# **I. Vue 3 có một số tính năng mới đặc biệt so với Vue 2, bao gồm:**

## 1. Cấu trúc runtime mới: Vue 3 đã cải tiến cấu trúc runtime của mình để tăng tốc độ và hiệu suất.



- Sự khác biệt là ở Vue 3 thay vì import cả thư viện Vue thì giờ ta chỉ import duy nhất module createApp để phục vụ mục đích hiện tại là tạo app Vue

- Đây là 1 trong những thay đổi lớn ở Vue 3. Kĩ thuật này gọi là tree shaking - loại bỏ những phần code thừa không dùng tới khi build và tối ưu code. Điều này dẫn tới app của chúng ta sẽ nhỏ đi đáng kể

- Ở Vue 3 hầu hết tất cả các module đều được thiết kế để có thể tree-shake (v-model, nextTick, watch,...), dùng tới module nào ta chỉ cần import module đó (hoặc Vue tự detect và import lúc build thay ta). Nhờ đó ứng dụng của chúng ta sẽ có size giảm đi đáng kể.

## 2. Tích hợp TypeScript: Vue 3 đã tích hợp TypeScript mặc định, cho phép bạn sử dụng TypeScript trong các component và các project của mình.

## 3. Fragments (hay còn gọi là multiple root nodes): Vue 3 cung cấp tính năng fragments cho phép bạn sắp xếp các element trong một component mà không cần phải sử dụng một element chứa.

<template>

<img src="./logo.png" class=”$arrt.class”>

<h1>Hello Vue 3!</h1>

<button @click="inc">Clicked {{ count }} times.</button>

<template>

// div Fragments

</template>

</template>

## 4. Composition API: Một API mới để xử lý logic cho các component với cách tính toán tốt hơn.

- Options API chúng ta sẽ phải phân tán code khắp nơi đúng không. Từ các lifecycle hooks đến computed hay watcher. Khi sửa thì phải kéo lên kéo xuống, kéo từ đầu file tới cuối file thực sự rất mệt mỏi (ví dụ khi đang sửa mouted mà phải sửa lại data thì kéo mệt nghỉ luôn). Việc phân tán này làm cho người đến sau tương đối mất thời gian để sửa code, hiểu logic.

- Tóm lại, API Thành phần và API Tùy chọn đều là những cách hợp lệ để xác định logic của thành phần Vue. API Thành phần cung cấp một cách chức năng và có thể tái sử dụng để tổ chức mã của bạn, trong khi API Tùy chọn cung cấp một cách tiếp cận hướng đối tượng truyền thống.

- Các lợi ích khác đối với Composition API :

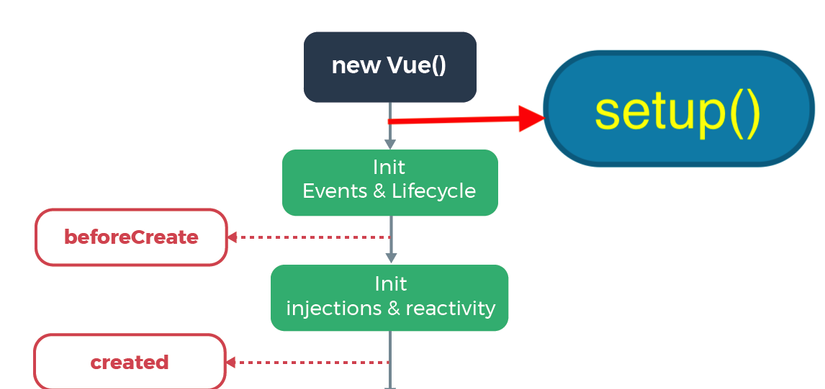
+ Thay thế mixins và tất cả các vấn đề như xung đột tên đi kèm với chúng.

+ Hỗ trợ kiểu tốt hơn , vì nó sử dụng hầu hết các hàm và biến thông thường, nên TypeScript không yêu cầu kiểu gõ phức tạp.

+ Các tệp nhỏ hơn - như đã đề cập, API Thành phần yêu cầu ít mã hơn.

