Vue.js

Part II
Components

Basic Building Blocks
<template>
<template>
  <div id="app">
    <img alt="Vue logo" src="/assets/logo.png">
    <HelloWorld msg="Welcome to Your Vue.js App"/>
  </div>
</template>

<script>
  import HelloWorld from './components/HelloWorld.vue'

  export default {
    name: 'app',
    components: {
      HelloWorld
    }
  }
</script>
props: {}
<template>
    <div class="hello">
        <h1>{{ msg }}</h1>
    </div>
</template>

<script>
    export default {
        name: 'HelloWorld',
        props: {
            msg: String,
        }
    }
</script>
props: {
  msg: String,
  // type check plus other validations
  age: {
    type: Number,
    default: 0,
    required: true,
    validator: function (value) {
      return value >= 0
    }
  }
}
Type

Any custom constructor function or an array of these. Will check if a prop has a given type, and will throw a warning if it doesn’t.

- String
- Number
- Boolean
- Array
- Object
- Date
- Function
- Symbol
Default

Specifies a default value for the prop. If the prop is not passed, this value will be used instead.

Object or array defaults must be returned from a factory function.

- **String** - “Some String”
- **Number** - 10
- **Boolean** - true
- **Array** - () => { return [1, 2, 3] }
- **Object** - () => { return new MyObject() }
- **Date** - Date.now()
- **Function** - () => { return null }
- **Symbol** - Symbol(“text”)
Required

Defines if the prop is required. In a non-production environment, a console warning will be thrown if this value is truthy and the prop is not passed.

- **Boolean**: true / false
Custom validator function that takes the prop value as the sole argument. In a non-production environment, a console warning will be thrown if this function returns a falsy value (i.e. the validation fails).

```javascript
function (value) {
  return value >= 0
}
```
props: {
    msg: String,
    // type check plus other validations
    age: {
        type: Number,
        default: 0,
        required: true,
        validator: function (value) {
            return value >= 0
        }
    }
}
data
Data

When defining a component, data must be declared as a function that returns the initial data object, because there will be many instances created using the same definition. If we use a plain object for data, that same object will be shared by reference across all instances created! By providing a data function, every time a new instance is created we can call it to return a fresh copy of the initial data.
How Changes Are Tracked

- Component Render Function
- Watcher
- Trigger re-render
- Collect as Dependency
- Notify
- Virtual DOM Tree
- “Touch”
- Data
  - getter
  - setter
methods
Methods

Methods to be mixed into the Vue instance. You can access these methods directly on the VM instance, or use them in directive expressions. All methods will have their `this` context automatically bound to the Vue instance.
Lifecycle hooks
beforeCreate
Called synchronously immediately after the instance has been initialized, before data observation and event/watcher setup.

created
Called synchronously after the instance is created. At this stage, the instance has finished processing the options which means the following have been set up: data observation, computed properties, methods, watch/event callbacks. However, the mounting phase has not been started, and the $el property will not be available yet.

beforeMount
Called right before the mounting begins: the render function is about to be called for the first time.

beforeUpdate
Called when data changes, before the DOM is patched. This is a good place to access the existing DOM before an update.

mounted
Called after the instance has been mounted, where el is replaced by the newly created vm.$el. If the root instance is mounted to an in-document element, vm.$el will also be in-document when mounted is called.

updated
Called after a data change causes the virtual DOM to be re-rendered and patched. The component’s DOM will have been updated when this hook is called, so you can perform DOM-dependent operations here. However, in most cases you should avoid changing state inside the hook.
beforeDestroy
Called right before a Vue instance is destroyed. At this stage the instance is still fully functional.

destroyed
Called after a Vue instance has been destroyed. When this hook is called, all directives of the Vue instance have been unbound, all event listeners have been removed, and all child Vue instances have also been destroyed.
Directives
**Expects**: string

Updates the element's *textContent*. If you need to update the part of *textContent*, you should use `{{ Mustache }}` interpolations.

```html
<span v-text="msg"></span>
<!-- same as -->
<span>{{msg}}</span>
```
**Expects:** string

Updates the element’s `innerHTML`. Note that the contents are inserted as plain HTML - they will not be compiled as Vue templates. If you find yourself trying to compose templates using `v-html`, try to rethink the solution by using components instead.

```html
<div v-html=""<h1>Hello</h1>""> </div>
```
**Expects:** *any*

Toggles the element’s `display` CSS property based on the truthy-ness of the expression value.

This directive triggers transitions when its condition changes.

```
<h1 v-show="true">Hello!</h1>
```
**Expects:** any

Conditionally render the element based on the truthy-ness of the expression value. The element and its contained directives / components are destroyed and re-constructed during toggles. If the element is a `<template>` element, its content will be extracted as the conditional block.

This directive triggers transitions when its condition changes.

```html
<div v-if="Math.random() > 0.5">
  Now you see me
</div>
```
Does not expect expression

**Restriction**: previous sibling element must have `v-if` or `v-else-if`.

Denotes the “else block” for `v-if` or a `v-if/v-else-if` chain.

```html
<div v-if="Math.random() > 0.5">
  Now you see me
</div>
<div v-else>
  Now you don't
</div>
```
v-else-if

**Expects:** any

**Restriction:** previous sibling element must have *v-if* or *v-else-if*.

Denotes the “else if block” for *v-if*. Can be chained.

```html
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
```
**Expects:** Array | Object | number | string | Iterable (since 2.6)

Render the element or template block multiple times based on the source data. The directive’s value must use the special syntax `alias in expression` to provide an alias for the current element being iterated on:

```html
<div v-for="item in items">
  {{ item.text }}
</div>

<div v-for="(item, index) in items"></div>

<div v-for="(val, key) in object"></div>

<div v-for="(val, name, index) in object"></div>
```
Shorthand: @

**Expects:** Function | Inline Statement | Object

**Argument:** event

Attaches an event listener to the element. The event type is denoted by the argument. The expression can be a method name, an inline statement, or omitted if there are modifiers present.

When used on a normal element, it listens to native DOM events only. When used on a custom element component, it listens to custom events emitted on that child component.

When listening to native DOM events, the method receives the native event as the only argument. If using inline statement, the statement has access to the special $event property: `v-on:click="handle('ok', $event)"`. 
v-on

<!-- method handler -->
<button v-on:click="doThis"></button>

<!-- inline statement -->
<button v-on:click="doThat('hello', $event)"></button>

<!-- shorthand -->
<button @click="doThis"></button>

<!-- key modifier using keyAlias -->
<input @keyup.enter="onEnter">

<!-- key modifier using keyCode -->
<input @keyup.13="onEnter"
**v-bind**

**Shorthand:** :  

**Expects:** any (with argument) | Object (without argument)

Dynamically bind one or more attributes, or a component prop to an expression. When used to bind the `class` or `style` attribute, it supports additional value types such as Array or Objects. When used for prop binding, the prop must be properly declared in the child component.

When used without an argument, can be used to bind an object containing attribute name-value pairs.
v-bind

<!-- bind an attribute -->
<img v-bind:src="imageSrc">

<!-- shorthand -->
<img :src="imageSrc">

<!-- with inline string concatenation -->
<img :src="/path/to/images/ + fileName">

<!-- class binding -->
<div :class="{ red: isRed }"></div>
<div :class="[classA, { classB: isB, classC: isC }]"></div>

<!-- style binding -->
<div :style="{ fontSize: size + 'px' }"></div>

<!-- binding an object of attributes -->
<div v-bind="{ id: someProp, 'other-attr': otherProp }"></div>
References

https://vuejs.org/v2/api/
Questions?

Next: Vue.js III