

leetcode-master / problems / 0020.有效的括号.md

youngyangyang04 Update

History

7 contributors



267 lines (212 sloc) 8.89 KB

[PDF下载](#) [代码随想录](#) [刷题](#) [微信群](#) [B站](#) [代码随想录](#) [知识星球](#) [代码随想录](#)

欢迎大家[参与本项目](#)，贡献其他语言版本的代码，拥抱开源，让更多学习算法的小伙伴们收益！

数据结构与算法应用往往隐藏在我们看不到的地方

## 20. 有效的括号

<https://leetcode-cn.com/problems/valid-parentheses/>

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串，判断字符串是否有效。

有效字符串需满足：

- 左括号必须用相同类型的右括号闭合。
- 左括号必须以正确的顺序闭合。
- 注意空字符串可被认为是有效字符串。

示例 1:

- 输入: "()"
- 输出: true

示例 2:

- 输入: "{[]}"
- 输出: true

示例 3:

- 输入: "[]"
- 输出: false

示例 4:

- 输入: "[]"
- 输出: false

示例 5:

- 输入: "{}"
- 输出: true

## 思路

### 题外话

**括号匹配是使用栈解决的经典问题。**

题意其实就像我们在写代码的过程中，要求括号的顺序是一样的，有左括号，相应的位置必须要有右括号。

如果还记得编译原理的话，编译器在词法分析的过程中处理括号、花括号等这个符号的逻辑，也是使用了栈这种数据结构。

再举个例子，linux系统中，cd这个进入目录的命令我们应该再熟悉不过了。

```
cd a/b/c/../../
```

这个命令最后进入a目录，系统是如何知道进入了a目录呢，这就是栈的应用（其实可以出一道相应的面试题了）

所以栈在计算机领域中应用是非常广泛的。

有的同学经常会想学的这些数据结构有什么用，也开发不了什么软件，大多数同学说的软件应该都是可视化的软件例如APP、网站之类的，那都是非常上层的应用了，底层很多功能的实现都是基础的数据结构和算法。

**所以数据结构与算法的应用往往隐藏在我们看不到的地方！**

这里我就不过多展开了，先来看题。

### 进入正题

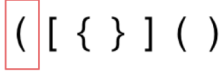
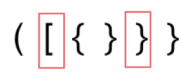
**由于栈结构的特殊性，非常适合做对称匹配类的题目。**

首先要弄清楚，字符串里的括号不匹配有几种情况。

一些同学，在面试中看到这种题目上来就开始写代码，然后就越写越乱。

建议要写代码之前要分析好有哪几种不匹配的情况，如果不动手之前分析好，写出的代码也会有很多问题。

先来分析一下 这里有三种不匹配的情况，

1. 第一种情况，字符串里左方向的括号多余了，所以不匹配。  括号匹配1
2. 第二种情况，括号没有多余，但是括号的类型没有匹配上。  括号匹配2
3. 第三种情况，字符串里右方向的括号多余了，所以不匹配。

( [ { } ] ) ) )

