目录

[基础 2](#_Toc512503391)

[介绍 2](#_Toc512503392)

[特点 2](#_Toc512503393)

[{{}} 和 {{{}}} 2](#_Toc512503394)

[过滤器 2](#_Toc512503395)

[class 3](#_Toc512503396)

[Vue 3](#_Toc512503397)

[生命周期相关 3](#_Toc512503398)

[实例化 3](#_Toc512503399)

[过程 3](#_Toc512503400)

[挂载 4](#_Toc512503401)

[挂载过程 4](#_Toc512503402)

[更新 4](#_Toc512503403)

[过程 4](#_Toc512503404)

[生命周期 4](#_Toc512503405)

[总结 5](#_Toc512503406)

[mounted 5](#_Toc512503407)

[activated和deactivated 5](#_Toc512503408)

[组件 6](#_Toc512503409)

[模板 6](#_Toc512503410)

[注意 6](#_Toc512503411)

[字符串模板 6](#_Toc512503412)

[dom模板 7](#_Toc512503413)

[属性要改成-连接 7](#_Toc512503414)

[data 7](#_Toc512503415)

[状态管理 7](#_Toc512503416)

[事件绑定 7](#_Toc512503417)

[.sync语法糖 8](#_Toc512503418)

[过渡动画 8](#_Toc512503419)

[其他指令 8](#_Toc512503420)

[v-model 8](#_Toc512503421)

[v-bind 9](#_Toc512503422)

[自定义指令 9](#_Toc512503423)

[mvvm 10](#_Toc512503424)

[简单写法 10](#_Toc512503425)

[经典理解 10](#_Toc512503426)

[双向绑定实现 11](#_Toc512503427)

[其他 11](#_Toc512503428)

# vuecli

## start配置

dev/start : dev-server.js => 判断 process.env.NODE\_ENV === 'testing' ，是的话表示测试调prod.conf，不是这表示开发，调dev.conf

build: prod.conf

process.env.NODE\_ENV: testing | develop | production

prod.conf : 通过process.env.NODE\_ENV判断是build还是test

prod.conf和dev.conf都要调bae.conf

config: 包含build和dev两个对象

test/unit/karma.conf.js调test.conf

# 基础

## 介绍

基于es5的defineProperty，而ie8不支持es5。

### 特点

组件插槽。

## {{}} 和 {{{}}}

{{{}}} 会将键和值一起显示

## 过滤器

过滤器可以用在两个地方：**mustache 插值和 v-bind 表达式** (后者从 2.1.0+ 开始支持)。

## class

除了字符串之外，还可以是对象(可以直接绑定数据里的一个对象，也可以构建对象)或数组(数组里面可以再嵌套对象)

## Vue

可以在多个文件引入Vue,调用Vue.use。因为是直接使用Vue对象，不是实例化的新对象。

# 生命周期相关

## 实例化

实例化时并不是深拷贝，和普通赋值一样。

添加新属性不会触发视图更新。

var data = { a: 1 }

var vm = new Vue({

el: '#example',

data: data

})

vm.$data === data // => true

挂在vue实例属性上的函数(比如watch)，不要使用箭头函数，因为实例化时会传this进去，代表这个实例。

### 过程

也就是执行new Vue({})的过程。

#### created=>beforeMount

先看有没有el,有往下执行，没有返回vm,等vm.mount(el)传进去时往下执行。再看tempate,有的话直接加载进render函数，没有就编译el.outerHTML。最后生成没有装载数据的$.el对象。

## 挂载

可以直接用名字挂载自定义标签。

### 挂载过程

props,computed,data上的数据都会挂载到this上面。

## 更新

this.data或者computed的数据变化，同时变化的部分对视图有影响时，才会触发生命周期更新。

prop不会触发更新，需要通过watch监听绑定到对应的data上。或者用computed

### 过程

类似mount,before时数据已经改变，实际节点没有改变。updated实际节点已经改变。

## 生命周期

a:'vue.js'

console.log(this.a)

console.log(this.$el)



在生命周期函数里获取传下来的属性值。

### 总结

渲染优先级：render函数选项 > template选项 > el.outerHTML.

### mounted

该组件完全挂载

### activated和deactivated

当有keep-alive时才有。

服务端渲染时无效。

当引入keep-alive 的时候，页面第一次进入，钩子的触发顺序created-> mounted-> activated，退出时触发deactivated。当再次进入（前进或者后退）时，只触发activated。

# 组件

## 模板

### 注意

自定义组件直接在html中使用可能不被浏览器识别（比如放在table中），可以使用is属性。

**应当注意，如果您使用来自以下来源之一的字符串模板，这些限制将不适用：**

* <script type="text/x-template">（写一个不被浏览器识别的type, 浏览器会认为他不是javascript脚本，从而不去解析执行里面的内容。它会被其它脚本解释后显示出来。）
* JavaScript 内联模板字符串
* .vue 组件

