Vue技術點之間的聯繫

Vue.js技術棧開發前端PC端項目：

* Vue.js語法
* Vue-router
* Axios
* Webpack
* Element-UI
* Vuex

Vue.js技術棧開發前端移動端項目：

* Vue.js語法
* Vue-router
* Axios
* Webpack
* Vant-UI
* Vuex

Vue基礎

## 1 Vue概述

### （1）創始

尤雨溪是Vue.js的創建者

2014年2月，Vue.js正是發佈

2015年10月27日，正是發佈1.0.0

2016年4月27日，發佈2.0的預覽版本

官網：https://cn.vuejs.org/v2/guide/

Vue很高效：20kb運行大小，超快虛擬DOM。

### （2）Vue是漸進式JavaScript框架

#### ① 框架和庫的區別

框架以提供基礎性的服務為主。比如Vue有雙向數據綁定的支撐，可以減少實際開發時的代碼量。

庫以提供API為主。比如jQuery。

現在框架和庫的界限不是很明顯了。所以很多時候把兩者理解成一個東東。

Vue很靈活，在一個庫和一套完整框架之間自如伸縮。

#### ② 漸進式

聲明式渲染→組件系統→客戶端路由→集中式狀態管理→項目構建

## 2 Vue基本使用

  <body>

    <div id="app">

      <!-- {{}} 是插值表達式，作用是把數據填充到HTML標籤裡 -->

      <div>{{msg}}</div>

      <!-- 差值表達式支持基本的計算操作。比如下面的標籤裡顯示的數字是3。-->

      <div>{{1+2}}</div>

      <!-- 只要符合JS語法規則的表達式都可以放在這裡。 -->

      <div>{{msg + '------' + 123}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      new Vue({

        el: "#app", //元素的掛載位置，值可以是CSS選擇器或者DOM元素。掛載是指把數據關聯到頁面的某個標籤裡。

        // data是模型數據，即開發中遇到的業務相關數據，值是一個對象。

        data: {

          msg: "Hello World by Vue",

        },

      });

    </script>

  </body>

Vue不需要直接操作DOM了。

Vue代碼運行原理分析

Vue.js把Vue代碼編譯為原生JS代碼，以便瀏覽器解析。

## 3 Vue模板語法

### （1）前端渲染

Vue是用來做前端渲染的。

前端渲染即把從數據庫請求回來的數據填充到HTML標籤中。

前端渲染的產物是靜態HTML內容，這內容最終被瀏覽器呈現給用戶。

### （2）前端渲染的方式

#### ① 原生js拼接字符串

缺點：不同開發人員的代碼風格差別很大；隨著業務越來越複雜，後期的維護變得逐漸困難起來。

#### ② 使用前端模板引擎，比如art-template

優點：大家都遵循同樣的規則寫代碼，代碼可讀性明顯提高，方便後期維護。

缺點：沒有專門提供事件機制。

#### ③ 使用vue特有的模板語法

模板語法概覽：

* 插值表達式
* 指令
* 事件綁定
* 屬性綁定
* 樣式綁定
* 分支循環結構

### （3）什麼是指令？

指令的本質就是自定義屬性。

比如下面的abc=123就是一個自定義屬性。標準屬性例子：a標籤中的href屬性。

<div id="app" abc=123>

指令的格式：以v-開始，後面跟個字符串，比如v-cloak。

### （4）v-cloak指令用法

* 差值表達式存在的問題：閃動（如果網絡不好，頁面會先顯示花括號，再把花括號替換為相關的數據內容，這個過程就叫做閃動，給用戶造成不好的體驗。）
* 如何解決該問題？使用v-cloak指令

    <style type="text/css">

      [v-clock] {

        display: none;

      }

    </style>

      <div v-cloak>{{msg}}</div>

* 解決該問題的原理：先通過樣式隱藏內容，然後在內存中進行值的替換，替換好值之後再顯示最終的結果。

### （5）數據綁定指令v-text，v-html，v-pre

1. v-text 填充純文本
   * 沒有閃動問題
   * 比插值表達式更加簡潔
   * 推薦用v-text而非插值表達式
2. v-html 填充HTML片段
   * 存在安全問題
   * 本網站內部數據可以使用，來自第三方的數據不可以用
3. v-pre 填充原始信息
   * 顯示原始信息，跳過編譯過程（分析編譯過程）