我们的代码只要覆盖了这三种不匹配的情况，就不会出问题，可以看出 动手之前分析好题目的重要性。

动画如下：

## 20.有效括号

第一种情况：已经遍历完了字符串，但是栈不为空，说明有相应的左括号没有右括号来匹配，所以return false

第二种情况：遍历字符串匹配的过程中，发现栈里没有要匹配的字符。所以return false

第三种情况：遍历字符串匹配的过程中，栈已经为空了，没有匹配的字符了，说明右括号没有找到对应的左括号return false

那么什么时候说明左括号和右括号全都匹配了呢，就是字符串遍历完之后，栈是空的，就说明全都匹配了。

分析完之后，代码其实就比较好写了，

但还有一些技巧，在匹配左括号的时候，右括号先入栈，就只需要比较当前元素和栈顶相不相等就可以了，比左括号先入栈代码实现要简单的多了！

实现C++代码如下：

```
class Solution {
public:
    bool isValid(string s) {
        stack<int> st;
        for (int i = 0; i < s.size(); i++) {
            if (s[i] == '(') st.push(')');
            else if (s[i] == '{') st.push('}');
            else if (s[i] == '[') st.push(']');
            // 第三种情况：遍历字符串匹配的过程中，栈已经为空了，没有匹配的字符了，说明右括号没有找到
            // 第二种情况：遍历字符串匹配的过程中，发现栈里没有我们要匹配的字符。所以return false
            else if (st.empty() || st.top() != s[i]) return false;
            else st.pop(); // st.top() 与 s[i]相等，栈弹出元素
        }
        return st.empty();
    }
};
```

```

    }
    // 第一种情况：此时我们已经遍历完了字符串，但是栈不为空，说明有相应的左括号没有右括号来匹
    return st.empty();
}
};

```

技巧性的东西没有固定的学习方法，还是要多看多练，自己总灵活运用了。

## 其他语言版本

Java:

```

class Solution {
    public boolean isValid(String s) {
        Deque<Character> deque = new LinkedList<>();
        char ch;
        for (int i = 0; i < s.length(); i++) {
            ch = s.charAt(i);
            //碰到左括号，就把相应的右括号入栈
            if (ch == '(') {
                deque.push(')');
            } else if (ch == '{') {
                deque.push('}');
            } else if (ch == '[') {
                deque.push(']');
            } else if (deque.isEmpty() || deque.peek() != ch) {
                return false;
            } else { //如果是右括号判断是否和栈顶元素匹配
                deque.pop();
            }
        }
        //最后判断栈中元素是否匹配
        return deque.isEmpty();
    }
}

```

Python:

```

class Solution:
    def isValid(self, s: str) -> bool:
        stack = [] # 保存还未匹配的左括号
        mapping = {")": "(", "]": "[", "}": "{"}
        for i in s:
            if i in "([{": # 当前是左括号，则入栈
                stack.append(i)
            elif stack and stack[-1] == mapping[i]: # 当前是配对的右括号则出栈
                stack.pop()
            else: # 不是匹配的右括号或者没有左括号与之匹配，则返回false
                return False
        return stack == [] # 最后必须正好把左括号匹配完

```

Go:

```

func isValid(s string) bool {
    hash := map[byte]byte{'(':')', '[':']', '{':'}'}
    stack := make([]byte, 0)
    if s == "" {
        return true
    }

    for i := 0; i < len(s); i++ {
        if s[i] == '(' || s[i] == '[' || s[i] == '{' {
            stack = append(stack, s[i])
        } else if len(stack) > 0 && stack[len(stack)-1] == hash[s[i]] {
            stack = stack[:len(stack)-1]
        } else {
            return false
        }
    }
    return len(stack) == 0
}

```

Ruby:

```

def is_valid(strs)
    symbol_map = {'(' => ')', '[' => ']', '{' => '}' }
    stack = []
    strs.size.times {|i|
        c = strs[i]
        if symbol_map.has_key?(c)
            top_e = stack.shift
            return false if symbol_map[c] != top_e
        else
            stack.unshift(c)
        end
    }
    stack.empty?
end

```

Javascript:

```

var isValid = function (s) {
    const stack = [];
    for (let i = 0; i < s.length; i++) {
        let c = s[i];
        switch (c) {
            case '(':
                stack.push(')');
                break;
            case '[':
                stack.push(']');
                break;
            case '{':
                stack.push('}');
                break;
            default:
                if (c !== stack.pop()) {

```

```
        return false;
    }
}
}
return stack.length === 0;
};
// 简化版本
var isValid = function(s) {
    const stack = [],
        map = {
            "(": ")",
            "{": "}",
            "[": "]"
        };
    for(const x of s) {
        if(x in map) {
            stack.push(x);
            continue;
        }
        if(map[stack.pop()] !== x) return false;
    }
    return !stack.length;
};
```

- 
- 作者微信: [程序员Carl](#)
  - B站视频: [代码随想录](#)
  - 知识星球: [代码随想录](#)