

10 | 动态渲染组件：如何实现Vue的动态渲染组件？

2022-12-14 杨文坚 来自北京

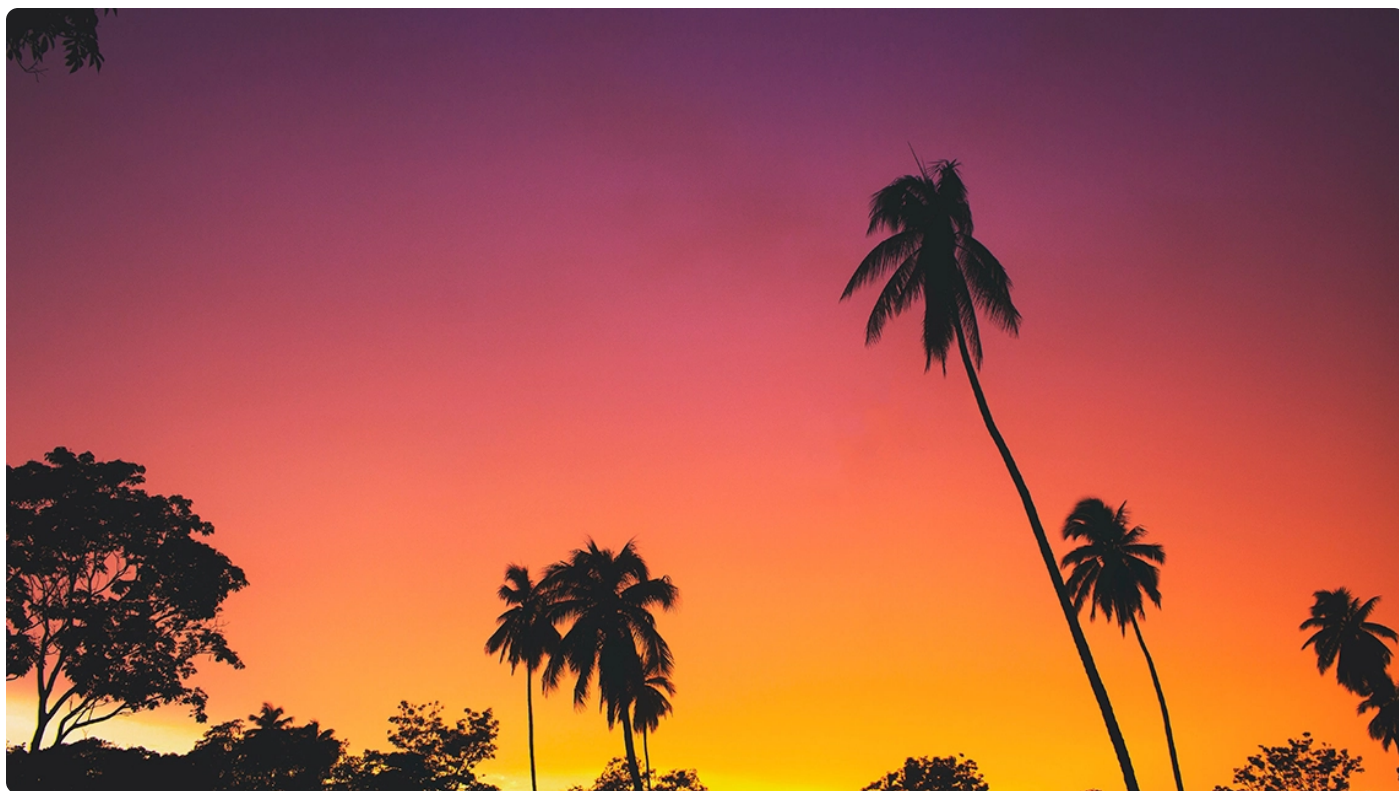


天下无鱼

<https://shikey.com/>

《Vue 3 企业级项目实战课》

[课程介绍 >](#)



讲述：杨文坚

时长 12:20 大小 11.26M



你好，我是杨文坚。

前两节课，我们已经入门了 Vue.js 3.x 自研组件库的开发和组件库的主题方案设计，还了解了按钮组件这一基础组件的开发流程及其主题化实现。

组件库，核心是将不同类型的功能组件聚合起来，提供给业务前端开发者按需选择，用到实际前端页面上。接下来的几节课，我们主要会一步步开发不同类型的 Vue.js 3.x 的组件，打造属于自己的 Vue.js 3.x 企业级自研组件库。

