VUE.JS 入門

Vue 2 新手必備的姿勢

太難的我都不會

太深入的我也不會

大神 Kuro 整理的簡報

所以接下來我會簡介這些主題

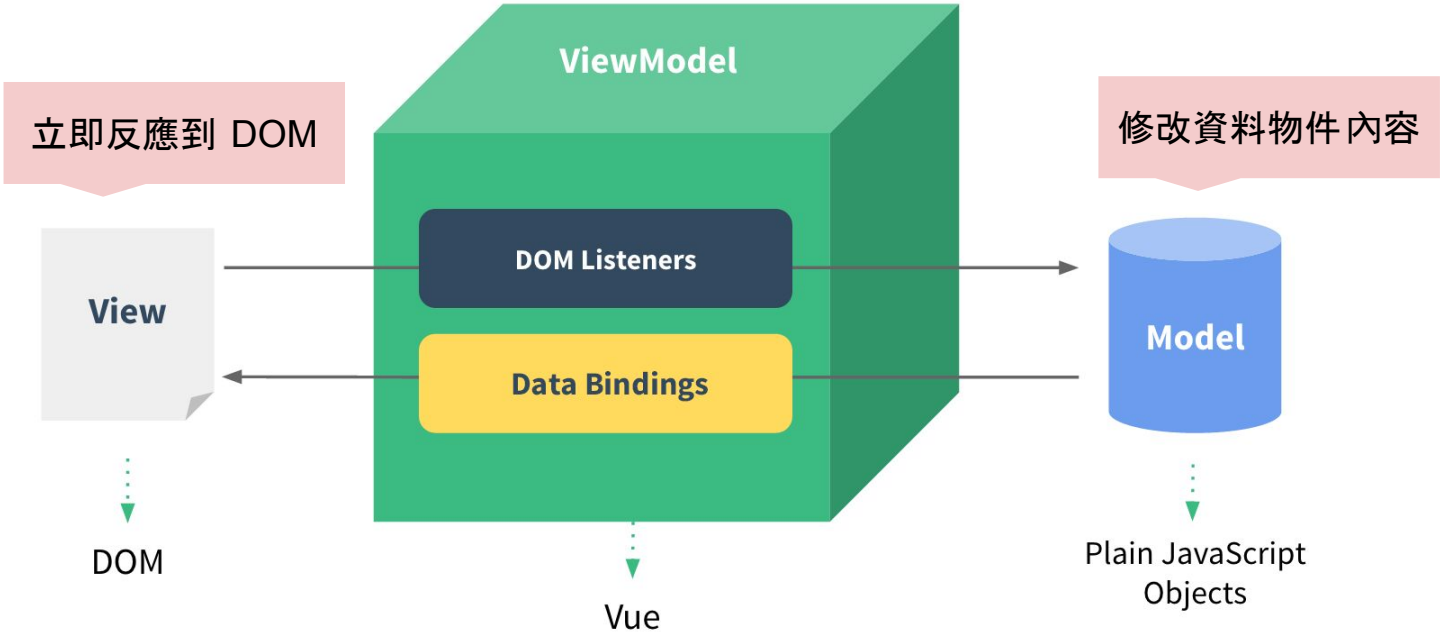
- 認識 Vue.js
- 開發必備利器
- 把 Vue.js 放進專案裡
- 撰寫 Vue 元件
- 常用的 Vue 寫法
- 父元件與子元件的溝通方法
- 元件狀態管理
- SPA 路由

認識 Vue.js

Vue.js 的目標與特色

- 儘可能簡單的 API (Simple API)
- 響應式的資料綁定 (Reactive Data Binding)
- 可組合式的視圖元件 (Composable View Components)
- 只專注在視圖層
- 可以跟其他 library 整合
- 完美開發單頁應用程式 (SPA; Single Page Application)

