Vue 技能进阶：使用设计模式写出优雅的前端代码

本文针对 Vue 中如何控制组件子树之外的东西，探讨了四种解决方案，并展示了每种解决方案的优缺点。希望读者能从中受到启发。

问你个问题，以前你可能从来没想过：

> 有没有办法从子组件填充父组件插槽？

最近一位同事问我这个问题，答案很简单：

可以。

但我找出的解决方案可能与你想到的有很大区别。

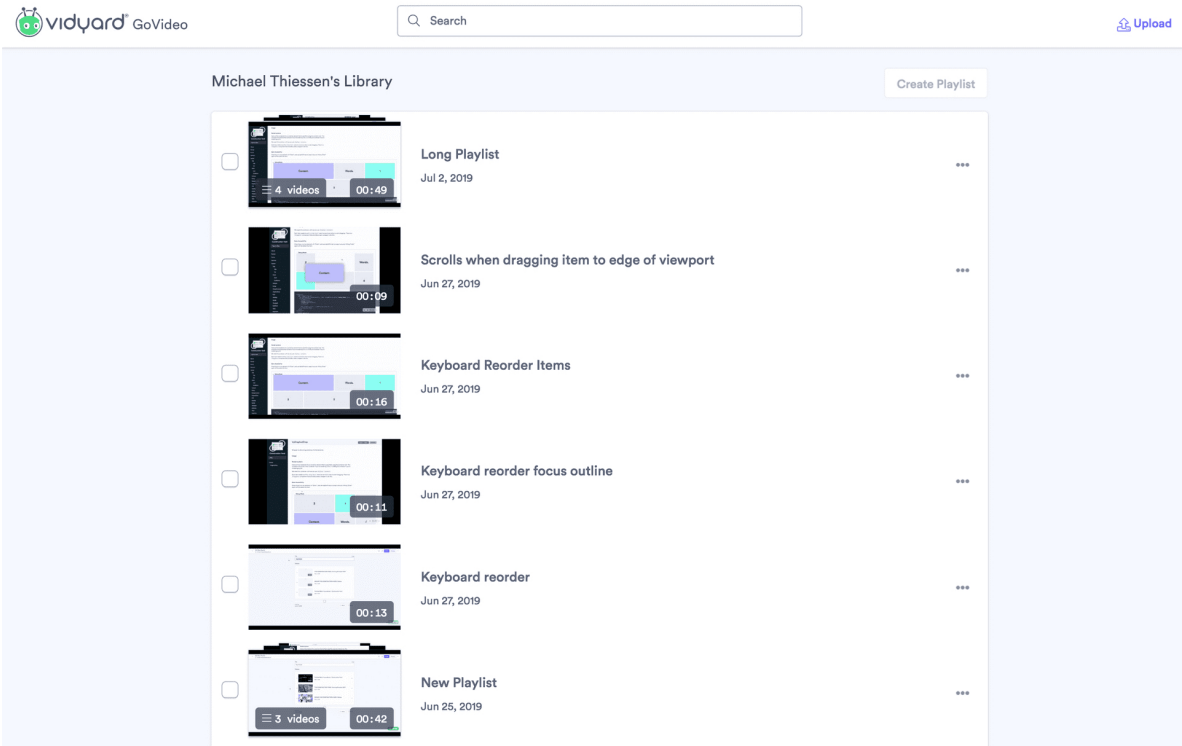
我一开始想到的法子非常糟糕，然后试了好几次才找出来这个问题的最佳方案，起码我觉得够好了。

