1. 题目
2. 题干

请从字符串中找出一个最长的不包含重复字符的子字符串，计算该最长子字符串的长度。

1. 示例

示例 1:

输入: "abcabcbb"

输出: 3

解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。

示例 2:

输入: "bbbbb"

输出: 1

解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。

示例 3:

输入: "pwwkew"

输出: 3

解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。

  请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。

1. 题解
2. 思路

首先，从题干来看，题目要找的是最长的无重复子串长度。这种找子串的题目第一反应就是动态规划，遍历字符串，根据前面的结果递推出后面的结果。因此，我们可以设置一个dp数组，dp[j]代表着以s[j]为结尾最长不重复子串的长度。接下来，就需要来设置动态规划的转移方程了，对于dp[j]而言，已知dp[j-1]，我们要求以s[j]为结尾的最长无重复子串的长度，因此如果在dp[j-1]长度的子串中没有与s[j]相同的字符，那么dp[j] = dp[j-1]+1；如果在dp[j-1]中出现了与s[j]相同的字符，位置为w，则dp[j] = j-w; 通过这个动态转移方法，可以遍历字符串，求出最长的无重复子串长度。

出现的字符位置用一个map来存储，因为每个字符只需要找到离它最近的那个相同字符即可，因此map中每个字符仅保存一个位置是合理的，及时更新最新的位置。同时，为了优化空间，由于仅仅使用了dp[j-1],dp[j],以及最大值变量，因此我们只需要不断地更新这三个变量即可，不需要创建整个dp数组。

1. 代码实现

Java :

class Solution {

    public int lengthOfLongestSubstring(String s) {

        if(s == null || s.length() == 0){

            return 0;

        }

        HashMap<Character, Integer> map = new HashMap<>();

        int n = s.length();

        int tmp = 0, ans = 0;

        for(int i = 0; i < n; ++i){

            // 取当前字符相同的最近位置

            int left = map.getOrDefault(s.charAt(i), -1);

            //更新当前字符位置

            map.put(s.charAt(i), i);

            // 动态方程更新， tmp 代表着dp[i-1]

            tmp = tmp < i - left ? tmp +1 : i - left;

            // ans 代表着 最大的长度

            ans = Math.max(ans, tmp);

        }

        return ans;

    }

}