# [**Vue3 的8种组件通信方式**](https://segmentfault.com/a/1190000041240135)

## Vue3 组件通信方式

props  
$emit  
expose / ref  
$attrs  
v-model  
provide / inject  
Vuex  
mitt

**Vue3 通信使用写法**

## 1. props

**用 props 传数据给子组件有两种方法，如下**

**方法一，混合写法**

**// Parent.vue 传送**

**<child :msg1="msg1" :msg2="msg2"></child>**

**<script>**

**import child from "./child.vue"**

**import { ref, reactive } from "vue"**

**export default {**

**data(){**

**return {**

**msg1:"这是传级子组件的信息1"**

**}**

**},**

**setup(){**

**// 创建一个响应式数据**

**// 写法一 适用于基础类型 ref 还有其他用处，下面章节有介绍**

**const msg2 = ref("这是传级子组件的信息2")**

**// 写法二 适用于复杂类型，如数组、对象**

**const msg2 = reactive(["这是传级子组件的信息2"])**

**return {**

**msg2**

**}**

**}**

**}**

**</script>**

**// Child.vue 接收**

**<script>**

**export default {**

**props: ["msg1", "msg2"],// 如果这行不写，下面就接收不到**

**setup(props) {**

**console.log(props) // { msg1:"这是传给子组件的信息1", msg2:"这是传给子组件的信息2" }**

**},**

**}**

**</script>**

**方法二，纯 Vue3 写法(语法糖)**

**// Parent.vue 传送**

**<child :msg2="msg2"></child>**

**<script setup>**

**import child from "./child.vue"**

**import { ref, reactive } from "vue"**

**const msg2 = ref("这是传给子组件的信息2")**

**// 或者复杂类型**

**const msg2 = reactive(["这是传级子组件的信息2"])**

**</script>**

**// Child.vue 接收**

**<script setup>**

**// 不需要引入 直接使用**

**// import { defineProps } from "vue"**

**const props = defineProps({**

**// 写法一**

**msg2: String**

**// 写法二**

**msg2:{**

**type:String,**

**default:""**

**}**

**})**

**console.log(props) // { msg2:"这是传级子组件的信息2" }**

**</script>**

注意：  
如果父组件是混合写法，子组件纯 Vue3 写法的话，是接收不到父组件里 data 的属性，只能接收到父组件里 setup 函数里传的属性。

如果父组件是纯 Vue3 写法，子组件混合写法，可以通过 props 接收到 data 和 setup 函数里的属性，但是子组件要是在 setup 里接收，同样只能接收到父组件中 setup 函数里的属性，接收不到 data 里的属性。

官方也说了，既然用了 3，就不要写 2 了，[前端培训](https://link.segmentfault.com/?enc=Z7jTWKKbY8mbbnJG0NDn9Q%3D%3D.zR%2FdSwUzTS34XDRKu3nQog6P8rgNMKtmuK36ZupxZNA%3D)所以不推荐混合写法。下面的例子，一律只用纯 Vue3 的写法，就不写混合写法了。