範例代碼：

  <body>

    <div id="app">

      <div v-text="msg"></div>

      <div v-html="msg1"></div>

      <div v-pre>{{msg}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      new Vue({

        el: "#app",

        data: {

          msg: "Hello World by Vue",

          msg1: "<h1>HTML標籤h1</h1>",

        },

      });

    </script>

  </body>

顯示的樣子：



### （6）數據響應式和v-once

響應式類別：

* HTML5中的響應式（屏幕尺寸的變化導致樣式的變化）
* 數據的響應式（數據的變化導致頁面內容的變化）

數據綁定：把數據填充到標籤中（v-text，v-html，v-pre的功能，它們綁定的數據默認都是響應式的）

v-once只編譯一次，顯示內容之後不再具有響應式功能，可以提高性能。

### （7）雙向綁定數據v-model

在input中添加v-model功能，可以修改data中msg的值，進而影響div中的顯示內容

  <body>

    <div id="app">

      <div>{{msg}}</div>

      <input type="text" v-model="msg" />

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      new Vue({

        el: "#app",

        data: {

          msg: "Hello World by Vue",

        },

      });

    </script>

  </body>

### （8）與雙向數據綁定相關的MVVM設計思想

* M（model）
  + Plain JavaScript Objects，比如上面的例子中data中的數據
* V（view）
  + 本質上是DOM元素
* VM（view-model）
  + 實現控制邏輯，把上面兩者結合在一起。



### （9）事件綁定

v-on指令用法

<input type="button" v-on:click="num++" />

v-on指令簡寫

<input type="button" @click="num++" />

事件函數的調用方式：

* 直接綁定函數名稱

<button v-on:click="say">Hello</button>

* 調用函數（要傳遞參數的話，只能用這種方式）

<button v-on:click="say()">Hi</button>

函數定義在methods屬性中

示例代碼

  <body>

    <div id="app">

      <div>{{num}}</div>

      <button v-on:click="say">Hello</button>

      <button v-on:click="say()">Hi</button>

      <button v-on:click="inc">點擊+1</button>

      <button v-on:click="inc()">點擊+1</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          msg: "Hello World by Vue",

          num: 1,

        },

        methods: {

          say: function () {

            console.log("LOL");

          },

          inc: function () {

            // this === vm。在這裡，this指的是Vue的實例對象vm

            this.num++;

          },

        },

      });

    </script>

  </body>

### （10）事件函數參數傳遞

<button v-on:click="say('hi', $event)">Hi</button>

如果沒用到$event，傳參數的時候可以不寫$event。

下面這種直接綁定的方式默認了會傳遞$event參數

<button v-on:click="say">Hello</button>

示例代碼

  <body>

    <div id="app">

      <!-- 傳遞1個參數 -->

      <button v-on:click="say1('LOL1')">Hi 1個參數</button>

      <!-- 傳遞多個參數 -->

      <button v-on:click="say2('LOL2','lala2', $event)">Hi 多個參數</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          msg: "Hello World by Vue",

          num: 1,

        },

        methods: {

          say1: function (*p1*) {

            console.log(p1);

          },

          say2: function (*p1*, *p2*, *event*) {

            console.log(p1, p2, event);

          },

        },

      });

