第05课：$attrs，$listeners 两兄弟

在2.4版本中，有关 $attrs 和 $listeners 这两个实例属性用法还是比模糊，深层次挖掘将会非常有用，因为在项目中深层次组件交互的话可能就需要 Vuex 助力了，但是如果只是一个简单的深层次数据传递，或者进行某种交互时需要向上通知顶层或父层组件数据改变时，杀鸡用牛 VUX 可能未免有点多余！

什么情况才会显得多余，如果我们纯通过 props 一层一层向下传递，再通过 watch 或者 data 进行过渡，如果只是单向数据深层能传递，进行监听改变深传递的数据，不进行跨路由之间页面的共享的话，用这两个属性非常便捷。

组件与组件之间大胆解耦

有些开发者，特别对 Vuex 没有深入理解和实战经验的时候，同时对组件与组件多层传递时，不敢大胆的解耦组件，只能进行到父子组件这个层面，而且组件复用率层面上也有所下降。

$attr 与 interitAttrs 之间的关系

interitAttrs：

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。

注意：这个选项不影响 class 和 style 绑定。

what?

官网上并没有给出一点 demo，语意上看起来还是比较官方的，理解起来总是有点不太友好，通过一些 demo 来看看发生了什么。

子组件

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

父组件

```
<template>
  <div class="hello">
     <demo :first="firstMsg" :second="secondMessage"></demo>
  </div>
</template>

<script>
  import Demo from './Demo.vue'
  export default {
    name: 'hello',
    components: {
      Demo
    },
    data () {
      return {
        firstMsg: 'first props',
        secondMessage: 'second props'
      }
    },
  }
</script>
```


父组件在子组件中进行传递 firstMsg 和secondMsg 两个数据，在子组件中，应该有相对应的 props 定义的接收点，如果在 props 中定义了，你会发现无论是 firstMsg 和 secondMsg 都成了子组件的接收来的数据了，可以用来进行数据展示和行为操作。

虽然在父组件中在子组件模版上通过 props 定义了两个数据，但是子组件中的 props 只接收了一个，只接收了 firstMsg，并没有接收 secondMsg，没有进行接收的此时就会成为子组件根无素的属性节点。

```
<div class="hello" second="secondMessage"></div>
```

事件代理

当我们用 v-for 渲染大量的同样的 DOM 结构时，但是每个上面都加一个点击事件，这个会导致性能问题，那我们可以通过 HTML5 的 data 的自定义属性做事件代理。

父组件改动

```
<template>
  <div class="hello" @click="ff">
     <demo :first="firstMsg" :data-second="secondMsg"></demo>
     <demo :first="firstMsg" :data-second="secondMsg"></demo>
     <demo :first="firstMsg" :data-second="secondMsg"></demo>
     <demo :first="firstMsg" :data-second="secondMsg"></demo>
  </div>
</template>

<script>
  import Demo from './Demo.vue'
  export default {
    name: 'hello',
    components: {
      Demo
    },
    data () {
      return {
        firstMsg: 'first props',
        secondMsg: 'secondMsg'
      }
    },
    methods: {
      ff (e) {
        if(e.target.dataset.second == 'secondMsg') {
            console.log('通过事件委托拿到了自定义属性')
        }
      }
    }
  }
</script>
```


经过改动之后，在父组件中，把向子组件传递的参数名改成了 HTML 自定义的 data-second 属性，同样在子组件中不进行 props 接收，就顺其自然的成为了子组件每一个根节点的自定义属性。

通过事件冒泡的原理，然而可以从e.target.dataset.second 就能找对应的 Dom 节点进行逻辑操作。

同样，在子组件模版上可以绑定多个自定义属性，在子组件包裹的外层进行一次监听，通过 data 自定义属性拿到循环出来组件的对应的数据，进行逻辑操作。

interitAttrs = false 发生了什么？

```
<template>
   <div>
      {{first}}
   </div>
</template>
<script>
export default {
   name: 'demo',
   props: ['first'],
   inheritAttrs: false,
}
</script>
```

对子组件进行一个改动，我们加上 inheritAttrs: false，从字面上的翻译的意思，取消继承的属性，然而 props 里仍然没有接收 seconed，发现就算 props 里没有接收 seconed，在子组件的根元素上并没有绑定任何属性。

> $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。

在前面的例子中，子组件 `props` 中并没有接受 `seconed`，设置选项 `inheritAttrs: false`，同样也不会作为根元素的属性节点，整个没有接收的数据都被 `$attr` 实例属性给接收，里面包含着所有父组件传入而子组件并没有在 Props 里显示接收的数据。

