第07课：你不知道的中央事件通信

中央事件通信，就像一根线一样，把两个组件的通信用一根线连接起来。前面几节课讲了父子组件通信与深层次嵌套组件通信，并且已经通过各种 Api 和良好的解决方案，但是同级组件怎么办，无论用 `$emit` `v-on` `v-model` `.sync` `$attr`与`$listeners` 都不适用，以上这些都是 基于嵌套的父子组件进行通信 。

> 同级组件通信，也是一种常见的通信模式，在一个大的容器下(父组件)底下有两个平级的组件，两个组件进行数据交或者行为交互，在 Api 的方法里也没有专门的设计。

通过 $emit， v-on， $on 三者结合使用

这种操作是非常复杂的，如果你能良好掌握以上三个 Api 进行同级组件的通信，那你对这三个 Api 已经完全掌握了。这种方法是一种过渡方法， b->a a->c ， 意思是 a 去通知 b ， b 对 a 进行一个监听，当 a 监听到件事，在进行向 c 触发， c 的内部再进行监听，这样就形成了一个过渡链条。但是代码上就不显的那么直观了，多个触发事件，多个监听事件，一旦这种平级组件需要通信多了，那么代码就有一种很难维护的感觉。

> 实例demo

同级子组件 First

```
<template>
  <div>
    <p @click="$emit('fromFirst','来自A组件')">first组件</p>
  </div>
</template>

<script>
  export default {
    name: 'first'
  }
</script>
```

按着上面的讲解的顺序，先定义一个同级子组件，当点击的时候向外触发一个 eventName 为 fromFirst 的事件，传递一个 来自A组件 的参数这就形成了 b->a 让 a 去监听事件，让 b 去触发事件。

父组件

```
<template>
   <div>
     <p>父组件</p>
     <first v-on:fromFirst="hanlderFromA"></first>
     <second ref="second"></second>
  </div>
<template>
<script>
     import First from './first.vue'
  import Second from './second.vue'
  export default {
    name: 'login',
    components: {
      First,
      Second
    },
    methods: {
      hanlderFromA (Bmsg) {
        let second = this.$refs.second
        second.$emit('fromLogin', Bmsg)
      }
    }
  }
<script>
```


- 父组件中引入了两个同级组件 First / Second ，还是延续 b->a 。此时 a 就是这个父组件，再梳理一下知识点， v-on与$emit 是进行 父子组件事件通信 ，作用在父子组件两个层面上，在 First 组件模版上进行一个 v-on监听 ， 一旦监听到触发 fromFirst 事件，则进行 hanlderFromA 函数。
- 接下来是 a->c 这个阶段， $emit与$on 都是作用在同一个组件的实列上，通过 this.$refs 拿到 Second 组件的实列，在执行 hanlderFromA函数 时再告诉 c 组件进行通信，同时把从 b 接收到的参数再次传入。

以上很明显能看出 A（父组件）只是一个过渡体，也可以说是一个真实的中央体，进行中央事件的派发。

同级子组件 Second

```
<template>
  <div>
    <p>{{Bmsg}}</p>
    <p>second组件</p>
  </div>
</template>

<script>
  export default {
    name: 'second',
    created () {
      this.$on('fromLogin', (Bmsg) => {
        this.Bmsg = Bmsg
        console.log('通信成功')
      })
    },
    data () {
      return {
        Bmsg: ''
      }
    }
  }
</script>
```


Second 组件是被通信的一方，在 a(父组件) 进行触发，然而在 c(second) 组件中进行监听，一旦监听到了 fromLogin 事件，可以做你想做得改变数据，行为操作都不是问题了。

这就 是b->a a->c 的模式，我只能用一句话说，复杂！实在是复杂，那必然有简单的方法。在了解更简单的方法之前，先了解一下 ES6 模块的运行机制 。

> ES6 模块的运行机制

JS 引擎对脚本静态分析的时候，遇到模块加载命令 import ，就会生成一个 只读引用 。ES6 export 的原始值变了， import加载 的值也会跟着变。因此， ES6 模块是动态引用，并且不会缓存值，模块里面的变量绑定其所在的模块。

> 举个例子

```
// lib.js
export let counter = 3;
export function incCounter() {
  counter++;
}

// main.js
```

```
import { counter, incCounter } from './lib';
console.log(counter); // 3
incCounter();
console.log(counter); // 4
```