今天，我们先学习组件库里常见的一种组件类型，**动态渲染组件**。

什么是动态渲染组件？

在平时工作中用 Ant Design 或者 Element Plus 等前端组件库时，相信你经常会用到对话框、消息提醒和侧边抽屉等组件，来做一些信息的动态显示和动态操作，这类组件都可以通过函数式的方式直接调用。



这类组件从功能上分类，一般被称为“反馈类型组件”。如果从技术实现方式上归类，就可以归纳为“动态渲染组件”。**那我们如何从技术实现的角度，理解动态渲染组件呢？**

从字面意义可以看出来，动态渲染组件就是通过“动态”的方式来“渲染”组件，不需要像常规 Vue.js 3.x 组件那样，把组件注册到模板里使用。

所以，动态渲染组件的两个技术特点就是：

- 以直接函数式地使用来执行渲染，使用者不需要写代码来挂载组件；
- 组件内部实现了动态挂载和卸载节点的操作。

如果我们把这 2 个技术特点放在 Vue.js 3.x 的框架体系内来理解，你会发现：**Vue.js 3.x 动态渲染组件在页面上是独立于“Vue.js 主应用”之外的渲染。**

因为，动态渲染组件，在项目应用里，只是调用组件的函数（或者方法），然后创建一个独立于“Vue.js 主应用”外的一个“Vue.js 副应用”，动态挂载（mount）在 HTML 动态创建的 DOM 上。如果要关闭动态渲染组件，就需要再次触发一个关闭函数（或者方法），把这个“Vue.js 的副应用”卸载（unmount），最后把动态创建的 DOM 也一并销毁回收。这样，就完成动态渲染组件从挂载到卸载的一个完整的生命周期。

那么，动态渲染组件一般可以实现组件库里的哪些功能组件？或者说，组件库的建设一般需要哪些动态渲染组件呢？

工作开发中高频用到的动态渲染组件有**消息提醒组件（Message）和对话框组件（Dialog）**。所以，接下来我们会围绕这两个组件动态渲染的技术实现方案展开。

不过，在实现 Message 和 Dialog 这两个动态渲染组件之前，我们首要的工作是知道实现动态组件前要准备什么。

实现动态渲染组件需要做什么技术准备？

组件库是提供给开发者使用的，除了功能，开发者最关心的当然就是组件的 API（接口）使用方式，所以**动态渲染组件 API 设计尤为重要**。



我们前面说过，动态渲染组件是直接通过函数方法触发调用，所以我们首先要准备的就是对这个函数方法的 API 的设计。

需要关注的是，这里不仅仅要考虑技术层面的 API 设计，我们还需要考虑到组件库使用者的开发使用体验。我建议在设计函数方法的 API 的时候，以组件库的使用者视角作为出发点，换位思考一下，如果你作为组件库的使用者，使用一个函数方法来触发一个动态渲染组件时，是不是使用方式越简单越好，学习成本越低越好？

所以，我们在设计动态渲染组件的 API 时，**使用步骤尽量少，使用代码尽量精简**。

而动态渲染组件整个生命周期，最核心的就是“动态挂载”和“动态卸载”两个步骤，所以 API 的设计，可以直接围绕着“挂载”和“卸载”两个来设计，最简单的可以设计成如下代码：

 复制代码

```
1 import { Module } from 'xxxx'
2
3 // 动态组件挂载
4 Module.open({ /* 组件参数 */ });
5
6 // 动态组件卸载
7 Module.close();
```

如果考虑到组件不是单例的，而是多实例共存的，可以这么设计 API：

 复制代码

```
1 import { Module } from 'xxxx'
2
3 // 创建动态组件 mod1
4 const mod1 = Module.create({ /* 组件参数 */ });
5 // 挂载渲染 mod1
6 mod1.open();
7 // 卸载动态组件 mod1
8 mod1.close();
9
10
11
12 // 创建动态组件 mod2
```

```
13 const mod2 = Module.create({ /* 组件参数 */ });
14 // 挂载渲染 mod2
15 mod2.open();
16 // 卸载动态组件 mod2
17 mod2.close();
```



