Vue基础篇

(陈华旺-chenhuawang@itany.com)

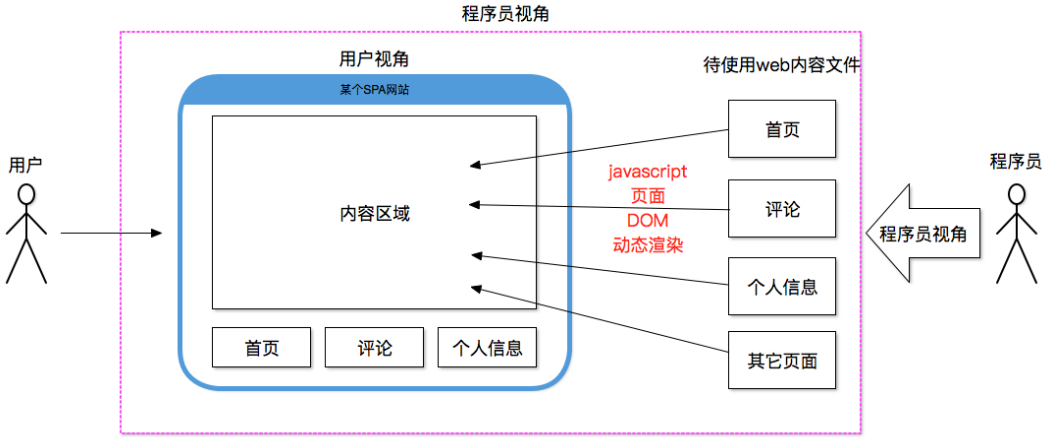
目录 [Vue基础篇]

> 库：对功能和布局特性进行封装，提供易用方法的JS，CSS，HTML 等文件
> 框架：针对设计思想实现的项目开发结构和环境定义语法（特定的运行环境，独立使用语法）

1、设计模式

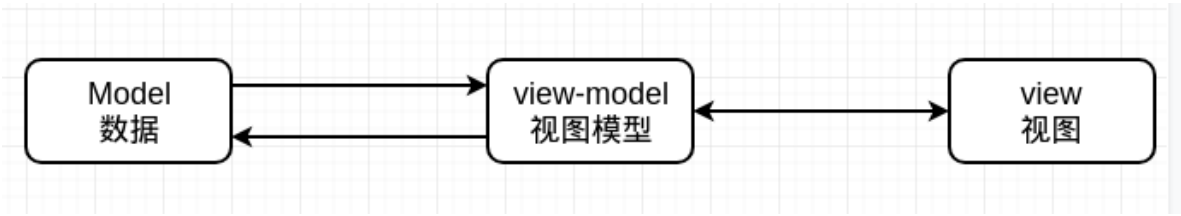
1.1、SPA

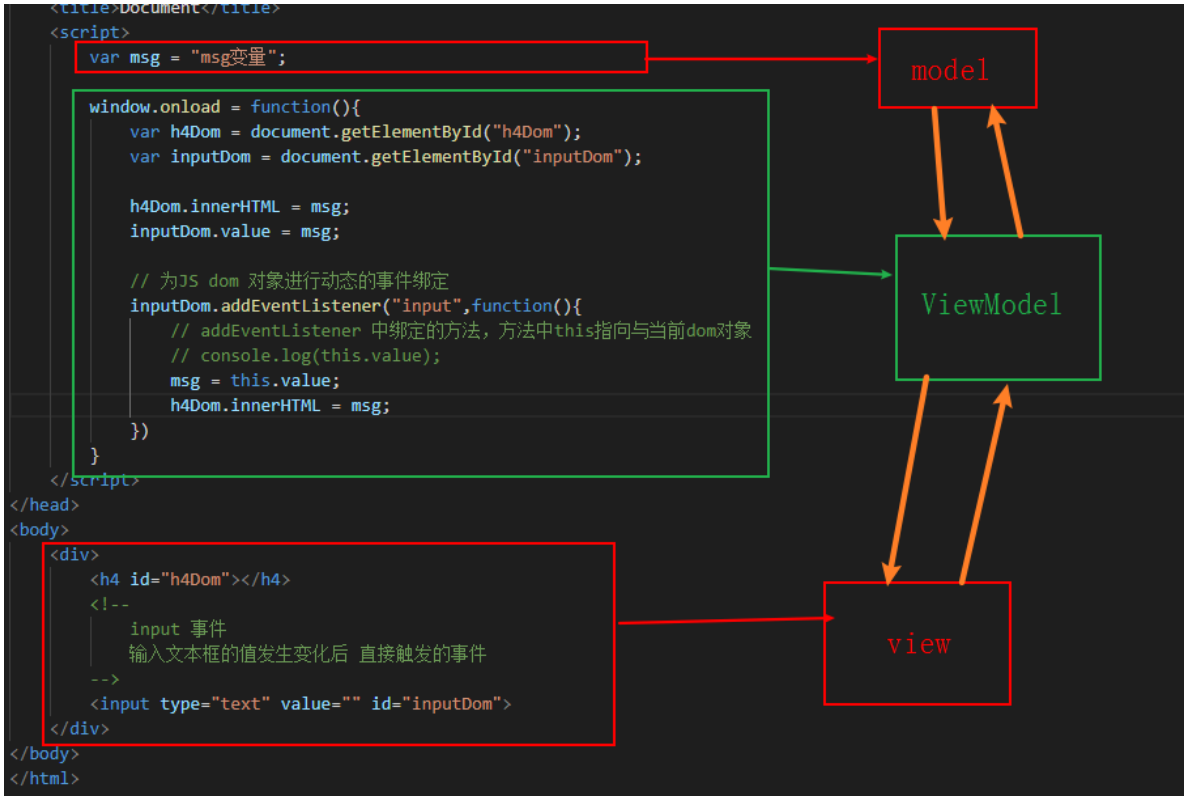
- SPA：single page application 单页应用程序
  - SPA应用将所有的**活动局限于一个 Web 页面中**，仅在该 Web 页面初始化时加载相应的 HTML 、js 、CSS
  - SPA应用一旦页面加载完成（容器HTML文件），SPA 不会因为用户的操作而进行页面的重新加载或跳转
  - SPA应用利用 JavaScript 动态的变换 HTML，从而实现UI与用户的交互



1.2、MVVM

- mvvm：model-view-viewModel 模型 视图 视图模型
  - 模型：指的是构成页面内容的相关数据（包含：前端定义的数据，后端传递的数据）
  - 视图：指的是呈现给开发这和用户查看的展示数据的页面
  - 视图模型：mvvm设计模式的核心思想，它是连接view和model的桥梁。
  - **实现MVVM设计思想的框架，基本上都完成对DOM功能的极限封装，开发者几乎不用操作JS-DOM就可以完成页面的数据的关联交换**





> Tips：前端实现MVVM设计思想的框架（Vue,React,Angular,微信小程序.....），其目的都是为了高度封装view-model 的交互过程，让开发这只用关心页面构成和数据构成，无需花费大量时间关心数据和页面的状态关系

2、Vue简介

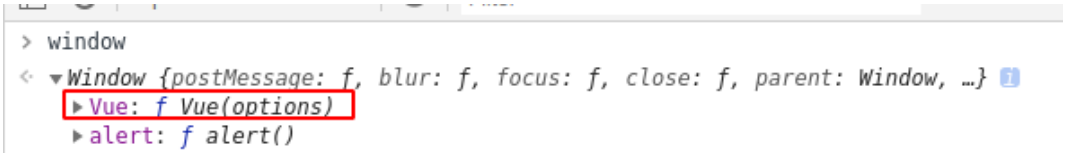
- Vue (读音 /vjuː/，类似于 **view**) 是一套用于构建用户界面的**渐进式框架**
  - 渐进式框架：在使用和学习式方式简单，学习成本较低，随着深入学习根据需求进行功能扩展
- Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合
- Vue借鉴React和Angular的部分代码设计，并提高了易用性和轻量化

3、Vue的页面基本使用

- vue并不适合直接使用 页面方式进行语法定义；
- 页面使用方式只是vue为了让开发者在学习语法时可以快速掌握；
- 获取vue的核心语法库
  - 通过地址 https://cdn.jsdelivr.net/npm/vue/dist/vue.js 下载vue核心语法包
  - 使用 npm 进行Vue语法库的下载 `npm install vue`
- 页面在Vue库

```
<script src="../js/vue.js"></script>
```

  - 页面装载Vue核心语法后，会在浏览器window对象中提供一个全局的构造方法**Vue**



```
> Vue
< f Vue (options) {
      if (!(this instanceof Vue)
      ) {
        warn('Vue is a constructor and should be called with the `new` keyword');
      }
      this._init(options);
    }
>
```

- **Vue函数为一个JS对象构造器，使用时需要通过 new 关键字进行 Vue 对象创建**

- 页面基本关联和应用

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <!--
        开发规范：
            变量和方法名的人为规定：变量和方法名一定具有意义，变量使用名词，方法使用动词
            JS的人为规范中：构造函数变量名 首字母 大写
                            私有属性和方法以 _ 开头  （提示开发者该属性和方法不能外部直接调用）
     -->
    <!--
        Vue核心库装载后，提供的 全局Vue 实际上是一个 JS 环境下对象构造器函数

        页面使用时需要通过 new 关键字 创建 基于Vue构造器的 实例对象

     -->
    <!-- <script src="./js/vue.js"></script> -->
    <!-- 默认关闭了调试和日志功能 -->
    <script src="./js/vue.min.js"></script>
    <script>
        // document.querySelector(el);
        // document.querySelectorAll(el);

        window.onload = function(){
            var vm = new Vue({
                // key:value
                // 该配置项key的取值为 Vue框架定义的具有特点意义的相关属性
                // el:element select DOM 选择器
                //    取值：string ==> 符合CSS元素选择器语法规范的字符串
                // el:"#app", // vue实例和页面容器的关联关系=>当前实例可使用vue语法的页面范围
                el:".aaa", // 只针对于第一个匹配的元素生效 => dom.querySelector(el);
                // 实例的数据仓库
                //     提供给开发者定义自定义变量的代码区域
                //     被定义在数据仓库中的变量，可以在当前实例的容器范围内使用
                data:{
                    msg:"msg变量",
                    aaa:20,
                    bbb:[1,2,3,4],
                }
            });
            // console.log(vm);
        }
```

```
    </script>
</head>
<body>
    <h4>ccccc</h4>
    <div id="app" class="aaa">
        <!--
            插值表达式：将对应该容器实例中，数据仓库里的变量取值到页面
        -->
        <h4>{{msg}}</h4>
        <!-- 指令 -->
        <input type="text" v-model="msg">
        <hr>
        <p>{{ aaa }}</p>
    </div>
    <div class="aaa">
        <h3>{{ msg }}</h3>
        <p>{{ aaa }}</p>
    </div>
</body>
</html>
```

- Vue官方开发调试工具 `vue devtools` ：工具在开发环境下可以实现浏览器对vue功能的基本监控

4、Vue的全局环境配置

- 在vue项目运行启动前，对vue的运行环境进行相关功能设置
  - 开启关闭调试工具，关闭开启控制台日志和警告，关闭开启调试工具.....
  - 所有的Vue全局环境设置依赖于Vue的全局配置对象`Vue.config`

