【前端面试】如何判断字符串括号是否匹配?

前言

这是一道考察基础数据结构与算法的题目，如果没有学过数据结构，可能刚开始还有点摸不着头脑，但是如果你学过数据结构，那么看到这道题我相信你就有很清晰的思路。今天我们就来剖析剖析这道题。

1.实现目标

这道题目其实背景与我们的实际开发场景还有比较大的关系，比如有这样一串字符串：

```
dsa(dsadsa{dhk)s})}
```

我们想要知道上面的括号是否一一匹配，就好比我们编写代码时候，括号是一一对应的。我们在举几个例子，大家就知道什么叫做括号匹配了。

括号匹配的字符串：

```
ads[dsad{dsad(dsads)dsadsa}dsad]
```

括号不匹配的字符串：

```
asda(ds[dshd]ds(dsad])
```

看了上面的例子应该就明白什么叫做括号匹配的字符串了，我们总结一下题目要求。

需求：

> 假如我们有一个字符串：esae（dsad[dsa}）dsa。我们需要判断这个字符串中的括号是否匹配的上。

输入：

> dsad{ds(dsads)a}

输出：

> true

输入：

> asda(ds[dshd]ds(dsad])

输出：

> false

2.实现思路

很多没有接触过数据结构或者不熟悉的小伙伴来说，他们的思路可能非常的硬核，比如他们可能会有**如下思路：**

> 直接使用循环，保存每种括号的数量，然后比较正反括号的数量是否相等，相等则代表括号匹配。

上面的思路在某一些场景下是可以返回正确结果的，但是很多场景下是无法正确判断的，比如下面这种场景：

```
{a[b(c])}
```

上面这个字符串的正反括号数量上是匹配的，但是它是一个括号匹配的字符串吗？很明显不是，它的(对应的是]，很明显是错误的。

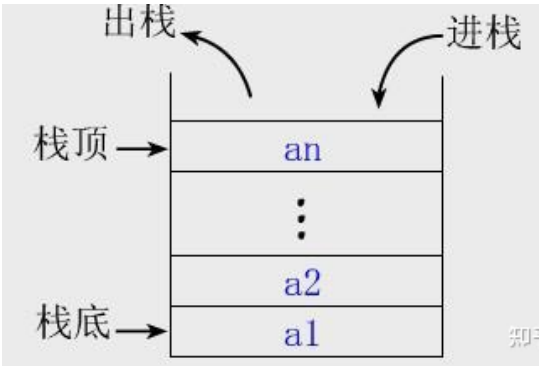
正确思路：

我们可以使用栈这种数据结构来实现这个算法题，栈不是一个实实在在的东西，它也只是一个逻辑模型，一种概念而已，就好比我们的数学公式，它只是一种理念，我们需要在实际的题目运用它。

简单介绍下栈的概念：

> 栈是一种线性存储结构，它有着”先进先出“的特点。

通过图我们能够更好地理解，如下图所示，每个栈都有栈顶和栈底，大家可以把栈想象成一个放餐盘的桶，我们最后放入的餐盘是不是会被最先拿出来，转换过来就是最后放入栈内的元素会有限出栈，也就是先进先出原则。



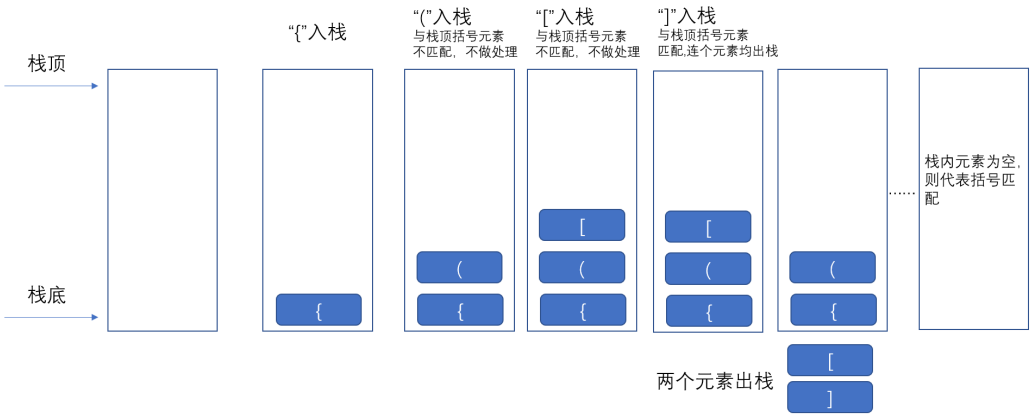
栈结合题目：

那么我们如何将我们的字符串匹配与栈结合呢?

我们可以把字符串中的所有括号拿出来，然后依次入栈，如果括号匹配了，那么两个元素都出栈，当括号入栈和出栈完毕，且栈内没有元素后，我们就说这个字符串是括号匹配的。

我们可以通过一张图来理解，**如下图所示：**

字符串：{c([a]b)}
提取括号：{([])}



通过上图应该就很好理解判断字符串括号匹配的原理了，主要就是利用了栈的先进先出操作，以及栈顶元素与入栈元素的额比较。

3.具体实现

我们知道了题目做什么并且有了解题思路，那么我们就可以具体去实现它了。

这里需要注意：**我们采用的是 JS 实现，所以直接采用数组来当作栈，当然，栈和数组是没有任何关系的，我们只是借助数组来模拟栈的理念罢了。**

代码如下：

```
<script>
  function bracketMatch(str) {
    const length = str.length;
    if (length === 0) {
      return true
    }
    const stack = []; // 借助数组模拟栈
    const leftBracket = '([{'; // 定义左括号
    const rightBracket = ')]}'; // 定义右括号
    for (let index = 0; index < length; index++) {
      const s = str[index];
      if (leftBracket.includes(s)) {
        // 如果出现左括号，压栈
        stack.push(s)
      } else if (rightBracket.includes(s)) {
        // 如果出现右括号，需要判断栈顶元素与之是否匹配，是否需要出栈
        const top = stack[stack.length - 1]; // 栈顶元素
        // 左右括号是否匹配
        if (isMatch(top, s)) {
          stack.pop(); // 出栈，注意这儿没有压栈操作
        }
      }
    }
```

```
    return stack.length === 0; // 长度为 0 代表括号匹配
  }
  // 判断左右括号是否匹配
  function isMatch(left, right) {
    if (left === '{' && right === '}') {
      return true;
    } else if (left === '[' && right === ']') {
      return true;
    } else if (left === '(' && right === ')') {
      return true;
    } else {
      return false
    }
  }

  // 测试
  console.log(bracketMatch('a{dsa}(sas)[dsa]')); // true
  console.log(bracketMatch('a{dsa}(sas[)dsa]')); // false
</script>
```

上段代码中我们直接利用了数组来模拟战，使用 push 和 pop 操作来模拟出栈和压栈的操作。

思路其实挺简单的，就是一个入栈和压栈操作，遇到左括号直接压栈，遇到右括号则与栈顶元素匹配一下，匹配上了直接出栈，否则返回 false。

总结

这道算法题考察的是对栈的理解与应用，如果没有学过数据结构与算法，或者没有做过算法题，其实这道题相对来说还是比较难的，但是一旦知道了原理，就非常简单。就好比我们看魔术，不知道魔术原理之前感觉很神奇，知道魔术原理之后瞬间感觉也就那样。