Reactive Data Binding

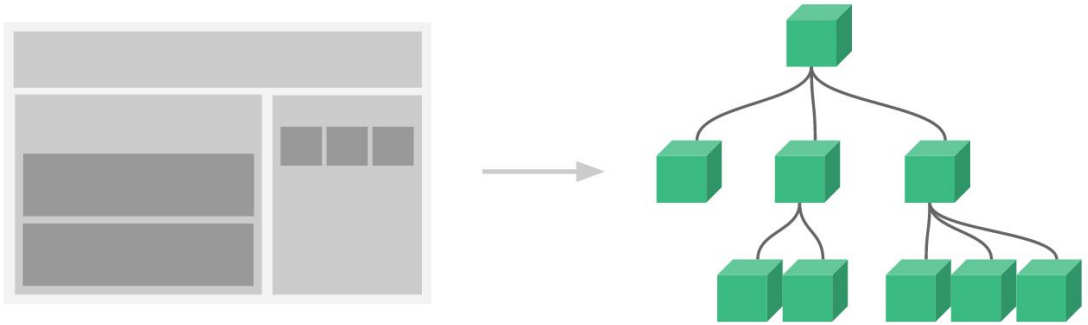


```
<div id="app">
 <h1>{{ title }}</h1>
 <p>{{ message }}</p>
 <input type="text" v-model="message" />
</div>

<script src="https://cdn.jsdelivr.net/vue/latest/vue.js"></script>
<script>
new Vue({
 el: '#app',
 data() {
  return {
   title: 'My first Vue app',
   message: 'Hello Vue.js!'
  };
 }
});
</script>
```

執行結果

Composable View Components

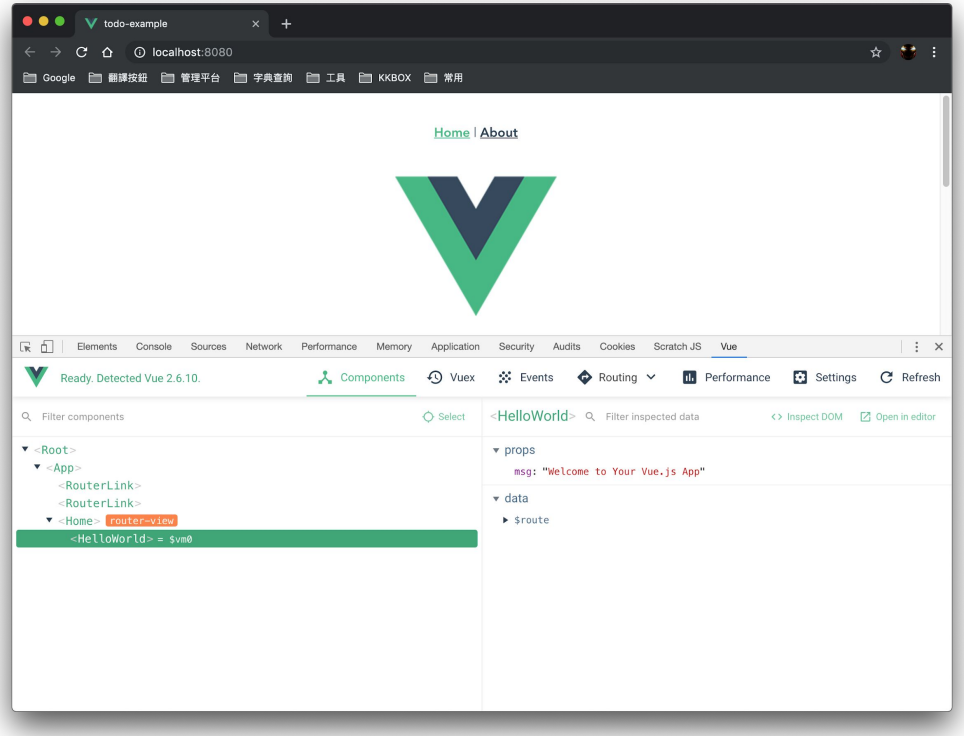


分析頁面元件

轉換成樹狀結構

開發必備利器

Chrome Extension: Vue.js devtools



IDE 支援

- VS Code - Vetur
  - Extensions > Search Extensions in Marketplace > Vetur
- WebStorm / PhpStorm - Vue.js
  - Preferences > Plugins > Marketplace > Search plugins in marketplace > Vue.js

把 Vue.js 放進專案裡

方法一：直接引用 Vue.js (不推薦)

```
<body>
<div id="app"></div> <!-- 掛載點 -->

<script src="https://cdn.jsdelivr.net/vue/latest/vue.js"></script>
<script>
new Vue({
   template: "<h1>Hello World!!</h1>"
}).$mount("#app");
</script>

</body>

// 這時候 Vue 是運行在 development 模式
// 會包含 template compiler
// 在 production 上用時，要把 vue.js 換成 vue.min.js
```