如果动态组件在其生命周期还需要添加一些节点，可以这么来设计：

复制代码

```
1 import { Module } from 'xxxx'
2
3 // 创建动态组件 mod1
4 const mod1 = Module.create({ /* 组件参数 */ });
5 // 挂载渲染 mod1
6 mod1.open();
7 // 更新组 mod1 内容
8 mod1.update({ /* 更新内容参数 */ })
9 // 卸载动态组件 mod1
10 mod1.close();
```

完成了动态组件的函数方法 API 设计，接下来，我们学习 Vue.js 3.x 实现组件的动态挂载和卸载操作。

前面说了，Vue.js 3.x 动态渲染组件本质就是渲染一个独立的 Vue.js 3.x 的应用，只是独立于本身页面主应用存在，那么，组件渲染挂载的时候，就需要一个动态的挂载节点 DOM，用 Vue.js 3.x 创建一个 App 挂载到这个动态节点 DOM 上面。

后续到组件生命周期结束时，也就是卸载动态渲染组件时，再用 Vue.js 3.x 卸载 App 的方法，把组件从 DOM 上卸载掉，同时把动态创建的 DOM 给销毁掉。

整个过程，我们可以用最简单的 Vue.js 3.x 代码实现：

复制代码

```
1 import { defineComponent, createApp, h } from 'vue';
2
3 // 用 JSX 语法实现一个Vue.js 3.x的组件
4 const ModuleComponent = defineComponent({
5   setup(props, context) {
6     return () => {
7       return (
8         <div>这是一个动态渲染的组件</div>
```

```

9      });
10    };
11  }
12 });
13
14 // 实现动态渲染组件的过程
15
16 export const createModule = () => {
17   // 创建动态节点DOM
18   const dom = document.createElement('div');
19   // 把 DOM 追加到页面 body标签里
20   const body = document.querySelector('body') as HTMLBodyElement;
21   const app = createApp({
22     render() {
23       return h(DialogComponent, {});
24     }
25   });
26
27
28   // 返回当前组件的操作实例
29   // 其中封装了挂载和卸载组件的方法
30   return {
31     open(): () => {
32       // 把组件 ModuleComponent 作为一个独立应用挂载在 DOM 节点上
33       app.mount(dom);
34     },
35     close(): () => {
36       // 卸载组件
37       app.unmount();
38       // 销毁动态节点
39       dom.remove();
40     }
41   }
42 }

```



天下无鱼

<https://shikey.com/>

上面封装的一个最简单的动态渲染组件，可以这么使用：

 复制代码

```

1 import { createModule } from './xxxx';
2
3 // 创建和渲染组件
4 const mod = createModule();
5
6 // 挂载渲染组件
7 mod.open();
8
9 // 卸载关闭组件
10 mod.close();

```

通过这些代码演示和注释，相信你可以一目了然地看到动态渲染组件一个完整的开发实现过程。



在这个例子里，我们通过一个 **JSX** 语法实现的动态渲染组件，而 **JSX** 的语法比较灵活，在动态渲染组件的实现上有很大的发挥空间，我们还可以用 **JSX** 语法，把一个 **Vue.js 3.x** 的动态组件实现在一个独立的 **JS** 文件中，不需要额外新建 **Vue** 文件来书写组件实体。

那么问题来了，我们能不能用不同于 **JSX** 语法的另一种语法，也就是模板语法，来实现动态渲染组件呢？

答案是可以的。

如何实现一个动态 Message 组件？

实现动态 **Message** 组件前，你可以先看一下最终的功能效果：



可以看出，**Message** 组件主要功能是显示消息提示，在挂载渲染后的一段时间内自动关闭，所以，这个组件除了能实现 **API** 控制的挂载渲染和卸载关闭，也需要支持默认自动关闭。

想实现的效果我们知道了，动态渲染组件的技术准备前面也都掌握了，接下来，我们就实现这个 **Message** 动态渲染组件。

主要分四步：

- 第一步，先用 Vue.js 3.x 模板语法实现 Message 的模板组件；
- 第二步，封装 open 函数来控制挂载渲染这个模板语法的 Message 组件；
- 第三步，封装 close 函数来控制卸载关闭这个组件；
- 第四步，扩展 open 函数，内置一个定时器来控制延时自动关闭组件。



