# **vuejs**

# **VUE基础语法学习文档**

王宇轩 from BUAA

2019.07

# 目 录

[一、引入vue框架](#_Toc15008_WPSOffice_Level1) [1](#_Toc15008_WPSOffice_Level1)

[二、基础功能](#_Toc31202_WPSOffice_Level1) [2](#_Toc31202_WPSOffice_Level1)

[2.1 前后端数据定向传输](#_Toc31202_WPSOffice_Level2) [2](#_Toc31202_WPSOffice_Level2)

[2.2 vue指令介绍](#_Toc8507_WPSOffice_Level2) [2](#_Toc8507_WPSOffice_Level2)

[2.2.1 指令简介](#_Toc31202_WPSOffice_Level3) [2](#_Toc31202_WPSOffice_Level3)

[2.2.1 部分指令具体内容介绍](#_Toc22966_WPSOffice_Level3) [4](#_Toc22966_WPSOffice_Level3)

[三、VUE框架基础部分讲解](#_Toc8507_WPSOffice_Level1) [18](#_Toc8507_WPSOffice_Level1)

[3.1 数据与方法](#_Toc30421_WPSOffice_Level2) [18](#_Toc30421_WPSOffice_Level2)

[3.1.1 变量赋值](#_Toc16966_WPSOffice_Level3) [18](#_Toc16966_WPSOffice_Level3)

[3.1.2 vue实例自身的属性方法](#_Toc32134_WPSOffice_Level3) [18](#_Toc32134_WPSOffice_Level3)

[3.1.3 vue对象的方法介绍](#_Toc29324_WPSOffice_Level3) [19](#_Toc29324_WPSOffice_Level3)

[3.1.4 实例生命周期](#_Toc2248_WPSOffice_Level3) [20](#_Toc2248_WPSOffice_Level3)

[3.2 vue模板语法介绍](#_Toc23747_WPSOffice_Level2) [23](#_Toc23747_WPSOffice_Level2)

[3.2.1 前端文本添加](#_Toc14663_WPSOffice_Level3) [23](#_Toc14663_WPSOffice_Level3)

[3.2.2 组件特性设置](#_Toc7756_WPSOffice_Level3) [23](#_Toc7756_WPSOffice_Level3)

[3.2.3 使用js代码](#_Toc13252_WPSOffice_Level3) [24](#_Toc13252_WPSOffice_Level3)

[3.3 计算属性和侦听器](#_Toc22966_WPSOffice_Level2) [24](#_Toc22966_WPSOffice_Level2)

[3.3.1 计算属性](#_Toc22520_WPSOffice_Level3) [24](#_Toc22520_WPSOffice_Level3)

[3.3.2 计算属性和方法的关系](#_Toc26361_WPSOffice_Level3) [25](#_Toc26361_WPSOffice_Level3)

[3.3.3 计算属性和监听属性](#_Toc2766_WPSOffice_Level3) [26](#_Toc2766_WPSOffice_Level3)

[3.3.4 计算属性的getter和setter](#_Toc8628_WPSOffice_Level3) [26](#_Toc8628_WPSOffice_Level3)

[3.3.5 侦听器的使用](#_Toc14078_WPSOffice_Level3) [27](#_Toc14078_WPSOffice_Level3)

[3.3.6 \_.类型的有关函数](#_Toc1436_WPSOffice_Level3) [30](#_Toc1436_WPSOffice_Level3)

[3.4 class和style绑定](#_Toc28120_WPSOffice_Level2) [30](#_Toc28120_WPSOffice_Level2)

[3.4.1 绑定html的class](#_Toc22785_WPSOffice_Level3) [30](#_Toc22785_WPSOffice_Level3)

[3.4.2 绑定html的style](#_Toc17302_WPSOffice_Level3) [32](#_Toc17302_WPSOffice_Level3)

[3.4.3 真值（truthy）介绍](#_Toc29754_WPSOffice_Level3) [33](#_Toc29754_WPSOffice_Level3)

[3.5 条件渲染](#_Toc2800_WPSOffice_Level2) [33](#_Toc2800_WPSOffice_Level2)

[3.6 列表渲染](#_Toc30691_WPSOffice_Level2) [34](#_Toc30691_WPSOffice_Level2)

[3.6.1 维护状态](#_Toc4571_WPSOffice_Level3) [34](#_Toc4571_WPSOffice_Level3)

[3.6.2 数组更新检测](#_Toc5710_WPSOffice_Level3) [35](#_Toc5710_WPSOffice_Level3)

[3.6.3 对象变更的检测](#_Toc14686_WPSOffice_Level3) [36](#_Toc14686_WPSOffice_Level3)

[3.6.4 显示过滤/排序后的结果](#_Toc9633_WPSOffice_Level3) [37](#_Toc9633_WPSOffice_Level3)

[3.7 事件处理](#_Toc12874_WPSOffice_Level2) [38](#_Toc12874_WPSOffice_Level2)

[3.7.1 监听和处理事件](#_Toc27762_WPSOffice_Level3) [38](#_Toc27762_WPSOffice_Level3)

[3.7.2 事件修饰符](#_Toc4450_WPSOffice_Level3) [40](#_Toc4450_WPSOffice_Level3)

[3.7.3 键盘鼠标事件获取](#_Toc28766_WPSOffice_Level3) [41](#_Toc28766_WPSOffice_Level3)

[四、VUE表单绑定有关](#_Toc30421_WPSOffice_Level1) [44](#_Toc30421_WPSOffice_Level1)

[4.1 v-model绑定简介](#_Toc12787_WPSOffice_Level2) [44](#_Toc12787_WPSOffice_Level2)

[4.2 不同类型数据的绑定](#_Toc20990_WPSOffice_Level2) [44](#_Toc20990_WPSOffice_Level2)

[4.3 值绑定详解](#_Toc31435_WPSOffice_Level2) [49](#_Toc31435_WPSOffice_Level2)

[4.4 其他修饰符](#_Toc10949_WPSOffice_Level2) [50](#_Toc10949_WPSOffice_Level2)

[4.5 自定义组件使用v-model](#_Toc3107_WPSOffice_Level2) [51](#_Toc3107_WPSOffice_Level2)

# 一、引入vue框架

使用vue框架，一种是利用完整的vue-cli进行（较难，熟悉node.js等构建工具时可以进行尝试），第二种是直接引入script文件接入vue的一些功能。

其中直接引入vue的两个生产版本如下：

1.开发版本：https://cdn.jsdelivr.net/npm/vue/dist/vue.js（包含所有功能及报错，并且是格式化的代码）

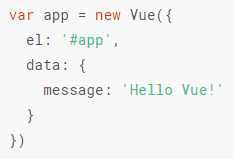
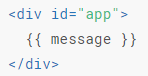
2.生产版本：https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js（最好是有一个确切的版本，来保证稳定性）

