浅谈vue3学习之道

知识点

- 调试环境搭建
- vue3初体验
- 源码结构分析
- Composition API
- 数据响应式革新
- vue3展望

调试环境搭建

- 迁出Vue3源码：`git clone https://github.com/vuejs/vue-next.git`
- 安装依赖：`yarn`

  > 注意只能yarn装，别用npm

- 添加SourceMap文件：
  - rollup.config.js

    ```
    // 76行添加如下代码
    output.sourcemap = true
    ```

  - 修改ts配置，tsconfig.json

    ```
    "sourceMap": true
    ```

- 编译：`yarn dev`

  > 生成结果：packages\vue\dist\vue.global.js

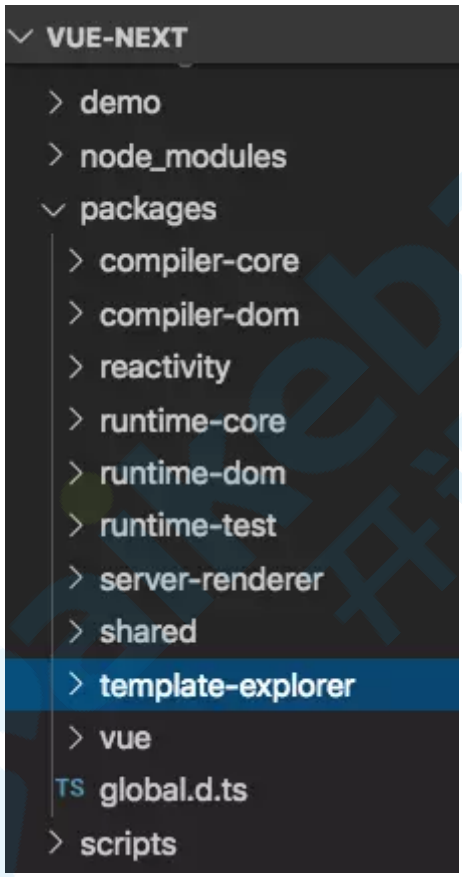
vue3初体验

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>hello vue3</title>
    <script src="../dist/vue.global.js"></script>
</head>

<body>
    <div id='app'></div>
```

```
    <script>
        const App = {
            template: `<h1>{{message}}</h1>`,
            data: { message: 'Hello Vue 3!' }
        }
        Vue.createApp().mount(App, '#app')
    </script>
</body>
</html>
```

源码结构



源码位置是在package文件件内，实际上源码主要分为两部分，编译器和运行时环境。

- 编译器
  - compiler-core 核心编译逻辑
    - 模板解析
    - 代码生成
  - compiler-dom 针对浏览器的编译逻辑
    - v-html
    - v-text
    - v-model
- 运行时环境
  - runtime-core 运行时核心
    - component
    - directive
    - lifecycle
    - watch

        - vnode
    - runtime-dom 运行时针对浏览器的逻辑
        - nodeOps
        - patchProp
    - runtime-test 测试环境仿真，主要为了解决单元测试问题的逻辑 在浏览器外完成测试比较方便
- reactivity 响应式逻辑
- template-explorer 模板解析器 可以这样运行

```
yarn dev template-explorer
```

- vue 代码入口，整合编译器和运行时
- server-renderer 服务器端渲染（TODO）
- share 公用方法

Composition API

Composition API字面意思是组合API，它是为了实现**基于函数**的**逻辑复用机制**而产生的。

基本使用

02-composition-api.html

```
const {
    createApp,
    reactive,
    onMounted,
    ref, toRefs,
    computed, watch
} = Vue;

// 声明组件
const App = {
    template: `
        <div @click="onClick">count: {{ count }}</div>
        <div>doubleCount: {{doubleCount}}</div>
    `,
    setup() {
        const state = reactive({ count: 0 })
        function onClick() {
            state.count += 1
        }

        onMounted(() => {
            console.log('App挂载! ');
        })

        const doubleCount = computed(() => state.count * 2)

        watch(() => {
            console.log(`count变了：${state.count}`)
        })
```

```
        return {...toRefs(state), onClick, doubleCount}
    }
}

createApp().mount(App, '#app')
```

体验逻辑组合

```
const { createApp, reactive, onMounted, onUnmounted, toRefs } = Vue;

