The Vue.js Cheat Sheet

flaviocopes.com/vue-cheat-sheet

Published Jul 10, 2018

Directives

Directives are attributes identified by the `v-` prefix.

Directive	Description
`v-text`	uses the property as the text value of the element
`v-html`	uses the property as the text value of the element, interpreting HTML
`v-if`	show an element only if the conditional is true
`v-else`	shows an alternative element if the preceding `v-if` is false
`v-else-if`	adds an else if block for a `v-if` construct
`v-show`	similar to `v-if`, but adds the element to the DOM even if falsy. Just sets it to `display: none`.
`v-for`	iterates over an array or iterable object
`v-on`	listen to DOM events
`v-bind`	reactively update an HTML attribute
`v-model`	sets up a two-way binding for form inputs. used in form elements, updates the model when the user changes the form field value
`v-once`	applies the property just once, and never refreshes it even if the data passed changes

`v-bind` and `v-on` have a shorthand format:

```
<a v-bind:href="url">...</a>
<a :href="url">...</a>

<a v-on:click="doSomething">...</a>
<a @click="doSomething">...</a>
```

Example of `v-if` / `v-else` / `v-else-if`:

```
<div v-if="type === 'A'">
  it's A
</div>
<div v-else-if="type === 'B'">
  it's B
</div>
<div v-else-if="type === 'C'">
  it's C
</div>
<div v-else>
  it's neither one
</div>
```

Conditionals

You can embed a conditional in an expression using the ternary operator:

```
{{ isTrue ? 'yes' : 'no' }}
```

Working with form elements

To make the model update when the change event occurs, and not any time the user presses a key, you can use `v-model.lazy` instead of just `v.model`.

Working with input fields, `v-model.trim` is useful because it automatically removes whitespace.

And if you accept a number instead than a string, make sure you use `v-model.number`.

Modifying events

I use `click` as an example, but applies to all possible events

- `v-on:click.native` trigger a native DOM event instead of a Vue event
- `v-on:click.stop` stop the click event propagation
- `v-on:click.passive` makes use of the passive option of addEventListener
- `v-on:click.capture` use event capturing instead of event bubbling
- `v-on:click.self` make sure the click event was not bubbled from a child event, but directly happened on that element
- `v-on:click.once` the event will only be triggered exactly once
- `v-on:submit.prevent`: call `event.preventDefault()` on the triggered submit event, used to avoid a form submit to reload the page

For more on propagation, bubbling/capturing see my JavaScript events guide.

Mouse event modifiers

- `v-on:click .left` triggers only on left mouse button click
- `v-on:click .right` triggers only on right mouse button click
- `v-on:click .middle` triggers only on middle mouse button click

Submit an event only if a particular key is pressed

- `v-on:keyup.enter`
- `v-on:keyup.tab`
- `v-on:keyup.delete`
- `v-on:keyup.esc`
- `v-on:keyup.up`
- `v-on:keyup.down`
- `v-on:keyup.left`
- `v-on:keyup.right`

Keyboard event modifiers

Only trigger the event if a particular keyboard key is also pressed:

- `.ctrl`
- `.alt`
- `.shift`
- `.meta` (cmd on Mac, windows key on Win)

v-bind

- `v-bind .prop` bind a prop instead of an attribute
- `v-bind .camel` use camelCase for the attribute name
- `v-bind .sync` a syntactic sugar that expands into a `v-on` handler for updating the bound value. See this.

Lifecycle Hooks

- `beforeCreate` called before the app is created
- `created` called after the app is created
- `beforeMount` called before the app is mounted on the DOM
- `mounted` called after the app is mounted on the DOM
- `beforeDestroy` called before the app is destroyed
- `destroyed` called after the app is destroyed
- `beforeUpdate` called before a property is updated
- `updated` called after a property is updated
- `activated` called when a kept-alive component is activated
- `deactivated` called when a kept-alive component is deactivated

Built-in components

Vue provides 5 built-in components:

- `<component>`
- `<transition>`
- `<transition-group>`

- `<keep-alive>`
- `<slot>`

Global Configuration of the Vue object

The `Vue.config` object has these properties, which you can modify when you create the instance:

Property	Description
`silent`	defaults to false, if true suppress logs and warnings
`optionMergeStrategies`	allows to define a custom merging strategy for options
`devtools`	defaults to true in development, and false in production. You can override those values.
`errorHandler`	allows to set an error handler function. Useful to hook Sentry and other similar services
`warnHandler`	allows to set a warning handler function, similar to `errorHandler`, but for warnings instead of errors
`ignoredElements`	used to let Vue ignore custom elements defined outside of it, like *Web Components*.
`keyCodes`	let you define custom key aliases for `v-on`
`performance`	defaults to false. If set to true, traces the performance of Vue components in the Browser DevTools.
`productionTip`	defaults to true. Set to false to disable the warning “you’re in development mode” during development in the console.

