vue3源码剖析02

学习目标

- 编译器原理
- vue3编译过程剖析
- vue3编译优化策略
- vue3 patch算法剖析

编译器原理

template => ast => render

模板

```
<div id="app">
  <h2>{{msg}}</h2>
</div>
```

抽象语法树

```
AST:                                    template-explorer.global.js:29464
▼ {type: 0, children: Array(1), helpers: Array(4), compon
  ents: Array(0), directives: Array(0), …}
    cached: 0
  ▶ children: [{…}]
  ▶ codegenNode: {type: 13, tag: ""div"", props: {…}, chi…
  ▶ components: []
  ▶ directives: []
  ▶ helpers: (4) [Symbol(toDisplayString), Symbol(createV…
  ▶ hoists: []
  ▶ imports: []
  ▶ loc: {start: {…}, end: {…}, source: "<div id="app">↵ …
    temps: 0
    type: 0
  ▶ __proto__: Object
```

渲染函数

```
import { toDisplayString as _toDisplayString, createVNode as _createVNode,
openBlock as _openBlock, createBlock as _createBlock } from "vue"

export function render(_ctx, _cache) {
  return (_openBlock(), _createBlock("div", { id: "app" }, [
    _createVNode("h2", null, _toDisplayString(_ctx.msg), 1 /* TEXT */)
  ]))
}
```

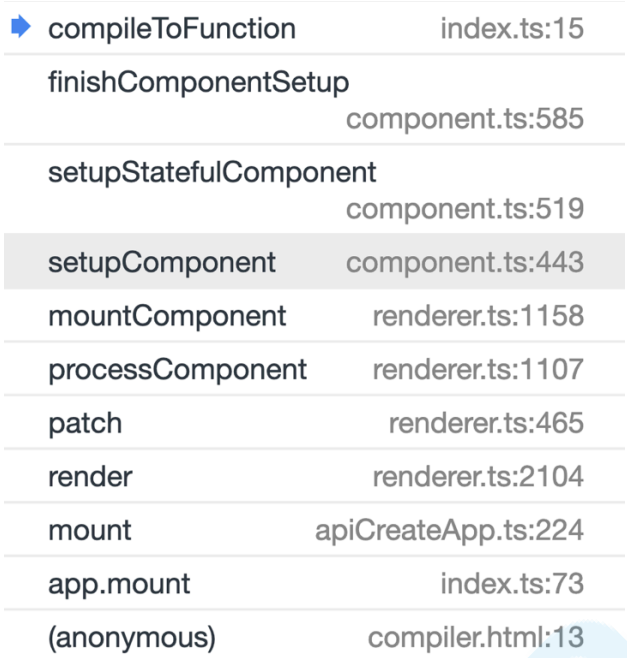
Vue3编译过程剖析

测试代码

```
<div id="app">
  {{foo}}
</div>

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

整体流程



template获取

app.mount()获取了template，vue/index.ts

```
const { mount } = app
app.mount = (containerOrSelector: Element | string): any
  const container = normalizeContainer(containerOrSelect
  if (!container) return
  const component = app._component  component = {templa
  if (!isFunction(component) && !component.render && !c
    component.template = container.innerHTML  container
  }
  // clear content before mounting
  container.innerHTML = ''  container = div#app {align:
  const proxy = mount(container)
  container.removeAttribute('v-cloak')
  container.setAttribute('data-v-app', '')
  return proxy
}
```

编译template

compile将传入template编译为render函数，component.ts

```
if (__DEV__) {
  startMeasure(instance, `compile`)  instance = {uid:
}
Component.render = compile(Component.template, {
  isCustomElement: instance.appContext.config.isCustom
  delimiters: Component.delimiters
})
if (__DEV__) {
  endMeasure(instance, `compile`)
```

实际执行的是baseCompile，compiler-dom/src/index.ts

第一步解析-parse：解析字符串template为抽象语法树ast

```
const ast = isString(template) ? baseParse(template, optio
const [nodeTransforms, directiveTransforms] = getBaseTrans
  prefixIdentifiers
)
```

```
Object
  cached: 0
▶ children: [{…}]
  codegenNode: undefined
▶ components: []
▶ directives: []
▶ helpers: []
▶ hoists: []
▶ imports: []
▶ loc: {start: {…}, end: {…}, source: "↵
  temps: 0
  type: 0
▶ __proto__: Object
```

第二步转换-transform：解析属性、样式、指令等