## 2. $emit

// Child.vue 派发

**<template>**

**// 写法一**

**<button @click="emit('myClick')">按钮</buttom>**

**// 写法二**

**<button @click="handleClick">按钮</buttom>**

**</template>**

**<script setup>**

**// 方法一 适用于Vue3.2版本 不需要引入**

**// import { defineEmits } from "vue"**

**// 对应写法一**

**const emit = defineEmits(["myClick","myClick2"])**

**// 对应写法二**

**const handleClick = ()=>{**

**emit("myClick", "这是发送给父组件的信息")**

**}**

**// 方法二 不适用于 Vue3.2版本，该版本 useContext()已废弃**

**import { useContext } from "vue"**

**const { emit } = useContext()**

**const handleClick = ()=>{**

**emit("myClick", "这是发送给父组件的信息")**

**}**

**</script>**

**// Parent.vue 响应**

**<template>**

**<child @myClick="onMyClick"></child>**

**</template>**

**<script setup>**

**import child from "./child.vue"**

**const onMyClick = (msg) => {**

**console.log(msg) // 这是父组件收到的信息**

**}**

**</script>**

## 3. expose / ref

父组件获取子组件的属性或者调用子组件方法

// Child.vue

**<script setup>**

**// 方法一 不适用于Vue3.2版本，该版本 useContext()已废弃**

**import { useContext } from "vue"**

**const ctx = useContext()**

**// 对外暴露属性方法等都可以**

**ctx.expose({**

**childName: "这是子组件的属性",**

**someMethod(){**

**console.log("这是子组件的方法")**

**}**

**})**

**// 方法二 适用于Vue3.2版本, 不需要引入**

**// import { defineExpose } from "vue"**

**defineExpose({**

**childName: "这是子组件的属性",**

**someMethod(){**

**console.log("这是子组件的方法")**

**}**

**})**

**</script>**

**// Parent.vue 注意 ref="comp"**

**<template>**

**<child ref="comp"></child>**

**<button @click="handlerClick">按钮</button>**

**</template>**

**<script setup>**

**import child from "./child.vue"**

**import { ref } from "vue"**

**const comp = ref(null)**

**const handlerClick = () => {**

**console.log(comp.value.childName) // 获取子组件对外暴露的属性**

**comp.value.someMethod() // 调用子组件对外暴露的方法**

**}**

**</script>**

## 4. attrs

attrs：包含父作用域里除 class 和 style 除外的非 props 属性集合

// Parent.vue 传送

**<child :msg1="msg1" :msg2="msg2" title="3333"></child>**

**<script setup>**

**import child from "./child.vue"**

**import { ref, reactive } from "vue"**

**const msg1 = ref("1111")**

**const msg2 = ref("2222")**

**</script>**

**// Child.vue 接收**

**<script setup>**

**import { defineProps, useContext, useAttrs } from "vue"**

**// 3.2版本不需要引入 defineProps，直接用**

**const props = defineProps({**

**msg1: String**

**})**

**// 方法一 不适用于 Vue3.2版本，该版本 useContext()已废弃**

**const ctx = useContext()**

**// 如果没有用 props 接收 msg1 的话就是 { msg1: "1111", msg2:"2222", title: "3333" }**

**console.log(ctx.attrs) // { msg2:"2222", title: "3333" }**

**// 方法二 适用于 Vue3.2版本**

**const attrs = useAttrs()**

**console.log(attrs) // { msg2:"2222", title: "3333" }**

**</script>**

## 5. v-model

可以支持多个数据双向绑定

// Parent.vue

**<child v-model:key="key" v-model:value="value"></child>**

**<script setup>**

**import child from "./child.vue"**

**import { ref, reactive } from "vue"**

**const key = ref("1111")**

**const value = ref("2222")**

**</script>**

**// Child.vue**

**<template>**

**<button @click="handlerClick">按钮</button>**

**</template>**

**<script setup>**

**// 方法一 不适用于 Vue3.2版本，该版本 useContext()已废弃**

**import { useContext } from "vue"**

**const { emit } = useContext()**

**// 方法二 适用于 Vue3.2版本，不需要引入**

**// import { defineEmits } from "vue"**

**const emit = defineEmits(["key","value"])**

**// 用法**

**const handlerClick = () => {**

**emit("update:key", "新的key")**

**emit("update:value", "新的value")**

**}**

**</script>**

## 6. provide / inject

provide / inject 为依赖注入  
provide：可以让我们指定想要提供给后代组件的数据或  
inject：在任何后代组件中接收想要添加在这个组件上的数据，不管组件嵌套多深都可以直接拿来用

**// Parent.vue**

**<script setup>**

**import { provide } from "vue"**

**provide("name", "沐华")**

**</script>**

**// Child.vue**

**<script setup>**

**import { inject } from "vue"**

**const name = inject("name")**

**console.log(name) // 沐华**

**</script>**

## 7. Vuex

**// store/index.js**

**import { createStore } from "vuex"**

**export default createStore({**

**state:{ count: 1 },**

**getters:{**

**getCount: state => state.count**

**},**

**mutations:{**

**add(state){**

**state.count++**

**}**

**}**

**})**

**// main.js**

**import { createApp } from "vue"**

**import App from "./App.vue"**

**import store from "./store"**

**createApp(App).use(store).mount("#app")**

**// Page.vue**

**// 方法一 直接使用**

**<template>**

**<div>{{ $store.state.count }}</div>**

**<button @click="$store.commit('add')">按钮</button>**

**</template>**

**// 方法二 获取**

**<script setup>**

**import { useStore, computed } from "vuex"**

**const store = useStore()**

**console.log(store.state.count) // 1**

**const count = computed(()=>store.state.count) // 响应式，会随着vuex数据改变而改变**

**console.log(count) // 1**

**</script>**

## 8. mitt

**Vue3 中没有了 EventBus 跨组件通信，但是现在有了一个替代的方案 mitt.js，原理还是 EventBus**

**先安装 npm i mitt -S**

**然后像以前封装 bus 一样，封装一下**

**mitt.js**

**import mitt from 'mitt'**

**const mitt = mitt()**

**export default mitt**

**然后两个组件之间通信的使用**

**// 组件 A**

**<script setup>**

**import mitt from './mitt'**

**const handleClick = () => {**

**mitt.emit('handleChange')**

**}**

**</script>**

**// 组件 B**

**<script setup>**

**import mitt from './mitt'**

**import { onUnmounted } from 'vue'**

**const someMethed = () => { ... }**

**mitt.on('handleChange',someMethed)**

**onUnmounted(()=>{**

**mitt.off('handleChange',someMethed)**

**})**

**</script>**