VUE：一套用来构建用户界面的渐进式JavaScript 框架

**特点**：

* 采用组件化模块，提高代码复用率、且让代码更好的维护
* 声明式编码，让编码人员无须操作DOM。提高开发效率
* 使用虚拟DOM+优秀的Diff算法，尽量复用DOM节点
* 编码简洁, 体积小, 运行效率高, 适合移动/PC端开发
* 它本身只关注UI, 可以轻松引入vue插件或其它第三库开发项目
* 遵循MVVM模式
  + MVVM分为三个部分：分别是M（Model，模型层 ），V（View，视图层），VM（ViewModel，V与M连接的桥梁，也可以看作为控制器）
    - M：模型层，主要负责业务数据相关；
    - V：视图层，顾名思义，负责视图相关，细分下来就是html+css层；
    - VM：V与M沟通的桥梁，负责监听M或者V的修改，是实现MVVM双向绑定的要点
  + **ViewModel是Vue.js的核心，它是一个Vue实例**,MVVM支持双向绑定，意思就是当M层数据进行修改时，VM层会监测到变化，并且通知V层进行相应的修改，反之修改V层则会通知M层数据进行修改，以此也实现了视图与模型层的相互解耦

**VUE配置对象**：—🡪不支持箭头函数

-el:”选择器”或”获取到的元素”,

-data:建议写为函数形式，{{msg:任意类型}，{}，{}},或data：function(){return {{msg:任意类型}，{}，{}}}

-methods:{a(){},b(){}} 用于存放方法,this自动绑定实例，可以在指令表达式中使用

-computed:{function(){}} 计算属性,书写为函数形式调用为属性形式，可以直接调用

-watch : { 监听对象：{depp：true 是否深度监听对象的内部值，immediate;true 是否立即执行 ,function(){执行函数}} } 简写：watch：{对象(){}} 配置默认不开启

-template:Vue模板（以子组件，模板，原始普通DOM生成虚拟DOM）

-props: props一定要在配置项中接收，否则不能使用，props的值也直接放在了vc上，所以可以直接在模板中访问，props:["car","girl"],//默认props的写法，car为key值

-filters:function(data，传参){ return } 过滤器使用：{{data | fn1 | fn2(参)}},全局使用过滤器Vue.filter("fn1",callback)

-directives:指令(e,o){ //参数1是当前书写指令的DOM 参数2是配置对象} 全局使用自定义组件 Vue.directive('fn',callback)

-components:注册子组件（关联，子组件需要写在前面，）组件名命名时不能大写(可大写**首字母**)，注意不要与html原有标签不重复，使用时不区分大小写，不区分单双标签

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

**子组件**：配置与vm基本一致，但没有el---，独有name配置用于描述，

《template》上可以给模板上的元素设置ref属性，这个ref属性其实可以看做是id的替代品当ref被设置以后，在vc或者vm上的$refs对象属性上就有了当前ref作为key，DOM节点作为值的一组键值对我们可以通过 this.$refs.XXX得到DOM节点

**双大括号语法**：

-书写的数据直接从实例 vm上寻找

-所有的类型都是以字符串展示(直接加引号),null,undefined转为空

-插值语法可以书写js表达式,(3元，赋值，运算，或与非。。。)

-表达式中的字符需要加上引号

**$mount** :

-如果 Vue 实例在实例化时没有收到 el 选项，则它处于“未挂载”状态，没有关联的 DOM 元素。

--可以使用 vm.$mount() 手动地挂载一个未挂载的实例

--$mount挂在和el属性挂载本质上是没有任何区别的

--$mount可以延迟进行挂载

应用: vm.$mount("#app")

**数据代理**：

原理：-遍历data对象中所有的属性，通过Object.defineProperty()一个一个都添加到vm上。

-随后vm上就拥有了\_data中所有的属性，且都有自己的getter/setter。

-getter/setter内部操作（读/写）的是\_data中对应的属性。

Vue的数据代理：

- 插值语法的数据都是从Vue的实例vm对象上寻找

- Vue中的data都会放在vm实例的\_data属性上

- 我们可以在插值语法中通过\_data.XXX找到对应的数据，但是这样未免太麻烦

- Vue中设置了数据代理，在vm上直接设置了对应的属性，当我们直接访问vm上的数据属性的时候，会直接操作\_data属性中的数据

- Vue在实例化的时候，或遍历data配置中的属性，然后通过defineProperty方法给每一个属性配置了一个代理，放在了vm身上，供插值语法直接使用

**Vue指令**：

**v-bind**：强制绑定

- v-bind：可简写为 :

