Vue2学习

Vue-router----路由（应对大型应用）

Vuex----状态管理（更好的开发体验）

Vue-cli----脚手架（非常容易地构建项目）

概念：

指令：指令带有前缀 v-，以表示它们是 Vue.js 提供的特殊属性，指令只封装 DOM 操作。例如（v-bind:title=”message”）含义将这个元素节点的 title 属性和 Vue 实例的 message 属性绑定到一起。

组件：组件代表一个自给自足的独立单元 —— 有自己的视图和数据逻辑，让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树。

申明式渲染：

1｛｛message｝｝ DOM 文本到数据

2 v-bind:title=”message” DOM 文本到数据

条件式与循环：

1 v-if=”true” 不仅可以绑定 DOM 文本到数据，也可以绑定 DOM ****结构****到数据。

2 v-for=”item in items” 绑定数据到数组来渲染一个列表。

处理用户输入：

1 v-on:click=”show1” 用 v-on 指令绑定一个监听事件用于调用我们 Vue 实例中定义的方法。简写：@click=”show1”

2 v-model=”message” 在表单输入和应用状态中做双向数据绑定。

用组件构建（应用）：

1 Vue.component('todo-item',{

props:['todo'], //子元素通过 props 接口实现了与父亲元素很好的解耦。 //我们应该将数据从父作用域传到子组件，它能够接受一个[属性](https://vuefe.cn/v2/guide/components.html" \l "Props)字段。这里的意思为将父作用域的数据传递给子组件中的todo属性

template:'<li>{{todo.text}}</li>',

});

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

将colorList中的每个item的值赋值给todo属性，使子元素直接操作todo属性上的信息，从而达到父元素与子元素的解耦。

Vue实例

构造器：

1 var vm = new Vue({ }); 创建一个 ****Vue 的根实例,****需要传入一个****选项对象****，它可以包含数据(data)、模板(template)、挂载元素(el)、方法(methods)、过滤函数（filters）、生命周期钩子（created、mounted、updated、destriyed）、计算属性（computed）、监听（watch）、局部组件（components）、指令（directives）、混合对象（mixins）等选项。

可以扩展 Vue 构造器:var MyComponent = Vue.extend({

// 扩展选项

})

// 所有的 `MyComponent` 实例都将以预定义的扩展选项被创建

var myComponentInstance = new MyComponent()