- Composition API sẽ được sử dụng cùng với những gì đã có ở Vue 2 hiện tại

+ setup sẽ được chạy và trả về giá trị trước hết tất cả các các hooks khác như beforeCreate, created,....



Mọi thứ trong setup tồn tại ở 1 thế giới riêng và chỉ những thứ trả về của setup mới được truy cập từ thế giới bên ngoài.

- Cách sử dụng:



## 5. Proxy: Ở Vue 3 cơ chế xử lý reactive đã được thay thế hoàn toàn bằng [Proxy](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy).

Đối tượng Proxy ( Ủy quyền) cho phép bạn tạo một proxy cho một đối tượng khác, proxy này có thể chặn và xác định lại các hoạt động cơ bản cho đối tượng đó.

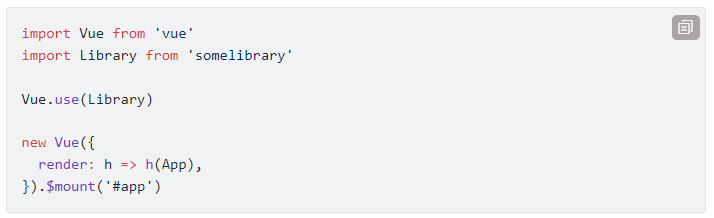
## 6. Lazy initialization: Vue 3 đã giới thiệu tính năng lazy initialization cho các component con để tăng tốc độ và hiệu suất của ứng dụng. (defineAsyncComponent)

## 7. Teleport(<Teleport to="body">): Tính năng teleport cho phép bạn chuyển các element giữa các component một cách dễ dàng.

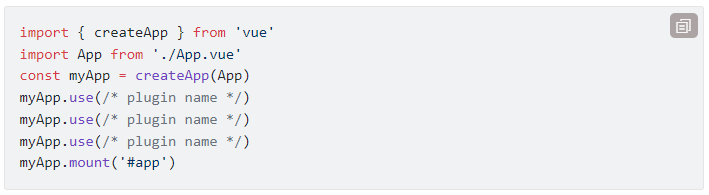
- Teleport hữu ích đó là việc tạo tooltip, modal, các thành phần con không bị ảnh hưởng css của thành phần cha.

## 8. Global Mounting

- Vue 2



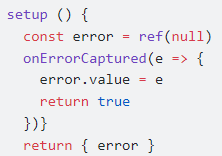
- Vue 3



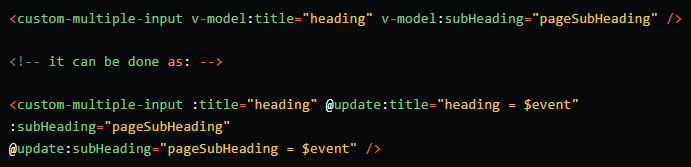
## 8. \*\*\*Tính năng Suspense(default và fallback): Vue 3 cung cấp tính năng Suspense để giúp bạn tải dữ liệu một cách bất đồng bộ và tránh tình trạng "lặp".



onErrorCaptured: Vue 3 cũng cung cấp cho ta 1 hook đó là onErrorCaptured để bắt lỗi nếu trong quá trình xử lý các tác vụ async trong setup() xảy ra lỗi

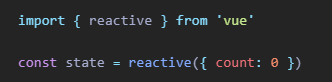


## 9. Multiple V-models



# **II. Thành phần thay đổi cơ bản**

## 1. Reactive:



- Các đối tượng phản ứng là các Proxy JavaScript và hoạt động giống như các đối tượng bình thường. Sự khác biệt là Vue có thể theo dõi quyền truy cập thuộc tính và các thay đổi của một đối tượng

- Reactive Proxy vs Original

const raw = {}

const proxy = reactive(raw)

console.log(proxy === raw) // false

console.log(reactive(raw) === proxy) // true

console.log(reactive(proxy) === proxy) // true

Quy tắc này cũng áp dụng cho các đối tượng lồng nhau. Do tính phản ứng sâu, các đối tượng lồng nhau bên trong một đối tượng phản ứng cũng là các proxy:

const proxy = reactive({})

const raw = {cout:0}

proxy.nested = raw

console.log(proxy.nested.cout === raw) // false

- Hạn chế:

+ Nó chỉ hoạt động đối với các loại đối tượng (đối tượng, mảng và các loại như Map và Set). Nó không thể chứa các kiểu nguyên thủy như string, number hoặc boolean.