这是一个棘手的 Vue 架构问题，但也是一个非常有趣的问题。

本文将逐一介绍这些解决方案，并分析它们的缺陷所在。最后我们会探讨最佳方案。

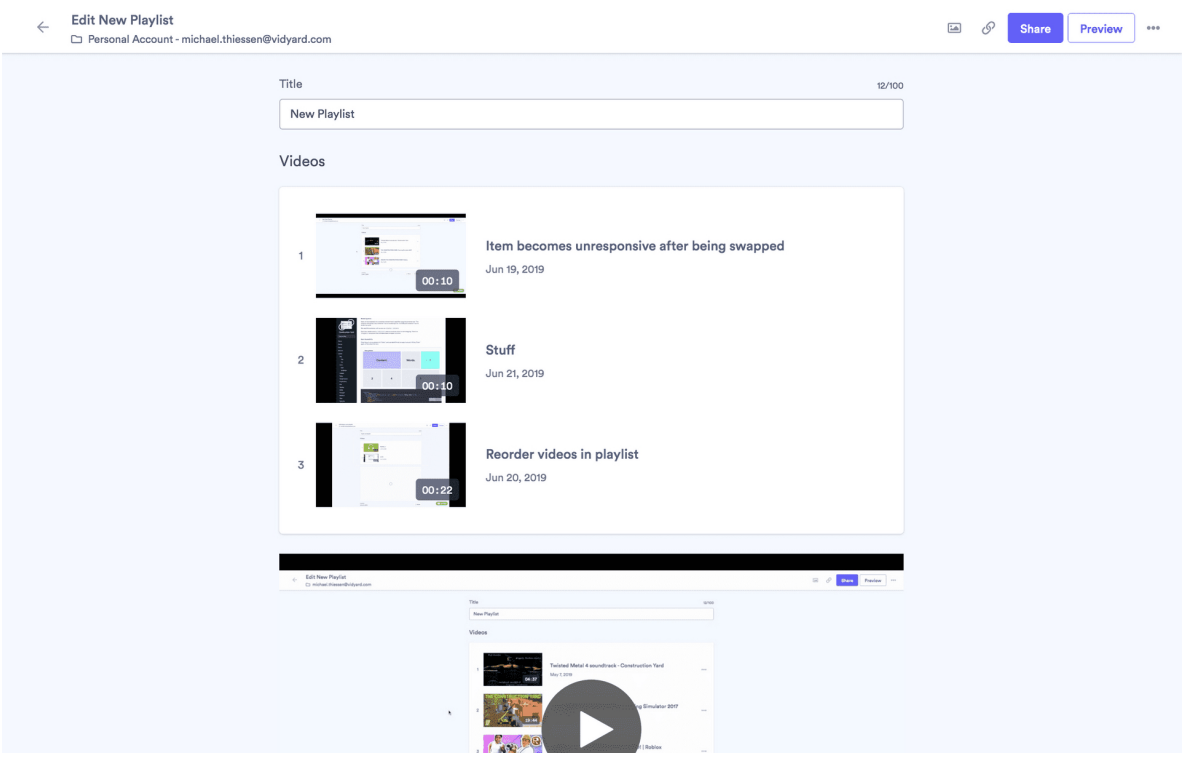
不过一开始我们是怎么遇到这个问题的呢？

为什么提出这个复杂的问题？



在我们的应用程序中有一个顶栏，其中包含各种按钮、一个搜索栏和其他一些控件。

它显示的内容根据你所在的页面略有差异，因此我们需要一种按页配置它的方法。



为此，我们希望每个页面都能配置顶栏。

看起来很简单，但这里有一个问题：这个顶栏（我们称之为 ActionBar）实际上是主布局骨架的一部分，它长成这样：

```
<template>
  <div>
    <FullPageError />
    <ActionBar />
    <App />
  </div>
</template>
```

这里根据你所在的页面 / 路径动态注入 App。

ActionBar 有一些插槽，我们可以用它来作配置。但我们如何从 App 组件控制这些插槽呢？

定义问题

首先应该搞清楚我们究竟想要解决什么问题。

我们来看一个组件，它包含一个子组件和一个插槽：

```
// Parent.vue
<template>
<div>
<Child />
<slot />
</div>
</template>
```

我们可以像这样填充 Parent 的插槽：

```
// App.vue
<template>
  <Parent>
    <p>This content goes into the slot</p>
  </Parent>
</template>
```

这里没有什么太花哨的...

填充子组件的插槽很容易，这就是常见的插槽用法。

但有没有办法可以从 Child 组件内部控制进入 Parent 组件 slot 的内容呢?

更一般地说:

> 我们可以让一个子组件来填充父组件的插槽吗?

我们来看看我想出的第一个解决方案。

Props down，events up

看到这个问题，我的第一反应就是我们经常说的一句口头禅:

> Props down，events up

数据只能使用 props 才能通过组件树向下流动。你只能靠发送 events 来回传数据给树。

也就是说如果我们需要让子组件与父组件通信，就会使用事件。

所以我们会使用事件将内容传递到 ActionBar 插槽!

