            size--;
        }
        return e;
    }

    public boolean isEmpty() {
        return size == 0;
    }
}
```

```
输入："()()"
输出：""
解释：原语化分解得到 "()" + "()"，  删除每个部分中的最外层括号后得到 "" + "" = ""
```

```
输入："((()))(())"
输出："()()()"
解释：原语化分解得到 "((()))" + "(())"，  删除每个部分中的最外层括号后得到 "()()" + "()" =
"()()()"
```

```
输入："((()))(())(()(())) "
输出："()()()()(()) "
解释：原语化分解得到 "((()))" + "(())" + "(()(()))"，删除每个部分中的最外层括号后得到 "()
()" + "()" + "()(())" = "()()()()(())"
```