# 第06课：智能组件与木偶组件的正确用法

Vue 中在组件层面的数据和行为通信，前五章通过一些 demo 和进行了深入总结，包括以下几点：
* data 与 props 的数据存放要素
* 单个组件 `$emit` 与 `$on` 的通信，父子组件 v-on 与 `$emit` 的通信
* .sync 和 v-model 双向绑定的模式
* `$attrs` 与 `$listeners` 深层次数据传递与行为交互的运用模式

以上涵盖了大量组件与组件之间的通信模式，只有能熟练掌握以上知识点，接下来才能对智能组件与木偶组件写法和封装有准确用法。

## 智能组件原理

智能组件可以称为第三方通用组件，也可以称之为业务型公用组件，与父组件之间的关系是完全解耦的，只能通过 props 进行数据传递，event 进行事件传递，不依赖于任何环境，只需要传递相应的数据和事件，就能得到你想要的操作。



## 木偶组件原理

木偶组件是为了业务页面进行拆分而形成的组件模式。比如一个页面，可以分多个模块，而每一个模块与其余页面并没有公用性，只是纯粹拆分。

还有一个方面则是复合组件的联动用法。当一个智能组件是由两个组件组成的一个复合智能组件，而它的子组件与父组件之间就有一个木偶的原理，因为两者是相互的，在开发者调用并需保持它们的关系性、规范性，一旦改变其本身的模式则会无效。

> 木偶组件的拆分简便用法

对于每一个木偶组件在定义之前，你必然会知道它将作用于哪个页面，在哪一层，都是有一个准确的不变性，取决于你对页面的拆分深度和数量。

## $parent 组件通信

`$parent` 指向当前组件的父组件，可以拿到父组件的整个实例。前面已经说了，木偶组件可以明确的知道运用在每个 spa 页面对应路由的第几层组件，可能是当前页面的子组件，孙子组件，或者更深的层次。而想和父组件进行通信的话，在不考虑复用的前题下，可以明确如何与父组件进行数据通信或者行为通信。

## 父组件

```
<template>
  <div class="hello">
     {{msg}}
     <demo></demo>
  </div>
</template>

<script>
  import Demo from './Demo.vue'
  export default {
    name: 'hello',
    components: {
      Demo
    },
    data () {
      return {
        msg: '父组件',
      }
    }
  }
</script>
```



## 子组件

```
<template>
   <div>
      <p>{{demoMsg}}</p>
      <p @click="handleClick">子组件</p>
   </div>

</template>
<script>
export default {
   name: 'demo',
   data () {
     return {
        demoMsg : ''
```

```
        }
    },
    methods: {
      handleClick () {
         let msg = this.$parent.msg
         this.demoMsg = msg
         this.$parent.msg = '父组件数据被改了'
      }
    }
}
</script>
```

demo 组件已经明确的知道是 Hello 组件的子组件，也可以是 demo 组件是 Hello 组件的木偶组件，通过 `$parent` 就可以随意取到和改动父组件实例的属性（数据）。同样这也并不违反数据的单向流的原则，可以对比一下通过 v-on 和 `$emit` 或者 v-model，.sync 这几种方法，不但方便很多，还更加快捷，并且明确了组件的位置，就像木偶一样，永远不会变，它的父组件永远只会是同一个。

### $parent 调用父组件方法

父组件



```
...
methods : {
    parentMethods () {
        console.log('调用父组件的方法')
    }
}
```

### 子组件

```
this.$parent.parentMethods()
```

同样可以调用父件的方法，通过子组的调用去执行父组件的方法。此方法是在父组件内部执行的，在某些场景下就会显得很便捷，后面会给出例子。

## $children 的组件通信

`$children` 也是针对于木偶组件的应用，它和 `$parent` 相反，此 Api是对于一个组件来说，已经明确知道它的子组件，也可能是一个子组件集，准确地拿到想要的子组件实例，或者子组件集实列 `$children` 可以通过父组件拿到子组件的实例，它是以一个数组的形式包裹。

## 父组件

```
<template>
  <div class="hello">
     <p @click='handlerClick'>父组件</p>
     <demo></demo>
     <demo></demo>
  </div>
</template>

<script>
  import Demo from './Demo.vue'
  export default {
    name: 'hello',
    components: {
      Demo
    },
    methods: {
      handlerClick () {
        console.log(this.$children)
        this.$children.forEach(item => {
           item.demoMsg = '通过$children改变'
        })
      }
    }
  }
</script>
```



## 子组件

```
<template>
   <div>
      <p>{{demoMsg}}</p>
   </div>

</template>
<script>
export default {
   name: 'demo',
   data () {
     return {
        demoMsg : ''
     }
   }
}
</script>
```

此时已经不是通过子组件去与父组件通信，而是用父组件与子组件通信， $parent 与 $children 就形成了一个父子组件互相通信的机制，还是那句重点一句 只适合木偶组件的模式 。

