判断一个字符串是否括号匹配

1. 栈

栈是一种先进后出的数据结构，队列是一种先进先出的数据结构，队列可以想像成在食堂排队打饭，先去的人先打饭，打完就走，后去的人排队等前面的人做完了才去。栈和队列相反。

先入栈，后出栈。

2. 数组和栈的关系

数组和栈没有任何关系。

数组是一个物理结构，是真实存在的，和编程语言有关，不同编程语言数组用法也不同。 栈是一个逻辑结构，是不存在的，是一种思维，和编程语言无关，每一种编程语言都是上面的概念。

3. 判断一个字符串是否括号匹配

有一个字符串 `"{a[b(c)d]e}f"` ，括号完全匹配。

下面的字符串括号就不匹配：

- `{a[bc}d` 缺少 `]`
- `{a[b}c]d` 中 `[` 无法匹配到

解体思路：

- 运用栈的思维，遇到左括号，就入栈
- 遇到右括号，判断栈里面最后一个元素，如果和该右括号不匹配，则表示括号不匹配；如果和该右括号匹配，则最后一个元素出栈，当遍历完字符串时，左括号全部出栈，则表示括号完全匹配。其他情况不匹配

```
// 判断是否为同一类型括号
function isSameType(left, right) {
    if (left === '[' && right === ']'
        || left === '{' && right === '}'
        || left === '(' && right === ')') {
        return true;
    }
    return false;
}
```

```
// 判断括号是否匹配
function match(str) {
    const len = str.length;
    if (!len) return true;
```

```
    let arr = [];

    let leftChars = '[{(';
    let rightChars = ']})';

    // 有循环，时间复杂度是 O(n)，整体时间复杂度是 O(n)
    // 空间复杂度是 O(n)
    for (let i = 0; i < len; i++) {
        const char = str[i];
        // includes 判断是否包含，本身时间复杂度是 O(n)，可是这里只有三种取值，寻找 1
次或 2 次或 3 次，故这里时间复杂度算为 O(1)
        if (leftChars.includes(char)) {
            arr.push(char);
        } else if (rightChars.includes(char)) {
            if (isSameType(arr[arr.length - 1], char)) {
                arr.pop();
            } else {
                return false;
            }
        }
    }

    return !arr.length;
}
```