全部开发者教程

构建 h 函数，生成 Vnode

01：前言

02：阅读源码：初见 h 函数，跟踪 Vue 3 源码实现基础逻辑

03：框架实现：构建 h 函数，处理 ELEMENT + TEXT_CHILDREN 场景

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

05：框架实现：构建 h 函数，处理 ELEMENT + ARRAY_CHILDREN 场景

06：源码阅读：h 函数，组件的本质与对应的 VNode

07：框架实现：处理组件的 VNode

08：源码阅读：h 函数，跟踪 Text 、 Comment、

Sunday • 更新于 2022-10-19

上一节 01：前言 03：框架实现：... 下一节

02：阅读源码：初见 h 函数，跟踪 Vue 3 源码实现基础逻辑

本小节我们通过 `h` 函数生成 `Element` 的 `VNode` 来去查看 `h` 函数的源码实现。

1. 创建测试实例 `packages/vue/examples/imooc/runtime/h-element.html`：

代码块

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

  const vnode = h('div', {
    class: 'test'
  }, 'hello render')

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

2. `h` 函数的代码位于 `packages/runtime-core/src/h.ts` 中，为 `174` 行增加 `debugger`

h 函数

1. 代码进入 `h` 函数

    1. 通过源码可知， `h` 函数接收三个参数：
    2. `type`： 类型。比如当前的 `div` 就表示 `Element` 类型
    3. `propsOrChildren`： `props` 或者 `children`
    4. `children`： 子节点
    5. 在这三个参数中，第一个和第三个都比较好理解，它的第二个参数代表的是什么意思呢？查看 官方示例 可知： `h` **函数存在多种调用方式**：

代码块

```
import { h } from 'vue'

// 除了 type 外，其他参数都是可选的
h('div')
h('div', { id: 'foo' })

// attribute 和 property 都可以用于 prop
// Vue 会自动选择正确的方式来分配它
h('div', { class: 'bar', innerHTML: 'hello' })

// class 与 style 可以像在模板中一样
// 用数组或对象的形式书写
h('div', { class: [foo, { bar }], style: { color: 'red' } })

// 事件监听器应以 onXxx 的形式书写
h('div', { onClick: () => {} })

// children 可以是一个字符串
h('div', { id: 'foo' }, 'hello')

// 没有 prop 时可以省略不写
```

索引目录

02：阅读源码：初 h 函数

```
h('div', [h('span', 'hello')])

// children 数组可以同时包含 vnode 和字符串
h('div', ['hello', h('span', 'hello')])
```

1. 这些内容在源码中也存在对应的说明（查看 `h.ts` 的顶部注释），并且这种方式在其他的框架或者 `web api` 中也是比较常见的。
2. 那么这样的功能是如何实现的呢？我们继续来看源代码

2. 以下为这一块逻辑的详细注释：

代码块

```
export function h(type: any, propsOrChildren?: any, children?: any): VNode {
  // 获取用户传递的参数数量
  const l = arguments.length
  // 如果用户只传递了两个参数，那么证明第二个参数可能是 props ，也可能是 children
  if (l === 2) {
    // 如果 第二个参数是对象，但不是数组。则第二个参数只有两种可能性：1. VNode 2.普通的 pr
    if (isObject(propsOrChildren) && !isArray(propsOrChildren)) {
      // 如果是 VNode，则 第二个参数代表了 children
      if (isVNode(propsOrChildren)) {
        return createVNode(type, null, [propsOrChildren])
      }
      // 如果不是 VNode， 则第二个参数代表了 props
      return createVNode(type, propsOrChildren)
    }
    // 如果第二个参数不是单纯的 object，则 第二个参数代表了 props
    else {
      return createVNode(type, null, propsOrChildren)
    }
  }
  // 如果用户传递了三个或以上的参数，那么证明第二个参数一定代表了 props
  else {
    // 如果参数在三个以上，则从第二个参数开始，把后续所有参数都作为 children
    if (l > 3) {
      children = Array.prototype.slice.call(arguments, 2)
    }
    // 如果传递的参数只有三个，则 children 是单纯的 children
    else if (l === 3 && isVNode(children)) {
      children = [children]
    }
    // 触发 createVNode 方法，创建 VNode 实例
    return createVNode(type, propsOrChildren, children)
  }
}
```

3. 最终代码将会触发 `createVNode` 方法：

    1. 代码进入 `createVNode`

        1. 此时三个参数的值为：

            1. `type`： `div`
            2. `props`： `{class: 'test'}`
            3. `children`： `hello render`