    </script>

  </body>

### （11）事件修飾符

.stop阻止冒泡

<a v-on:click.stop="handle">跳轉</a>

.prevent阻止默認行為

<a href="http://www.bing.com" v-on:click.prevent="handle">跳轉</a>

事件修飾符可以連著寫。寫的先後順序不同，有時會有不同的效果。

既阻止冒泡又阻止默認行為

<a href="http://www.bing.com" v-on:click.stop.prevent="handle">跳轉</a>

### （12）按鍵修飾符

.enter回車鍵

<input type="text" v-on:keyup.enter="handle" />

.delete刪除鍵

<input type="text" v-on:keyup.delete="handle" />

### （13）自定義按鍵修飾符

全局config.keyCodes對象

比如，按下a，會觸發handle函數

<input type="text" v-on:keyup.65="handle" />

### （14）屬性綁定v-bind

<a v-bind:href="url">跳轉</a>

縮寫形式

<a :href="url">跳轉</a>

url是寫在data裡面的數據

  <body>

    <div id="app">

      <a :href="url">跳轉</a>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          url: "http://www.bing.com",

        },

        methods: {

          handle: function () {

            console.log("handle");

          },

        },

      });

    </script>

  </body>

### （15）v-model底層實現原理

  <body>

    <div id="app">

      <!-- 下面兩種寫法都可以 -->

      <input type="text" v-on:input="handle" />

      <input type="text" v-on:input="msg=$event.target.value" />

      <div>{{msg}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          msg: "天朗氣清",

        },

        methods: {

          handle: function (*event*) {

            this.msg = event.target.value;

          },

        },

      });

    </script>

  </body>

### （16）樣式綁定

#### ① 對象語法

<div v-bind:class="{active:isActive}"></div>

示例代碼

    <style type="text/css">

      .active {

        width: 100px;

        height: 50px;

        border: 5px solid skyblue;

      }

      .error {

        background-color: yellow;

      }

    </style>

  <body>

    <div id="app">

      <div v-bind:class="{active:isActive1}">1</div>

      <div v-bind:class="{active:isActive, error:isError}">2</div>

      <button v-on:click="handle1">操縱邊框1</button>

      <button v-on:click="handle2">操縱邊框2</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          isActive1: true,

          isActive: true,

          isError: true,

        },

        methods: {

          handle1: function () {

            this.isActive1 = !this.isActive1;

          },

          handle2: function () {

            this.isActive = !this.isActive;

            this.isError = !this.isError;

          },

        },

      });

    </script>

  </body>

#### ② 數組語法

<div v-bind:class="[activeClass, errorClass]"></div>

示例代碼

    <style type="text/css">

      .active {

        width: 100px;

        height: 50px;

        border: 5px solid skyblue;

      }

      .error {

        background-color: yellow;

      }

    </style>

  <body>

    <div id="app">

      <div v-bind:class="[activeClass, errorClass]"></div>

      <button v-on:click="handle">操作背景色</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          activeClass: "active",

          errorClass: "error",

        },

        methods: {

          handle: function () {

            this.errorClass = "";

          },

        },

      });

    </script>

  </body>

#### ③ 對象綁定和數組綁定可以結合使用

    <style type="text/css">

      .active {

        width: 100px;

        height: 50px;

        border: 5px solid skyblue;

      }

      .error {

        background-color: yellow;

      }

      .test {

        color: salmon;

      }

    </style>

  <body>

    <div id="app">

      <div v-bind:class="[activeClass, errorClass,{test: isTest}]">test</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          activeClass: "active",

          errorClass: "error",

          isTest: true,

        },

        methods: {},

      });

    </script>

  </body>

#### ④ class綁定的值可以簡化操作

    <style type="text/css">

      .active {

        width: 100px;

        height: 50px;

        border: 5px solid skyblue;

      }

      .error {

        background-color: yellow;

      }

      .test {

        color: salmon;

      }

    </style>

  <body>

    <div id="app">

      <div v-bind:class="arrClasses">test</div>

      <div v-bind:class="objClasses">test</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          arrClasses: ["active", "error", "test"],

          objClasses: {

            active: true,

            error: true,

            test: true,

          },

        },

        methods: {},

      });

    </script>

  </body>

#### ⑤ 處理默認的class

原有的class不會被覆蓋。在原有的class基礎上追加class。

    <style type="text/css">

      .active {

        width: 100px;

        height: 50px;

        border: 5px solid skyblue;

      }

      .error {

        background-color: yellow;

      }

      .test {

        color: salmon;

