VueUse 最佳实践

我们在维护 VueUse 两年以来所积累的 Vue 组合式 API 的模式与最佳实践

ANTHONY FU

VUE·CONF

2022/12/10

Anthony Fu

Vue, Vite, Nuxt 核心团队成员

VueUse, Vitest, UnoCSS, Slidev 等开源项目作者

任职于 NuxtLabs

- antfu
- antfu7
- antfu@webtoo.ls
- antfu.me

Vue 组合式 API 工具合集

v9.6.0 | 1.7M/month | docs & demos | 251 functions

Stars 13k

同时支持 Vue 2 和 3

Tree-shake

TypeScript

支持 CDN

丰富的生态系统

Functions

Core State Elements Browser Sensors Network Animation Component Watch Reactivity Array Time Utilities

Add-ons Electron Firebase Head Integrations Math Motion Router RxJS SchemaOrg Sound

Sort by Category Name Updated

Filters Has Component Has Directive

Search...

State

createGlobalState - keep states in the global scope to be reusable across Vue instances

createInjectionState - create global state that can be injected into components

createSharedComposable - make a composable function usable with multiple Vue instances

useAsyncState - reactive async state

useDebouncedRefHistory - shorthand for `useRefHistory` with debounced filter

useLastChanged - records the timestamp of the last change

useLocalStorage - reactive LocalStorage

useManualRefHistory - manually track the change history of a ref when the using calls `commit()`

useRefHistory - track the change history of a ref

useSessionStorage - reactive SessionStorage

useStorage - reactive LocalStorage/SessionStorage

useStorageAsync - reactive Storage in with async support

State of VueUse

截至 2022/12/01

140万	NPM 月下载量
43万	文档月访问量
4.9万	开源项目使用者
1.3万	Stars on GitHub
1100天	项目诞生
364	贡献者
251	组合式工具
13	团队成员
10	生态插件

我们从中学到了什么？

建立"链接"

建立"链接"

Vue 与组合式 API

- 状态驱动 UI - single source of truth
- 状态的改变会**自动**更新 UI - 响应式
- 通过 `<template>` 模版，我们建立了**状态**和 **UI** 的链接
- 通过 `setup()` 函数，我们建立了**状态**和**逻辑**的链接

将 Ref 作为参数传递

编写组合式函数

实现

使用范例

简单函数

```ts
function add(a: number, b: number) {
  return a + b
}
```

```ts
let a = 1
let b = 2

let c = add(a, b) // 3
```

接受 Ref，
返回响应式的结果

```ts
function add(a: Ref<number>, b: Ref<number>) {
  return computed(() => a.value + b.value)
}
```

```ts
const a = ref(1)
const b = ref(2)

const c = add(a, b)
c.value // 3
```

同时接受 Ref 和纯值

```ts
function add(
  a: Ref<number> | number,
  b: Ref<number> | number
) {
  return computed(() => unref(a) + unref(b))
}
```

```ts
const a = ref(1)

const c = add(a, 5)
c.value // 6
```

`ref` 的内部实现

Vue 源码分析

```
function ref(input) {
  return isRef(input)
    ? input
    : createRef(input)
}
```

也就是说：

```
const foo = ref(123)
const bar = ref(foo)

foo === bar // true
```

> 窍门
>
> `ref()` 会直接返回已经是 Ref 的值

`unref` 的内部实现

Vue 源码分析

```
function unref(input) {
  return isRef(input)
    ? input.value
    : input
}
```

也就是说：

```
const foo = unref(123)

foo === 123 // true
```

> 窍门
>
> `unref()` 会直接返回非 Ref 的值

MaybeRef

类型工具

```ts
type MaybeRef<T> = Ref<T> | T
```

- 需要使用 **值** 时，使用 `unref()` 获取
- 需要使用 **Ref** 时，使用 `ref()` 获取

例如：

普通函数

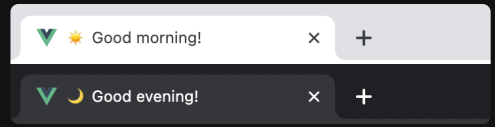
```ts
export function getTimeAgo(time: Date | number | string) {
  return format(time)
}
```

响应式函数

```ts
export function useTimeAgo(time: MaybeRef<Date | number | string>) {
  return computed(() => format(unref(time)))
}
```

例子

根据 暗色/亮色 模式，动态设置标题



一般用法

```ts
import { useDark, useTitle } from '@vueuse/core'
```

```ts
const isDark = useDark()
const title = useTitle()

watch(isDark, () => {
  title.value = isDark.value ? '🌙 Good evening!' : '☀ Good morning!'
})
```