在进入正式开发前，我们先定义以下参数的 TypeScript 类型，如下所示：

复制代码

```
1 // ./types.ts
2 export type MessageType = 'info' | 'success' | 'warn' | 'error';
3
4 export interface MessageParams {
5   text: string;
6   type?: MessageType;
7   duration?: number;
8 }
9
```

进入第一步，也就是用 Vue.js 3.x 模板语法先实现 Message 的模板组件，实现代码如下所示：

复制代码

```
1 <template>
2   <div v-if="show" :class="{ [baseClassName]: true, [typeClassName]: true }">
3     <div>{{ props.text }}</div>
4   </div>
5 </template>
6
7 <script lang="ts" setup>
8   import { onMounted, ref } from 'vue';
9   import { prefixName } from '../theme/index';
10  import type { MessageType } from './types';
11
12  const show = ref<boolean>(false);
13
14  onMounted(() => {
15    show.value = true;
16  });
17
18  const props = withDefaults(
19    defineProps<{
20      text?: string;
21      type?: MessageType;
```

```

22     }>(),
23     {
24         type: 'info'
25     }
26 );
27
28 const closeMessage = () => {
29     show.value = false;
30 };
31
32 defineExpose<{
33     closeMessage: () => void;
34 }>({ closeMessage: closeMessage });
35
36 const baseClassName = `${prefixName}-message`;
37 const typeClassName = `${baseClassName}-${props.type}`;
38 </script>
39

```



在上述代码中，我把 **Message** 组件的消息提醒划分成几种类型，用几种不同颜色来显示区别，具体实现方式是基于 **Less** 和 **CSS Variable**，如果你有什么不懂，可以翻看上一节课的内容。

进入第二步来封装 **open** 函数，代码片段如下所示：

复制代码

```

1 import { createApp, h } from 'vue';
2 import MessageComponent from './message.vue';
3 import type { MessageParams } from './types';
4
5 const Message = {
6     // 封装open函数
7     open(params: MessageParams) {
8         const dom = document.createElement('div');
9         const body = document.querySelector('body') as HTMLBodyElement;
10         let duration: number | undefined = params.duration;
11         if (duration === undefined) {
12             duration = 3000;
13         }
14         body.appendChild(dom);
15         const msg = h(MessageComponent, {
16             text: params.text,
17             type: params.type
18         });
19         const app = createApp({
20             render() {
21                 return msg;

```



```

22     }
23   });
24   // 挂载和渲染Message组件
25   app.mount(dom);
26
27   // 后续等待返回 close 函数
28   }
29 };
30
31 export default Message;
32

```



到了第三步，就是基于已有 open 函数来返回 close 函数，代码如下所示：

 复制代码

```

1  import { createApp, h } from 'vue';
2  import MessageComponent from './message.vue';
3  import type { MessageParams } from './types';
4
5  const Message = {
6    open(params: MessageParams) {
7      // 这里省略open函数渲染Message组件的代码
8
9      // 封装内部关闭函数
10     const internalClose = () => {
11       msg.component?.exposed?.['closeMessage']?.();
12       app.unmount();
13       dom.remove();
14     };
15
16     let timer: number | null = null;
17     if (duration > 0) {
18       timer = setTimeout(() => {
19         internalClose();
20       }, duration);
21     }
22
23     // 最后返回可控制Message关闭的close函数
24     return {
25       close: () => {
26         if (timer) {
27           clearTimeout(timer);
28           timer = null;
29         }
30         internalClose();
31       }
32     };
33   }
34 };
35

```

```
36 export default Message;
```



最后，第四步封装定时器到 `open` 函数中来控制挂载渲染这个模板语法的 `Message` 组件，最终实现代码所示：

复制代码

```
1 import { createApp, h } from 'vue';
2 import MessageComponent from './message.vue';
3 import type { MessageParams } from './types';
4
5 const Message = {
6   open(params: MessageParams) {
7     const dom = document.createElement('div');
8     const body = document.querySelector('body') as HTMLBodyElement;
9     let duration: number | undefined = params.duration;
10    if (duration === undefined) {
11      duration = 3000;
12    }
13    body.appendChild(dom);
14    const msg = h(MessageComponent, {
15      text: params.text,
16      type: params.type
17    });
18    const app = createApp({
19      render() {
20        return msg;
21      }
22    });
23    app.mount(dom);
24
25    const internalClose = () => {
26      msg.component?.exposed?.['closeMessage']?.();
27      app.unmount();
28      dom.remove();
29    };
30
31    let timer: number | null = null;
32    if (duration > 0) {
33      timer = setTimeout(() => {
34        internalClose();
35      }, duration);
36    }
37
38    return {
39      close: () => {
40        if (timer) {
41          clearTimeout(timer);
42          timer = null;
43        }
44      }
45    };
46  }
47 }
```

```
44     internalClose();
45   }
46   };
47 }
48 };
49
50 export default Message;
51
```