3.最新版本：https://cdn.jsdelivr.net/npm/vue（压缩版本，优化尺寸和速度）

# 二、基础功能

## 2.1 前后端数据定向传输

当前端有{{msg}}的内容时，如果该内容在一个被id或者class标识的组件中时，根据#id或者.class标识后，即可传输数据。



## 2.2 vue指令介绍

### 2.2.1 指令简介

指令是在引入vue框架后组件拥有的带有v-前缀的特殊特性，他们的预期赋值一般都是单个js表达式。

职责是当表达式的值改变时，将产生的连带影响表达在dom中，也就是显现在前端上。

**1. 指令的参数**

一个指令能接收一个参数，用:指定，例如v-bind:href，v-bind便可用于响应式的更新html特性。

v-on则是用于监听dom事件使用，和组件可添加的一些交互事件进行绑定。

动态参数：

在2.6.0版本新增的，可以用方括号括起来的js表达式作为一个指令的参数，例如：

此时attr便会被作为一个js表达式进行动态求值，相当于是将attr变量内的值放在冒号后面，例如当attr的值为”href”时，他就等于是v-bind:href了，可以实现js改变前端接收值得特性是什么。

一般动态参数应该有一个字符串的值，当赋值为null时则是异常情况，可被显性的用于移除绑定，如果是其他费字符串的值则会除法警告。

注：在动态参数中，一些字符例如空格和引号是无效的，所以注意，而且在dom中使用模板时也要注意回避大写键名（大写会在dom中全部转为小写）

**2. 指令的修饰符**

使用半角句号（英文句号）指明的特殊后缀，用于指出指令应该以什么特殊方式绑定，例如：

**3. 指令的缩写**

v-仅是在视觉提醒上表示这个vue提供的指令，但对于频繁使用的指令，v-就有些繁琐，同时在构建由 Vue 管理所有模板的单页面应用程序 (SPA - single page application) 时，v- 前缀也变得没那么重要了（都是使用的vue框架的话也没必要进行区分了）。因此，Vue 为 v-bind 和 v-on 这两个最常用的指令，提供了特定简写：

v-bind:href可缩写为:href



v-on:click可缩写为@click



而且:@也都是合法字符，所以能被正确解析。

### 2.2.1 部分指令具体内容介绍

**1. v-bind**

鼠标停留

通过为组件增加v-bind，可以绑定一个悬浮功能，进而可以为悬浮后页面的下一步操作进行设置，例如title就是提示消息，用title = “msg”即可实现，而msg即类似于{{msg}}，也是js中定义的数据。v-bind其实就是为当前组件的组成变量进行内容的重定义。



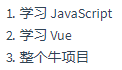


**2. v-for**

①．直接使用数组对象

循环结构，在for的””中指定元素和数组名，in和of均可，然后在添加数据内容时还是利用定位id然后设置的方法。





然后当数组中有多类数据时，直接加逗号然后进行字段名和内容的添加即可，每个数据内的元素都可看作一个json对象。

获得该数组时，只需要app4.todos即可，因为vue对象可以直接访问data字段中数据的。

在添加指令时，后面的todos是源数据数组，todo只是一个数据元素的别名。

在v-for块中，还可以访问所有父作用域的属性（data中的其他数据也可使用）；同时还支持一个可选的第二个参数，即当前项的索引（相当于从0开始的脚标）。（参数的内容只和顺序有关）

