

vue3源码剖析01



学习目标

- 调试环境准备
- 初始化流程分析
- 手写实现
- 自定义渲染器实战

调试环境准备

- 迁出Vue3源码: `git clone https://github.com/vuejs/vue-next.git`
- 安装依赖: `yarn --ignore-scripts`
- 生成sourcemap文件, package.json

```
"dev": "node scripts/dev.js --sourcemap"
```

- 编译: `yarn dev`

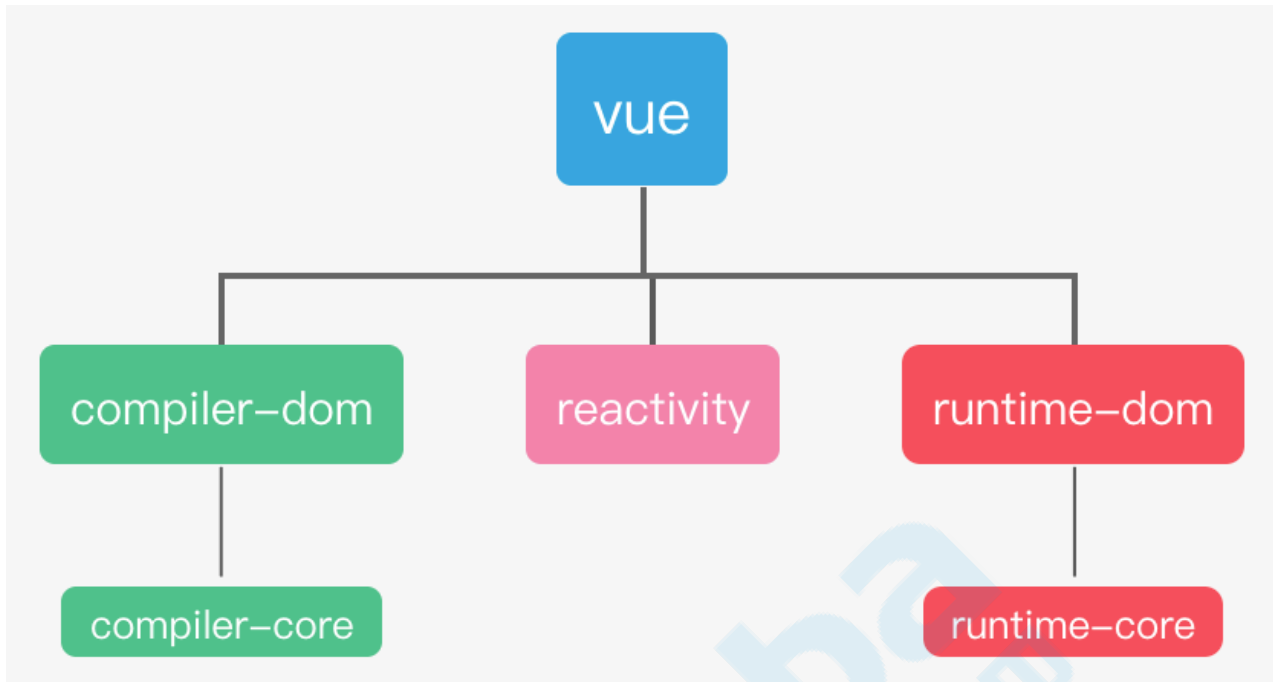
生成结果:

packages\vue\dist\vue.global.js

packages\vue\dist\vue.global.js.map

- 调试范例代码: `yarn serve`

vue3源码架构



初始化过程

小目标

- 捋清楚vue3初始化过程
- setup()是如何生效的

vue3初始化过程

createApp()是如何创建vue实例的；创建的vue实例执行mount()都做了些什么。

测试代码

```
<div id="app">
  <h1>vue3初始化流程</h1>
</div>

<script src="../../dist/vue.global.js"></script>
<script>
  const {createApp} = Vue
  createApp({}).mount('#app')
</script>
```