最终可以这么来使用：

 复制代码

```
1 import Message from './message';
2
3 // 自动关闭
4 Message.open({
5   text: '这是一个success类型的消息提醒组件，5秒后自动关闭',
6   type: 'success',
7   duration: 5000
8 })
9
10
11
12 const msg = Message.open({
13   text: '这是一个success类型的消息提醒组件，不会自动关闭',
14   type: 'success',
15   duration: 0
16 })
17 // 如果要关闭，就执行 msg.close() 来关闭这个组件
18
```

至此，我们就已经学会了用动态渲染组件开发思路，来实现一个消息提示的功能组件 Message。

不过到这儿，还没有大功告成。我们要做一个企业级项目，当然要按照企业级的用户体验来要求技术实现。不知道你有没有发现，我实现的这个 Message 组件，在显示消息和关闭消息过程中很突兀，**消息突然显示和消失，组件变化中间没有过渡，给人一种不友好的用户体验。**

这时候我们就需要用到动画过渡的效果，来消除组件显示和消失的突兀感觉，那在 Vue.js 3.x 组件开发中，如何实现组件的动画效果呢？

动态渲染组件的动画效果实现？

Vue.js 3.x 官方对组件动画过渡效果实现提供了一个内置的组件 `<transition>`，这个组件使用起来比较简单，有 JavaScript 和 CSS3 两种控制动画的方式，我们就选择 CSS3 这个比较简单的动画过渡方式来实现。



你可以先看一下最终的功能效果，如下动图所示：



基于上述的 Message 组件，我们可以加入 `<transition>` 内置组件来控制动画过渡。Vue 相关代码调整为：

复制代码

```
1 <template>
2   <Transition :name="fadeClassName">
3     <div v-if="show" :class="{ [baseClassName]: true, [typeClassName]: true }">
4       <div>{{ props.text }}</div>
5     </div>
6   </Transition>
7 </template>
8
9 <script lang="ts" setup>
10  import { onMounted, ref } from 'vue';
11  import { prefixName } from '../theme/index';
12  import type { MessageType } from './types';
13
14  const show = ref<boolean>(false);
15
16  onMounted(() => {
17    show.value = true;
18  });
19
20  const props = withDefaults(
```

```

21   defineProps<{
22     text?: string;
23     type?: MessageType;
24   }>(),
25   {
26     type: 'info'
27   }
28 );
29
30 const closeMessage = () => {
31   show.value = false;
32 };
33
34 defineExpose<{
35   closeMessage: () => void;
36 }>({ closeMessage: closeMessage });
37
38 const baseClassName = `${prefixName}-message`;
39 const typeClassName = `${baseClassName}-${props.type}`;
40 const fadeClassName = `${baseClassName}-fade`;
41 </script>
42

```



天下无鱼

<https://shikey.com/>

针对 `<transition>` 组件的 CSS3 动画过渡样式，在 Less 文件中补充样式代码如下：

```

1  .@{message-name}-fade-enter-active,
2  .@{message-name}-fade-leave-active {
3    transition: opacity 0.5s ease;
4  }
5
6  .@{message-name}-fade-enter-from,
7  .@{message-name}-fade-leave-to {
8    opacity: 0;
9  }

```

复制代码

Vue.js 3.x 内置动画效果组件 `<transition>`，结合 CSS3 实现动画过渡，**核心原理就是在运行过程中，自动检查目标组件是否使用了 CSS 的过渡动画样式**。如果使用了，会在适当的组件变化过程中添加或者删除对应的样式的 `className`。

