第5章

Vue 组件插槽机制的深入解析

本章深入探讨 Vue 中的组件插槽机制，专注于如何通过插槽实现父组件向子组件传递内容的功能。在本章中读者将学习到匿名插槽、具名插槽、作用域插槽及其动态和具名作用域插槽，并了解它们在实际开发中的应用场景。此外，还会讨论插槽内容的作用域以及如何使用默认内容来增强组件的复用性和灵活性。

5.1 插槽基础

5.1.1 父组件向子组件传递标签体

1. 绑定组件（标签）属性

在 Vue 中，父组件可以通过绑定子标签的属性（即 props）将数据传递给子组件，而子组件通过定义相应的 props 选项接收这些数据。这种方式非常适合传递具体的 JavaScript 值，如字符串、数字、对象等。

2. 传递标签体内容

除了通过属性传递数据外，父组件还可以直接通过子组件的标签体向子组件传递 HTML 或文本内容。这种情况下，子组件使用＜slot＞元素作为占位符，用于接收并渲染来自父组件的内容。

标签的三要素标签名、标签属性、标签体应用都在子标签中体现，**通过标签属性传递数据（子组件 props 接收），通过标签体传递数据（子组件 slot 接收）**，如图 5-1 所示。

父组件中使用子组件

```
模板中使用自定义标签：
传递子组件的标签体 Click me!
<fancy-button>
Click me!
</fancy-button>`
```

子组件的定义

```
子组件在 slot 中接收父组件
的标签体
<button>
<slot></slot>
</button>
```

子组件渲染结果

```
接收到父组件传递的标签体 Click me！
<button>
Click me!
</button>
```

图 5-1 父组件向子组件传递标签体 slot

3. 使用场景

在之前的章节中，我们已经了解到组件能够接收任意类型的 JavaScript 值作为 props，但组件要如何接收模板内容呢？在某些场景中，可能想要为子组件传递一些模板片段，让子组件在它们的组件中渲染这些片段，这时候就可以用插槽。

5.1.2 插槽内容与出口

在前面为了方便理解把插槽解释为父组件向子组件传递标签体，规范的说法为插槽内容（传递的标签体）与出口（接收标签体）。

Vue 的插槽（Slots）机制极大地增强了组件的灵活性和复用性，允许父组件决定子组件的部分或全部内容。这一机制围绕两个核心概念展开。

1. 插槽内容

- **概念**：插槽内容指的是父组件向子组件传递的实际数据或模板片段，可以是**任意的 HTML、文本或 Vue 组件**。
- **应用场景**：例如，在一个通用的布局组件中，父组件可以插入具体的页面内容，使布局组件适用于多种页面结构。

2. 插槽出口

- **定义**：插槽出口是子组件中通过＜slot＞标签定义的占位符，用于指示父组件内容应该插入的位置。
- **作用**：确保父组件提供的内容准确地嵌入子组件的指定位置，保持组件设计的灵活性和模块化。

5.2 无名插槽

无名插槽是最基础的插槽形式，它允许父组件向子组件传递内容，而无须指定特定的名称。这种方式简化了父子组件间的内容传递，并且增强了组件的灵活性。

5.2.1 定义与使用无名插槽

props 仅传递标签属性，利用 solt 把标签体传给子组件。

1. 子组件定义插槽出口

在子组件中，可以通过＜slot＞标签定义一个插槽出口，用于接收来自父组件的内容。以下是一个简单的示例，展示了如何创建一个带有无名插槽的子组件。

```
1.  //注册一个组件--子组件
2.  app.component("FancyButton", {
3.      template: `
4.      <button class="fancy-btn">
5.        <slot></slot><!-- 插槽出口 -->
6.      </button>`
7.  })
```

2. 父组件定义插槽内容

父组件可以通过将内容直接放入子组件的标签体中来定义插槽内容。这些内容会被传

递给子组件，并渲染到由＜slot＞标签标记的位置。

```
1.  //创建一个 app,传入父组件——父组件引入子组件
2.  const app = Vue.createApp({
3.      template: `
4.          <fancy-button>
5.            <!-- 插槽内容 -->
6.            Click me!
7.          </fancy-button>`
8.  })
```