方法二：用 vue-cli 3 建立專案 (推薦)

```
$ npm install -g @vue/cli # 安裝全域可用的 Vue CLI 工具

$ vue create example # 建立一個名稱為 example 的專案

$ cd example # 進入專案資料夾

$ yarn serve # 啟動 Webpack Dev Server

# 這時候 Vue 是運行在 runtime-only build
# 不包含 template compiler
```

VS Code / PhpStorm 讓 vue-cli 更好用

- VS Code
  - 認得 webpack 的 resolve alias
  - 自動在檔案存檔時做 eslint (prettier) fix / Vue file
- PhpStorm/WebStorm
  - 認得 webpack 的 resolve alias
  - 自動在檔案存檔時做 eslint (prettier) fix

註：resolve alias 是指在 import 時可以把 src 取代為 @，而不需要使用到相對路徑。

撰寫 Vue 元件

常用的基本元件屬性

```
new Vue({
 el,          // 要綁定的 DOM element ，用 CSS Selector 語法
              // 只用用 new Vue() 時需要，在 Vue file 裡不必指定
 data,        // 要綁定的資料， Vue.extend() 或 Vue file 中的 data 必須是函式
 props,       // 可用來接收父元件資料的屬性
 template,    // 樣版內容，必須是字串，而且必須在有 template compiler 的環境使用
 computed,    // 定義資料的 getter ，即需要計算後才能使用的屬性
 watch,       // 監控 data 或 props 裡的資料變化
 components,  // 定義子元件，可用 ES6 簡寫法，例如 { HelloWord }
 methods,     // 定義可以在元件或樣版內使用的方法
});
```

可自成一格的 Vue File (.vue) (強力推薦)

- 類似 Web Component
  - HTML + Style + Script
- 讓 Vue File 能被 import 的兩種方法
  - Browserify + Vueify (目前較少使用)
  - Webpack + vue-loader (vue-cli 預設已安裝)

Vue File 基本形式

```
<!-- Template -->
<template>
 <HelloWorld msg="Hello Vue.js!" />
</template>

<!-- Script -->
<script>
import HelloWorld from '@/components/HelloWorld.vue';
export default {
 components: { HelloWorld }
};
</script>

<!-- Style -->
<style> h1 { color: red; } </style>
<!-- 或 -->
<style src="./assets/example.css"></style>
```

Vue CLI 的元件樹狀結構

根元件

```
<!-- index.html -->

<body>
 <div id="app"></div>
 <!-- built files
      will be auto injected -->
</body>
```

```
// main.js

import Vue from "vue";
import App from "@/App.vue";

new Vue({
  render: h => h(App)
}).$mount("#app");
```

父元件 (App.vue)

```
<template>
 <div id="app">
  <HelloWorld
   msg="Welcome to
        Your Vue.js App" />
 </div>
</template>
```

```
<script>
import HelloWorld from
"./components/HelloWorld.vue";

export default {
 name: "app",
 components: {
  HelloWorld
 }
};
</script>
```

子元件 (components/HelloWorld.vue)

```
<template>
 <div class="hello">
  <h1>{{ msg }}</h1>
 </div>
</template>

<script>
export default {
 name: "HelloWorld",
 props: {
  msg: String
 }
};
</script>
```

全域元件

```
// main.js

import HelloWorld from "@/components/HelloWorld.vue";

Vue.component("HelloWorld", HelloWorld);
```

為什麼無法使用 template 屬性？

```
Vue.component("HelloWorld", {
 props: ["msg"],
 template: '<div class="hello"><h1>{{ msg }}</h1></div>'
});

// Vue CLI 預設是使用 runtime-only build，
// 它並不支援 template compiler，
// 只能改用 import .vue file 的方式來建立元件。
// 參考：https://www.imooc.com/article/17868
```

常用的 Vue 寫法

說真的，官方文件寫得很好

精華都在這兩份文件裡

- 官方教學
- 官方 API 文件

常用的 Lifecycle Hooks (Vue 2.0)

```
export default {
 created() { /* ... */ },  // 已可存取 this.$data
 mounted() { /* ... */ },  // 已可存取 this.$el
 updated() { /* ... */ },  // 狀態已更新
 beforeDestroy() { /* ... */ },  // 可以用來清除事件綁定
 destroyed() { /* ... */ } // 實例被銷毀
};
```

