的位置)。

Q

13: 源码逻辑: 5

?

··

1t

索引目录

从所有教程的词条中查询…

首页 > 慕课教程 > Vue3源码分析与构建方案 > 13:源码逻辑:场景五:求解最长递增子序列

全部开发者教程

≔

04: 框架实现: 场景一: 自前 向后的 diff 对比

05:源码阅读:场景二:自后 向前的 diff 对比

06: 框架实现: 场景二: 自后 向前的 diff 对比

07:源码阅读:场景三:新节 点多余旧节点时的 diff 比对

08: 框架实现: 场景三: 新节 点多余旧节点时的 diff 比对

09:源码阅读:场景四:旧节 点多于新节点时的 diff 比对

10:框架实现:场景四:旧节 点多于新节点时的 diff 比对

11: 局部总结: 前四种 diff 场 景的总结与乱序场景

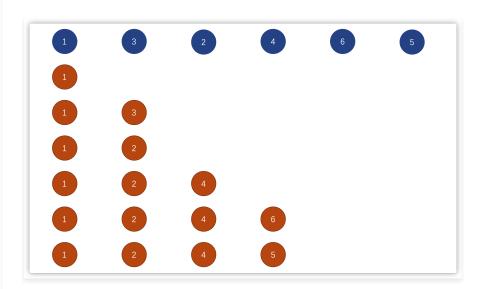
12: 前置知识:场景五:最长 递增子序列

13:源码逻辑:场景五:求解 最长递增子序列

13: 源码逻辑: 场景五: 求解最长递增子序列

我们查看 packages/runtime-core/src/renderer.ts 中的 2410 行代码,可以看到存在一个 getSequen ce 的函数,这个函数的作用就是用来 获取最长递增子序列的下标(必须要获取下标才知道当前元素所处

该函数算法来自于 维基百科(贪心 + 二分查找),该方法的执行逻辑,如 PPT 所示:



我们复制了 vue 3 中 getSequence 的所有代码,并为其加入了详细的备注,如下:

▶ 意见反馈

```
()代码块
    * 获取最长递增子序列下标
     * 维基百科: https://en.wikipedia.org/wiki/Longest_increasing_subsequence
     * 百度百科: https://baike.baidu.com/item/%E6%9C%80%E9%95%BF%E9%80%92%E5%A2%9E%E5%AD%90%E
5
     function getSequence(arr) {
     // 获取一个数组浅拷贝。注意 p 的元素改变并不会影响 arr
     // p 是一个最终的回溯数组,它会在最终的 result 回溯中被使用
8
     // 它会在每次 result 发生变化时,记录 result 更新前最后一个索引的值
10
    const p = arr.slice()
11
    // 定义返回值(最长递增子序列下标),因为下标从 0 开始,所以它的初始值为 0
12
     const result = [0]
     let i, j, u, v, c
13
14
     // 当前数组的长度
      const len = arr.length
      // 对数组中所有的元素进行 for 循环处理, i = 下标
      for (i = 0; i < len; i++) {
       // 根据下标获取当前对应元素
1.8
       const arrI = arr[i]
19
20
21
       if (arrI !== 0) {
         // 获取 result 中的最后一个元素,即: 当前 result 中保存的最大值的下标
23
         j = result[result.length - 1]
24
         // arr[j] = 当前 result 中所保存的最大值
         // arrI = 当前值
2.5
```

♡ 收藏教程

口 标记书签

```
27
       if (arr[j] < arrI) {</pre>
28
          p[i] = j
29
          // 把当前的下标 i 放入到 result 的最后位置
30
          result.push(i)
31
          continue
32
        // 不满足 arr[j] < arrI 的条件,就证明目前 result 中的最后位置保存着更大的数值的下标。
33
        // 但是这个下标并不一定是一个递增的序列,比如: [1,3] 和 [1,2]
35
        // 所以我们还需要确定当前的序列是递增的。
36
        // 计算方式就是通过: 二分查找来进行的
37
       // 初始下标
38
       u = 0
39
40
        // 最终下标
41
        v = result.length - 1
42
        // 只有初始下标 < 最终下标时才需要计算
43
        while (u < v) {
        // (u + v) 转化为 32 位 2 进制, 右移 1 位 === 取中间位置(向下取整)例如: 8 >> 1 = 4;
44
         // https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/R
45
46
         // c 表示中间位。即: 初始下标 + 最终下标 / 2 (向下取整)
47
         c = (u + v) \gg 1
48
         // 从 result 中根据 c (中间位), 取出中间位的下标。
49
         // 然后利用中间位的下标,从 arr 中取出对应的值。
50
        // 即: arr[result[c]] = result 中间位的值
          // 如果: result 中间位的值 < arrI,则 u (初始下标) = 中间位 + 1。即:从中间向右移动一位
51
         if (arr[result[c]] < arrI) {</pre>
52
53
           u = c + 1
54
         } else {
55
           // 否则,则 v(最终下标) = 中间位。即:下次直接从 0 开始,计算到中间位置 即可。
56
57
          }
58
        }
        // 最终,经过 while 的二分运算可以计算出:目标下标位 u
59
        // 利用 u 从 result 中获取下标, 然后拿到 arr 中对应的值: arr[result[u]]
        // 如果: arr[result[u]] > arrI 的,则证明当前 result 中存在的下标 《不是》 递增序列, 『
61
62
       if (arrI < arr[result[u]]) {</pre>
63
        if (u > 0) {
           p[i] = result[u - 1]
64
65
          }
          // 进行替换,替换为递增序列
66
67
          result[u] = i
68
69
70
     // 重新定义 u。此时: u = result 的长度
71
72
   u = result.length
73
   // 重新定义 v。此时 v = result 的最后一个元素
74
     v = result[u - 1]
75
   // 自后向前处理 result, 利用 p 中所保存的索引值,进行最后的一次回溯
76
     while (u-- > 0) {
      result[u] = v
77
78
      v = p[v]
79
    }
80
     return result
81
```

我们可以通过以上代码,对 getSequence([1, 3, 2, 4, 6, 5]) 进行测试, debugger 代码的执行逻辑(具体请查看视频),从而明确当前算法的逻辑。

明确以上逻辑之后,我们把该代码逻辑,直接复制到 vue-next-mini 中即可。

12: 前置知识: 场景五: 最长递增子序列 ◆ 上一节 下一节 ▶ 14: 源码阅读: 场景五: 乱序下的 diff 比对

✔ 我要提出意见反馈

⊡

② []

···

企业服务 网站地图 网站首页 关于我们 联系我们 讲师招募 帮助中心 意见反馈 代码托管

Copyright © 2022 imooc.com All Rights Reserved | 京ICP备 12003892号-11 京公网安备11010802030151号

□

?

.