拓展：Vue.js 面试、常见问题答疑

在过去的很多面试中，我会经常问候选人一些关于 Vue.js 的问题。这些问题从题面来看很简单，但仔细想又不是那么简单，不同的人，会答出不同的层次，从而更好地了解一个人对 Vue.js 的理解程度。

题目

v-show 与 v-if 区别

第一题应该是最简单的，提这个问题，也是想让候选人不那么紧张，因为但凡用过 Vue.js，多少知道 v-show 和 v-if 的区别，否则就没得聊了。不过这最简单的一道题，有三个层次，我会逐一追问。首先，基本所有人都会说到：

v-show 只是 CSS 级别的 display: none; 和 display: block; 之间的切换，而 v-if 决定是否会选择代码块的内容（或组件）。

回答这些，已经可以得到 50 分了，紧接着我会追问，什么时候用 v-show，什么时候用 v-if？到这里一部分人会比较吞吐，可能是知道，但表达不出来。我比较倾向的回答是：

频繁操作时，使用 v-show，一次性渲染完的，使用 v-if，只要意思对就好。

第二问可以得到 80 分了，最后一问很少有人能答上：**那使用 v-if 在性能优化上有什么经验？**这是一个加分项，要对 Vue.js 的组件编译有一定的理解。说一下期望的答案：

因为当 v-if="false" 时，内部组件是不会渲染的，所以在特定条件才渲染部分组件（或内容）时，可以先将条件设置为 false，需要时（或异步，比如 $nextTick）再设置为 true，这样可以优先渲染重要的其它内容，合理利用，可以进行性能优化。

绑定 class 的数组用法

动态绑定 class 应该不陌生吧，这也是最基本的，但是这个问题却有点绕，什么叫**绑定 class 的数组用法?** 我们看一下，最常用的绑定 class 怎么写：

```
<template>
  <div :class="{show: isShow}">内容</div>
</template>
<script>
  export default {
    data () {
      return {
        isShow: true
      }
    }
  }
</script>
```

绑定 class 的对象用法能满足大部分业务需求，不过，在复杂的场景下，会用到**数组**，来看示例：

```
<template>
  <div :class="classes"></div>
</template>
<script>
  export default {
    computed: {
      classes () {
        return [
          `${prefixCls}`,
          `${prefixCls}-${this.type}`,
          {
            [`${prefixCls}-long`]: this.long,
            [`${prefixCls}-${this.shape}`]:
!!this.shape,
            [`${prefixCls}-${this.size}`]:
this.size !== 'default',
            [`${prefixCls}-loading`]:
this.loading != null && this.loading,
            [`${prefixCls}-icon-only`]:
!this.showSlot && (!!this.icon ||
!!this.customIcon || this.loading),
            [`${prefixCls}-ghost`]: this.ghost
          }
        ];
      }
    }
  }
</script>
```

示例来自 iView 的 Button 组件，可以看到，数组里，可以是固定的值，还有动态值（对象）的混合。

计算属性和 watch 的区别

回答该题前，一般都会思考一下。很多人会偏题，直接去答计算属性和 watch 怎么用，这是不得分的，因为题目是问**区别**，并不是用法。

计算属性是自动监听依赖值的变化，从而动态返回内容，监听是一个过程，在监听的值变化时，可以触发一个回调，并做一些事情。

所以区别来源于用法，只是需要动态值，那就用计算属性；需要知道值的改变后执行业务逻辑，才用 watch，用反或混用虽然可行，但都是不正确的用法。

这个问题会延伸出几个问题：

1. computed 是一个对象时，它有哪些选项?
2. computed 和 methods 有什么区别?
3. computed 是否能依赖其它组件的数据?
4. watch 是一个对象时，它有哪些选项?

问题 1，已经在 16 小节介绍过，有 get 和 set 两个选项。

问题 2，methods 是一个方法，它可以接受参数，而 computed 不能；computed 是可以缓存的，methods 不会；一般在 v-for 里，需要根据当前项动态绑定值时，只能用 methods 而不能用 computed，因为 computed 不能传参。

问题 3，computed 可以依赖其它 computed，甚至是其它组件的 data。

问题 4，第 16 小节也有提到，有以下常用的配置：

- handler 执行的函数
- deep 是否深度
- immediate 是否立即执行

事件修饰符

这个问题我会先写一段代码：

```
<custom-component>内容</custom-component>
```

然后问：**怎样给这个自定义组件 custom-component 绑定一个原生的 click 事件?**

我一开始并不会问什么是事件修饰符，但是如果候选人说 <custom-component @click="xxx">，就已经错了，说明它对这个没有概念。这里的 @click 是自定义事件 click，并不是原生事件 click。绑定原生的 click 是这样的：

```
<custom-component @click.native="xxx">内容
</custom-component>
```