属性和方法：

1 每个 Vue 实例都会****代理****其 data 对象里所有的属性。Vm.a === data.a

2 Vue 实例暴露了一些有用的实例属性与方法。这些属性与方法都有前缀 $。如vm.$data vm.$el vm.$watch 等等。

3 实例生命周期（****生命周期钩子****）钩子的this指向调用它的Vue实例。

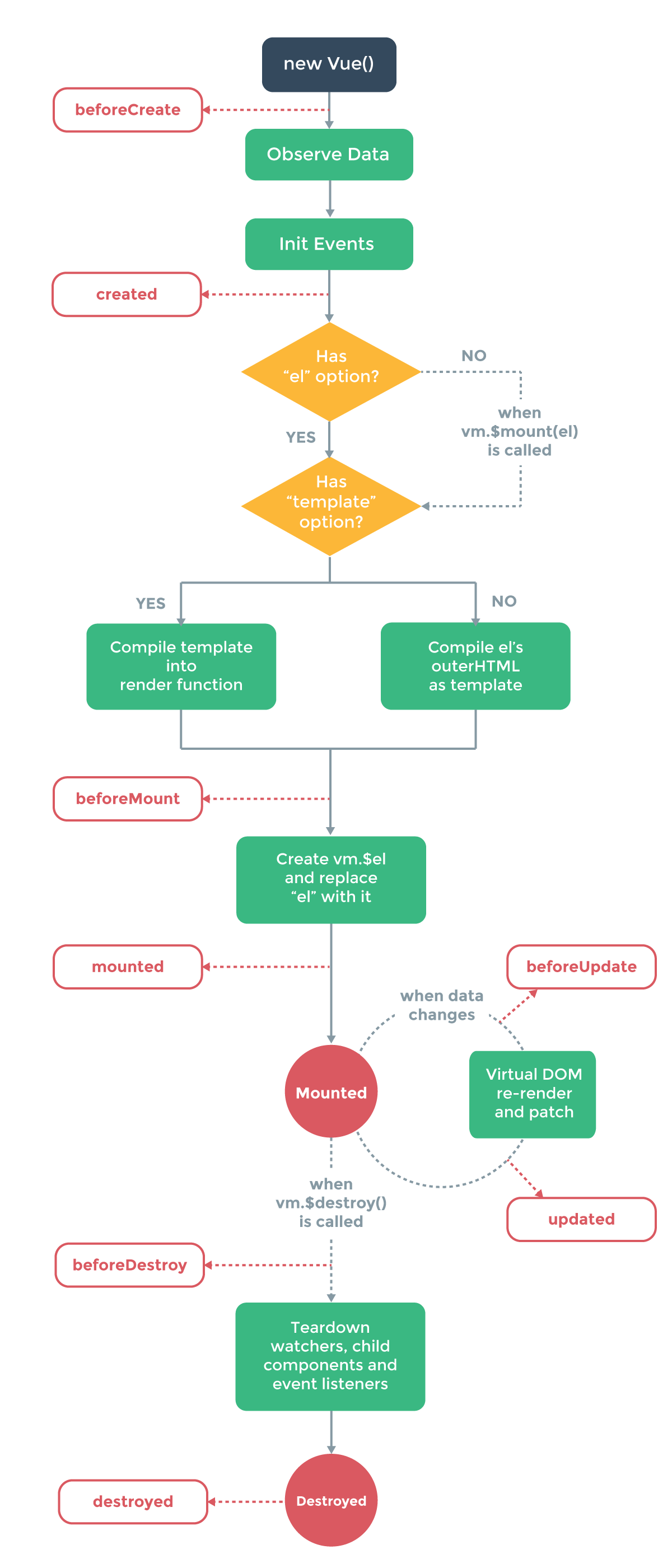
Created---- 这个钩子在实例被创建之后被调用。

Mounted----

Updated----

Destroyed----

生命周期图示





模版语法

插值：

文本：数据绑定最常见的形式就是使用 “Mustache” 语法（双大括号）的文本插值。如<span>{{msg}}</span>,通过使用 [v-once 指令](https://vuefe.cn/v2/api/" \l "v-once)，你也能执行一次性地插值。如<span v-once>{{msg}}</span>当数据改变时，插值处的内容不会更新。

纯HTML：使用指令v-html。如：<div v-html=”rawHtml”></div>

属性：使用v-bind指令。如：<div v-bind:id=”showId”></div>,<button v-bind:disabled="someDynamicCondition">Button</button>

使用js表达式：每个绑定都只能包含****单个表达式。****如：{{number + 1}}, {{ ok ? 'YES' : 'NO' }}, <div v-bind:id="'list-' + id"></div>

指令：（Directives）

参数：v-bind 指令被用来响应地更新 HTML 属性。如<a v-bind:href=’url’></a>

用v-on：ckick=“todo”。如：它用于监听 DOM 事件。

修饰符：（Modifiers）用于指出一个指令应该以特殊方式绑定。

如：.prevent----event.preventDefault()。 v-on:submit.prevent=”onSubmit”

过滤器：管道符--过滤器应该被添加在 ****mustache 插值****的尾部。过滤器设计目的就是用于文本转换。过滤器只能在 mustache 绑定和 v-bind 表达式中使用。如：{{ message | filterA | filterB }}，过滤函数写在vue实例对象filters属性中。

缩写：v-bind：href=“url” ==== ：href=“url”

V-on：click=“todo” === @click=“todo”

计算属性

计算属性：任何复杂逻辑，你都应当使用****计算属性****。

放于实例化的vue对象中的computed属性中。

计算属性vs Methods ****计算属性是基于它的依赖缓存，方法基于响应依赖。****

计算属性vs Watched Property Vue.js 提供了一个方法 $watch ，它用于观察 Vue 实例上的数据变动。

计算setter 计算属性默认只有 getter ，不过在需要时你也可以提供一个 setter 。

观察Watchers：异步操作（访问一个 API），除了 watch 选项之外，您还可以使用 [vm.$watch API](https://vuefe.cn/v2/api/" \l "vm-watch) 命令。官网很好的例子。

CLASS与STYLE绑定

绑定HTML Class：

对象语法 v-bind：class=“obj” 如：<p class="static" v-bind:class="{active:isActive}">是否添加active类取决于isActive的值。

数组语法 v-bind：class=“array” 如：<div v-bind:class="[activeClass, errorClass]">，<p v-bind:class="[active ? 'active' : '', error]">

用在组件上 当你在一个定制的组件上用到 class 属性的时候，这些类将被添加到根元素上面，这个元素上已经存在的类不会被覆盖。

声明组件时：

Vue.component(‘my-component’,{

Template:’<p class=”foo bar”></p>’,

});