断点调试createApp()

ensureRenderer() => renderer => createApp()

createAppAPI() => createApp

app.mount() => render() => patch() => processComponent() => mountComponent()

=> setupComponent() => setupRenderEffect()

执行流程

createApp()

packages/runtime-dom/src/index.ts

创建vue实例、扩展mount方法

createRenderer()/baseCreateRenderer()

packages/runtime-core/src/renderer.ts

创建renderer对象，它对外暴露3个重要方法 `render`、`hydrate`、`createApp`，其中 `render` 和 `hydrate` 的实际使用者是createApp()返回的vue实例对象。

createAppAPI(render, hydrate)

packages/runtime-core/src/apiCreateApp.ts

返回生产vue实例的createApp函数。

render的使用者是vue实例的mount方法

我们发现component()/directive()/use()/mixin()这些方法都变成了实例方法，它们也会返回实例本身，链式调用成为可能。

```
createApp({})
  .component('comp', { template: '<div>this is comp</div>' })
  .directive('focus', { mounted(el) { el.focus() } })
  .mount('#app')
```

filter方法被移除了

mount(rootContainer: HostElement, isHydrate?: boolean)

packages/runtime-core/src/apiCreateApp.ts

将 `createApp(rootComponent)` 中传入的根组件转换为vnode，然后渲染到宿主元素rootContainer中。

`render(vnode, container)`

将传入vnode渲染到容器container上。

`patch(n1, n2, container)`

将传入的虚拟节点 `n1` 跟 `n2` 进行对比，并转换为dom操作。初始化时 `n1` 并不存在，因此操作将是一次dom创建。

`mount(rootContainer)`

`packages/runtime-core/src/apiCreateApp.ts`

执行根组件挂载，创建其vnode，并将它render()出来

`render()`

`packages/runtime-core/src/renderer.ts`

执行补丁函数patch()将vnode转换为dom。

`patch(n1, n2, container)`

`packages/runtime-core/src/renderer.ts`

根据n2的类型执行相对应的处理函数。对于根组件，执行的是processComponent()

`processComponent()`

`packages/runtime-core/src/renderer.ts`

执行组件挂载或更新，由于首次执行时n1为空，因此执行组件挂载逻辑mountComponent()

`mountComponent()`

`packages/runtime-core/src/renderer.ts`

创建组件实例，执行setupComponent()设置其数据状态，其中就包括setup()选项的执行

setup()如何生效

在vue3中如果要使用composition-api，就需要写在setup()中，它是如何生效并和options-api和谐共处的？

测试代码

```
<div id="app">
  <h1>setup() 如何生效</h1>
  <p>{{foo}}</p>
</div>

<script src="../../dist/vue.global.js"></script>
<script>
  const { createApp, h, ref } = Vue

  createApp({
    setup() {
      const foo = ref('hello, vue3!')
      return { foo }
    }
  }).mount('#app')
</script>
```

执行过程

根组件执行挂载mount()时，执行渲染函数render()获取组件vnode，然后执行补丁函数patch()将其转换为真实dom，对于组件类型会调用processComponent()，这里会实例化组件并处理其setup选项。

▼ Call Stack	
➡ setup	02-setup.html:22
callWithErrorHandling	errorHandling.ts:69
setupStatefulComponent	component.ts:486
setupComponent	component.ts:441
mountComponent	renderer.ts:1149
processComponent	renderer.ts:1101
patch	renderer.ts:462
render	renderer.ts:2090
mount	apiCreateApp.ts:221
app.mount	index.ts:70
(anonymous)	02-setup.html:25

setupComponent()

packages/runtime-core/src/component.ts

初始化props、slots和data

setupStatefulComponent(instance, isSSR)

packages/runtime-core/src/component.ts

代理组件实例上下文，调用setup()

setup()会接收两个参数，分别是props和setupContext，可用于获取属性、插槽内容和派发事件

```
createApp({
  props: ['bar'], // 属性依然需要声明
  setup(props) {
    // 作为setupResult返回
    return { bar: props.bar }
  }
  // 传入rootProps
}, {bar: 'bar'}).mount('#app')
```

