



下载APP



06 | 新的代码组织方式：Composition API + <script setup> 到底好在哪里？

2021-10-29 大圣

《玩转Vue 3全家桶》

课程介绍 >



讲述：大圣

时长 12:01 大小 11.02M



你好，我是大圣，欢迎进入课程的第六讲。

在上一讲中，我带你搭建了项目的雏形，这是后面项目开发的起点。从今天开始，我就带你在这个骨架结构的基础之上，开始项目的实战开发。首先我们要掌握的，就是 Vue 3 的 Composition API + <script setup> 这种最新的代码组织方式。



**尤小右**

8月10日 09:06 来自 微博国际版

说认真的，如果你能用 Vue 3 却还在用 Options API，现在有了 <script setup> 没有理由不换 Composition API 了。 <script setup> + TS + Volar = 真香，是时候了。

@尤小右Vue 3.2 发布了! [网页链接](#)

8月10日 04:23 来自 新版微博 weibo.com

[168](#) | [76](#) | [433](#)

我们在前面的第三讲中，有详细地讲到过 Composition API，相信你对这个 API 的语法细节已经有所掌握了。那你肯定会很好奇，这个 <script setup> 又是什么？为什么尤雨溪要在微博上强推 <script setup> 呢？

别急，今天我就带你使用 Composition API 和 <script setup> 去重构第二讲的清单应用。在重构的过程中，你能逐渐明白，**Composition API 可以让我们更好地组织代码结构**，而让你感到好奇的 <script setup> 本质上是以一种更精简的方式来书写 Composition API。

Composition API 和 <script setup> 上手

首先我想提醒你，我们在这一讲中写代码的方式，就和前面的第二讲有很大的区别。

在第二讲中，我们开发清单应用时，是直接在浏览器里使用 Options API 的方式写代码；但在接下来的开发中，我们会直接用单文件组件——也就是 .vue 文件，的开发方式。这种文件格式允许我们把 Vue 组件的 HTML、CSS 和 JavaScript 写在单个文件内容中。下面我带你用单文件组件的方式，去重构第二讲做的清单应用。

我们现在已经搭建好了项目的骨架，以后在这个骨架之内会有很多页面和组件。从这里开始，我们就要逐步适应组件化的开发思路，新的功能会以组件的方式来组织。

按照上一讲制定的规范，首先，我们打开项目文件夹下面的 src 下的 components 目录，新建一个 Todolist.vue，并在这个文件里写出下面的代码：

```
1 <template>
```


[复制代码](#)

```
2   <div>
3     <h1 @click="add">{{count}}</h1>
4   </div>
5 </template>
6
7 <script setup>
8 import { ref } from "vue";
9 let count = ref(1)
10 function add(){
11   count.value++
12 }
13 </script>
14
15 <style>
16 h1 {
17   color: red;
18 }
19 </style>
```

在上述代码中，我们使用 `template` 标签放置模板、`script` 标签放置逻辑代码，并且用 `setup` 标记我们使用 `<script setup>` 的语法，`style` 标签放置 CSS 样式。

从具体效果上看，这段代码实现了一个累加器。在 `<script setup>` 语法中，我们使用引入的 `ref` 函数包裹数字，返回的 `count` 变量就是响应式的数据，使用 `add` 函数实现数字的修改。需要注意的是，对于 `ref` 返回的响应式数据，我们需要修改 `.value` 才能生效，而在 `<script setup>` 标签内定义的变量和函数，都可以在模板中直接使用。


实现累加器以后，我们再回到 `src/pages/Home.vue` 组件中，使用如下代码显示清单应用。在这段代码里，我们直接 `import TodoList.vue` 组件，然后 `<script setup>` 会自动把组件注册到当前组件，这样我们就可以直接在 `template` 中使用来显示清单的功能。

 复制代码

```
1 <template>
2   <h1>这是首页</h1>
3   <TodoList />
4 </template>
5
6 <script setup>
7 import TodoList from '../components/TodoList.vue'
8 </script>
```

