vue3源码剖析02

学习目标

- composition-api体验
- Vue3响应式源码学习
- 响应式原理：Vue2 vs Vue3
- 造轮子之旅

composition-api

文档

https://vue-composition-api-rfc.netlify.com

初体验

```
<div id="app">
  <h1>composition-api</h1>
  <p @click="add">{{state.counter}}</p>
  <p>{{state.doubleCounter}}</p>
</div>

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

  const app = createApp({
    setup() {
      const state = reactive({
        counter: 0,
        doubleCouter: computed(() => counter * 2)
      })
```

```
      const add = () => {
        data.counter++
      }

      return { state, add }
    },
  }).mount('#app')
</script>
```

更好的逻辑复用和代码组织

```
<meta charset="UTF-8">
<script src="../dist/vue.global.js"></script>

<div id="app">
  <h1>logic reuse</h1>
</div>

<script>
  const { createApp, reactive, onMounted, onUnmounted, toRefs } = Vue;

  // 鼠标位置侦听
  function useMouse() {
    // 数据响应化
    const state = reactive({ x: 0, y: 0 })
    const update = e => {
      state.x = e.pageX
      state.y = e.pageY
    }
    onMounted(() => {
      window.addEventListener('mousemove', update)
    })
    onUnmounted(() => {
      window.removeEventListener('mousemove', update)
    })
    // 转换所有key为响应式数据
    return toRefs(state)
  }
  // 事件监测
  function useTime() {
    const state = reactive({ time: new Date() })
    onMounted(() => {
      setInterval(() => {
        state.time = new Date()
      }, 1000)
    })
    return toRefs(state)
  }
```

```
  // 逻辑组合
  const MyComp = {
    template: `
      <div>x: {{ x }} y: {{ y }}</div>
      <p>time: {{time}}</p>
    `,
    setup() {
      // 使用鼠标逻辑
      const { x, y } = useMouse()
      // 使用时间逻辑
      const { time } = useTime()
      // 返回使用
      return { x, y, time }
    }
  }
  createApp(MyComp).mount('#app')
</script>
```

对比mixins，好处显而易见：

- x,y,time来源清晰
- 不会与data、props等命名冲突

更好的维护性

更好的类型推断

Vue最初选项API中存在大量this上下文，对TypeScript类型推断很不友好。在composition-api中仅利用纯变量和函数，规避了对this的使用，自然的拥有良好的类型推断能力。

