Template Syntax

Vue.js uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying Vue instance's data. All Vue.js templates are valid HTML that can be parsed by spec-compliant browsers and HTML parsers.

Under the hood, Vue compiles the templates into Virtual DOM render functions. Combined with the reactivity system, Vue is able to intelligently figure out the minimal number of components to re-render and apply the minimal amount of DOM manipulations when the app state changes.

If you are familiar with Virtual DOM concepts and prefer the raw power of JavaScript, you can also directly write render functions instead of templates, with optional JSX support.

Interpolations

Text

The most basic form of data binding is text interpolation using the "Mustache" syntax (double curly braces):

```
<span>Message: {{ msg }}</span>
```

The mustache tag will be replaced with the value of the `msg` property on the corresponding data object. It will also be updated whenever the data object's `msg` property changes.

You can also perform one-time interpolations that do not update on data change by using the v-once directive, but keep in mind this will also affect any binding on the same node:

```
<span v-once>This will never change: {{ msg }}</span>
```

Raw HTML

The double mustaches interprets the data as plain text, not HTML. In order to output real HTML, you will need to use the `v-html` directive:

```
<div v-html="rawHtml"></div>
```

The contents of this `div` will be replaced with the value of the `rawHtml` property, interpreted as plain HTML - data bindings are ignored. Note that you cannot use `v-html` to compose template partials, because Vue is not a string-based templating engine. Instead, components are preferred as the fundamental unit for UI reuse and composition.

> Dynamically rendering arbitrary HTML on your website can be very dangerous because it can easily lead to XSS vulnerabilities. Only use HTML interpolation on trusted content and never on user-provided content.

Attributes

Mustaches cannot be used inside HTML attributes, instead use a v-bind directive:

```
<div v-bind:id="dynamicId"></div>
```

It also works for boolean attributes - the attribute will be removed if the condition evaluates to a falsy value:

```
<button v-bind:disabled="isButtonDisabled">Button</button>
```

Using JavaScript Expressions

So far we’ve only been binding to simple property keys in our templates. But Vue.js actually supports the full power of JavaScript expressions inside all data bindings:

```
{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

<div v-bind:id="'list-' + id"></div>
```

These expressions will be evaluated as JavaScript in the data scope of the owner Vue instance. One restriction is that each binding can only contain one single expression, so the following will NOT work:

```
<!-- this is a statement, not an expression: -->

{{ var a = 1 }}

<!-- flow control won't work either, use ternary expressions -->

{{ if (ok) { return message } }}
```

> Template expressions are sandboxed and only have access to a whitelist of globals such as `Math` and `Date`. You should not attempt to access user defined globals in template expressions.

Directives

Directives are special attributes with the `v-` prefix. Directive attribute values are expected to be a single JavaScript expression (with the exception for `v-for`, which will be discussed later). A directive's job is to reactively apply side effects to the DOM when the value of its expression changes. Let's review the example we saw in the introduction:

```
<p v-if="seen">Now you see me</p>
```

Here, the `v-if` directive would remove/insert the `<p>` element based on the truthiness of the value of the expression `seen`.

Arguments

Some directives can take an "argument", denoted by a colon after the directive name. For example, the `v-bind` directive is used to reactively update an HTML attribute:

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

Here `href` is the argument, which tells the `v-bind` directive to bind the element's `href`attribute to the value of the expression `url`.

Another example is the `v-on` directive, which listens to DOM events:

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

Here the argument is the event name to listen to. We will talk about event handling in more detail too.

Modifiers

Modifiers are special postfixes denoted by a dot, which indicate that a directive should be bound in some special way. For example, the `.prevent` modifier tells the `v-on` directive to call `event.preventDefault()` on the triggered event:

```
<form v-on:submit.prevent="onSubmit"> ... </form>
```

You'll see other examples of modifiers later, for `v-on` and for `v-model`, when we explore those features.

Shorthands

The `v-` prefix serves as a visual cue for identifying Vue-specific attributes in your templates. This is useful when you are using Vue.js to apply dynamic behavior to some existing markup, but can feel verbose for some frequently used directives. At the same time, the need for the `v-` prefix becomes less important when you are building an SPA where Vue.js manages every template. Therefore, Vue.js provides special shorthands for two of the most often used directives, `v-bind` and `v-on`:

`v-bind` Shorthand

```html
<!-- full syntax -->

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

<!-- shorthand -->

<a :href="url"> ... </a>
```

`v-on` Shorthand

```html
<!-- full syntax -->

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

<!-- shorthand -->

<a @click="doSomething"> ... </a>
```

They may look a bit different from normal HTML, but `:` and `@` are valid chars for attribute names and all Vue.js supported browsers can parse it correctly. In addition, they do not appear in the final rendered markup. The shorthand syntax is totally optional, but you will likely appreciate it when you learn more about its usage later.