+ Vì tính năng theo dõi phản ứng của Vue hoạt động trên quyền truy cập thuộc tính, nên chúng ta phải luôn giữ cùng một tham chiếu đến đối tượng phản ứng.  
 let state = reactive({ count: 0 })

state = reactive({ count: 1 })

Điều đó cũng có nghĩa là khi chúng ta gán hoặc hủy cấu trúc thuộc tính của đối tượng phản ứng thành các biến cục bộ hoặc khi chúng ta chuyển thuộc tính đó vào một hàm, chúng ta sẽ mất kết nối phản ứng

const state = reactive({ count: 0 })

let n = state.count

n++

let { count } = state

count++

callSomeFunction(state.count)

## 2. Ref

Để giải quyết các hạn chế của reactive(), Vue cũng cung cấp một [ref()](https://vuejs.org/api/reactivity-core.html#ref) chức năng cho phép chúng ta tạo các "refs" phản ứng có thể chứa bất kỳ loại giá trị nào

import { ref } from 'vue'

const count = ref(0)

count.value = null // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Truy cập giá trị của ref thông qua thuộc tính .value

console.log(count.value) // 0

count.value++

console.log(count.value) // 1

const objectRef = ref({ count: 0 })

objectRef.value = { count: 1 }

Lưu ý:

const object = { foo: ref(1) }

{{ object.foo }} // show 1 === {{ object.foo.value }}

Biểu thức sau sẽ KHÔNG hoạt động

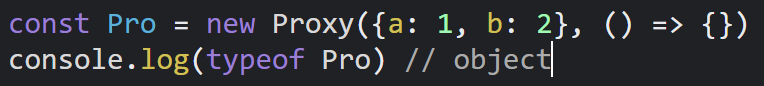
{{ object.foo + 1 }}

Hoạt động

const { foo } = object

{{ foo + 1 }}

## 3. Ref Unwrapping in Reactive Objects



const count = ref(0)

const state = reactive({

count

})

console.log(state.count) // 0

state.count = 1

console.log(count.value) // 1

const otherCount = ref(2)

state.count = otherCount

console.log(state.count) // 2

// original ref is now disconnected from state.count

console.log(count.value) // 1

## 4. Computed Properties

const author = reactive({

name: 'John Doe',

books: [

'Vue 2 - Advanced Guide',

'Vue 3 - Basic Guide',

'Vue 4 - The Mystery'

]

})

const publishedBooksMessage = computed(() => {

return author.books.length > 0 ? 'Yes' : 'No'

})

<p> {{ publishedBooksMessage }} </p>

<p> {{ publishedBooksMessage.value }} </p>

Computed Caching vs. Methods

<p>{{ calculateBooksMessage() }}</p>

function calculateBooksMessage() {

return author.books.length > 0 ? 'Yes' : 'No'

}

Computed tính toán và lưu vào bộ đệm, Func sẽ chạy lại khi phần phụ thuộc thay đổi

Writable Computed

// read-only property

const fullName = computed(() => {

  return `${firstName.value} ${lastName.value}`

})

const firstName = ref('John')

const lastName = ref('Doe')

const fullName = computed({

get() {

return firstName.value + ' ' + lastName.value

},

set(newValue) {

[firstName.value, lastName.value] = newValue.split(' ')

}

})

fullName.value = 'Sinh TV'

console.log(firstName.value) // Sinh

console.log(lastName.value) // TV

Lưu ý: Không nên thực hiện các yêu cầu không đồng bộ hoặc thay đổi DOM bên trong computed

## 5. Class and Style Bindings

- Examples:

const isActive = ref(true)

const hasError = ref(false)

<div

class="static"

:class="{ active: isActive, 'text-danger': hasError }"