handleSetupResult(instance, setupResult, isSSR)

packages/runtime-core/src/component.ts

处理setup返回结果，如果是函数则作为组件的渲染函数，如果是对象则对其做响应化处理。

自定义渲染器

可以自定义渲染器，将获取到的vnode转换为特定平台的特定操作。

范例：利用canvas画图

第一步：我们创建一个渲染器，需要给它提供节点和属性的操作

```
const { createRenderer } = Vue

// 创建一个渲染器，给它提供节点和属性操作
const nodeOps = {}
const renderer = createRenderer(nodeOps);
```

第二步：创建画布，我们通过扩展默认createApp做到这一点

```
// 保存画布和其上下文
let ctx;
let canvas;

// 扩展mount，首先创建一个画布元素
function createCanvasApp(App) {
  const app = renderer.createApp(App);
  const mount = app.mount
  app.mount = function(selector) {
    canvas = document.createElement('canvas');
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
    document.querySelector(selector).appendChild(canvas);
    ctx = canvas.getContext('2d');
    mount(canvas);
  }
  return app
}

// 创建app实例
createCanvasApp({}).mount('#app')
```

此时已经可以看到canvas，但是会报一个错误，是因为我们上面组件是空的，vue想要创建一个comment元素导致

第三步：添加模板

```

<script type="text/x-template" id="chart">
  <bar-chart :data="chartData"></bar-chart>
</script>

<div id="app"></div>

```

```

createCanvasApp({
  template: '#chart',
  data() {
    return {
      chartData: [
        { title: "青铜", count: 200, color: "brown" },
        { title: "砖石", count: 300, color: "skyblue" },
        { title: "星耀", count: 100, color: "purple" },
        { title: "王者", count: 50, color: "gold" }
      ]
    }
  },
}).mount('#app')

```

第四步：节点操作实现

```

// 保存canvas实例和上下文
let ctx, canvas
const nodeOps = {
  createElement: (tag, isSVG, is) => {
    // 创建元素时由于没有需要创建的dom元素，只需返回当前元素数据对象
    return {tag}
  },
  insert: (child, parent, anchor) => {
    // 我们重写了insert逻辑，因为在我们canvasApp中不存在实际dom插入操作
    // 这里面只需要将元素之间的父子关系保存一下即可
    child.parent = parent
    if (!parent.childs) {
      parent.childs = [child]
    } else {
      parent.childs.push(child)
    }

    // 只有canvas有nodeType，这里就是开始绘制内容到canvas
    if (parent.nodeType === 1) {
      draw(child)
    }
  },
  patchProp(el, key, prevValue, nextValue) {
    el[key] = nextValue;
  }
}

```



```
}  
}
```

第四步：绘图逻辑

```
const draw = (el, noClear) => {  
  if (!noClear) {  
    ctx.clearRect(0, 0, canvas.width, canvas.height)  
  }  
  if (el.tag === 'bar-chart') {  
    const { data } = el;  
    const barWidth = canvas.width / 10,  
          gap = 20,  
          paddingLeft = (data.length * barWidth + (data.length - 1) * gap) / 2,  
          paddingBottom = 10;  
  
    // x轴  
    // 柱状图  
    data.forEach(({ title, count, color }, index) => {  
      const x = paddingLeft + index * (barWidth + gap)  
      const y = canvas.height - paddingBottom - count  
      ctx.fillStyle = color  
      ctx.fillRect(x, y, barWidth, count)  
      // text  
    });  
  }  
  // 递归绘制子节点  
  el.children && el.children.forEach(child => draw(child, true));  
}
```

配置自定义组件白名单：

```
app.config.isCustomElement = tag => tag === 'bar-chart'
```

作业

按课上讲解手写vue3初始化流程

要求：学习中心提交代码截图和代码

通过标准：能够正常运转，完成既定功能