# Vue\_01

# 目标

* 在网页中实例化vue对象
* 知道vue数据绑定原理
* 熟练使用插值表示式(模板语法)
* 掌握vue指令

# 一、Vue基础

## 1.1、介绍

官网：<https://cn.vuejs.org/>

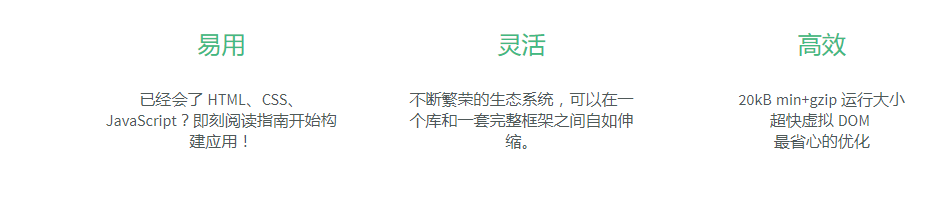
Vue.js是一套构建用户界面的渐进式框架。Vue 采用自底向上增量开发的设计。Vue 的核心库只关注视图层，它不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与单文件组件和 Vue 生态系统支持的库结合使用时，Vue 也完全能够为复杂的单页应用程序提供驱动。(spa单页面应用，所有的显示都在一个页面当中)

渐进式：一步一步，不是说你必须一次把所有的东西都用上

自底向上设计：是一种设计程序的过程和方法，就是先编写出基础程序段，然后再逐步扩大规模、补充和升级某些功能，实际上是一种自底向上构造程序的过程。



Vue从设计角度来讲，虽然能够涵盖这张图上所有的东西，但是你并不需要一上手就把所有东西全用上，都是可选的。声明式渲染和组件系统是Vue的核心库所包含内容，而路由、状态管理、构建工具都有专门解决方案。这些解决方案相互独立，你可以在核心的基础上任意选用其他的部件，不一定要全部整合在一起。



## 1.2、声明式渲染和组件化

* 声明式渲染

Vue.js 的核心是一个允许采用简洁的模板语法来声明式的将数据渲染进 DOM 的系统

# html

<div id="app">

<!-- 渲染 Hello Vue -->

{{ message }}

</div>

# js

var vm = new Vue({

el: '#app',

// 数据源

data: {

// 声明一下变量

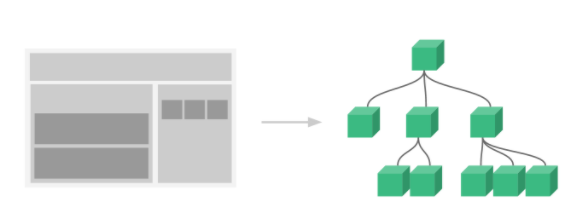
message: 'Hello Vue!'

}

})

* 组件化应用构建

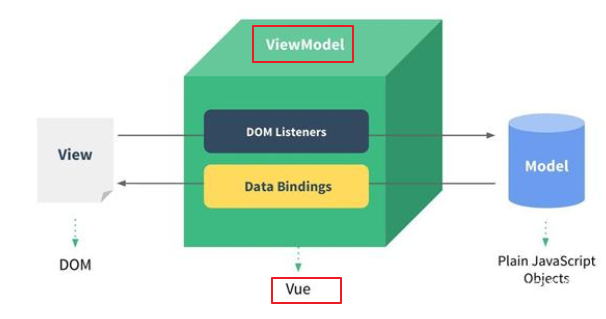
组件系统是 Vue 的另一个重要概念，因为它是一种抽象，允许我们使用小型、独立和通常可复用的组件构建大型应用。几乎任意类型的应用界面都可以抽象为一个组件树。



## 1.3、MVVM模式

MVVM是Model-View-ViewModel的简写。它本质上就是MVC 的改进版。MVVM 就是将其中的View 的状态和行为抽象化，让我们将视图 UI 和业务逻辑分开。当然这些事 ViewModel 已经帮我们做了，它可以取出 Model 的数据同时帮忙处理 View 中由于需要展示内容而涉及的业务逻辑。

vue使用MVVM响应式编程模型，避免直接操作DOM , 降低DOM操作的复杂性。



MVVM：页面输入改变数据，数据改变影响页面数据展示与渲染

M（model）：普通的javascript数据对象

V（view）：前端展示页面

VM（ViewModel）：用于双向绑定数据与页面，对于我们的课程来说，就是vue的实例

优点：

* 低耦合。视图（View）可以独立于Model变化和修改，一个ViewModel可以绑定到不同的View上，当View变化的时候Model可以不变，当Model变化的时候View也可以不变。
* 可重用性。你可以把一些视图逻辑放在一个ViewModel里面，让很多view重用这段视图逻辑。
* 独立开发。开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计。
* 可测试。界面素来是比较难于测试的，而现在测试可以针对ViewModel来写。

