# 【前端面试】Vue3中的自定义指令有用过吗?

## 前言

Vue提供了各种各样的指令供我们使用，比如v-model、v-bind等等，可以说指令是Vue的重要功能点之一。除了Vue内置的一些指令外，Vue还允许我们自己定义指令，相信学过Vue的小伙伴应该都或多或少知道自定义指令，自定义指令在有些场景下非常的好用，它可以为我们省去超过工作量。

但是Vue3和Vue2中的自定义指令有一些区别，今天我们就重点学习一下**Vue3中自定义指令如何使用?**

## 1.什么是自定义指令?

为了照顾一些对自定义指令不太熟悉的同学，我们先来了解一下什么是自定义指令。

**内置指令:**

在Vue中，诸如v-if、v-for、v-on等等被称之为内置指令，它们都是以v-开头的，我们无需注册即可在全局使用它们，内置指令提供了极大的方便给我们，比如v-for指令可以让我们快速循环出很多dom元素等等，类似下面的代码:

```
<ul>
  <li v-for="index in 10">小猪课堂</li>
</ul>
```

**自定义指令:**

虽然Vue已经提供了很多内置指令供我们使用，但是人都是贪婪的，总是不满足于现状。所以官方允许我们自定义指令，自定义指令就比较灵活了，我们可以使用任何名称来命名自定义指令，不过我们自定义指定还是需要以v-开头，比如v-focus、v-resize等等。

比如下面我们使用自定义指令

```
<input v-focus />
```

## 2.准备工作

为了方便演示以及更加贴近大家的日常开发环境，这里我们就利用Vite搭建一个最简单Vue3项目，在此项目中演示自定义指令。

**执行命令：**

```
npm create vite@latest my-vue-app --template vue-ts
```

**运行项目：**


我们这里删除了一些不必要的东西，只留了一个logo。

## 3.注册自定义指令

在Vue中，如果我们定义了一个组件，我们需要注册它才可以使用。自定义指令也是类似的原理，我们需要先注册自定义指令，然后才可以使用它。

组件可以注册为局部组件和全局组件，同理，我们的自定义指令也分为了**全局注册和局部注册**，接下来我们就来学习如何注册自定义指令。

### 3.1 全局注册

当我们的自定义指令采用全局注册时，我们在任意组件中都可以使用该自定义指令，全局注册的方式也非常简单。

**代码如下：**

```
// main.ts
import { createApp } from "vue";
import App from "./App.vue";
const app = createApp(App);
app.directive("focus", {
  // 在绑定元素的 attribute 前
  // 或事件监听器应用前调用
  created(el, binding, vnode, prevVnode) {
  },
  // 在元素被插入到 DOM 前调用
  beforeMount() {},
  // 在绑定元素的父组件
  // 及他自己的所有子节点都挂载完成后调用
  mounted() {},
  // 绑定元素的父组件更新前调用
  beforeUpdate() {},
  // 在绑定元素的父组件
  // 及他自己的所有子节点都更新后调用
```

```
  updated() {},
  // 绑定元素的父组件卸载前调用
  beforeUnmount() {},
  // 绑定元素的父组件卸载后调用
  unmounted() {},
});
app.mount("#app");
```

上段代码中我们借助Vue3提供的directive方法注册了一个全局的自定义指令，该方法接收两个参数：指令名称、指令钩子函数对象。

钩子函数对象和组件的生命周期一样，这也和Vue2中的自定义指令有着较大的区别。理解这些钩子函数也很简单：我们都知道自定义指令是作用在DOM元素上，那么自定义指令从绑定到DOM元素，再到DOM元素发生变化等等一系列操作，都对应了不同的钩子函数，比如当DOM元素插入到文档中时，自定义指令的mounted等钩子函数就会执行。

调用全局注册的自定义指令，**代码如下：**

```
<input type="text" v-focus>
```

我们可以在任意组件中调用它。

## 3.2 局部注册

通常当我们有很多组件需要使用同一个自定义指令时，我们便会采取全局注册自定义指令的方式。但是很多时候我们可能只需要再某一个组件内部使用自定义指令，这个时候就没必要再全局注册了，我们可以采用局部注册的方式。

