Vue.js 父子组件通信的十种方式

*面试官：Vue 中父子组件通信有哪些方式？*

自己先想一分钟。

无可否认，现在无论大厂还是小厂都已经用上了 Vue.js 框架，简单易上手不说，教程详尽，社区活跃，第三方套件还多。真的是前端开发人员必备技能。而且在面试当中也往往会问到关于 Vue 方面的各种问题，其中大部分面试官会问到如上这种问题。

最近一直在做 Vue项目代码层面上的优化，说实话，优化别人的代码真是件痛苦的事情，功能实现尚且不说，就说代码规范我就能再写出一篇文章来。真的是无规范不成方圆，规范这个东西太重要了！有点扯了，回到主题，咳咳，那就谈谈我对上面的面试题的理解吧，文笔有限，不妥之处，欢迎在文章结尾留言斧正啊，正啊，啊！

## 概述

几种通信方式无外乎以下几种：

Prop（常用）

$emit (组件封装用的较多)

.sync语法糖 （较少）

$attrs 和 $listeners (组件封装用的较多)

provide 和 inject （高阶组件/组件库用的较多）

其他方式通信

## 详述

下面逐个介绍，大神请绕行。

### **1. Prop**

英式发音：[prɒp]。这个在我们日常开发当中用到的非常多。简单来说，**我们可以通过 Prop 向子组件传递数据**。用一个形象的比喻来说，父子组件之间的数据传递相当于自上而下的下水管子，只能从上往下流，不能逆流。这也正是 Vue 的设计理念之单向数据流。而 Prop 正是管道与管道之间的一个衔接口，这样水（数据）才能往下流。说这么多，看代码：

<div id="app">

  <child :content="message"></child>

</div>

// Js

let Child = Vue.extend({

  template: '<h2>{{ content }}</h2>',

  props: {

    content: {

      type: String,

      default: () => { return 'from child' }

    }

  }

})

new Vue({

  el: '#app',

  data: {

    message: 'from parent'

  },

  components: {

    Child

  }

})

浏览器输出：

from parent

### **2. $emit**

英式发音：[iˈmɪt]。官方说法是**触发当前实例上的事件。附加参数都会传给监听器回调**。按照我的理解不知道能不能给大家说明白，先简单看下代码吧：

<div id="app">

  <my-button @greet="sayHi"></my-button>

</div>

let MyButton = Vue.extend({

  template: '<button @click="triggerClick">click</button>',

  data () {

    return {

      greeting: 'vue.js!'

    }

  },

  methods: {

    triggerClick () {

      this.$emit('greet', this.greeting)

    }

  }

})

new Vue({

  el: '#app',

  components: {

    MyButton

  },

  methods: {

    sayHi (val) {

      alert('Hi, ' + val) // 'Hi, vue.js!'

    }

  }

})

大致逻辑是酱婶儿的：当我在页面上点击按钮时，触发了组件 MyButton 上的监听事件 greet，并且把参数传给了回调函数 sayHi 。说白了，当我们从子组件 Emit（派发） 一个事件之前，其内部都提前在事件队列中 On（监听）了这个事件及其监听回调。其实相当于下面这种写法：

vm.$on('greet', function sayHi (val) {

  console.log('Hi, ' + val)

})

vm.$emit('greet', 'vue.js')

// => "Hi, vue.js"

### **3. .sync 修饰符**

这个家伙在 vue@1.x 的时候曾作为双向绑定功能存在，即子组件可以修改父组件中的值。因为它违反了单向数据流的设计理念，所以在 vue@2.0 的时候被干掉了。但是在 vue@2.3.0+ 以上版本又重新引入了这个 .sync 修饰符。但是这次它只是作为一个编译时的语法糖存在。它会被扩展为一个自动更新父组件属性的 v-on 监听器。说白了就是让我们手动进行更新父组件中的值了，从而使数据改动来源更加的明显。下面引入自官方的一段话：

*在有些情况下，我们可能需要对一个 prop 进行“双向绑定”。不幸的是，真正的双向绑定会带来维护上的问题，因为子组件可以修改父组件，且在父组件和子组件都没有明显的改动来源。*

既然作为一个语法糖，肯定是某种写法的简写形式，哪种写法呢，看代码：

<text-document

  v-bind:title="doc.title"

  v-on:update:title="doc.title = $event">

</text-document>

于是我们可以用 .sync 语法糖简写成如下形式：

<text-document v-bind:title.sync="doc.title"></text-document>

废话这么多，如何做到“双向绑定” 呢？让我们进段广告，广告之后更加精彩！ ... 好的，欢迎回来。假如我们想实现这样一个效果：改变子组件文本框中的值同时改变父组件中的值。怎么做？列位不妨先想想。先看段代码：

<div id="app">

  <login :name.sync="userName"></login> {{ userName }}

</div>

let Login = Vue.extend({

  template: `

    <div class="input-group">

      <label>姓名:</label>

      <input v-model="text">

    </div>

  `,

  props: ['name'],

  data () {

    return {

      text: ''

    }

  },

  watch: {

    text (newVal) {

      this.$emit('update:name', newVal)

    }

  }

})

new Vue({

  el: '#app',

  data: {

    userName: ''

  },

  components: {

    Login

  }

})

下面划重点，代码里有这一句话：

this.$emit('update:name', newVal)

官方语法是：update:myPropName 其中 myPropName 表示要更新的 prop 值。当然如果你不用 .sync 语法糖使用上面的 .$emit 也能达到同样的效果。仅此而已！