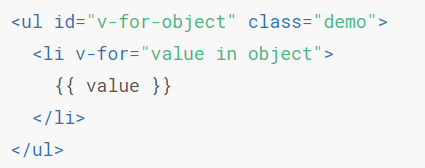
父作用域和索引的使用方法：

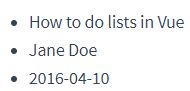




②．v-for中使用对象

这里对象的组成方式即为json对象的格式，按顺序排列着自己的字段名和值，因此可以使用v-for遍历一个json对象的属性，格式相同：



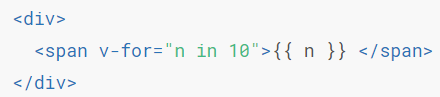


在对对象进行遍历时，第一个参数是属性的内容，第二个参数是键的名字，第三个是索引，也就是顺序，注意实际是脚标，所以是从0开始的。

注：遍历对象的键时，顺序是按Object.keys()的结果遍历，因此也不能保证它的结果在不同的js引擎下都一致，因为虽然方法同，但结果还是不一定相同。

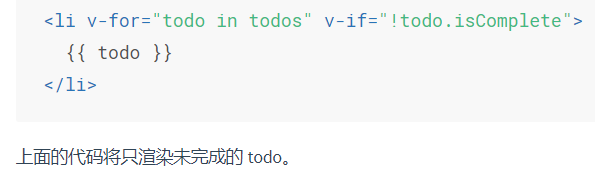
③．v-for的其他操作

可以将数组修改为一个整数，此时每一个元素既是从1一直循环到这个整数

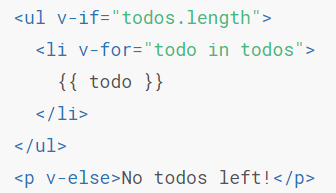


和v-if一样，在template中也可以使用v-for，效果和div中的相同。

在v-for和v-if同时使用时，如果位于一个节点，v-for的优先级高，所以v-if会重复运行于每个v-for循环中，当需要对每部分进行判断时，很有用



而当只是有条件的判断是否执行列表的渲染，就需要在v-for的外层元素上进行v-if的添加



④．自定义组件中使用v-for

在用for循环时，任何数据都不会传递到组件中，只是表面上进行使用而已，因此此时需要利用对象的prop来进行数据的传输：



不直接传入item的原因：会提高耦合性，而且这样明确数据来源也有利于重复使用。

todo列表的例子：



按顺序解释：

设定id用于添加数据

利用v-on指定表单提交的方法

label的for属性。。。加不换行的内容

然后添加输入框，和后台变量绑定，同时添加id和默认显示内容

按钮用于提交表单

然后为了显示列表内容设定ul

li和数组列表连接，获得元素和索引，然后为单个的显示的内容设定单独的key，title和监听元素删除后列表的特定变化（splice的用法？？）

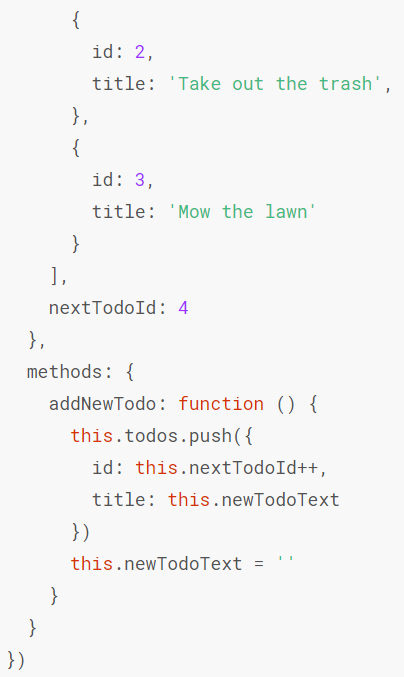
这里的is便是表示当前li元素其实是todo-item这个自定义组件，这样是在使用dom模板时必要的，因为在ul元素中只有li会被看做有效内容，因此这样不仅效果一样，而且还能避开一些潜在的浏览器解析错误

结束

解析dom模板的注意事项：

<https://cn.vuejs.org/v2/guide/components.html#%E8%A7%A3%E6%9E%90-DOM-%E6%A8%A1%E6%9D%BF%E6%97%B6%E7%9A%84%E6%B3%A8%E6%84%8F%E4%BA%8B%E9%A1%B9>





按顺序解释：

设定自己的组件

添加组建的html内容：文本和按钮，按钮同时添加了特定点击监听方法

设定一个新增的属性title和html代码中的title对应上

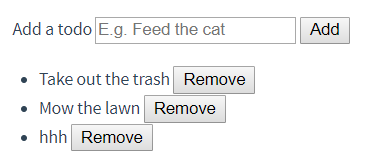
建立vue对象定位到传入数据的html组件中

data中设定todos列表和为了添加数据使用的下一个id以及下一个内容，下一个内容是和输入框连接的

设定当前列表可使用的一些方法，方法内容即添加新的元素，id和text都需要对应好，然后注意添加后需要重置文本的值

结束

效果图：



**3. v-on**

事件监听器，可调用在vue中定义的方法（method字段），同时还可以设定调用的情景，例如click点击等等

注意click后的字符串，可以是method内的方法名，也可以是裸露js代码，进行一些简单操作，例如修改变量的值等等。







在vue对象的方法中，this即相当于vue对象，通过this便可直接调用vue对象的数据进行处理，而且也不必触碰dom

**4. v-model**

可将表单输入和js中的内容进行实时绑定，只需为某个输入框添加model，然后model指向一个后端的变量即可实现输入框和另一个组件的信息同步显示，暂时认为仅可将输入框和显示组件进行匹配。





**5. v-once**

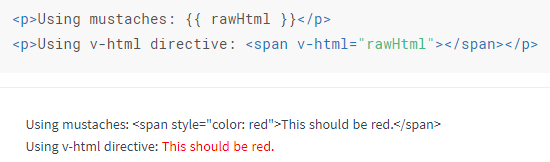
将组建的{{}}部分进行单次赋值的设定，当变量一次赋值后便不能被改变



**6. v-html**

使用：v-html=”变量”

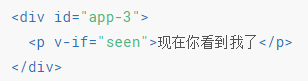
相当于将变量的内容放置于html页面中，同时还为视其为真正的html代码进行处理，如果直接利用{{变量}}向页面中传输内容的话，则仅会视其为普通的文本进行显示。



**7. v-if**

判断boolean类型变量，true显示，false不显示

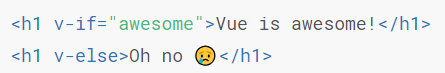
直接v-if = “msg”即可判断msg



在使用v-if指令时，除了直接加在原生组件上，还可以放在template组件上，template组件仅是作为一个临时的包裹使用，在使用时会自动被取消，仅显示内部的值。而且都会随着template的出现而出现，消失而消失。

**8.v-else**

在v-if判断变量并显示自身的同时，还可以用v-else添加一个else块，表示当if为错是显示的内容，在对的时候else自然是不会显示的。



注意有v-else指令的组件必须要紧跟在v-if或者v-else-if后面，才能起到和v-if匹配的效果。

**9.v-else-if**

2.1.0新增的，相当于在前一个if不成功时再进行的判断语句，即if(){}else{if(){}}中的第二个if()，同样需要跟在v-if或者v-else-if后面使用。

**10.v-show**

另一个根据条件展示元素的指令，并且和v-if的使用方法相同

和v-if的区别：

1.v-show不能在template中使用

2.v-show只是简单的修改组建的display属性，不会真正的销毁组件，组件还会被渲染并保留在dom中；而v-if时真正的渲染，会在显示和隐藏时队组建以及组建的事件监听器和子组件均进行销毁和重建

3.v-show不支持v-else指令的联合

4.v-show无论刚开始判断条件真假，组件都会被渲染，渲染后既是基于css的简单切换；而v-if只有当条件第一次变为真时，才进行第一次的渲染。

5.v-show有更高的初始渲染开销，v-if有更高的切换开销。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。

**999. 新组件的构建**

匿名构建一个vue对象然后对component进行操作即可

例如添加一个名为todo-item的组件，然后为组件添加特定的功能时，对组建后紧接着的{}进行当前组件部分属性的修改即可。

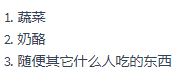
props：组件使用的变量

template：构成组件的html代码

然后和v-for的循环以及v-bind这个修改内容的指令结合，便可实现简易的循环展示以及不同部分有不同的效果。







解释：首先在js文件中定义了一个组件，组件是和自己的变量todo相结合的；然后在实际使用中，为了给todo进行赋值，所以先用v-for规定将要使用的变量，这里使用的变量所在的数组同样需要利用vue对象进行定向的规定；然后指定后就需要再为当前组件的prop进行映射，这里是由于todo变量是新建的，所以可以直接利用v-bind进行映射，映射成功后todo的操作便对应着item的操作；最后还可以为当前组件添加一些其他的特定功能（也就是非自定义prop的映射），例如每个item除了text还有一个id，那么就可以再利用v-bind进行指定，此时每个组件除了todo是特定的，key也是和变量相关联的了。

注：组件中使用指令进行赋值时，变量名一定是前端对应的，例如这个v-for的某元素名为item，后面在映射时就是利用item进行传值，然后当传值后，组件内部的处理就是利用自身的prop字段名进行处理了。