Data Binding

```
{{ text }} <!-- 永遠會綁定其值的變化 -->

<div v-once>{{ text }}</div> <!-- 第一次綁定後就不會再變化了 -->

<div v-html>{{ raw_html }}</div> <!-- 用來顯示 HTML -->

<div id="{{ list.id }}"></div> <!-- 這樣是錯誤的 -->

<div v-bind:id="list.id"></div> <!-- 用 v-bind 在標籤屬性中綁定 -->

註: v-bind 的值規則和 {{ ... }} 一致
```

Expression Binding

```
<!-- 合法 -->
{{ number + 1 }}
{{ ok ? 'YES' : 'NO' }}
{{ message.split('').reverse().join('') }}

<!-- 不合法 -->
{{ var a = 1 }}
{{ if (OK) { return message; } }}
{{ function () { return 'test'; } }}
```

註:簡單來說合法的 expresion 必須是 one single expression

猜猜以下這行是否合法？

```
{{ (a = 1 + (function(x) { if (x > 1) return 10; })(20)) }}
```

Directives

```
<!-- 控制是否出現 -->
<p v-if="greeting">Hello!</p>
<p v-show="greeting">Hello!</p>

<!-- 綁定資料 -->
<a v-bind:href="url"></a>
<!-- 簡易寫法 -->
<a :href="url"></a>

<!-- 設定事件回呼函式 -->
<a v-on:click="doSomething"></a>
<!-- 簡易寫法 -->
<a @click="doSomething"></a>
```

Modifiers

```
<!-- event.stopPropagation() -->
<a @click.stop="doThis"></a>

<!-- event.preventDefault() -->
<form @submit.prevent="onSubmit"></form>
<form @submit.prevent></form>

<!-- 組合兩個 modifier -->
<a @click.stop.prevent="doThat">

<!-- 按下 Enter -->
<input @keyup.enter="submit">
```

Computed Properties

```
<template>
 <div class="multiplication">
  <input v-model="factor1" />
  *
  <input v-model="factor2" />
  = {{ product }}
 </div>
</template>
```

```
<script>
export default {
 data() {
  return {
   factor1: 5, factor2: 10
  };
 },
 computed: {
  product() {
   return this.factor1 * this.factor2;
  }
 }
};
</script>
```

Watcher

```
export default {
 name: "HelloWorld",
 props: { msg: String },
 data() {
  return { title: "" }
 },
 watch: {
  // 第一個參數是改變後的新 值, 第二個參數是改變前的 值
  msg(newVal, oldVal) { /* ... */ },
  title(newVal, oldVal) { /* ... */ }
 }
};
```

CSS Class Binding

```
<div
 :class="['bold', {
  'italic': isItalic,
  'underline': isUnderline
 }]">
</div>

<!-- 輸出 -->
<div class="bold italic">
</div>
```

```
export default {
 data() {
  return {
   isItalic: true,
   isUnderline: false
  };
 }
};
```

v-for

```
<div v-for="item in items" :key="item.id">
 {{ item.message }}
</div>
<div v-for="item of items" :key="item.id">
 {{ item.message }}
</div>
<div v-for="item of items" :key="item.id">
 {{ $index }} {{ item.message }}
</div>
<div v-for="(item, index) in items" :key="item.id">
 {{ index }} {{ item.message }}
</div>

<!-- v-for 建議一定要跟著 :key，這是為了效能考量 -->
```

v-for (Template)

```
<template v-for="(item, index) in items">
 <!-- 特別注意 :key 不能綁在 template 上，只能放在內部的第一層標籤 -->
 <div :key="item.id">
  <div>{{ index }}</div>
  <div>{{ item.message }}</div>
 </div>
</template>
```

Methods and Event Handling

```
<template>
 <button @click="greet">
  Greet
 </button>
</template>
```

```
<script>
export default {
 data() {
  return { name: 'Vue.js' };
 },
 methods: {
  greet(event) {
   alert('Hello ' + this.name + '!');
   alert(event.target.tagName);
  }
 }
};
</script>
```