在每个应用程序组件中，我们都需要执行以下操作:

```
import SlotContent from './SlotContent';

export default {
  name: 'Application',
  created() {
    // As soon as this component is created we'll emit our events
  this.$emit('slot-content', SlotContent);
  }
};
{1}
```

我们将要放入插槽的内容打包成 SlotContent 组件（名称不重要）。一旦创建了应用程序组件，我们就会发出 slot-content 事件，并传递我们想要使用的组件。

我们的骨架组件如下所示:

```
<template>
  <div>
    <FullPageError />
    <ActionBar>
      <Component :is="slotContent" />
    </ActionBar>
    <App @slot-content="component => slotContent = component" />
  </div>
</template>
```

它将侦听该事件，并将 slotContent 设置为我们的 App 组件发送来的内容。然后我们使用内置 Component 动态渲染该组件。

通过事件传递组件感觉很奇怪，因为它并不是我们的应用程序中“发生”的事情。这只是应用程序的一种设计方式。

还好有一种方法可以不用这么多事件。

寻找其他选项

由于 Vue 组件就是 Javascript 对象，我们可以添加想要的任何属性。

我们可以将它作为字段添加到组件中，这样就无需使用事件传递插槽内容了：

```
import SlotContent from './SlotContent';

export default {
  name: 'Application',
  slotContent: SlotContent,
  props: { /***/ },
  computed: { /***/ },
};
```

我们得稍微改变一下骨架中访问此组件的方式：

```
<template>
  <div>
    <FullPageError />
    <ActionBar>
      <Component :is="slotContent" />
    </ActionBar>
    <App />
  </div>
</template>
```

```
import App from './App';
import FullPageError from './FullPageError';
import ActionBar from './ActionBar';

export default {
name: 'Scaffold',
components: {
App,
FullPageError,
ActionBar,
}
data() {
return {
slotContent: App.slotContent,
}
},
};
```

这更像是静态配置，更好看更整洁。

但这样还是不对。

理想情况下，我们不会在代码中混合使用范例，并且所有内容都会是声明式的。

但在这里，我们不是将组件组合在一起，而是将它们作为 Javascript 对象传递。

我们最好用正常的 Vue 方式让想要的内容出现在插槽里。

考虑一下 portal

这里就可以用到 portal 了。

它们完全按你期望的那样行事。你可以把任何内容从某处传送到另一处。在本文的示例中，我们是从其他地方的 DOM 中的一个位置“传送”元素过来。

无论组件树是什么样的，我们都能够控制组件在 DOM 中渲染的位置。

例如，假设我们想要填充模态。但是我们的模态必须在页面的根部渲染，这样才能正确地覆盖它。首先，我们将指定想要放在模态中的内容：

```
<template>
<div>
<!-- Other components -->
<Portal to="modal">
Rendered in the modal.
</Portal>
</div>
</template>
```

然后在我们的模态组件中放另一个 portal 来渲染该内容：

```
<template>
<div>
<h1>Modal</h1>
<Portal from="modal" />
</div>
</template>
```

这肯定是一种改进，因为现在我们实际上是在编写 HTML 而不是简单地传递对象。这种方法更具声明性，并且更容易看到应用程序中发生了什么。

但有些情况下看到应用内发生了什么并不那么容易。

因为 portal 在幕后做了一些手脚来渲染不同位置的元素，所以它完全打破了 Vue 中渲染 DOM 的机制。看起来你在正常渲染元素，但它根本不能正常工作。这可能会带来很多麻烦和陷阱。

这里还有一个大问题，我们稍后会讲。

很显然，至少在将组件添加到 $options 属性时事情是不一样的。

我认为还有更好的方法。

状态提升

“状态提升”是一个前端开发圈子所用的术语。

它的意思是你将状态从子组件移动到父组件或祖父组件。你是顺着组件树向上移动。

这会对应用程序的体系结构产生深远的影响。而对于我们的目的来说，它实际上开辟了一个完全不同的，更简单的解决方案。

这里的“状态”是我们试图传递到 ActionBar 组件槽中的内容。

但是该状态包含在 Page 组件里，我们无法将页面特定的逻辑移动到布局组件中。我们的状态必须保持在我们动态渲染的 Page 组件中。

所以我们必须提升整个 Page 组件才能提升状态。

目前我们的 Page 组件是 Layout 组件的子组件：

```
<template>
<div>
<FullPageError />
<ActionBar />
<Page />
</div>
</template>
```

我们得调换它们的关系才能提升它，让 Layout 组件成为 Page 组件的子组件。我们的 Page 组件看起来像这样：

```
<template>
<Layout>
<!-- Page-specific content -->
</Layout>
</template>
```

