**基礎寫法:**

      <script>

        //vue寫法

        var app =new Vue({

            el : '#app', //實例創建後負責的區域，可用於tag、class、id等js的Element，

區域內部的所有DOM都能使用

            data : {     //此實例包含的變數，被掛載的區域都能使用

                firstName : 'lucas',

lastName: 'dong'

            },

watch: { //監聽firstName，若firstName 變化則執行function

                firstName : function(){

                    this.change = "1"

                }

            },

methods: { //此實例包含的函式，被掛載的區域都能使用

                handleBtnClick : function() {

                   this.inputValue = ''; //在實例內可用this取得實例的變數

                }

            },

computed : { //計算屬性，用於合併數據，且使用後會被放進緩存，除非依賴物件被修改

                fullName : function() { //才會再次進來function執行

                    return this.firstName + " " + this.lastName;

                },

fullName2 : { //也可以有2種形式，直接使用會拿取get，若付值給他會拿set

                    get : function() {

                        return this.firstName + " " + this.lastName;

                    },

                    set : function(value) {

                        var arr = value.split(" ");

                        this.firstName = arr[0];

                        this.lastName = arr[1];

                    }

                }

            }

        })

        //傳統JS寫法

        var dom = document.getElementById('app');

        dom.innerHTML = 'hello world';

    </script>

**取得VUE內部數據:**

setTimeout(function(){

            app.$data.content = 'bye world' //取得vue實例的內容，必須在前面加上$

        }, 2000)

**VUE模板語法:**

 <div id="app">

        <input type="text" v-model = "inputValue"/> <!--將input的數值綁定到Vue實例的變數中，此為雙向綁定input或vue實力的變數更改，兩者都會改變，也可以綁不同的input，後面回詳細說明-->

        <button v-on:click = 'handleBtnClick'>提交</button> <!--將button的click事件綁到Vue的handBtnClick函數-->

        <ul>

            <li v-for = "itme in list">{{itme}}</li><!--將實例內的list變數以item名稱

        </ul> foreach-->

    </div>

<todo-item v-bind:content = "item" v-for = "item in list"></todo-item>

<!—v-bind用於將變數綁定傳給子組件 -->

<div v-text = "name + ' lee' "></div><!--將vue實例裡面的name直接插入innerTEXT-->

<div v-html = "name + ' lee' "></div><!--將vue實例裡面的name直接插入innerTEXT-->

v-text不會解析HTML tag，v-html會解析HTML tag，也可以於後面直接加東西

<div id="app">

       <div v-if = "show">{{message}}</div>

<div v-show = "show">{{message}}</div>

    </div>

    <script>

        var vm = new Vue({

            el : "#app",

            data : {

                show : true,

                message : "hello"

            }

        });

  </script>

 //v-if會依照show的boolean來決定是否顯示此DIV，也可以用判斷是EX:a===b，可以搭配v-else或v-else-if使用

 //v-show會依照show的boolean來決定是否顯示此DIV

v-if如果為false 此div是完全不存在，但是v-show如果為false 此div是存在的，但display:none

<div id="app">

       <div v-for="(item, key, index) in list"> //將list 以item 名foreach出來，第二個參數   
 {{index}}-{{key}}-{{item}} //key為鍵值，index為數據在第幾個

        </div>

    </div>

<script>

        var vm = new Vue({

            el : "#app",

            data : {

              list : {

                 name : "lucas",

                 age : 28,

                 gender : "male"

               }

            }

        });

    </script>

**VUE全局組件創建方法(可以直接用)**

Vue.component("TodoItem", {     //命名時單字以大駝峰表示，使用時必須小寫且用 - 連結

            props : ['content'], //接收外部由bind傳入的值

            template : "<li>{{this.content}}</li>"

        })

 <todo-item v-bind:content = "item" v-for = "item in list"></todo-item> <!--使用時必須小寫且用 - 連結-->

**VUE局部組件創建方法(必須引入實例中才能用)**

  var TodoItem = {

            props : ['content'], //接收外部由bind傳入的值

            template : "<li>{{content}}</li>"

        }

 var app =new Vue({

            el : '#app',

            components : {TodoItemConponents:TodoItem}, //局部組件必須寫進實例才可使用，

            data : {    //前值為組件寫入後的名稱，後值為要寫入的組件名稱

                inputValue : ''

            },

            methods: {

                handleBtnClick : function() {

                 ．．．

                }

            }

        })

**生命週期函數(不需放在method裡)**

  var vm = new Vue({

            el : "#app",

            data : {

               message : 'Vue實例生命週期測試'

            },

            beforeCreate : function() { //在實例創建時，內部初始化後外部初始化前執行

                console.log('beforeCreate');

            },

            created : function(){ //在實例創建時，內部初始化及外部初始化後執行

                console.log('created');

            },

            beforeMount : function() {

                console.log('beforeMount');//在實例準備渲染頁面前執行

            },

            mounted : function() { //在頁面渲染後執行

                console.log('mounted');

            },

            beforeDestroy : function() { //在實例刪除前執行

                console.log('beforeDestroy');

            },

            destroyed : function() { //在實例刪除後執行

                console.log('destroyed');

            },

            beforeUpdate : function() { //在實例更新數據前執行

                console.log('beforeUpdate');

            },

            updated : function() { //在實例更新數據後執行

                console.log('updated');

            },

            errorCaptured : function() { //在子組件捕獲錯誤訊息時調用

                console.log('errorCaptured');