这个时候我们就把清单功能独立出来了，可以在任意你需要的地方复用。在课程的后续内容中，我会详细给你介绍基于组件去搭建应用的方式。**通过这种方式，你可以实现对业务逻辑的复用。这样做的好处就是，如果有其他页面也需要用到这个功能，可以直接复用过去。**

然后，我们就可以基于新的语法实现之前的清单应用。下面的代码就是把之前的代码移植过来后，使用 ref 包裹的响应式数据。在你修改 title 和 todos 的时候，注意要修改响应式数据的 value 属性。

 复制代码

```
1 <template>
2   <div>
3     <input type="text" v-model="title" @keydown.enter="addTodo" />
4     <ul v-if="todos.length">
5       <li v-for="todo in todos">
6         <input type="checkbox" v-model="todo.done" />
7         <span :class="{ done: todo.done }"> {{ todo.title }}</span>
8       </li>
9     </ul>
10  </div>
11 </template>
12
13 <script setup>
14 import { ref } from "vue";
15 let title = ref("");
16 let todos = ref([{title: '学习Vue', done: false}])
17
18 function addTodo() {
19   todos.value.push({
20     title: title.value,
21     done: false,
22   });
23   title.value = "";
24 }
25 </script>
```

计算属性

在第二讲开发的清单应用中，我们也用到了计算属性，在 Composition API 的语法中，计算属性和生命周期等功能，都可以脱离 Vue 的组件机制单独使用。我们向 TodoList.vue 代码块中加入下面的代码：

```
1 <template>
2   <div>
3     <input type="text" v-model="title" @keydown.enter="addTodo" />
4     <button v-if="active < all" @click="clear">清理</button>
5     <ul v-if="todos.length">
6       <li v-for="todo in todos">
7         <input type="checkbox" v-model="todo.done" />
8         <span :class="{ done: todo.done }"> {{ todo.title }}</span>
9       </li>
10    </ul>
11    <div v-else>暂无数据</div>
12    <div>
13      全选<input type="checkbox" v-model="allDone" />
14      <span> {{ active }} / {{ all }} </span>
15    </div>
16  </div>
17 </template>
18
19 <script setup>
20 import { ref,computed } from "vue";
21 let title = ref("");
22 let todos = ref([{'title':'学习Vue',done:false}])
23
24 function addTodo() {
25   ...
26 }
27 function clear() {
28   todos.value = todos.value.filter((v) => !v.done);
29 }
30 let active = computed(() => {
31   return todos.value.filter((v) => !v.done).length;
32 });
33 let all = computed(() => todos.value.length);
34 let allDone = computed({
35   get: function () {
36     return active.value === 0;
37   },
38   set: function (value) {
39     todos.value.forEach((todo) => {
40       todo.done = value;
41     });
42   },
43 });
44 </script>
```

在这这段代码中，具体的计算属性的逻辑和第二讲一样，区别仅在于 computed 的用法上。你能看到，第二讲的 computed 是组件的一个配置项，而这里的 computed 的用法是单独引入使用。

Composition API 拆分代码

讲到这里，可能你就会意识到，之前的累加器和清单，虽然功能都很简单，但也属于两个功能模块。如果在一个页面里有这两个功能，那就需要在 data 和 methods 里分别进行配置。但这样的话，数据和方法相关的代码会写在一起，在组件代码行数多了以后就不好维护。**所以，我们需要使用 Composition API 的逻辑来拆分代码，把一个功能相关的数据和方法都维护在一起。**

但是，所有功能代码都写在一起的话，也会带来一些问题：随着功能越来越复杂，script 内部的代码也会越来越多。因此，我们可以进一步对代码进行拆分，把功能独立的模块封装成一个独立的函数，真正做到按需拆分。


在下面，我们新建了一个函数 useTodos：

[复制代码](#)

```
1 function useTodos() {
2   let title = ref("");
3   let todos = ref([{ title: "学习Vue", done: false }]);
4   function addTodo() {
5     todos.value.push({
6       title: title.value,
7       done: false,
8     });
9     title.value = "";
10  }
11  function clear() {
12    todos.value = todos.value.filter((v) => !v.done);
13  }
14  let active = computed(() => {
15    return todos.value.filter((v) => !v.done).length;
16  });
17  let all = computed(() => todos.value.length);
18  let allDone = computed({
19    get: function () {
20      return active.value === 0;
21    },
22    set: function (value) {
23      todos.value.forEach((todo) => {
24        todo.done = value;
25      });
26    },
27  });
28  return { title, todos, addTodo, clear, active, all, allDone };
29 }
```


