

Node Imports & Vue.js

A review with some tips and info of our week within vue and node.js





Vue.js

Vue.js is an open-source model-view-viewmodel JavaScript framework for building user interfaces and single-page applications. It was created by Evan You, and is maintained by him and the rest of the active core team members coming from various companies such as Netlify and Netguru.





Webpack

Webpack is a static module bundler for JavaScript applications — it takes all the code from your application and makes it usable in a web browser. Modules are reusable chunks of code built from your app's JavaScript, `node_modules`, images, and the CSS styles which are packaged to be easily used in your website.



webpack




require() & import

We can import node modules into our application with require() and import. This makes the library or plugin available within our application. No CDN's required.

```
// All of the imports/requires
var axios = require('axios');
// A special/work-around kind of import/require for vue.js
// Vue needs additional files and build tools so this way works
// for our current gulp/webpack setup:
import Vue from 'vue/dist/vue.js';
import tippy from 'tippy.js';
import anime from 'animejs/lib/anime.es.js';

(function(){

// Vue World
var app = new Vue({
  el: '#app',
  data: {
    message: 'News Items'
```



Vue data - very important & useful

Try to use the vue data object as much as possible. It can be used to create switch logic (true/false) to update the DOM/UI, store strings, api data etc. The more you use the data object, the more you will find ***vue.js** to be useful.*

When this **data** changes, the view will re-render. It should be noted that properties in **data** are only reactive if they existed when the instance was created. That means if you add a new property, like:

```
vm.b = 'hi'
```

JS

Then changes to **b** will not trigger any view updates. If you know you'll need a property later, but it starts out empty or non-existent, you'll need to set some initial value. For example:

```
data: {  
  newTodoText: '',  
  visitCount: 0,  
  hideCompletedTodos: false,  
  todos: [],  
  error: null  
}
```

JS



v-if

A v-if binding will conditionally render something in the DOM. Render meaning the html will not exist unless the condition is true.

```
<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>
```



v-show

The difference is that an element with v-show will always be rendered and remain in the DOM; v-show only toggles the display CSS property of the element. V-if will not render the element at all, unless true.

```
<h1 v-show="ok">Hello!</h1>
```



Vue.js v-for

We can use the v-for directive to render a list of items based on an array. The v-for directive requires a special syntax in the form of `item in items`, where `items` is the source data array and `item` is an alias for the array element being iterated on:

Mapping an Array to Elements with `v-for`


We can use the `v-for` directive to render a list of items based on an array. The `v-for` directive requires a special syntax in the form of `item in items`, where `items` is the source data array and `item` is an alias for the array element being iterated on:

```
<ul id="example-1">
  <li v-for="item in items" :key="item.message">
    {{ item.message }}
  </li>
</ul>
```

HTML

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { message: 'Foo' },
      { message: 'Bar' }
    ]
  }
})
```

JS



Vue events = inside html

Vue allows you to place events within the html. This speeds up development, less code = better && faster.

```
<div id="example-2">
  <!-- `greet` is the name of a method defined below -->
  <button v-on:click=greet>Greet</button>
</div>
```

HTML

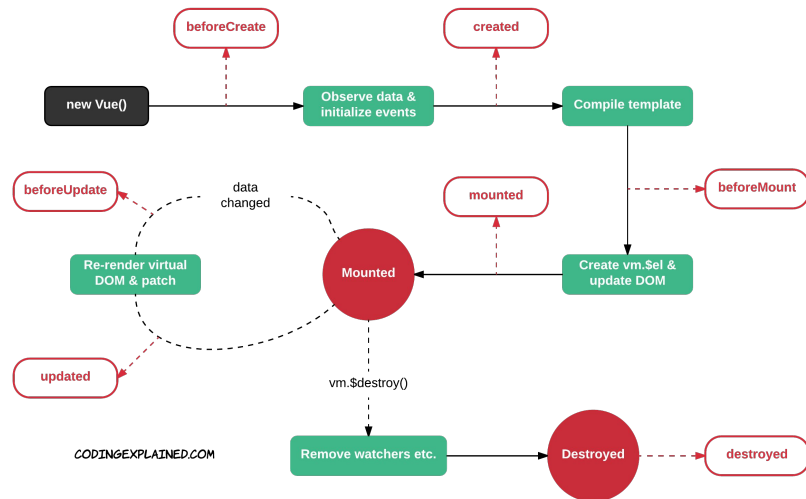
```
var example2 = new Vue({
  el: '#example-2',
  data: {
    name: 'Vue.js'
  },
  // define methods under the `methods` object
  methods: {
    greet: function (event) {
      // `this` inside methods points to the Vue instance
      alert('Hello ' + this.name + '!')
      // `event` is the native DOM event
      if (event) {
        alert(event.target.tagName)
      }
    }
  }
})
```

JS

*Remember... you can always go Super Lazy Dev Status and use
@click="greet"*

Vue lifecycle hooks

Vue has inbuilt methods that you can use to start code. 'created' and 'mounted' can be useful tools if you want to start code after the Vue application is ready and rendered.



```
// Vue World
var app = new Vue({
  el: '#app',
  data: {
    message: 'News Items',
    aboutPage: 'About Page',
    news: false,
    display: true,
    displayPanel: false
  },
  mounted () {
    console.log('vue is mounted')
  },
  methods: {
    goAboutPage: function () {
      console.log('working switch')
      this.display = false;
    },
    goHomePage: function () {
      console.log('working switch')
      this.display = true;
    }
  }
})
```



Vue filters

Vue provides internal processing and render methods called **filters**. Filters are primarily used for processing strings and numbers for final output. E.g. - changing a date, applying a currency, capitalizing words and switching values are all examples of filters.