></div>

const classObject = reactive({

active: true,

'text-danger': false

})

<div :class="classObject"></div>

or

const isActive = ref(true)

const error = ref(null)

const classObject = computed(() => ({

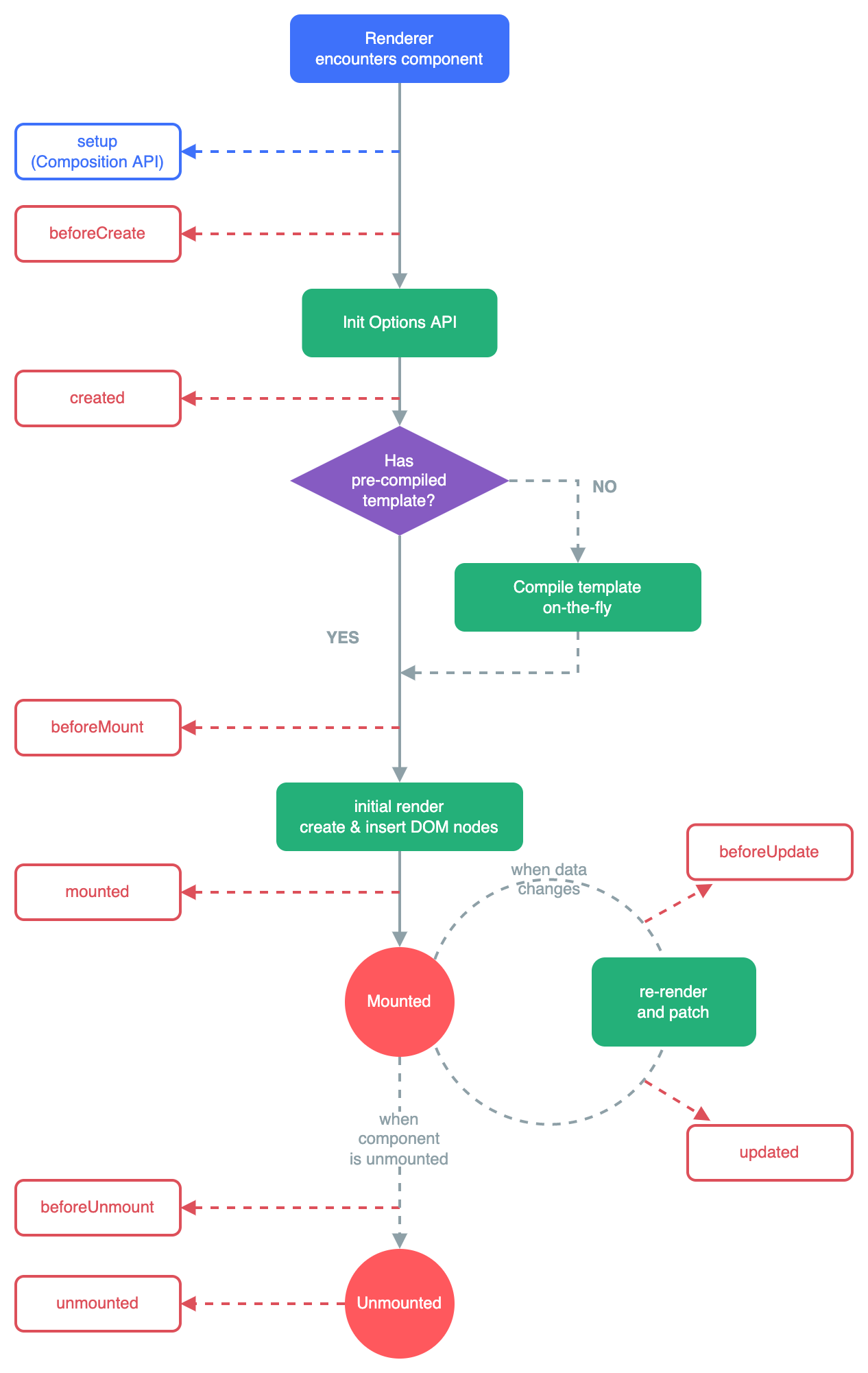
active: isActive.value && !error.value,

'text-danger': error.value && error.value.type === 'fatal'

}))



## 6. Lifecycle Hooks

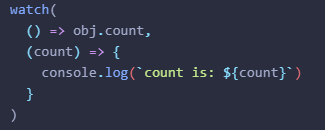


## 7. Watcher

- watch(): (computed refs, a reactive object, a getter function, or an array of multiple sources)







const obj = reactive({ count: 0 })

watch(obj, (newValue, oldValue) => {

// Note: `newValue` will be equal to `oldValue` here

// because they both point to the same object!

