基础：Vue.js 组件的三个 API：prop、event、slot

如果您已经对 Vue.js 组件的基础用法了如指掌，可以跳过本小节，不过当做复习稍读一下也无妨。

组件的构成

一个再复杂的组件，都是由三部分组成的：prop、event、slot，它们构成了 Vue.js 组件的 API。如果你开发的是一个通用组件，那一定要事先设计好这三部分，因为组件一旦发布，后面再修改 API 就很困难了，使用者都是希望不断新增功能，修复 bug，而不是经常变更接口。如果你阅读别人写的组件，也可以从这三个部分展开，它们可以帮助你快速了解一个组件的所有功能。

属性 prop

prop 定义了这个组件有哪些可配置的属性，组件的核心功能也都是它来确定的。写通用组件时，props 最好用**对象**的写法，这样可以针对每个属性设置类型、默认值或自定义校验属性的值，这点在组件开发中很重要，然而很多人却忽视，直接使用 props 的数组用法，这样的组件往往是不严谨的。比如我们封装一个按钮组件 <i-button>：

```
<template>
  <button :class="'i-button-size' + size"
:disabled="disabled"></button>
</template>
<script>
  // 判断参数是否是其中之一
  function oneOf (value, validList) {
    for (let i = 0; i < validList.length; i++) {
      if (value === validList[i]) {
        return true;
      }
    }
    return false;
  }

  export default {
    props: {
      size: {
        validator (value) {
          return oneOf(value, ['small', 'large',
'default']);
        },
        default: 'default'
      },
      disabled: {
        type: Boolean,
        default: false
      }
    }
  }
</script>
```

使用组件:

```
<i-button size="large"></i-button>
<i-button disabled></i-button>
```

组件中定义了两个属性：尺寸 size 和 是否禁用 disabled。其中 size 使用 validator 进行了值的自定义验证，也就是说，从父级传入的 size，它的值必须是指定的 **small**、**large**、**default** 中的一个，默认值是 default，如果传入这三个以外的值，都会抛出一条警告。

要注意的是，组件里定义的 props，都是**单向数据流**，也就是只能通过父级修改，组件自己不能修改 props 的值，只能修改定义在 data 里的数据，非要修改，也是通过后面介绍的自定义事件通知父级，由父级来修改。

在使用组件时，也可以传入一些标准的 html 特性，比如 **id**、**class**：

```
<i-button id="btn1" class="btn-submit"></i-
button>
```

这样的 html 特性，在组件内的 <button> 元素上会继承，并不需要在 props 里再定义一遍。这个特性是默认支持的，如果不期望开启，在组件选项里配置 inheritAttrs: false 就可以禁用了。

插槽 slot

如果要给上面的按钮组件 <i-button> 添加一些文字内容，就要用到组件的第二个 API：插槽 slot，它可以分发组件的内容，比如在上面的按钮组件中定义一个插槽：

```
<template>
  <button :class="'i-button-size' + size"
:disabled="disabled">
    <slot></slot>
  </button>
</template>
```

这里的 <slot> 节点就是指定的一个插槽的位置，这样在组件内部就可以扩展内容了：

```
<i-button>按钮 1</i-button>
<i-button>
  <strong>按钮 2</strong>
</i-button>
```

当需要多个插槽时，会用到具名 slot，比如上面的组件我们再增加一个 slot，用于设置另一个图标组件：

```
<template>
  <button :class="'i-button-size' + size"
:disabled="disabled">
    <slot name="icon"></slot>
    <slot></slot>
  </button>
</template>
```

```
<i-button>
  <i-icon slot="icon" type="checkmark"></i-icon>
  按钮 1
</i-button>
```

这样，父级内定义的内容，就会出现在组件对应的 slot 里，没有写名字的，就是默认的 slot。

在组件的 <slot> 里也可以写一些默认的内容，这样在父级没有写任何 slot 时，它们就会出现，比如：

```
<slot>提交</slot>
```

自定义事件 event

现在我们给组件 <i-button> 加一个点击事件，目前有两种写法，我们先看自定义事件 event（部分代码省略）：

```
<template>
  <button @click="handleClick">
    <slot></slot>
  </button>
</template>
<script>
  export default {
    methods: {
      handleClick (event) {
        this.$emit('on-click', event);
      }
    }
  }
</script>
```