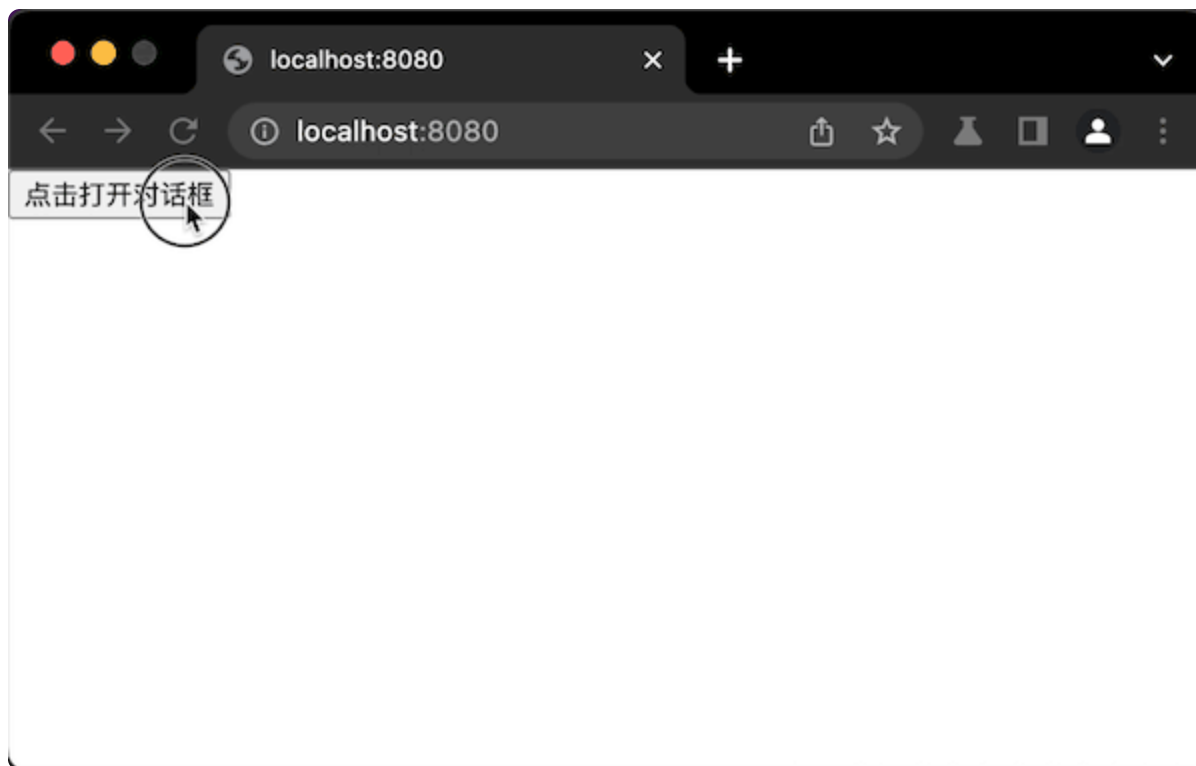
在 `<transition>` 基于 CSS3 动画样式实现的动画过渡效果中，真正实现过渡的动画效果是 CSS3 动画样式，内置的 `<transition>` 组件只是帮助开发者更加方便和合理地把握动画的执行时机（如果你还想了解更多 Vue.js 3.x 动画效果的实现，可以参阅 [官方文档](#)）。

如何实现一个动态 Dialog 组件？

现在，看我们实现的 Message 组件，你会发现只是显示消息提醒，没有任何交互操作。那么，动态渲染组件实现的功能组件，有哪些是既可以显示消息，又可以交互操作的呢？

我们最经常遇到的就是 Dialog 组件，也就是对话框组件。接下来我们就用动态渲染组件的开发思路来开发一个对话框组件。

实现动态 Dialog 组件前，我还是先给你看一下最终的功能效果，如下动图所示：



看了这个动图，是不是觉得很眼熟？没错，就是我们在第 4 节课中，用 JSX 语法实现的那个对话框组件。

上次我们是从 JSX 语法角度来分析实现过程，这节课我们再从动态渲染组件的角度来讲解这个实现过程，具体步骤如下：

- 第一步，实现 Dialog 的实体组件，用 JSX 语法或模板语法都可以。这里，Dialog 组件要用 Emit 方式注册好回调事件；
- 第二步，封装 createDialog 函数来创建一个 Dialog 的实例。这个过程要注意配置好 Dialog 的回调函数等操作；
- 第三步，封装 close 函数来控制卸载关闭这个组件。