})

obj.count++

watch(

() => state.someObject,

(newValue, oldValue) => {

// Note: `newValue` will be equal to `oldValue` here

// \*unless\* state.someObject has been replaced

},

{ immediate: true, deep: true }

)

- watchEffect()

- Chạy ngay lập tức, không cần chỉ định immediate: true  
- Trong quá trình thực hiện, nó sẽ tự động theo dõi các phần phụ thuộc, không cần truyền tham số phụ thuộc.

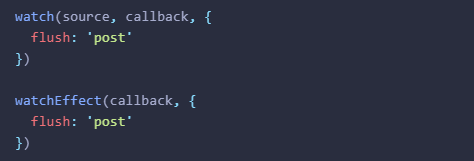
- watchEffect() sẽ loại bỏ gánh nặng phải duy trì danh sách phụ thuộc theo cách thủ công. Ngoài ra, nếu cần watch một số thuộc tính trong cấu trúc dữ liệu lồng nhau watchEffect() hiệu quả hơn trình theo dõi sâu, vì nó sẽ chỉ theo dõi các thuộc tính được sử dụng trong lệnh gọi lại, thay vì theo dõi đệ quy tất cả chúng.  
- watchEffect chỉ theo dõi các phụ thuộc trong quá trình thực thi đồng bộ của nó . Khi sử dụng nó với một cuộc gọi lại không đồng bộ, chỉ các thuộc tính được truy cập trước lần await đánh dấu đầu tiên mới được theo dõi.



- watch so với watchEffect

* watch chỉ theo dõi phần phụ thuộc rõ ràng. Nó sẽ không theo dõi bất cứ thứ gì được truy cập bên trong cuộc gọi lại. Ngoài ra, cuộc gọi lại chỉ kích hoạt khi phần phục thuộc đã thực sự thay đổi. watch tách biệt theo dõi phụ thuộc khỏi tác dụng phụ, giúp kiểm soát chính xác hơn thời điểm gọi lại sẽ kích hoạt.
* watchEffect tự động theo dõi mọi thuộc tính phản ứng được truy cập trong quá trình thực thi đồng bộ của nó. Điều này thuận tiện hơn và thường dẫn đến mã ngắn hơn, nhưng làm cho các phụ thuộc phản ứng của nó ít rõ ràng hơn.

- truy cập Dom tại watch  
Thuộc tính flush: 'post' cho phép watch truy cập dom tại thời điểm Vue đã cập nhật nó.



Ngoài ra có thể sử dụng hook watchPostEffect()

- Stopping a watcher

const unwatch = watchEffect(() => {})

// ...later, when no longer needed

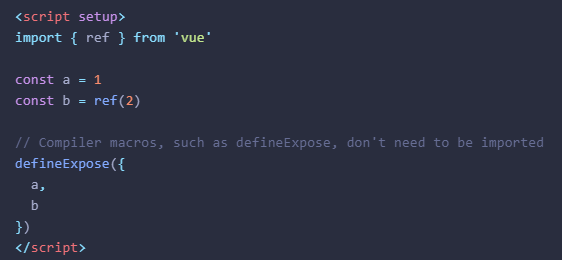
unwatch()