使用组件时：

<my-component class="baz boo"></my-component>

绑定内联样式：

对象语法 v-bind:style=“styleObj”如：<div v-bind:style="obj">222</div>

数组语法 v-bind:style=“styleArray”如：<div v-bind:style="[baseStyles, overridingStyles]">

自动添加前缀 如 transform ，Vue.js 会自动侦测并添加相应的前缀。

条件渲染

v-if：

<template>中v-if条件组 我们可以把一个 <template> 元素当做包装元素，并在上面使用 v-if，最终的渲染结果不会包含它。

V-else 无参数必须紧跟在 v-if 元素或者 v-else-if的后面——否则它不能被识别。

v-else-if v-else-if 必须跟在 v-if 或者 v-else-if之后。

使用key控制元素的可重用 官网例子----切换登录方式，Vue 提供一种方式让你可以自己决定是否要复用元素。你要做的是添加一个属性 key ，key必须带有唯一的值。

v-show：

<h1 v-show="ok">Hello!</h1>只是简单的切换了display属性，注意 v-show 不支持 <template> 语法。

v-show vs v-if：

V-if 使用：运行时条件不大可能改变则使用 v-if 较好。高消耗

v-show：如果需要频繁切换使用 v-show 较好。

V-if with v-for：当与v-for一起使用时，v-for具有比v-if更高的优先级。

列表渲染

v-for：

基本用法 v-for=“item in/of items”如：<li v-for=”item in items”>{{item.msg}}</li>

带有索引 v-for=“（item，index） in/of items”

Template v-for 如：<template v-for=”item in items”> other</template>

对象v-for 如：<li v-for="value in object">{{ value }}</li>

带有key-val对 如：<li v-for="(val, key) in obj">{{key}} + {{val}}</li>

带有三个参数 如：<div v-for="(value, key, index) in object">

{{ index }}. {{ key }} : {{ value }}

</div>在遍历对象时，是按 Object.keys() 的结果遍历，不同js引擎有差异。

范围v-for <span v-for="n in 10">{{n}}</span>

组件和v-for 如：<my-component v-for="item in items"></my-component>

为了传递迭代数据到组件里，我们要用 props；自定义事件：@remove="todos.splice( index, 1 )" 触发自定义事件：@click="$emit('remove')"

v-for with v-if v-for具有比v-if更高的优先级。 这意味着v-if将分别在循环的每次迭代上运行。渲染某些部分项的节点时非常有用。

若你的意图是按条件跳过循环的执行，你可以将v-if放在一个包装元素（或[<template>](https://vuefe.cn/v2/guide/conditional.html" \l "Conditional-Groups-with-v-if-on-lt-template-gt)）里面。

Key：为了给 Vue 一个提示，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素，你需要为每项提供一个唯一 key 属性。理想的 key 值是每项都有唯一 id。

如：<div v-for="item in items" :key="item.id"> <!-- 内容 --> </div>

数组更新检测：

变异方法 (mutation method)Vue 包含一组观察数组的变异方法，所以它们也将会触发视图更新。就是改变原数组。

Push()、pop()、shift()、unshift()、splice()、sort()、reverse();

重塑数组 用新数组去特换原数组。

filter()、concat()、slice();

注意事项 由于 JavaScript 的限制， Vue 不能检测以下变动的数组。

1当你利用索引直接设置一个项时，例如： vm.items[indexOfItem] = newValue

解决方案：Vue.set( vm.items,indexOfItem, newValue );

Vm.items.splice( indexOfItem, 1, newValue );

2当你修改数组的长度时，例如： vm.items.length = newLength

解决方案：Vue.set( vm.items,indexOfItem, newVlue );

显示过滤/排序结果：

我们想要显示一个数组的过滤或排序副本，而不实际改变或重置原始数据。两种方式：

1计算属性。

2methods。如<li v-for="n in even(numbers)">{{ n }}</li> even（）为事件

事件处理器

监听事件：

v-on 指令监听 DOM 事件来触发一些 JavaScript 代码。

<button v-on:click="counter += 1">增加 1</button>

方法事件处理器：方法在methods中

<button v-on:click="greet">按钮</button>

内联处理器方法：

<button v-on:click="greet( 'ffff' )">按钮</button>；有时也需要在内联语句处理器中访问原生 DOM 事件。可以用特殊变量 $event 把它传入方法。如： <button v-on:click="greet( 'ffff', $event )"> submit</button>在方法中用