// 鼠标位置侦听逻辑
function useMouse() {
    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)
    })
    return toRefs(state)
}
// 时间监测逻辑
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()
        return { x, y, time }
    }
}
createApp().mount(MyComp, '#app')
```

对比mixins

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>composition api</title>
</head>

<body>
    <div id="app"></div>
    <script src="../dist/vue.global.js"></script>
    <script>
        const { createApp } = Vue;

        const useMouse = {
            data() {
                return { x: 0, y: 0 }
            },
            methods: {
                update(e) {
                    this.x = e.pageX
                    this.y = e.pageY
                }
            },
            mounted() {
                window.addEventListener('mousemove', this.update)
            },
            beforeDestroy() {
                window.removeEventListener('mousemove', this.update)
            }
        }

        const useTime = {
            data() {
                return { time: new Date() }
            },
            mounted() {
                setInterval(() => {
                    this.time = new Date()
                }, 1000)
            }
        }

        const MyComp = {
            template: `<div>x: {{ x }} y: {{ y }}</div>
                    <p>time: {{time}}</p>`,
            mixins: [useMouse, useTime]
        }
        createApp().mount(MyComp, '#app')
    </script>
</body>
</html>
```

vue3响应式原理

- Vue2响应式的一些问题：
  - 响应化过程需要遍历data,props等，消耗较大
  - 不支持Set/Map、Class、数组等类型
  - 新加或删除属性无法监听
  - 数组响应化需要额外实现
  - 对应的修改语法有限制
- Vue3响应式原理：使用ES6的Proxy来解决这些问题。

```
function reactive(data) {
  if (typeof data !== 'object' || data === null) {
    return data
  }
  // Proxy相当于在对象外层加拦截
  // http://es6.ruanyifeng.com/#docs/proxy
  const observed = new Proxy(data, {
    // 获取拦截
    get(target, key, receiver) {
      console.log(`获取${key}：${Reflect.get(target, key, receiver)}`)
      // Reflect用于执行对象默认操作，Proxy的方法它都有对应方法
      // Reflect更规范、更友好
      // http://es6.ruanyifeng.com/#docs/reflect
      const val = Reflect.get(target, key, receiver)
      // 若val为对象则定义代理
      return typeof val === 'object' ? reactive(val) : val
    },
    // 新增、更新拦截
    set(target, key, value, receiver) {
      console.log(`设置${key}为：${value}`)
      return Reflect.set(target, key, value, receiver)
    },
    // 删除属性拦截
    deleteProperty(target, key) {
      console.log(`删除${key}`)
      return Reflect.deleteProperty(target, key)
    }
  })
  return observed
}
```

测试代码

```
const data = {
  foo: 'foo',
  obj: {a:1},
  arr: [1,2,3]
}
const react = reactive(data)
// get
react.foo
react.obj.a
react.arr[0]
```

```
// set
react.foo = 'fooooo'
react.obj.a = 10
react.arr[0] = 100

// add
react.bar = 'bar'
react.obj.b = 10
react.arr.push(4)
react.arr[4] = '5'

// delete
delete react.bar
delete react.obj.b
react.arr.splice(4, 1)
delete react.arr[3]
```

vue3展望

vue3适合我吗？会迅速取代vue2吗？我从以下点出发给出个人看法

- 升级是否平滑？

  Vue3会兼容之前写法，仅Portal、Suspense等少量新api需要看看，Composition API则是可选的

- 相关生态是否跟上？

  正式版发布还有一段时间，相关工具、生态、库都跟上需要时间，vue3也许明年不会有大需求

- vue3比vue2好吗？
  - 杀手级特性：Composition API
  - 用户体验：响应式革新、time-slicing
  - 更好的类型推断支持
  - 兼容性