# 三、VUE框架基础部分讲解

## 3.1 数据与方法

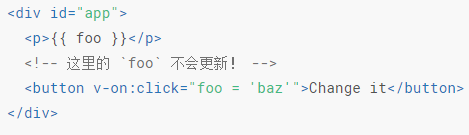
### 3.1.1 变量赋值

在前面我们可以看到vue对象中是可以通过字段名增加数据的，这里的数据同样可以是在js中定义或者获取到的变量，使用时直接用变量名表示即可。

不过当这样赋值后，两个位置的变量是共享的，也就是只传入了引用，因此当任意一个改变后，另一个也会进行相应的改变（仅针对实例的vue对象，匿名的不会有此效果）。

若要取消绑定，也就是互补影响，需要对将要传入的变量进行freeze方法处理，设变量名为foo，则需要调用方法：Object.freeze(foo)来冻结foo，此时foo或者foo内部的值就不能被同步改变了。





### 3.1.2 vue实例自身的属性方法

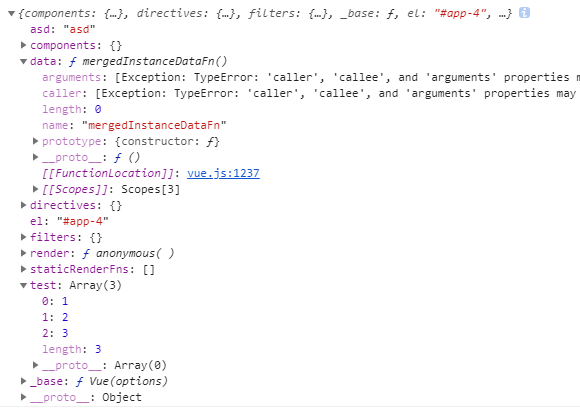
除了vue对象中的数据属性，vue还有一些实例属性和方法，在建立后他们会增加一个前缀$，来和用户的属性区分开，包括el，data属性和watch方法。

实例详介：

<https://cn.vuejs.org/v2/api/#%E5%AE%9E%E4%BE%8B%E5%B1%9E%E6%80%A7>

而若要访问data，则需要访问\_data，名字会自动改变，所以只有app4.\_data才能访问到这里的data数组，不过没必要多这一步。

当在vue对象中字自定义新的数据时，如果是自带的字段名，在定义后可以利用$字段名或者\_字段名直接访问；如果不是自带的，例如wyx这种字段名，会储存在vue对象的$options处：，options里储存的其实是该vue对象中的所有内容，包括非自带字段名数据、自带简单类型变量的内容或者地址。



### 3.1.3 vue对象的方法介绍

**1.watch实例方法**

调用时vue.$watch(‘a’,function(){})

当vue对象中属性a改变时，function方法便会被执行。

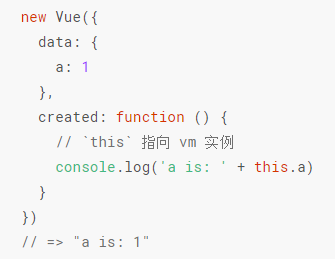
### 3.1.4 实例生命周期

一个vue对象在被建立后是有一个过程的，例如：建立对象、设置数据监听、编译模板、实例挂载和dom连接等等。

因此基于生命周期，vye对象有一些称为生命周期钩子的函数，他们分别会在vue对象的不同阶段被调用，生命周期函数中同样可以像method方法中那样，用this表示当前对象进行操作。

1.created

在vue被创建后执行代码：



其他的钩子函数：

mounted、updated 和 destroyed

注意不能在选项属性或回调上使用箭头函数，例如：

created:()=> console.log(this.a)或vm.$watch('a', newValue => this.myMethod())

因为箭头函数没有this，他会作为一个本地变量一直寻找，进而出现一些错误：Uncaught TypeError: Cannot read property of undefined 或 Uncaught TypeError: this.myMethod is not a function 之类的错误。

生命周期图示：





## 3.2 vue模板语法介绍

### 3.2.1 前端文本添加

mustache方法，即双大括号{{}}，以此将前端和后端进行连接

若要是规定该文本仅赋值以此，可利用v-once指令，此时在一次赋值后即使变量内容改变，显示也不会改变。

利用{{}}显示的均为直接的文本信息，不会解析html代码，因此可以为组件增加指令v-html=”变量名”，来使得当前组件的内容替换为变量名所包含的html代码所表示的内容。若直接{{变量名}}的话就会直接展示html代码。



注：使用html插值时，动态渲染的任意 HTML 可能会非常危险，因为它很容易导致 XSS 攻击。请只对可信内容使用 HTML 插值，绝不要对用户提供的内容使用插值。

### 3.2.2 组件特性设置

直接{{}}是不能对组件的一些特性进行设置的，如果需要进行设置，则应该利用v-bind进行处理。直接为组件添加指令：v-bind:特性 = “”，即可将当前组件的特性修改为引号中的值。

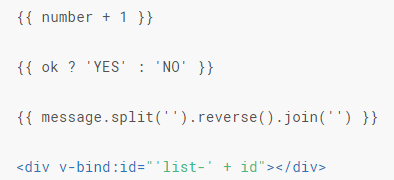




当特性的值是布尔类型时（默认，只要存在就是true），v-bind在对这类特性处理时，传入的变量值如果不是true而是null，undefined，false时，这个特性甚至不会存在于该组件中，因为只要存在就是true。

### 3.2.3 使用js代码

无论是{{}}还是指令后接的””中的内容，直接变量或者表达式即可，他们都会被作为js代码进行解析，不过注意，一个引号中仅可有一个表达式，因此不能使用定义，判断处理等语句和流控制



注：表达式只能访问部分全局变量，例如MATH和DATE，不应该在模板表达式中视图访问用户定义的全局变量。

## 3.3 计算属性和侦听器

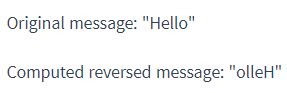
### 3.3.1 计算属性

在前端使用变量的计算式时，直接前端计算表达式太过麻烦，所以可以在vue对象的一个名为computed字段名的字段中，可以建立一个方法，来对js数据进行简易计算。





vm实例即全局



这里的reversedMessage便是一个计算属性（直接可以看做一个属性，它的值既是方法返回的值，并且调用时和正常的属性相同），直接调用这个计算属性即可得到message的逆转字符串，并且是始终和message匹配着。

### 3.3.2 计算属性和方法的关系

方法是需要有()来表示的，因为可能需要传入参数，并且在防止时也需要在method字段中，方法和计算属性本质是一样的，但是在缓存上有些区别，因为方法的话是需要每次调用函数的（不会有缓存，次次重新调用），而计算属性的话只要加载一次（因为是响应式依赖，只有变化才会重新加载）就能直接使用了。