这个函数就是把那些和清单相关的所有数据和方法，都放在函数内部定义并且返回，这样这个函数就可以放在任意的地方来维护。


而我们的组件入口，也就是 <script setup> 中的代码，就可以变得非常简单和清爽了。在下面的代码中，我们只需要调用 useTodos，并且获取所需要的变量即可，具体的实现逻辑可以去 useTodos 内部维护，代码可维护性大大增强。

 复制代码

```
1 <script setup>
2 import { ref, computed } from "vue";
3
4 let count = ref(1)
5 function add(){
6     count.value++
7 }
8
9 let { title, todos, addTodo, clear, active, all, allDone } = useTodos();
10 </script>
```

我们在使用 Composition API 拆分功能时，也就是执行 useTodos 的时候，ref、computed 等功能都是从 Vue 中单独引入，而不是依赖 this 上下文。其实你可以把组件内部的任何一段代码，从组件文件里抽离出一个独立的文件进行维护。

现在，我们引入追踪鼠标位置的需求进行讲解，比如我们项目中可能有很多地方需要显示鼠标的坐标位置，那我们就可以在项目的 src/utils 文件夹下面新建一个 mouse.js。我们先从 Vue 中引入所需要的 ref 函数，然后暴露一个函数，函数内部和上面封装的 useTodos 类似，不过这次独立成了文件，放在 utils 文件下独立维护，提供给项目的所有组件使用。

 复制代码

```
1 import {ref} from 'vue'
2
3 export function useMouse(){
4
5     const x = ref(0)
6     const y = ref(0)
7
8     return {x, y}
9
10 }
```

想获取鼠标的位置，我们就需要监听 `mousemove` 事件。这需要在组件加载完毕后执行，在 Composition API 中，我们可以直接引入 `onMounted` 和 `onUnmounted` 来实现生命周期的功能。

看下面的代码，组件加载的时候，会触发 `onMounted` 生命周期，我们执行监听 `mousemove` 事件，从而去更新鼠标位置的 `x` 和 `y` 的值；组件卸载的时候，会触发 `onUnmounted` 生命周期，解除 `mousemove` 事件。

[复制代码](#)

```
1
2
3 import {ref, onMounted,onUnmounted} from 'vue'
4
5 export function useMouse(){
6   const x = ref(0)
7   const y = ref(0)
8   function update(e) {
9     x.value = e.pageX
10    y.value = e.pageY
11  }
12  onMounted(() => {
13    window.addEventListener('mousemove', update)
14  })
15
16  onUnmounted(() => {
17    window.removeEventListener('mousemove', update)
18  })
19  return { x, y }
20 }
```

完成了上面的鼠标事件封装这一步之后，我们在组件的入口就可以和普通函数一样使用 `useMouse` 函数。在下面的代码中，上面的代码返回的 `x` 和 `y` 的值可以在模板任意地方使用，也会随着鼠标的移动而改变数值。

[复制代码](#)

```
1 import {useMouse} from '../utils/mouse'
2
3 let {x,y} = useMouse()
```


相信到这里，你一定能体会到 Composition API 对代码组织方式的好处。简单来看，**因为 ref 和 computed 等功能都可以从 Vue 中全局引入，所以我们就可以把组件进行任意颗粒度的拆分和组合**，这样就大大提高了代码的可维护性和复用性。

<script setup> 好用的功能

Composition API 带来的好处你已经掌握了，而 <script setup> 是为了提高我们使用 Composition API 的效率而存在的。我们还用累加器来举例，如果没有 <script setup>，那么我们需要写出下面这样的代码来实现累加器。