      }

      .base {

        font-size: 28px;

      }

    </style>

  <body>

    <div id="app">

      <div class="base" v-bind:class="arrClasses">test</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          arrClasses: ["active", "error", "test"],

        },

        methods: {},

      });

    </script>

  </body>

#### ⑥ style樣式處理

方式1

  <body>

    <div id="app">

      <div

        v-bind:style="{border: borderStyle, width:widthStyle, height:heightStyle}"

      >

        test

      </div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          borderStyle: "1px solid skyblue",

          widthStyle: "200px",

          heightStyle: "100px",

        },

        methods: {},

      });

    </script>

  </body>

方式2

  <body>

    <div id="app">

      <div v-bind:style="objStyles">test</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          objStyles: {

            border: "1px solid skyblue",

            width: "200px",

            height: "100px",

          },

        },

        methods: {},

      });

    </script>

  </body>

方式3，用數組的方式把樣式對象羅列上，前面有的樣式會被覆蓋，沒有的會加上

  <body>

    <div id="app">

      <div v-bind:style="[objStyles,overrideStyles]">test</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          objStyles: {

            border: "1px solid skyblue",

            width: "200px",

            height: "100px",

          },

          overrideStyles: {

            border: "5px solid #355c7d",

            backgroundColor: "pink",

          },

        },

        methods: {},

      });

    </script>

  </body>

### （17）分支結構

分支結構：

* v-if
* v-else
* v-else-if
* v-show

v-if不會渲染標籤；v-show會把標籤渲染出來，如果不需要顯示，則把標籤設置為display:none，如果需要顯示，則不給標籤設置display:none。

### （18）循環結構

v-for遍歷數組

<li v-for=’item in list’>{{item}}</li>

<li v-for=’(item, index) in list’>{{item}} + ‘---’ + {{index}}</li>

（其中item和index都是自定義的，愛叫啥叫啥）

key的作用：幫助vue區分不同的元素，從而提高性能

<li :key=’item.id’ v-for=’(item, index) in list’>{{item}} + ‘---’ + {{index}}</li>

<li :key=’index’ v-for=’(item, index) in list’>{{item}} + ‘---’ + {{index}}</li>

v-for遍歷對象

<div v-for=’(value, key, index) in object’></div>

v-if和v-for結合使用遍歷對象：滿足了某個條件才渲染標籤。同理，v-if可以和上述的v-for遍歷數組結合使用。

<div v-if=’value===12’ v-for=’(value, key, index) in object’></div>

（其中value、key、index都是自定義的，愛叫啥叫啥）

## 4 基礎案例

4-1 Vue選項卡案例

聲明式編程：模板的結構和最終顯示的效果基本一致。

## 5 Vue常用特性

### （1）表單操作

* input單行文本
* radio單選框
* checkbox多選框
* textarea多行文本
* select下拉多選

用:prevent阻止表單的默認提交操作

<input type="submit" value="提交" @click:prevent="handle" />

複選框的寫法示例代碼：

  <body>

    <div id="app">

      <span>爱好：</span>

      <input type="checkbox" id="ball" value="1" v-model="hobbies" />

      <label for="ball">篮球</label>

      <input type="checkbox" id="sing" value="2" v-model="hobbies" />

      <label for="sing">唱歌</label>

      <input type="checkbox" id="code" value="3" v-model="hobbies" />

      <label for="code">写代码</label>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          hobbies: ["2", "3"],

        },

        methods: {},

      });

    </script>

  </body>

下拉框-單選 示例代碼

  <body>

    <div id="app">

      <span>职业：</span>

      <select v-model="occupation">

        <option value="0">请选择职业...</option>

        <option value="1">教师</option>

        <option value="2">软件工程师</option>

        <option value="3">律师</option>

      </select>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          occupation: 2,

        },

        methods: {},

      });

    </script>

  </body>

下拉框-多選 示例代碼

  <body>

    <div id="app">

      <span>职业：</span>

      <select v-model="occupation" multiple="true">（也可以把=“true”省略）

        <option value="0">请选择职业...</option>

        <option value="1">教师</option>

        <option value="2">软件工程师</option>

        <option value="3">律师</option>

      </select>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          occupation: ["2", "3"],

