



动态组件 & 插槽 & 自定义指令



黑马程序员™
www.itheima.com

传智播客旗下
高端IT教育品牌

目录 Contents

◆ 动态组件

◆ 插槽

◆ 自定义指令

1. 什么是动态组件

动态组件指的是动态切换组件的显示与隐藏。



2. 如何实现动态组件渲染

vue 提供了一个内置的 `<component>` 组件，专门用来实现动态组件的渲染。示例代码如下：

```
1 data() {  
2   // 1. 当前要渲染的组件名称  
3   return { comName: 'Left' }  
4 }  
5  
6 <!-- 2. 通过 is 属性，动态指定要渲染的组件 -->  
7 <component :is="comName"></component>  
8  
9 <!-- 3. 点击按钮，动态切换组件的名称 -->  
10 <button @click="comName = 'Left'">展示 Left 组件</button>  
11 <button @click="comName = 'Right'">展示 Right 组件</button>
```

3. 使用 keep-alive 保持状态

默认情况下，切换动态组件时**无法保持组件的状态**。此时可以使用 vue 内置的 `<keep-alive>` 组件保持动态组件的状态。示例代码如下：

```
1 <keep-alive>
2   <component :is="comName"></component>
3 </keep-alive>
```

4. keep-alive 对应的生命周期函数

当组件被缓存时，会自动触发组件的 `deactivated` 生命周期函数。

当组件被激活时，会自动触发组件的 `activated` 生命周期函数。

```
1 export default {  
2   created() { console.log('组件被创建了') },  
3   destroyed() { console.log('组件被销毁了') },  
4  
5   activated() { console.log('Left 组件被激活了!') },  
6   deactivated() { console.log('Left 组件被缓存了!') },  
7 }
```

5. keep-alive 的 include 属性

include 属性用来指定：只有名称匹配的组件会被缓存。多个组件名之间使用英文的逗号分隔：

```
1 <keep-alive include="MyLeft,MyRight">
2   <component :is="comName"></component>
3 </keep-alive>
```