由于Vue3中有<script setup> 和<script>两种写法，两种写法对应的自定义指令的注册写法不太一样。

**<script setup>中注册：**

```
<script setup lang="ts">
// 在模板中启用 v-focus
const vFocus = {
  // 在绑定元素的 attribute 前
  // 或事件监听器应用前调用
  created(el, binding, vnode, prevVnode) {},
  // 在元素被插入到 DOM 前调用
  beforeMount() {},
  // 在绑定元素的父组件
  // 及他自己的所有子节点都挂载完成后调用
  mounted() {},
  // 绑定元素的父组件更新前调用
  beforeUpdate() {},
  // 在绑定元素的父组件
  // 及他自己的所有子节点都更新后调用
  updated() {},
```

```
  // 绑定元素的父组件卸载前调用
  beforeUnmount() {},
  // 绑定元素的父组件卸载后调用
  unmounted() {},
};
</script>
```

上段代码中我们直接定义了一个vFocus的变量，而这个变量其实就是我们的自定义指令，有些小伙伴可能会感到奇怪，为什么一个变量就成了一个自定义指令?

其实在Vue3中，只要以小写字母v开头的驼峰命名的变量都可以作为一个自定义指令使用，比如上段代码中vFocus就可以在模板中通过v-focus的指令形式使用。

**<script>中使用：**

在Vue3中没有使用<script setup>时，我们可以通过属性的形式注册自定义指令，这一点就和Vue2非常的类似。

**代码如下：**

```
export default {
  setup() {
    /*...*/
  },
  directives: {
    // 在模板中启用 v-focus
    focus: {
      /* 指令钩子函数 */
    }
  }
}
```

## 4.钩子函数和参数详解

前一节我们只简单讲了一下在Vue3中如何全局注册和局部注册自定义指令。但是我们的重点应该侧重于自定义指令中的钩子函数以及钩子函数接收的参数，毕竟我们使用自定义指令是为了完成一些需求的。

### 4.1 钩子函数

自定义指令中的钩子函数几乎和Vue3中的生命周期函数一样，这也是和Vue2中自定义指令的重大区别之一。

如果你理解了Vue3中的生命周期函数，那么你理解自定义指令的钩子函数就非常简单了。大家可以简单这样理解：**Vue3的生命周期函数是相对于一个应用或者一个组件而言的，而自定义组件的钩子函数是相对于一个DOM元素以及它的子元素而言的。**

我们来测试一下自定义指令的钩子函数，简单改造下App.vue文件，注册一个局部自定义指令v-focus，然后将该自定义指令绑定在input元素上。

**代码如下：**

```
<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <input type="text" v-focus />
</template>
<script setup lang="ts">
// 在模板中启用 v-focus
const vFocus = {
  // 在绑定元素的 attribute 前
  // 或事件监听器应用前调用
  created() {
    console.log("我是钩子函数:created");
  },
  // 在元素被插入到 DOM 前调用
  beforeMount() {
    console.log("我是钩子函数:beforeMount");
  },
  // 在绑定元素的父组件
  // 及他自己的所有子节点都挂载完成后调用
  mounted() {
    console.log("我是钩子函数:mounted");
  },
  // 绑定元素的父组件更新前调用
  beforeUpdate() {
    console.log("我是钩子函数:beforeUpdate");
  },
  // 在绑定元素的父组件
  // 及他自己的所有子节点都更新后调用
  updated() {
    console.log("我是钩子函数:updated");
  },
  // 绑定元素的父组件卸载前调用
  beforeUnmount() {
    console.log("我是钩子函数:beforeUnmount");
  },
  // 绑定元素的父组件卸载后调用
  unmounted() {
    console.log("我是钩子函数:unmounted");
  },
};
</script>
```

