# Vue3

# 目标

* 掌握composition Api的书写格式

# 一、安装

## 1.1、介绍

官网：https://v3.cn.vuejs.org/

npm install vue@next

# 实例对比

## 2.1、创建实例

//vue2创建实例

new Vue({

el: '#root',

data: {

...

},

components: {

...

}

})

//vue3创建实例

const { createApp } = vue

const app=createApp({

data: {

...

},

components: {

...

}

})

app.mount('#app');

## 2.2、实例图

Vue3实例图:

https://cn.vuejs.org/assets/lifecycle.16e4c08e.png

Vue2实例图:  
https://v2.cn.vuejs.org/images/lifecycle.png

从实例图上看,

创建实例后挂载到页面上时，从替换变成了追加

销毁时的生命周期函数名称做了改变。

## 2.3、属性和方法

Vue 没有内置支持防抖和节流，但可以使用 [Lodash](https://lodash.com/) 等库来实现。

<script src="https://unpkg.com/lodash@4.17.20/lodash.min.js"></script>  
<script>  
  Vue.createApp({  
    methods: {  
      // 用 Lodash 的防抖函数  
      click: \_.debounce(function() {  
        // ... 响应点击 ...  
     }, 500)  
   }  
 }).mount('#app')  
</script>

## 2.4、条件渲染和列表渲染

vue3:当v-if与v-for一起使用时，v-if具有比 v-for 更高的优先级。

Vue2:当 v-if 与 v-for一起使用时，v-for 具有比 v-if 更高的优先级。

//vue2

<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Document</title>  
  <script src="lib/vue.js"></script>  
</head>  
<body>  
  <div id="root">  
    <ul>  
      <li v-for="item of list" :key="item" v-if="false">  
       {{ item }}  
      </li>  
    </ul>  
  </div>  
</body>  
<script>  
new Vue({  
    el: '#root',  
    data: {  
      list: ['a', 'b', 'c', 'd']  
   }  
 })  
</script>  
</html>

//vue3

<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Document</title>  
  <script src="lib/vue.global.js"></script>  
</head>  
<body>  
  <div id="root">  
    <ul>  
      <li v-for="item of list" :key="item" v-if="false">  
       {{ item }}  
      </li>  
    </ul>  
  </div>  
</body>  
<script>  
  const { createApp } = window.Vue  
​  
  createApp({  
    data () {  
      return {  
        list: ['a', 'b', 'c', 'd']  
     }  
   }  
 }).mount('#root')  
</script>  
</html>

如何确保哪个的优先级高，审查元素查看效果

## 2.5、父组件给子组件传值

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="root">

<my-title :title="title"></my-title>

<my-title></my-title>

</div>

</body>

<script>

/\*\*

1.父组件在调用子组件的地方，添加自定义的属性, 属性的值就是传递给子组件的值

如果属性的值是变量，boolean,number类型，对象，数组，null，undefined，

需要使用绑定属性

\*/

// 2.在子组件定义的地方，添加一个props选项，有三种写法

// 2.1 props 的值为数组，数组的元素为 之前定义的属性名

// 2.2 props 的值为对象，key值为自定义的属性名，value值为数据类型

// 2.3 props 的值为对象，key值为自定义的属性名，value值又为一个对象

// 该对象的 type 属性可以表示 数据类型

// 该对象的 default 属性可以表示 属性的默认值（父组件在调用子组件时可以不传递）--- 如果属性值是对象或者数组，default写为函数，返回相应的值

// 该对象的 required 属性表示该属性是必须传递的，即使 使用了 default 也要传递数据

// 该对象的 validator 属性可以实现自定义验证函数, 如果返回值为false，会弹出警告

// 在子组件的模版中可以通过 自定义的属性名 渲染数据

const { createApp } = window.Vue

const myTitle = {

// props: ['title'],

// props: {

// title: String

// },

props: {

title: {

type: String,

default: '10000',

validator (val) {

return val === '100'

