[滑动窗口真滴简单!] 闪电五连鞭带你秒杀12道中档题 (附详情解析) - 无重复字符的最长子串

leetcode-cn.com/problems/longest-substring-without-repeating-characters/solution/yi-ge-mo-ban-miao-sha-10dao-zhong-deng-n-sb0x

最后更新时间: 2021/09/19

不像动态规划，绝大部分滑动窗口类题目本质上真的不算是难题，经过有效的训练就可以熟练掌握。本文中Eason给大家分享一套滑动窗口的思维框架 (共五步-五连鞭)，非常好记和容易理解。掌握它之后，你可以一口气秒杀**12道中等难度**的同类型题目 (卧槽？12道？是的，而且给全解析，再不点赞还是人？)，从而帮助你再遇见滑动窗口类型题目的时候不再胆怯！

PS：在这里我就不教大家什么是滑动窗口啦，这个概念并不难，leetcode上类似的科普文也有很多，所以我就不班门弄斧了。如果读者完全没有听说过这个概念，烦请先花10分钟弄懂个大概后再来阅读本文

废话不多说，直接上框架 (伪代码)

```
class Solution:
    def problemName(self, s: str) -> int:
        # Step 1: 定义需要维护的变量们 (对于滑动窗口类题目，这些变量通常是最小长度，最大长度，或者哈希表)
        x, y = ..., ...

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(s)):
            # Step 3: 更新需要维护的变量, 有的变量需要一个if语句来维护 (比如最大最小长度)
            x = new_x
            if condition:
                y = new_y

            '''
            ------------- 下面是两种情况，读者请根据题意二选1 -------------
            '''
            # Step 4 - 情况1
            # 如果题目的窗口长度固定：用一个if语句判断一下当前窗口长度是否达到了限定长度
            # 如果达到了，窗口左指针前移一个单位，从而保证下一次右指针右移时，窗口长度保持不变,
            # 左指针移动之前, 先更新Step 1定义的(部分或所有)维护变量
            if 窗口长度达到了限定长度:
                # 更新 (部分或所有) 维护变量
                # 窗口左指针前移一个单位保证下一次右指针右移时窗口长度保持不变

            # Step 4 - 情况2
            # 如果题目的窗口长度可变: 这个时候一般涉及到窗口是否合法的问题
            # 如果当前窗口不合法时, 用一个while去不断移动窗口左指针, 从而剔除非法元素直到窗口再次合法
            # 在左指针移动之前更新Step 1定义的(部分或所有)维护变量
            while 不合法:
                # 更新 (部分或所有) 维护变量
                # 不断移动窗口左指针直到窗口再次合法

        # Step 5: 返回答案
        return ...
```

看不懂？ mode问题，在我们做 `3. 无重复字符的最长子串` 这道题前先用这个模板先套一道简单题 `643. 子数组最大平均数 I` (不计算到中等题目中，基本的良心还是得有)
如果套完还不懂，不要担心，让我们再套一道，套完仍然不懂，let's 再套一道.....
我们有共13 (12中1简) 道题，请相信自己，套着套着，咱们终究会弄懂的！

643. 子数组最大平均数 I

```
class Solution:
    def findMaxAverage(self, nums: List[int], k: int) -> float:
        # Step 1
        # 定义需要维护的变量
        # 本题求最大平均值 (其实就是求最大和)，所以需要定义sum_, 同时定义一个max_avg (初始值为负
无穷)
        sum_, max_avg = 0, -math.inf

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(nums)):
            # Step 3: 更新需要维护的变量 (sum_, max_avg), 不断把当前值积累到sum_上
            sum_ += nums[end]
            if end - start + 1 == k:
                max_avg = max(max_avg, sum_ / k)

            # Step 4
            # 根据题意可知窗口长度固定，所以用if
            # 窗口首指针前移一个单位保证窗口长度固定, 同时提前更新需要维护的变量 (sum_)
            if end >= k - 1:
                sum_ -= nums[start]
                start += 1
        # Step 5: 返回答案
        return max_avg
```

