



下载APP



## 23 | 弹窗：如何设计一个弹窗组件？

2021-12-13 大圣

《玩转Vue 3全家桶》

课程介绍 &gt;

**讲述：大圣**

时长 08:24 大小 7.71M



你好，我是大圣。

上一讲我们剖析了表单组件的实现模式，相信学完之后，你已经掌握了表单类型组件设计的细节，表单组件的主要功能就是在页面上获取用户的输入。

不过，用户在交互完成之后，还需要知道交互的结果状态，这就需要我们提供专门用来反馈操作状态的组件。这类组件根据反馈的级别不同，也分成了很多种类型，比如全屏灰色遮罩、居中显示的对话框 Dialog，在交互按钮侧面显示、用来做简单提示的 tooltip，<sup>11</sup> 及右上角显示信息的通知组件 Notification 等，这类组件的交互体验你都可以在

[Element3 官网](#)感受。



今天的代码也会用 Element3 的 Dialog 组件和 Notification 进行举例，在动手写代码实现之前，我们先从这个弹窗组件的需求开始说起。

## 组件需求分析

我们先来设计一下要做的组件，通过这部分内容，还可以帮你继续加深一下对单元测试 Jest 框架的使用熟练度。我建议你在设计一个新的组件的时候，也试试采用这种方式，先把组件所有的功能都罗列出来，分析清楚需求再具体实现，这样能够让你后面的工作事半功倍。

首先无论是对话框 Dialog，还是消息弹窗 Notification，它们都由一个弹窗的标题，以及具体的弹窗的内容组成的。我们希望弹窗有一个关闭的按钮，点击之后就可以关闭弹窗，弹窗关闭之后还可以设置回调函数。

下面这段代码演示了 dialog 组件的使用方法，通过 title 显示标题，通过 slot 显示文本内容和交互按钮，而通过 v-model 就能控制显示状态。

[复制代码](#)


```
1 <el-dialog
2   title="提示"
3   :visible.sync="dialogVisible"
4   width="30%"
5   v-model:visible="dialogVisible"
6 >
7   <span>这是一段信息</span>
8   <template #footer>
9     <span class="dialog-footer">
10       <el-button @click="dialogVisible = false">取 消</el-button>
11       <el-button type="primary" @click="dialogVisible = false">确 定</el-button>
12     </span>
13   </template>
14 </el-dialog>
```

这类组件实现起来和表单类组件区别不是特别大，我们首先需要做的就是**控制好组件的数据传递**，并且使用 Teleport 渲染到页面顶层的 body 标签。

像 Dialog 和 Notification 类的组件，我们只是单纯想显示一个提示或者报错信息，过几秒就删除，如果在每个组件内部都需要写一个 <Dialog v-if>，并且使用 v-if 绑定变量的方式控制显示就会显得很冗余。

所以，这里就要用到一种调用 Vue 组件的新方式：我们可以使用 JavaScript 的 API 动态地创建和渲染 Vue 的组件。具体如何实现呢？我们以 Notification 组件为例一起看一下。

下面的代码是 Element3 的 Notification 演示代码。组件内部只有两个 button，我们不需要书写额外的组件标签，只需要在 `<script setup>` 中使用 `Notification.success` 函数，就会在页面动态创建 Notification 组件，并且显示在页面右上角。


 复制代码

```
1 <template>
2   <el-button plain @click="open1"> 成功 </el-button>
3   <el-button plain @click="open2"> 警告 </el-button>
4 </template>
5 <script setup>
6   import { Notification } from 'element3'
7
8   function open1() {
9     Notification.success({
10       title: '成功',
11       message: '这是一条成功的提示消息',
12       type: 'success'
13     })
14   }
15   function open2() {
16     Notification.warning({
17       title: '警告',
18       message: '这是一条警告的提示消息',
19       type: 'warning'
20     })
21   }
22
23
24 </script>
```

## 弹窗组件实现


分析完需求之后，我们借助单元测试的方法来实现这个弹窗组件（单元测试的内容如果记不清了，你可以回顾 [第 20 讲](#)）。

我们依次来分析 Notification 的代码，相比于写 Demo 逻辑的代码，这次我们体验一下实际的组件和演示组件的区别。我们来到 element3 下面的 `src/components/Notification/notifucation.vue` 代码中，下面的代码构成了组件的主体框架，我们不去直接写组件的逻辑，而是先从测试代码来梳理组件的功能。