    2. 代码执行：

代码块

```
  const shapeFlag = isString(type)
      ? ShapeFlags.ELEMENT
      : __FEATURE_SUSPENSE__ && isSuspense(type)
      ? ShapeFlags.SUSPENSE
      : isTeleport(type)
      ? ShapeFlags.TELEPORT
```

```
      ? ShapeFlags.STATEFUL_COMPONENT
      : isFunction(type)
      ? ShapeFlags.FUNCTIONAL_COMPONENT
      : 0
```

3. 最终得到 `shapeFlag` 的值为 `1`， `shapeFlag` 为当前的 **类型标识**：

    1. 这个 `1` 代表的是什么意思呢？查看 `packages/shared/src/shapeFlags.ts` 的代码
    2. 根据 `enum ShapeFlags` 可知： `1` 代表为 `Element`
    3. 即**当前** `shapeFlag = ShapeFlags.Element`

4. 代码继续执行，触发 `createBaseVNode`：

    1. 进入 `createBaseVNode`
    2. 执行：

    代码块

```
    const vnode = {
      __v_isVNode: true,
      __v_skip: true,
      type,
      props,
      key: props && normalizeKey(props),
      ref: props && normalizeRef(props),
      scopeId: currentScopeId,
      slotScopeIds: null,
      children,
      component: null,
      suspense: null,
      ssContent: null,
      ssFallback: null,
      dirs: null,
      transition: null,
      el: null,
      anchor: null,
      target: null,
      targetAnchor: null,
      staticCount: 0,
      shapeFlag,
      patchFlag,
      dynamicProps,
      dynamicChildren: null,
      appContext: null
    } as VNode
```

    3. 生成 `vnode` 对象，此时生成的 `vnode` 值为：

    代码块

```
  anchor: null
  appContext: null
  children: "hello render"
  component: null
  dirs: null
  dynamicChildren: null
  dynamicProps: null
  el: null
  key: null
  patchFlag: 0
  props: {class: 'test'}
  ref: null
  scopeId: null
  shapeFlag: 1
  slotScopeIds: null
  ssContent: null
  ssFallback: null
  staticCount: 0
  suspense: null
```

```
21    targetAnchor: null
22    transition: null
23    type: "div"
24    __v_isVNode: true
25    __v_skip: true
```

剔除对我们无用的属性之后，得到：

代码块

```
1     children: "hello render
2     props: {class: 'test'}
3     shapeFlag: 1 // 表示为 Element
4     type: "div"
5     __v_isVNode: true
```

4. 代码执行 `normalizeChildren(vnode, children)`

    1. 进入 `normalizeChildren` 方法
    2. 代码进入最后的 `else`，执行 `type = ShapeFlags.TEXT_CHILDREN`，执行完成之后，`type = 8`，此时的 `8` 表示为 `ShapeFlags.TEXT_CHILDREN`
    3. **注意**：最后执行 `vnode.shapeFlag |= type`

        1. 此时 `vnode.shapeFlag` 原始值为 `1`，即 `ShapeFlags.ELEMENT`
        2. `type` 的值为 `8`，即 `ShapeFlags.TEXT_CHILDREN`
        3. 而 `|=` 表示为 按位或赋值 运算：`x |= y` 意为 `x = x | y`

            1. 即：`vnode.shapeFlag |= type` 表示为 `vnode.shapeFlag = vnode.shapeFlag | type`
            2. 代入值后表示 `vnode.shapeFlag = 1 | 8`
            3. `1` 是 `10` 进制，转化为 `32` 位的二进制之后为：

                1. `00000000 00000000 00000000 00000001`

            4. `8` 是 `10` 进制，转化为 转化为 `32` 位的二进制之后为：

                1. `00000000 00000000 00000000 00001000`

            5. 两者进行 按位或赋值 之后，得到的二进制为：

                1. `00000000 00000000 00000000 00001001`
                2. 转化为 `10进制` 即为 `9`

    4. 所以，此时 `vnode.shapeFlag` **的值为** `9`

至此，整个 `h` 函数执行完成，最终得到的打印有效值为：

代码块

```
1     children: "hello render
2     props: {class: 'test'}
3     shapeFlag: 9 // 表示为 Element | ShapeFlags.TEXT_CHILDREN 的值
4     type: "div"
5     __v_isVNode: true
```

由以上代码可知：

1. `h` 函数内部本质上只处理了参数的问题
2. `createVNode` 是生成 `vnode` 的核心方法
3. 在 `createVNode` 中第一次生成了 `shapeFlag = ShapeFlags.ELEMENT`，表示为：是一个 `Element` 类型
4. 在 `createBaseVNode` 中，生成了 `vnode` 对象，并且对 `shapeFlag` 的进行 `|=` 运算，最终得到的 `sh`

我要提出意见反馈