现在再来让我们看看 `3. 无重复字符的最长子串` 这道题，你会发现其实写法也是差不多的，就是多了个哈希表然后if变成了while

3. 无重复字符的最长子串

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        # Step 1: 定义需要维护的变量, 本题求最大长度，所以需要定义max_len, 该题又涉及去重，因此还需要一个哈希表
        max_len, hashmap = 0, {}

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(s)):
            # Step 3
            # 更新需要维护的变量 (max_len, hashmap)
            # i.e. 把窗口末端元素加入哈希表，使其频率加1，并且更新最大长度
            hashmap[s[end]] = hashmap.get(s[end], 0) + 1
            if len(hashmap) == end - start + 1:
                max_len = max(max_len, end - start + 1)

            # Step 4:
            # 根据题意，  题目的窗口长度可变: 这个时候一般涉及到窗口是否合法的问题
            # 这时要用一个while去不断移动窗口左指针, 从而剔除非法元素直到窗口再次合法
            # 当窗口长度大于哈希表长度时候 (说明存在重复元素)，窗口不合法
            # 所以需要不断移动窗口左指针直到窗口再次合法, 同时提前更新需要维护的变量 (hashmap)
            while end - start + 1 > len(hashmap):
                head = s[start]
                hashmap[head] -= 1
                if hashmap[head] == 0:
                    del hashmap[head]
                start += 1
        # Step 5: 返回答案 (最大长度)
        return max_len
```

没懂？没事，让我们再来做个类似的 `159. 至多包含两个不同字符的最长子串` ，这道题和 `3. 无重复字符的最长子串` 几乎一模一样

159. 至多包含两个不同字符的最长子串

```
class Solution:
    def lengthOfLongestSubstringTwoDistinct(self, s: str) -> int:
        # Step 1:
        # 定义需要维护的变量, 本题求最大长度，所以需要定义max_len,
        # 该题又涉及计算不重复元素个数，因此还需要一个哈希表
        max_len, hashmap = 0, {}

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(s)):
            # Step 3
            # 更新需要维护的变量 (max_len, hashmap)
            # 首先，把当前元素的计数加一
            # 一旦哈希表长度小于等于2(之多包含2个不同元素)，尝试更新最大长度
            tail = s[end]
            hashmap[tail] = hashmap.get(tail, 0) + 1
            if len(hashmap) <= 2:
                max_len = max(max_len, end - start + 1)

            # Step 4:
            # 根据题意,  题目的窗口长度可变: 这个时候一般涉及到窗口是否合法的问题
            # 这时要用一个while去不断移动窗口左指针, 从而剔除非法元素直到窗口再次合法
            # 哈希表长度大于2的时候 (说明存在至少3个重复元素)，窗口不合法
            # 所以需要不断移动窗口左指针直到窗口再次合法, 同时提前更新需要维护的变量 (hashmap)
            while len(hashmap) > 2:
                head = s[start]
                hashmap[head] -= 1
                if hashmap[head] == 0:
                    del hashmap[head]
                start += 1
        # Step 5: 返回答案 (最大长度)
        return max_len
```

还是没有什么感觉? 没问题，再来一道

209. 长度最小的子数组

```
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        # Step 1: 定义需要维护的变量, 本题求最小长度，所以需要定义min_len, 本题又涉及求和，因此还需要一个sum变量
        min_len, sum_ = math.inf, 0

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(nums)):
            # Step 3: 更新需要维护的变量 (min_len, sum_)
            sum_ += nums[end]

            # 这一段可以删除，因为下面的while已经handle了这一块儿逻辑，不过写在这也没影响
            if sum_ >= target:
                min_len = min(min_len, end - start + 1)

            # Step 4
            # 这一题这里稍微有一点特别: sum_ >= target其实是合法的，但由于我们要求的是最小长度，
            # 所以当sum_已经大于target的时候继续移动右指针没有意义，因此还是需要移动左指针慢慢逼近答案
            # 由于左指针的移动可能影响min_len和sum_的值，因此需要在移动前将它们更新
            while sum_ >= target:
                min_len = min(min_len, end - start + 1)
                sum_ -= nums[start]
                start += 1
        # Step 5：返回答案 (最小长度)
        if min_len == math.inf:
            return 0
        return min_len
```

没有看懂? 没问题，没问题, 再来一道

1695. 删除子数组的最大得分