两种模板都可以使用挂载和直接使用创建时返回的实例两种方式。

### 字符串模板

template: '\

<span>\

$\

<input\

ref="input"\

v-bind:value="value"\

v-on:input="updateValue($event.target.value)"\

>\

</span>\

',

### dom模板

template:'#aaa',

## 属性要改成-连接

|  |
| --- |
| Vue.component('child', {  // camelCase in JavaScript  props: ['myMessage'],  template: '<span>{{ myMessage }}</span>'  }) |

|  |
| --- |
| <!-- kebab-case in HTML -->  <child my-message="hello!"></child> |

## data

通过 Vue 构造器传入的各种选项大多数都可以在组件里用。data 是一个例外，它必须是函数。(其实就是防止data是引用类型，导致多个复用组件实际公用一个数据，一个变其他都变了。)

## 状态管理

通过实例化组件时，只是传递引用类型地址的特性。所以可以类型实例化多个组件使用一个引用类型数据。

# 事件绑定

v-on

v-on:click="warn('Form cannot be submitted yet.', $event)"

所以$event不是必须放在第一个。

其它的修饰符只能对原生的 DOM 事件起作用，.once 修饰符还能被用到自定义的**[组件事件](https://cn.vuejs.org/v2/guide/components.html" \l "使用-v-on-绑定自定义事件)**上。

缺点：一个事件只能绑定一个方法

## .sync语法糖

|  |
| --- |
| <comp :foo="bar" @update:foo="val => bar = val"></comp> |

这里update:foo中的:没有特殊含义，update:foo是一个整体字符串，只是一种约定的规则。

# 过渡动画

<transition> 的默认行为 - 进入和离开同时发生。(切换时产生两个元素，一个离开，一个进入)

# 其他指令

## v-model



所以父组件用v-model时，子组件需要用this.$emit('input', Number(formattedValue))，传递回去。

## v-bind

将数据与html属性绑定，数据改变时，html属性也会相应的改变（本来是静态的）

## 自定义指令

包括全局指令，组件内局部指令，通过钩子去执行。

不用钩子时会被两个钩子默认调用。

Vue.directive('my-directive', function () {

// 这里将会被 `bind` 和 `update` 调用

})

# mvvm

## 简单写法



## 经典理解

　Model：很简单，就是业务逻辑相关的数据对象，通常从数据库映射而来，我们可以说是与数据库对应的model。

　　View：也很简单，就是展现出来的用户界面。

　　基本上，绝大多数软件所做的工作无非就是从数据存储中读出数据，展现到用户界面上，然后从用户界面接收输入，写入到数据存储里面去。所以，对于数据存储(model)和界面(view)这两层，大家基本没什么异议。但是，如何把model展现到view上，以及如何把数据从view写入到model里，不同的人有不同的意见。

　　MVC派的看法是，界面上的每个变化都是一个事件，我只需要针对每个事件写一堆代码，来把用户的输入转换成model里的对象就行了，这堆代码可以叫controller。

　　而MVVM派的看法是，我给view里面的各种控件也定义一个对应的数据对象，这样，只要修改这个数据对象，view里面显示的内容就自动跟着刷新，而在view 里做了任何操作，这个数据对象也跟着自动更新，这样多美。所以：

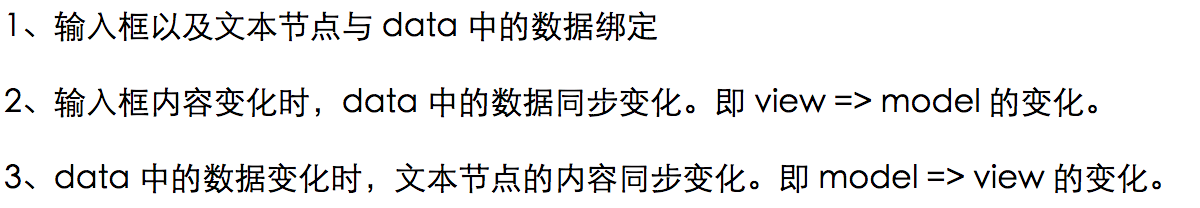
　　ViewModel：就是与界面(view)对应的Model。因为，数据库结构往往是不能直接跟界面控件一一对应上的，所以，需要再定义一个数据对象专门对应view上的控件。而ViewModel的职责就是把model对象封装成可以显示和接受输入的界面数据对象。

　　至于viewmodel的数据随着view自动刷新，并且同步到model里去，这部分代码可以写成公用的框架，不用程序员自己操心了。

　　简单的说，ViewModel就是View与Model的连接器，View与Model通过ViewModel实现双向绑定。

## 双向绑定实现

https://www.cnblogs.com/kidney/p/6052935.html?utm\_source=gold\_browser\_extension



根据这个原理：只是部分数据进入了视图层里，并不是this上的所有数据都进去了。比如$store,

## 其他

1.x以后v-repeat变成v-for