```
transform(
  ast,
  extend({}, options, {  options = {isVoidTag: ƒ, isNative
    prefixIdentifiers,
    nodeTransforms: [
      ...nodeTransforms,
      ...(options.nodeTransforms || []) // user transforms
    ],
    directiveTransforms: extend(
      {},
      directiveTransforms,
      options.directiveTransforms || {} // user transforms
    )
  })
)
```

第三步生成-generate：将ast转换为渲染函数

```
return generate(
  ast,
  extend({}, options, {
    prefixIdentifiers
  })
)
```

编译优化

静态节点提升

```
<div id="app">
  <h2>msg</h2>
</div>
```

```
import { createVNode as _createVNode, openBlock as
_openBlock, createBlock as _createBlock } from "vue"

const _hoisted_1 = { id: "app" }
const _hoisted_2 = /*#__PURE__*/_createVNode("h2", null,
"msg", -1 /* HOISTED */)

export function render(_ctx, _cache) {
  return (_openBlock(), _createBlock("div", _hoisted_1, [
    _hoisted_2
  ]))
}

// Check the console for the AST
```

补丁标记和动态属性记录

```
<div id="app">
  <h2>msg</h2>
  <div :title="title">aaa</div>
</div>
```

```
export function render(_ctx, _cache) {
  return (_openBlock(), _createBlock("div", _hoisted_1, [
    _hoisted_2,
    _createVNode("div", { title: _ctx.title }, "aaa", 8 /* PROPS */, ["title"]
)
  ]))
}
```

缓存事件处理程序

```
<div @click="onClick(id)">hello, vue3!</div>
```

```
export function render(_ctx, _cache) {
  return (_openBlock(), _createBlock("div", _hoisted_1, [
    _hoisted_2,
    _createVNode("div", { title: _ctx.title }, "aaa", 8 /* PROPS */, ["title"]
),
    _createVNode("div", {
      onClick: _cache[1] || (_cache[1] = $event => (_ctx.onClick(_ctx.id)))
    }, "hello, vue3!")
  ]))
}
```

块 block

```
<div id="app">
  <div>
    <div></div>
    <div></div>
    <div>
      <span>{{msg}}</span>
    </div>
  </div>
</div>
```

```
export function render(_ctx, _cache) {
  return (_openBlock(), _createBlock("div", _hoisted_1, [
    _createVNode("div", null, [
      _hoisted_2,
      _hoisted_3,
      _createVNode("div", null, [
        _createVNode("span", null, _toDisplayString(_ctx.msg), 1 /* TEXT */)
      ])
    ])
  ]))
}
```

Vue3虚拟dom和patch算法

vue3对vnode结构做了调整以适应编译器的优化策略，相对应的patch算法也会利用这些变化提高运行速度

新的vnode结构

测试代码

patch.html

```
<div id="app">
  <h1>patch</h1>
  <p>{{foo}}</p>
</div>

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

```
  setTimeout(() => {
    app.foo = 'fooooooooooo'
  }, 1000);
</script>
```

创建VNode

mount()执行时，创建根组件VNode，packages/runtime-core/src/apiCreateApp.ts

```
mount(rootContainer: HostElement, isHydrate?: boolean):
  if (!isMounted) {
    const vnode = createVNode(rootComponent as Component
    // store app context on the root VNode.
```

渲染VNode

render(vnode, rootContainer)方法将创建的vnode渲染到根容器上。

```
} else {
  render(vnode, rootContainer)
}
isMounted = true
```

初始patch

传入oldVnode为null，初始patch为创建行为。

```
  }
} else {
  patch(container._vnode || null, vnode, container)
}
```

使用mountComponent将n2转换为dom

```
} else {
    mountComponent(
      n2,
      container,
      anchor,
      parentComponent,
      parentSuspense,
      isSVG,
      optimized
    )
}
```