我们的 Layout 组件现在看起来像这样，这里可以使用插槽来插入页面内容：

```
<template>
<div>
<FullPageError />
<ActionBar />
<slot />
</div>
</template>
```

但这并不能让我们定制任何内容。我们必须在 Layout 组件中添加一些命名槽，以便传入应该放入 ActionBar 的内容。

最简单的方法是使用一个槽来完全替换 ActionBar 组件：

```
<template>
<div>
<FullPageError />
<slot name="actionbar">
<ActionBar />
</slot>
<slot />
</div>
</template>
```

这样，如果你没有指定“actionbar”插槽，我们将用默认的 ActionBar 组件。但你仍然可以使用自己自定义的 ActionBar 配置覆盖此插槽：

```
<template>
<Layout>
<template #actionbar>
<ActionBar>
<!-- Custom content that goes into the action bar -->
</ActionBar>
</template>
<!-- Page-specific content -->
</Layout>
</template>
```

对我来说这是一条理想的路径，但它确实需要你重构页面布局。这可能是一项艰巨的任务，具体取决于你的应用程序的构建方式。

如果你用不了这种方法，下一个备选方案应该是 2 号，也就是使用 $options 属性。它是最干净的方法，最容易被阅读代码的人理解。

还可以更简单一些

前面我们定义问题时是用比较宏观的方式来描述的：

> 我们可以让子组件来填充父组件的插槽吗？

但实际上这个问题与具体的 props 无关。更简单来说是让一个子组件控制在它自己的子树之外渲染的内容。

把这个问题根据最常见的场景改写一下：

> 组件控制在其子树外渲染的内容的最佳方法是什么？

换成这个角度来审视我们之前的方案，就能得出一些有趣的新观点。

将事件发送给父组件

因为我们的组件不能直接影响在它的子树外发生的事情，所以我们要找这样的一个组件；其子树包含我们试图控制的目标元素。

然后让它为我们解决问题就行了。

静态配置

我们只要向其他组件提供必要的信息就行了，不用主动要求其他组件代表我们做事。

Portal

前面这三种方法有一种共有模式，

所以可以做出如下断言：

组件无法控制其子树之外的东西。

（它的证明留给读者练习）

所以这里的方法都是用某种手段让另一个组件处理我们的需求，并控制我们真正感兴趣的那个元素。

之所以 Portal 在这里表现更好，是因为它允许我们将所有这些通信逻辑封装到单独的组件中。

状态提升

终于轮到真正不一样的方案了，状态提升是比之前那几个方法更简单、更强大的技术。

我们的主要障碍在于我们想要控制的东西是在子树之外。

最简单的解决方案：

将目标元素移动到子树中，这样我们就可以控制它了！

状态提升——以及操纵该状态的逻辑——让我们可以拥有更大的子树，并把我们的目标元素包含在该子树中。

如果你能做到这一点，这就是解决这种问题和相关的一系列问题的最简单方法。

记住，这并不一定要提升整个组件。你还可以重构应用程序，将一段逻辑移动到树中更高的组件中。

这真的只是依赖注入而已

很熟悉软件工程设计模式的读者可能已经注意到，我们在这里所做的就是依赖注入——这是我们在软件工程中已经使用了数十年的技术。

它的一个用途是用来制作易于配置的代码。在我们的示例中，我们对每个 Page 所用的 Layout 组件给出了不一样的配置。

当我们调换 Page 和 Layout 组件的关系时，这种做法就是所谓的控制反转。

在基于组件的框架中，父组件控制子组件的行为（因为它在前者的子树内），因此我们选择让 Page 控件控制 Layout 组件，而不是让 Layout 组件控制 Page。

为了做到这一点，我们为 Layout 组件提供了插槽。

正如你所见，使用依赖注入可以使我们的代码更加模块化、更易于配置。

总结

我们探讨了解决这个问题的四种方法，展示了每种解决方案的优缺点。然后我们进一步将问题一般化，探讨该如何控制组件子树之外的东西。

我希望读者能明白状态提升和依赖注入是两种非常有用的模式。它们是你工具库中的绝佳道具，因为它们可以应用在无数软件开发问题上。

但最重要的是，我希望你记住下面的结论：

我们用一些常见的软件模式就能将一个问题丑陋的解决方案变成非常优雅的方案。

很多问题都可以通过这种方式解决——研究丑陋、复杂的方案并将其转化为更简单、更容易的解决途径。

英文原文：https://michaelnthiessen.com/advanced-vue-controlling-parent-slots