简介

ⓘ 你正在阅读的是 Vue 3 的文档！

Vue 2 中文文档已迁移至 v2.cn.vuejs.org。
想从 Vue 2 升级？请参考迁移指南。

在 VueMastery 上观看视频课程学习 Vue

什么是 Vue？

Vue (发音为 /vjuː/，类似 **view**) 是一款用于构建用户界面的 JavaScript 框架。它基于标准 HTML、CSS 和 JavaScript 构建，并提供了一套声明式的、组件化的编程模型，帮助你高效地开发用户界面。无论是简单还是复杂的界面，Vue 都可以胜任。

下面是一个最基本的示例：

```js
import { createApp } from 'vue'

createApp({
  data() {
    return {
      count: 0
    }
  }
}).mount('#app')
```

template

```
<div id="app">
  <button @click="count++">
    Count is: {{ count }}


  </button>
</div>
```

结果展示

Count is: 0

上面的示例展示了 Vue 的两个核心功能：

声明式渲染：Vue 基于标准 HTML 拓展了一套模板语法，使得我们可以声明式地描述最终输出的 HTML 和 JavaScript 状态之间的关系。

响应性：Vue 会自动跟踪 JavaScript 状态变化并在改变发生时响应式地更新 DOM。

你可能已经有了些疑问——先别急，后面的文档中我们会详细介绍每一个细节。现在，请继续看下去，以确保你对 Vue 作为一个框架到底提供了什么有一个宏观的了解。

> ⓘ 预备知识
>
> 文档接下来的部分假设你对 **HTML**、**CSS** 和 **JavaScript** 已经基本熟悉。如果你对前端开发完全陌生，最好不要直接在一开始针对一个框架进行学习——最好是掌握了基础知识再回到这里。你可以通过这篇 **JavaScript** 概述来检验你的 **JavaScript** 知识水平。如果之前有其他框架的经验会很有帮助，但不是必须的。

渐进式框架

Vue 是一个框架，也是一个生态。其功能覆盖了大部分前端开发常见的需求。但 Web 世界是十分多样化的，不同的开发者在 Web 上构建的东西可能在形式和规模上会有很大的不同。考虑到这一点，Vue 的设计非常注重灵活性和“可以被逐步集成”这个特点。根据你的需求场景，你可

以用不同的方式使用 Vue：

无需构建步骤，渐进式增强静态的 HTML

在任何页面中作为 Web Components 嵌入
单页应用 (SPA)
全栈 / 服务端渲染 (SSR)
Jamstack / 静态站点生成 (SSG)
开发桌面端、移动端、WebGL，甚至是命令行终端中的界面

如果你是初学者，可能会觉得这些概念有些复杂。别担心！理解教程和指南的内容只需要具备基础的 HTML 和 JavaScript 知识。你即使不是这些方面的专家，也能够跟上。

如果你是有经验的开发者，希望了解如何以最合适的方式在项目中引入 Vue，或者是对上述的这些概念感到好奇，我们在使用 Vue 的多种方式中讨论了有关它们的更多细节。

无论再怎么灵活，Vue 的核心知识在所有这些用例中都是通用的。即使你现在只是一个初学者，随着你的不断成长，到未来有能力实现更复杂的项目时，这一路上获得的知识依然会适用。如果你已经是一个老手，你可以根据实际场景来选择使用 Vue 的最佳方式，在各种场景下都可以保持同样的开发效率。这就是为什么我们将 Vue 称为“渐进式框架”：它是一个可以与你共同成长、适应你不同需求的框架。

单文件组件

在大多数启用了构建工具的 Vue 项目中，我们可以使用一种类似 HTML 格式的文件来书写 Vue 组件，它被称为**单文件组件** (也被称为 `*.vue` 文件，英文 Single-File Components，缩写为 **SFC**)。顾名思义，Vue 的单文件组件会将一个组件的逻辑 (JavaScript)，模板 (HTML) 和样式 (CSS) 封装在同一个文件里。下面我们将用单文件组件的格式重写上面的计数器示例：

vue

```
<script>
export default {
  data() {
    return {
      count: 0
    }
```

```
  }
}
</script>


<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

单文件组件是 Vue 的标志性功能。如果你的用例需要进行构建，我们推荐用它来编写 Vue 组件。你可以在后续相关章节里了解更多关于单文件组件的用法及用途。但你暂时只需要知道 Vue 会帮忙处理所有这些构建工具的配置就好。

API 风格

Vue 的组件可以按两种不同的风格书写：**选项式 API** 和**组合式 API**。

选项式 API (Options API)

使用选项式 API，我们可以用包含多个选项的对象来描述组件的逻辑，例如 `data`、`methods` 和 `mounted`。选项所定义的属性都会暴露在函数内部的 `this` 上，它会指向当前的组件实例。

vue