“链接”用法

```ts
const isDark = useDark()
const title = computed(() => isDark.value ? '🌙 Good evening!' : '☀ Good morning!')

useTitle(title)
```

在 VueUse 中查看：useTitle

进一步深入

使其变得更加灵活

- `computed()` 可以接受一个函数，返回一个 `Ref` 对象
- 我们接受响应式的 `Ref` 对象作为参数

在 VueUse 9.0 中，我们引入了一个新的模式：

```
1  const isDark = useDark()
2  const title = computed(() => isDark.value ? '🌙 晚上好!' : '☀ 早上好!')
3
4  useTitle(title)
```

变为：

```
1  const isDark = useDark()
2
3  useTitle(() => isDark.value ? '🌙 晚上好!' : '☀ 早上好!')
```

我们称之为 **"Reactive Getter" 响应式的获取器**

`MaybeComputedRef`

```ts
/**
 * 可能是一个 Ref，或者一个字面值
 */
export type MaybeRef<T> = T | Ref<T>

/**
 * 可能是一个 Ref，一个字面值，或者一个 Getter 函数
 */
export type MaybeComputedRef<T> = MaybeRef<T> | (() => T) | ComputedRef<T>
```

`resolveRef`

```ts
function resolveRef<T>(input: MaybeRef<T>): Ref<T> {
  return typeof input === 'function'
    ? computed(input)
    : ref(input)
}
```

如果传入的是一个函数，我们使用 `computed()` 创建一个 `Ref` 对象，否则交由 `ref()` 处理

`resolveUnref`

```ts
function resolveUnref<T>(input: MaybeRef<T>): T {
  return typeof input === 'function'
    ? input()
    : unref(input)
}
```

如果传入的是一个函数，我们直接呼叫以取值，否则交由 `ref()` 处理

响应性语法糖

详情请见文档 https://vuejs.org/guide/extras/reactivity-transform

```
let count = $ref(0) // count 在类型上是 number 值
count = 1 // 可以直接赋值
console.log(count)
```

```
let count = ref(0)
count.value = 1
console.log(count.value)
```

使用限制

```
watch(
  count, // !! 这会导致响应式丢失
  () => {}
)
```

```
watch(
  () => count, // 应该使用一个 Getter 函数
  () => {}
)
```

结合响应性语法糖

在 VueUse 中，结合 Reactive Getter，我们可以：

```
let count = $ref(0)

useTitle(() => `Count: ${count}`)

const storage = useStorage('count', () => count)
```

Reactify 响应式化

使用魔法构造“链接”

VueUse 提供了工具函数 `reactify()`，可以将一个普通函数转换为一个响应式的函数

```ts
import { reactify } from '@vueuse/core'

function getTimeAgo(time: Date | number | string) {
  return format(time)
}

const useTimeAgo = reactify(getTimeAgo)
```

自动 unref 入参，并将返回值包装为 computed

```ts
const date = ref(new Date())

const timeago1 = useTimeAgo(date) // Computed<string>

const timeago2 = useTimeAgo(() => Date.now() - 1000) // Computed<string>
```

在 VueUse 中查看：reactify

副作用清理

`watch` 的自动清理

以及 `watchEffect` `computed` 等内置函数

```vue
<script setup>
import { watch, ref } from 'vue'

const count = ref(0)

watch(count, () => {
  console.log('Count: ' + count.value)
})
</script>
```

> 窍门
>
> 当组件被销毁时，`watch()` 会自动被移除。

自己编写时 Composables 的副作用清理

```
export function useEventListener(name, handler) {
  window.addEventListener(name, handler)

  onUnmounted(() => {
    window.removeEventListener(name, handler)
  })
}
```

窍门

使用 `onUnmounted()` 来注册清理函数

手动清理的能力

`watch()` 会返回一个清理函数，以便手动清理

```
const count = ref(0)
const stop = watch(count, () => {
  console.log('Count: ' + count.value)
})

count.value += 1 // Count: 1
stop()
count.value += 1 // 无输出
```

手动清理的能力

在编写 Composables 时

```
export function useEventListener(name, handler) {
  window.addEventListener(name, handler)

  const cleanup = () => {
    window.removeEventListener(name, handler)
  }

  onUnmounted(cleanup)

  return cleanup
}
```

使用方法：

```
const stop = useEventListener('mousedown', () => {})

stop() // 移除事件监听
```

但是...

有时候会有点麻烦...