 复制代码

```
1 <template>
2   <div class="el-notification">
3     <slot />
4   </div>
5 </template>
6
7 <script>
8
9 </script>
10
11 <style lang="scss">
12 @import '../styles/mixin';
13
14 </style>
```

结合下面的代码可以看到，我们进入到了内部文件 Notification.spec.js 中。下面的测试代码中，我们期待 Notification 组件能够渲染 el-notification 样式类，并且内部能够通过属性 title 渲染标题；message 属性用来渲染消息主体；position 用来渲染组件的位置，让我们的弹窗组件可以显示在浏览器四个角。

 复制代码

```
1 import Notification from "../Notification.vue"
2 import { mount } from "@vue/test-utils"
3
4 describe("Notification", () => {
5
6   it('渲染标题title', () => {
7     const title = 'this is a title'
8     const wrapper = mount(Notification, {
9       props: {
10         title
11       }
12     })
13     expect(wrapper.get('.el-notification__title').text()).toContain(title)
14   })
15
16   it('信息message渲染', () => {
17     const message = 'this is a message'
18     const wrapper = mount(Notification, {
19       props: {
20         message
21       }
22     })
23     expect(wrapper.get('.el-notification__content').text()).toContain(message)
24   })
25 })
```

```
25   it('位置渲染', () => {
26     const position = 'bottom-right'
27     const wrapper = mount(Notification, {
28       props: {
29         position
30       }
31     })
32     expect(wrapper.find('.el-notification').classes()).toContain('right')
33     expect(wrapper.vm.verticalProperty).toBe('bottom')
34     expect(wrapper.find('.el-notification').element.style.bottom).toBe('0px')
35   })
36
37   it('位置偏移', () => {
38     const verticalOffset = 50
39     const wrapper = mount(Notification, {
40       props: {
41         verticalOffset
42       }
43     })
44     expect(wrapper.vm.verticalProperty).toBe('top')
45     expect(wrapper.find('.el-notification').element.style.top).toBe(
46       `${verticalOffset}px`
47     )
48   })
49
50 })
51
52
```

这时候毫无疑问，测试窗口会报错。我们需要进入 `notification.vue` 中实现代码逻辑。

下面的代码中，我们在代码中接收 `title`、`message` 和 `position`，使用 `notification__title` 和 `notification__message` 渲染标题和消息。

[📄 复制代码](#)

```
1  <template>
2    <div class="el-notification" :style="positionStyle" @click="onClickHandler">
3      <div class="el-notification__title">
4        {{ title }}
5      </div>
6
7      <div class="el-notification__message">
8        {{ message }}
9      </div>
10
11     <button
12       v-if="showClose"
13       class="el-notification__close-button">
```

```
14     @click="onCloseHandler"
15   ></button>
16 </div>
17 </template>
18 <script setup>
19   const instance = getCurrentInstance()
20   const visible = ref(true)
21   const verticalOffsetVal = ref(props.verticalOffset)
22
23   const typeClass = computed(() => {
24     return props.type ? `el-icon-${props.type}` : ''
25   })
26
27   const horizontalClass = computed(() => {
28     return props.position.endsWith('right') ? 'right' : 'left'
29   })
30
31   const verticalProperty = computed(() => {
32     return props.position.startsWith('top') ? 'top' : 'bottom'
33   })
34
35   const positionStyle = computed(() => {
36     return {
37       [verticalProperty.value]: `${verticalOffsetVal.value}px`
38     }
39   })
40 </script>
41
42 <style lang="scss">
43   .el-notification {
44     position: fixed;
45     right: 10px;
46     top: 50px;
47     width: 330px;
48     padding: 14px 26px 14px 13px;
49     border-radius: 8px;
50     border: 1px solid #ebeef5;
51     background-color: #fff;
52     box-shadow: 0 2px 12px 0 rgba(0, 0, 0, 0.1);
53     overflow: hidden;
54   }
55 </style>
```

