Sunday • 更新于 2022-10-19

上一节 03：框架实现：... 05：框架实现：... 下一节

04：源码阅读：h 函数，跟踪 ELEMENT + ARRAY_CHILDREN 场景下的源码实现

在前两节我们处理了 `h` 函数下比较简单的场景： `Element + Text Children` 。

那么这一小节，我们来看 `Element + Array Children` 的场景。

我们先来看测试实例 `packages/vue/examples/imooc/runtime/h-element-ArrayChildren.html` ：

代码块

```
<script>
  const { h } = Vue

  const vnode = h('div', {
    class: 'test'
  }, [
    h('p', 'p1'),
    h('p', 'p2'),
    h('p', 'p3')
  ])

  console.log(vnode);
</script>
```

最终打印为（剔除无用的）：

代码块

```
{
  "__v_isVNode": true,
  "type": "div",
  "props": { "class": "test" },
  "children": [
    {
      "__v_isVNode": true,
      "type": "p",
      "children": "p1",
      "shapeFlag": 9
    },
    {
      "__v_isVNode": true,
      "type": "p",
      "children": "p2",
      "shapeFlag": 9
    },
    {
      "__v_isVNode": true,
      "type": "p",
      "children": "p3",
      "shapeFlag": 9
    }
  ],
  "shapeFlag": 17
}
```

1. `children`：为数组
2. `shapeFlag`： `17`

而这两点，也是 `h` 函数处理这种场景下，最不同的地方。

那么下面我们就跟踪源码，来看一下这次 `h` 函数的执行逻辑，由测试案例可知，我们一共触发了 `4` 次 `h` 函数：

1. 第一次触发 `h` 函数：

```
1    h('p', 'p1')
```

2. 进入 `_createVNode` 方法，此时的参数为：

```
function _createVNode(
  type: VNodeTypes | ClassComponent | typeof NULL_DYNAMIC_COMPONENT,  type = "p"
  props: (Data & VNodeProps) | null = null,  props = null
  children: unknown = null,  children = "p1"
  patchFlag: number = 0,  patchFlag = 0
  dynamicProps: string[] | null = null,  dynamicProps = null
  isBlockNode = false  isBlockNode = false
```

3. 触发 `createBaseVNode` 时， `shapeFlag = 1`

    1. 进入 `createBaseVNode`
    2. 触发 `normalizeChildren(vnode, children)`

        1. 进入 `normalizeChildren`
        2. 进入 `else`，执行 `type = ShapeFlags.TEXT_CHILDREN`，此时 `type = 8`
        3. 最后执行 `vnode.shapeFlag |= type` ，得到 `vnode.shapeFlag = 9`

4. 以上整体流程与 **02 小节** 看到的完全相同

接下来是 **第二次、第三次** 触发 `h` 函数，这两次触发代码流程与第一次相同，我们可以跳过。

1. 进入到 **第四次** 触发 `h` 函数：

```
1    h('div', {
2        class: 'test'
3      }, [
4        h('p', 'p1'),
5        h('p', 'p2'),
6        h('p', 'p3')
7      ])
```

2. 此时进入到 `_createVNode` 时的参数为：

```
494 function _createVNode(
495   type: VNodeTypes | ClassComponent | typeof NULL_DYNAMIC_COMPONENT,  type = "div"
496   props: (Data & VNodeProps) | null = null,  props = {class: 'test'}
497   children: unknown = null,  children = (3) [{…}, {…}, {…}]
498   patchFlag: number = 0,  patchFlag = 0
499   dynamicProps: string[] | null = null,  dynamicProps = null
500   isBlockNode = false  isBlockNode = false
501 ): VNode {
```

    1. 展开 `children` 数据为 **解析完成之后的** `vnode`：

```
▼children: Array(3)
  ▶0: {__v_isVNode: true, __v_skip: true, type: 'p',
  ▶1: {__v_isVNode: true, __v_skip: true, type: 'p',
  ▶2: {__v_isVNode: true, __v_skip: true, type: 'p',
    length: 3
```

3. 代码继续，计算 `shapeFlag = 1`

1. 进入 `createBaseVNode`
2. 执行 `normalizeChildren(vnode, children)`：
    1. 进入 `normalizeChildren`
    2. 因为当前 `children = Array`，所以代码会进入到 `else if (isArray(children))`
    3. 执行 `type = ShapeFlags.ARRAY_CHILDREN`，即：`type = 16`
    4. 接下来执行 `vnode.shapeFlag |= type`
        1. 此时 `vnode.shapeFlag = 1`，转化为二进制：

```
00000000 00000000 00000000 00000001
```

        2. 此时 `type = 16`，转化为二进制：

```
00000000 00000000 00000000 00010000
```

        3. 所以最终 `|=` 之后的二进制为：

```
00000000 00000000 00000000 00010001
```

        转化为 `10进制` 为 `17`

代码执行结束。

由以上代码可知，当我们处理 `ELEMENT + ARRAY_CHILDREN` 场景时：

1. 整体的逻辑并没有变得复杂
2. 第一次计算 `shapeFlag`，依然为 `Element`
3. 第二次计算 `shapeFlag`，因为 `children` 为 `Array`，所以会进入 `else if (array)` 判断

03：框架实现：构建 h 函数，处理 ELEMEN... ◂ 上一节　　下一节 ▸ 05：框架实现：构建 h 函数，处理 ELEMEN...

我要提出意见反馈

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

意见反馈　收藏教程　标记书签