greet( msg, event ){ event.preventDefault(); alert( msg ); }

事件修饰符：

如：<a v-on:click.stop.prevent="doThis"></a>

.stop === 阻止单击事件冒泡

.prevent === 默认行为

.capture === 添加事件侦听器时使用事件捕获模式

.self === 只当事件在该元素本身（而不是子元素）触发时触发回调

.once === 该函数只触发一次

按键修饰符：

如：<input v-on:keyup.17.67="greet( '你确定要复制吗？')">

.enter === 13 .tab .esc .ctrl

.delete .space .up .alt

.down .left .right .shift .meta

可以通过全局 config.keyCodes 对象[自定义按键修饰符别名](https://vuefe.cn/v2/api/" \l "keyCodes)：

如：Vue.config.keyCodes.f1 = 112

表单控件绑定

基础用法：

文本：<input type="text" v-model="msg" >

多行文本：<textarea v-model="message" placeholder="add multiple lines"></textarea>

复选框：<input type="checkbox" id="checkbox" v-model="checked">

多个勾选框，绑定到同一个数组：

<label>jack<inputtype="checkbox"value="jack"v-model="checkedNames"></label>

<label>fan<inputtype="checkbox"value="fan"v-model="checkedNames"></label>

单选按钮：

<label>one<inputtype="radio"id="one"value="one"v-model="picked"></label>

<label>two<input type="radio" id="two" value="two" v-model="picked"></label>

选择列表：

<select v-model="picked">

<option value="">选择</option>

<option value="A">A</option>

<option value="B">B</option>

</select>

单选和多选区别：单选----picked为String；多选----picked为Array；

动态选项，用 v-for 渲染：

<select v-model="selected">

<option v-for="option in options" v-bind:value="option.value">

{{ option.text }}

</option>

</select>

绑定value：

对于单选按钮radio及select选择列表选项， v-model 绑定的 value 通常是静态字符串（对于checkbox勾选框是逻辑值）

但是有时我们想绑定 value 到 Vue 实例的一个动态属性上，这时可以用 v-bind 实现，并且这个属性的值可以不是字符串。

复选框：<input type="checkbox" v-model="toggle" v-bind:true-value="a" v-bind:false-value="b">

单选按钮：<input type="radio" v-model="pick" v-bind:value="a">

选择列表设置：

<select v-model="selected">

<option :value="{number:123}" >123</option>

</select>

修饰符：.lazy .number .trim

在 "change" 而不是 "input" 事件中更新 如：<input v-model.lazy="msg" >

如果想自动将用户的输入值转为 Number 类型（如果原值的转换结果为 NaN 则返回原值） 如：<input v-model.number="age" type="number">

如果要自动过滤用户输入的首尾空格 如：<input v-model.trim="msg">

组件（Component）

定义：

组件可以扩展 HTML 元素，封装可重用的代码。组件也可以是原生 HTML 元素的形式，以 js 特性扩展。

使用组件：

全局注册 注册一个全局组件Vue.component(tagName, options);如：Vue.component('my-component', {// 选项})，必须在vue实例之前注册组件。

局部注册 通过使用组件实例选项注册，可以使组件仅在另一个实例/组件的作用域中可用。如：

new Vue({ components: {

'my-component': Child

}

})

使用 在父实例的模块中以自定义元素 <my-component></my-component> 的形式使用。

DOM模版解析说明：<ul> ， <ol>，<table> ， <select> 等在其内部只能存放指定子元素。

解决方案：如： 其中my-row为自定义标签名。

<table>

<tr is="my-row"></tr>

</table>

Data必须是函数 （在组件中）data(){ return data; ｝

构建组件 父组件通过 ****props**** 向下传递数据给子组件，子组件通过 ****events**** 给父组件发送消息，总结为 ****props down, events up**** 。

Prop：

使用Prop传递数据 在组件注册中props: ['message'],

camelCase vs kebab-case 当使用非字符串模版时，prop的名字形式会从 camelCase 转为 kebab-case（短横线隔开）。

动态Prop 用 v-bind 动态绑定 props 的值到父组件的数据中。每当父组件的数据变化时，该变化也会传导给子组件。如：

<input v-model="parentMsg">

<child v-bind:my-message="parentMsg"></child>

字面量语法 vs 动态语法 <comp v-bind:some-prop="1"></comp>此时子组件接受到的some-prop为number类型，如直接这样写some-prop=“1”为String类型。

单向数据流 当父组件的属性变化时，将传导给子组件，但是不会反过来。

两种改变 prop 的情况：

定义一个局部 data 属性，并将 prop 的初始值作为局部数据的初始值。

定义一个 computed 属性，此属性从 prop 的值计算得出。组件中的computed属性。

注意在 JavaScript 中对象和数组是引用类型，指向同一个内存空间，如果 prop 是一个对象或数组，在子组件内部改变它****会影响****父组件的状态。