通过 $emit，就可以触发自定义的事件 on-click ，在父级通过 @on-click 来监听：

```
<i-button @on-click="handleClick"></i-button>
```

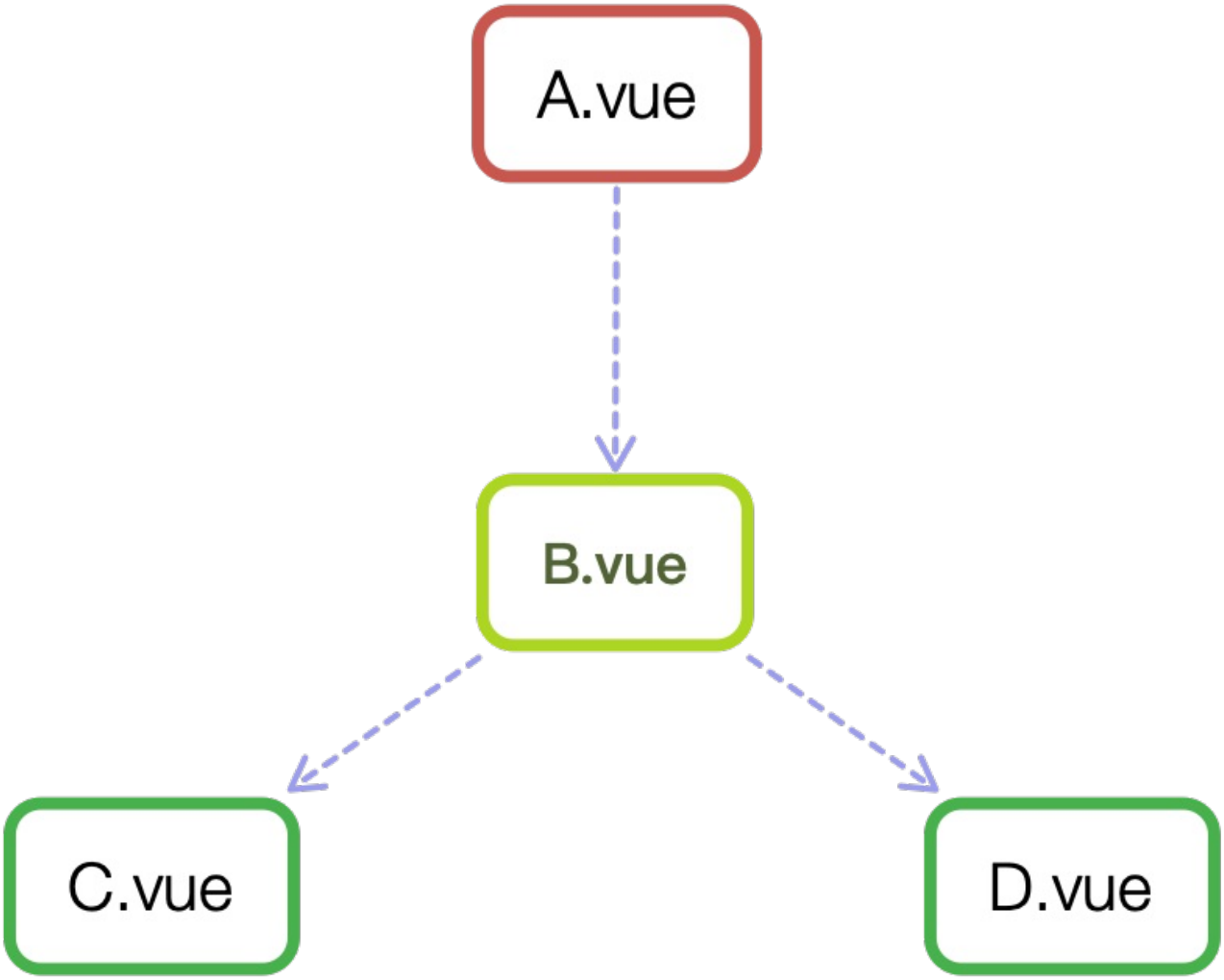
上面的 click 事件，是在组件内部的 <button> 元素上声明的，这里还有另一种方法，直接在父级声明，但为了区分原生事件和自定义事件，要用到事件修饰符 .native，所以上面的示例也可以这样写：

```
<i-button @click.native="handleClick"></i-button>
```