然后我们新增测试代码，设置弹窗是否显示关闭按钮以及关闭弹窗之后的回调函数。我们希望点击关闭按钮之后，就能够正确执行传入的 `onClose` 函数。

[复制代码](#)

```
1 it('set the showClose ', () => {
```

```
2     const showClose = true
3     const wrapper = mount(Notification, {
4       props: {
5         showClose
6       }
7     })
8     expect(wrapper.find('.el-notification__closeBtn').exists()).toBe(true)
9     expect(wrapper.find('.el-icon-close').exists()).toBe(true)
10  })
11
12  it('点击关闭按钮', async () => {
13    const showClose = true
14    const wrapper = mount(Notification, {
15      props: {
16        showClose
17      }
18    })
19    const closeBtn = wrapper.get('.el-notification__closeBtn')
20    await closeBtn.trigger('click')
21    expect(wrapper.get('.el-notification').isVisible()).toBe(false)
22  })
23
24  it('持续时间之后自动管理', async () => {
25    jest.useFakeTimers()
26
27    const wrapper = mount(Notification, {
28      props: {
29        duration: 1000
30      }
31    })
32    jest.runTimersToTime(1000)
33    await flushPromises()
34    expect(wrapper.get('.el-notification').isVisible()).toBe(false)
35  })
```

到这里，Notification 组件测试的主体逻辑就实现完毕了，我们拥有了一个能够显示在右上角的组件，具体效果你可以参考后面这张截图。





进行到这里，距离完成整体设计我们还差两个步骤。

首先，弹窗类的组件都需要直接渲染在 body 标签下面，弹窗类组件由于布局都是绝对定位，如果在组件内部渲染，组件的 css 属性（比如 Transform）会影响弹窗组件的渲染样式，为了避免这种问题重复出现，弹窗组件 Dialog、Notification 都需要渲染在 body 内部。

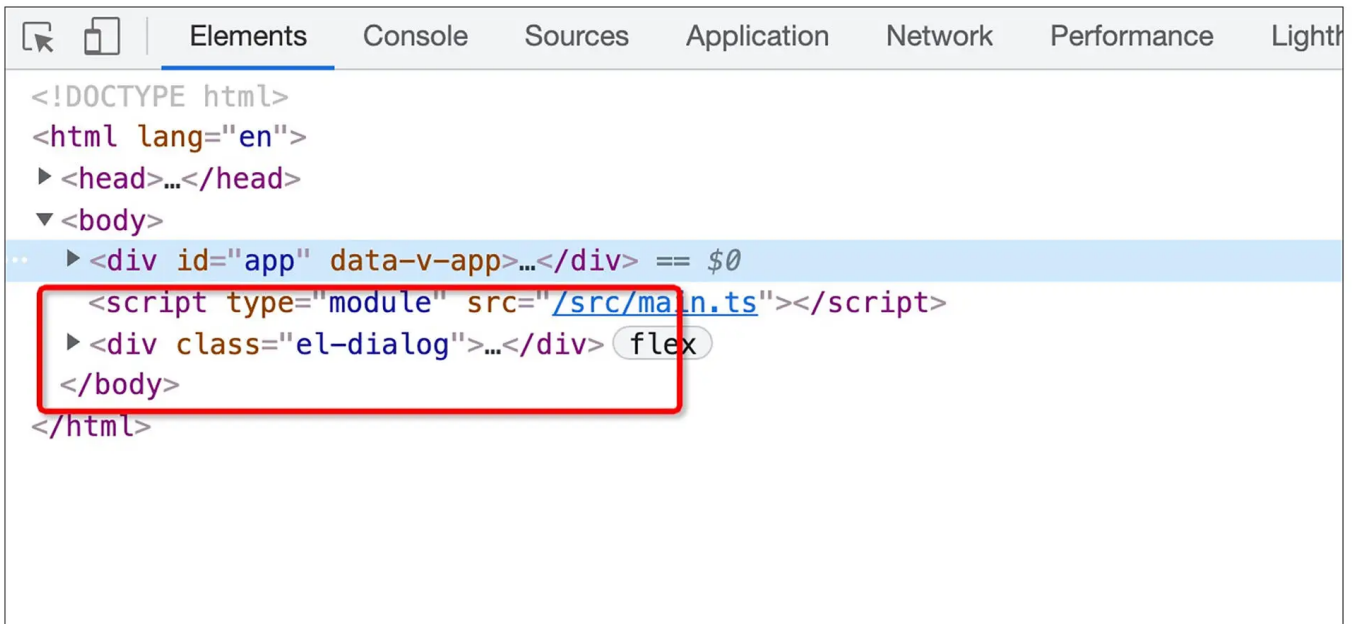
Dialog 组件可以直接使用 Vue3 自带的 Teleport，很方便地渲染到 body 之上。在下面的代码中，我们用 teleport 组件把 dialog 组件包裹之后，通过 to 属性把 dialog 渲染到 body 标签内部。