**输出结果：**

```
我是钩子函数:created
我是钩子函数:beforeMount
我是钩子函数:mounted
```

上段代码很简单，就是在钩子函数中打印了一句话而已，当我们刷新页面的时候，会执行3个钩子函数，因为我们绑定自定指令的DOM元素没有发生改变，所以其它钩子函数并未执行。我们可以试着修改一些代码，尝试触发其它钩子函数。

**代码如下：**

```
<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <div v-focus>
    <input type="text" />
    <div>{{ message }}</div>
  </div>
  <button @click="changeMsg">修改message</button>
</template>
<script setup lang="ts">
import { ref } from "vue";
let message = ref<string>("小猪课堂");
const changeMsg = () => {
  message.value = "张三";
};
// 在模板中启用 v-focus
const vFocus = {
  // 钩子函数
};
</script>
```

**页面显示：**


当我们点击按钮修改message后，会发现自定义指令中的钩子函数执行了。

**输出结果：**

```
我是钩子函数:beforeUpdate
我是钩子函数:updated
```

为了方便大家理解，我们总结一下各个钩子函数的执行时机：

- created：在绑定元素的 attribute 前或事件监听器应用前调用。
- beforeMount：在元素被插入到 DOM 前调用。
- mounted：在绑定元素的父组件及他自己的所有子节点都挂载完成后调用。
- beforeUpdate：绑定元素的父组件更新前调用。
- updated：在绑定元素的父组件及他自己的所有子节点都更新后调用。
- beforeUnmount：绑定元素的父组件卸载前调用。
- unmounted：绑定元素的父组件卸载后调用。

## 4.2 钩子函数参数详解

我们使用自定义指令的目的就是为了灵活的操作DOM以及在钩子函数中处理我们的业务逻辑，所以Vue3将一些我们可能会用到的参数传递给了钩子函数，Vue3和Vue2中的自定义指令传递的参数都很类似。

在前面的代码中，我们可以看到有下面这样一段代码：

```
created(el, binding, vnode, prevVnode) {}
```

上段代码中的created钩子函数接收了4个参数，这4个参数分别代表什么呢？我们一起来看看。

**参数详解：**

- el：指令绑定到的元素。这可以用于直接操作 DOM。
- binding：一个对象，包含以下属性
  - value：传递给指令的值。例如在 v-my-directive="1 + 1" 中，值是 2。
  - oldValue：之前的值，仅在 beforeUpdate 和 updated 中可用。无论值是否更改，它都可用。
  - arg：传递给指令的参数 (如果有的话)。例如在 v-my-directive:foo 中，参数是 "foo"。
  - modifiers：一个包含修饰符的对象 (如果有的话)。例如在 v-my-directive.foo.bar 中，修饰符对象是 { foo: true, bar: true }。
  - instance：使用该指令的组件实例。
  - dir：指令的定义对象。
- vnode：代表绑定元素的底层 VNode。

- prevNode：之前的渲染中代表指令所绑定元素的 VNode。仅在 beforeUpdate 和 updated 钩子中可用。

我们可以打印一下这些参数，能够更好的理解。

**代码如下：**

```
created(el: HTMLElement, binding: DirectiveBinding, vnode: VNode, prevVnode:
  console.log(el);
  console.log(binding);
  console.log(vnode);
  console.log(prevVnode);
},
```

**输出结果：**

```
<div>
  <input type="text">
  <div>小猪课堂</div>
</div>
{dir: {…}, instance: Proxy, value: undefined, oldValue: undefined, arg: undefined, …}
  arg: undefined
  dir: {created: ƒ, beforeMount: ƒ, mounted: ƒ, beforeUpdate: ƒ, updated: ƒ, …}
  instance: Proxy {__v_skip: true}
  modifiers: {}
  oldValue: undefined
  value: undefined
  [[Prototype]]: Object
{__v_isVNode: true, __v_skip: true, type: 'div', props: null, key: null, …}
null
```