# 二、vue基础使用

## 2.1、传统dom操作

使用js对html页面结构中的指定的区域输出数据



## 2.2、使用vue实现

在html页面中使用好vue需要完成如下步骤即可

* 引入vue.js文件(下载)
* 定义给vue.js管理的dom元素(给div定义一个ID)
* 创建一个 Vue 的实例，并声明要渲染的数据源
* 在给定的dom元素容器内，绑定数据源中变量名称{{变量名}}
* Vue参数对象属性

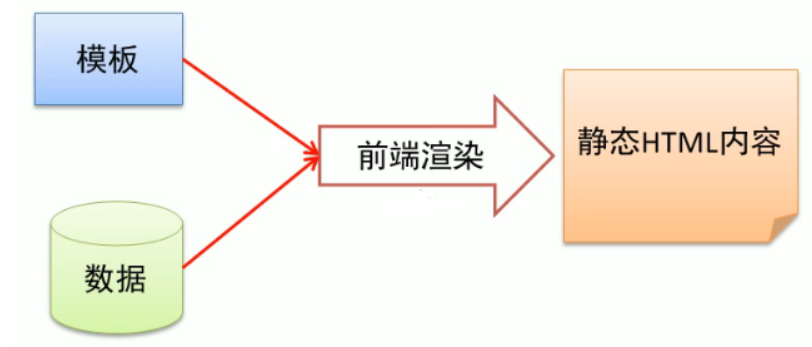
el：元素挂载的位置，值可以是CSS选择器或DOM元素

data：模型数据，值是一个对象

* 插值表达式

将数据填充到HTML标签中

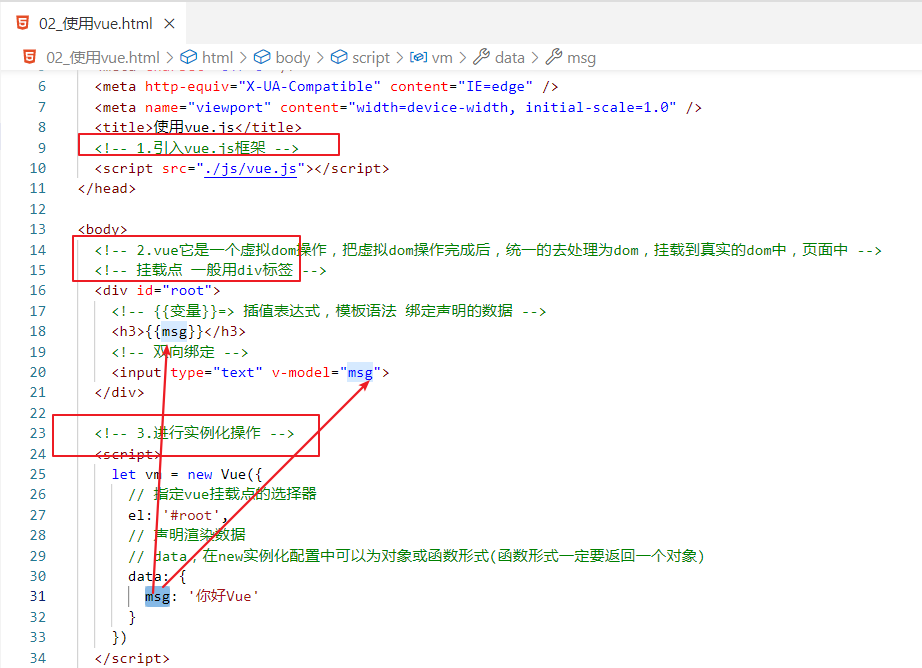
支持基本的JavaScript计算操作，例如算术运算、字符串拼接等



下载vue.js

<https://cn.vuejs.org/v2/guide/>



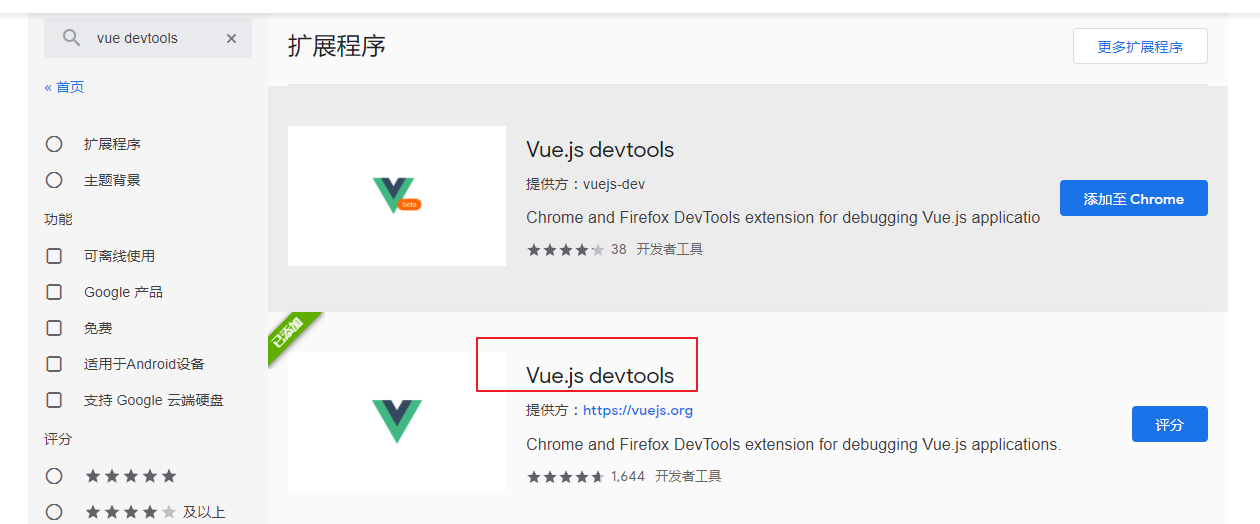


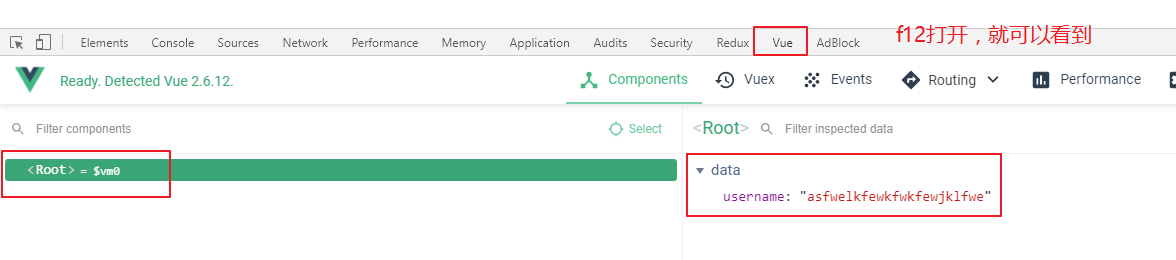
## 2.3、vue devtools工具安装

通过chrome中的谷歌插件商店安装Vue Devtools工具，此工具帮助我们进行vue数据调试所用，一定要安装。

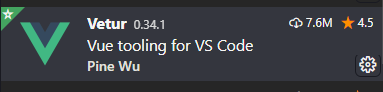