Prop验证 type：String、Number、Boolean、Function、Object、Array、Null

required：true

default：100

自定义事件： 子组件要把数据传递回父组件。

使用v-on绑定自定义事件 $on(eventName) 监听事件； $emit(eventName) 触发事件。父组件可以在使用子组件的地方直接用 v-on 来监听子组件触发的事件。

给组件绑定原生事件

在某个组件的根元素上监听一个原生事件。可以使用 .native 修饰 v-on 。如：

<my-component v-on:click.native="doTheThing"></my-component>

使用自定义事件的表单输入组件 官网例子。

非父子组件通信

用一个空的 Vue 实例作为中央事件总线；

var bus = new Vue();

bus.$emit('id-selected',1); // 触发组件 A 中的事件

bus.$on('id-selected',function( id ){ // 在组件 B 创建的钩子中监 听事件});更复杂的情况使用专门的 [状态管理模式](https://vuefe.cn/v2/guide/state-management.html).

使用slot分发内容： （在组合组件时，内容分发API非常有用） 为了让组件可以组合，我们需要一种方式来混合父组件的内容与子组件自己的模板。这个过程被称为 ****内容分发，****使用特殊的 <slot> 元素作为原始内容的插槽。

编译作用域 父组件模板的内容在父组件作用域内编译；子组件模板的内容在子组件作用域内编译。

单个Slot 除非子组件模板包含至少一个 <slot> 插口，否则父组件的内容将会被****丢弃****。当子组件模板只有一个没有属性的 slot 时，父组件整个内容片段将插入到 slot 所在的 DOM 位置，并替换掉 slot 标签本身。

具名的Slot <slot> 元素可以用一个特殊的属性 name 来配置如何分发内容。多个 slot 可以有不同的名字。具名 slot 将匹配内容片段中有对应 slot 特性的元素。

仍然可以有一个匿名 slot ，它是****默认 slot**** ，作为找不到匹配的内容片段的备用插槽。如果没有默认的 slot ，这些找不到匹配的内容片段将被抛弃。

作用域插槽 用作使用一个（能够传递数据到）可重用模板替换已渲染元素。在父级中，具有特殊属性 scope 的 <template> 元素，表示它是作用域插槽的模板。scope 的值对应一个临时变量名，此变量接收从子组件中传递的 prop 对象。如：

<template scope="props">

<span>{{ props.text }}</span>

</template>作用域插槽更具代表性的用例是列表组件。

动态组件：

多个组件可以使用同一个挂载点，然后动态地在它们之间切换。使用保留的 <component> 元素，动态地绑定到它的 is 特性。如：

<component :is="currentView"> </component>

Keep-alive如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。

<keep-alive><component :is="currentView"></component>

</keep-alive>

杂项：

编写可复用的组件 Vue 组件的 API 来自三部分 - props, events 和 slots；

Props 允许外部环境传递数据给组件

Events 允许组件触发外部环境的副作用

Slots 允许外部环境将额外的内容组合在组件中。

子组件的索引 JavaScript 中直接访问子组件，ref 为子组件指定一个索引 ID 。如：<user-profile ref="profile"></user-profile> var parent = new Vue({ el: '#parent' })则访问子组件 var child=parent.$refs.profile;

异步组件 Vue.component('child',function( resolve, reject ){

resolve({ template:`<div>I am async!</div>`, }); });

组件命名约定 推荐用kebab-case

递归组件 1组件在它的模板内可以递归地调用自己，不过，只有当它有 name 选项时才可以。2当你利用Vue.component全局注册了一个组件, 全局的ID作为组件的 name 选项，被自动设置。

组件之间的循环引用 假设您正在构建一个文件目录树，像是 Mac 下的 Finder 或是 Windows 下的文件资源管理器。

内联模版 如果子组件有 inline-template 特性，组件将把它的内容当作它的模板，而不是把它当作分发内容。

X-Templates 另一种定义模版的方式是在 JavaScript 标签里使用 text/x-template 类型，并且指定一个id。 少使用

<script type="text/x-template" id="hello-world-template">

<p>Hello hello hello</p>

</script>

Vue.component('hello-world', {

template: '#hello-world-template'

})

使用v-once的低消耗静态组件 用 v-once 将渲染结果缓存起来。如：

Vue.component('terms-of-service', {

template: '

<div v-once>

<h1>Terms of Service</h1>

... a lot of static content ...