## 8. Template Refs

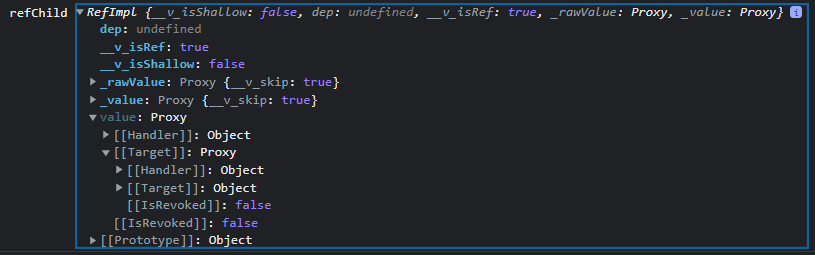
- Function Refs: ref thuộc tính cũng có thể được liên kết với một hàm, hàm này sẽ được gọi trong mỗi lần cập nhật thành phần và cung cấp cho bạn sự linh hoạt hoàn toàn về nơi lưu trữ tham chiếu phần tử



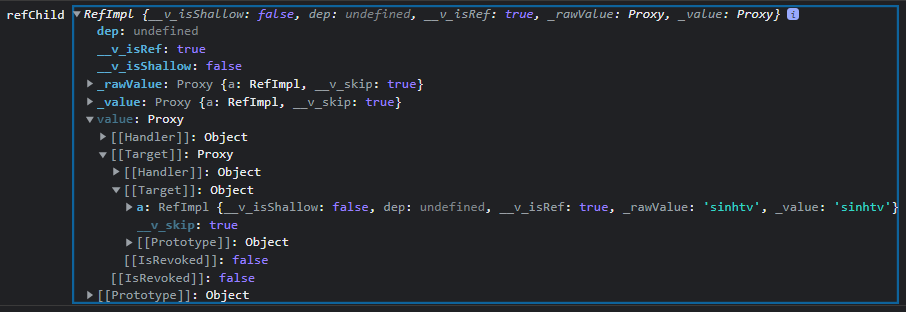
- Để truy cập các biến bên trong component con sử dụng <script setup> chúng ta dùng defineExpose



- Ref không sử dụng defineExpose



- Ref có sử dụng defineExpose



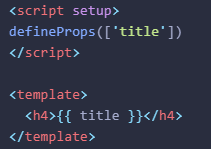
## 9. Components Basics

Tệp .vue được gọi là [Single-File Component](https://vuejs.org/guide/scaling-up/sfc.html) (SFC)

Trong SFC, bạn nên sử dụng PascalCase tên thẻ cho các thành phần con để phân biệt với các thành phần HTML gốc.



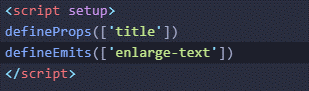
Passing Props

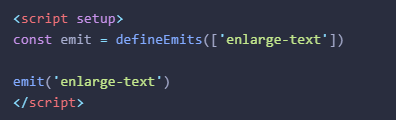


const props = defineProps(['title'])

console.log(props.title)

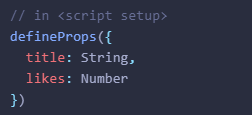
Listening to Events





# **III. Components In-Depth**

## Props



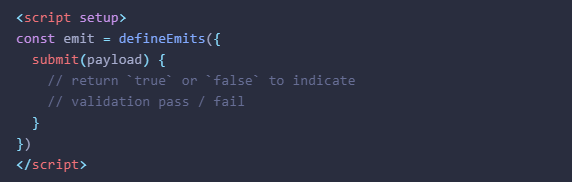
Quy ước: PascalCase cho các thẻ thành phần và kebab-case cho props



Bạn có thể sử dụng v-bind mà không cần đối số ( v-bind thay vì :prop-name). Ví dụ


## Emit



Lưu ý: $emit('someEvent') camelCase. <MyComponent @some-event.once="callback" /> kebab-case

## V-model





## Fallthrough Attributes

Vô hiệu hóa kế thừa thuộc tính: inheritAttrs: false

Sau đó có thể truy cập thông qua thuộc tính $attrs.

Đối $attrs tượng bao gồm tất cả các thuộc tính không được khai báo bởi thành phần props hoặc emits (ví dụ: class, style, v-on, v.v.).

onUpdated() Để xem sử dụng attrs mới nhất

## Slot

## Async components