具体实现代码也跟第 4 课实现的代码一致，我把它再贴出来了，你可以再看看。这个是 JSX 实现的 Dialog 实体组件：



复制代码

```
1 // ./dialog.tsx
2 import { defineComponent } from 'vue';
3 import { prefixName } from '../theme/index';
4
5 export const DialogComponent = defineComponent({
6   props: {
7     text: String
8   },
9   emits: ['onOk'],
10  setup(props, context) {
11    const { emit } = context;
12    const onOk = () => {
13      emit('onOk');
14    };
15    return () => {
16      return (
17        <div class={` ${prefixName}-dialog-mask`} >
18          <div class={` ${prefixName}-dialog`} >
19            <div class={` ${prefixName}-dialog-text`} >{props.text}</div>
20            <div class={` ${prefixName}-dialog-footer`} >
21              <button class={` ${prefixName}-dialog-btn`} onClick={onOk}>
22                确定
23              </button>
24            </div>
25          </div>
26        </div>
27      );
28    };
29  }
30 });
```

以下是封装了函数方法调用的动态渲染组件的方式：

复制代码

```
1 import { createApp, h } from 'vue';
2 import { DialogComponent } from './dialog';
3
4 function createDialog(params: { text: string; onOk: () => void }) {
5   const dom = document.createElement('div');
6   const body = document.querySelector('body') as HTMLBodyElement;
7   body.appendChild(dom);
8   const app = createApp({
9     render() {
```



```
10     return h(DialogComponent, {
11       text: params.text,
12       onOk: params.onOk
13     });
14   }
15 });
16 app.mount(dom);
17
18 return {
19   close: () => {
20     app.unmount();
21     dom.remove();
22   }
23 };
24 }
25
26 const Dialog: { createDialog: typeof createDialog } = {
27   createDialog
28 };
29
30 export default Dialog;
```



最后你会发现，**Dialog** 动态渲染组件和 **Message** 动态渲染组件，实现流程是类似的，核心是要用函数方法来控制组件的“动态挂载”和“动态卸载”。

总结

这节课核心内容就是 **Vue.js 3.x** 动态渲染组件的实现思路，我再总结一下具体的实现步骤：

- 第一步：设计动态渲染组件的使用函数方法的 **API**，**API** 越简洁越好，核心是要控制组件渲染的挂载和卸载的生命周期；
- 第二步：基于 **Vue.js 3.x** 实现动态渲染组件的原理，核心是要在页面上动态创建 **DOM**，再用 **Vue.js 3.x** 创建一个独立应用来“承载”这个组件，挂载在这个动态 **DOM** 上面；
- 第三步：关闭动态组件时，要卸载这个 **Vue.js 3.x** 的“独立应用”，卸载完再销毁这个动态 **DOM**。

在实现 **Vue.js 3.x** 动态渲染组件的时候，还有几点需要你多加注意：

- 在组件的挂载和卸载过程中，尽量用 **Vue.js 3.x** 的内置 **<transition>** 组件来实现动画过渡效果，提升用户体验，减少组件动态显示和消失的突兀感觉。

- 动态组件在关闭后，注意要记得销毁组件挂载的动态 DOM，释放没必要的内存占用，减少内存泄漏的风险。




思考题

Vue.js 3.x 动态渲染组件是通过创建一个新的 Vue.js 应用来渲染组件的，跟页面原有的 Vue.js 应用是互相独立的，那么如何实现这两个 Vue.js 应用的数据通信呢？

欢迎在留言区分享你的想法，参与讨论，如果对今天的内容有疑问，也欢迎留言，下节课见。

[🔗 完整的代码在这里](#)

分享给需要的人，Ta购买本课程，你将得 18 元

 生成海报并分享

 赞 3  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](#) 09 | 主题方案和基础组件：如何设计组件库的主题方案？

[下一篇](#) 11 | 布局组件：如何实现自研组件库的布局方案？



技术领导力实战笔记 2022

从实操中提升你的领导力

TGO 鲲鹏会

数十位优秀管理者的真知灼见

肖军 / 苏宁金科 CTO
王璞 / DatenLord 联合创始人
郭炜 / 前易观数据 CTO
肖德时 / 前数人云 CTO
林晓峰 / GrowingIO 副总裁
于游 / 马泷医疗集团 CTO
王植萌 / 去哪儿网高级技术总监
胡广寰 / 酷家乐技术 VP
舒超 / 星汉未来 CTO



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。