Vue Methods

A Vue method is a function associated with the Vue instance.

Methods are defined inside the methods property:

```
new Vue({

 methods: {

  handleClick: function() {

    alert('test')

  }

 }

})
```

Methods are especially useful when you need to perform an action and you attach a v-on directive on an element to handle events. Like this one, which calls handleClick when the element is clicked:

```
<template>

 <a @click="handleClick">Click me!</a>

</template>
```

Pass parameters to Vue.js methods

Methods can accept parameters.

In this case, you just pass the parameter in the template:

```
<template>

 <a @click="handleClick('something')">Click me!</a>

</template>
new Vue({
 methods: {
  handleClick: function(text) {
   alert(text)
  }
 }
})
```

How to access data from a method

You can access any of the data properties of the Vue component by using this.propertyName:

```
<template>

 <a @click="handleClick()">Click me!</a>

</template>
<script>
export default {
 data() {
  return {
```

```
    name: 'Flavio'
  }
 },
 methods: {
  handleClick: function() {
    console.log(this.name)
  }
 }
}
</script>
```

We don’t have to use this.data.name, just this.name. Vue does provide a transparent binding for us. Using this.data.name will raise an error.

Vue Events

v-on allows you to listen to DOM events, and trigger a method when the event happens. Here we listen for a click event:

```
<template>

 <a v-on:click="handleClick">Click me!</a>

</template>
<script>
export default {
 methods: {
  handleClick: function() {
    alert('test')
```

```
  }
 }
}
</script>
```

You can pass parameters to any event:

```
<template>

 <a v-on:click="handleClick('test')">Click me!</a>

</template>
<script>
export default {
 methods: {
  handleClick: function(value) {
   alert(value)
  }
 }
}
</script>
```

Small bits of JavaScript (a single expression) can be put directly into the template:

```
<template>

 <a v-on:click="counter = counter + 1">{{counter}}</a>

</template>
<script>
export default {
 data: function() {
  return {
```

```
    counter: 0
  }
 }
}
</script>
```

click is just one kind of event. A common event is submit, which you can bind using v-on:submit.

v-on is so common that there is a shorthand syntax for it, @:

Event directive modifiers

Vue offers some optional event modifiers you can use in association with v-on, which automatically make the event do something without you explicitly coding it in your event handler.

One good example is .prevent, which automatically calls preventDefault()on the event.

In this case, the form does not cause the page to be reloaded, which is the default behavior:

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

Other modifiers include .stop, .capture, .self, .once, .passive and they are described in detail in the official docs.

```
<!-- the click event's propagation will be stopped -->
<a v-on:click.stop="doThis"></a>

<!-- the submit event will no longer reload the page -->
```

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

<!-- modifiers can be chained -->
<a v-on:click.stop.prevent="doThat"></a>

<!-- just the modifier -->
<form v-on:submit.prevent></form>

<!-- the click event will be triggered at most once -->
<a v-on:click.once="doThis"></a>
```

Key Modifiers

When listening for keyboard events, we often need to check for specific keys. Vue allows adding key modifiers for `v-on` when listening for key events:

```
<!-- only call `vm.submit()` when the `key` is `Enter` -->
<input v-on:keyup.enter="submit">
```

You can directly use any valid key names exposed via KeyboardEvent.key as modifiers by converting them to kebab-case.

```
<input v-on:keyup.page-down="onPageDown">
```

In the above example, the handler will only be called if `$event.key` is equal to `'PageDown'`.

Key Codes

> The use of keyCode events is deprecated and may not be supported in new browsers.

Using keyCode attributes is also permitted:

```
<input v-on:keyup.13="submit">
```

Vue provides aliases for the most commonly used key codes when necessary for legacy browser support:

- .enter
- .tab
- .delete (captures both “Delete” and “Backspace” keys)
- .esc
- .space
- .up
- .down
- .left
- .right

> A few keys (.esc and all arrow keys) have inconsistent key values in IE9, so these built-in aliases should be preferred if you need to support IE9.

You can also define custom key modifier aliases via the global config.keyCodes object:

```
// enable `v-on:keyup.f1`
Vue.config.keyCodes.f1 = 112
```

Vue Filters

Filters are a functionality provided by Vue components that let you apply formatting and transformations to any part of your template dynamic data.

They don't change a component's data or anything, but they only affect the output.

Say you are printing a name:

```
<template>

 <p>Hi {{ name }}!</p>

</template>
<script>
export default {
 data() {
  return {
    name: 'Flavio'
  }
 }
}
</script>
```

What if you want to check that name is actually containing a value, and if not print 'there', so that our template will print "Hi there!"?

Enter filters:

```
<template>

 <p>Hi {{ name | fallback }}!</p>

</template>
<script>
export default {
 data() {
  return {
   name: 'Flavio'
  }
 },
 filters: {
  fallback: function(name) {
   return name ? name : 'there'
  }
 }
}
</script>
```

Notice the syntax to apply a filter, which is | filterName.

A single filter is a function that accepts a value and returns another value.

The returned value is the one that's actually printed in the Vue.js template.

Filters can be chained, by repeating the pipe syntax:

{{ name | fallback | capitalize }}

This first applies the fallback filter, then the capitalize filter

Advanced filters can also accept parameters, using the normal function parameters syntax:

```
<template>

 <p>Hello {{ name | prepend('Dr.') }}</p>

</template>
<script>
export default {
 data() {
  return {
   name: 'House'
  }
 },
 filters: {
  Prepend: function (name, prefix) {
   return `${prefix} ${name}`
  }
 }
}
</script>
```

If you pass parameters to a filter, the first one passed to the filter function is always the item in the template interpolation (name in this case), followed by the explicit parameters you passed.

Vue Life-Cycle

All lifecycle hooks automatically have their this context bound to the instance, so that you can access data, computed properties, and methods.