Instance Properties & Methods

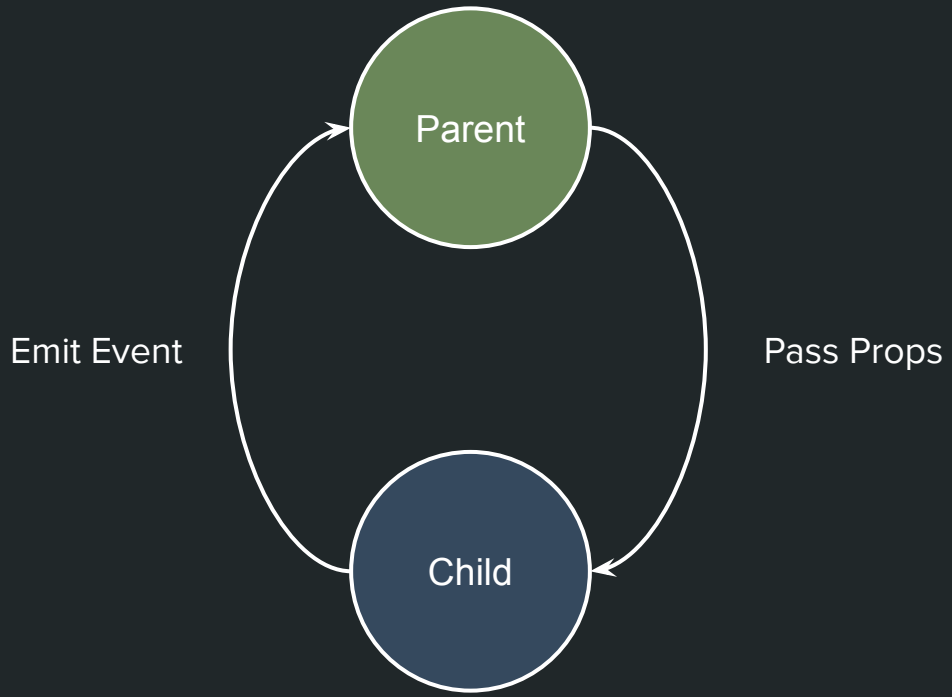
```
// 直接把 Vue 實例當成物件來操作，而不綁到 DOM 上
var vm = new Vue({
 data() {
  return { player: { name: 'Hero' }};
 }
});

console.log(vm.$data);
console.log(vm.$el);

vm.$watch('player.name', function (newVal, oldVal) {
 console.log(newVal, oldVal)
});
```

父元件與子元件的溝通方法

基本概念



Parent to Child (props)

```
// App.vue (Parent)
<template>
 <HelloWorld :msg="message" />
</template>

<script>
import HelloWorld from
 "@/components/HelloWorld.vue";
export default {
 data() {
  return { message: "Hello!!" };
 },
 components: { HelloWorld }
};
</script>
```

```
// HelloWorld.vue (Child)
<template>
 <h1>{{ msg }}</h1>
</template>

<script>
export default {
 props: {
   msg: String
 }
};
</script>
```

Parent to Child (props)

```
// App.vue (Parent)
<template>
 <HelloWorld :msg="message" />
</template>

<script>
import HelloWorld from
 "@/components/HelloWorld.vue";
export default {
 data() {
  return { message: "Hello!!" };
 },
 components: { HelloWorld }
};
</script>
```

```
// HelloWorld.vue (Child)
<template>
 <h1>{{ msg }}</h1>
</template>

<script>
export default {
 props: {
   msg: String
 }
};
</script>
```

Child to Parent (event)

```
// MyForm.vue (Child)
<template>
 <input @keyup.enter="sendMessage">
</template>
<script>
export default {
 methods: {
  sendMessage() {
   this.$emit("send-message",
              "message")
   }
 }
}
</script>
```

```
// Main.vue (Parent)
<template>
 <my-form
  @send-message="appendMessage" />
</template>

<script>
import MyForm from './MyForm.vue'
export default {
 methods: {
  appendMessage(msg) { /* ... */ }
 },
 components: { MyForm }
}
</script>
```

Child to Parent (event)