### **4. $attrs 和 $listeners**

官网对 $attrs 的解释如下：

*包含了父作用域中不作为 prop 被识别 (且获取) 的特性绑定 (class 和 style 除外)。当一个组件没有声明任何 prop 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——在创建高级别的组件时非常有用。*

官网对 $listeners 的解释如下：

*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。*

我觉得 $attrs 和 $listeners 属性像两个收纳箱，一个负责收纳属性，一个负责收纳事件，都是以对象的形式来保存数据。看下面的代码解释：

<div id="app">

  <child

    :foo="foo"

    :bar="bar"

    @one.native="triggerOne"

    @two="triggerTwo">

  </child>

</div>

从 Html 中可以看到，这里有俩属性和俩方法，区别是属性一个是 prop 声明，事件一个是 .native 修饰器。

let Child = Vue.extend({

  template: '<h2>{{ foo }}</h2>',

  props: ['foo'],

  created () {

    console.log(this.$attrs, this.$listeners)

    // -> {bar: "parent bar"}

    // -> {two: fn}

    // 这里我们访问父组件中的 `triggerTwo` 方法

    this.$listeners.two()

    // -> 'two'

  }

})

new Vue({

  el: '#app',

  data: {

    foo: 'parent foo',

    bar: 'parent bar'

  },

  components: {

    Child

  },

  methods: {

    triggerOne () {

      alert('one')

    },

    triggerTwo () {

      alert('two')

    }

  }

})

可以看到，我们可以通过 $attrs 和 $listeners 进行数据传递，在需要的地方进行调用和处理，还是很方便的。当然，我们还可以通过 v-on="$listeners" 一级级的往下传递，子子孙孙无穷尽也！

一个插曲！

当我们在组件上赋予了一个非Prop 声明时，编译之后的代码会把这些个属性都当成原始属性对待，添加到 html 原生标签上，看上面的代码编译之后的样子：

<h2 bar="parent bar">parent foo</h2>

这样会很难看，同时也爆了某些东西。如何去掉？这正是 inheritAttrs 属性的用武之地！给组件加上这个属性就行了，一般是配合 $attrs 使用。看代码：

// 源码

let Child = Vue.extend({

  ...

  inheritAttrs: false, // 默认是 true

  ...

})

再次编译：

<h2>parent foo</h2>

### **5. provide / inject**

他俩是对CP, 感觉挺神秘的。来看下官方对 provide / inject 的描述：

*provide 和 inject 主要为高阶插件/组件库提供用例。并不推荐直接用于应用程序代码中。并且这对选项需要一起使用，以允许一个祖先组件向其所有子孙后代注入一个依赖，不论组件层次有多深，并在起上下游关系成立的时间里始终生效。*

看完描述有点懵懵懂懂！一句话总结就是：小时候你老爸什么东西都先帮你存着等你长大该娶媳妇儿了你要房子给你买要车给你买只要他有的尽量都会满足你。下面是这句话的代码解释：

<div id="app">

  <son></son>

</div>

let Son = Vue.extend({

  template: '<h2>son</h2>',

  inject: {

    house: {

      default: '没房'

    },

    car: {

      default: '没车'

    },

    money: {

      // 长大工作了虽然有点钱

      // 仅供生活费，需要向父母要

      default: '￥4500'

    }

  },

  created () {

    console.log(this.house, this.car, this.money)

    // -> '房子', '车子', '￥10000'

  }

})

new Vue({

  el: '#app',

  provide: {

    house: '房子',

    car: '车子',

    money: '￥10000'

  },

  components: {

    Son

  }

})

### **6. 其他方式通信**

除了以上五种方式外，其实还有：

**EventBus**

思路就是声明一个全局Vue实例变量 EventBus , 把所有的通信数据，事件监听都存储到这个变量上。这样就达到在组件间数据共享了，有点类似于 Vuex。但这种方式只适用于极小的项目，复杂项目还是推荐 Vuex。下面是实现 EventBus 的简单代码：

<div id="app">

  <child></child>

</div>

// 全局变量

let EventBus = new Vue()

// 子组件

let Child = Vue.extend({

  template: '<h2>child</h2>',

  created () {

    console.log(EventBus.message)

    // -> 'hello'

    EventBus.$emit('received', 'from child')

  }

})

new Vue({

  el: '#app',

  components: {

    Child

  },

  created () {

    // 变量保存

    EventBus.message = 'hello'

    // 事件监听

    EventBus.$on('received', function (val) {

      console.log('received: '+ val)

      // -> 'received: from child'

    })

  }

})

**Vuex**

官方推荐的，Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。

**$parent**

父实例，如果当前实例有的话。通过访问父实例也能进行数据之间的交互，但极小情况下会直接修改父组件中的数据。

**$root**

当前组件树的根 Vue 实例。如果当前实例没有父实例，此实例将会是其自己。通过访问根组件也能进行数据之间的交互，但极小情况下会直接修改父组件中的数据。

**broadcast / dispatch**

他俩是 vue@1.0 中的方法，分别是事件广播 和 事件派发。虽然 vue@2.0 里面删掉了，但可以模拟这两个方法。可以借鉴 Element 实现。有时候还是非常有用的，比如我们在开发树形组件的时候等等。

## 总结

啰嗦了这么多，希望看到的同学或多或少有点收获吧。不对的地方还请留言指正，不胜感激。父子组件间的通信其实有很多种，就看你在哪些情况下去用。不同场景不同对待。前提是你要心中有数才行！通过大神之路还有很远，只要每天看看社区，看看文档，写写Demo，每天进步一点点，总会有收获的。