虽然在 main.js 执行程序的时候加载了 count ，但是 count 在 lib.js 和在 main.js 里形成了一个引用关系，一旦libs内部的 export 导出的 counter 发生变化时， main.js 中同样会发生变化。

> 通过额外的实例进行简单的中央事件处理

定义一个额外的实例进行一个事件的中转，对于 ES6 模块的运行机制已经有了一个讲解，当模块内部发生变化的时候，引入模块的部分同样会发生变化，当又一个额外的实例对加载机制进行引入进行 $emit与$on 进行绑定通信，能轻而易举解决问题，通过 b->a->c 的模式直接过渡。

> 定义一个中央事件实例

```
import Vue from 'vue'

export default new Vue()
```


new 一个 Vue 的实例，然后把这个实例能过 es6 模块机制导出。

父组件改动

```
<template>
  <div>
     <p>父组件</p>
     <first></first>
     <second></second>
  </div>
</template>

<script>
  import First from './first.vue'
  import Second from './second.vue'
  export default {
    name: 'login',
    components: {
      First,
      Second
```

```
    }
  }

</script>
```

在父这里只需要进行两个同组件的引入，可以删除任何过渡的方式。

同级子组件 First 改动

```
<template>
  <div>
    <p @click="handleClick">first组件</p>
  </div>
</template>

<script>
  import Bus from './bus.js'
  export default {
    name: 'first',
    methods: {
      handleClick () {
        Bus.$emit('fromFirst', '来自A的组件')
      }
    }
  }
</script>
```


在 first同级 组件中把 bus 实例引入，点击时让 bus 实例触发一个 fromFirst 事件，这里你可能已经理解 module 加载机制配合在单个实例上用 $emit和$on 进行通信绑定，往下看。

同级子组件Second改动

```
<template>
  <div>
    <p>{{Bmsg}}</p>
    <p>second组件</p>
  </div>
</template>

<script>
  import Bus from './bus.js'
  export default {
    name: 'second',
    created () {
```

```
        Bus.$on('fromFirst', ( Amsg )=> {
          this.Bmsg = Amsg
          console.log('同级组件交互成功')
        })
      },
      data () {
        return {
          Bmsg: ''
        }
      }
    }
</script>
```

同样也引入 bus 实列，通过 bus 用 $on 监听 fromFirst 事件，因为 bus实例 与 bus.js 里的 export defalt new Vue 关系是一个引用关系，当代码执行后，无论 first 或者 second 组件通过 bus实例 形成了一个 中央事件链条 ，这种方法不但直观，也更加便捷。

> 中央事件的延生 跨组件深层次交互

既然同级组件可以用中央事件去过渡，那深层次嵌套不同级组件可以吗？那你肯定第一时间用到了 Vuex，但我一直认为 Vuex 操作大量的数据联动性非常有用，但是如果只是一个改变数据，或者执行事件，用起来反而更加直观。

> 将要模拟的方案：

- a 组件
  - first 组件 -> firstInner 组件
  - second 组件 -> secondInner 组件

当 firstInner组件 可能会与 second组件 或者 secondInner组件 发生跨组件深层次交互也同样可以用中央事件去进行过渡 ，如果说 vuex 是顶层共享数据源，那么中央事件就是顶层共享通信网 。

> demo 示例

前面的所有父组件都不写代码了，只展示一下firstInner 组件、secondInner 组件。

firstInner 组件

```
<template>
  <div>
    <p @click="handleClick">firstInner组件</p>
```

```
  </div>
</template>

<script>
  import Bus from './bus.js'
  export default {
    name: 'first',
    methods: {
      handleClick () {
        Bus.$emit('fromFirstInner', '来自firstInner组件')
      }
    }
  }
</script>
```

SecondInner 组件

```
<template>
  <div>
    <p>secondInner组件</p>
  </div>
</template>

<script>
  import Bus from './bus.js'
  export default {
    name: 'secondInner',
    created () {
      Bus.$on('fromFirstInner',(msg) => {
         console.log(msg)
      })
    }
  }
</script>
```


无论你想通信的两个组件嵌到在任何地方，它们的关系是如何的，只需要通过中央事件的处理，都能完成，同时还可以进行一对多的中央事件处理方式。在程序代码可控的情况下，没有什么是不可行的，只要数据量的变动是在可控范围之内，做一个中央事件网去行成一个通信网络，也是一个不错的选择。