在父组件中明确 demo 组件是子组件，通过 $children 拿到所有 demo 组件的实例，通过 forEach 循环改变每个子组件的实例属。因为 data 里所有属性(数据)都是通过 object.defineproperty 来进行数据劫持，把 data 里的属性都绑到 Vue 实例上。从中我们可以轻而易举的得到它。

> 智能组件的运用

智能组件可能是业务组件也可能是第三方通用组件，总归是多个组件公用的子组件，因为它可能服务多个组件或者页面，当嵌入不同组件里，所需要展求的业务能力也是有所区别的，因此称之为智能组件。

## 举一个例子：

比方说一个智能组件 A，将嵌入 B，C 组件做为子组件：

当A嵌入到B中需要显示文案 嵌入B组件中

当A嵌入到C中需要显示文案 嵌入C组件中

通过向智能传递一个数据和标识，告诉它我需要你展示什么？



## 父组件

```
<template>
  <div class="hello">
     <p>父组件</p>
     <demo type='A'></demo>
  </div>
</template>

<script>
  import Demo from './Demo.vue'
  export default {
    name: 'hello',
    components: {
      Demo
    }
  }
</script>
```

子组件

```
<template>
   <div>
      <p>{{type==='B'?'嵌入B的组件':'嵌入C的组件'}}</p>
   </div>

</template>
<script>
export default {
   name: 'demo',
   props: ['type']
}
</script>
```

对于智能组件你永远不知道你将作用于哪个组件之下，这本身就是一个不定因素，特别对于通用组件，这将会暴露各种方法和 props 数据，只有传递数据传递事件去做自己想做的事件，智能组件（也是一个封装模块），会根据传入的数据和事件去做内部封装后所做的事情，而你并不可以轻意的随便改动它。

## 智能组件里的木偶组件



智能组件与木偶组件同时可以相互嵌套，可以作用在 复合组件上 。一般复合组件是都是通三方通用组件称之为智能组件，但是复合组件的父组件和子组件同样可以互相成为对方的 木偶组件 ，两者可以成为 相互依赖的关系 。无论从代码量和理解，调用都会很方便，木偶组件相比智能组件 更方便理解和简洁 ，但是 功能上就比较单一 。

### 通过一个 accordion 折叠面板来理解智能组件中的木偶组件

accordion属于第三方通用组件，同样也是一个复合组件。

### Accordion 组件

```
<template>
    <div>
        <slot></slot>
    </div>
</template>
<script>
    export default {
        props : ['repeat'],
        methods : {
            open (uid) {
```

```
                this.$children.forEach(item => {
                    if(item._uid != uid){
                        item.close = false
                    }
                })
            }
        }
    }
</script>
```

AccordionItem 组件

```
<template>
<div>
   <p @click='handleClick'>{{title}}</p>
   <div v-show='close' >
      <slot></slot>
   </div>
</div>
</template>

<script>
    export default {
        props : ['title'],
        data () {
            return {
                close : false
            }
        },
        created () {
            if(this.$parent.repeat === true) {
                this.close = true
            }
        },
        methods : {
            handleClick () {
                this.$parent.open(this._uid)
                this.close = !this.close
            }
        }
    }
</script>
```



# 最后的调用

```
<template>
  <div class="hello">
     <accordion :repeat='true'>
         <accordion-item title='vueTitle'>vue</accordion-item>
         <accordion-item title='vue-routerTitle'>vue-
router</accordion-item>
         <accordion-item title='vuex-Title'>vuex</accordion-item>
     </accordion>
  </div>
</template>

<script>
  import Accordion from './accordion.vue'
  import AccordionItem from './accordion-item.vue'
  export default {
    name: 'hello',
    components: {
      Accordion,
      AccordionItem
    }
  }
</script>
```



先从智能组件这个方面说起，无论是 accordion 还是 accordion-item 同向外暴露一个 props进行你希望的操作。

accordion 暴露了一个 repeat，当 repeat 为 true 的时候则把所有 item 项初始化都进行展开。

accordion-item 暴露了一个 title，可以随意传入你想设计的标题。

以上这些往往都是一些不定因素，也不知道它可能会嵌套在哪个页面组件的哪一层，这就是复合组件的智能方面。

再从木偶组件这个方面论一下。accordion 与 accordion-item 两者是父子组件关系，这种关系是不可变的，想要用到这个复合组件，accordion 与 accordion-item 必须保证确定的父子组件关系，并且缺一不可，从中就能体现出两者的木偶性。

accordion-item 通过 `$parent` 调用 accordion 父组件的 open 方法，
而 accordion 通过 `$children` 拿到每一个 accordion-item 子组件的实例，进行显示隐藏的转换。两者很充分形成了一个对木偶关系，这种父子关系是永远断不了的。

> 总结：
> 木偶组件：子组件只能有一个爹，必须是唯一的，而且父子俩长得一模一模，谁离开谁都活不了。

智能组件：子组件可以有N个爹，非唯一性，而且父子长得不一定要一样，子组件可能会有N个爹的特性，子组件离开哪个爹都能继续生存。