## 5.指令传值

我们讲解钩子函数参数时，里面有一个binding参数，这个参数是一个对象，它里面有很多属性，而这些属性中有些就是指令传的值。

我们使用v-if或者v-for指令时，通常后面都接了=号，比如v-if="boolean"等等，等号后面的值就是我们传递给指令的值。

自定义指令也一样，也可以像内置指令那样传值。

**代码如下：**

```
<div v-focus:foo="'小猪课堂'">
</div>
```

接下来我们在钩子函数中将binding打印出来看看。

输出结果：

```
{dir: {…}, instance: Proxy, value: '小猪课堂', oldValue: undefined, arg: 'message', …}
  arg: "message"
  dir: {created: ƒ, beforeMount: ƒ, mounted: ƒ, beforeUpdate: ƒ, updated: ƒ, …}
  instance: Proxy {__v_skip: true}
  modifiers: {}
  oldValue: undefined
  value: "小猪课堂"
  [[Prototype]]: Object
```

可以看到binding对象中的value就是=号后面的值，而v-focus:foo中的foo就是binging中的arg。

看到这里大家回想一下某些内置指令的用法：v-bind:on=""，v-model=""。从这里可以看出，指令的arg是动态的，那么我们的自定义指令的arg也可以是动态的，比如下面这种写法。

**代码如下：**

```
<div v-example:[arg]="value"></div>
```

上段代码中的arg就是一个变量。

## 6.指令简写

从前几节可以看出，自定义指令提供了非常多的钩子函数，但是在实际的需求开发中，我们可能并不需要这么多钩子函数，所以Vue3和Vue2一样，都提供了自定义指令的简写形式给我们。比如我们经常用到的可能就是mounted和updated钩子函数了，所以官方直接针对这两种情况间提供了简写方法。

**代码如下：**

```
<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <div v-focus="'#ccc'">
    <input type="text" />
    <div>{{ message }}</div>
  </div>
  <button @click="changeMsg">修改message</button>
</template>
<script setup lang="ts">
import { DirectiveBinding, ref} from "vue";
let message = ref<string>("小猪课堂");
const changeMsg = () => {
  message.value = "张三";
};
// 在模板中启用 v-focus
const vFocus = (el: HTMLElement, binding: DirectiveBinding) => {
  // 这会在 `mounted` 和 `updated` 时都调用
  el.style.backgroundColor = binding.value;
};
</script>
```

上段代码中我们省去了自定义指令中的所有钩子函数，直接简写为了一个箭头函数。我们在div上绑定了自定义指令，并且传入了一个颜色参数。

**输出结果：**


当刷新页面时，div的背景色发生了变化。

## 7.自定义指令案例

防抖是我们在开发中很大概率会遇到的，通常小伙伴们都是自己写一个防抖函数，或者直接调用某些库的防抖函数来实现防抖，我们这次使用自定义指令的方式来方便的实现防抖。

代码如下：

```
<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <button v-debounce="changeMsg">修改message</button>
</template>
<script setup lang="ts">
import { DirectiveBinding, ref, VNode } from "vue";
let message = ref<string>("小猪课堂");
const changeMsg = () => {
  message.value = "张三";
  console.log('改变messag');
};
const vDebounce = (el: HTMLElement, binding: DirectiveBinding) => {
  let timer: null | number = null;
  el.addEventListener("click", () => {
    if (timer) {
      clearTimeout(timer);
    }
    timer = setTimeout(() => {
      binding.value(); // value = changeMsg
    }, 1000);
  });
};
</script>
```

**输出结果：**

```
    改变messag
```

上段代码中我们点击按钮可以更改message，如果采用@click的方式来触发方法，那么不断点击将会不停的触发方法。如果采用我们自定义指令v-debounce的形式来触发方法，那么就会实现方法的防抖，一定时间内只会触发一次。

可以把v-debounce自定义指令注册到全局，因为很多地方都需要防抖。

## 总结

自定义指令的用处非常多，如果你领略到了它的魅力，那么我相信你一定会爱上它的。Vue3和Vue2自定义指令在注册和使用上有一点，不同，不过原理都是一样的，所以如果你有Vue2的基础，学会Vue3的自定义指令简直就是信手拈来。