复制代码

```
1   <teleport
2     :disabled="!appendToBody"
3     to="body"
4   >
5     <div class="el-dialog">
6       <div class="el-dialog__content">
7         <slot />
8       </div>
9     </div>
10  </teleport>
```



这时我们使用浏览器调试窗口，就可以看到 Dialog 标签已经从当前组件移动到了 body 标签内部，如下图所示。



但是 Notification 组件并不会在当前组件以组件的形式直接调用，我们需要像 Element3 一样，能够使用 js 函数动态创建 Notification 组件，给 Vue 的组件提供 Javascript 的动态渲染方法，这是弹窗类组件的特殊需求。

## 组件渲染优化

我们先把测试代码写好，具体如下。代码中分别测试函数创建组件，以及不同配置和样式的通知组件。

复制代码

```
1 it('函数会创建组件', () => {
2   const instanceProxy = Notification('foo')
3   expect(instanceProxy.close).toBeTruthy()
4 })
5
6 it('默认配置 ', () => {
7   const instanceProxy = Notification('foo')
8
9   expect(instanceProxy.$props.position).toBe('top-right')
10  expect(instanceProxy.$props.message).toBe('foo')
11  expect(instanceProxy.$props.duration).toBe(4500)
12  expect(instanceProxy.$props.verticalOffset).toBe(16)
13 })
14 test('字符串信息', () => {
15   const instanceProxy = Notification.info('foo')
16
```

```
17   expect(instanceProxy.$props.type).toBe('info')
18   expect(instanceProxy.$props.message).toBe('foo')
19 })
20 test('成功信息', () => {
21   const instanceProxy = Notification.success('foo')
22
23   expect(instanceProxy.$props.type).toBe('success')
24   expect(instanceProxy.$props.message).toBe('foo')
25 })
```

现在测试写完后还是会报错，因为现在 Notification 函数还没有定义，我们要能通过 Notification 函数动态地创建 Vue 的组件，而不是在 template 中使用组件。

在 [JSX 那一讲](#) 中我们讲过，template 的本质就是使用 h 函数创建虚拟 Dom，如果我们自己想动态创建组件时，使用相同的方式即可。

在下面的代码中我们使用 Notification 函数去执行 createComponent 函数，使用 h 函数动态创建组件，实现了动态组件的创建。

[复制代码](#)

```
1 function createComponent(Component, props, children) {
2   const vnode = h(Component, { ...props, ref: MOUNT_COMPONENT_REF }, children)
3   const container = document.createElement('div')
4   vnode[COMPONENT_CONTAINER_SYMBOL] = container
5   render(vnode, container)
6   return vnode.component
7 }
8 export function Notification(options) {
9   return createNotification(mergeProps(options))
10 }
11
12 function createNotification(options) {
13   const instance = createNotificationByOpts(options)
14   setZIndex(instance)
15   addToBody(instance)
16   return instance.proxy
17 }
```

创建组件后，由于 Notification 组件同时可能会出现多个弹窗，所以我们需要使用数组来管理通知组件的每一个实例，每一个弹窗的实例都存储在数组中进行管理。

下面的代码里，我演示了怎样用数组管理弹窗的实例。Notification 函数最终会暴露给用户使用，在 Notification 函数内部我们通过 createComponent 函数创建渲染的容器，然后通过 createNotification 创建弹窗组件的实例，并且维护在 instanceList 中。

[复制代码](#)

```
1  const instanceList = []
2  function createNotification(options) {
3    ...
4    addInstance(instance)
5    return instance.proxy
6  }
7  function addInstance(instance) {
8    instanceList.push(instance)
9  }
10 ;['success', 'warning', 'info', 'error'].forEach((type) => {
11   Notification[type] = (options) => {
12     if (typeof options === 'string' || isVNode(options)) {
13       options = {
14         message: options
15       }
16     }
17     options.type = type
18     return Notification(options)
19   }
20 })
21
22 // 有了instanceList，可以很方便的关闭所有信息弹窗
23 Notification.closeAll = () => {
24   instanceList.forEach((instance) => {
25     instance.proxy.close()
26     removeInstance(instance)
27   })
28 }
```