```
// 取消 Vue 所有的日志与警告 , 取值类型：boolean 默认值：false
Vue.config.silent = true;

//配置是否允许 vue-devtools 检查代码 , 取值类型：boolean
//开发版本默认为 true，生产版本默认为 false。生产版本设为 true 可以启用检查。
Vue.config.devtools = true

// 设置为 false 以阻止 vue 在启动时生成生产提示 , 取值类型：boolean 默认值：true
Vue.config.productionTip = false;
```

5、基本交互

5.1、插值表达式

- 语法：`Mustache语法` `{{ }}` **语法只提供书写规范，不提供功能**
- 功能：取决于JS框架为其赋予的 功能特性 == **在不同的框架下 语法功能能不同**
- **特性：响应式数据功能：**
  - HTML标签通过插值表达式绑定Vue的数据变量，当变量发生变化时该标签会重新渲染加载。
  - 插值表达式只能对 Vue 数据变量变化作为响应，而无法修改变量
- 对于Vue框架而言：只能被用于 html 标签的主体内容中，不能用于标签的其它位置
- 对于Vue框架而言：只能绑定对应 Vue对象数据仓库中定义的变量，或者 **简单的 JS 表达式** 和 **JS 内置对象**

```
<标签>{{ Vue对象数据仓库变量|JS表达式|JS内置对象 }}</标签>
```

> Tips:当Vue数据仓库中变量名称和JS内置对象名称相同时，Vue将优先使用仓库中变量

- 对于不同类型的数据 ，为保证页面输出正确结果，vue插值表达式 对所有 变量调用了自定义的 **toString()**方法

```
var _toString = Object.prototype.toString;

function isPlainObject (obj) {
    return _toString.call(obj) === '[object Object]'
}

function toString (val) {
    return
        // 判断传入的值是不是 null 或者 undefined
        val == null ?
            '':
            // val 是不是数组 或
            Array.isArray(val) || (isPlainObject(val) && val.toString === _toString)?
                // 将数组和对象转换为 字符串模式
                JSON.stringify(val, null, 2):
                // 将 val 强制转换为 string (和变量直接执行原生JS toString 方法效果一样)
                String(val)
}
```

- 值表达式 底层调用的 是 DOM 对象的 textContent 属性 进行值得写入操作
  - **html格式字符串将不被解析**
  - js 转义符将不被识别

```
<body>
    <!--
        html 标签中可以定义哪些数据类型值？
            html页面无论运行环境发生什么变化 ， 被展示的数据最终都以字符串形式呈现
     -->
    <div id="app">
        <!--
            基本用法：{{ 变量名称 }} 进行数据获取展示
            取值范围：当前实例对应容器的数据仓库中的变量
            取值类型：any
```

```
            vue框架对插值表达式取值的变量 作为字符串转换操作==>vue核心语法中重新定义了toString方法
            + null 和 undefined 最终会以 “” 字符串方式输出
            + Array 和 [object Object] 会使用 JSON.stringify()
            + 除了上述数据其他数据一律使用 强制类型转换 String()
        -->
        <p>string:{{ msg }}</p>
        <p>number:{{ num }}</p>
        <p>boolean:{{ flag }}</p>
        <p>array:{{ arr }}</p>
        <p>Object:{{ user }}</p>
        <p>null:->{{ arg1 }}<-</p>
        <p>undefined:->{{ arg2 }}<-</p>
        <p>Image：{{ imgDom }}</p>
        <p>Date:{{ day }}</p>
        <!--
            vue的插值表达式可以直接，以JS语法调用 匿名变量
        -->
        <p>number匿名变量：{{ 100 }}</p>
        <p>string匿名变量：{{ "字符串" }}</p>
        <p>boolean匿名变量：{{ true }}</p>
        <p>array匿名变量：{{ [1,2,3,4,5] }}</p>
        <p>object匿名变量：{{ {a:1,b:2} }}</p>
        <p>Date匿名变量：{{ new Date() }}</p>
        <p>对象：{{ user }}</p>
        <p>对象中的属性：{{ user.name }}</p>
        <p>数组元素：{{ arr[0] }}</p>

        <!--
            插值表达式可取值：JS表达式
            + 在插值表达式的定义范围内，可以直接进行简单的 js 运算
                - 四则运算
                - 逻辑运算
                - 比较运算
                - 赋值运算
                - 三目运算
            * 总结：插值表达式在vue环境下运算时保留原变量类型，当运算结束后向页面输出时转换为字符串类型
        -->
        <p>四则：{{ 1+1 }}</p>
        <p>四则：{{ num+1 }}</p>
        <p>逻辑：{{ flag&&false }}</p>
        <p>比较:{{ num<=99 }}</p>
        <!-- 修改对应变量 -->
        <p>赋值运算：{{ msg='新字符串' }}</p>
        <p>三目运算：{{ flag?"真":"假" }}</p>
        <!--
            JS内置对象 (Math)
         -->
        <p>Math:{{ Math }}</p>
        <p>Math:{{ Math.pow(2,2) }}</p>
        <p>Math:{{ Math.random() }}</p>
        <p>Math:{{ Math.PI }}</p>
        <!--
            插值表达式使用 标签格式字符串
                + 插值表达式实际上是通过调用 textContent 方式向标签中定义数据变量
                + textContent 和 innerText 在标签格式字符串处理上效果一样
                    - innerText 当文本中出现 \n 会将 \n 提花为<br> : 文本解析属性
```

```
                    - textContent 当文本中出现 \n 直接保留特性向页面输出
         -->
        <div>
            {{ htmlStr }}
        </div>
    </div>
</body>
<script>
    new Vue({
        el:"#app",
        data:{
            msg:"测试数据",
            num:100,
            flag:true,
            arr:[1,2,3,4],
            user:{
                name:"tom",
                age:23
            },
            arg1:null,
            arg2:undefined,
            imgDom:new Image(),
            day:new Date(),
            // 不要使用JS内置关键字
            // Math:"vue示例自定义的math"
            htmlStr:"<h3>h3标签</h3>",
            str:"aaaa\nbbbbbb\n\tccccc"
        }
    })
</script>
```

5.2、基础指令

- 为开发者 提供 在页面中进行 特殊功能的属性描述方法
  - 语法：Vue指令以 `v-名称` 结构定义
  - 位置：**指令只能被用于 html 容器的标签属性上** `<标签 v-指令="" ></标签>`
  - 实现：指令本身实际上就是一个JS方法的特殊封装，页面定义的指令只是对方法的调用和触发
  - 功能：通过指令可实现 HTML标签写入，标签判断、标签循环、标签事件绑定、标签属性绑定.....
- 完整语法：
  - `v-指令名[:参数][.修饰符][=取值]`
  - `v-指令名[:参数][.修饰符]["取值"]`
    - 参数：对当前指令操作范围进行限制
    - 修饰符：限制指令功能的触发条件

> Tips：
>
> 1、普通指令取值范围和插值表达式基本一致，可取Vue数据仓库中定义的变量，可取匿名变量，可取JS内置对象、可进行简单的四则运算；
>
> 2、对于特殊指令 `v-on` 只能绑定Vue方法仓库中的自定义方法，或绑定简单JS表达式

- **指令特性：无痕迹特性==代码开发时标签上的vue语法表达式，在项目运行时会被删除，不会保留**

> vue对象和容器完成语法解析后，不会在浏览器上保留语法定义规范

1、v-text

- 取值：`string`
- 功能：更新元素的 `textContent`。如果要更新部分的 `textContent` ，需要使用 `{{ Mustache }}` 插值。
- 示例：`<span v-text="msg"></span>`

```
<body>
    <div id="app">
        <!-- v-text 等效于 插值表达式 -->
        <p>msg:{{ msg }}</p>
        <p v-text=" 'msg:'+msg "></p>
        <!--
            HTML 规范中并不强制要求 标签属性取值一定要定义 引号
        -->
        <p v-text=msg ></p>
    </div>
</body>
<script>
    new Vue({
        el:"#app",
        data:{
            msg:"变量msg"
        }
    })
</script>
```

2、v-html

- 取值：`string`
- 功能：更新元素的 `innerHTML`
- 示例：`<div v-html="html"></div>`

```
<body>
    <div id="app">
        <!-- v-html -->
        <p>{{ htmlStr }}</p>
        <p v-text="htmlStr"></p>
        <p v-html="htmlStr"></p>
    </div>
</body>
<script>
    new Vue({
        el:"#app",
        data:{
            htmlStr:"<h3>html格式字符串</h3>"
        }
    })
</script>
```

3、v-pre

- 取值：**不需要表达式**，该指令为boolean类型属性
    - 写表示 true（启用功能）不写表示 false（不启用功能）
- 功能：跳过这个元素和它的子元素的编译过程。可以用来显示原始 Mustache 标签。跳过大量没有指令的节点会加快编译。

- 示例：`<span v-pre>{{ 该语法会直接显示在页面 }}</span>`

```
<body>
    <div id="app">
        <!--
            v-pre 指令不是html的pre标签功能
            v-pre指令用于限定被当前属性绑定的标签和标签内部的vue语法不被执行
        -->
        <!-- <pre>adsdasd
            asdasdasd
        </pre> -->
        <p><span v-pre>{{ msg }}是vue插值表达式取值语法，msg值为：</span>{{ msg }}
</p>
    </div>
</body>
<script>
    new Vue({
        el:"#app",
        data:{
            msg:"变量msg"
        }
    })
</script>
```

4、v-once

- 取值：**不需要表达式**，该指令为boolean类型属性
  - 写表示 true（启用功能）不写表示 false（不启用功能）
- 功能：对当前元素和内部元素vue功能执行**一次**，程序执行过程不在对该元素范围内的vue功能进行重新执行
- 示例：`<span v-once>该区域vue功能只在初始化时执行一次 {{msg}}</span>`

```
<body>
    <div id="app">
        <p>{{ msg }}</p>
        <p v-pre>{{ msg }}</p>
        <!--
            v-once修饰的标签内部vue语法可以正常执行一次
            当第一次执行完成后，内部的vue所有语法彻底失效
        -->
        <p v-once>{{ msg }}</p>
    </div>
</body>
<script>
    new Vue({
        el:"#app",
        data:{
            msg:"变量msg"
        }
    })
</script>
```

5、v-cloak

- 取值：**不需要表达式**，该指令为boolean类型属性（**vue没有为该指令增加任何功能**）

- 官方解释：该指令可以在vue示例未被初始化前，隐藏vue相关构成模板
- 功能：实现在vue功能构建完成前，隐藏浏览上vue语法表达式，**该指令本身不具有特殊功能，需配合css样式实现效果**
- 示例：

```
[v-cloak] {
  display: none;
}
```

```
<div v-cloak>
  {{ message }}
</div>
```

> Tips：v-cloak指令功能主要利用了 **指令特性** + css 样式实现，因此所有的vue指令都可以实现该功能

6、v-on

- 缩写：`@` ，项目中可用 `@` 替代 `v-on:`
- 语法：