3. 渲染结果

（1）父组件引用子标签，将子标签标签体的内容“Click me”（插槽内容）传入子标签＜slot＞插槽出口中，如图 5-2 所示。

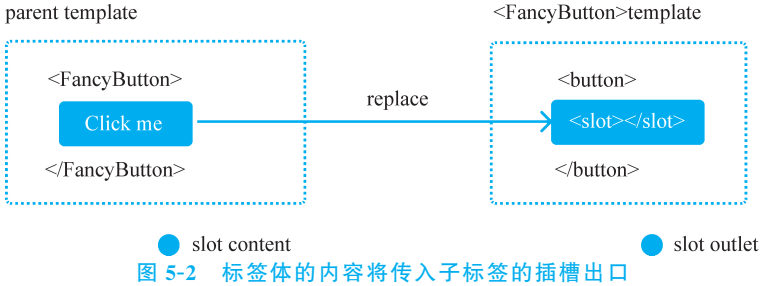


图 5-2　标签体的内容将传入子标签的插槽出口

（2）最终渲染出的 DOM 如下：

```
<button class="fancy-btn">Click me!</button>
```

通过使用插槽，＜FancyButton＞仅负责渲染外层的＜button＞（以及相应的样式），而其内部的内容由父组件提供。运行结果如图 5-3 所示，代码文件为 introduction\component\solt\outlet.html。

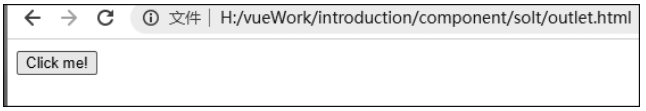


图 5-3　插槽内容与出口示例的运行结果

5.2.2　渲染作用域

插槽内容继承自父组件的作用域，这意味着可以在插槽内容中访问父组件的数据和方法。这是因为插槽内容实际上是在父组件模板中定义的，所以它遵循 JavaScript 的词法作用域规则。

1. 父组件定义插槽内容

父组件中引入子组件 fancy-button，定义子组件的插槽内容，就是标签体中的内容。＜fancy-button＞**插槽内容引用了父标签的数据 message**，＜fancy-button＞插槽的内容将传入子标签的插槽出口。

```
1.  const app = Vue.createApp({
2.      data(){return {message:"Click me!"}},
```

```
    template: `
      <fancy-button>
          <!-- 插槽内容 -->
          {{message}}
      </fancy-button>`
})
```

2. 插槽内容无法访问子组件的数据

Vue 模板中的表达式只能访问其定义时所处的作用域，这和 JavaScript 的词法作用域规则是一致的。父组件模板中的表达式只能访问父组件的作用域，因此它不能直接访问子组件内部的数据或方法；子组件模板中的表达式只能访问子组件的作用域。

3. 运行结果

运行结果如图 5-4 所示，最终页面上展示了一个带有文本“Click me!”的按钮，该文本是由父组件提供的 message 数据动态生成的。此行为确保了即使子组件内部发生变化，只要父组件正确地提供了所需的内容，用户界面就能保持一致。代码文件为 introduction\component\solt\slot-scope-parent-.html。



图 5-4　插槽内容访问父组件数据

5.2.3　默认内容

默认内容是指当父组件没有提供特定内容时，子组件可以提供的后备内容。这种方式增强了组件的灵活性，确保即使在父组件未定义内容的情况下，子组件也能展示合理的信息。

1. 默认内容

在子组件插槽出口中填入内容作为默认内容：

```
//注册一个组件--子组件
app.component("FancyButton", {
    template: `
    <button class="fancy-btn">
      <slot>
        <!-- 默认内容 -->
          Click me!!!
      </slot> <!-- 插槽出口 -->
    </button>`
})
```