如果不写 .native 修饰符，那上面的 @click 就是**自定义事件** click，而非**原生事件** click，但我们在组件内只触发了 on-click 事件，而不是 click，所以直接写 @click 会监听不到。

组件的通信

一般来说，组件可以有以下几种关系：



A 和 B、B 和 C、B 和 D 都是父子关系，C 和 D 是兄弟关系，A 和 C 是隔代关系（可能隔多代）。组件间经常会通信，Vue.js 内置的通信手段一般有两种：

- ref：给元素或组件注册引用信息；

- $parent / $children：访问父 / 子实例。

这两种都是直接得到组件实例，使用后可以直接调用组件的方法或访问数据，比如下面的示例中，用 ref 来访问组件（部分代码省略）：

```
// component-a
export default {
  data () {
    return {
      title: 'Vue.js'
    }
  },
  methods: {
    sayHello () {
      window.alert('Hello');
    }
  }
}
```

```
<template>
  <component-a ref="comA"></component-a>
</template>
<script>
  export default {
    mounted () {
      const comA = this.$refs.comA;
      console.log(comA.title);  // Vue.js
      comA.sayHello();  // 弹窗
    }
  }
</script>
```

$parent 和 $children 类似，也是基于当前上下文访问父组件或全部子组件的。

这两种方法的弊端是，无法在**跨级**或**兄弟**间通信，比如下面的结构：

```
// parent.vue
<component-a></component-a>
<component-b></component-b>
<component-b></component-b>
```

我们想在 component-a 中，访问到引用它的页面中（这里就是 parent.vue）的两个 component-b 组件，那这种情况下，就得配置额外的插件或工具了，比如 Vuex 和 Bus 的解决方案，本小册不再做它们的介绍，读者可以自行阅读相关内容。不过，它们都是依赖第三方插件的存在，这在开发独立组件时是不可取的，而在小册的后续章节，会陆续介绍一些黑科技，它们完全不依赖任何三方插件，就可以轻松得到任意的组件实例，或在任意组件间进行通信，且适用于任意场景。

结语

本小节带您复习了 Vue.js 组件的核心知识点，虽然这并没有完全覆盖 Vue.js 的 API，但对于组件开发来说已经足够了，后续章节也会陆续扩展更多的用法。

基于 Vue.js 开发独立组件，并不是新奇的挑战，坦率地讲，它本质上还是 JavaScript。掌握了 Vue.js 组件的这三个 API 后，剩下的便是程序的设计。在组件开发中，最难的环节应当是解耦组件的交互逻辑，尽量把复杂的逻辑分发到不同的子组件中，然后彼此建立联系，在这其中，计算属性（computed）和混合（mixins）是两个重要的技术点，合理利用，就能发挥出 Vue.js 语言的最大特点：把状态（数据）的维护交给 Vue.js 处理，我们只专注在交互上。

当您最终读完本小册时，应该会总结出和笔者一样的感悟：Vue.js 组件开发，玩到最后还是在拼 JavaScript 功底。对于每一位使用 Vue.js 的开发者来说，阅读完本小册都可以尝试开发和维护一套属

于自己的组件库，并乐在其中，而且你会越发觉得，一个组件或一套组件库，就是融合了前端精髓的产出。

扩展阅读

- Vue 组件通信之 Bus (https://juejin.im/post/5a4353766fb9a044fb080927)
- Vuex 通俗版教程 (https://juejin.im/entry/58cb4c36b123db00532076a2)