</div>

'

})

深入响应式原理

如何追踪变化： 只有data 对象中的数据才是响应式的。

将vue实例中的data选项用Object.defineProperty()转化为getter和setter。

变化检测问题：

将响应属性添加到嵌套的对象上 1 Vue.set(vm.someObject, 'b', 2)；

2 vm.$set(this.someObject, ’b’, 2);(vue的实例方法)

已有对象上添加一些属性 保证响应式蓝色为新加属性。

this.someObject = Object.assign({}, this.someObject, { a: 1, b: 2 })

声明响应式属性：

由于 Vue 不允许动态添加根级响应式属性，所以你必须在初始化实例前声明根级响应式属性，哪怕只是一个空值。

异步更新队列：

使用Promise.then 和 MutationObserver，备用setTimeout(fn, 0)。

为了在数据变化之后等待 Vue 完成更新 DOM ，可以在数据变化之后立即使用 Vue.nextTick(callback) 全局的。这样回调函数在 DOM 更新完成后就会调用。

在组件内使用 vm.$nextTick()，this指向当前Vue实例。

过渡效果

Vue 在插入、更新或者移除 DOM 时，提供多种不同方式的应用过渡效果。可以配合Animate.css等第三方库。

单元素/组件的过渡： transition 给任何元素和组件添加 entering/leaving 过渡。 如：

<transition name="fade">

<p v-if="show">hello</p>

</transition>

过渡的-css-类名 v-enter》》 v-enter-active》》 v-enter-to》》v-leave》》v-leave-active》》v-leave-to。以上完成效果均删除相应的类。使用 <transition name="b"> 可以重置前缀，比如 v-enter 替换为 b-enter。

Css过渡 常用的过渡都是使用 CSS 过渡。

CSS动画 CSS 动画用法同 CSS 过渡，区别是在动画中 v-enter 类名在节点插入 DOM 后不会立即删除，而是在 animationend事件触发时删除。

自定义过渡类名 enter-class enter-active-class enter-to-class

eave-class leave-active-class leave-to-class

如：配合animated.css使用

<transition name="b" enter-active-class="animated tada" leave-active-class="animated bounceOutRight">

<p class="block" v-show="show">hello</p>

</transition>

同时使用Transitions 和 Animations

Vue 为了知道过渡的完成，必须设置相应的事件监听器。它可以是 transitionend 或 animationend。

Javscript钩子 对应的方法写进methods选项中在enter和leave中

function (el, done) {

// ...

done()只有js中done（）必须的

},

<transition

v-on:before-enter="beforeEnter"

v-on:enter="enter"

v-on:after-enter="afterEnter"

v-on:enter-cancelled="enterCancelled"

v-on:before-leave="beforeLeave"

v-on:leave="leave"

v-on:after-leave="afterLeave"

v-on:leave-cancelled="leaveCancelled"

></transition>

使用v-bind:css="false"解决bug

初始渲染的过渡： appear 特性设置节点的在初始渲染的过渡

<transition

appear

appear-class="custom-appear-class"//css类方式，也可使用js钩子

appear-to-class="custom-appear-to-class"

appear-active-class="custom-appear-active-class"

>

<!-- ... -->

</transition>

多个元素的过渡：

在同一个transition标签中，同一标签多次出现要用key属性区分。

过渡模式

in-out: 新元素先进行过渡，完成之后当前元素过渡离开。

out-in: 当前元素先进行过渡，完成之后新元素过渡进入。如：

<transition name="fade" mode="out-in">

<!-- ... the buttons ... -->

</transition>

多个组件的过渡：

<transition name="component-fade" mode="out-in">

<component :is="view"></component>

</transition>

列表过渡：用<transition-group> 组件。

列表的进入和离开过渡

产生的标签默认为 <span>。你也可以通过 tag 属性更换为其他渲染元素，内部的元素****必须****具有唯一的 key 属性。如：

<transition-group name="list" tag="p">

<span v-for="item in items" :key="item">

{{ item }}

</span>

</transition-group>

列表的位移过渡 **v-move **特性****，它会在元素的改变定位的过程中应用。如：

<transition-group name="flip-list" tag="ul">//结合flip-list-move使用

<li v-for="item in items" v-bind:key="item">

{{ item }}

</li>

</transition-group>

列表的渐进过渡

通过 data 属性与 JavaScript 通信 ，就可以实现列表的渐进过渡。

可复用的过渡：

要创建一个可复用过渡组件，你需要做的就是将 <transition> 或者<transition-group> 作为根组件，然后将任何子组件放置在其中就可以了。

动态过渡：

在 Vue 中即使是过渡也是数据驱动的！动态过渡最基本的例子是通过 name 特性来绑定动态值。

过渡状态

状态动画与watcher：官网3个例子

通过 watcher 我们能监听到任何数值属性的数值更新。

动态状态转换：官网例子。

就像 Vue 的过渡组件一样，数据背后状态转换会实时更新，这对于原型设计十分有用。

通过组件组织过渡：

管理太多的状态转换的很快会接近到 Vue 实例或者组件的复杂性，幸好很多的动画可以提取到专用的子组件。

Render函数

基础：

每个[具名 slot](https://vuefe.cn/v2/guide/components.html" \l "Named-Slots) 有其相应的属性（例如：slot="foo" 中的内容将会在 vm.$slots.foo 中被找到）。default 属性包括了所有没有被包含在具名 slot 中的节点,vm.$slots.default.