2. 父组件无插槽内容的时候

父组件中引入子组件 fancy-button，子组件的插槽内容为空的时候（没有标签体），**子组件用插槽出口中的默认内容渲染**：

```
//创建一个 app，传入父组件——父组件引入子组件
const app = Vue.createApp({
    template: `
```

```
        <fancy-button/>
        `
})
```

3. 运行结果

插槽默认内容运行结果如图 5-5 所示，子组件将渲染按钮，并显示默认文本“Click me!!!”。代码文件为 introduction\component\solt\solt-default.html。

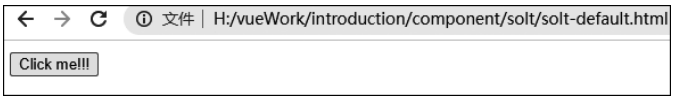


图 5-5　插槽默认内容

5.3　具名插槽

5.3.1　定义与使用具名插槽

具名插槽允许多个插槽存在于同一个子组件中，并通过命名来区分它们。这在构建复杂布局或需要多个插入点的场景中非常有用。

1. 定义子组件插槽出口名

在子组件模板中，可以通过<slot name="slot-name"></slot>语法定义具名插槽。在子组件 mylist 模板中，定义两个插槽出口，并且命名了插槽出口。

```
//定义 slot 插槽
app.component("mylist", {
    template: `
    <div>
        <slot name="title-slot"></slot>
        <ul>
        <slot name="items-slot"></slot>
        </ul>
    </div>`
});
```

2. 父组件插槽内容标记插槽出口名

父组件可以使用<template v-slot：name>或其简写<template #name>来为具名插槽提供内容。这样做不仅避免了额外 DOM 节点的生成，还帮助组织和管理复杂的插槽内容。

当使用具名插槽并将内容插入父组件时，**使用<template>标签作为包装器**。这样做有以下几个原因。

- **避免额外 DOM 节点**：使用<template>标签包裹插槽内容可以避免在渲染时生成不必要的额外 DOM 元素。Vue 不会渲染<template>本身，仅渲染它的内容。
- **组织与逻辑分组**：<template>可以作为逻辑上的分组元素，帮助组织和管理插槽内容，特别是当插槽内容较为复杂，包含多个元素或条件渲染逻辑时。
- **绑定属性与事件**：虽然在这个特定的例子中没有体现，但当需要在插槽内容上绑定属性或事件处理器时，使用<template>作为容器可以提供一个正确的上下文。

3. 父组件示例

这里传递的标签体(插槽内容)不是字符串,而是一个模板片段。

(1) 父组件引入子组件<mylist>,内含两个具名插槽内容。

第一个插槽内容名为 title-slot,直接渲染了一个<list-title>组件,并将 title 属性绑定到父组件的 title 数据属性。

第二个插槽名内容为 items-slot,使用 v-for 指令遍历 items 数组,并为每个 item 渲染一个<list-items>组件,将每个 item 作为内容传递。

```
const app = Vue.createApp({
    data() {
        return {
            title: "自定义列表",
            items: ['Web 框架技术', 'Web 框架实战', '前端 Vue 实战']
        }
    },
    template: `
      <mylist>
        <template #title-slot>
            <list-title  :title="title"></list-title>
        </template>
        <template #items-slot>
            <list-items  v-for="item in items" :item="item"></list-items>
        </template>
      </mylist>`
})
```

(2) 父组件在子组件<mylist>插槽内容中引入了两个组件,两个子组件定义如下。其中定义了两个展示组件：list-title 和 list-items,分别接收 title 和 item 作为属性,并在模板中展示它们。这两个子组件在后面例子中也会用到,后面就不再强调这两个子组件了。

```
//组件标签
app.component("list-title", {
    //组件属性
    props: ['title'],
    //组件模板
    template: '<div>{{title}}</div>'
});
app.component("list-items", {
    props: ['item'],
    template: '<li>{{item}}</li>'
});
```