```
<script>
export default {
  // data() 返回的属性将会成为响应式的状态
  // 并且暴露在 `this` 上
  data() {
    return {
      count: 0
    }
```

```
    },

    // methods 是一些用来更改状态与触发更新的函数
    // 它们可以在模板中作为事件监听器绑定
    methods: {
      increment() {
        this.count++
      }
    },

    // 生命周期钩子会在组件生命周期的各个不同阶段被调用
    // 例如这个函数就会在组件挂载完成后被调用
    mounted() {
      console.log(`The initial count is ${this.count}.`)
    }
  }
  </script>

  <template>
    <button @click="increment">Count is: {{ count }}</button>
  </template>
```

在演练场中尝试一下

组合式 API (Composition API)

通过组合式 API，我们可以使用导入的 API 函数来描述组件逻辑。在单文件组件中，组合式 API 通常会与 `<script setup>` 搭配使用。这个 `setup` attribute 是一个标识，告诉 Vue 需要在编译时进行一些处理，让我们可以更简洁地使用组合式 API。比如， `<script setup>` 中的导入和顶层变量/函数都能够在模板中直接使用。

下面是使用了组合式 API 与 `<script setup>` 改造后和上面的模板完全一样的组件：

vue

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

// 响应式状态
const count = ref(0)

// 用来修改状态、触发更新的函数
function increment() {
```

```
function increment() {
  count.value++
}

// 生命周期钩子
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

在演练场中尝试一下

该选哪一个？

两种 API 风格都能够覆盖大部分的应用场景。它们只是同一个底层系统所提供的两套不同的接口。实际上，选项式 API 是在组合式 API 的基础上实现的！关于 Vue 的基础概念和知识在它们之间都是通用的。

选项式 API 以“组件实例”的概念为中心 (即上述例子中的 `this`)，对于有面向对象语言背景的用户来说，这通常与基于类的心智模型更为一致。同时，它将响应性相关的细节抽象出来，并强制按照选项来组织代码，从而对初学者而言更为友好。

组合式 API 的核心思想是直接在函数作用域内定义响应式状态变量，并将从多个函数中得到的状态组合起来处理复杂问题。这种形式更加自由，也需要你对 Vue 的响应式系统有更深的理解才能高效使用。相应的，它的灵活性也使得组织和重用逻辑的模式变得更加强大。

在组合式 API FAQ 章节中，你可以了解更多关于这两种 API 风格的对比以及组合式 API 所带来的潜在收益。

如果你是使用 Vue 的新手，这里是我们的大致建议：

> 在学习的过程中，推荐采用更易于自己理解的风格。再强调一下，大部分的核心概念在这两种风格之间都是通用的。熟悉了一种风格以后，你也能够很快地理解另一种风格。
>
> 在生产项目中：

当你不需要使用构建工具，或者打算主要在低复杂度的场景中使用 Vue，例如渐进增强的应用场景，推荐采用选项式 API。

当你打算用 Vue 构建完整的单页应用，推荐采用组合式 API + 单文件组件。

在学习阶段，你不必只固守一种风格。在接下来的文档中我们会为你提供一系列两种风格的代码供你参考，你可以随时通过左上角的 **API 风格偏好**来做切换。

还有其他问题？

请查看我们的 **FAQ**。

选择你的学习路径

不同的开发者有不同的学习方式。尽管在可能的情况下，我们推荐你通读所有内容，但你还是可以自由地选择一种自己喜欢的学习路径！

尝试互动教程

适合喜欢边动手边学的读者。

继续阅读该指南

该指南会带你深入了解框架所有方面的细节。

查看示例

浏览核心功能和常见用户界面的示例。

在 GitHub 上编辑此页