Vue是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。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。另一方面，当与[现代化的工具链](https://cn.vuejs.org/v2/guide/single-file-components.html)以及各种[支持类库](https://github.com/vuejs/awesome-vue#libraries--plugins)结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

作用：动态构建用户界面

模板页面：html+js

1 指令：指令属性的属性值是一个js表达式

2 插值：在标签体文本中使用{{}}包含一个表达式，自动显示对应的值

new Vue 配置对象

el：element，值为选择器，用来找到某个标签，指定模板的区域

data：为模板提供数据

MVVM

Model data js对象

View 模板 dom

ViewModel vue

数据绑定：模板解析model中的data数据

Dom监听：监听input变化更新data数据

模板语法

1 双大括号表达式（插值）

2 指令一：强制数据绑定 v-bind:href=’url’ :href=’url’ 根据url动态变化

3 指令二：绑定事件监听 v-on:click=’test1’ @click=’test1’ methods:{test1(){…}}

计算属性和监视

Object.defineProperty(obj,prop,descriptor)

descriptor 配置对象，属性描述符

数据描述符 {value:’’ , writable:false}

存取描述符 {get(){} 当读取属性值时调用，返回属性值，this是obj

,set(value){ }} value是当前属性最新的值。

当修改属性值后调用，监视属性值变化，this是obj

计算属性

computed:{

fullname1(){return this.name1+this.name2},

fullname2:{

get(){return this.name1},

//get调用的时机：读取属性1初始第一次调用 2依赖的数据发生改变

set(value){this.name1=value}

//set调用的时机：当前属性值发生改变

}

}

计算属性会缓存在一个对象中，属性名为key，value为最新的属性值

{name1:’A-B’,name2:’C-D’}

什么时候会缓存？get()返回一个新的值/主动改变属性值

监视

data对象中所有层次的属性都有对应的set监视（数组除外）。

watch:{

name1(value){this.fullname2=value+this.name2} name1的值变化

}

vm.$watch(‘name2’,function(value){this.fullname2=this.name1+value}) name2的值变化

class与style绑定 样式

class绑定 :class=’xxx’ 类名不确定

:class=’{classA:hasA,classB:hasB}’ 对象语法：类名确定，不确定有没有

Style绑定 :style=”{color:’activeColor’,fontSize:fontSize+’px’}”

条件渲染

<p v-if=’ok’>成功</p>

<p v-else>失败</p> false时删除p标签

<p v-show=’ok’>成功</p>

<p v-show=’!ok’>失败</p> false时display:none

列表渲染

遍历数组

<li v-for=”(person,index) in persons” :key=”person.id ”>

{{index}}--{{person.id}}--{{person.name}}--{{person.age}}

</li>

数组改变，更新界面需要调用重写的数组方法 this.persons.splice()

使用this.persons[index]=value可以改变数组，不能更新页面，因为数组没有set监视

遍历对象

<li v-for=”(value,key) in persons[1]” :key=”key”>

{{key}}--{{value}}

</li>

事件处理

1 绑定监听

<button @click=”test1(“abc”,$event)”>test1</button>

test1(msg,event){alert(msg,event)}

2 事件修饰符

阻止事件默认行为：

<a href=<http://www.baidu.com> @click.prevent=”test2”>去百度</a>

test2(event){alert(“点击了”)}

停止事件冒泡：

<div @click=”test3”>

<div @click.stop=”test4”>内部</div>

</div>

3 按键修饰符

<input type=”text” v-model=”msg” @keyup.enter=”test5”>

Test5(event){if(event.keyCode===13){alert(this.msg)}}

表单输入绑定

vm的生命周期

beforeCreate

created

beforeMount

mounted 初始显示后立即执行一次，执行异步操作：定时器、ajax请求

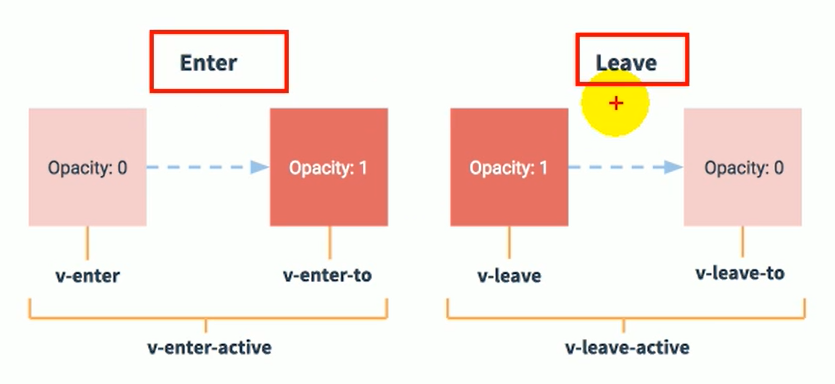
beforeUpdate

updated

beforeDestroy 清除定时器

destroyed

过渡与动画



外部包裹一层标签<transition name=’xxx’></transition>，v为标签中的name

.xxx-enter-active 指定显示的过渡样式

.xxx-enter .xxx-leave-to 指定隐藏时的样式

过滤器

对于显示的数据进行格式化处理

{{date | dateFormat}}

{{date | dateFormat(‘YYYY-MM-DD’)}}

Vue.filter(‘dateFormat’,function(value,formatStr){

return moment(value).format(formatStr || ‘YYYY-MM-DD HH:mm:ss’)

})

指令 用来操作标签

内置指令

v-text 更新元素的textContent

v-html 更新元素的innerHTML

ref 添加唯一标识

{{msg}}

<p v-text=”msg”></p> 显示纯文本

<p v-html=”msg”></p>

<p ref=”content”>sss</p>

alert(this.$refs.content.innerHTML)

自定义指令

注册全局指令

Vue.directive(‘upper-text’,function(el,binding){

el.innerText = binding.value.toUpperCase()

//el为指令属性所在的标签对象，binding为包含相关数据的对象

})

注册局部指令

directives:{

‘lower-text’ (el,binding){

el.innerText = binding.value.toLowerCase()

}}

组件间通信

父子A和B props ref

A向B传递数据，B用props接收

B向A传递数据，通过调用A传递的函数实现

兄弟A和B 借助父组件C通信

A调用C传递的函数改变C中定义的数据，即A向C传递数据，然后C向B传递该数据

事件：

绑定事件监听 原生/自定义：都是自己做

分发事件 原生：浏览器自动分发事件 自定义：执行分发事件

全局事件总线 任意组件通信

创建一个vm对象作为全局事件总线对象，并挂载到Vue的原型对象上。

Vue.prototype.$globalEventBus = new Vue()

通过全局事件总线对象分发自定义事件

this.$globalEventBus.$emit(‘event’, data)

通过全局事件总线对象绑定自定义事件监听

mounted(){

this.$globalEventBus.$on(‘event’, func)

}

slot插槽

父向子通信，传递带数据的标签结构，标签在父组件解析

向路由组件传递数据

params/query: <router-link to=”/home/news/123?zzz=456”>

将请求参数映射成props: props: route => ({id: route.params.id})

变相props: <route-view msg=”abc”>

编程式路由导航

this.$router.push(path) 相当于点击路由链接(可以返回到当前路由界面)

this.$router.replace(path) 用新路由替换当前路由(不可以返回到当前路由)

this.$router.back() 请求返回上一个记录路由

缓存路由组件

<keep-alive>

<router-view></router-view>

</keep-alive>

Vuex