```
<button v-on[:参数][.修饰符][.修饰符]……="取值"></button>
<button @[参数][.修饰符][.修饰符]……="取值"></button>
```

- 取值：`Function | Inline Statement(行内表达式) | Object | Array`
- 参数：eventName（事件名称）
  - 什么是JS的事件名？
- 功能：绑定元素事件监听器，事件类型由参数指定
- v-on 绑定的事件必须是vue对象 **方法仓库** 中的一个自定义方法，**且该函数中的this为当前vue的实例对象**

```
new Vue({
    // 当前vue实例对应的方法仓库
    methods:{

    }
})
```

绑定示例：

```
<script>
    function show(e){
        console.log("show",e);
    }
    function print(e){
        console.log("print",e);
    }
    window.onload = function(){
        var btnDom = document.getElementById("btn1");
        // 为指定DOM元素增加事件关联
        //      eventName 事件名
        //      callback 事件触发的执行函数
```

```
        //     options 事件行为描述配置（事件冒泡，事件捕获，默认事件）
        // btnDom.addEventListener(eventName ,callback [,options]);
        btnDom.addEventListener("click",function(){
            console.log("click1");
        })
        btnDom.addEventListener("click",function(){
            console.log("click2");
        })
        btnDom.addEventListener("click",function(){
            console.log("click3");
        })
    }
</script>
<body>
    <!--
        1、标签上以 on开头的事件定义功能=>标签事件功能赋值
    -->
    <!--
        静态事件绑定：以标签属性关联事件方法
        动态事件绑定：以JS语法方式为元素关联事件定义
     -->
    <input type="button" value="按钮1" onclick="show(event)" onclick="print()">
    <input type="button" value="按钮2" id="btn1">
    <hr>
    <div id="app">
        <!--
            v-on 指令的事件绑定,采用addEventListener方式实现事件绑定
                - 事件名用于描述绑定的事件类型
                - 完成的是事件的监听
            + v-on 绑定的事件执行函数，必须为当前容器对应Vue示例的方法仓库中，定义的自定义函数
        -->
        <input type="button" value="事件绑定1" v-on:click="show()" v-on:click="print()">
        <hr>
        <!-- 事件绑定的取值方式 -->
        <!--
            同事件类型的多处理函数绑定方式，执行先后取决于定义方法的先后
                1、在事件绑定时以 ，  方式隔开多个事件
                2、通过 v-on 的取数组类型值进行定义
        -->
        <input type="button" value="事件绑定2" v-on:click="print(),show()">
        <input type="button" value="事件绑定3" v-on:click=" [ print(),show() ] ">
        <hr>
        <!--
            不同类型的事件处理函数绑定，执行先后取决事件的调用先后
                1、直接在标签上以 vue 事件绑定语法进行多次绑定即可
                2、通过 v-on 的取对象类型值进行定义
                    - 当v-on 取值为对象时，不能使用 参数 修饰事件类型
                    - key描述事件类型，value用于指定事件处理函数的名称 （**不能加括号**）
        -->
        <input type="button" value="事件绑定4"
               v-on:click="clickFun()"
               v-on:mousedown="mousedownFun()">
        <input type="button" value="事件绑定5"
               v-on=" { click:clickFun,mousedown:mousedownFun } " >
        <hr>
```

```
        <!--
            行内表达式，事件绑定的行内表达式一定是一个赋值表达式
         -->
         <p>msg: {{ msg }}</p>
         <input type="button" value="修改变量msg值" @click="setMsg()">
         <input type="button" value="修改变量msg值" @click=" msg='新值' ">
    </div>
</body>
<script>
    new Vue({
        el:"#app",
        data:{
            msg:"旧值"
        },
        // 当前实例的方法仓库
        methods:{
            // key:value
            // key 自定义方法名
            // value 取值为 Function 类型，描述该函数执行体
            show:function(){
                console.log("vue中自定义方法 show")
            },
            print:function(){
                console.log("vue中自定义方法 print")
            },
            // ES6 提供对象定义简写方式
            //  1、当定义的属性取值为匿名函数时可以直接简写为 属性名(){}
            //  2、定义的属性名和取值变量名名称一致，可以直接简写属性名
            clickFun(){
                console.log("clickFun");
            },
            mousedownFun(){
                console.log("mousedownFun");
            },
            setMsg(){
                // console.log(this); // vue源码中，将vue实例对象的 **一级** 箭头函数通过 call 方式将this指向了window
                // 1、vue将数据仓库和方法仓库中的定义值，直接构建在了当前vue实例的根属性上
                // console.log("setMsg",this.msg);
                // this.show();
                this.msg = "新值";
            }
        }
    })
</script>
```

参数示例

```
   <script>
       function inputFun(arg) {
           var dom = document.getElementById("aa");
           console.log("input输入",dom.value,arg)
       }
       function changeFun() {
           console.log("change输入")
       }
```

```
        function inputEvent(e) {
            console.log("event",e)
        }
    </script>
    <!-- <input type="button" value="" onclick="show( 'str',100,true )"> -->
    <!-- 定义的方法中 形参 this 指向与当前的 dom 对象 -->
    <input type="text" id="aa" oninput="inputFun(this.value)"
onchange="changeFun()">

    <hr>
    <!--
        HTML和JS 构成中 event 事件源对象
        event.target  ==> 记录当前事件触发的DOM元素
    -->
    <input type="button" value="点击事件" onclick="inputFun(event)">
    <input type="text" oninput="inputFun(event)">
    <input type="text" oninput="inputFun(event.target.value)">
    <input type="button" value="点击事件" onclick="inputFun">


    <hr>
    <!--
        js的 形参 和 实参
            形参：就是js方法定义时吗，描述的该方法所要接收的参数的 参数名称
            实参：js方法执行时传入的固定参数
    -->
    <div id="app" >
        <!--
            vue事件绑定时的形参传递,执行方法是可以根据需求直接传入实参值
                1、vue事件绑定传入的方法参数，可以是符合JS语法习惯的相关值和表达式
                2、vue的事件绑定可以直接以 当前示例数据仓库变量作为参数
        -->
        <input type="button" value="输出info变量-btn1" @click="printInfo( 'btn1'
)">
        <input type="button" value="输出info变量-btn2" @click="printInfo( 'btn2'
)">
        <input type="button" value="输出info变量-btn3" @click="printInfo( 'btn3'
)">
        <hr>
        <input type="button" value="输出Math对象" @click="printArgs( Math )">
        <input type="button" value="每次执行数据一个随机数据" @click="printArgs(
Math.random() )">
        <hr>
        <input type="button" value="输出info变量" @click="printArgs( info )">
        <input type="button" value="输出msg变量" @click="printArgs( msg )">
        <hr>
        <!-- vue示例容器中的标签，绑定的事件如果传入this，此时的this恒定指向与 window
对象 -->
        <input type="button" value="传入形参this" @click="printThis( this )">
        <hr>
        <!--
            vue事件默认从vue示例的数据仓库和方法中取对应的变量
            vue将常用的事件源对象 重新定义为 $event 等同于 普通 js中 event事件源
        -->
        <input type="text" @input=" printValue( $event ) ">
        <input type="text" @input=" printValue( $event.target ) ">
        <input type="text" @input=" printValue( $event.target.value,'ssss' ) ">
        <hr>
```

```
        <!--
            vue的事件绑定 可以不用定义 事件名称后的 ()
            ==> 这种事件绑定方式，是vue提供给组件化数据传递使用的事件绑定方式  ？？？？
        -->
        <input type="button" value="事件绑定" @click="printMsg()">
        <input type="button" value="事件绑定" @click="printMsg">
    </div>
<script>
    new Vue({
        el: "#app",
        data:{
            info:"info默认值",
            msg:"msg默认值"
        },
        methods:{
            printMsg(){
                console.log("事件触发")
            },
            printInfo(arg){
                console.log(arg,":",this.info)
            },
            printArgs(arg){
                // console.log(Math);
                // console.log(this.info);
                console.log(arg);
            },
            printThis(arg){
                console.log(arg);
            },
            printValue(arg,arg1){
                console.log("用户输入值: ",arg,arg1);
            }
        }
    })
</script>
```

- **修饰符： 事件传递机制处理功能**
- `.stop` - 调用 `event.stopPropagation()`。
- `.prevent` - 调用 `event.preventDefault()`。
- `.capture` - 添加事件侦听器时使用 capture 模式（事件捕获模式）
- `.self` - 只当事件是从侦听器绑定的元素本身触发时才触发回调。

下述修饰符用于修饰事件出发的按键

- `.{keyCode-按键编码 | keyAlias-按键描述}` - 只当事件是从特定键触发时才触发回调。
- `.left` - (2.2.0) 只当点击鼠标左键时触发 或者 键盘方向左键。
- `.right` - (2.2.0) 只当点击鼠标右键时触发 或者 键盘方向右键
- `.up` 键盘方向上键
- `.down` 键盘方向下键
- `.middle` - (2.2.0) 只当点击鼠标中键时触发。

一次事件修饰符

- `.once` - 只触发一次回调。
- **`.native` - 监听组件根元素的原生事件。=> 应用于组件化技术的 ？？？？**

- `.passive` - (2.3.0) 以 `{ passive: true }` 模式添加侦听器（启用默认事件功能）

> Tips：addEventListener() 事件绑定中的 passive 属性和 preventDefault 功能的关系
>
> 元素事件每次被触发，浏览器都会去查询被执行行为是否有preventDefault阻止该次事件的默认动作。
>
> 加上**passive就是为了告诉浏览器，不用查询了，执行 方法中没用preventDefault阻止默认动作。**
>
> 这里一般用在滚动监听，@scoll，@touchmove 中。因为滚动监听过程中，移动每个像素都会产生一次事件，每次都使用都进行查询prevent会使滑动卡顿。通过passive将内核线程查询跳过，可以大大提升滑动的流畅度。
>
> **注：Vue时间绑定时，passive和prevent冲突，不能同时绑定在一个监听器上。**

7、v-show

- 取值：`any`
- 功能：根据表达式的boolean结果，**切换元素的 `display` CSS 属性，控制元素显示隐藏**
- 示例：`<p v-show=" flag "></p>`

```
<div id="app">
    <h1 v-show=" true ">h1标签1</h1>
    <h1 v-show=" false ">h2标签2</h1>
    <h1 v-show=" '' ">h2标签3</h1>
    <h1 v-show=" 'asdasda' ">h2标签4</h1>
    <h1 v-show=" 0 ">h2标签5</h1>
    <h1 v-show=" 1 ">h2标签6</h1>
    <h1 v-show=" [] ">h2标签7</h1>
    <h1 v-show=" {} ">h2标签8</h1>
    <hr>
    <h1 v-show=" flag ">请登录</h1>
    <p v-show=" !flag ">当前用户tom,账户余额300</p>
</div>
<script>
    new Vue({
        el:"#app",
        data:{
            flag:true,
            num:10
        }
    })
</script>
```

8、v-if、v-else-if、v-else

- 取值：
  - v-if：`any`
  - v-else-if：`any`
  - v-else：**不需要表达式**，该指令为boolean类型属性
- 用法：根据表达式的boolean结果，**执行元素的创建（createElement）和删除操作（removeElement）**
- 示例：

```
<div v-if="type === 'A'">A</div>
<div v-else-if="type === 'B'">B</div>
<div v-else>Not A/B/C</div>
```

- `v-else` 指令的上一个元素 必须使用了 `v-if` 或者 `v-else-if`
- `v-else-if` 指令的上一个元素 必须使用了 `v-if`