}

}

},

template: <h1>hello- {{ title }}</h1>

}

const app = createApp({

components: {

myTitle

},

data () {

return {

title: '标题'

}

}

})

const vm = app.mount('#root')

</script>

</html>

# 组合式Api

## 3.1、介绍

组合式api也叫Composition api。是vue3中的核心内容，它相对于vue2中的选项式api(Option api),提供更高性能的一种写法。

非官网地址：<https://vue3js.cn/vue-composition-api/>

官网地址：

https://cn.vuejs.org/guide/extras/composition-api-faq.html

## 3.2、setup

setup 函数是一个新的组件选项。作为在组件内使用 Composition API 的入口点。

## 3.3、调用时机

创建应用实例，然后初始化props，紧接着就调用setup函数。从生命周期钩子的视角来看，它会在beforeCreate钩子之前被调用

vue3 options 版本生命周期钩子使用

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="root"></div>

</body>

<script>

const { createApp } = window.Vue

const app = createApp({

beforeCreate () {

console.log('beforeCreate')

},

created () {

console.log('created')

},

setup () {

console.log('setup')

},

beforeMount() {

console.log('beforeMount')

},

mounted () {

console.log('mounted')

}

})

app.mount('#root')

</script>

</html>

\*\*与 2.x 版本生命周期相对应的组合式 API\*\*

beforeCreate -> 废弃，使用 setup()

created -> 废弃，使用 setup()

beforeMount -> onBeforeMount

mounted -> onMounted

beforeUpdate -> onBeforeUpdate

updated -> onUpdated

beforeDestroy -> onBeforeUnmount 替换

destroyed -> onUnmounted 替换

errorCaptured -> onErrorCaptured

## 3.4、体验vue3

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="app">

{{ count }} -- {{ obj.a }}

<button @click="add">加</button>

</div>

</body>

<script>

const { createApp, ref, reactive, onMounted } = Vue

const app = createApp({

setup () {

// ref 定义的初始值 需要 通过value 修改

// reactive 定义的初始值 不需要

const count = ref(10)

const obj = reactive({ // 创建对象

a: 1

})

const add = () => {

console.log(count)

count.value += 1

obj.a += 1

}

onMounted(() => {

console.log('onMounted')

})

return {

count,

obj,

add

}

}

})

app.mount('#app');

</script>

</html>

3.5、watchEffect

修改标题,监听数据的变化

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="app">

{{ count }} -- {{ obj.a }}

<button @click="add">加</button>

</div>

</body>

<script>

const { createApp, ref, reactive, onMounted, watchEffect } = Vue

const app = createApp({

setup () {

// ref 定义的初始值 需要 通过value 修改

// reactive 定义的初始值 不需要

const count = ref(10)

const obj = reactive({ // 创建对象

a: 1

})

const add = () => {

console.log(count)

count.value += 1

obj.a += 1

}

onMounted(() => {

console.log('onMounted')

})

watchEffect(() => {

document.title = count.value

})

return {

count,

obj,

add

}

}

})

app.mount('#app');

</script>

</html>

抽离

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="app">

{{ count }}

<button @click="add">加</button>

</div>

</body>

<script type="module">

import useCount from './16 useCount.js'

import useTitle from './16 useTitle.js'

const { createApp } = Vue

const app = createApp({

setup () {

const { count, add } = useCount()

useTitle(count)

return {

count,

add

}

}

})

app.mount('#app');

</script>

</html>

useCount.js

const { ref } = Vue

const useCount = () => {

const count = ref(10)

const add = () => {

count.value += 1

}

return {

count,

add

}

}

export default useCount

useTitle.js

const {watchEffect} = Vue

const useTitle = (count) => {

watchEffect(() => {

document.title = count.value

})

}

export default useTitle

## 3.5、父子组件传值

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="app">

<my-header title="hello vue3"></my-header>

</div>

</body>

<script>

const { createApp, h } = Vue

const Header = {

// props: ['title'],

props: {

title: String

},

template: `

<header>{{ title }}</header>

`

}

const app = createApp({

components: {

MyHeader: Header

},

setup(props) {

}

})

app.mount('#app')