不过注意，如果计算属性中有不是响应依赖的函数，那么计算属性就不会改变了，例如Data.now()，这种就需要用方法定义，每次调用时获得数据。

注：有一个性能开销比较大的计算属性 A，它需要遍历一个巨大的数组并做大量的计算。然后我们可能有其他的计算属性依赖于 A 。如果没有缓存，我们将不可避免的多次执行 A 的 getter，因为变化时每次都需要重新加载，并且生成新的缓存。如果你不希望有缓存，请用方法来替代。

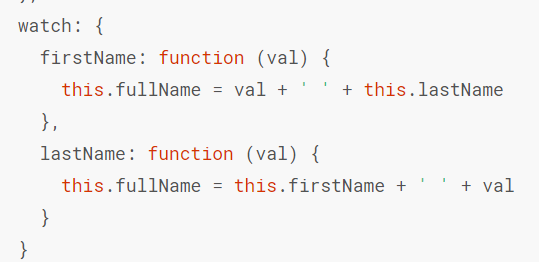
### 3.3.3 计算属性和监听属性

当需要监视数据的变化时，可能会用到watch，但很可能会滥用，此时可以使用计算属性而不是命令式的watch来进行数据的监听。

例如简易的两个变量构成一个变量时，只需将这个合成变量作为一个计算属性进行定义，然后和两个变量连接即可，否则还需要次次监听那两个变量，来对合成变量进行修改。



注：watch中对某个变量的监听器可以接收该变量数值修改后的值，可对这个值进行一些操作：



### 3.3.4 计算属性的getter和setter

如果在computed中直接变量:function(){}，则只是一个单独的getter，等于是变量:{get:function(){}}，除了get还可以有一个set，当为变量赋值时会执行set，当获取该变量时（即要显示变量时）会调用get方法：



### 3.3.5 侦听器的使用

watch选项是一个相比计算属性更加广泛，更通用响应数据变化的方法，=。当需要在数据变化时执行异步或开销较大的操作时，这个方式是最有用的。

例如：

将一个输入框和前端进行绑定，进行一些实时改变的操作







解释：

首先是按顺序

el：根据id绑定即将操作的div

data：定义数据，分别和输入框和前端展示的信息绑定

watch：设定监听的对象，这里是监听输入框的内容，也就是question，其中第一个参数是修改前的内容，第二个参数是修改后的参数，此时即可对内容进行判断或者操作等等，参数只和位置有关。

d：表示生命周期钩子，当对象被创建时运行的方法，这里既是设定

methods：定义的几个方法，getanswer是判断输入框的内容是否包括?，如果不包含则会显示特定字符然后返回。否则就会正常进行，例如首先修改内容为think表示思考，然后调用api即可。调用时可先用this将当前对象进行储存，来方便操作。例如使用axios的函数，先利用get寻到地址；然后用then进行get后的操作，这里是获取response，然后利用\_.的函数转化为yes和no进行输入；then之后还可以用catch来获取异常，方法是一样的用function进行处理即可。

### 3.3.6 \_.类型的有关函数

待学习

## 3.4 class和style绑定

首先对组件的class和id属性进行重新说明，当只有一个单词或者用-连接时，class和id仅由这一个字符串所代表的#或者.控制。

而当class由空格隔开的几个字符串隔开时，该组件可以同时被这隔开的多个.控制，当有重复的属性出现时会以最后出现的为主。而当id的字符串被这样隔开时，是不能同时被多个#控制的，而只能被带有空格的一整个字符串代表的#来进行控制。

### 3.4.1 绑定html的class

**1.传入变量**

直接v-bind:class=”变量名”，即可让这个组件拥有.变量名的格式规定。

**2.传入表达式**

通过传给v-bind:class一个对象，可以动态的切换class

例如：

此时class存在与否就和isActive这个属性的值相关联（如果isActive为真，则当前标签有名为active的class选择器，真值介绍在3.4.3）

而且这种class的指定方式也是可以和普通的class共存的

他们显示的顺序和在组件中出现的顺序相同，也就是class1 class2 class3这样同理的，当class后仅是变量时，就代表着一个名字；当后面是一个{}的表达式时，会根据冒号后的值决定字段名是否作为class使用。

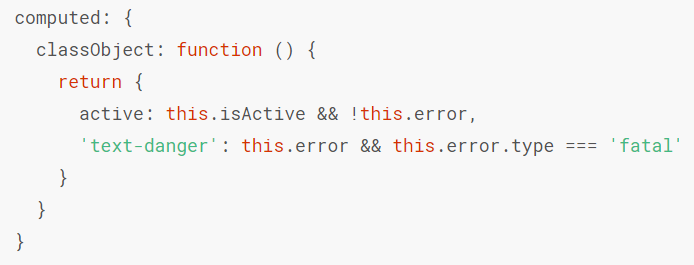
除了{}的表达式，还可以是三元表达式：

，判断第一个，如果真则就有第二个这个名称的选择器，否则是第三个。而’’则表示是空的，也就是不添加class。

**3.传入json对象**

如果是一个json对象的话，会将json对象的每个字段值进行判断，判断还是和{}同理，根据值决定字段名是否作为class使用。

当然无论是json对象还是一个变量，都可以用计算属性来进行说明，只需返回时按格式返回即可。正常变量直接return，如果要返回json对象只需：



可达到和直接传入json对象同样的效果。

**4.传入数组**



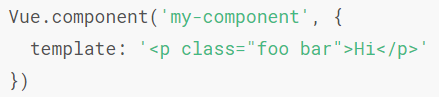


效果既是一样的

当然数组的各个位置，也可转化为{}的表达式或者直接三元表达式，也是一样的效果。

**5.用在自定义的组件中**

当在自定义组件中提前添加了class属性时，他们会在所有class属性的最前面，即：





等同于：越底层的标签越在前面，也越早起作用

其他用法都是相同的。

### 3.4.2 绑定html的style

**1.直接添加**

同样是利用v-bind:style=”{}”即可，{}中是和css样式中相同的json对象样式，其中各数值也同样可以用变量或表达式来进行定义。





**2.添加对象**

或者也可以和class一样，直接传入一个json对象，然后在vue对象的data中用一个同名变量来进行一些值的设定。





**3.添加数组**



只要保证数组内的每个元素均为style格式的json对象即可

**4.vue添加style的其他操作**

使用需要添加浏览器引擎前缀的 CSS 属性时，如 transform，Vue.js 会自动侦测并添加相应的前缀。

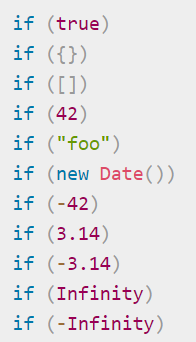
在2.3.0后可以为style中绑定的一个属性提供一个包含多个值的数组，常用于提供多个带前缀的值，例如：