-v-bind:attr="xxx":给attr属性强制绑定xxx数据

-"xxx" xxx的位置可以书写js语法

-class: 字符串写法适用于：要绑定的样式，类名不确定。

**:class="str" str:'happy'**

对象写法适用于：要绑定的样式，个数确定，类名也确定，但不确定用不用

**:class="obj" obj={atguigu1:true,}**

数组写法适用于：要绑定的样式，个数不确定、类名也不确定

**:class="arr" arr = []**

-styele: v-bind:style = “JavaScript对象” 的对象语法十分直观——看着非常像 CSS，但其实是一个 对象。

CSS property 名可以用驼峰式 (camelCase) 或短横线分隔 (kebab-case，记得用引号括起来) 来命名<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>

直接绑定到一个样式对象通常更好，这会让模板更清晰

变量

**v-model**:数据的双向绑定

-input标签 v-model：value="xxx"可简写为v-model="xxx"

-单个复项框 绑定的是当前框的"xxx"的状态(boolean值)

-多个复选框 绑定到同1个数组值存在数组中

-单选框 绑定的是当前框的"xxx"的值(vlaue值)

-选择框(select) 绑定的是select框的"xxx"的值

数据绑定的原理：MVVM

**v-on**:事件绑定

-书写格式：v-on：事件类型 = "事件函数"

-简写为@事件类型 = "事件函数"

-存在e，

-事件函数可以传递参数，加上（）也只在事件触发时调用

-- <!-- 如果传参覆盖了event对象,则可以书写$event实参来传递event事件对象 -->

--<button @mouseenter="enter('go',$event)">按钮</button> 顺序一一对应

——事件修饰符: 阻止默认事件 @click.prevent="handleClick2"

阻止传播 @click.stop="handleClick2"

阻止传播和默认事件 @click.stop.prevent="handleClick2"

键盘修饰符:按键为回车键才触发的：@keyup.13="keyup"

或者 @keyup.enter="keyup"

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

**v-if**:true执行 是否在DOM树中渲染:与js中的if语句类似。

**v-else：false/v -else-if:true** 可以使用 v-else 指令来表示 v-if 的“else 块”,v-else 元素必须紧跟在带v-if 或者 v-else-if 的元素的后面，否则它将不会被识别

**v-show**:true 在DOM中渲染但是否显示出来，也是控制元素的显示和隐藏，但是只是简单地切换元素的 CSS property display。

**v-for**： <li v-for="(item,index) in data "> 用于数据的遍历

数组：item,index

对象：value，key，index

字符串：value，index

数字：n==》 会循环生成n次 用于重复生成 例3：num-index 1-0 2-1 3-2

**v-text：**会直接覆盖掉当前元素中所有的内容，不会解析任何标签

**v-html：**会直接覆盖掉当前元素中所有的内容，可以解析标签(可能导致漏洞，"xss")

如果要更新部分内容 请使用插值语法(双大括号语法)

**v-pre:**标签不进行解析直接显示内容，

**v-once:**第一次加载时读取并解析变量，再次的修改不生效

计算属性:

-插值语法 书写繁琐

-封装函数 每次使用都会调用函数

-计算属性 computed |

-相对比封装函数减少了调用

-计算属性监视内部使用的属性的变化，一旦发生变化，计算属性就要重新计算

侦听属性: -侦听器watch是侦听一个特定的值，当该值变化时执行特定的函数

-被监听的属性可以是属性(data)也可以是计算属性(computed)

-配置：immediate:true 可以让数据没有发生变化时，Vue初次工作时就执行一次监视的回调

-配置deep:true可以监测对象内部值改变（监视多层）

对比：

区别：computed能完成的功能，watch都可以完成。watch能完成的功能，computed不一定能完成，最具代表性的：watch可以进行异步操作。

watch擅长处理的场景：一个数据影响多个数据 (搜索数据)，侧重在【监视】，核心是：xxx变了，我要做???事。无需return，靠内部逻辑去完成要做的事，可以开启异步任务。

computed擅长处理的场景：一个数据受多个数据影响 (购物车商品结算的时候)，侧重在【算】，核心是：计算出来的值。靠return来输出计算的结果。不能开启异步任务。

类似：当某一个数据（称它为依赖数据）发生变化的时候，所有依赖这个数据的“相关”数据“自动”发生变化，也就是自动调用相关的函数去实现数据的变动。

对methods:methods里面是用来定义函数的，很显然，它需要手动调用才能执行。而不像watch和computed那样，“自动执行”预先定义的函数。

数据劫持与代理：**见晨测**

修饰符:

组件通行：

Props：

自定义事件：

事件总线：

订阅与发布：

插槽：

Vuex：

路由传参：

Params：

Query：

Props：

动画：

与或非：