</script>

</html>

子组件给父组件传值使用。context.emit

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="app">

<my-header title="hello vue3" @my-event="getData"></my-header>

</div>

</body>

<script>

const { createApp, h } = Vue

const Header = {

// props: ['title'],

props: {

title: String

},

template: `

<header>{{ title }}

<button @click="sendData">传值给父组件</button>

</header>

`,

setup(props, context) {

console.log(props)

console.log(context)

const sendData = () => {

context.emit('my-event', 1000)

}

return {

sendData

}

}

}

const app = createApp({

components: {

MyHeader: Header

},

setup(props) {

const getData = (val) => {

console.log(val)

}

return {

getData

}

}

})

app.mount('#app')

</script>

</html>

## 3.6、reactive

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="app">

{{ count }} - {{ data.x }} - {{ data.y }}

</div>

</body>

<script>

const { createApp, ref, reactive, onMounted } = Vue

const app = createApp({

setup(props) {

const count = ref(100)

const data = reactive({

x: 1,

y: 2

})

onMounted(() => {

count.value = 1000

data.x = 11

data.y = 22

})

return {

count,

data

}

}

})

app.mount('#app')

</script>

</html>

## 3.7、watch和watchEffect

//侦听器数据源可以是一个具有返回值的 getter 函数，也可以直接是一个 ref：

// 侦听一个 getter

const state = reactive({ count: 0 })

watch(

() => state.count,

(count, prevCount) => {

/\* ... \*/

}

)

// 直接侦听一个 ref

const count = ref(0)

watch(count, (count, prevCount) => {

/\* ... \*/

})

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="app">

{{ count }} -- {{ obj.a }}

<button @click="add">加</button>

<button @click="clear">清除watchEffect</button>

</div>

</body>

<script>

const { createApp, ref, reactive, onMounted, watchEffect } = Vue

const app = createApp({

setup () {

// ref 定义的初始值 需要 通过value 修改

// reactive 定义的初始值 不需要

const count = ref(10)

const obj = reactive({ // 创建对象

a: 1

})

const add = () => {

console.log(count)

count.value += 1

obj.a += 1

}

onMounted(() => {

console.log('onMounted')

})

const stop = watchEffect(() => {

document.title = count.value

})

const clear = () => {

stop()

}

return {

count,

obj,

add,

clear

}

}

})

app.mount('#app');

</script>

</html>

多个监听

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="app">

{{ count }} -- {{ obj.a }}

<button @click="add">加</button>

<button @click="clear">清除watchEffect</button>

</div>

</body>

<script>

const { createApp, ref, reactive, onMounted, watchEffect, watch } = Vue

const app = createApp({

setup () {

// ref 定义的初始值 需要 通过value 修改

// reactive 定义的初始值 不需要

const count = ref(10)

const obj = reactive({ // 创建对象

a: 1

})

const add = () => {

console.log(count)

count.value += 1

obj.a += 1

}

onMounted(() => {

console.log('onMounted')

})

// watch(() => {

// return obj.a

// }, (newVal, oldVal) => {

// console.log(newVal, oldVal)

// })

// watch(() => {

// return count.value

// }, (newVal, oldVal) => {

// console.log(newVal, oldVal)

// })

watch([

() => obj.a,

() => count.value

], ([newObja, newCountValue], [oldObja, oldCountValue]) => {

console.log(newObja, oldObja)

console.log(newCountValue, oldCountValue)

})

const clear = () => {

stop()

}

return {

count,

obj,

add,

clear

}

}

})

app.mount('#app');

</script>

</html>

## 3.8、模板refs

ref的两种用途

1. 响应式
2. 找DOM节点

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="app">

{{ count }}

<button ref="btnRef">按钮</button>

<my-com ref="comRef"></my-com>

<button @click="getData">获取信息</button>

</div>

</body>

<script>

const { createApp, ref } = Vue

const Com = {

setup () {

const msg = ref('hello world')

const fn = () => {

console.log('child')