```
filters: {  
  // A filter that reverses text  
  reverseText: function (value) {  
    // This reverses the text  
    return value.split('').reverse().join('')  
  },  
}
```

```
d. -->  
{{item.description | reverseText}}
```

In the above we create a filter called 'reverseText' then use it in the declarative render below.



Vue methods

You can write custom functions/methods within vue. This will allow you to control your application in any way you like....

We can use the `v-on` directive to listen to DOM events:

```
<div id="example">
  <button v-on:click="greet">Greet</button>
</div>
```

HTML

We are binding a click event listener to a method named `greet`. Here's how to define that method in our Vue instance:

```
var vm = new Vue({
  el: '#example',
  data: {
    name: 'Vue.js'
  },
  // define methods under the 'methods' object
  methods: {
    greet: function (event) {
      // 'this' inside methods point to the Vue instance
      alert('Hello ' + this.name + '!')
      // 'event' is the native DOM event
      alert(event.target.tagName)
    }
  }
})
// you can invoke methods in JavaScript too
vm.greet() // -> 'Hello Vue.js!'
```

JS



Vue Components

Components are reusable Vue instances with a name: in this case. We can use components as a custom element inside a root Vue instance created with new Vue.

Here's an example of a Vue component:

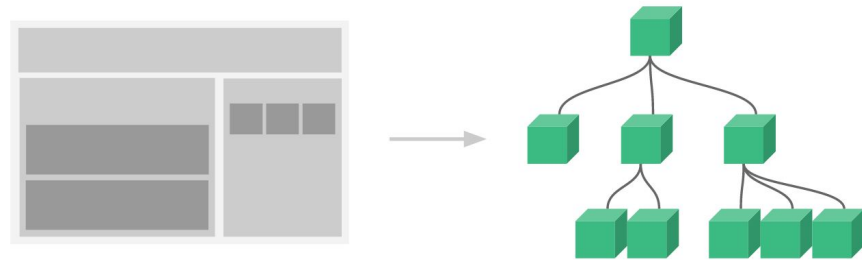
```
JS
// Define a new component called button-counter
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button v-on:click="count++">You clicked me {{ count }} times.</button>'
})
```

Components are reusable Vue instances with a name: in this case, `<button-counter>`. We can use this component as a custom element inside a root Vue instance created with `new Vue`:

```
HTML
<div id="components-demo">
  <button-counter></button-counter>
</div>
```

Why Components?

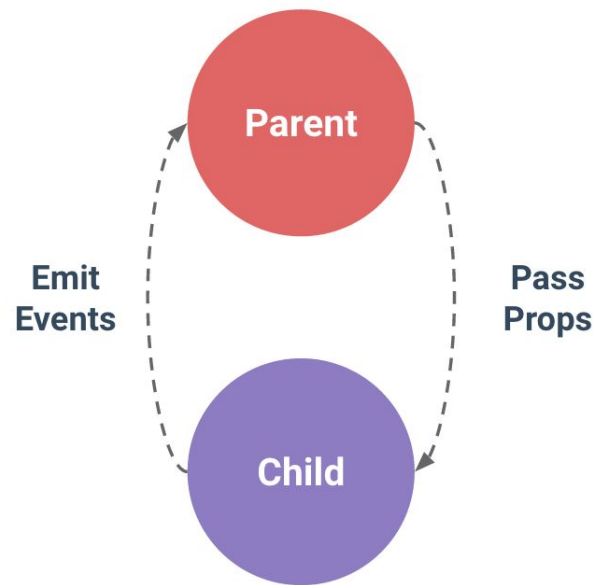
Components are reusable and help us to modularize an application. This means separating logic, style and structure into mini packets or modules of code. The goal is to make an app more understandable and easy to manage for any possible updates or changes.



Think of ways to separate your app into modules in order to make it more logical and easy to manage. Only use components when needed, to make a large app more structured.

Component communication

Because components are separate entities we need to use special syntax and approaches for sending data and instructions from one to the other. To do this we can use props and the `$emit` event process.



We pass emit **events** from the child *up to the parent*. We pass **props** *down from the parent to the child*.



Vue Event \$emit

To pass data or an instruction from a child component to a parent use `this.$emit()`.

***Note:** You may need to write additional code in the html or `vue.instance/methods`.

```
// This statement emits an event  
this.$emit('about-go', false)
```

```
<home-button @about-go="receiveEmitData">
```

Here we are using the \$emit method within a vue method, then referencing it on the vue.html.



Vue props

Vue props are how we share data from a parent to a child component. The process is easier than `$emit()`. We simply declare the prop in the component, then reference it in the html with a bind. Note the prop refers to data coming from a parent component.

```
new Vue({  
  el: '#blog-posts-events-demo',  
  data: {  
    posts: [/* ... */],  
    postFontSize: 1  
  }  
})
```

```
<div id="blog-posts-events-demo">  
  <div :style="{ fontSize: postFontSize + 'em' }">  
    <blog-post  
      v-for="post in posts"  
      v-bind:key="post.id"  
      v-bind:post="post"  
    ></blog-post>  
  </div>  
</div>
```