```
class Solution:
    def maximumUniqueSubarray(self, nums: List[int]) -> int:
        # Step 1
        # 定义需要维护的变量, 本题最大得分，所以需要定义当前得分sum_和最大得分max_sum
        # 本题又涉及去重 (题目规定子数组不能有重复)，因此还需要一个哈希表
        sum_, max_sum, hashmap = 0, 0, {}

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(nums)):
            # Step 3
            # 更新需要维护的变量 (sum_, hashmap)
            # sum和hashmap需要更新就不说了，max_sum当且仅当哈希表里面没有重复元素时 (end -
start + 1 == len(hashmap)) 更新
            tail = nums[end]
            sum_ += tail
            hashmap[tail] = hashmap.get(tail, 0) + 1
            if end - start + 1 == len(hashmap):
                max_sum = max(max_sum, sum_)

            # Step 4
            # 根据题意,  题目的窗口长度可变: 这个时候一般涉及到窗口是否合法的问题
            # 这时要用一个while去不断移动窗口左指针, 从而剔除非法元素直到窗口再次合法
            # 哈希表里面有重复元素时 (end - start + 1 > len(hashmap)) 窗口不合法
            # 所以需要不断移动窗口左指针直到窗口再次合法, 同时提前更新需要维护的变量 (hashmap，
sum_)
            while end - start + 1 > len(hashmap):
                head = nums[start]
                hashmap[head] -= 1
                if hashmap[head] == 0:
                    del hashmap[head]
                sum_ -= nums[start]
                start += 1
        # Step 5: 返回答案
        return max_sum
```

依旧没完全懂? 没问题，没问题, 没问题, 再来一道!

438. 找到字符串中所有字母异位词

```
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        # Step 1:
        # 定义需要维护的变量
        # 本文需要对比两组字符串是否为异位词，所以用哈希表 (abc和bac是异位词是因为他们对应的哈希表相等)
        # 同时我们需要找到所有合法解，所以还需要一个res数组
        res, hashmap = [], {}

        # Step 1.1： 同时把p的哈希表也建立了 (这个哈希表不需要维护，为定值)
        hashmap_p = {}
        for char in p:
            hashmap_p[char] = hashmap_p.get(char, 0) + 1

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(s)):
            # Step 3: 更新需要维护的变量 (hashmap)， 如果hashmap == hashmap_p，代表找到了一个解，加入到res
            hashmap[s[end]] = hashmap.get(s[end], 0) + 1
            if hashmap == hashmap_p:
                res.append(start)

            # Step 4
            # 根据题意可知窗口长度固定，所以用if
            # 窗口左指针前移一个单位保证窗口长度固定, 同时提前更新需要维护的变量 (hashmap)
            if end >= len(p) - 1:
                hashmap[s[start]] -= 1
                if hashmap[s[start]] == 0:
                    del hashmap[s[start]]
                start += 1
        # Step 5: 返回答案
        return res
```

再来一题！

567. 字符串的排列

```
class Solution:
    def checkInclusion(self, s1: str, s2: str) -> bool:
        # Step 1
        # 定义需要维护的变量
        # 因为和排列相关 (元素相同，顺序可以不同)，使用哈希表
        hashmap2 = {}

        # Step 1.1: 同时建立s1的哈希表 (这个哈希表不需要维护，为定值)
        hashmap1 = {}
        for char in s1:
            hashmap1[char] = hashmap1.get(char, 0) + 1

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(s2)):
            # Step 3: 更新需要维护的变量 (hashmap2)， 如果hashmap1 == hashmap2，代表s2包含
s1的排列，直接return
            tail = s2[end]
            hashmap2[tail] = hashmap2.get(tail, 0) + 1
            if hashmap1 == hashmap2:
                    return True