}

return {

msg,

fn

}

}

}

const app = createApp({

components: {

'my-com': Com

},

setup () {

// ref 定义的初始值

const count = ref(10)

// ref 可以DOM操作

const btnRef = ref(null)

// ref 获取子组件的实例

const comRef = ref(null)

const getData = () => {

console.log(btnRef.value)

console.log(comRef.value)

}

return {

count,

btnRef,

comRef,

getData

}

}

})

app.mount('#app');

</script>

</html>

## 3.9、teleport

无teleport时:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

<style>

.modal {

position: fixed;

top: 0;

right: 0;

bottom: 0;

left: 0;

background-color: rgba(0, 0, 0, 0.5);

display: flex;

justify-content: center;

align-items: center;

}

</style>

</head>

<body>

<div id="root" style="position: relative;">

<h3>Vue3中teleport</h3>

<div>

<modal-button></modal-button>

</div>

</div>

</body>

<script>

const { createApp, ref } = window.Vue

const ModalButton = {

template: `

<button @click="modalOpen = true">

打开全屏的模态框

</button>

<div v-if="modalOpen" class="modal">

<div>

这是一个模态框

<button @click="modalOpen = false">

关闭

</button>

</div>

</div>

`,

setup(props) {

const modalOpen = ref(false)

return {

modalOpen

}

}

}

const app = createApp({

components: {

ModalButton

}

})

app.mount('#root')

</script>

</html>

使用teleport

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

<style>

.modal {

position: fixed;

top: 0;

right: 0;

bottom: 0;

left: 0;

background-color: rgba(0, 0, 0, 0.5);

display: flex;

justify-content: center;

align-items: center;

}

</style>

</head>

<body>

<div id="root" style="position: relative;">

<h3>Vue3中teleport</h3>

<div>

<modal-button></modal-button>

</div>

</div>

</body>

<script>

const { createApp, ref } = window.Vue

const ModalButton = {

template: `

<button @click="modalOpen = true">

打开全屏的模态框

</button>

<teleport to="body"> //to属性指定父标签

<div v-if="modalOpen" class="modal">

<div>

这是一个模态框

<button @click="modalOpen = false">

关闭

</button>

</div>

</div>

</teleport>

`,

setup(props) {

const modalOpen = ref(false)

return {

modalOpen

}

}

}

const app = createApp({

components: {

ModalButton

}

})

app.mount('#root')

</script>

</html>

## 3.10、计算属性

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="lib/vue.global.js"></script>

</head>

<body>

<div id="app">

{{ count }} -- {{ double }}

<button @click="add">加</button>

</div>

</body>

<script>

const { createApp, ref, computed } = Vue

const app = createApp({

setup () {

// ref 定义的初始值 需要 通过value 修改

// reactive 定义的初始值 不需要

const count = ref(10)

const add = () => {

count.value += 1

}

const double = computed(() => count.value \* 2)

return {

count,

add,

double

}

}

})

app.mount('#app');

</script>

</html>

## 3.11、toRefs

<template>

<div>

<h1>Example Page{{name}}</h1>

</div>

</template>

<script>

import {reactive,toRefs} from 'vue';

export default {

/\*

toRefs:

将响应式对象中所有属性包装为ref对象,并返回包含这些ref对象的普通对象

应用:当从合成函数返回响应式对象时,toRefs 非常有用,

这样消费组件就可以在不丢失响应式的情况下对返回函数的对象进行分解使用

\*/

setup(){

const obj=reactive({

name:"张三",

age:20

})

console.log('obj',obj);

let resObj=toRefs(obj);

console.log('resObj',resObj);

return {...resObj}

}

}

</script>

# Proxy

<https://v3.cn.vuejs.org/guide/reactivity.html#vue-%E5%A6%82%E4%BD%95%E8%B7%9F%E8%B8%AA%E5%8F%98%E5%8C%96>

Vue2 数据劫持 Object.defineProperty

Proxy 是一个对象，它包装了另一个对象，并允许你拦截对该对象的任何交互。