此时只会渲染数组中最后一个被浏览器支持的值。在本例中，如果浏览器支持不带浏览器前缀的 flexbox，那么就只会渲染 display: flex，按顺序进行判断。

### 3.4.3 真值（truthy）介绍

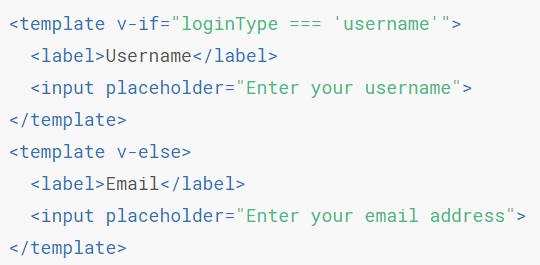
javascript中的真值，是在本该在布尔值判断的过程中，被转化为真值的值。js中假值之外的值均为真值，假值：false，0，””，null，undefined，NaN。js中的真值：

此时均会执行if后紧跟的操作。

## 3.5 条件渲染

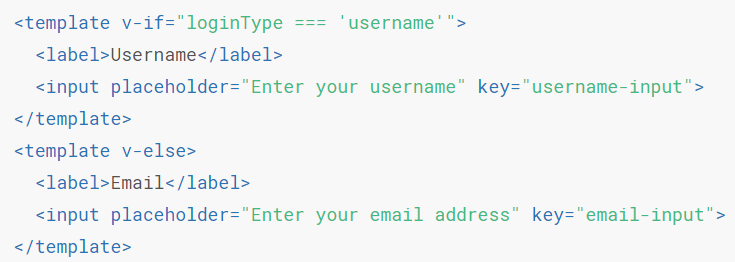
本部分是有关一些vue条件指令的使用介绍，可见2.2.1 部分指令具体内容介绍中的v-if，v-else，v-else-if，v-show的使用介绍。

在使用指令时还可以用key管理可复用的元素



例如当使用template切换不同输入框时，仅能实现修改文本内容和提示内容的，无法真正的改变输入框中的内容。

此时既是虽然修改了input，但是实际上将这两个元素视为了一个，也就是没有完全独立，而是进行了复用，为了实现独立，需要添加一个key指令。



此时js代码都不必改变也可实现两个输入框独立，也就是当切换时会重新渲染输入框，包括已经输入完成的值。而label由于没加key，所以还是高效复用的，不会重新渲染。

## 3.6 列表渲染

列表渲染既是渲染一个列表，一般是使用v-for来进行的

因此关于v-for将数组对应位一组元素的介绍，在指令详解中的v-for中可进行查看。

在利用v-for显示后，还需要一些其他的维护等操作进行，来保证未来正确的执行。

### 3.6.1 维护状态

在vue使用v-for渲染元素列表时，会默认使用“就地更新”的策略，也就是当数据项的顺序被改变，vue也不会移动dom元素来匹配数据项的顺序，而是就地更新每个元素，之确保它们在每个索引位置正确渲染，也就是不会重新加载。

为了给vue一个提示，以便它们能跟踪每个节点的身份，从而冲永和重新排序现有元素，最好为每项提供一个key，可以使用for循环中的临时数据的某个值来实现，所以最好提前设定好一个这样的属性。

注：在设定时最好使用字符串或者数值类型的值来当做key，而不是非基本类型值的数组，对象等。

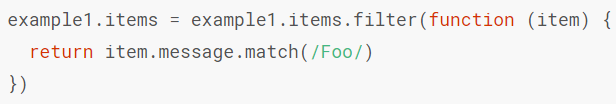
有关key的更多细节，可参考key的api文档。

<https://cn.vuejs.org/v2/api/#key>

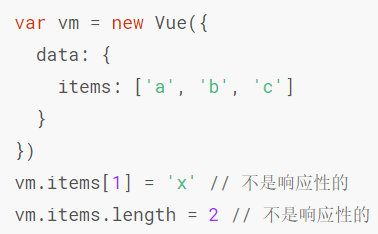
### 3.6.2 数组更新检测

数组的改变（不生成新数组，只是改变）：在vue中封装了一部分方法，包括：push，pop，shift，unshift，splice，sort，reverse

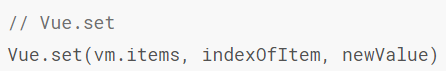
数组的替换（生成新数组，不改变原数组）：非变异方法，例如filter，concat，slice等等，使用这些函数时可以用新数组替换旧数组。

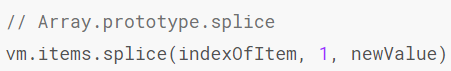


这种替换的操作其实还是很高效的，即这种用含有相同元素的数组去替换原来数组的操作。fliter筛选器还可用在显示内容的处理上。

注：不能检测到的变动：1.利用索引直接修改一个数组项时，2.当修改数组的长度时。items既是一个数组。

为了解决第一个问题，有两种方法可以实现：





同样也可用实例方法进行设定：



解决第二个问题，用splice可以实现：



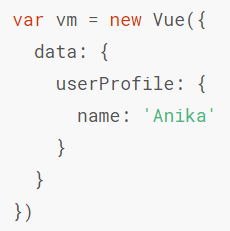
### 3.6.3 对象变更的检测

由于js的限制，vue是不能检测对象属性的添加和删除的

对于已经创建的实例对象，vue也不允许动态添加根级别的响应式属性。但是可以用vue.set方法向嵌套对象添加新的响应式属性。

方法介绍：

例子：

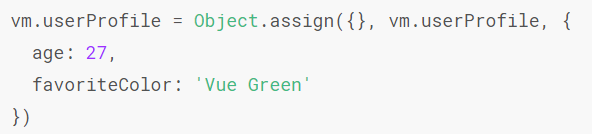


除了这个全局变量还可以用当前对象的实例方法进行同样的操作：



创建新值需要全局进行

而当为一个对象赋值多个新属性时，例如利用Object.assign()或\_.entend()方法，这种时候最好用一个临时对象和当前对象的属性合并创建一个新的对象，然后进行赋值，这样才能得到新的响应式属性。