[复制代码](#)

```
1 <script >
2 import { ref } from "vue";
3 export default {
4   setup() {
5     let count = ref(1)
6     function add() {
7       count.value++
8     }
9     return {
10       count,
11       add
12     }
13   }
14 }
15 </script>
```

在上面的代码中，我们要在 <script> 中导出一个对象。我们在 setup 配置函数中写代码时，和 Options 的写法比，也多了两层嵌套。并且，我们还要在 setup 函数中，返回所有需要在模板中使用的变量和方法。上面的代码中，setup 函数就返回了 count 和 add。

使用 <script setup> 可以让代码变得更加精简，这也是现在开发 Vue 3 项目必备的写法。除了我们上面介绍的功能，<script setup> 还有其它一些很好用的功能，比如能够使用顶层的 await 去请求后端的数据等等，我们会在后面的项目中看到这种使用方法。

style 样式的特性

除了 script 相关的配置，我也有必要给你介绍一下 style 样式的配置。比如，在 style 标签上，当我们加上 scoped 这个属性的时候，我们定义的 CSS 就只会应用到当前组件的元素

上，这样就很好地避免了一些样式冲突的问题。

我们项目中的样式也可以加上如下标签：

[复制代码](#)

```
1 <style scoped>
2 h1 {
3   color: red;
4 }
5 </style>>
```

这样，组件就会解析成下面代码的样子。标签和样式的属性上，新增了 data- 的前缀，确保只在当前组件生效。

[复制代码](#)

```
1 <h1 data-v-3de47834="">1</h1>
2 <style scoped>
3 h1[data-v-3de47834] {
4   color: red;
5 }
6 </style>
```

如果在 scoped 内部，你还想写全局的样式，那么你可以用:global 来标记，这样能确保你可以很灵活地组合你的样式代码（后面项目中用到的话，我还会结合实战进行讲解）。而且我们甚至可以通过 v-bind 函数，直接在 CSS 中使用 JavaScript 中的变量。

在下面这段代码中，我在 script 里定义了一个响应式的 color 变量，并且在累加的时候，将变量随机修改为红或者蓝。在 style 内部，我们使用 v-bind 函数绑定 color 的值，就可以动态地通过 JavaScript 的变量实现 CSS 的样式修改，点击累加器的时候文本颜色会随机切换为红或者蓝。

[复制代码](#)

```
1 <template>
2   <div>
3     <h1 @click="add">{{ count }}</h1>
4   </div>
5 </template>
6
7 <script setup>
```

```
8 import { ref } from "vue";
9 let count = ref(1)
10 let color = ref('red')
11 function add() {
12   count.value++
13   color.value = Math.random()>0.5? "blue":"red"
14 }
15 </script>
16
17 <style scoped>
18 h1 {
19   color:v-bind(color);
20 }
21 </style>>
```

点击累加器时文本颜色的切换效果，如下图所示：



总结

我们来总结一下今天都学到了什么吧。今天的主要任务就是使用 Composition API + <script setup> 的语法复现第二讲的清单应用，我们首先通过累加器的例子介绍了 ref 这个函数的使用；之后我们讲到，在 Composition API 的语法中，所有的功能都是通过全局引入的方式使用的，并且通过 <script setup> 的功能，我们定义的变量、函数和引入的组件，都不需要额外的生命周期，就可以直接在模板中使用。

然后，我们通过把功能拆分成函数和文件的方式，掌握到 Composition API 组织代码的方式，我们可以任意拆分组件的功能，抽离出独立的工具函数，大大提高了代码的可维护性。

最后我们还学习了 style 标签的特殊属性，通过标记 scoped 可以让样式只在当前的组件内部生效，还可以通过 v-bind 函数来使用 JavaScript 中的变量去渲染样式，如果这个变量是响应式数据，就可以很方便地实现样式的切换。

相信学完今天这一讲，你一定会对我们为什么需要 Composition API 有更进一步的认识，而对于 <script setup> 来说，则可以帮助我们更好且更简洁的写 Compostion 的语法。在后面，我们的项目会全部使用 Composition API + <script setup> 来进行书写。