3. 代码框架

代码框架如下,代码文件为 introduction\component\solt\slot-scoped-named.html。

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
```

```
    <title>Title</title>
</head>

<body>
    <!--View 层-->
    <div id="app">
    </div>

    <!--1.导入 Vue.js-->
    <script src="https://cdn.staticfile.org/vue/3.4.15/vue.global.js"></
script>
    <script>
        const app = Vue.createApp({
            …
        })

        //定义 slot 插槽
        app.component("mylist", {
            …
        });
        //组件标签
        app.component("list-title", {
          …
        });
        app.component("list-items", {
          …
        });
        //2.挂载 app
        const vm = app.mount("#app")
    </script>
</body>

</html>
```

4. 运行结果

具名插槽示例的运行结果如图 5-6 所示。

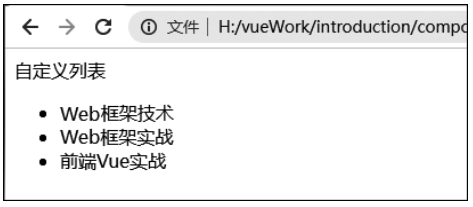


图 5-6　具名插槽示例的运行结果

在 Vue 中，使用组件来组织页面和组织代码，类似于搭积木，每一个组件都是一个积木，使用一些相同或者不同组件就能搭建出想要的页面。slot（插槽）是组件功能的重要组成部分，插槽必须用于组件才有意义。它为组件提供了对外的接口，允许从组件外部传递内容，并将这部分内容放置到指定的位置。

5.3.2 动态插槽名

动态插槽名允许根据数据动态决定使用哪个具名插槽。这是通过 v-slot:[variableName]语法实现的，其中 variableName 是一个包含插槽名称的变量。

动态插槽使用示例如下。代码文件为 introduction\component\solt\slot-named-dynamic.html。

```
1.  const app = Vue.createApp({
2.      data() {
3.          return {
4.              title: "自定义列表",
5.              items: ['Web 框架技术', 'Web 框架实战', '前端 Vue 实战'],
6.              titleSlot:"title-slot",
7.              itemsSlot:"items-slot"
8.          }
9.      },
10.     template: `
11.       <mylist>
12.         <template v-slot:[titleSlot]>
13.             <list-title  :title="title"></list-title>
14.         </template>
15.         <template v-slot:[itemsSlot]>
16.             <list-items  v-for="item in items" :item="item"></list-items>
17.         </template>
18.     </mylist>`
19. })
```

＜template v-slot:[titleSlot]＞：这里的 titleSlot 值从 data 中获取，动态指定了一个插槽名称，用于插入标题组件＜list-title＞。标题组件接收来自父组件的 title 属性。

＜template v-slot:[itemsSlot]＞：同理，itemsSlot 也是动态插槽名称，用于插入项目列表组件＜list-items＞。列表组件通过 v-for 循环遍历 items 数组，为每个项目创建一个列表项。

5.4 作用域插槽

作用域插槽(Scoped Slots)是 Vue.js 框架中的一项高级功能，它允许父组件访问子组件中的数据或方法，并基于这些数据动态地控制子组件部分内容的渲染。这一特性增强了组件的复用性和灵活性，同时保持了视图和数据的分离。

5.4.1 作用域插槽概述

1. 作用域插槽

在 Vue 中，默认情况下，插槽内容只能访问父组件的数据作用域。然而，在某些场景下，插槽的内容可能需要同时使用父组件和子组件的数据。为此，Vue 引入了作用域插槽的概念，使得子组件可以在渲染时将一部分数据提供给插槽内容，从而实现更复杂的数据交互和展示逻辑。特点如下。

- **数据流动**：打破了 Vue 单向数据流的规则，允许数据从子组件流向父组件(仅限于

插槽内部)。

- **灵活性**：父组件可以完全控制如何展示子组件传来的数据，增强了组件的可定制性。
- **复用性**：子组件无须关心数据如何展示，只须提供数据和逻辑，增强了组件的复用价值。
- **解耦**：数据和展示分离，使得用户界面逻辑更加清晰，易于维护。

2. 基本用法

1) 子组件定义插槽出口

在子组件中，**通过 v-bind 将需要传递给父组件的数据绑定到插槽上**。通常，这些数据会被封装成一个对象传递。下例中 mylist 子组件定义了数据 items，定义了插槽出口，并在插槽出口绑定了子组件数据**<slot v-bind="{items }"></slot >**。