这样添加的仅是普通属性：

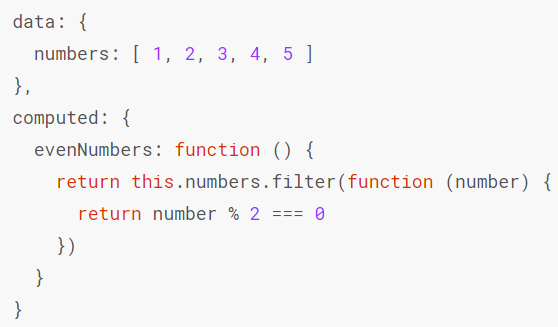


### 3.6.4 显示过滤/排序后的结果

也就是在显示时根据一定的方法进行筛选或者排序，然后再进行输出。

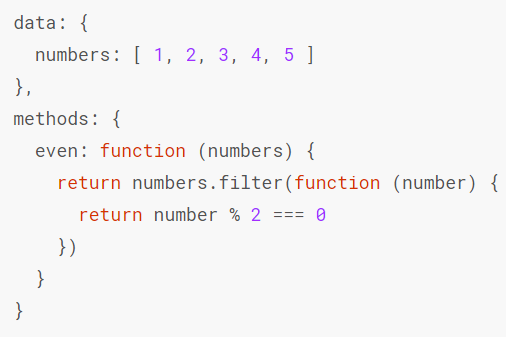
此时可以对数组建立一个计算属性来进行获得特定操作后的数组

例如：



此时evenNumbers即是numbers数组中所有2的倍数的汇总数组，制造方法即对数组使用filter方法，然后在方法内部对参数进行判断和返回（参数number可以换成其他的，代表的就是数组中的各个元素），返回的值便会被按顺序储存在新数组中。

或者使用方法也能达到一样的效果：



只是前者在调用时需要n in evenNumbers（数组已经经过了处理），后者是n in even(numbers)（方法调用再进行处理），因为后面的只要是数组对象即可，和得到的方法无关。

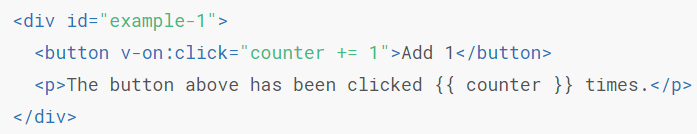
## 3.7 事件处理

### 3.7.1 监听和处理事件

利用v-on可以监听dom事件，并且在触发时运行一些js代码

这些代码可以是简单的计算式子，也可以是自己定义的方法的名字

计算式子的例子：



定义方法例子1：

在vue对象的method中定义即，然后直接将名字作为方法使用





这里的event是指原生的点击事件，每次有事件发生时都会有的，需要用参数获取。

定义方法例子2：

定义好方法后，也可以用js调用函数的方法，直接方法名(参数)进行调用





传入参数时也可以用特殊变量$event表示原始的dom事件，也就是点击鼠标移动等原生事件，传入方法中只需：



然后在对应方法中处理即可。

### 3.7.2 事件修饰符

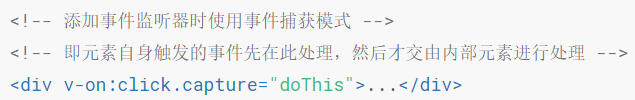
调用一些event.preventDefault() 或 event.stopPropagation()的方法是很常见的，因此vue还为v-on指令提供了事件修饰符，通过.指令来表示。

包括的有：stop，prevent，capture，self，once，passive等

他们分别表示的含义和使用方法如下：







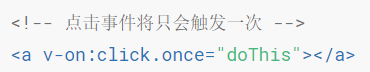


capture：先处理外部事件，内部事件也会被触发但是会后响应

self：只会处理自身的操作，即使有其他组件重叠的位置被操作，也不会去处理

2.1.4新增

以上的修饰符都是只能对原生的dom事件起作用的修饰符，而once修饰符还能用到自定义的组件事件上。



2.3.0新增

对应的是 addEventListener 中的 passive 选项

直接触发某事件的默认行为并且不会等待其完成（包括prevent），可提高移动端的性能，注意当prevent和passive一起使用时是无效的，因为passive也会使得prevent被忽略，而且浏览器可能会发送警告。

.passive会告诉浏览器不想阻止事件的默认行为，并且一直如此。



而且修饰符也可以串联，而且是按顺序进行的，先前面的再后面的



而且也不必非要添加处理事件，直接修饰也可以：



顺序例子：

v-on:click.prevent.self 会阻止所有的点击，而 v-on:click.self.prevent 只会阻止对元素自身的点击

### 3.7.3 键盘鼠标事件获取

**1.按键修饰符**

在监听键盘事件时，经常需要检查详细的按键，vue也为v-on在监听键盘事件时添加了按键修饰符

监听键盘事件：v-on:keyup

回车：.enter

而且也可以将 KeyboardEvent.key 暴露的任意有效按键名转换为 kebab-case 来作为修饰符

例如：



注：获得的键盘输入值会被储存在$event.key中。

**2.按键码**

修饰符之外同样也可以用按键码来进行事件的监听：

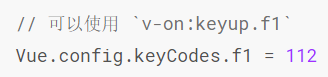
keyCode的用法已经被废弃了，可能不会被最新的浏览器支持，但是旧的还是可以的，使用方法即是：

同时为了在必要情况下支持旧浏览器，vue还提供了一些常用按键码的别名：

如enter，tab，delete（删除和退格），esc，space，up，down，left，right

注：一些按键，例如esc和所有方向键，在ie9中有不同的key值，所以最好使用别名来监听

也可以自定义按键修饰符别名，利用config.keyCodes对象可以实现：



**3.系统修饰符**

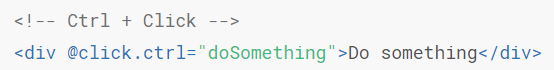
2.1.0新增的

可以用ctrl，alt，shift，meta实现对相应按键的监听

注：不同的操作系统有些按键对应的键盘按钮是不同的，例如mac中meta对应的是command，windows中meta对应的是微软微标键，sun中meta对应的是实心宝石键。其他键盘上meta多被标记为META或Meta

而且这些修饰符还可以实现多键同时按下的事件判断，例如：





而且这种修饰键和常规按键是不同的，例如和keyup事件一起使用时，事件触发时修饰键一定要是按下的状态，即只有在按住 ctrl 的情况下释放其它按键，才能触发 keyup.ctrl，单单释放 ctrl 也不会触发事件。如果你想要这样的行为，请为 ctrl 换用 keyCode：keyup.17。