该问题会引申很多，比如常见的事件修饰符有哪些？如果你能说上 .exact，说明你是个很爱探索的人，会大大加分哦。

.exact 是 Vue.js 2.5.0 新加的，它允许你控制由精确的系统修饰符组合触发的事件，比如：

```
<!-- 即使 Alt 或 Shift 被一同按下时也会触发 -->
<button @click.ctrl="onClick">A</button>

<!-- 有且只有 Ctrl 被按下的时候才触发 -->
<button @click.ctrl.exact="onCtrlClick">A</button>


<!-- 没有任何系统修饰符被按下的时候才触发 -->
<button @click.exact="onClick">A</button>
```

你可能还需要了解常用的几个事件修饰符：

- .stop
- .prevent
- .capture
- .self

而且，事件修饰符在连用时，是有先后顺序的。

组件中 data 为什么是函数

为什么组件中的 data 必须是一个函数，然后 return 一个对象，而 new Vue 实例里，data 可以直接是一个对象?

因为组件是用来复用的，JS 里对象是引用关系，这样作用域没有隔离，而 new Vue 的实例，是不会被复用的，因此不存在引用对象的问题。

keep-alive 的理解

这是个概念题，主要考察候选人是否知道这个用法。简单说，就是把一个组件的编译缓存起来。在第 14 节有过详细介绍，也可以看看 Vue.js 的文档 (https://cn.vuejs.org/v2/guide/components-dynamic-async.html#在动态组件上使用-keep-alive)。

递归组件的要求

回答这道题，首先你得知道什么是**递归组件**。而不到 10% 的人知道递归组件。其实在实际业务中用的确实不多，在独立组件中会经常使用，第 14 节和 15 节专门讲过递归组件。那回到问题，递归组件的要求是什么？主要有两个：

- 要给组件设置 **name**；
- 要有一个明确的结束条件。

自定义组件的语法糖 v-model 是怎样实现的

在第 16 节已经详细介绍过，这里的 v-model，并不是给普通输入框 <input /> 用的那种 v-model，而是在自定义组件上使用。既然是语法糖，就能够还原，我们先还原一下：

```
<template>
  <div>
    {{ currentValue }}
    <button @click="handleClick">Click</button>
  </div>
</template>
<script>
  export default {
    props: {
      value: {
        type: Number,
        default: 0
      }
    },
    data () {
      return {
        currentValue: this.value
      }
    },
    methods: {
      handleClick () {
        this.currentValue += 1;
        this.$emit('input', this.currentValue);
      }
    },
    watch: {
      value (val) {
        this.currentValue = val;
      }
    }
  }
</script>
```

这个组件中，只有一个 props，但是名字叫 value，内部还有一个 currentValue，当改变 currentValue 时，会触发一个自定义事件 @input，并把 currentValue 的值返回。这就是一个 v-model 的语法糖，它要求 props 有一个叫 value 的项，同时触发的自定义事件必须叫 input。这样就可以在自定义组件上用 v-model 了：

```
<custom-component v-model="value"></custom-component>
```

如果你能说到 model 选项，绝对是加分的。

Vuex 中 mutations 和 actions 的区别

主要的区别是，actions 可以执行异步。actions 是调用 mutations，而 mutations 来修改 store。

Render 函数

这是比较难的一题了，因为很少有人会去了解 Vue.js 的 Render 函数，因为基本用不到。Render 函数的内容本小册已经很深入的讲解过了，遇到这个问题，一般可以从这几个方面来回答：

- 什么是 Render 函数，它的使用场景是什么。
- createElement 是什么?
- Render 函数有哪些常用的参数?

说到 Render 函数，就要说到虚拟 DOM（Virtual DOM）,Virtual DOM 并不是真正意义上的 DOM，而是一个轻量级的 JavaScript 对象，在状态发生变化时，Virtual DOM 会进行 Diff 运算，来更新只需要被替换的 DOM，而不是全部重绘。

它的使用场景，就是完全发挥 JavaScript 的编程能力，有时需要结合 JSX 来使用。

createElement 是 Render 函数的核心，它构成了 Vue Virtual DOM 的模板，它有 3 个参数：

```
createElement () {
  // {String | Object | Function}
  // 一个 HTML 标签，组件选项，或一个函数
  // 必须 return 上述其中一个
  'div',
    // {Object}
    // 一个对应属性的数据对象，可选
    // 您可以在 template 中使用
    {
    // 详细的属性
  },
    // {String | Array}
    // 子节点（VNodes），可选
    [
    createElement('h1', 'hello world'),
    createElement(MyComponent, {
      props: {
        someProps: 'foo'
      }
    }),
    'bar'
  ]
}
```