```
<div id="app">
    <h1 v-if=" flag ">请登录</h1>
    <p v-if=" !flag ">当前用户tom,账户余额300</p>
    <input type="button" value="显示隐藏" @click=" flag=!flag ">
    <hr>
    <p v-if="num<10">小于10</p>
    <p v-else-if="num==10">等于10</p>
    <p v-else>大于10</p>
    <hr>
    <p v-if="num<10">小于10</p>
    <p v-if="num==10">等于10</p>
    <p v-if="num>10">大于10</p>
    <hr>
    <p v-show="num<10">小于10</p>
    <p v-show="num==10">等于10</p>
    <p v-show="num>10">大于10</p>
    <input type="button" value="+" @click=" num = num+1 ">
    <input type="button" value="-" @click=" num = num-1 ">
</div>
<script>
    new Vue({
        el:"#app",
        data:{
            flag:true,
            num:10
        }
    })
</script>
```

9、v-for

- 取值：`Array | Object | number | string`
- 功能：基于数据多次渲染元素或模板块
- 语法：`<标签 v-for=" 取值表达式 in 待循环值 "></标签>`
  - 取值表达式：可直接定义临时变量；
    - 临时变量，**具有vue实例数据仓库的读取特性**，
    - 临时变量，**只能在当前循环标签的属性和内部进行使用**
    - 临时变量，在相同容器和示例范围内，循环临时变量优先级高于vue实例数据仓库的变量

```
<!--
    1、items 类型为 Array
        -> item 为数组循环时当前循环到的数组元素  [1,2,3,4]
    2、items 类型为 Object
        -> item 为对象循环时当前循环到的键值对的 值
    3、items 类型为 number
        -> item 数值从1开始的累加值
```

```
    4、items 类型为 string
        -> item 从字符串下标0开始的每次循环的 字符
-->
<div v-for="item in items">
  {{ item.text }}
</div>
<!--
    1、items 类型为 Array
        -> item 为数组循环式当前循环到的数组元素
        -> arg2 表示当前数组循环到的下标值（索引）
        -> arg3 只在循环对象时有效
    2、items 类型为 Object
        -> item 为对象循环时当前循环到的键值对的 值
        -> arg2 为对象循环时当前循环到的键值对的 键
        -> arg3 vue为对象循环提供的计数器（从 0 ）
    3、items 类型为 number
        -> item 数值从1开始的累加值
        -> arg2 循环次数的统计 （从 0 ）
    4、items 类型为 string
        -> item 从字符串下标0开始的每次循环的 字符
        -> arg2 字符下标（索引）
-->
<div v-for="(item [,arg2] [,arg3]) in items">
  {{ item.text }}
</div>
```

- 取值表达式：也可以为数组索引指定别名 (或者用于对象的键)：

```
<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>
```

- **辅助渲染**：`v-for` 指令为提高性能采用部分标签渲染操作（**只针对与需要添加和删除的标签进行渲染操作**）
  - `v-for` 默认不改变整体，而是**重复替换使用已有元素**，该方式会导致页面排序功能展示出现问题。
  - 为迫使其`v-for`重新排序元素，需要提供一个 `key` 的特殊属性，为每个元素提供唯一key值 **不能使用下标**

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

10、v-bind

- 缩写：`:`，项目中可用 `:` 替代 `v-bind`
- 语法：

```
<p v-bind[:参数][.修饰符]="取值"></p>
<p :[参数][.修饰符]="取值"></p>
```

- 取值：`any (对应属性取值) | Object (对应属性取值)`
- 参数：`attrOrProp (optional)`

    - attr:HTML标签属性（规范所定义的可以直接在 标签上定义的 具有功能的属性）
    - prop:HTML标签对应的DOM属性（标签转为JS DOM对象后，所具有的属性）
- 作用：为指定标签绑定需要完成操作的 属性 值
- 修饰符：
    - `.prop` - 被用于绑定 DOM 属性 (property)。
    - `.camel` - (2.1.0+) 将 kebab-case 特性名转换为 camelCase. (从 2.1.0 开始支持)
    - **`.sync` (2.3.0+) 语法糖，会扩展成一个更新父组件绑定值的 `v-on` 侦听器。？？**
- 用法：（**class style boolean类型属性**）

  动态地绑定一个或多个特性，或一个组件 prop 到表达式。

  在绑定 `class` 或 `style` 特性时，支持其它类型的值，如数组或对象。可以通过下面的教程链接查看详情。

  在绑定 prop 时，prop 必须在子组件中声明。可以用修饰符指定不同的绑定类型。

  没有参数时，可以绑定到一个包含键值对的对象。注意此时 `class` 和 `style` 绑定不支持数组和对象。
- 示例：

```
<!-- 绑定一个属性 -->
<img v-bind:src="imageSrc">
```

11、指令总结

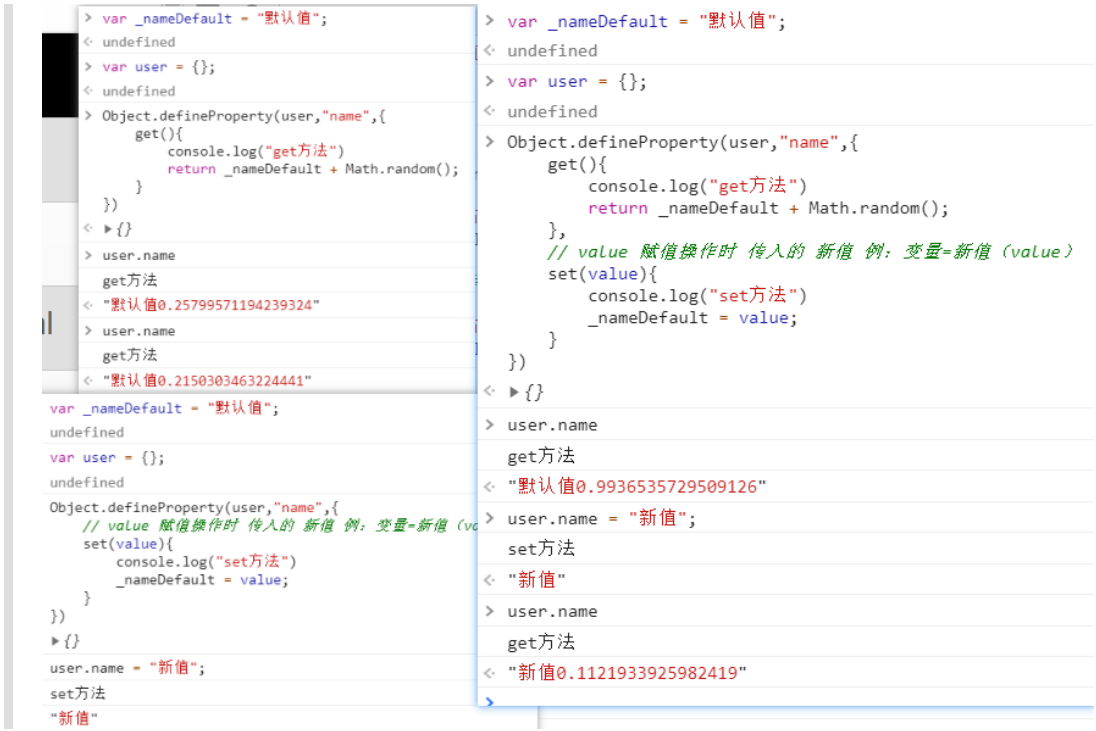
- **功能指令：`v-pre`、`v-cloak`、`v-on`、`v-once`**
- **构成指令：`v-text`、`v-html`、`v-show`、`v-if`、`v-else`、`v-else-if`、`v-for`、`v-bind`**
- Vue项目的构成指令和插值表达式具有相同的**响应式特性**
    - **响应式功能：实际上就是所谓的数据状态同步操作；特指项目构成中，内存中变量数据变化，页面会及时做出响应（页面重构渲染）**

> Tips：扩展阅读《现代 js 框架存在的根本原因》
>
> 英文原文：https://medium.com/dailyjs/the-deepest-reason-why-modern-javascript-frameworks-exist-933b86ebc445
>
> 中文翻译：https://www.zcfy.cc/article/the-deepest-reason-why-modern-javascript-frameworks-exist

5.3、响应式原理

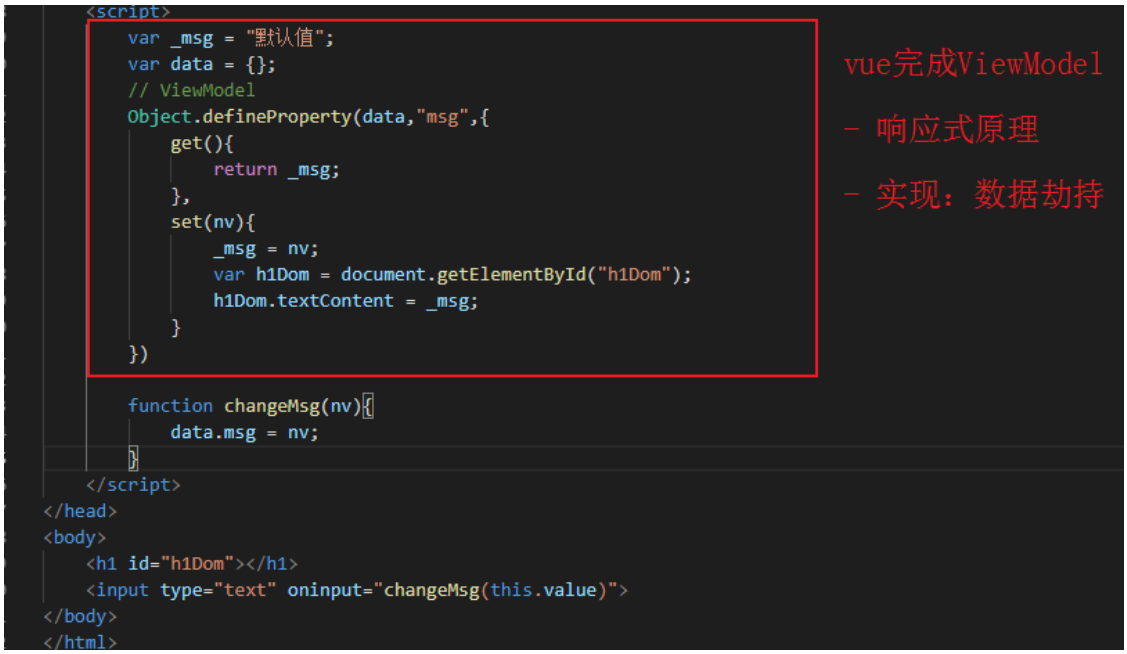
- 响应式原来实现基础：`Object.defineProperty(obj, prop, descriptor)`

> obj：被操作的目标对象。 prop：被操作的属性名称。 descriptor：详细定义，被操作的属性所具有的相关功能。
>
> - get（function）：拦截属性取值操作行为，为取值操作提供扩展功能定义区，默认为 undefined。
> - set（function）：拦截属性赋值操作行为，为属性赋值操作提供扩展功能定义区，默认为 undefined。