            # Step 4:
            # 根据题意可知窗口长度固定，所以用if
            # 窗口左指针前移一个单位保证窗口长度固定, 同时提前更新需要维护的变量 (hashmap2)
            if end >= len(s1) - 1:
                head = s2[start]
                hashmap2[head] -= 1
                if hashmap2[head] == 0:
                    del hashmap2[head]
                start += 1
        # Step 5： 没有在s2中找到s1的排列，返回False
        return False
```

再再来一题 (声音开始颤抖)！

487. 最大连续1的个数 II

```
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        # Step 1
        # 定义需要维护的变量
        # 因为是求最大长度，所以有max_len，又同时涉及计数 (0的个数不能超过1个)，所以还要一个哈希
表
        max_len, hashmap = 0, {}

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(nums)):
            # Step 3: 更新需要维护的变量 (hashmap， max_len)
            tail = nums[end]
            hashmap[tail] = hashmap.get(tail, 0) + 1
            if hashmap.get(0, 0) <= 1:
                max_len = max(max_len, end - start + 1)

            # Step 4
            # 根据题意， 题目的窗口长度可变: 这个时候一般涉及到窗口是否合法的问题
            # 这时要用一个while去不断移动窗口左指针, 从而剔除非法元素直到窗口再次合法
            # 当hashmap里面0的个数大于1的时候，窗口不合法
            # 所以需要不断移动窗口左指针直到窗口再次合法, 同时提前更新需要维护的变量 (hashmap)
            while hashmap.get(0, 0) > 1:
                head = nums[start]
                hashmap[head] -= 1
                start += 1
        # Step 5: 返回答案 (最大长度)
        return max_len
```

再再再来一题 (声音变得沙哑)！下面一题是上面一题的变种，运用该模板，只需要改**一个**参数

1004. 最大连续1的个数 III

```
class Solution:
    def longestOnes(self, nums: List[int], k: int) -> int:
        max_len, hashmap = 0, {}

        start =  0
        for end in range(len(nums)):
            tail = nums[end]
            hashmap[tail] = hashmap.get(tail, 0) + 1
            if hashmap.get(0, 0) <= k:
                max_len = max(max_len, end - start + 1)

            # 相比较于上一题，只需要把1改成k
            while hashmap.get(0, 0) > k:
                head = nums[start]
                hashmap[head] -= 1
                start += 1
        return max_len
```

`[再 for i in range(4)]`来一题！(已经接近失声)

1208. 尽可能使字符串相等

```
class Solution:
    def equalSubstring(self, s: str, t: str, max_cost: int) -> int:
        # Step 1: 定义需要维护的变量
        # 因为是求最大长度，所以有max_len，又同时涉及计算开销 (和求和一个道理)， 所以还要一个
cur_cost
        cur_cost, max_len = 0, 0

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(t)):
            # Step 3
            # 更新需要维护的变量 (cur_cost)
            # 每一对字符的order差值就是当前时间点的开销，直接累积在cur_cost上即可
            # cur_cost只要不超过最大开销，就更新max_len
            cur_cost += abs(ord(s[end]) - ord(t[end]))
            if cur_cost <= max_cost:
                max_len = max(max_len, end - start + 1)

            # Step 4
            # 根据题意，  题目的窗口长度可变: 这个时候一般涉及到窗口是否合法的问题
            # 这时要用一个while去不断移动窗口左指针, 从而剔除非法元素直到窗口再次合法
            # 当cur_cost大于最大开销时候，窗口不合法
            # 所以需要不断移动窗口左指针直到窗口再次合法 (cur_cost <= max_cost)
            while cur_cost > max_cost:
                cur_cost -= abs(ord(s[start])-  ord(t[start]))
                start += 1
        # Step 5: 返回答案 (最大长度)
        return max_len
```

再.....咳咳

1052. 爱生气的书店老板