常用的参数，主要是指上面第二个参数里的值了，这个比较多，得去看 Vue.js 的文档。

怎样理解单向数据流

这个概念出现在组件通信。父组件是通过 prop 把数据传递到子组件的，但是这个 prop 只能由父组件修改，子组件不能修改，否则会报错。子组件想修改时，只能通过 $emit 派发一个自定义事件，父组件接收到后，由父组件修改。

一般来说，对于子组件想要更改父组件状态的场景，可以有两种方案：

1. 在子组件的 data 中拷贝一份 prop，data 是可以修改的，但 prop 不能：

```
export default {
  props: {
    value: String
  },
  data () {
    return {
      currentValue: this.value
    }
  }
}
```

2. 如果是对 prop 值的转换，可以使用计算属性：

```
export default {
  props: ['size'],
  computed: {
    normalizedSize: function () {
      return this.size.trim().toLowerCase();
    }
  }
}
```

如果你能提到 v-model 实现数据的双向绑定、.sync 用法，会大大加分的，这些在第 16 节已经详细介绍过。

生命周期

Vue.js 生命周期 (https://cn.vuejs.org/v2/api/#选项-生命周期钩子) 主要有 8 个阶段：

- 创建前 / 后（beforeCreate / created）：在 beforeCreate 阶段，Vue 实例的挂载元素 el 和数据对象 data 都为 undefined，还未初始化。在 created 阶段，Vue 实例的数据对象 data 有了，el 还没有。

- 载入前 / 后（beforeMount / mounted）：在 beforeMount 阶段，Vue 实例的 $el 和 data 都初始化了，但还是挂载之前为虚拟的 DOM 节点，data 尚未替换。在 mounted 阶段，Vue 实例挂载完成，data 成功渲染。

- 更新前 / 后（beforeUpdate / updated）：当 data 变化时，会触发 beforeUpdate 和 updated 方法。这两个不常用，且不推荐使用。

- 销毁前 / 后（beforeDestroy / destroyed）：beforeDestroy 是在 Vue 实例销毁前触发，一般在这里要通过 removeEventListener 解除手动绑定的事件。实例销毁后，触发 destroyed。

组件间通信

本小册一半的篇幅都在讲组件的通信，如果能把这些都吃透，基本上 Vue.js 的面试就稳了。

这个问题看似简单，却比较大，回答时，可以拆分为几种场景：

1. 父子通信：

   父向子传递数据是通过 props，子向父是通过 events（$emit）；通过父链 / 子链也可以通信（$parent / $children）；ref 也可以访问组件实例；provide / inject API。

2. 兄弟通信：

   Bus；Vuex；

3. 跨级通信：

   Bus；Vuex；provide / inject API。

除了常规的通信方法，本册介绍的 dispatch / broadcast 和 findComponents 系列方法也可以说的，如果能说到这些，说明你对 Vue.js 组件已经有较深入的研究。

路由的跳转方式

一般有两种：

1. 通过 <router-link to="home">，router-link 标签会渲染为 <a> 标签，在 template 中的跳转都是用这种；
2. 另一种是编程式导航，也就是通过 JS 跳转，比如 router.push('/home')。

Vue.js 2.x 双向绑定原理

这个问题几乎是面试必问的，回答也是有深有浅。基本上要知道核心的 API 是通过 Object.defineProperty() 来劫持各个属性的 setter / getter，在数据变动时发布消息给订阅者，触发相应的监听回调，这也是为什么 Vue.js 2.x 不支持 IE8 的原因（IE 8 不支持此 API，且无法通过 polyfill 实现）。

Vue.js 文档已经对 深入响应式原理 (https://cn.vuejs.org/v2/guide/reactivity.html) 解释的很透彻了。

什么是 MVVM，与 MVC 有什么区别

MVVM 模式是由经典的软件架构 MVC 衍生来的。当 View（视图层）变化时，会自动更新到 ViewModel（视图模型），反之亦然。View 和 ViewModel 之间通过双向绑定（data-binding）建立联系。与 MVC 不同的是，它没有 Controller 层，而是演变为 ViewModel。

ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而 View 和 Model 之间的同步工作是由 Vue.js 完成的，我们不需要手动操作 DOM，只需要维护好数据状态。

结语

一个人的简历，是由简单到复杂再到简单，技术是无止尽的，接触的越多，越能感到自己的渺小。