<https://chrome.google.com/webstore?utm_source=chrome-ntp-icon>

<https://chrome.zzzmh.cn/>



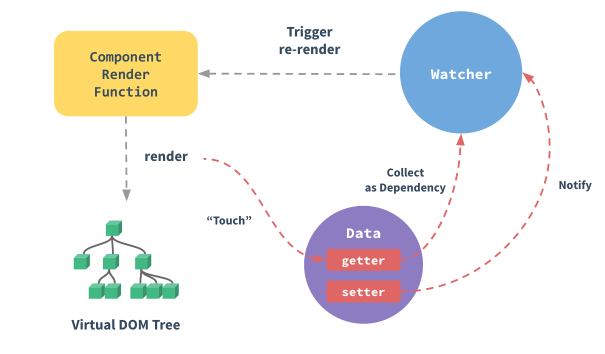


* 在vscode中安装插件



## 2.4、Vue实现数据绑定的原理

当把一个普通的JavaScript对象传给Vue实例的data选项，Vue将遍历此对象所有的属性，使用Object.defineProperty(vue2.x),vue3.x中使用了**Proxy**类把这些属性全部转为getter/setter(数据劫持)。在属性被访问和修改时通知变化。每个组件实例都有相应的 watcher 实例对象，它会在组件渲染的过程中把属性记录为依赖，之后当依赖项的 setter 被调用时，会通知 watcher 重新计算，从而致使它关联的组件得以更新。



vue实现数据响应式

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<div id="root">

<h3 v-text="title"></h3>

<hr>

<input type="text" v-model='title'>

</div>

<script>

// 数据劫持

let data = {

title: '我是一个标题',

};

