```
import Foundation
/// **剑指 Offer 19. 正则表达式匹配**
/// 请实现一个函数用来匹配包含`.`和`*`的正则表达式。
/// 模式中的字符`。`表示任意一个字符,而`*`表示它前面的字符可以出现任意次(含0次)。
/// 在本题中, 匹配是指字符串的所有字符匹配整个模式。
/// 例如,字符串`"aaa"`与模式`"a.a"`和`"ab*ac*a"`匹配,但与`"aa.a"`和`"ab*a"`均不匹配。
class Solution {
   func isMatch(_ s: String, _ p: String) -> Bool {
       /// 获取字符串 s 的长度, 并加 1
       let m = s.count + 1
       /// 获取字符串 p 的长度, 并加 1
       let n = p.count + 1
       /// 创建二维数组 dp, 用于动态规划
       var dp = [[Bool]](repeating: [Bool](repeating: false, count: n),
        count: m)
       /// 初始化 dp[0][0]为 true,表示空字符串匹配空正则表达式
       dp[0][0] = true
       /// 处理 p 的特殊情况,如果出现` '*'`,则 dp[0][j]的值与 dp[0][j-2]相关
       for j in stride(from: 2, to: n, by: 2) {
           dp[0][j] = (dp[0][j - 2] &&
                      p[p.index(p.startIndex, offsetBy: j - 1)] == "*")
       }
       /// 开始动态规划,填充 dp 数组
       for i in 1..<m {
           for j in 1..<n {
               if p[p.index(p.startIndex, offsetBy: j - 1)] == "*" {
                  // 如果出现 '*'、则 dp[i][i] 的值与 dp[i][i - 2] 和 dp[i -
                   1][j] 相关
                  dp[i][j] = (dp[i][j - 2] | |
                              (dp[i - 1][j] \&\&
                               (s[s.index(s.startIndex, offsetBy: i - 1)] ==
                                p[p.index(p.startIndex, offsetBy: j - 2)] ||
                                p[p.index(p.startIndex, offsetBy: j - 2)] ==
                                 ".")))
               } else {
                  /// 否则, dp[i][j] 的值与 dp[i-1][j-1] 相关
                  dp[i][j] = (dp[i - 1][j - 1] &&
                              (p[p.index(p.startIndex, offsetBy: j - 1)] ==
                               s[s.index(s.startIndex, offsetBy: i - 1)] ==
                                p[p.index(p.startIndex, offsetBy: j - 1)]))
               }
           }
       }
       /// 返回结果,即 s 是否匹配 p
       return dp[m-1][n-1]
   }
}
```