Methods of the Vue object

Method	Description
`Vue.extend`	allows to subclass the Vue object, to create a custom profile
`Vue.nextTick`	defers the callback to be executed after the next DOM update cycle
`Vue.set`	add a property to the object
`Vue.delete`	delete a property from the object
`Vue.directive`	set (or get) a global directive
`Vue.filter`	set (or get) a global filter
`Vue.component`	set (or get) a global component

Method	Description
`Vue.use`	install a Vue.js plugin
`Vue.mixin`	set a global mixin
`Vue.compile`	compile a template string into a render function
`Vue.version`	returns the currently installed version of Vue

Options passed to a Vue object

When initializing a Vue object, you pass in an object:

```
const vm = new Vue({

})
```

This object accepts a number of properties.

Property	Description
`data`	allows to pass a set of reactive data that will be used by the Vue app. All reactive properties must be added at initialization time, you can't add new ones later.
`props`	it's a set of attributes that are exposed to parent components as input data.
`propsData`	default data for props. Only useful during testing
`methods`	a set of methods that are defined on the Vue instance
`computed`	like methods, but cached internally
`watch`	allows to watch properties, and call a function when they change

Example of defining data, methods and computed properties:

```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  methods: {
    reverseMessageAsMethod: function () {
      return this.message.split('').reverse().join('')
    }
  },
  computed: {
    // a computed getter
    reversedMessage: function () {
      // `this` points to the vm instance
      return this.message.split('').reverse().join('')
    }
  }
})

console.log(vm.reverseMessageAsMethod) // => 'olleH'
vm.message = 'Goodbye'
console.log(vm.reversedMessage) // => 'eybdooG'
```

DOM

- `el` sets the DOM element where the instance mounts on. It can be a CSS Selector, or an HTMLElement
- `template` is a template, represented as a string, that will replace the mounted element
- `render` alternatively to define the template, you can define a template using a render function
- `renderError` set an alternative output when the function attached to `render` fails

Vue instance assets

- `directives` the set of directives to associate to the Vue instance
- `filters` the set of filters to associate to the Vue instance
- `components` the set of components to associate to the Vue instance

Vue composition options

- `parent` specifies the parent instance
- `mixins` sets an array of mixin objects
- `extends` extend another component

Other Vue object options

- `name` setting a name to the component lets you invoke it, useful in debugging or when you need to recursively add a component in its template
- `functional` if true, sets the component to be stateless (no `data`) and instanceless (no `this`), making it more lightweight

- `model` allows to customize the property used in events, useful for example when interacting with forms
- `comments` defaults to false. If set to true, retains the HTML comments that are put in templates

Instance properties

Given an instance of Vue, stored into a variable `const vm = new Vue(/*...*/)`, you can inspect and interact with it.

Properties of a Vue instance

- `vm.$data` the data object associated to the instance
- `vm.$props` the props the instance has received
- `vm.$el` the DOM element to which the instance is bound
- `vm.$options` the object used to instantiate the Vue instance
- `vm.$parent` the parent instance
- `vm.$root` the root instance (if this is the root instance, this points to itself)
- `vm.$children` an array of children instances
- `vm.$slots` an array of the associated slots contained in the template
- `vm.$scopedSlots` an array of the associated scoped slots
- `vm.$refs` an object that contains a property for each element pointed by a `ref` attribute defined in the template
- `vm.$isServer` true if the Vue instance is running on the server (useful in server-side rendering)
- `vm.$attrs` an object of attributes that are provided to the component but not defined as props
- `vm.$listeners` an object of `v-on` event listeners assigned to the component

Methods Data

- `vm.$watch` set up a watcher for property changes in the Vue data. It can also watch for value changes inside objects
- `vm.$set` set a property
- `vm.$delete` delete a property

Events

- `vm.$emit` triggers a custom event on the `vm` Vue instance
- `vm.$on` listen for a custom event on the `vm` Vue instance
- `vm.$once` like `$on`, but listens only once
- `vm.$off` removes an event listener from the Vue instance

Lifecycle Methods

- `vm.$mount` mount a Vue instance on a DOM element, in case it was not mounted yet

- `vm.$forceUpdate` force the `vm` Vue instance to re-render. Does not force child components to rerender.
- `vm.$nextTick` accepts a callback and schedules that for the next DOM update cycle
- `vm.$destroy` destroys the application and remove all child components, observers and listeners