        },

        methods: {},

      });

    </script>

  </body>



手動多選是shift+鼠標左鍵單擊

### （2）表單域修飾符

* number: 轉化為數值
* trim: 去掉開始和結尾的空格
* lazy: 將input事件切換為change事件

input事件是每次輸入內容時，都會被觸發；change事件是失去焦點時才被觸發。

示例代碼

  <body>

    <div id="app">

      <input type="number" v-model.number="age" />

      <input type="text" v-model.trim="info" />

      <input type="text" v-model.lazy="msg" />

      <div>{{msg}}</div>

      <button @click="handle">点击</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          age: "",

          info: "",

          msg: "",

        },

        methods: {

          handle: function () {

            console.log(this.info.length);

          },

        },

      });

    </script>

  </body>

### （3）自定義指令

#### ① 不帶參數

在內置指令不滿足需求的情況下，自定義指令應運而生。

自定義指令的語法規則，比如獲取元素焦點：

在script裡定義指令：

      Vue.directive("focus", {

        inserted: function (*el*) {

          //獲取元素的焦點

          // el表示指令綁定的元素

          el.focus();

        },

      });

在標籤裡使用指令：

      <input type="text" v-focus />

#### ② 攜帶參數

inserted和bind大多數情況下通用，只不過觸發時機不同。兩者使用的參數也是一樣的。

inserted：被綁定元素插入父節點時調用（僅保證父節點存在，但不一定已被插入文檔中）。

bind：只調用一次，指令第一次綁定到元素時調用。可以進行一次性的初始化設置。

在script裡定義

      Vue.directive("color", {

        bind: function (*el*, *binding*) {

          console.log(binding.value.color);

          el.style.backgroundColor = binding.value.color;

        },

      });

在標籤裡使用

<input type="text" v-color='{color:"skyblue"}' />

或者這樣：

  <body>

    <div id="app">

      <input type="text" v-color="msg" />

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      Vue.directive("color", {

        bind: function (*el*, *binding*) {

          el.style.backgroundColor = binding.value.color;

        },

      });

      var vm = new Vue({

        el: "#app",

        data: {

          msg: {

            color: "skyblue",

          },

        },

        methods: {},

      });

    </script>

  </body>

#### ③ 局部指令

局部指令只能在本組件中使用，而全局指令沒有這個限制。

  <body>

    <div id="app">

      <input type="text" v-color="msg" />

      <input type="text" v-focus />

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          msg: {

            color: "skyblue",

          },

        },

        methods: {},

        directives: {

          color: {

            bind: function (*el*, *binding*) {

              el.style.backgroundColor = binding.value.color;

            },

          },

          focus: {

            inserted: function (*el*) {

              el.focus();

            },

          },

        },

      });

    </script>

  </body>

### （4）計算屬性

為何需要計算屬性？

表達式的計算邏輯可能會比較複雜，使用計算屬性可以使模板內容更加簡潔。

* 寫函數的時候記得return
* 使用的時候直接寫函數的名字（比如下面的reverseString），不用帶括號

  <body>

    <div id="app">

      <div>{{msg}}</div>

      <div>{{reverseString}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          msg: "Hello",

        },

        methods: {},

        computed: {

          reverseString: function () {

            return this.msg.split("").reverse().join("");

          },

        },

      });

    </script>

  </body>

### （5）計算屬性和方法的區別

* 計算屬性是基於它們的依賴（即data中的數據）進行緩存的。（只要data中相關的數據不變化，計算屬性只會計算一次，然後存在緩存中，接下來每次調用都用緩存中的結果。）
* 方法（methods）不存在緩存一說。（每次都會重新調用方法）
* 調用計算屬性時不需要加括號。
* 調用方法需要加括號。