2.5.0新增

.exact修饰符，允许控制由精确的系统修饰符组合触发的事件：



2.2.0新增

鼠标按键修饰符：left，right，middle，他们会限制处理函数响应鼠标的按钮

# 四、VUE表单绑定有关

## 4.1 v-model绑定简介

首先在引入了vue后，可以利用v-model将前端输入框直接和后端的变量进行双向绑定，包括input，textarea和select。

注：使用v-model后，会忽略所有表单元素的value，checked，selected等特性的初始值，而总是将vue实例的数据作为数据来源，设定初始值时应该在组件的data选项中声明初始值。

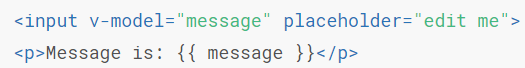
v-model指令在内部还会为不同的输入元素使用不同的属性并抛出不同的事件，例如：text 和 textarea 元素使用 value 属性和 input 事件；checkbox 和 radio 使用 checked 属性和 change 事件；select 字段将 value 作为 prop 并将 change 作为事件。

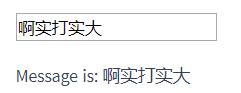
注：对一些需要使用输入法的语言，v-model不会在输入法组合文字过程中得到更新，如果需要处理这个事件，就需要使用input事件。

## 4.2 不同类型数据的绑定

**1.文本**

直接v-model绑定一个普通变量即可：





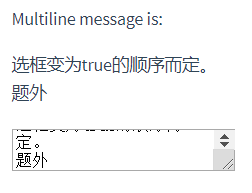
**2.多行文本**

此时直接按原样绑定一个变量的话，是不能完美展示各行内容的

而应该为显示多行文本处的元素样式进行修改，设置white-space: pre-line

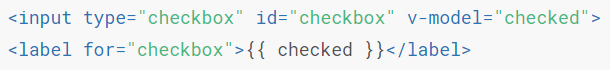
这样便可在和文本变量绑定后，保持换行的行为：





**3.复选框**

单个复选框的值可以和单个的布尔值进行绑定：



此时checked的值就会和复选框的值绑定，但是和文本框不同，这类input只有选中和未选中两种，所以checked的值就是true或者false。

对于多个复选框，可以和数组进行绑定：



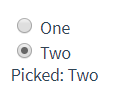


当这样利用v-model和数组绑定后，每次选择都会根据true和false判断是否将自己的value传入数组中，而且如果有多个true，传入数组内值的顺序就会按多选框变为true的顺序而定。

题外话：当直接在html中用{{}}的方式展示一个数组时，会直接显示数组在js中冒号后的部分，也就是[‘’,’’]这样的。

**4.单选按钮**





和多选框传入数组中的方法类似，可以为每个单选框设定一个value，此时单选框被选择时，自身的value值便会被传入v-model指向的变量中，此时直接展示该变量即可显示选择的值。

**5.单选选择框**

这里的选择框是指下拉框的那种

这类是基于select和option实现的：



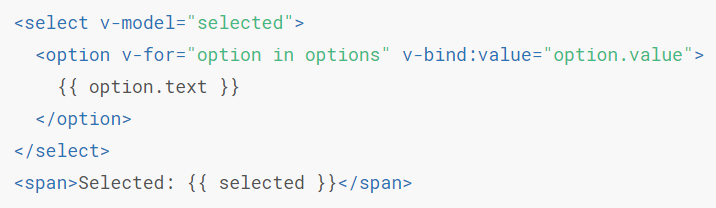


选择框在绑定变量时，只需要在最外的select处利用v-model进行绑定即可，此时传入变量的值：默认是option中的文本，如果option有一个value的话就会传入value的内容。

在select选择框组件中，可以为option添加一个disabled的名字，表示它是在select中默认的选择。

注：如果v-model表达式的初始值没有匹配任何一个选项，select元素将被渲染为“未选中”状态，在ios中，这会导致用户无法选择第一个选项，因为此时ios不会触发change事件，所以设定最开始的这个值为空时更合理的。

在直接显示选择框的内容之外，还可以用数组和select结合来进行动态选项的渲染：





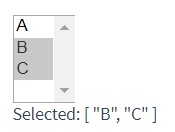
实际就是动态为option设置不同的value，然后代替option的内容传输到selected变量中再进行展示。

**6.多选选择框**

同样是绑定到一个数组中







选择框变为多选：增加一个multiple属性即可，此时是通过鼠标长按划范围的方式进行选择的。

和复选框中不同的，各个option传入数组的值是根据option出现的顺序来的：

## 4.3 值绑定详解

对于单选按钮，复选框和选择框的选项，v-model绑定的值通常是静态字符串（对复选框来说也可以是布尔值），此时绑定值的内容既是input或者option中自己的属性value对应的值。如果不存在value，则会默认是true，false或者option的内容。

当然如果需要绑定动态属性，用v-bind直接设定value的值也是可行的。

1.复选框

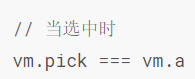


有特殊的true-value和false-value，可分别表示选中和未选中的值。

注：复选框的这两个属性和value是不会互相影响的，而且当表单提交时，不会包含未被选中的复选框，也就是false-value是不会被上传的，虽然vm中的值如此。如果需要一定上传一个值，需要换为单选按钮。

2.单选按钮

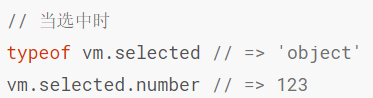




3.选择框

这里是使用的v-bind传入对象的方法，此时selected获得的值会是value的对象，只有selected.number时才能访问到123



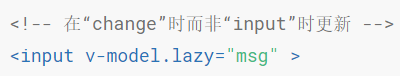


## 4.4 其他修饰符

v-model还有几个专用的修饰符，来进行一些额外功能的设定

**1. .lazy**

默认情况下v-model在每次input事件触发后都会进行同步，通过v-model.lazy可以实现，只有当使用change事件时才进行同步：



**2. .number**

需要将输入的值转为数值类型并且储存时，v-model.number可以实现。而且注意，如果不加这个，即使input的type是number，返回值时也会是字符串，是不能被parseFloat()等函数进行解析的。



**3. .trim**

通过该修饰符可以过滤掉用户输入的首尾空白字符然后进行储存



## 4.5 自定义组件使用v-model

参考自定义输入组件文档可对v-model和组件的结合有更深入的了解：

<https://cn.vuejs.org/v2/guide/components-custom-events.html#%E8%87%AA%E5%AE%9A%E4%B9%89%E7%BB%84%E4%BB%B6%E7%9A%84-v-model>