How to create custom directives in Vue

Custom directives allow you to extend the functionality of Vue websites in an extensible and modular way. Here are detailed instructions.

Custom directives allow you to extend the functionality of Vue websites in a modular and possible way. Here are detailed instructions.

Directives or directives/directives are programming constructs that define how the interpreter and compiler will handle input for an operation. The Vue directive extends the functionality of the HTML elements in the Vue template, allowing direct manipulation of the DOM.

You can use directives in Vue to add event listeners, among other operations. You will attach additional attributes to the HTML element to use the directive in your application.

Structure of Directive in Vue

Directives in Vue are prefixed with **v-** to distinguish them from regular HTML attributes. **The v-** prefix tells the Vue compiler that this attribute is a Vue directive, so it can process & apply the directive's behavior to the HTML element.

Here's an example that shows how the v-show attribute can be used to display the content of an h2 element:

Hello Vue

Vue.js has more built-in directives like **v-bind** , **v-if** , and **v-on** , allowing you to implement tasks like data binding, conditional rendering, condition handling and much more.

Defining custom directives in Vue

You can define custom directives to add new, reusable functionality to your Vue.js app. Creating custom directives requires two main steps. First, you will register the directive locally or globally. You will then define the directive's behavior with the lifecycle hook.

Register custom directives

You can register a custom directive in Vue locally or globally, depending on the intended scope. However, it is a more common practice to register directives globally. This ensures the commands are available everywhere in your Vue application.

You can register custom directives locally if you plan to use them in a simple Vue component. Here's how you can register the **v-changecolor** directive locally:

Learn about custom directives

This block of code represents the local registration of a custom directive in the Vue component. It defines the **v-changecolor** directive as an object in the camel case, **changeColor** .

To register the directive on a global scope, go to the **main.js** file in the root of the application directory and define it there:

```
// main.js import { createApp } from 'vue' import App from './App.vue' const app
```

This program registers a local custom directive using the **app.directive** method . It calls **app.directive** to register the **changecolor** directive in the app, then uses the app.mount method to mount the app to an HTML element with the app ID.

You can choose between local or global options to register custom directives based on your application's needs.

Define an instruction behavior

You can define custom directives as lifecycle hook objects. They define the behavior of the instruction and get the component to which the instruction is bound.

Some examples of lifecycle hooks are **created** , **mounted** and **updated** . Each hook provides functionality for interacting with the component at a specific stage.

You can use the lifecyce hook to access the HTML component of an element in the Document Object Model (DOM) after the Vue.js compiler mounts the application. Conversely, the lifecycle updated hook can perform additional updates to the component after you've refactored the component.

Here's how you can define an object, containing some lifecycle hooks for a Vue directive:

```
const directiveObject = { mounted(el, binding, vnode) { }, updated(el, binding, v
```

Custom directives can have the same behavior for **mounted** and **updated** hooks , no need for other lifecycle hooks. Also, many custom directives you create will just hook **mounted** and **updated** .

In such cases, directives are usually defined as functions, not objects with lifecycle hooks:

```
app.directive('changecolor', (el, binding, vnode) => { const message = 'Are you s
```

This code block defines a custom global directive with the string **changecolor** as the first argument. The second argument is a shorthand function that defines the behavior of the instruction.

As the name suggests, the **v-changecolor** directive changes the color of any HTML element you attach it to. This directive can randomly change the color of an HTML element when clicked.

The el parameter represents the HTML element you attached to the directive. **The binding** parameter is an object containing a property that defines how the directive should be applied. **The binding.value** parameter allows you to choose a default color when creating a Vue app. **The vnode** parameter contains information about the Vue.js virtual node associated with this component.

The changecolor directive uses a **JavaScript** event listener to capture the event that fires when you click an HTML element. **The confirm()** method displays a dialog box asking you to confirm if you want to randomly change the color of the element.

To test the generated directive, set up a Vue app similar to the one below:

Hello Vue

Learn custom directives

Can't Wait to Get Started {{ name }}

Observe this code assigns red color to **h2** tag but no color to h3 tag. When you preview the Vue app in the browser, it will look like this:

Click on both **Hello Vue** and **Learn custom directives** , you will see that the tag set to red will remain red but the tag that is not value will change to a random color.

This will happen after you confirm the selection, as shown below:

Custom directives provide low-level access to the DOM, help you create functionality across different Vue application components, improve app scalability. By following the steps in this article, you can create custom directives to ease your application's development.

You finished reading the article "**How to create custom directives in Vue**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.