为了验证事实，可以在子组件中加上

```
created () {
       console.log(this.$attrs)
    }
```

打印出来则是一个对象
{second: “secondMsg”, third: “thirdMsg”}

warn

想要通 $attr 接收，但必须要保证设置选项 inheritAttrs: false，不然会默认变成根元素的属性节点。

开头说了，最有用的情况则是在深层次组件运用的时候，创建第三层孙子组件，作为第二层父组件的子组件，在子组件引入的孙子组件，在模版上把整个 $attr 当数作数据传递下去，中间则并不用通过任何方法去手动转换数据。

子组件

```
<template>
   <div>
      <next-demo v-bind="$attrs"></next-demo>
   </div>
</template>
```

孙子组件

```
<template>
  <div>
      {{second}}{{third}}
  </div>
</template>

<script>
  export default {
     props : [ 'second' , 'third']
  }
</script>
```


孙子组件在 props 接收子组件中通过 $attr 包裹传来的数据，同样是通过父组件传来的数据，只是在子组件用了 $attrs 进行了统一接收，再往下传递，最后通过孙子组件进行接收。

以此类推孙子组件仍然不想接收，再传入下级组件，我们仍然需要对孙子组件实力选项进行设置选项 inheritAttrs: false，否则仍然会成为孙子组件根元素的属性节点。

从而利用 $attrs 来接收 props 为接收的数据再次向下传递是一件很方便的事件，深层次接收数据我们理解了，那从深层次向层请求改变数据如何实现。意思就是让顶层数据和最底层数据进行一个双向绑定。

$listeners

listeners 可以认为是监听者。

向下如何传递数据已经了解了，面临的问题是如何向顶层的组件改变数据，父子组件可以通过 v-model，.sync，v-on 等一系列方法，深层及的组件可以通过 `$listeners` 去管理。

`$listeners` 和 `$attrs` 两者表面层都是一个意思，`$attrs` 是向下传递数据，`$listeners` 是向下传递方法，通过手动去调用 `$listeners` 对象里的方法，原理就是 `$emit` 监听事件，`$listeners` 也可以看成一个包裹监听事件的一个对象。

父组件

```
<template>
  <div class="hello">
     {{firstMsg}}
     <demo v-on:changeData="changeData" v-on:another = 'another'>
</demo>
  </div>
</template>

<script>
  import Demo from './Demo.vue'
  export default {
    name: 'hello',
    components: {
      Demo
    },
    data () {
      return {
        firstMsg: '父组件',
      }
    },
    methods: {
      changeData (params) {
         this.firstMsg = params
      },
      another () {
        alert(2)
      }
    }
  }
</script>
```


在父组件中引入子组件，在子组件模板上面进行 changeData 和 another 两个事件监听，其它这两个监听事件并不打算被触发，而是直接被调用，再简单的理解则是向下传递两个函数。

```
<template>
   <div>
      <p @click="$emit('another')">子组件</p>
      <next-demo  v-on='$listeners'></next-demo>
   </div>

</template>
<script>
import NextDemo from './nextDemo.vue'
export default {
   name: 'demo',
   components: {
       NextDemo
   },
   created () {
     console.log(this.$listeners)
   },
}
</script>
```

在子组件中，引入孙子组件 nextDemo。在子组件中像 $attrs 一样，可以用 $listeners 去整体接收监听的事件，{changeData: ƒ, another: ƒ} 以一个对象去接收，此时在父组件中子组件模板上监听的两个事件不但可以被子组件实例属性 $listeners 去整体接收，并且同时可以在子组件进行触发。

同样在孙子 nextDemo 组件中，继续向下传递，通过 v-on 把整个 $listeners 所接收的事件传递到孙子组件中，只是通过 $listeners 把其所有在父组件拿到监听事件一并通过 $listeners 一起传递到孙子组件里。

孙子组件

```
<template>
  <div class="hello">
      <p @click='$listeners.changeData("change")'>孙子组件</p>
  </div>
</template>

<script>
export default {
  name: 'demo',
  created () {
      console.log(this.$listeners)
  },
}
</script>
```

依然能拿到从子组中传递过来的 `$listeners` 所有的监听事件，此时并不是通过 `$emit` 去触发，而是像调用函数一样， `$emit` 只是针对于父子组件的双向通信， `$listeners` 包了一个对象，分别是 changeData 和 another，通过$listeners.changeData('change')等于直接触发了事件，执行监听后的回调函数，就是通过函数的传递，调用了父组件的函数。

通过 `$attrs` 和 `$listeners` 可以很愉快地解决深层次组件的通信问题，更加合理的组织你的代码。

> 下篇导读

以上介绍了如何在高层组件向下传递数据，在底层组件向上通知改变数据或者进行一些行为操作，而 `$listeners` 就像是调用了父组件的函数一样，看上去根本没有什么区别，你可能会想用 `$parents`，`$children` 一样能做到。不是不可用，而是在什么情况下适合用，通过下篇介绍木偶组建和智能组件好好理一下正确场景下如何准确利用 Api 进行行为交互、数据交互。