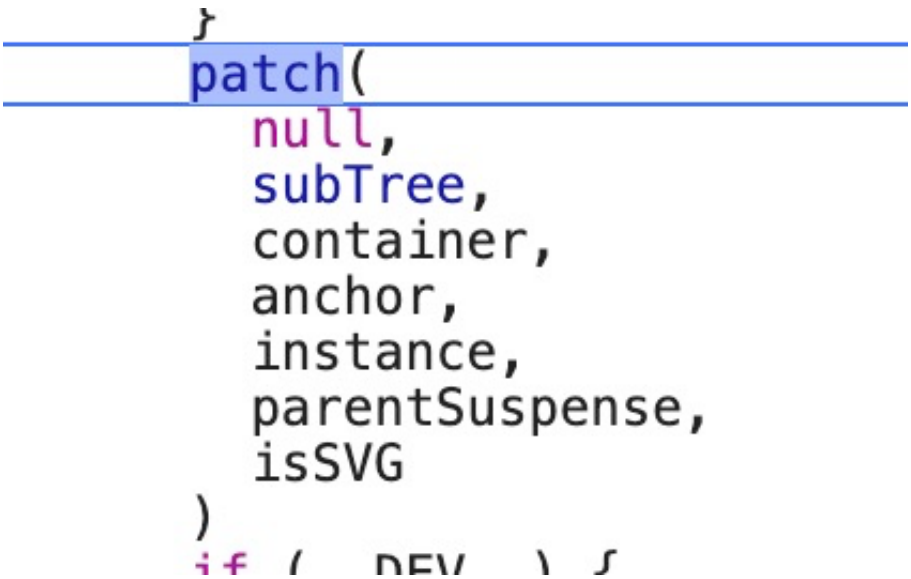
创建一个渲染副作用，执行render，获得vnode之后，在执行patch转换为dom

```
setupRenderEffect(
  instance,
  initialVNode,
  container,
  anchor,
  parentSuspense,
  isSVG,
  optimized
)
```

setupRenderEffect在初始化阶段核心任务是执行instance的render函数获取subTree

```
}
const subTree = (instance.subTree = renderComponentRoot(instance))
```

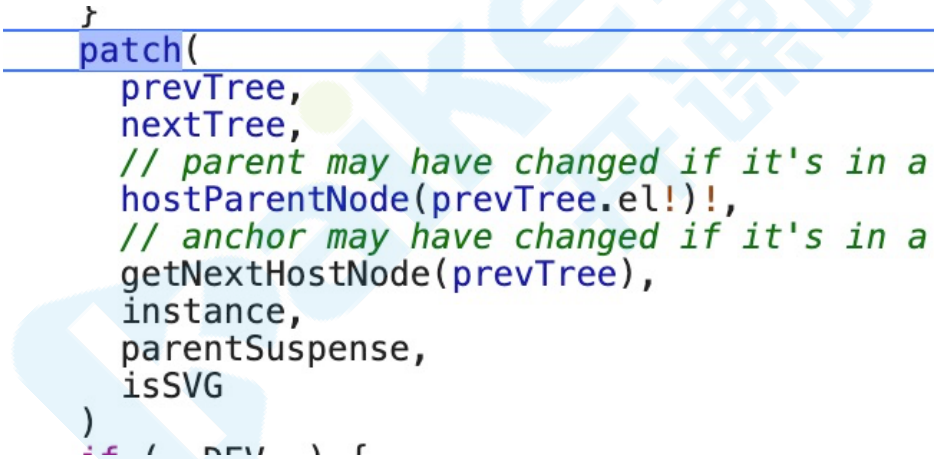
最后patch这个subTree



更新流程

更新阶段，patch函数对比新旧vnode，得出dom操作内容。

componentEffect中会调用patch，并传入新旧两个vnode



多个子元素更新

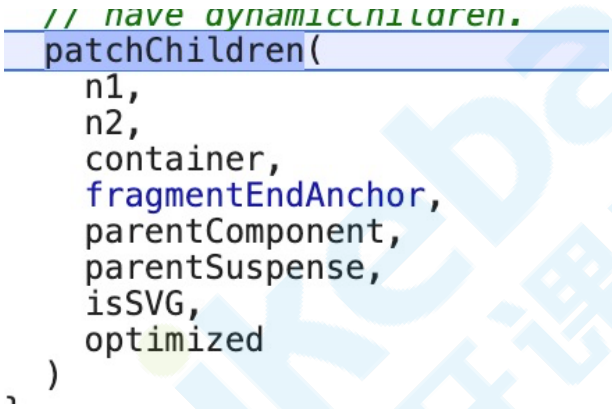
如果同时存在多个子元素，比如使用v-for时的情况：

```
<div id="app">
  <div v-for="item in arr" :key="item">{{item}}</div>
</div>

<script src="../dist/vue.global.js"></script>
<script>
  const { createApp, h } = Vue
  createApp({
    data() {
```

```
      return {
        arr: ['a', 'b', 'c', 'd']
      }
    },
    mounted() {
      setTimeout(() => {
        this.arr.splice(1, 0, 'e')
      }, 1000);
    },
  }).mount('#app')
</script>
```

典型的重排操作，使用patchChildren更新



设置了key的情况下，走patchKeyedChildren

```
// ['a', 'b', 'c', 'd']
// ['a', 'e', 'b', 'c', 'd']

// 1.从开始同步：掐头
// ['b', 'c', 'd']
// ['e', 'b', 'c', 'd']

// 2.从结尾同步：去尾
// []
// ['e']

// 3.新增
```