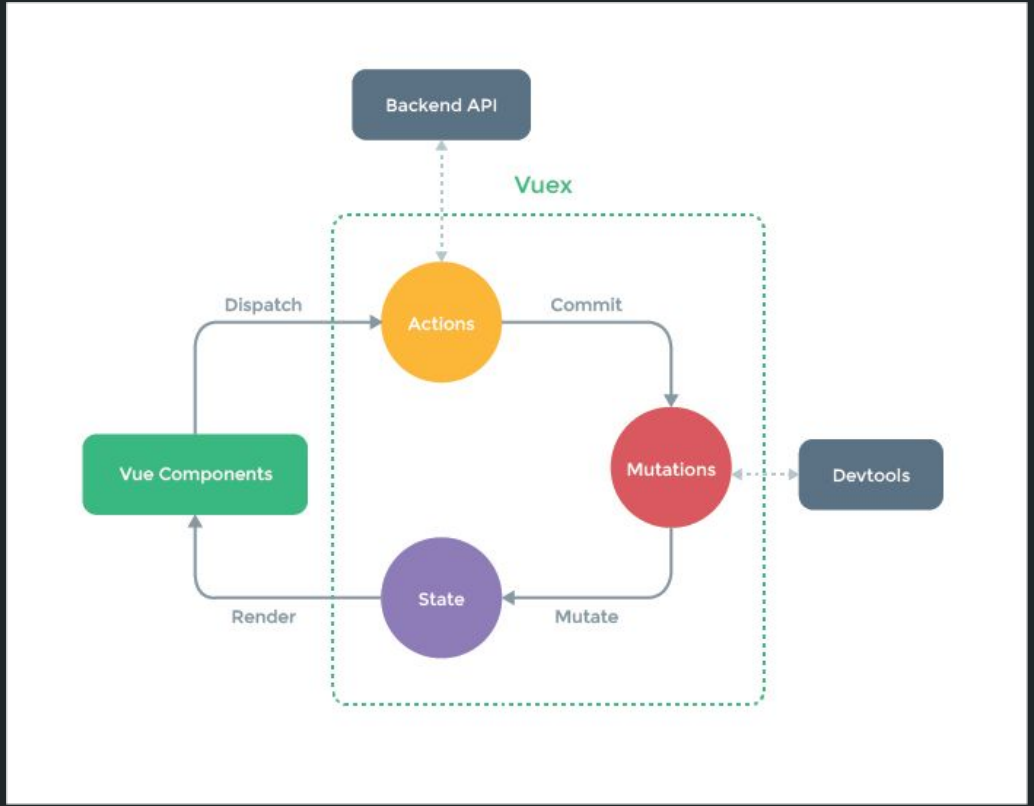
```
// MyForm.vue (Child)
<template>
 <input @keyup.enter="sendMessage">
</template>
<script>
export default {
 methods: {
  sendMessage() {
   this.$emit("send-message",
              "message")
   }
 }
}
</script>
```

```
// Main.vue (Parent)
<template>
 <my-form
  @send-message="appendMessage" />
</template>

<script>
import MyForm from './MyForm.vue'
export default {
 methods: {
  appendMessage(msg) { /* ... */ }
 },
 components: { MyForm }
}
</script>
```

元件狀態管理

Vuex


Backend API
Vuex
Dispatch
Actions
Commit
Vue Components
Mutations
Devtools
State
Render
Mutate

建立 Store

```
export default new Vuex.Store({
 // state 只能透過 mutation 改變
 state: {
   message: ""
 },
 // 外部需要用 getter
 // 來存取 state 中的屬性
 getters: {
   message: state => state.message
 },
 mutations: {
   // mutation 的第一個參數必須是 state
   // 第二個參數為 payload
   setMessage(state, message) {
     state.message = message;
   }
 },
 actions: {
   // action 必須是 async 方法
   // 第一個參數是 store
   // 第二個參數是 payload
   // 需要多個參數時可以組成 object 的形式傳入
   async receiveMessage({ commit }) {
     const res = await axios.get("/message");
     // 用 commit 方法提交一個 mutation
     commit("setMessage", res.data.message);
   }
 }
});
```

mapGetters: 將 state 屬性對應到 computed

```
<template>
 <div class="hello"> <h1>{{ message }}</h1> </div>
</template>

<script>
import { mapGetters } from "vuex";

export default {
 name: "HelloWorld",
 computed: {
   // 把 store 的 message getter 對應成元件的 computed 屬性
   ...mapGetters(["message"])
 }
};
</script>
```

