Given a string, your task is to count how many palindromic substrings in this string. The substrings with different start indexes or end indexes are counted as different substrings even they consist of same characters.

**Input:** "abc"

**Output:** 3

**Explanation:** Three palindromic strings: "a", "b", "c".

|  |
| --- |
| 错误想法1  用DP  一个队列，存储全部回文，这些回文的最后一个元素一定是第i-1个char  然后对所有的回文判断1）回文+char[i] 是否是回文 2） 回文前1个+回文+char[i]是否是回文  重复上述步骤，直到结束。  尝试优化了很多步骤，但是结果还是不好，因为本质是重复判断过多 |
| 问题点：  1 队列里有n个回文，第s个回文的1）场景，可能是第l个回文的2）场景，因此发生重复计算。  解决方式是记录当前回文的起始点，如果有重复就不计数。但是这样的代价非常大。重复计算非常多。  在计算一个长回文+char[i]的时候，其实会重复计算里面的一个短回文+char[i]。  时间复杂度非常差。 |
| 反思点：  一开始dp里要存储的内容，就不对，导致后面复杂计算。  所以再遇到DP题，要考虑的是，dp里存储的是否是最简单最有效的内容。  DP的核心是减少重复计算。上述思想其实根本就不算dp了。 |
| 改进点1：  dp[i][j] 存储的是第i到第j个子字符串，是否是回文（1-是回文，0-不是回文），如果chars[i-1] == chars[j+1]同时dp[i][j]==1，说明i~j是回文且i~j头尾各外扩一个的字符相等，那i-1~j+1也是回文。  但是，这样的时间复杂度 还是n2  还有优化空间！ |
| 改进点2：  简单粗暴但是快速的方法  从0~len-1，1）检测以第i个char为中心向左右扩张，有多少个回文 2）检测，以第i和i+1个char为中心，向左右扩张，有多少个回文 |

花了很长时间，再一次陷在原来的思维里难以自拔。

每次都在说，

如果重新想一个解法，沉没成本太大，所以就很容易陷入当前的绝境。

如果解不出来就慌张 ，脑子就更加乱套了。

但是这都不是借口。

1. 想好算法后，先预估时间复杂度，不着急写代码，及时止损
2. 直播解题也是压力测试。今日个人感觉有进步。