```
var _nameDefault = "默认值";
undefined
var user = {};
undefined
Object.defineProperty(user,"name",{
    get(){
        console.log("get方法")
        return _nameDefault + Math.random();
    },
    // value 赋值操作时 传入的 新值 例：变量=新值（value）
    set(value){
        console.log("set方法")
        _nameDefault = value;
    }
})
{}
user.name
get方法
"默认值0.9936535729509126"
user.name = "新值";
set方法
"新值"
user.name
get方法
"新值0.1121933925982419"
```

- `Object.defineProperty()` 功能也被称之为 javaScript 的 **数据劫持**
- `Object.defineProperty()` 是JS语法中提供**可控变量定义方式**



```
<script>
    var _msg = "默认值";
    var data = {};
    // ViewModel
    Object.defineProperty(data,"msg",{
        get(){
            return _msg;
        },
        set(nv){
            _msg = nv;
            var h1Dom = document.getElementById("h1Dom");
            h1Dom.textContent = _msg;
        }
    })

    function changeMsg(nv){
        data.msg = nv;
    }
</script>
</head>
<body>
    <h1 id="h1Dom"></h1>
    <input type="text" oninput="changeMsg(this.value)">
</body>
</html>
```

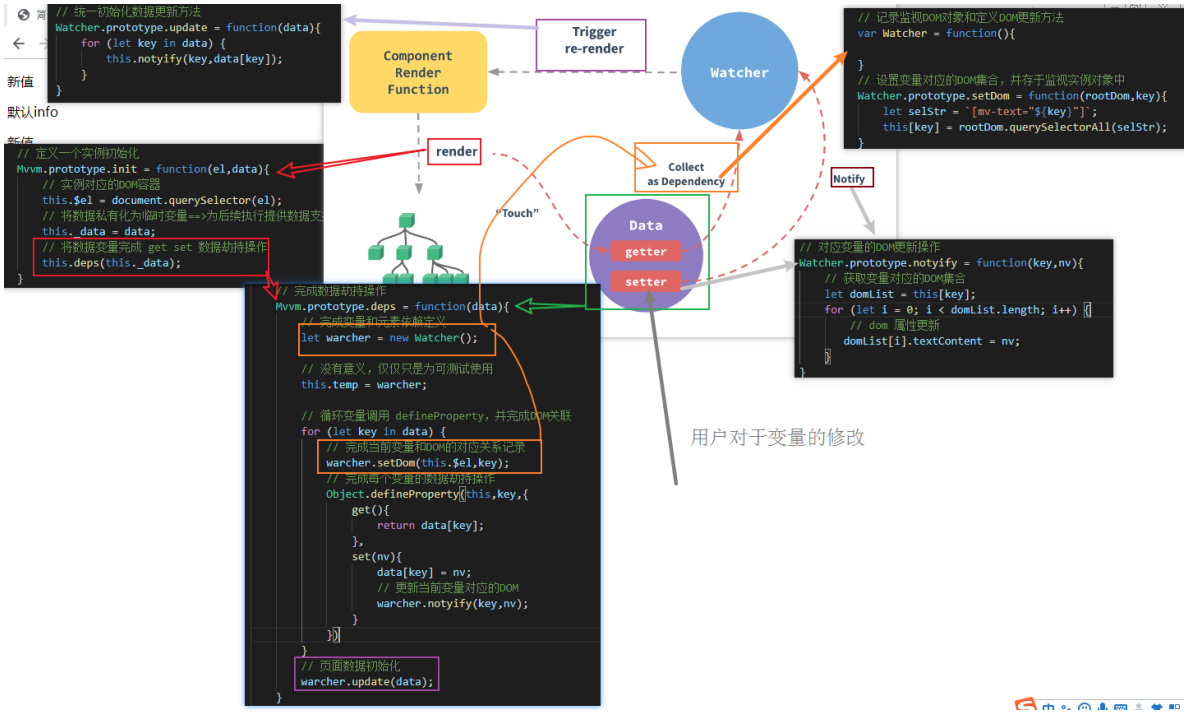
vue完成ViewModel

- 响应式原理
- 实现：数据劫持

- ES6 为了加强该功能，提出了一个新的对象 proxy，实际是 Object.defineProperty() 强化型
- **Vue3.0 核心语法中将全面使用 proxy 替代 Object.defineProperty()**

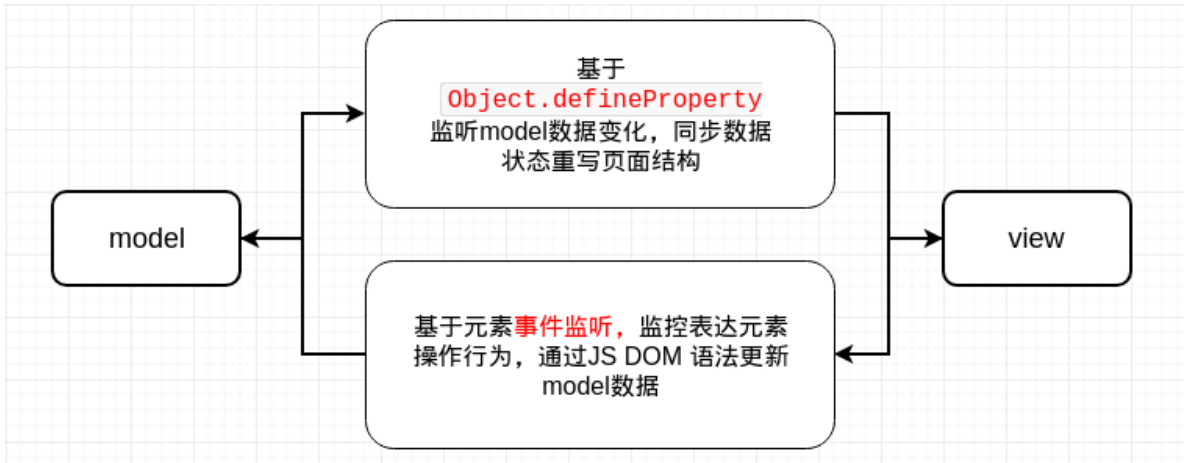
- 基本使用：
  - 语法：`Object.defineProperty(obj, prop, descriptor)`
  - 参数：
    - `obj` 要在其上定义属性的对象。
    - `prop` 要定义或修改的属性的名称。

    - `descriptor`将被定义或修改的属性描述符。
- 返回：被传递给函数的对象。
- **vue响应式简单实现**
    - 非侵入式：框架设计思路=框架定义完成后，后续框架升级，不会对开发者已经完成的项目造成较大区域的代码修改
    - vue框架的浏览器适用性：Vue 不支持 IE8 以及更低版本浏览器



5.4、双向数据绑定指令v-model

- v-model:双向数据绑定指令=> JS变量变化主动渲染页面，**页面数据变化**更新JS变量
- 基于 **响应式原理** 和元素 **事件监听** 实现响应式功能



- 取值：随表单控件类型不同而不同。
- 限制：**仅限于HTML表单可输入或者可选择元素**，`<input>`、`<select>`、`<textarea>`
- 语法：`<input type="text" v-model="name">`,name变量为vue实例数据仓库中可以双向修改的变量
- 修饰符：
    - `.lazy` 取代 `input` 监听 `change` 事件
    - `.number` 输入字符串转为有效的数字
    - `.trim` 输入首尾空格过滤

```
<script>
    function getSelectValue() {
        var selDom = document.getElementById("selectDom");
        console.log(selDom.value); // value属性为 select 标签的DOM属性，描述的是被选中的option的值(value,textContent)
        console.log(selDom.options)// 获取当前 select标签所包含的所有 option标签
        console.log(selDom.selectedIndex)// 获取当前 被选中的 option 的下标
        // 通过修改  selectedIndex 可以直接切换option选中状态
        selDom.selectedIndex = 4;

    }
    function selectIndex() {
        var selDom = document.getElementById("selectDom");
        // 通过修改  selectedIndex 可以直接切换option选中状态
        // selDom.selectedIndex = 4;
        // 通过对 select标签的 value属性赋值，完成自动匹配选择
        selDom.value = "OPT3";
    }
</script>
<body>
    <select id="selectDom" onchange="getSelectValue()">
        <optgroup label="组别A">
            <option value="OPT1">opt1</option>
            <option>opt2</option>
        </optgroup>
        <optgroup label="组别B">
            <option value="OPT3">opt3</option>
            <option value="OPT4">opt4</option>
            <option value="OPT5">opt5</option>
        </optgroup>
    </select>
    <input type="button" value="选中下标4" onclick="selectIndex()">
    <hr>
    <div id="app">
        <p>msg:{{ msg }}</p>
        <!--
            1、单行文本框 => 文本框，颜色选择器，时间选择器，数值文本框……
        -->
        <input type="text" v-model="msg"><br>
        <!-- v-mode 等同于 v-bind 和 v-on 的统一定义 -->
        <input type="text" :value="msg" @input=" msg = $event.target.value "><br>
        <p :style=" 'color:'+colorStr+';' ">colorStr: {{ colorStr }}</p>
        <input type="color" v-model="colorStr"><br>
        <input type="color" :value="colorStr" @input=" colorStr= $event.target.value  ">
        <hr>
        <!--
            2、多行文本域
                v-model 对于 多行文本域 操作的是 多行文本域的DOM对象的 value 属性
         -->
        <pre>{{ textMsg }}</pre>
        <textarea cols="10" rows="4" v-model="textMsg"></textarea>
        <textarea cols="10" rows="4" :value="textMsg" @input=" textMsg = $event.target.value "></textarea>
        <hr>
        <!--
            3、单选按钮
```

```
                HTML 单选按钮以 name属性分组，以checked属性描述选中，以 value描述选中值
                v-model 通过调用 单选按钮的 value 完成单选按钮取值操作，通过单选按钮定义的value和关联变量完成 checked 比较操作
        -->
        男：<input type="radio" value="m" :checked=" radioStr == 'm' " @change=" radioStr=$event.target.value ">
        女：<input type="radio" value="f" :checked=" radioStr == 'f' " @change=" radioStr=$event.target.value ">
        未知：<input type="radio" value="unkown" :checked=" radioStr == 'unkown' "
            @change=" radioStr=$event.target.value ">
        <br>
        男：<input type="radio" value="m" v-model="radioStr">
        女：<input type="radio" value="f" v-model="radioStr">
        未知：<input type="radio" value="unkown" v-model="radioStr">
        <br>
        <!--
            在vue的v-model 绑定单选按钮后，如果标签未提供 value属性值，vue会默认为value取值 null
        -->
        男：<input type="radio" v-model="radioFlag">
        女：<input type="radio" v-model="radioFlag">
        未知：<input type="radio" value="aa" v-model="radioFlag">
        <hr>
        <!--
            4、多选框
                HTML 复选按钮以 name属性分组，以checked属性描述选中，以 value描述选中值