```
1.  //定义 slot 插槽
2.  app.component("mylist", {
3.      data() {
4.          return {
5.              items: ['Web 框架技术', 'Web 框架实战', '前端 Vue 实战'],
6.          }
7.      },
8.      template: `
9.      <div>
10.         <ul>
11.         <slot  v-bind="{items}"></slot>
12.         </ul>
13.     </div>`
14. });
```

2) 父组件定义插槽内容

在父组件中，**通过 v-slot 或简写的 # 语法来定义插槽模板**，并通过**解构赋值来接收子组件传递的数据**。下例中，在父组件中引入了子组件 mylist，插槽内容用 template 包裹，通过 v-slot 定义插槽模板，并解构出子组件传来的数据<template v-slot="{items}">。

```
1.  const app = Vue.createApp({
2.      data() {
3.          return{ title: "自定义列表"}
4.      },
5.
6.      template: `
7.        <list-title  :title="title"></list-title>
8.        <mylist>
9.          <template v-slot="{items}">
10.             <list-items  v-for="item in items" :item="item"></list-items>
11.         </template>
12.       </mylist>`
13. })
14.
```

3) 运行结果

运行效果和具名插槽示例一样，具体如图 5-6 所示。区别在于数据提供不一样，这里数

据 items 由子组件提供，具名插槽示例中数据 items 由父组件提供。另外为了演示无具名插槽，把 list-title 子标签移出子标签 mylist 插槽内容。代码文件为 introduction\component\solt\slot-Scoped.html。

5.4.2 具名作用域插槽

具名作用域插槽结合了具名插槽和作用域插槽的优点，既允许定义多个具名插槽，又能在每个插槽中传递子组件的数据。

1. 子组件定义插槽出口

在子组件中定义具名插槽，并绑定子组件的数据。组件模板 template 属性定义了两个具名插槽。

第一个插槽＜slot name＝"title-slot" v-bind＝"{title}"＞＜/slot＞，具名 title-slot，通过 **v-bind＝"{title}"将组件的 title 属性传递给插槽**，供父组件使用。

第二个插槽＜slot name＝"items-slot" v-bind＝"{items}"＞＜/slot＞，具名 items-slot，**通过 v-bind＝"{items}"将组件的 items 数组传递给插槽**，同样供父组件自定义渲染列表项内容。

```
1.       //定义 slot 插槽
2.  app.component("mylist", {
3.      data() {
4.          return {
5.              title: "自定义列表",
6.              items: ['Web 框架技术', 'Web 框架实战', '前端 Vue 实战'],
7.          }
8.      },
9.      template: `
10.     <div>
11.         <slot name="title-slot" v-bind="{title}"></slot>
12.         <ul>
13.         <slot name="items-slot" v-bind="{items}"></slot>
14.         </ul>
15.     </div>`
16. });
```

2. 父组件中定义插槽内容

父组件模板中的子组件插槽内容＜template＞标签内使用 v-slot 指令的**简写法（＃）指定了插槽名称**，并且可以直接在模板中**解构传递的对象**。代码中，**＃title-slot＝"{title}"和＃items-slot＝"{items}"正确地接收了来自 mylist 组件的数据**。并通过**花括号解构了传递的数据对象，如{title}和{items}**。

```
1.  const app = Vue.createApp({
2.
3.      template: `
4.        <mylist>
5.          <template #title-slot={title}>
6.              <list-title  :title="title"></list-title>
7.          </template>
8.          <template #items-slot="{items}">
```