// 观察数据

observe(data)

// 给input绑定事件

document.querySelector('[v-model]').addEventListener('input', function () {

let key = this.getAttribute('v-model')

data[key] = this.value.trim()

})

function observe(target) {

if (!isObject(target)) return;

for (let key in target) {

defineReactive(target, key, target[key])

}

}

// 数据劫持

function defineReactive(target, key, value) {

Object.defineProperty(target, key, {

get() {

console.log('get')

return value

},

set(v) {

if (v != value) {

value = v

console.log('set')

// 更新视图

updateView(value, key)

}

}

})

}

function updateView(value, key) {

document.querySelectorAll('[v-text]').forEach(node => {

let attrValue = node.getAttribute('v-text')

if (key === attrValue) {

if (node.nodeName === 'INPUT') {

node.value = value;

} else {

node.innerHTML = value;

}

}

})

}

function isObject(target) {

// return target !== null && typeof target === 'object' && !(target instanceof Array)

// return Object.prototype.toString.call(target) === '[object Object]'

return ({}).toString.call(target) === '[object Object]'

}

</script>

</body>

</html>

# 三、模板语法

## 3.1、插值表达式

插值表达式是vue框架提供的一种在html模板中绑定数据的方式，使用{{变量名}}方式绑定Vue实例中data中的数据变量。会将绑定的数据实时的显示出来。

# 支持写法

{{变量、js表达式、三目运算符、方法调用等}}

<div id="app">

<h3>{{name}}</h3>

<h3>{{name + '--好的'}}</h3>

<h3>{{ 1 + 1 }}</h3>

<!-- 使用函数 -->

<h3>{{title.substr(0,6)}}</h3>

<!-- 三目运算 -->

<h3>{{ age>22 ? '成年':'未成年'}}</h3>

</div>

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

<script>

new Vue({

el: '#app',

data: {

title: '我是一个标题，你们看到没有',

name: '张三',

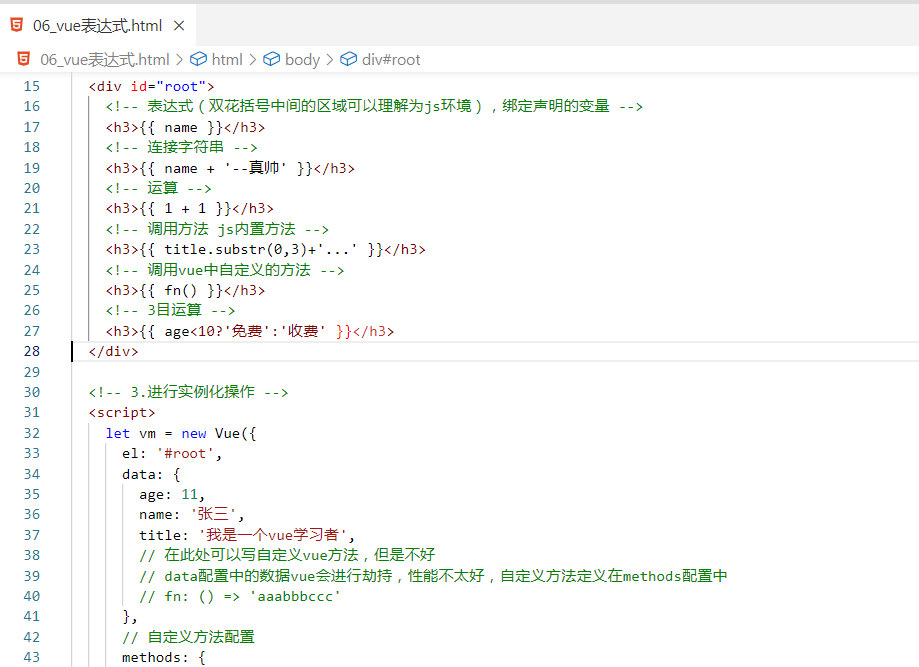
age: 20

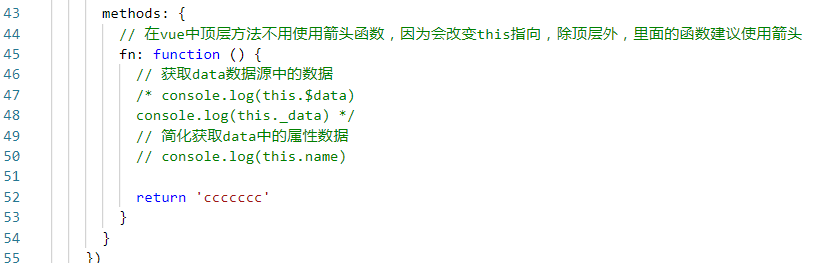
}

})