### （6）偵聽器

#### ① 偵聽器基礎

偵聽器用於偵聽data。一旦data變化，偵聽器通知其綁定的方法。

偵聽器應用場景：數據變化時執行異步或開銷較大的操作。

示例：

修改input裡面的姓或者名，下面div裡顯示的全名跟著變化。

  <body>

    <div id="app">

      <div>

        <span>姓：</span>

        <span>

          <input type="text" v-model="familyName" />

        </span>

      </div>

      <div>

        <span>名：</span>

        <span>

          <input type="text" v-model="givenName" />

        </span>

      </div>

      <div>{{fullName}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          familyName: "Zhu",

          givenName: "Yan",

          fullName: "Zhu Yan",

        },

        watch: {

          familyName: function (*val*) {

            this.fullName = val + " " + this.givenName;

          },

          givenName: function (*val*) {

            this.fullName = this.familyName + " " + val;

          },

        },

      });

    </script>

  </body>

#### ② 偵聽器案例：檢測用戶名是否可用

需求：輸入框中輸入姓名，失去焦點時驗證是否存在。如果存在，提示重新輸入；如果不存在，提示可以使用。

步驟：

1. 通過v-model實現數據綁定
2. 需要提供提示信息
3. 需要偵聽器監聽輸入信息的變化
4. 需要修改觸發的事件

示例代碼

  <body>

    <div id="app">

      <div>

        <span>用戶名：</span>

        <span>

          <input type="text" v-model.lazy="uname" />

        </span>

        <span>{{tip}}</span>

      </div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          uname: "",

          tip: "",

        },

        methods: {

          checkName: function (*uname*) {

            // 調用接口，可以使用定時任務的方式模擬接口調用

            var that = this;

            setTimeout(function () {

              // 模擬接口調用

              if (uname == "admin") {

                that.tip = "用戶名已存在，請更換一個";

              } else {

                that.tip = "用戶名可以使用";

              }

            }, 1000);

          },

        },

        watch: {

          /\*

          偵聽器

          1 採用偵聽器監聽用戶名的變化

          2 調用後台接口進行驗證

          3 根據驗證的結果調整提示信息

          \*/

          uname: function (*val*) {

            // 修改提示信息

            this.tip = "正在驗證...";

            // 調用後台接口驗證用戶名是否合規

            this.checkName(val);

          },

        },

      });

    </script>

  </body>

### （7）過濾器

#### ① 過濾器基本用法

格式化數據，比如將字符串格式化為首字母大寫，將日期格式化為指定的格式等。

自定義過濾器

Vue.filter(‘過濾器名稱’, function(value){

// 過濾器業務邏輯

})

使用過濾器

<div>{{msg | upper }}</div>

<div>{{msg | upper | lower }} </div> （可以同時使用多個過濾器，把前面的過濾器返回的結果作為下一個過濾器接收的參數）

<div v-bind:id= “id | formatId”></div>

支持局部過濾器

filters: {

capitalize: function(){}

}

全局過濾器在所有地方都可以使用，局部過濾器只在組件內部使用。

  <body>

    <div id="app">

      <input type="text" v-model="msg" />

      <div>{{msg | upper}}</div>

      <div :abc="msg | upper">testing</div>

      <div>{{msg | lower}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      // 過濾器

      Vue.filter("upper", function (*val*) {

        return val.charAt(0).toUpperCase() + val.slice(1);

      });

      var vm = new Vue({

        el: "#app",

        data: {

          msg: "",

        },

        filters: {

          lower: function (*val*) {

            return val.charAt(0).toLowerCase() + val.slice(1);

          },

        },

      });

    </script>

  </body>

#### ② 帶參數的過濾器案例

定義過濾器

Vue.filter(‘format’, function(value, arg1){

// value是下面的date，arg1是下面的’yyyy-mm-dd’

}

使用過濾器

<div>{{date | format(‘yyyy-mm-dd’)}}</div>

案例：使用過濾器格式化日期

  <body>

    <div id="app">

      <div>{{date | format('yyyy-mm-dd')}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      // 過濾器