Vue3中响应式源码学习

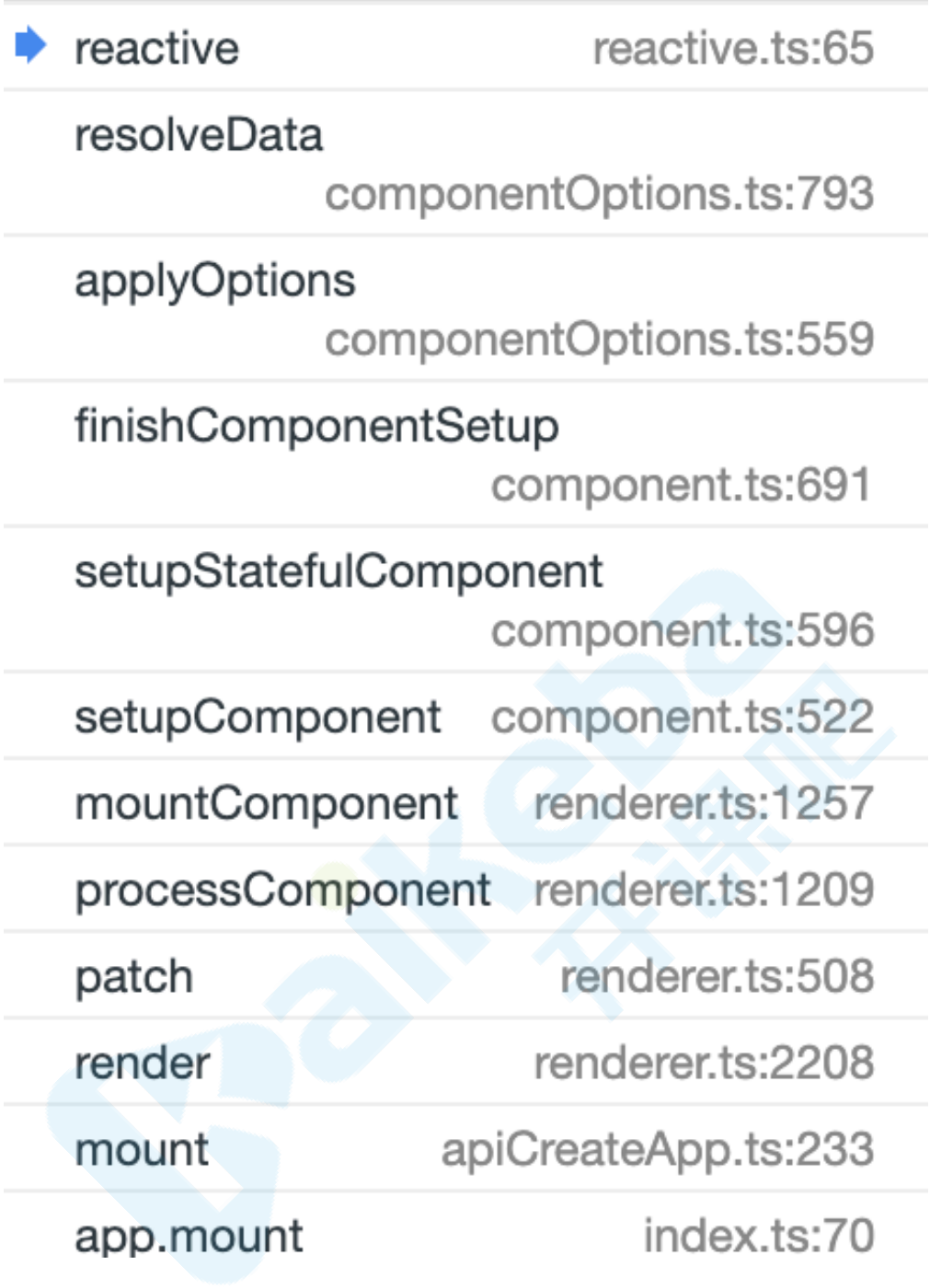
测试代码

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

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

整体流程

applyOptions中对data选项做**响应式处理使用的是reactive函数**



setupRenderEffect函数中**使用effect函数做依赖收集**

```
const setupRenderEffect: SetupRenderEffectFn = (
  instance,  instance = {uid: 0, vnode: {…}, type: {…},
  initialVNode,  initialVNode = {__v_isVNode: true, __v
  container,  container = div#app {align: "", title: ""
  anchor,  anchor = null
  parentSuspense,  parentSuspense = null
  isSVG,  isSVG = false
  optimized  optimized = false
) => {
  // create reactive effect for rendering
  instance.update = effect(function componentEffect() {
    if (!instance.isMounted) {
      let vnodeHook: VNodeHook | null | undefined
      const { el, props } = initialVNode
      const { bm, m, parent } = instance
```

响应式原理：vue2 vs vue3

数据变化可侦测，从而对使用数据的地方进行更新。

vue2的方式

Object.defineProperty()

```
// 拦截每个key，从而可以侦测数据变化
function defineReactive(obj, key, val) {
  Object.defineProperty(obj, key, {
    get() {
      return val
    },
    set(v) {
      val = v
      update()
    }
  })
}

function update() {
  console.log(obj.foo);
}
const obj = {}
defineReactive(obj, 'foo', 'foo')

obj.foo = 'fooooooo'
```

vue3的方式

Proxy

```
// 代理整个对象，从而侦测数据变化
function defineReactive(obj) {
  return new Proxy(obj, {
    get(target, key) {
      return target[key]
    },
    set(target, key, val) {
      target[key] = val
      update()
    }
  })
}

function update() {
  console.log(obj.foo);
}

const obj = {}
const observed = defineReactive(obj)

observed.foo = 'fooooooo'
```

Vue2 vs Vue3

vue2中需要递归遍历对象所有key，速度慢

```
// 1.对象响应化：遍历每个key，定义getter、setter
function observe(obj) {
  if (typeof obj !== 'object' || obj == null) {
    return
  }

  const keys = Object.keys(obj)
  for (let i = 0; i < keys.length; i++) {
    const key = keys[i]
    defineReactive(obj, key, obj[key])
  }
}
```

```
function defineReactive(obj, key, val) {
  observe(val)

  Object.defineProperty(obj, key, {
    get() {
      return val
    },
    set(newVal) {
      if (newVal !== val) {
        observe(newVal)
        val = newVal
        dep.notify()
      }
    }
  })
}
```

数组响应式需要额外实现

```
// 数组响应化：覆盖数组原型方法，额外增加通知逻辑
const originalProto = Array.prototype
const arrayProto = Object.create(originalProto)
;['push', 'pop', 'shift', 'unshift', 'splice', 'reverse', 'sort'].forEach(
  method => {
    arrayProto[method] = function() {
      originalProto[method].apply(this, arguments)
      dep.notify()
    }
  }
)
```

新增或删除属性无法监听，需要使用特殊api

```
Vue.set(obj, 'foo', 'bar')
Vue.delete(obj, 'foo')
```

不支持Map、Set、Class等数据结构

作业

按课上讲解手写vue3响应式相关API: reactive、effect、trigger、track等
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