createElement参数：

完整数据对象 见官网示例。

约束 VNodes 必须唯一（每个元素内容唯一）

使用javascript代替模版功能：

v-if and v-for Vue 的 render 函数不会提供template 中的 v-if和 v-for， 可以被JavaScript 的 if/else 和 map 重写。

v-model 也需要自己实现其功能。在render函数中通过createElement中的domProps、on.input选项进行模拟。

Event & key Modifiers .capture>> ! .once>>~ 简写

其他事件通过原生替代。.stop>>e.stopPropagation()

.prevent>>e.preventDefault() .self>>if(e.target !== e.currentTarget) return

.enter>>if(e.keyCode !== 13) return .ctrl.alt.shift.meta>>if(!e.ctrlKay)return

Slots 通过this.$slots数组访问slot的内容 ；通过this.$scopedSlots数组访问范围slot的内容 。

JSX：

这就是会有一个 [Babel plugin](https://github.com/vuejs/babel-plugin-transform-vue-jsx" \t "https://vuefe.cn/v2/guide/_blank) 插件，用于在 Vue 中使用 JSX 语法的原因，它可以让我们回到于更接近模板的语法上。

函数化组件:

组件为 functional， 这意味它是无状态（没有 data），无实例（没有 this 上下文）。组件需要的一切都是通过上下文（context）。包括

props: 提供props 的对象(this.level 更新为 context.props.level)

children: VNode 子节点的数组(this.$slots.default 更新为 context.children)

slots: slots 对象

data: 传递给组件的 data 对象

parent: 对父组件的引用

Slots() 和 children对比 可以使用slots().default筛选出没有name属性的slot标签或者没有slot属性的其他标签。

模版编译：

自定义指令

全局注册：使用是v-focus

Vue.directive('focus',{

inserted( el ){ //钩子函数

el.focus(); //聚集元素

},

});

局部注册：

directives:{

focus:{

inserted( el ){ //钩子函数

el.focus();

}

}

}

钩子函数：

bind: 只调用一次，指令第一次绑定到元素时调用，用这个钩子函数可以定义一个在绑定时执行一次的初始化动作。

inserted: 被绑定元素插入父节点时调用（父节点存在即可调用，不必存在于 document 中）。

update: 被绑定元素所在的模板更新时调用，而不论绑定值是否变化。通过比较更新前后的绑定值，可以忽略不必要的模板更新（详细的钩子函数参数见下）。

componentUpdated: 被绑定元素所在模板完成一次更新周期时调用。

unbind: 只调用一次， 指令与元素解绑时调用。

钩子函数的参数：

el: 指令所绑定的元素，可以用来直接操作 DOM 。

binding：为对象，包括（name，value，oldValue，expression，arg，modifiers）

vnode: Vue 编译生成的虚拟节点，查阅 [VNode API](https://vuefe.cn/v2/api/" \l "VNode接口) 了解更多详情。

oldVnode: 上一个虚拟节点，仅在 update 和 componentUpdated 钩子中可用。

函数简写：

Vue.directive('color-swatch', function (el, binding) {

el.style.backgroundColor = binding.value

})

对象字面量：

<div v-demo="{ color: 'white', text: 'hello!' }"></div>

混合

选项合并：

混合对象的 钩子将在组件自身钩子 ****之前**** 调用 ，methods, components 和 directives，将被混合为同一个对象。 两个对象键名冲突时，取组件对象的键值对。

全局混合： this.$options可以访问vue实例的选项。

自定义选项混合策略：

自定义选项将使用默认策略，即简单地覆盖已有值。 如果想让自定义选项以自定义逻辑混合，可以向Vue.config.optionMergeStrategies；如；

Vue.config.optionMergeStrategies.myOption = function (toVal, fromVal) {

// return mergedVal

}

插件

开发插件： 插件通常会为Vue添加全局功能。

MyPlugin.install = function (Vue, options) {

// 1. 添加全局方法或属性

Vue.myGlobalMethod = function () {

// 逻辑...