      Vue.filter("format", function (*value*, *arg*) {

        if (arg === "yyyy-mm-dd") {

          var result = "";

          var mm = value.getMonth() + 1;

          mm = mm < 10 ? "0" + mm : mm;

          var dd = value.getDate();

          dd = dd < 10 ? "0" + dd : dd;

          return (result = value.getFullYear() + "-" + mm + "-" + dd);

        }

      });

      var vm = new Vue({

        el: "#app",

        data: {

          date: new Date(),

        },

      });

    </script>

  </body>

### （8）生命週期





上面的生命週期函數又叫“鉤子”

重點關注mounted。這個函數一旦被觸發，表示頁面完成了初始化，頁面的模板內容存在，可以向模板裡填充數據了。

  <body>

    <div id="app">

      <div>{{msg}}</div>

      <button @click="update">更新</button>

      <button @click="destroy">銷毀</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          msg: "生命週期",

        },

        methods: {

          update: function () {

            this.msg = "hello";

          },

          destroy: function () {

            this.$destroy();

          },

        },

        beforeCreate: function () {

          console.log("beforeCreate");

        },

        created: function () {

          console.log("created");

        },

        beforeMount: function () {

          console.log("beforeMount");

        },

        mounted: function () {

          console.log("mounted");

        },

        beforeUpdate: function () {

          console.log("beforeUpdate");

        },

        updated: function () {

          console.log("updated");

        },

        beforeDestroy: function () {

          console.log("beforeDestroy");

        },

        destroyed: function () {

          console.log("destroyed");

        },

      });

    </script>

  </body>

## 6 綜合案例

### （1）數組相關API

#### ① 變異方法和替換數組

變異方法：數組的原生方法不是響應式的，Vue把原生的數組API處理成了響應式，這樣只要我們操作數組中的數據，頁面中的模板內容會隨之變化。

變異方法會修改原有數據

* push()
* pop()
* shift()
* unshift()
* splice()
* sort()
* reverse()

替換數組：不會影響原始數組，總是返回一個新數組。

* filter()
* concat()
* slice()

示例代碼

  <body>

    <div id="app">

      <div>

        <span>

          <input type="text" v-model="fname" />

          <button @click="add">添加</button>

          <button @click="del">刪除</button>

          <button @click="change">替換</button>

        </span>

      </div>

      <ul>

        <li :key="index" v-for="(item,index) in list">{{item}}</li>

      </ul>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          fname: "",

          list: ["apple", "orange", "banana"],

        },

        methods: {

          add: function () {

            this.list.push(this.fname);

          },

          del: function () {

            this.list.pop();

          },

          change: function () {

            this.list = this.list.slice(0, 2); //從0開始，截取2個

          },

        },

      });

    </script>

  </body>

#### ② 修改響應式數據

如下2個API既可以處理數組，也可以處理對象，都是響應式的。兩者形式不同，作用一樣。

Vue.set(vm.items, indxOfItem, newValue)

vm.$set(vm.items, indxOfItem, newValue)

參數1表示要處理的數字名稱

參數2表示要處理的數組的索引

參數3表示要處理的數組的值

  <body>

    <div id="app">

      <div>{{info.gender}}</div>

      <ul>

        <li :key="index" v-for="(item,index) in list">{{item}}</li>

      </ul>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

    <script type="text/javascript">

      var vm = new Vue({

        el: "#app",

        data: {

          fname: "",

          list: ["apple", "orange", "banana"],

          info: {

            name: "list",

            age: 12,

          },

        },

      });

      // vm.list[1]='lemon';//用索引修改了的數據不是響應式的，這樣修的話，雖然data有變化，但是頁面不會有變化。

      Vue.set(vm.list, 2, "lemon"); //這種方法可取，還可以用下面的方法

      // vm.$set(vm.list, 1, "lemon");

      vm.$set(vm.info, "gender", "female");

    </script>

  </body>

### （2）項目目標

① 實現靜態列表效果

② 基於數據實現模板效果

③ 處理每行的操作按鈕

### （3）添加圖書

實現表單的靜態效果

添加圖書表單域數據綁定

添加按鈕事件綁定

實現添加業務邏輯