            ** v-model对于复选按钮会根据绑定变量类型 切换操作方式
                - 非数组
                    + v-model 不再关心 标签value属性，此时 vue会以 true描述选中值，false描述未选中值
                    + checked 直接取 绑定的变量
                - 数组
                    + v-model 判断被选中的复选框是否存在value属性值
                        > 如果存在将该属性值写入绑定的数组中
                        > 如果不存在会直接向数组中写入 null
                    + checked 判断每个复选按钮在 绑定的数组中 是否存在 相同 取值
            ** v-model 在复选框绑定取值时，如果绑定的变量为数组类型，数组中会以标签选中先后添加数据
        -->
        <input type="checkbox" value="a" v-model="checkData">
        <input type="checkbox" value="b" v-model="checkData">
        <!-- <input type="checkbox" v-model="checkData">
        <input type="checkbox" v-model="checkData"> -->
        <input type="checkbox" value="c" v-model="checkData">
        <input type="checkbox" value="d" v-model="checkData">
        <hr>
        <!--
            ** 复选框的 v-model操作时的辅助属性(复选按钮的单一用途上)
                - true-value 复选框选中是时的值
                - false-value 复选框未选中时的值
         -->
        <input type="checkbox" true-value="选中" false-value="未选中" v-model="info">
        <hr>
        <!--
```

```
            5、下拉列表
                html 中的下拉列表 所具有的特殊性质
                    select 只能接收 option、optgroup 标签作为渲染依据
                    optgroup 标签不参与 select 选项操作
                    option 只能定义 取值文本，不能定义相关标签字符
                    页面构成时，通过option boolean 类型属性 selected 描述 选中项
                    如果构成option没有存在 selected属性的标签，默认选中第一个option
                JS 中下拉列表对象 ==> 参考当页前面的代码

            v-model对于 下拉列表具有下述操作行为要求
                + v-model 只能被绑定在 select 标签上
                    - v-model 的拆分 ==> select DOM v-bind:value  v-on:change 获取select的value值

        -->
        <select v-model="selectData">
            <optgroup label="组别A">
                <option value="OPT1">opt1</option>
                <option>opt2</option>
            </optgroup>
            <optgroup label="组别B">
                <option value="OPT3">opt3</option>
                <option value="OPT4">opt4</option>
                <option value="OPT5">opt5</option>
            </optgroup>
        </select>
        <hr>
        <select :value="selectData" @change=" selectData=$event.target.value " >
            <optgroup label="组别A">
                <option value="OPT1">opt1</option>
                <option>opt2</option>
            </optgroup>
            <optgroup label="组别B">
                <option value="OPT3">opt3</option>
                <option value="OPT4">opt4</option>
                <option value="OPT5">opt5</option>
            </optgroup>
        </select>
    </div>
</body>
<script>
    new Vue({
        el: "#app",
        // data 数据仓库中定义的数据变量具有 可读可写特性
        data: {
            msg: "字符串",
            colorStr: "#000000",
            textMsg: "",
            radioFlag: true,
            radioStr: "unkown",
            // checkData:""
            // checkData:["a","b"]
            checkData: [],
            info: "选中",
            selectData: "OPT1"
        }
    })
</script>
```

```
<body>
    <!--
        - .lazy 取代 input 监听 change 事件
        - .number 输入字符串转为有效的数字
            + 如果输入的值不能转换为 数值，会直接获取 string 文本
            + 如果输入的值可以转为为 数值，会转为数值后 赋值给变量
            + 如果输入的值是开头是数值，中存在非数值，会调用 Number.parseFloat 完成逐位转换

        - .trim  输入首尾空格过滤 （等效于 JS 中的 字符串 trim 方法）

        v-model 会对输入框文本完成 两个事件的绑定，input change
            + 默认 v-model使用 input
            + 需要切换input事件为change事件  ==> .lazy
     -->
    <div id="app">
        <p>{{ msg }}</p>
        <input type="text" v-model.lazy="msg"><br>
        <input type="text" :value="msg" @input=" msg = $event.target.value "><br>
        <hr>
        <input type="text" v-model.number="num">
        <hr>
        <input type="text"  v-model="msg">
        <input type="text"  v-model.trim="msg">
    </div>
</body>
<script>
    new Vue({
        el: "#app",
        data:{
            msg:"msg",
            num:null
        }
    })
</script>
```

6、数据控制

- 为开发提供自定义数据处理接口，通过开发者自定义代码完成相对复杂的数据处理和监控
- 数据控制的两大特点：数据包装、数据监控

6.1、计算属性 (computed)

- **数据包装处理 + 数据变量的监控**
- 构建方式：

```
new Vue({
    data:{},
    // 属于示例的一个特殊数据仓库
    computed:{
        // 构建计算属性
    }
    methods:{}
})
```

- 类型：`computed:{ key:value }`
  - key：取值类型string，用于 **定义计算属性变量名称** ，**计算属性具有 vue普通数据仓库的变量功能，同时具有vue方法仓库中this特性**
  - value：定义计算属性的相关取值
    - 取值为 Function 时，提供计算属性取值功能，此时该计算属性为**只读属性**
    - 取值为 Object { get:Function,set:Function } 时，提供计算属性取值和修改功能
- 应用场景：应用于 **页面中相对复杂的数据输出表达式**
- 功能：用于控制数据在页面输出前，对数据进行包装处理
- 特性：
  - **计算属性 在对数据进行处理包装时，需要依赖一个当前Vue对象中普通属性**
  - **计算属性的结果 ，会随着依赖的普通属性的变换发生变换（重新调用方法）**
  - **计算属性的变量名称不能和 vue实例中其它数据仓库的名称一样**

```
<body>
    <div id="app">
        <div>
            <label for="">name:</label>
            <input type="text" v-model="name">
        </div>
        <div>
            <label for="">age:</label>
            <input type="number" v-model.number="age">
        </div>
        <div>
            <label for="">sex:</label>
            <input type="text" v-model="sex">
        </div>
        <div>
            <label for="">email:</label>
            <input type="text" v-model="email">
        </div>
        <input type="button" value="提交注册"
               :disabled=" name==''||age==''||sex==''||email=='' ">
        <hr>
        <input type="button" value="提交注册" :disabled=" checkData ">
        <hr>
        <p>调用计算属性：{{ checkDataTemp }}</p>
        <hr>
        <p>getMsg:{{ getMsg }}</p>
        <input type="text" v-model="getMsg">
        <input type="button" value="修改getMsg" @click=" getMsg = '新值' ">
    </div>
</body>
<script>
    new Vue({
        el: "#app",
        data: {
            name: "1",
            age: "2",
            sex: "3",
            email: "4",
            msg:"测试字符串"
        },
        computed: {
```

```
            // name:function(){},
            // 计算属性变量在页面中使用时，会直接调用该计算属性的执行方法
            // 计算属性的处理方法如果需要实现功能，必须为方法提供返回值
            checkDataTemp:function(){
                console.log("checkData被执行",this);

                //  定义方法执行逻辑

                return "测试字符串";
            },
            checkData:function(){
                // 完成变量 name age sex email 值判断
                console.log("checkData计算属性方法被执行");
                // console.log(this.name,this.sex,this.age,this.email);
                // let flag = true;
                // if(this.name.length<3||this.name.length>6){

                // }
                var nameflag = this.name.length<3||this.name.length>6;
                var ageflag = this.age<0;
                var sexflag = this.sex=="";
                var emailReg = /\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*/;
                var emailflag = !emailReg.test(this.email);
                // return boolean;
                // false 按钮可点，true 按钮不可点
                return nameflag||ageflag||sexflag||emailflag;
                // return
this.name==""||this.age==""||this.sex==""||this.email=="";
            },
            // getMsg(){
            //     return "计算属性getMsg";
            // }
            getMsg:{
                // 当计算属性取值为Object时，必须遵循vue计算属性对象定义语法
                // get 属性对应的方法，是为当前计算属性提供 取值的功能定义
                get:function(){
                    // return "getMsg计算属性";
                    return this.msg;
                },
                // set 属性对应的方法，是为当前计算属性提供 赋值的功能定义
                //      set 方法构建时。vue对该方法默认提供一个形参，
                //                      该形参是当前方法被调用时的修改值
                set:function(nv){
                    // console.log(nv);
                    this.msg = nv;
                }
            }
        },
    })
</script>
```

6.2、过滤器 (filters)

- **完成数据包装处理**
- 功能：将向页面写入的数据，经过特定的方法 进行 包装处理，将处理后的结果展示到页面中，**页面的最终结果上来说，过滤器和计算属性功能相似**

- 范围：滤器可以用在两个地方：**双花括号插值和 v-bind 表达式** (后者从 2.1.0+ 开始支持)
- 语法：在页面中通过管道符`|`，连接待处理变量和过滤器方法

```
<!-- 在双花括号中 -->
{{ 待处理变量 | 过滤器方法名 }}
{{ 待处理变量 | 过滤器方法名() }}

<!-- 在 `v-bind` 中 -->
<div v-bind:id="待处理变量 | 过滤器方法名"></div>
<div v-bind:id="待处理变量 | 过滤器方法名()"></div>
```

1、局部过滤器

- **仅限于 定义过滤器时 所参考的 vue实例对象的 容器中**
  - 局部过滤器的定义方式: **需要定义在一个 已经存在的 vue实例中**
  - 局部过滤器的使用范围：**在定义过滤器的 vue对象的 容器才可使用**
- 定义局部过滤器

```
new Vue({
    filters:{
        // key 描述过滤器名称
        // value 描述过滤器 执行方法
        testFilter:function(){
            return "";  //过滤器而言，因为需要将结果展示在页面中，所有方法必须存在返回值
        }
    }
})
```

2、全局过滤器

- **使用 Vue 装载 新的 过滤器方法，提供给当前项目中所有的vue实例使用**
  - 全局过滤器的定义方式: **通过为 Vue 构造方法增加 新的方法 完成 过滤器的定义（全局配置）**
  - 全局过滤器的使用范围：**所有的vue实例的容器中都可以使用**
- 全局过滤器定义
  - 全局定义 需要依赖于 Vue构造器对象
  - **全局过滤器 一定要在 vue实例构建之前定义**
  - 语法：**Vue.filter(id, [definition])**
    - id == name : 定义过滤器名称(过滤器的方法名) 取值为 string，具有**唯一性**
    - definition ： 过滤器的执行方法（**和局部过滤的方法特性一样**），是一个匿名方法
  - **如果全局过滤器名称和局部过滤器名称相同**
    - **过滤器可以共存，但定义局部过滤器的实例中无法使用全局过滤器，因为局部过滤器优先级高于全局过滤器**
- 定义全局过滤器

```
Vue.filter("formatDate",function(date){
    console.log("全局过滤器: ",date);
    return "全局过滤器";
})
```

```
<body>
```

```
    <div id="app">
        <div class="film-source" v-for="source in sources">
            <!-- <span
                class="star-img"
                v-for="c in 5"
                :class=" Math.ceil( source ) >= c ? 'open':'close' "
            ></span> -->
            <span
                class="star-img"
                v-for="c in 5"
                :class=" source | starClass(c) "
            ></span>
            <span class="star-source">{{ source }}</span>
        </div>
        <hr>
        <p>{{ source | testFilter }}</p>
        <p>{{ 3.7 | testFilter( 'a' ) }}</p>
        <p>{{ 3.8 | testFilter( 'a','b' ) }}</p>
        <p>{{ 3.9 | testFilter( 'a','b','c' ) }}</p>
        <hr>
        <p>当前时间：{{ day | formatData() }}</p>
    </div>
    <div id="root">
        <!-- <p>{{ "4.1" | testFilter }}</p> -->
        <p>当前时间：{{ day | formatData() }}</p>
    </div>
</body>
<script>
    // 全局过滤器定义必须在vue实例构建之前
    // Vue.filter(id, [definition]) // 单一过滤器定义
    //      id : 定义过滤器名称 ==> 过滤器名称必须唯一
    //      definition :  当前过滤器的执行方法