思考题

最后给你留一个思考题，Composition API 和 <script setup> 虽然能提高开发效率，但是带来的一些新的语法，比如 ref 返回的数据就需要修改 value 属性；响应式和生命周期也需要 import 后才能使用等等，很多人也在社区批评这是 Vue 造的“方言”，你怎么看呢？

欢迎你在留言区分享你的想法，当然也推荐你把这一讲推荐给你自己的朋友、同事。我们下一讲见！

分享给需要的人，Ta订阅后你可得 **20 元现金奖励**

 生成海报并分享

 赞 10

 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | 项目启动：搭建Vue 3工程化项目第一步

精选留言 (25)

 写留言



|| 认证

2021-10-29

每节课都有很多收获！

✿ Options API vs Composition API

字面上, 选项 API 与 组合 API, 细品, 这反映了设计面向的改变：

1. 选项，谁的选项，关键在“谁”。谁？组件。也是 Vue2.x 的设计基础。组件有什么，...

展开 ∨

**cwang**

2021-10-29

谢谢大圣老师的讲解。其中清单应用中，独立出来的useTodos函数，放在了哪里进行维护？并且，在使用这个函数时：`let { title,} = useTodos()`，是不是还要对它进行一个引用？谢谢。

5

3

**王俊**

2021-10-29

建议增加代码库，实时拉取更新，现在的代码片段看着比较麻烦

1

2

**Condor Hero**

2021-10-29

本次课程的知识点可以在 Vue3 官网进行详细学习：

1. 组合式 API：<https://v3.cn.vuejs.org/api/composition-api.html#setup>
2. 单文件组件 <script setup>：<https://v3.cn.vuejs.org/api/sfc-script-setup.html>
3. 单文件组件样式特性：<https://v3.cn.vuejs.org/api/sfc-style.html>

展开 ∨



1

**ch3cknull**

2021-10-29

关于导入问题，antfu大神有一个插件unplugin-auto-import，可以自动注入依赖项，不用import

<https://github.com/antfu/unplugin-auto-import>

...

展开 ∨



1

@

2021-10-29

composition API 更接近于写原生js的思维

展开 ∨





懒懒想睡觉
2021-10-29

对ref不太熟悉哦 😊

展开 ▾



peterpc
2021-10-29

这节课的内容感觉少，但是都是干货，至于方言，哪个框架还没有自己的小黑科技，好用就好



奇奇
2021-10-29

大圣，能不能搞一个 git 仓库来放每一讲的课程代码内容呢



666de6
2021-10-29

把清单demo重新写了一遍，确实.value是有点不适应，其他还好。Composition API和<script setup>的体验还是可以的，还有css通过v-bind使用js变量是针不戳



乐叶
2021-10-29

生命周期可以重复使用，引入util下通用功能，涉及生命周期，以前只能通过mixin混入，现在都可以直接引入使用。

展开 ▾



Warn
2021-10-29

示例中ref变量可以使用const来替代let定义的。

style使用v-bind来调用JavaScript定义的变量，受教了，感觉对于主题定制很友好啊（希望大圣老师能在以后的课程中，继续穿插一些这种style的小技能点比如sass的v-deep等等）。

...

展开 ▾



**Farewell**

2021-10-29

hooks真香

展开 ∨

**超**

2021-10-29

请问大圣，如果把useTodos单独抽成一个js文件，改js里的文件，浏览器界面不会自动热重载，有啥解决方法没？是改webpack的配置吗？

展开 ∨

**七桐木**

2021-10-29

感觉组合式API和选项式API只不过把代码的组织方式调整了一下，把功能相近的进行抽离，更方便于维护，习惯了就好

展开 ∨

**tequ11Aneio**

2021-10-29

这特么和hooks有毛区别啊！！！！

展开 ∨

**undefined**

2021-10-29

大圣老师以后可以辛苦贴一下源码地址嘛，以后文件多了可能比较需要！

**速冻**

2021-10-29

大圣老师，有课程相关的github项目吗

展开 ∨

**速冻**

2021-10-29

打卡打卡 ☺

展开



Geek_a84b8d

2021-10-29

方言不方言的不重要 好用就行

展开