目录 Contents

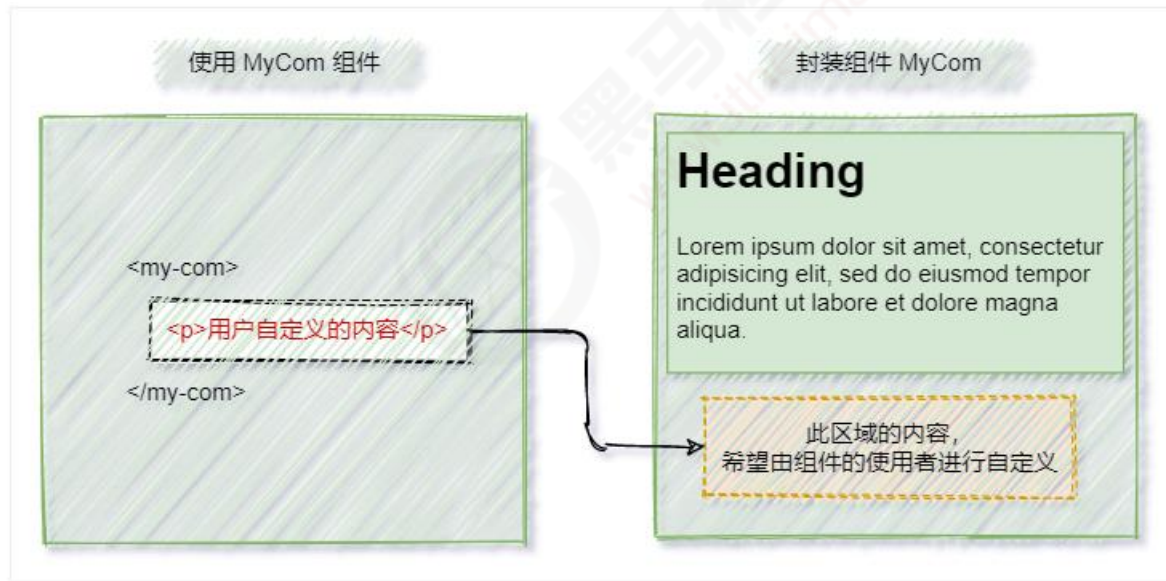
◆ 动态组件

◆ 插槽

◆ 自定义指令

1. 什么是插槽

插槽（Slot）是 vue 为组件的封装者提供的功能。允许开发者在封装组件时，把不确定的、希望由用户指定的部分定义为插槽。



可以把插槽认为是组件封装期间，为用户预留的**内容的占位符**。

2. 体验插槽的基础用法

在封装组件时，可以通过 `<slot>` 元素定义插槽，从而为用户预留内容占位符。示例代码如下：

```
1 <template>
2   <p>这是 MyCom1 组件的第 1 个 p 标签</p>
3   <!-- 通过 slot 标签，为用户预留内容占位符（插槽） -->
4   <slot></slot>
5   <p>这是 MyCom1 组件最后一个 p 标签</p>
6 </template>
```

```
1 <my-com-1>
2   <!-- 在使用 MyCom1 组件时，为插槽指定具体的内容 -->
3   <p>~~~用户自定义的内容~~~</p>
4 </my-com-1>
```

2.1 没有预留插槽的内容会被丢弃

如果在封装组件时没有预留任何 `<slot>` 插槽，则用户提供的任何自定义内容都会被丢弃。示例代码如下：

```
1 <template>
2   <p>这是 MyCom1 组件的第 1 个 p 标签</p>
3   <!-- 封装组件时吗，没有预留任何插槽 -->
4   <p>这是 MyCom1 组件最后一个 p 标签</p>
5 </template>
```

```
1 <my-com-1>
2   <!-- 自定义的内容会被丢弃 -->
3   <p>~~~用户自定义的内容~~~</p>
4 </my-com-1>
```

2.2 后备内容

封装组件时，可以为预留的 `<slot>` 插槽提供**后备内容**（默认内容）。如果组件的使用者没有为插槽提供任何内容，则后备内容会生效。示例代码如下：

```
1 <template>
2   <p>这是 MyCom1 组件的第 1 个 p 标签</p>
3   <slot>这是后备内容</slot>
4   <p>这是 MyCom1 组件最后一个 p 标签</p>
5 </template>
```

3. 具名插槽

如果在封装组件时需要预留多个插槽节点，则需要为每个 `<slot>` 插槽指定具体的 `name` 名称。这种带有具体名称的插槽叫做“具名插槽”。示例代码如下：

```
1 <div class="container">
2   <header>
3     <!-- 我们希望把页头放这里 -->
4     <slot name="header"></slot>
5   </header>
6   <main>
7     <!-- 我们希望把主要内容放这里 -->
8     <slot></slot>
9   </main>
10  <footer>
11    <!-- 我们希望把页脚放这里 -->
12    <slot name="footer"></slot>
13  </footer>
14 </div>
```

注意：没有指定 `name` 名称的插槽，会有隐含的名称叫做“`default`”。

3.1 为具名插槽提供内容

在向具名插槽提供内容的时候，我们可以在一个 `<template>` 元素上使用 `v-slot` 指令，并以 `v-slot` 的参数形式提供其名称。示例代码如下：

```
1 <my-com-2>
2   <template v-slot:header>
3     <h1>滕王阁序</h1>
4   </template>
5
6   <template v-slot:default>
7     <p>豫章故郡，洪都新府。</p>
8     <p>星分翼轸，地接衡庐。</p>
9     <p>襟三江而带五湖，控蛮荆而引瓯越。</p>
10  </template>
11
12  <template v-slot:footer>
13    <p>落款：王勃</p>
14  </template>
15 </my-com-2>
```

3.2 具名插槽的简写形式

跟 v-on 和 v-bind 一样，v-slot 也有缩写，即把参数之前的所有内容 (v-slot:) 替换为字符 #。例如 v-slot:header 可以被重写为 #header:

```
1 <my-com-2>
2   <template #header>
3     <h1>滕王阁序</h1>
4   </template>
5
6   <template #default>
7     <p>豫章故郡，洪都新府。</p>
8     <p>星分翼轸，地接衡庐。</p>
9     <p>襟三江而带五湖，控蛮荆而引瓯越。</p>
10  </template>
11
12  <template #footer>
13    <p>落款：王勃</p>
14  </template>
15 </my-com-2>
```

4. 作用域插槽

在封装组件的过程中，可以为预留的 `<slot>` 插槽绑定 props 数据，这种带有 props 数据的 `<slot>` 叫做“作用域插槽”。示例代码如下：

```
1 <tbody>
2   <!-- 下面的 slot 是一个作用域插槽 -->
3   <slot v-for="item in list" :user="item"></slot>
4 </tbody>
```