</script>

注：{{}}括起来的区域，就是一个就是js语法区域，在里面可以写部份的js语法。不能写 var a = 10; 分支语句 循环语句





## 3.2、指令

指令(Directives)就是vue给html标签提供的一些自定义属性，这样属性都是带有 **v-** 前缀的特殊属性。指令特性的值预期是单个JS表达式(v-for是例外情况)。指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于DOM。

指令作用：

* 操作dom
* 设置主题
* 权限限制
* 表单验证

v-html 解析html指令 注：尽量避免使用，容易造成危险 (XSS跨站脚本攻击)

v-text 输出文本信息

-----------------------------------------------------

<div id="app">

<!-- 转义html -->

<h3>{{html}}</h3>

<!-- 解析 html -->

<div v-html="html"></div>

<!-- 转义html -->

<div v-text="html"></div>

</div>

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

<script>

new Vue({

el: '#app',

data: {

html:'<a href="http://www.baidu.com/">百度一下</a>'

}

})

</script>



# 四、常用指令

指令扩展了html标签的功能、大部分的指令的值是js的表达式，取代了DOM操作。

## 4.1、v-cloak指令

在页面渲染时,往往因为加载数据或代码过多,页面上的模板语法表达式会出现暂时性获取不到值,而直接显示表达式，等到加载结束后，值又会正常显示出来。这个过程很短暂，给人一种闪动的感觉。

为了解决这个不好的体验,我们使用v-cloak指令，在页面没有渲染完成前，给有模板语法表达式的地方加一个隐藏样式。等加载完成后，样式也消失了。

这种情况现在很少见了，我们只能按下面的方式模拟一下。



## 4.2、v-pre指令

跳过这个元素和它的子元素的编译过程，跳过大量没有指令的节点会加快编译。

<span v-pre>不需要编译，直接可以运行</span>



## 4.3、v-once指令

只渲染元素和组件一次，之后元素和组件将失去响应式功能

<div id="app">

<h3>{{message}}</h3>

<hr>

<!-- 动态修改message值，此绑定将不会发生改变 -->

<div v-once>{{message}}</div>

</div>

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

<script>

const vm = new Vue({

el: '#app',

data: {

message: '你好世界'

}

})

</script>



## 4.4、v-if和v-show

根据表达式的布尔值(true/false)进行判断是否渲染该元素

<div id="app">

<!--

v-if是对dom元素的移除和创建

v-show是通过修改标签的display值

-->

<!-- 如果isShow的值是true ,就显示p标签 -->

<p v-if="isShow">我是p标签中的内容</p>

<p v-show="isShow">我是p标签中的内容</p>

</div>

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

<script>

const vm = new Vue({

el: '#app',

data: {

isShow: true

}

})

</script>

注：v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 v-show 较好；如果在运行时条件很少改变，则使用 v-if 较好。

# 多条件判断

v-if … v-else-if v-else



## 4.5、v-bind指令

动态地绑定一个或多个属性

<!-- v-bind绑定href属性值 -->

<a v-bind:href='url'>跳转</a>

<!-- v-bind绑定href属性值（简写形式） -->

<a :href='url'>跳转</a>

## 4.6、v-for

根据一组数组或对象的选项列表进行渲染。

v-for指令需要使用 (item,index) in 数组或对象 形式的特殊语法，同时还需要指定key值,key的作用在vue进行新旧数据比对渲染页面里，如果有key值会提升比对性能，加快渲染，key使用唯一的值来赋值。

<div id="app">

<ul>

<!-- 数组循环 -->

<li v-for="u in users" :key="u">{{ u }}</li>

</ul>

<hr>

<ul>

<!-- 对象循环 -->

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

</ul>

</div>

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

<script>

const vm = new Vue({

el: '#app',

data: {

users: [1, 2, 3],

obj: {

username: 'zhangsan',

age: 28,

gender: 'male'

}

}

})

</script>

## 4.7、v-on

绑定事件监听器（事件绑定）

<!-- 常规写法 -->

<button v-on:click="事件实现方法()"></button>

<!-- 缩写 -->

<button @click="事件实现方法()"></button>

# 绑定好事件实现方法后需要在Vue对象中的methods对象中实现对应的绑定方法

methods: {

functionName(arg1,arg2,arg3,...){

// something to do

},

....