            },

            activated(){

                console.log('activated'); //被keep-alive缓存的组件被調用時執行

            },

            deactivated(){

                console.log('deactivated'); //被keep-alive缓存的组件停止調用時執行

            }

        });

**Class動態修改方法**

**Boolean修改法**

<div v-on:click = "handleDivClick"

     v-bind:class = "{activated : isActivated}" >

* // activated 會依isActivated的布林值顯示，isActivated為true 則class=”activated”，
* 反之，class=””

**陣列變數修改法**

<div v-on:click = "handleDivClick"

     v-bind:class = "[activated]" >

//直接讀取data內的變數內容

 var vm = new Vue({

            el : "#app",

            data : {

              activated : "",

            },

            methods : {

                handleDivClick : function () {

                    this.activated = this.activated === "activated" ? "" : "activated"

                }

            }

        });

**style動態修改方法**

 <div v-on:click = "handleDivClick"

             v-bind:style = "[styleObj]">

var vm = new Vue({

            el : "#app",

            data : {

                styleObj : {

                    color : ""

                },

            },

            methods : {

                handleDivClick : function () {

                    this.styleObj.color = this.styleObj.color === "red" ? "" : "red";

                }

            }

        });

**Key值應用**

 <div id="app">

       <div v-if = "show">用戶名: <input key="username"/></div>

       <div v-else = "show">EMAIL: <input key="password"/></div>

    </div>

若沒有KEY值if else轉換時，因為虛擬DOM的機制，系統會嘗試複用前狀態的input，會導致舊狀態的input值跑到新狀態的input，所以盡量要在每個DOM上加上唯一的key值

 <div v-for="(item, index) in list" v-bind:key="index">

       {{index}}-{{item}}

  </div>

為了提升循環顯示的性能，我們會再v-for中加上唯一的key值，但不建議用index，因為在頻繁處理這些v-for出來的DOM時，會導致一些性能問題或資料順序問題，建議在後端撈資料時，一併將id傳至前端當key值

**佔位符號**

 <template v-for="(item, index) in list"> //不會顯示於渲染頁面中

          <span>{{index}}-{{item}}</span>

  </template>

**補充:**

Vue只能使用Vue提供的改變data方法或將data整個等於另一個新的數據，否則頁面不會自動依改變渲染，如:push、pop、shift、unshift、splice、sort、reverse

vm.$data.list.push(‘test’) 、 vm.$data.list=[‘test’] 、vm.$set(vm.list, “address” ,” somewhere”)

Vue.set(vm.list, 1 , 5)

**事件綁定寫法比較**

 <button v-on:click = "handleClick">button</button> //一般寫法

  var vm = new Vue({

            el : "#app",

            methods : {

                handleClick : function(e) {

                    console.log(e);

                }

            }

        });

 <button v-on:click = "handleClick($event, 1, 2, 3)">button</button> //傳值寫法

 var vm = new Vue({

            el : "#app",

            methods : {

                handleClick : function(e, one , two, three) {

                    console.log(e);

                }

            }

        });

**事件修飾符(阻止預設及冒泡)**

<form action="/abc" v-on:click.prevent><!--prevent會阻擋預設行為，此處是阻擋action-->

    <input type="submit">

</form>

<form action="/abc" v-on:click.prevent = "handleClick"><!—也可以直接在後面加上function-->

    <input type="submit">

</form>

<div  v-on:click.self = "handleClick"> <!--self會阻擋子層的冒泡行為，所以點hello將不動作-->

       nice

      <div>hello</div>

</div>

<div  v-on:click.capture = "handleClick"> <!--capture會將多事件的冒泡執行順序改變，原本點擊in -->

      out <!—會由內執行至外，而capture會變成由外執行至內 -->

     <div v-on:click.capture = "handleClickInner">in</div>

</div>

**補充:**

冒泡行為:

1. 父層包子層，當父層判斷onClick事件時，子層也包含在內，所以點擊子層也算在父層內，導致父層函數啟動。
2. 父層包子層，當父子層都有onClick事件時，點擊子層會因為包在父層內，先執行子層事件後，會跑去執行父層事件。

**按鍵修飾符**

<input v-on:keydown.enter = "handleKeyDown"/> <!--限制必須按下enter後，才執行函數-->

<input v-on:keydown.ctrl = "handleKeyDown"/> <!--若要觸發此函數，必須按下Ctrl加任意鍵-->

ctrl可以改成alt、shift、meta，功能是一樣的

**游標修飾符**

<input @click.right = "handleClick"/> <!--必須是右鍵點擊才能執行-->

<input @click.middle = "handleClick"/> <!--必須是中鍵點擊才能執行-->

<input @click.left= "handleClick"/> <!--必須是左鍵點擊才能執行-->

**v-model補充(含修飾符)**

<input type="text" v-model = "value"/> <!--基本用法，會抓取input內容更新-->

<input type = "checkbox" v-model = "value"/> <!—會以true或false去更新-->

<input type = "radio" value = "123" v-model = "value"/> <!—會以value值去更新-->

 <select v-model = "value"> <!—會以選中的內容去更新，若有value會優先抓取-->

       <option value="1">A</option>

       <option>B</option>

       <option>C</option>

</select>

<input type="text" v-model.lazy = "value"> <!--失焦後才會更新資料-->

<input type="text" v-model.number = "value">

<!—因為text會統一type為字串，使用number後，會判斷是否為數字，是的話以數字型態更新

，但如果判定為數字後，將會限制輸入數字以外的字串-->

<input type="text" v-model.trim = "value"> <!—會去除字串前後的空格後，才更新資料-->