```
class Solution:
    def maxSatisfied(self, customers: List[int], grumpy: List[int], minutes: int) ->
int:
        # Step 1
        # 定义需要维护的变量,
        # 因为涉及求和所以定义sum_和max_sum, 同时需要知道老板什么时候'发动技能'，再定义一个
max_start
        sum_, max_sum, max_start = 0, 0, 0

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(customers)):
            # Step 3
            # 更新需要维护的变量 (sum_)
            # 注意：这里只要当老板在当前时间点会发脾气的时候才维护
            # sum_就不说了，和前面N道题的维护方法一样，新多出来的max_start也就是记录一样时间点而
已，没什么fancy的
            if grumpy[end] == 1:
                sum_ += customers[end]
            if sum_ > max_sum:
                max_sum = sum_
                max_start = start

            # Step 4
            # 根据题意可知窗口长度固定 (老板技能持续时间固定)，所以用if
            # 窗口左指针前移一个单位保证窗口长度固定, 同时提前更新需要维护的变量 (sum_,
max_avg)
            if end >= minutes - 1:
                if grumpy[start]:
                    sum_ -= customers[start]
                start += 1

        # 这里对比其他题目多了一小步: 在找到老板发动技能的最大收益时间点(max_start)后
        # 需要把受技能影响时间段中的grumpy全部置0 - 代表老板成功压制了自己的怒火
        for i in range(max_start, max_start + minutes):
            grumpy[i] = 0

        # Step 5: 再遍历一遍数组求customer总数量并且返回结果
        res = 0
        for i in range(len(customers)):
            if not grumpy[i]:
                res += customers[i]
        return res
```

一片寂静.....

1423. 可获得的最大点数

```
class Solution:
    # 这题相比前面的题目加了一丢丢小的变通: 题目要求首尾串最大点数，其实就是求非首尾串的连续序列的最小点数
    def maxScore(self, cardPoints: List[int], k: int) -> int:
        # 特解
        n = len(cardPoints)
        if k == n:
            return sum(cardPoints)

        # Step 1
        # 定义需要维护的变量，因为涉及求和所以定义sum_和min_sum
        m = n - k
        sum_, min_sum = 0, math.inf

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(n):
            # Step 3
            # 更新需要维护的变量 (sum_)
            sum_ += cardPoints[end]

            # Step 4
            # 根据题意可知窗口长度固定，所以用if
            # 窗口左指针前移一个单位保证窗口长度固定, 同时提前更新需要维护的变量 (min_sum，sum_)
            if end >= m - 1:
                min_sum = min(min_sum, sum_)
                sum_ -= cardPoints[start]
                start += 1
        # Step 5: 返回答案 (总点数减去非首尾串的连续序列的最小点数就可以得到首尾串的最大点数)
        return sum(cardPoints) - min_sum
```

1151. 最少交换次数来组合所有的 1

```
class Solution:
    def minSwaps(self, data: List[int]) -> int:
        # 先数出一共有多少个1，输出来的个数就是窗口的长度
        num_ones = 0
        for i in range(len(data)):
            if data[i] == 1:
                num_ones += 1

        # Step 1
        # 定义需要维护的变量，求最小swap次数其实就是求窗口中0个数的最小值，因此定义num_zeros,
min_num_zeros
        num_zeros, min_num_zeros = 0, math.inf

        # Step 2: 定义窗口的首尾端 (start, end)， 然后滑动窗口
        start = 0
        for end in range(len(data)):
            # Step 3
            # 更新需要维护的变量 (num_zeros， min_num_zeros)
            if data[end] == 0:
                num_zeros += 1
            if end - start + 1 == num_ones:
                min_num_zeros = min(min_num_zeros, num_zeros)

            # Step 4
            # 根据题意可知窗口长度固定 (数组1的总个数)，所以用if
            # 窗口左指针前移一个单位保证窗口长度固定, 同时提前更新需要维护的变量 (num_zeros)
            if end >= num_ones - 1:
                if data[start] == 0:
                    num_zeros -= 1
                start += 1