}

* 事件处理函数传参 -- 事件对象--- event对象

<!-- 事件处理函数调用：直接写函数名 -->

<button @click="say"></button>

<!-- 事件处理函数调用：常规调用 -->

<button @click="say($event)"></button>

# 注：如果没有参数时，可以不写小括号，默认就会把event事件对象绑定到实现的方法中，如果需要传参数时，则通过 $event 来手动传递到实现的方法中

* 事件修饰符

用来处理事件的特定行为

<!-- 阻止冒泡 -->

<button @click.stop="doThis"></button>

<!-- 阻止默认行为 -->

<a @click.prevent="doThis"></a>

<!-- 只执行一次 -->

<div @click.once="incr()">自增一下</div>

<!-- 绑定的元素本身触发时才触发回调 -->

<ul @click.self="incr()">

<li>你好世界</li>

</ul>

<!-- 串联修饰符 -->

<button @click.stop.prevent="doThis"></button>

* 按键修饰符

在监听键盘事件时，我们经常需要检查详细的按键。Vue 允许为 v-on 在监听键盘事件时添加按键修饰符。

<!-- 只有在 `key` 是 `Enter` 回车键的时候调用 -->

<input @keyup.enter="submit">

<!-- Ctrl + Click -->

<div @click.ctrl="doSomething">Do something</div>

你还可以通过全局 config.keyCodes 对象自定义按键修饰符别名(需脚手架演示)：

// 可以使用 `v-on:keyup.f1`

Vue.config.keyCodes = {f2:113}

<input @keyup.f2="add()" value="aaaa">

## 4.8、样式绑定

### 4.8.1、class样式处理

* 对象语法

.active {

color: red;

}

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

data: {

isActive: true

}

* 数组语法

.active {

color: red;

}

<div v-bind:class="[activeClass]">数组写法</div>

data: {

activeClass: 'active'

}

### 4.8.2、绑定style

* 对象语法

<div :style="{color: redColor, fontSize: '20px'}">对象写法</div>

data: {

redColor: 'red'

}

* 数组语法

<div v-bind:style="[color, fontSize]">数组写法</div>

data: {

color: {

color: 'red'

},

fontSize: {

'font-size': '20px'

}

}

## 4.9、v-model

作用: 表单元素的绑定，实现了双向数据绑定，通过表单项可以更改数据。

v-model会忽略所有表单元素的value、checked、selected特性的初始值,而总是将Vue实例的数据作为数据来源，应该在data选项中声明初始值。

* 绑定文本框

<p>{{message}}</p>

<input type='text' v-model='message'>

<!-- v-model其实是语法糖,它是下面这种写法的简写 -->

<!-- 语法糖：这种语法对语言的功能并没有影响，但是更方便程序员使用 -->

<input v-bind:value='message' v-on:input='message = $event.target.value' />

data: {

message: 'message默认值'

}

* 绑定多行文本框

<textarea v-model="message"></textarea>

data: {

message: '我是多行文本内容'

}

注意：多行文本框中使用插值表达式 无效

* 绑定单个复选框

# 绑定一个复选框

<input type="checkbox" v-model="checked">

data:{

checked:true

}

* 绑定多个复选框

此种方式需要input标签提供value属性

<input type="checkbox" value="html" v-model="checkedNames">

<input type="checkbox" value="css" v-model="checkedNames">

<input type="checkbox" value="js" v-model="checkedNames">

data:{

// 如果数组中有对应的value值，则此checkbox会被选中

checkedNames:[]

}

* 绑定单选框

需要input提供value属性的值

男<input type="radio" value="男" v-model="sex">

女<input type="radio" value="女" v-model="sex">

data: {

sex: ''

}

* 绑定下拉框

<select v-model="selected">

<option>请选择</option>

<option>HTML</option>

<option>CSS</option>

<option>JS</option>

</select>

data: {

selected: ''

}

* 修饰符

.lazy 失去焦点时触发(延时更新数据源)

.number 输入值转为数值类型

.trim 自动过滤用户输入的首尾空白字符