    // 过滤器工具方法
    //      1、严谨性
    Vue.filter("formatData",function(target){
        // 全局过滤器的执行方法 所需遵循特性和局部过滤器方法一致
        // console.log("全局过滤器formatData");
        if( !(target instanceof Date) ){
            return target;
        }
        var y = target.getFullYear();
        var m = target.getMonth()+1;
        var d = target.getDate();

        var h = target.getHours();
        var mm = target.getMinutes();
        var s = target.getSeconds();
        return `${y}年${m}月${d}日 ${h}:${mm}:${s}`;
    })


    new Vue({
        el:"#app",
        data:{
            source:3.6,
            sources:[3.6,2.4,4.5],
            day:new Date()
```

```
        },
        // computed: {
        //     getClass(){
        //         return  Math.ceil( this.source ) >= c ? 'open':'close'
        //     }
        // },
        // 为当前vue示例定义过滤器方法==>普通方法
        filters:{
            // key:value
            // key 过滤器方法名称（id=唯一性）
            // value 过滤方法调用的执行函数
            //       vue中构建的所有过滤器方法必然存在第一个形参 target
            //              target 为 | 管道符左侧的待处理数据
            //                  过滤器方法形参定义，第一个形参始终为 | 管道符号左侧的待处理数据
            //                  从第二个形参开始，参数值来自于 | 管道符号右侧，过滤方法执行时依次传入的额外参数
            //              过滤器方法必须存在 处理后的数据返回值
            testFilter:function(target,a,b,c){
                // arguments 为JS中所有方法形参的集合
                console.log(arguments);
                console.log("过滤器方法",target,a,b,c);
                return target + "分";
            },
            starClass(target,c){
                // Math.ceil( source ) >= c ? 'open':'close'
                return Math.ceil( target ) >= c ? 'open':'close'
                // return "open"
            }
        }
    });
    new Vue({
        el:"#root",
        data:{
            day:new Date()
        },
        // filters:{
        //     formatData(){
        //         console.log("局部过滤器");
        //     }
        // }
    })

</script>
```

6.3、监视器 (watch)

- **数据监控调用**
- 功能：构建一个对Vue实例中数据仓库中变量（data，computed）的监控方法，**实现当数据变化时执行额外扩展方法**
- 组件内构建方式：

```
var vm = new Vue({
    watch: {
        key:value
    },
})
```

- key (string) ：被监视的数据变量名称 **或对象路径表示方式**
  - **！注意！：对象路径表示形式只能用于Vue的监视器定义时**
- value (Function|Object) : 定义监视器执行方式
  - 取值 Function : 定义基础的数据监控方法
  - 取值 Object : 定义可扩展数据监控配置

```
{
    handler: Function 定义监控方法
    deep: Boolean 是否开启深度监视
    immediate: Boolean 是否开启初始化触发
}
```

```
<body>
    <div id="app">
        <!-- <input type="text" v-model="wd" @input="sendBidu()"> -->
        <input type="text" v-model="wd">
        <input type="button" value="百度一下" @click="sendBidu()">
        <input type="button" value="添加结果" @click=" wd = 'html' ">
        <hr>
        <!-- <p>{{ result }}</p> -->
        <ul>
            <li v-for="(item, index) in result" @click=" wd = item ">{{ item }}
</li>
        </ul>
    </div>
</body>
<script>
    new Vue({
        el:"#app",
        data:{
            wd:"html",
            result:[]
        },
        watch: {
            // 监控数据变量的变化操作
            // key:value
            //   key 监控器名称==> 描述当前vue示例中的数据仓库中 哪个变量需要监控
            //   value 描述监控变量变化后的执行方法
            //         方法内部的this 依然指向与 当前vue实例对象
            //         方法默认具有 两个 形参值，一个为修改前的数据，一个修改后的数据
            // wd:function(newValue,oldValue){
            //     // console.log("wd发生变化时",this.wd)
            //     console.log(newValue,oldValue)
            //     this.sendBidu();
            // }

            // 监视器定义时可以取值 Object
```

```
            //   Object:{ handler:定义回调函数 ,immediate: Boolean 是否开启初始化触发
  }
            wd:{
                handler:function(){
                    // handler 等效于 监视器直接定义方法的功能
                    this.sendBidu();
                },
                immediate:true
            }
        },
        methods: {
            sendBidu(){
                // console.log(this.wd);
                $.ajax({
                    url:"https://www.baidu.com/su",
                    methods:"get",
                    dataType:"jsonp",
                    data:{
                        wd:this.wd
                    },
                    jsonp:"cb",
                    success:(data)=>{
                        console.log(data,this);
                        this.result = data.s;
                    }
                })
            }
        },
    })
</script>
```

```
<body>
    <div id="app">
        <!-- 引用类型数据变量的监控 -->
        <pre>{{ user }}</pre>
        <input type="text" v-model="user.name">
        <input type="text" :value="user.name" @input="
user.name=$event.target.value ">
        <hr>
        <input type="text" v-model="user.age">
        <input type="button" value="resetUser" @click="resetUser()">
        <pre>{{ arr }}</pre>
        <input type="button" value="setArr" @click=" arr[1]='aa' ">
        <input type="button" value="push" @click=" pushItem() ">
    </div>
</body>
<script>
    new Vue({
        el:"#app",
        data:{
            user:{
                name:"tom",
                age:23
            },
            arr:[1,2,3,4]
        },
        watch: {
```

```
            // 监视器默认只监控 定义的引用变量，不会监控引用变量中的属性变化
            // user(nv,ov){
            //     console.log("user变量变化");
            // },
            user:{
                // 无差别监视 == 当开启深度监视时，只要对象中的任意属性发生变化，都会执行监控方法
                handler(nv,ov){
                    // 当监控的数据为应用类型变量时，nv和ov都是修改后的变量结果
                    console.log("user变量变化",nv,ov);
                },
                // deep: Boolean 是否开启深度监视
                deep:true // 当前的引用类型监控器不仅监控该变量地址的变化，同时监控属性的变化
            },
            // 必须 只监控 一个对象中的一个固定属性
            //   key 取 对象属性路径的字符串形式，描述只要监控的对象属性
            "user.name":function(){
                console.log("user.name的监视器")
            },
            arr(){
                console.log("arr变量变化");
            }
        },
        methods: {
            resetUser(){
                this.user = {
                    name:"jack"
                }
            },
            pushItem(){
                /*
                    调用变异方法
                        变异方法是vue重写的数组方法：保留原方法功能的同时，提供监控调用
                            push()，pop()，shift()，unshift()，splice()，sort()，reverse()
                    由于 JavaScript 的限制，Vue 不能检测以下数组的变动：
                        当你利用索引直接设置一个数组项时，例如：vm.items[indexOfItem] = newValue
                        当你修改数组的长度时，例如：vm.items.length = newLength
                */
                this.arr.push("aaa");
                console.log(this);
            }
        },
    })
</script>
```

- 组件外构建

```
var vm = new Vue({……});
var unwatch = vm.$watch( expOrFn, callback, [options] )
```

  - expOrFn（string|Function）被监视的数据变量名称、**对象路径表示方式**、**多变量配置方法**
    - 取值为string ：被监视的数据变量名称**或对象路径表示方式**
    - 取值为Function：被监视的**多变量配置方法**
  - callback（Function）：定义的监控处理方法
  - options （Object）：定义监控扩展功能

```
{
    deep: Boolean 是否开启深度监视
    immediate: Boolean 是否开启初始化触发
}
```

  - 返回值 unwatch （Function）：返回一个用于关闭销毁销毁当前监控的方法

```
<head>
    <script src="./js/vue.js"></script>
</head>
<body>
    <div id="app">
        <h4>msg:{{ msg }}</h4>
        <input type="text" v-model="msg">
        <hr>
        <h4>info:{{ info }}</h4>
        <input type="text" v-model="info">
        <hr>
        <p>{{ user }}</p>
        <input type="text" v-model="user.name">
        <hr>
        <p>{{ a+b }}</p>
        <input type="text" v-model.number="a">
        <input type="text" v-model.number="b">

    </div>
</body>
<script>
    var vm = new Vue({
```

```
        el: "#app",
        data: {
            msg: "msg变量",
            info: "info变量",
            a: 1,
            b: 2,
            user: {
                name: "tom"
            }
        },
        // 组件内的监控器无法关闭
        watch: {
            msg(nv, ov) {
                console.log(nv, ov);
            },
            // a(){
            //     console.log("监视方法----");
            // },
            // b(){
            //     console.log("监视方法----");
            // }
        },
    });

    // vm.$watch(expOrFn,callback [,options])
    vm.$watch("info", function (nv, ov) {
        console.log(nv, ov);
    })
    vm.$watch("user", function () {
        console.log("user");
    }, {
            deep: true
        })
    // $watch 可以返回一个 当前 监控属性的监控关闭方法
    var unwatch = vm.$watch("user.name", function () {
        console.log("user.name");
        unwatch();
    })
    // unwatch 为一个 function ,该方法调用会直接关闭对应的监控器watch
    console.log(unwatch)


    // 多变量共同监控
    vm.$watch(function () {
        // console.log(this); // 就是当前实例的指向
        // return this.a + this.b; // 会影响 监控 回调的返回结果
        // return { a:this.a, b:this.b }
        return [ this.a,this.b ]
    }, function (nv, ov) {
        console.log("监视方法->", nv, ov);
    })
</script>
```

6.4、计算属性、过滤器、监视器

- 计算属性
    - 依赖于Vue实例,只能在实例中定义使用

    - 调用时不能接收额外参数，必须依赖vue实例中的 一个或 多个固定数据变量
    - 计算属性默认是只读属性，但是可以在定义时使用对象模式，开启可读写模式
    - **计算属性会对结果进行缓存操作**
        1. 如果依赖变量没有变化，计算属性方法不被触发，直接从缓存中进行读取；
        2. 如果依赖变量发生变化，会重新执行一次方法
    - 计算属性是被作为一个数据仓库变量方式使用
- 过滤器
    - 可以根据需要选择全局过滤器或局部过滤器
    - 调用时可以接收多个参数，其中包含待处理数据，因此可以不依赖于固定vue实例
    - 过滤器只能完成对于被过滤数据的读取操作，无法进行设置操作
    - **过滤不具有缓存特性，页面中定义调用一次必然会重新执行一次**
    - 过滤是被作为一个特殊的处理方法使用
- 监视器
    - 依赖于Vue实例的固定变量，可以在实例中定义，也可以在全局中通过实例对象进行定义
    - 监视器不能被调用，只能由Vue检测变量变化自动执行，方法默认自带两个参数 oldValue和newValue
    - 监视可以完成对固定变量的监控操作
    - 监视器被作为Vue功能的扩展接口使用

7、页面模板和render函数

- Vue对象构建时需要为对象指定页面构成，该构成称之为模板
- 在基本构成vue对象时通过 el 指定**构成的页面模板**，**同时指定页面位置**

1、模板属性 (template)

- 该属性用于提供组件化开发支持
- Vue实例添加 `template` 属性，可以独立定义页面构成模板
- template 构成的模板 **最终会替换到 el 的指向位置**
- 示例:

```
var vm = new Vue({
    el: '#app',
    template:StringDOM | StringEl
});
```

    - StringDOM：HTML标签的字符串定义方式
    - StringEl：HTML元素选择器

2、模板渲染函数（render）

- 该属性用于提供模块化开发支持
- Vue实例添加 `render` 属性函数，可以独立定义JS模板构成函数
- render 构成的模板 **具有最高优先权** ， **最终会替换到 el 的指向位置**，且template属性失效
- 示例:

```
var vm = new Vue({
    el: '#app',
    data:{
        title:"标题"
    },
    render: function (createElement) {
        return createElement('h4', 'render'+this.title);
    }
});
```

- render 属性取值为一个固定函数`function (createElement) {}`，该函数返回构成的模板
- createElement 为渲染函数的固定参数，**该参数可用于以方法方式构建页面模板**

8、实例属性和方法

- 实例属性和实例方法就是项目运行过程中
  - **通过Vue构造函数创建的对象（实例）**
  - 通过实例可以**直接访问的属性和方法**==>是vue对象的普通属性和方法
  - 在满足上述两个条件的情况下，访问的属性和方法为Vue环境提供特殊功能或特殊值的参数==>实例属性和方法
- Vue中对部分特殊的属性和功能方法进行特殊指代定义，用于提供独立的执行和获取方式
- Vue的实例属性和方法以统一规范以`$`开头（**是定义在 Vue原型上的属性和方法**）
- 相关实例属性和方法
  - vm.$el：描述当前Vue实例使用的根 DOM 元素。
  - vm.$data：描述当前Vue实例**观察**的数据对象。
  - vm.$options：构建当前 Vue 实例的初始化选项。通过new Vue 传入的参数，构建初始化配置项
  - vm.$refs：返回一个对象，记录当前Vue实例模板中，**定义了ref属性**的所有 DOM 元素。
    - 提供一个DOM元素获取的接口对象，为开发者提供简化的DOM元素调用方法

```
<head>
    <link rel="stylesheet" href="./css/swiper.css">
    <script src="./js/swiper.js"></script>
    <script src="./js/vue.js"></script>
</head>
<body>
    <div id="app">
        <h1 ref="h1Dom">dom 对象</h1>
        <input type="button" value="获取rootDom" @click="printRootDom()">
        <input type="button" value="获取Data" @click="printData()">
        <input type="button" value="获取Options" @click="printOptions()">
        <hr>
        <input type="button" value="初始化swiper实例" @click="initSwiper()">
        <div class="swiper-container" ref="loop">
            <div class="swiper-wrapper">
                <div class="swiper-slide">slider1</div>
                <div class="swiper-slide">slider2</div>
                <div class="swiper-slide">slider3</div>
            </div>
        </div>
        <hr>
```

```
    </div>
    <hr>
    <input type="button" value="打印vue实例容器对象" onclick="printVueDom()">
    <input type="button" value="打印vue实例数据观察者对象"
onclick="printVueData()">
    <hr>
</body>
<script>
    // var mySwiper = new Swiper('.swiper-container', {
    //     autoplay: true,//可选选项，自动滑动
    // })
    var vm = new Vue({
        el: "#app",
        data: {
            msg: "msg变量",
            info: "info变量"
        },
        methods: {
            printRootDom() {
                // 当前实例对应的DOM容器对象
                console.log(this.$el);
            },
            printData() {
                // this.msg
                // 当前实例对应的被劫持观察的 数据仓库对象
                console.log(this.$data);
            },
            printOptions() {
                // 通过开发者传入的配置项，构建出来的用于创建vue实例的整合结果
                console.log(this.$options);
            },
            initSwiper() {
                // 获取当前vue实例的DOM元素对象
                // 只能获取当前vue实例对应的容器中具有 ref 属性的DOM元素
                // 会以 ref属性取值的名字作为key，对应的DOM作为value存放在$refs中
                console.log(this.$refs);
                console.log(this.$refs.h1Dom);

                new Swiper(this.$refs.loop, {
                    autoplay: true,//可选选项，自动滑动
                })
            }
        },
    });

    function printVueDom() {
        console.log(vm.$el);
    }
    function printVueData() {
        console.log(vm.$data);
    }
</script>
```

- vm.$mount：手动挂在Vue实例==> 补充 在构建vue实例时 未提供 el 属性配置时使用

```
<head>
    <script src="./js/vue.js"></script>
```

```
    <style>
        .box {
            position: absolute;
            top: 0px;
            bottom: 0px;
            left: 0px;
            right: 0px;
            background-color: rgba(0, 0, 0, 0.3);
            /* width: 100%;
            height: 200px; */
        }

        .content {
            margin: 0 auto;
            width: 400px;
            background-color: white;
        }
    </style>
</head>
<body>
    <hr>
    <input type="button" value="模拟弹出效果" onclick="showBox()">
    <input type="button" value="创建容器DOM" onclick="initBox()">
    <input type="button" value="将容器写入页面" onclick="appendDom()">

    <hr>
    <div id="app"></div>
    <hr>
    <template id="box">
        <div class="box" id="app">
            <div class="content">
                <h1>msg:{{ msg }}</h1>
                <h1>info:{{ info }}</h1>
                <input type="text" v-model="info">
            </div>
        </div>
    </template>
</body>
<script>
    var vm = new Vue({
        // el: "#app",
        template:"#box",
        data: {
            msg: "msg变量",
            info: "info变量"
        }
    });
    console.log(vm);
    console.log(vm.$el);

    function showBox(){
        // vm.$mount([元素选择器]);//手动指定EL的容器指向
        // 1、创建vue实例对应的容器DOM
        // 2、将容器DOM根据 元素选择器 写入页面
        vm.$mount("#app");
        console.log(vm.$el);
    }
    function initBox(){
```

```
        // 1、创建vue实例对应的容器DOM
        vm.$mount();
        console.log(vm.$el);
    }
    function appendDom(){
        document.body.appendChild(vm.$el);
    }
</script>
```

- vm.$destroy：手动销毁Vue实例==> 只会销毁vue的实例对象，不会销毁与其关联的页面容器

```
<head>
    <script src="./js/vue.js"></script>
</head>
<body>
    <div class="box" id="app">
        <div class="content">
            <h1>msg:{{ msg }}</h1>
            <h1>info:{{ info }}</h1>
            <input type="text" v-model="info">
        </div>
    </div>
    <hr>
    <input type="button" value="销毁vue实例" onclick="destVue()">
</body>
<script>
    var vm = new Vue({
        el: "#app",
        data: {
            msg: "msg变量",
            info: "info变量"
        }
    });
    function destVue(){
        vm.$destroy(); // 销毁vue实例的
    }
</script>
```

- vm.$nextTick：将执行函数体延迟到页面DOM更新完成后执行

```
<head>
    <link rel="stylesheet" href="./css/swiper.css">
    <script src="./js/swiper.js"></script>
    <script src="./js/vue.js"></script>
</head>
<body>
    <div id="app">
        <h1 ref="h1Dom">{{ msg }}</h1>
        <input type="button" value="赋值msg" @click="setMsg()">
        <hr>
        <div class="swiper-container" ref="loop">
            <div class="swiper-wrapper">
                <div class="swiper-slide"
                     v-for="(item, index) in loops"
                     :key="index"
```

```
                >
                    slider:{{ item }}
                </div>
            </div>
        </div>
        <input type="button" value="initSwiper()" @click="initSwiper()">
        <hr>
    </div>
</body>
<script>
    var vm = new Vue({
        el: "#app",
        data: {
            msg: "msg变量",
            loops:[]
        },
        methods: {
            setMsg() {
                console.log("赋值前：", this.$refs.h1Dom.innerHTML);
                // 下述代码不是直接立即生效的代码，需要一定执行时间
                this.msg = "新值"; // 异步过程
                // this.$nextTick(callback)  callback 取值function，异步的回调函数
                this.$nextTick(() => {
                    console.log("nextTick:", this.$refs.h1Dom.innerHTML);
                });
                console.log("赋值后：", this.$refs.h1Dom.innerHTML);
            },
            initSwiper(){
                this.loops = [1,2,3,4,5];
                this.$nextTick(()=>{
                    new Swiper(this.$refs.loop, {
                        autoplay: true,//可选选项，自动滑动
                    })
                })
            }
        },
    });
</script>
```

- vm.$watch：构建一个对Vue实例中数据仓库中变量（data，computed）的监控方法
- vm.$set：等同于 `Vue.set` ，手动为实例中没有数据监听的变量添加监视功能
  - 手动添加数据劫持操作（ `Object.defineProperty(obj, prop, descriptor)` ）
  - **$set 不能完成对 vue的 data（vm.$data）添加新属性劫持**
- vm.$delete：等同于 `Vue.delete` ，手动为实例中的变量删除监视功能
  - 手动删除数据劫持操作，包含数据的删除
  - **$delete不能完成对 vue的 data（vm.$data）属性劫持的删除操作**

```
<head>
    <script src="./js/vue.js"></script>
</head>
<body>
    <div id="app">
        <h1>{{ result.name }}</h1>
        <h1>{{ result.info }}</h1>
```

```
        <!-- v-model指令在操作时，如果发现被绑定的属性在数据仓库中未被定义
            v-model会主动完成该属性的 劫持操作
        -->
        <input type="text" v-model="result.info">
        <ul>
            <li v-for="(item, index) in result.loops" :key="index">{{ item }}
</li>
        </ul>
        <input type="button" value="模拟异步数据加载-result"
@click="loadResultData()">
        <input type="button" value="模拟异步数据加载-loops"
@click="loadLoopData()">
        <input type="button" value="模拟异步数据加载-info"
@click="loadInfoData()">
        <hr>
        <h1>{{ testObj.message }}</h1>
        <input type="text" v-model="testObj.message">
        <input type="button" value="删除message监控" @click="removeLinst()">
        <input type="button" value="添加message属性" @click="addMessage()">
        <hr>
        <input type="button" value="为$data添加test" @click="addTest()">
    </div>
</body>
<script>
    var vm = new Vue({
        el: "#app",
        data: {
            result:{},
            testObj:{
                message:"初始化页面"
            }
        },
        methods: {
            loadResultData(){
                // 无法在vue实例运行时，再为新属性添加数据劫持操作
                // this.result.name = "测试name变量";
                // this.$set(targetObj,keyName,value)
                this.$set(this.result,"name","测试name变量");
            },
            loadLoopData(){
                // 无法在vue实例运行时，再为新属性添加数据劫持操作
                // this.result.loops = "南京,上海,苏州,镇江,扬州,合
肥".split(",");
                this.$set(this.result,"loops","南京,上海,苏州,镇江,扬州,合
肥".split(","))
            },
            loadInfoData(){
                this.result.info = "新info数据";
            },
            removeLinst(){
                // this.$delete(targetObj,key)
                this.$delete(this.testObj,"message");
            },
            addMessage(){
                this.testObj.message = "新值";
            },
            addTest(){
                this.$set(this.$data,"test","aaaa");
```

```
            }
        },
    });
</script>
```