mapActions: 將 action 對應為元件方法

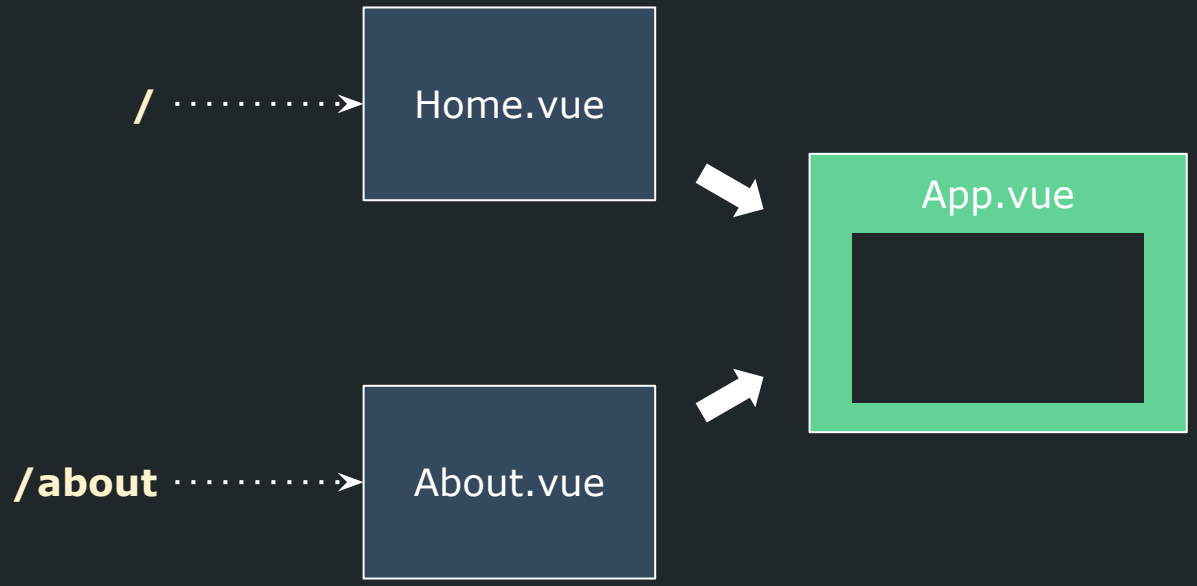
```
<template>
 <div class="home">
   <button @click="receiveMessage">Renew Message</button>
 </div>
</template>

<script>
import { mapActions } from "vuex";

export default {
 methods: {
   // 把 store 的 receiverMessage action 對應成元件的 method
   ...mapActions(["receiveMessage"])
 }
};
</script>
```

SPA 路由

Vue-Router



用 router-view / router-link 建立 Layout 元件

```
// App.vue
<template>
 <div id="app">
  <div id="nav">
   <router-link to="/">Home</router-link> |
   <router-link to="/about">About</router-link>
  </div>
  <router-view></router-view>
 </div>
</template>

<script>
export default { /* ... */ }
</script>
```

用 router-view 建立 Layout 元件

```
// App.vue
<template>
 <div id="app">
  <div id="nav"> <!-- 用 router-link 指向次頁面的路徑 -->
   <router-link to="/">Home</router-link> |
   <router-link to="/about">About</router-link>
  </div>
  <router-view></router-view>
 </div>
</template>

<script>
export default {}
</script>
```

用 router-view 建立 Layout 元件

```
// App.vue
<template>
 <div id="app">
  <div id="nav">
   <router-link to="/">Home</router-link> |
   <router-link to="/about">About</router-link>
  </div>
  <router-view></router-view><!-- 用 router-view 挖放次頁面的洞 -->
 </div>
</template>

<script>
export default {}
</script>
```

建立次頁面

```
// Home.vue
<template>
  <h1>{{ title }}</h1>
</template>
<script>
export default {
  data () {
    return {
      title: 'Home'
    }
  }
}
</script>
```

建立 routes (路徑與次頁面的對應)

```
// router.js
import Home from "@/views/Home.vue";

export default new Router({
 <!-- 啟用 history.pushState() -->
 mode: "history",
 base: process.env.BASE_URL,
 routes: [
  { path: "/", name: "home", component: Home },
  { path: "/about", name: "about",
    <!-- 動態載入子頁面 -->
    component:
    () => import(/* webpackChunkName: "about" */"@/views/About.vue") }
 ]
});
```

總結

我喜歡 Vue.js 的... (個人意見)

- 要記的很少, 能做的很多。
- 貼近 HTML 的 Vue File 寫法, 對設計人員友善。
- 語法糖好吃又直覺。
- 做 Prototype 很快。
- 可以無痛跟第三方 library 整合。
- 有強大的中文文件和外國社群支援。
- Laravel 老爸推薦的。

還有很多沒講到...
但太深的我不會...

參考資源

- 官方中文文件 - Vue.js
- 第一次用 Vue.js 就愛上 [改] - Kuro Hsu
- VueJS 元件之間資料溝通傳遞的方式 - Kuro Hsu

Q & A

別問太難的，我真的不會