4.1 使用作用域插槽

可以使用 **v-slot:** 的形式，接收作用域插槽对外提供的的数据。示例代码如下：

```
1  <my-com-3>
2    <!-- 1. 接收作用域插槽对外提供的数据 -->
3    <template v-slot:default="scope">
4      <tr>
5        <!-- 2. 使用作用域插槽的数据 -->
6        <td>{{scope}}</td>
7      </tr>
8    </template>
9  </my-com-3>
```

4.2 解构插槽 Prop

作用域插槽对外提供的数据对象，可以使用**解构赋值**简化数据的接收过程。示例代码如下：

```
1  <my-com-3>
2    <!-- v-slot: 可以简写成 # -->
3    <!-- 作用域插槽对外提供的数据对象，可以通过“解构赋值”简化接收的过程 -->
4    <template #default="{user}">
5      <tr>
6        <td>{{user.id}}</td>
7        <td>{{user.name}}</td>
8        <td>{{user.state}}</td>
9      </tr>
10   </template>
11 </my-com-3>
```

目录 Contents

- ◆ 动态组件
- ◆ 插槽
- ◆ 自定义指令



1. 什么是自定义指令

vue 官方提供了 v-text、v-for、v-model、v-if 等常用的指令。除此之外 vue 还允许开发者自定义指令。



黑马程序员
www.itheima.com

2. 自定义指令的分类

vue 中的自定义指令分为两类，分别是：

- 私有自定义指令
- 全局自定义指令



3. 私有自定义指令

在每个 vue 组件中，可以在 `directives` 节点下声明私有自定义指令。示例代码如下：

```
1 directives: {  
2   color: {  
3     // 为绑定到的 HTML 元素设置红色的文字  
4     bind(el) {  
5       // 形参中的 el 是绑定了此指令的、原生的 DOM 对象  
6       el.style.color = 'red'  
7     }  
8   }  
9 }
```



4. 使用自定义指令

在使用自定义指令时，需要加上 **v-** 前缀。示例代码如下：

```
1 <!-- 声明自定义指令时，指令的名字是 color -->
2 <!-- 使用自定义指令时，需要加上 v- 指令前缀 -->
3 <h1 v-color>App 组件</h1>
```

5. 为自定义指令动态绑定参数值

在 template 结构中使用自定义指令时，可以通过等号（=）的方式，为当前指令动态绑定参数值：

```
1 data() {  
2   return {  
3     color: 'red' // 定义 color 颜色值  
4   }  
5 }  
6  
7 <!-- 在使用指令时，动态为当前指令绑定参数值 color -->  
8 <h1 v-color="color">App 组件</h1>
```


6. 通过 **binding** 获取指令的参数值

在声明自定义指令时，可以通过形参中的**第二个参数**，来接收指令的参数值：

```
1 directives: {  
2   color: {  
3     bind(el, binding) {  
4       // 通过 binding 对象的 .value 属性，获取动态的参数值  
5       el.style.color = binding.value  
6     }  
7   }  
8 }
```



7. update 函数

bind 函数只调用 1 次：当指令第一次绑定到元素时调用，当 DOM 更新时 bind 函数不会被触发。update 函数会在每次 DOM 更新时被调用。示例代码如下：

```
1 directives: {  
2   color: {  
3     // 当指令第一次被绑定到元素时被调用  
4     bind(el, binding) {  
5       el.style.color = binding.value  
6     },  
7     // 每次 DOM 更新时被调用  
8     update(el, binding) {  
9       el.style.color = binding.value  
10    }  
11  }  
12 }
```



8. 函数简写

如果 `insert` 和 `update` 函数中的逻辑完全相同，则对象格式的自定义指令可以简写成函数格式：

```
1 directives: {  
2   // 在 insert 和 update 时，会触发相同的业务逻辑  
3   color(el, binding) {  
4     el.style.color = binding.value  
5   }  
6 }
```

9. 全局自定义指令

全局共享的自定义指令需要通过 “`Vue.directive()`” 进行声明，示例代码如下：

```
1 // 参数1: 字符串, 表示全局自定义指令的名字
2 // 参数2: 对象, 用来接收指令的参数值
3 Vue.directive('color', function(el, binding) {
4   el.style.color = binding.value
5 })
```



总结

- ① 能够掌握 **keep-alive** 元素的基本使用
 - `<keep-alive>` 标签、`include` 属性
- ② 能够掌握**插槽**的基本用
 - `<slot>` 标签、具名插槽、**作用域插槽**、后备内容
- ③ 能够知道如何**自定义指令**
 - 私有自定义指令 `directives: {}`
 - **全局自定义指令** `Vue.directive()`



传智播客旗下高端IT教育品牌