}

// 2. 添加全局资源

Vue.directive('my-directive', {

bind (el, binding, vnode, oldVnode) {

// 逻辑...

}

})

// 3. 注入组件

Vue.mixin({

created: function () {

// 逻辑...

}

})

// 4. 添加实例方法

Vue.prototype.$myMethod = function (options) {

// 逻辑...

}

}

使用插件：

通过全局方法 Vue.use() 使用插件。如：

Vue.use(MyPlugin, { someOption: true }) // 相当于调用 MyPlugin.install(Vue)

在webpack中：

var Vue = require('vue')

var VueRouter = require('vue-router')

// 不要忘了调用此方法

Vue.use(VueRouter)

单文件组件

全局定义(Global definitions) 强制要求每个 component 中的命名不得重复

字符串模板(String templates) 缺乏语法高亮，在 HTML 有多行的时候，需要用到丑陋的 \

不支持CSS(No CSS support) 意味着当 HTML 和 JavaScript 组件化时，CSS 明显被遗漏

没有构建步骤(No build step) 限制只能使用 HTML 和 ES5 JavaScript, 而不能使用预处理器，如 Pug (formerly Jade) 和 Babel

文件扩展名为 .vue 的 ****single-file components(单文件组件)**** 为以上所有问题提供了解决方法，并且还可以使用 Webpack 或 Browserify 等构建工具。

[vue-loader](https://github.com/vuejs/vue-loader" \t "https://vuefe.cn/v2/guide/_blank)----这个模板使用 [Webpack](https://webpack.github.io/" \t "https://vuefe.cn/v2/guide/_blank)，一个能将多个模块打包成最终应用的模块打包工具。

[vue-cli](https://github.com/vuejs/vue-cli" \t "https://vuefe.cn/v2/guide/_blank)-----工具生成新的项目。

vue-router----

vuex----

生产环境部署

删除警告：

Webpack：

var webpack = require('webpack')

module.exports = {

// ...

plugins: [

// ...

new webpack.DefinePlugin({

'process.env': {

NODE\_ENV: '"production"'

}

}),

new webpack.optimize.UglifyJsPlugin({

compress: {

warnings: false

}

})

]

}

预编译模版：

提取组件CSS： Webpack + vue-loader (vue-cli 已经配置好了官方的 webpack 模板)

跟踪运行时的错误： 如果在组件渲染时出现运行错误，错误将会被传递至全局 Vue.config.errorHandler 配置函数（如果已设置）。利用这个钩子函数和错误跟踪服务（如 [Sentry](https://sentry.io/" \t "https://vuefe.cn/v2/guide/_blank)，它为 Vue 提供[官方集成](https://sentry.io/for/vue/" \t "https://vuefe.cn/v2/guide/_blank)），可能是个不错的主意。

路由

官网路由： 对于大多数单页面应用，都推荐使用官方支持的[vue-router库](https://github.com/vuejs/vue-router" \t "https://vuefe.cn/v2/guide/_blank)。

状态管理

官方插件：

解决：由于多个状态分散的跨越在许多组件和交互间各个角落，大型应用复杂度也经常逐渐增长。vuex 甚至集成到 [vue-devtools](https://github.com/vuejs/vue-devtools" \t "https://vuefe.cn/v2/guide/_blank)，无需配置即可访问时光旅行。

用途：如果你有一处需要被多个实例间共享的状态，可以简单地通过维护一份数据来实现共享。

单元测试

配置和工具：

任何兼容基于模块的构建系统都可以正常使用，但如果你需要一个具体的建议，可以使用 [Karma](http://karma-runner.github.io/" \t "https://vuefe.cn/v2/guide/_blank) 进行自动化测试。

编写可被测试的组件：

断言异步更新：

服务端渲染

服务端渲染：

预渲染（prerendering）：

预渲染不像服务器渲染那样即时编译HTML,预渲染只是在构建时为了特定的路由生成特定的几个静态页面。其优势是预渲染的设置更加简单，可以保持前端是一个完整的静态站。

webpack可以很简单地通过[prerender-spa-plugin](https://github.com/chrisvfritz/prerender-spa-plugin" \t "https://vuefe.cn/v2/guide/_blank)来添加预渲染。

流式响应：****流式****渲染

组件缓存：

设置

缓存的理想组件

静态的组件 (例如 总是尝试一样的HTML,所以 serverCacheKey 函数可以被返回 true)

列表组件（当有大量列表，缓存他们可以改善性能）

通用UI组件 (例如 buttons, alerts, 等等 - 至少他们通过props获取数据而不是 slots或者子组件)

构建过程，路由和Vuex状态管理：