        # Step 5: 返回答案 (如果min_num_zeros依旧是math.inf说明数组没有1存在，不能swap，返回
0即可)
        if min_num_zeros == math.inf:
            return 0
        return min_num_zeros
```

OK.....到目前为止，12道中等题全部结束。读者到此应该可以感觉到这12道题目其实就是遵循一个模子，沿用楼主share的思路只需要修改个别变量就可以全部秒杀！当然，模板只是对思路进行浓缩，楼主不建议大家硬背模板，而是建议先结合模板和几道样题，通过笔纸反复模拟出窗口的滑动过程去加深理解滑动窗口类型题目的共通之处。

Eason在此祝愿大家今后在面试遇见同类型的题目可以直接秒杀，如果看官觉得本文有帮助，希望可以三连以示鼓励！

上一篇：2

48

条评论 >_

编辑

预览

精选评论(2)

做了3-4道题, 的确总结的挺好的; 1.我觉得首先最重要的是要审题, 读懂这是不是滑动窗口能解决的题目; 2.第二还是要审题, 搞懂, 初始值是什么, 比如求和要sum, 去重用dict(), 合法/不合法的定义, 合法需要更新的数值, 不合法需要更新的数值, 还有不合法需要移动start指针; 3.最后修补一下边界情况, 比如完全没有符合的返回0就好;

感觉总结的蛮好的, 做了4+题目, 遇到这种滑动窗口的题目, 就可以沉得住气了

我点赞了我是人了，祝好运，祝我上腾讯！！！

评论(48)

好！！！ 茅塞顿开 自己写了两题看了这个回答 越看越有味道！

写的太好了，做了几题就领会了这个模板，遇到滑动窗口的题可以自己写出来了，感谢大佬的总结

这个模板还可以套在340题上 写了个java的

```
class Solution {
    public int lengthOfLongestSubstringKDistinct(String s, int k) {
        int l=0;
        Map<Character, Integer> map = new HashMap<>();
        int size = 0;
        for(int r =0; r< s.length(); r++){
            char cur = s.charAt(r);
            map.put(cur, map.getOrDefault(cur,0) +1);
            if (map.size() <=k){
                size= Math.max(size, r-l+1);
            }

            while(map.size() >k) {
                char left = s.charAt(l);
                int count = map.getOrDefault(left, 0)-1;
                if (count == 0){
                    map.remove(left);
                }else{
                    map.put(left, count);
                }
                l++;
            }
        }
        return size;
    }
}
```

很受用，感谢！

套用模板解219题

```
class Solution:
    def containsNearbyDuplicate(self, nums: List[int], k: int) -> bool:

        #step1：定义需要维护的变量
        record=set()#长度最多为k+1的哈希表

        #step2：定义长度为k+1的滑动窗口，判断窗口内是否存在两个取值相同的元素
        start=0
        for end in range(len(nums)):
            #step3：更新需要维护的变量&判断是否满足题目条件
            if nums[end] in record:
                return True
            record.add(nums[end])

            if end-start+1>=k+1:#长度大于k+1了
                record.remove(nums[start])#删除最左边的元素
                start+=1
        return False
```

学习了，成功按照思路解决了一道题，感谢感谢

忍不住评论下 大佬nb！

大佬nb

大佬牛逼拜读

emmm，在大佬面前我讲一下自己的想法，对于第一题k固定的情况，其实一个if就够了，因为达到了第一个if的条件必然满足第二个if的条件。

```
code block
```

import math class Solution: def findMaxAverage(self, nums: List[int], k: int) -> float: sum,maxaver=0,-math.inf start=0 for end in range(len(nums)): sum+=nums[end]#长度没达到k之前，不断累积值至sum中 if end-start+1==k:#计算平均值 maxaver=max(maxaver,sum/k) sum-=nums[start] start+=1 return maxaver

大佬有个问题请教一下，无重复元素子串这道题。按照你的方法答案是对的，但是哈希表里的元素是重复的，这是不是不合理啊，虽然我们只求max_len.假设序列pwwkew，我一步步走下来最后哈希表是这样的：hashmap{w:2,k:1,e:1}@Eason

写的真好

写的超级好

学习了学习了，以后就套用这个模板