例如：

```
function useMouse() {
  const stop1 = useEventListener('mousedown', () => {})
  const stop2 = useEventListener('mouseup', () => {})
  const stop3 = useEventListener('mousemove', () => {})

  const cleanup = () => {
    stop1()
    stop2()
    stop3()
  }

  return cleanup
}
```

Effect 作用域（Effect Scope）

在 Vue 3.1 中引入

```
import { effectScope } from 'vue'

const scope = effectScope()
scope.run(() => {
  const count = ref(0)
  const doubled = computed(() => counter.value * 2)

  watch(doubled, () => console.log(doubled.value))

  useEventListener('mousedown', () => {})
  useEventListener('mouseup', () => {})
  useEventListener('mousemove', () => {})
})

// to dispose all effects in the scope
scope.stop()
```

详情请见文档：https://vuejs.org/api/reactivity-advanced.html#effectscope

`onScopeDispose`

让 Composables 更好地与 `effectScope` 一起工作，可以将 `onUnmounted` 替换为 `onScopeDispose`：

```
export function useEventListener(name, handler) {
  window.addEventListener(name, handler)

- onUnmounted(() => {
+ onScopeDispose(() => {
    window.removeEventListener(name, handler)
  })
}
```

这样可以在作用域被销毁时调用清理函数。

> 窍门
>
> - 一个组件可以被理解为一个特殊的 **作用域**
> - `onUnmounted` 可以被理解为一个特殊 `onScopeDispose`

Recap

- 使用 `ref()` 和 `unref()` 来创建和解引用响应式，以建立连接
- 使用 `resolveRef()` 和 `resolveUnref()` 来扩张灵活性，更好的配合响应式语法糖
- 使用 `reactify()` 来将一般函数转换为响应式函数
- 使用 `onScopeDispose()` 来注册清理函数

VueUse 全家桶

@vueuse/core

VueUse 核心函数

@vueuse/shared

VueUse 内部共享函数

@vueuse/components

Renderless 组件包装

@vueuse/math

响应式的数学工具

@vueuse/head

网页头信息操作

@egoist

@vueuse/sound

声音播放相关的组合式函数

@Tahul

@vueuse/motion

动画相关的组合式函数

@Tahul

@vueuse/guesture

交互相关的组合式函数

@Tahul

@vueuse/schema-org

Schema.org 结构生成工具

@harlan-zw

@vueuse/integrations

常用第三方库的组合式整合

@vueuse/firebase

Firebase 整合

@vueuse/rxjs

RxJS 整合

VueUse
Collection of Vue Composition Utilities

Collection of Essential Vue Composition Utilities

Get Started | Functions | Add-ons | View on GitHub

Feature Rich
200+ functions for you to choose from

Seamless migration
Works for both Vue 3 and 2

Fully tree shakeable
Only take what you want

Type Strong
Written in TypeScript, with full TS docs

Flexible
Passing refs as arguments, fully customizable, configurable event filters and targets

No bundler required
Usable via CDN, without any bundlers

请查阅 vueuse.org 了解更多

Special Sponsor

NuxtLabs

Platinum Sponsors


Evan You

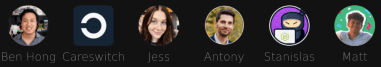
Astro

Gold Sponsors

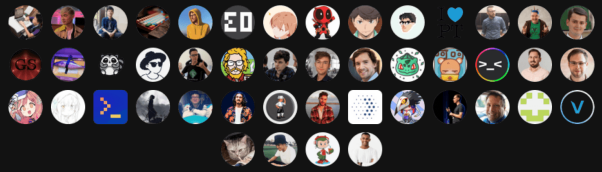
Vue Mastery

Vercel Community

Silver Sponsors

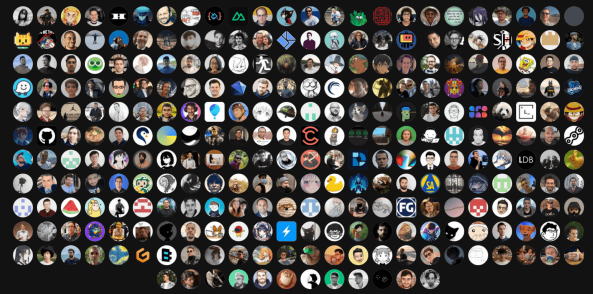

Ben Hong Careswitch Jess Antony Stanislas Matt

Sponsors



Backers

Past Sponsors



♥ Sponsor me at GitHub

谢谢！

幻灯片可在 antfu.me 查看与下载