最后，我带你简单回顾下我们都做了什么。在正式动手实现弹窗组件前，我们分析了弹窗类组件的风格。弹窗类组件主要负责用户交互的反馈。根据显示的级别不同，它可以划分成不同的种类：既有覆盖全屏的弹窗 Dialog，也有负责提示消息的 Notification。

这些组件除了负责渲染传递的数据和方法之外，还需要能够脱离当前组件进行渲染，**防止当前组件的 css 样式影响布局**。因此 Notification 组件需要渲染到 body 标签内部，而 Vue 提供了 Teleport 组件来完成这个任务，我们通过 Teleport 组件就能把内部的组件渲染到指定的 dom 标签。

之后，我们需要给组件提供 JavaScript 调用的方法。我们可以使用 `Notification()` 的方式动态创建组件，利用 `createNotification` 即可动态创建 Vue 组件的实例。

对于弹窗组件来说可以这样操作：首先通过 `createNotification` 函数创建弹窗的实例，并且给每个弹窗设置好唯一的 `id` 属性，然后存储在数组中进行管理。接着，我们通过对 `createNotification` 函数返回值的管理，即可实现弹窗动态的渲染、更新和删除功能。

## 总结

正文里已经详细讲解和演示了弹窗组件的设计，所以今天的总结我想变个花样，再给你说说 TDD 的事儿。

很多同学会觉得写测试代码要花一定成本，有畏难心理，觉得自己不太会写测试，这些“假想”给我们造成了“TDD 很难实施”的错觉。实际上入门 TDD 并没有这么难。按照我的实践经验来看，先学会怎么写测试，再学习怎么重构，基本上就可以入门写 TDD 了。

就拿我们这讲的实践来说，我们再次应用了**测试驱动开发**这个方式来实现弹窗组件，把整体需求拆分成一个个子任务，逐个击破。根据设计的需求写好测试代码之后，测试代码就会检查我们的业务逻辑有没有实现，指导我们做相应的修改。

咱们的实践过程抽象出来，一共包括四个步骤：写测试 -> 运行测试 (报错) -> 写代码让测试通过 -> 重构的方式。这样的开发模式，今后你在设计组件库时也可以借鉴，不但有助于提高代码的质量和可维护性，还能让代码有比较高的代码测试覆盖率。

## 思考题

最后留一个思考题，现在我们设计的 `Notification` 组件的 `message` 只能支持文本消息，如果想支持传入其他组件，应该如何实现？

欢迎你在评论去分享你的答案，也欢迎你把这一讲的内容分享给你的同事和朋友们，我们下一讲再见。

[生成海报并分享](#)[赞 6](#)[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 22 | 表单：如何设计一个表单组件？

下一篇 24 | 树：如何设计一个树形组件？

## 更多课程推荐

# 跟月影学可视化

## 系统掌握图形学与可视化核心原理

月影

奇虎 360 奇舞团团长

可视化 UI 框架 SpriteJS 核心开发者



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 (7)

[写留言](#)

.K

2021-12-15

测试代码和实现代码有的地方都没对应上，代码有的也没帖全，我估计我们新手跟着这篇文章一个字一个字抄都运行不出来

展开 ∨

作者回复：你好，这一讲主要是演示和剖析element3源码，后面会有ts手把手写一个mini弹窗的加餐的



1

**pzz**

2021-12-14

烧脑

展开 ∨

**pzz**

2021-12-14

这几节课直接垮了

展开 ∨

编辑回复：有啥疑问可以留言讨论哦~

**宇HY**

2021-12-14

git上有组件的完整源码么

展开 ∨

作者回复：源码用的是element3作为演示代码，<https://github.com/hug-sun/element3/blob/master/packages/element3/src/components/Notification/src/Notification.js>

**link**

2021-12-14

怎么又不上ts了

展开 ∨

作者回复：后面的三个组件倾向于演示实际的组件要考虑的因素，就用的是element3的代码作为案例了



费城的二鹏

2021-12-13

思考题的实现方案，可以采用 slot 的方式吗？

展开 ∨



T1M

2021-12-13

最近4讲信息量都好大啊！头疼中.....

展开 ∨

作者回复：慢慢理解哈，不同类型的组件需要的知识点也不太一样

