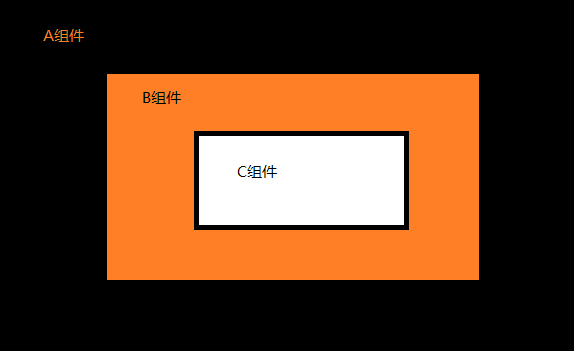
## **前言**

多级组件嵌套需要传递数据时，通常使用的方法是通过vuex。如果仅仅是传递数据，而不做中间处理，使用 vuex 处理，未免有点杀鸡用牛刀。Vue 2.4 版本提供了另一种方法，使用 v-bind=”$attrs”, 将父组件中不被认为 props特性绑定的属性传入子组件中，通常配合 interitAttrs 选项一起使用。之所以要提到这两个属性，是因为两者的出现使得组件之间跨组件的通信在不依赖 vuex 和事件总线的情况下变得简洁，业务清晰。

**首先分析以下应用场景：**



## **A 组件与 B 组件之间的通信： （父子组件）**

如上图所示，A、B、C三个组件依次嵌套，按照 Vue 的开发习惯，父子组件通信可以通过以下方式实现：

* A to B 通过props的方式向子组件传递，B to A 通过在 B 组件中 $emit, A 组件中 v-on 的方式实现
* 通过设置全局Vuex共享状态，通过 computed 计算属性和 commit mutation的方式实现数据的获取和更新，以达到父子组件通信的目的。
* Vue Event Bus，使用Vue的实例，实现事件的监听和发布，实现组件之间的传递。

往往数据在不需要全局的情况而仅仅是父子组件通信时，使用第一种方式即可满足。

## **A 组件与 C 组件之间的通信： （跨多级的组件嵌套关系）**

如上图，A 组件与 C 组件之间属于跨多级的组件嵌套关系，以往两者之间如需实现通信，往往通过以下方式实现：

* 借助 B 组件的中转，从上到下props依次传递，从下至上，$emit事件的传递，达到跨级组件通信的效果
* 借助Vuex的全局状态共享
* Vue Event Bus，使用Vue的实例，实现事件的监听和发布，实现组件之间的传递。

很显然，第一种通过props和$emit的方式，使得组件之间的业务逻辑臃肿不堪，B组件在其中仅仅充当的是一个中转站的作用。如使用第二种 Vuex的方式，某些情况下似乎又有点大材小用的意味，（仅仅是想实现组件之间的一个数据传递，并非数据共享的概念）。第三种情况的使用在实际的项目操作中发现，如不能实现很好的事件监听与发布的管理，往往容易导致数据流的混乱，在多人协作的项目中，不利于项目的维护。

$attrs以及$listeners的出现解决的就是第一种情况的问题，B 组件在其中传递props以及事件的过程中，不必在写多余的代码，仅仅是将$attrs以及$listeners向上或者向下传递即可。

## **示例代码**

如下所示：

### **A组件（App.vue）**

<template>

<div id="app">

<child1

:p-child1="child1"

:p-child2="child2"

v-on:test1="onTest1" //此处监听了两个事件，可以在B组件或者C组件中直接触发

v-on:test2="onTest2">

</child1>

</div>

</template>

<script>

import Child1 from './Child1.vue';

export default {

data () {

return {};

},

components: { Child1 },

methods: {

onTest1 () {

console.log('test1 running...');

},

onTest2 () {

console.log('test2 running');

}

}

};

</script>

### **B组件（Child1.vue）**

<template>

<div class="child-1">

<p>in child1:</p>

<p>props: {{pChild1}}</p>

<p>$attrs: {{$attrs}}</p>

<hr>

<!-- C组件中能直接触发test的原因在于 B组件调用C组件时 使用 v-on 绑定了$listeners 属性 -->

<!-- 通过v-bind 绑定$attrs属性，C组件可以直接获取到A组件中传递下来的props（除了B组件中props声明的） -->

<child2 v-bind="$attrs" v-on="$listeners"></child2>

</div>

</template>

<script>

import Child2 from './Child2.vue';

export default {

props: ['pChild1'],

data () {

return {};

},

inheritAttrs: false,

components: { Child2 },

mounted () {

this.$emit('test1');

}

};

</script>

结果：

in child1:

props: v\_child1

$attrs: { “p-child2”: “v\_child2”}

### **C 组件 (Child2.vue)**

<template>

<div class="child-2">

<p>in child2:</p>

<p>props: {{pChild2}}</p>

<p>$attrs: {{$attrs}}</p>

<hr>

</div>

</template>

<script>

export default {

props: ['pChild2'],

data () {

return {};

},

inheritAttrs: false,

mounted () {

this.$emit('test2');

}

};

</script>

结果：

in child2:

props: v\_child2

$attrs: {}

## **知识点总结**

**$attrs**

 包含了父作用域中不被认为 (且不预期为) props 的特性绑定 (class 和 style 除外)。当一个组件没有声明任何 props 时，这里会包含所有父作用域的绑定 (class 和 style 除外)，并且可以通过 v-bind=”$attrs” 传入内部组件——在创建更高层次的组件时非常有用。

**$listeners**

包含了父作用域中的 (不含 .native 修饰器的) v-on 事件监听器。它可以通过 v-on=”$listeners” 传入内部组件——在创建更高层次的组件时非常有用。

**inheritAttrs**

默认情况下父作用域的不被认作 props 的特性绑定 (attribute bindings) 将会“回退”且作为普通的 HTML 特性应用在子组件的根元素上。当撰写包裹一个目标元素或另一个组件的组件时，这可能不会总是符合预期行为。通过设置 inheritAttrs 到 false，这些默认行为将会被去掉。而通过 (同样是 2.4 新增的) 实例属性 $attrs 可以让这些特性生效，且可以通过 v-bind 显性的绑定到非根元素上。

上述特性的使用完全可以降低在不使用Vuex以及事件总线的情况下，组件跨级props以及事件传递的复杂度。