**[data](https://cn.vuejs.org/v2/api/" \l "data" \o "data)**

* **类型**：Object | Function
* **限制**：组件的定义只接受 function。
* **详细**：

Vue 实例的数据对象。Vue 将会递归将 data 的属性转换为 getter/setter，从而让 data 的属性能够响应数据变化。**对象必须是纯粹的对象 (含有零个或多个的 key/value 对)**：浏览器 API 创建的原生对象，原型上的属性会被忽略。大概来说，data 应该只能是数据 - 不推荐观察拥有状态行为的对象。

一旦观察过，不需要再次在数据对象上添加响应式属性。因此推荐在创建实例之前，就声明所有的根级响应式属性。

实例创建之后，可以通过 vm.$data 访问原始数据对象。Vue 实例也代理了 data 对象上所有的属性，因此访问 vm.a 等价于访问 vm.$data.a。

以 \_ 或 $ 开头的属性 **不会** 被 Vue 实例代理，因为它们可能和 Vue 内置的属性、API 方法冲突。你可以使用例如 vm.$data.\_property 的方式访问这些属性。

当一个**组件**被定义，data 必须声明为返回一个初始数据对象的函数，因为组件可能被用来创建多个实例。如果 data 仍然是一个纯粹的对象，则所有的实例将**共享引用**同一个数据对象！通过提供 data 函数，每次创建一个新实例后，我们能够调用 data 函数，从而返回初始数据的一个全新副本数据对象。

如果需要，可以通过将 vm.$data 传入 JSON.parse(JSON.stringify(...)) 得到深拷贝的原始数据对象。

**示例**：

var data = { a: 1 }

// 直接创建一个实例

var vm = new Vue({

data: data

})

vm.a // => 1

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

// Vue.extend() 中 data 必须是函数

var Component = Vue.extend({

data: function () {

return { a: 1 }

}

})

注意，如果你为 data 属性使用了箭头函数，则 this 不会指向这个组件的实例，不过你仍然可以将其实例作为函数的第一个参数来访问。

data: vm => ({ a: vm.myProp })

[**props**](https://cn.vuejs.org/v2/api/#props)

* **类型**：Array<string> | Object
* **详细**：

props 可以是数组或对象，用于接收来自父组件的数据。props 可以是简单的数组，或者使用对象作为替代，对象允许配置高级选项，如类型检测、自定义验证和设置默认值。

你可以基于对象的语法使用以下选项：

* + type: 可以是下列原生构造函数中的一种：String、Number、Boolean、Array、Object、Date、Function、Symbol、任何自定义构造函数、或上述内容组成的数组。会检查一个 prop 是否是给定的类型，否则抛出警告。Prop 类型的**[更多信息在此](https://cn.vuejs.org/v2/guide/components-props.html" \l "Prop-%E7%B1%BB%E5%9E%8B)**。
  + default: any  
    为该 prop 指定一个默认值。如果该 prop 没有被传入，则换做用这个值。对象或数组的默认值必须从一个工厂函数返回。
  + required: Boolean  
    定义该 prop 是否是必填项。在非生产环境中，如果这个值为 truthy 且该 prop 没有被传入的，则一个控制台警告将会被抛出。
  + validator: Function  
    自定义验证函数会将该 prop 的值作为唯一的参数代入。在非生产环境下，如果该函数返回一个 falsy 的值 (也就是验证失败)，一个控制台警告将会被抛出。你可以在**[这里](https://cn.vuejs.org/v2/guide/components-props.html" \l "Prop-%E9%AA%8C%E8%AF%81)**查阅更多 prop 验证的相关信息。
* **示例**：
* // 简单语法
* Vue.component('props-demo-simple', {
* props: ['size', 'myMessage']
* })
* // 对象语法，提供验证
* Vue.component('props-demo-advanced', {
* props: {
* // 检测类型
* height: Number,
* // 检测类型 + 其他验证
* age: {
* type: Number,
* default: 0,
* required: true,
* validator: function (value) {
* return value >= 0
* }
* }
* }

})

### [propsData](https://cn.vuejs.org/v2/api/#propsData)

* **类型**：{ [key: string]: any }
* **限制**：只用于 new 创建的实例中。
* **详细**：

创建实例时传递 props。主要作用是方便测试。

* **示例**：
* var Comp = Vue.extend({
* props: ['msg'],
* template: '<div>{{ msg }}</div>'
* })
* var vm = new Comp({
* propsData: {
* msg: 'hello'
* }

})

### [computed](https://cn.vuejs.org/v2/api/#computed)

* **类型**：{ [key: string]: Function | { get: Function, set: Function } }
* **详细**：

计算属性将被混入到 Vue 实例中。所有 getter 和 setter 的 this 上下文自动地绑定为 Vue 实例。

注意如果你为一个计算属性使用了箭头函数，则 this 不会指向这个组件的实例，不过你仍然可以将其实例作为函数的第一个参数来访问。

computed: {

aDouble: vm => vm.a \* 2

}

计算属性的结果会被缓存，除非依赖的响应式属性变化才会重新计算。注意，如果某个依赖 (比如非响应式属性) 在该实例范畴之外，则计算属性是**不会**被更新的。

* **示例**：
* var vm = new Vue({
* data: { a: 1 },
* computed: {
* // 仅读取
* aDouble: function () {
* return this.a \* 2
* },
* // 读取和设置
* aPlus: {
* get: function () {
* return this.a + 1
* },
* set: function (v) {
* this.a = v - 1
* }
* }
* }
* })
* vm.aPlus // => 2
* vm.aPlus = 3
* vm.a // => 2

vm.aDouble // => 4

### [methods](https://cn.vuejs.org/v2/api/#methods)

* **类型**：{ [key: string]: Function }
* **详细**：

methods 将被混入到 Vue 实例中。可以直接通过 VM 实例访问这些方法，或者在指令表达式中使用。方法中的 this 自动绑定为 Vue 实例。

注意，**不应该使用箭头函数来定义 method 函数** (例如 plus: () => this.a++)。理由是箭头函数绑定了父级作用域的上下文，所以 this 将不会按照期望指向 Vue 实例，this.a 将是 undefined。

* **示例**：
* var vm = new Vue({
* data: { a: 1 },
* methods: {
* plus: function () {
* this.a++
* }
* }
* })
* vm.plus()

vm.a // 2

### [watch](https://cn.vuejs.org/v2/api/#watch)

* **类型**：{ [key: string]: string | Function | Object | Array }
* **详细**：

一个对象，键是需要观察的表达式，值是对应回调函数。值也可以是方法名，或者包含选项的对象。Vue 实例将会在实例化时调用 $watch()，遍历 watch 对象的每一个属性。

* **示例**：
* var vm = new Vue({
* data: {
* a: 1,
* b: 2,
* c: 3,
* d: 4,
* e: {
* f: {
* g: 5
* }
* }
* },
* watch: {
* a: function (val, oldVal) {
* console.log('new: %s, old: %s', val, oldVal)
* },
* // 方法名
* b: 'someMethod',
* // 该回调会在任何被侦听的对象的 property 改变时被调用，不论其被嵌套多深
* c: {
* handler: function (val, oldVal) { /\* ... \*/ },
* deep: true
* },
* // 该回调将会在侦听开始之后被立即调用
* d: {
* handler: 'someMethod',
* immediate: true
* },
* e: [
* 'handle1',
* function handle2 (val, oldVal) { /\* ... \*/ },
* {
* handler: function handle3 (val, oldVal) { /\* ... \*/ },
* /\* ... \*/
* }
* ],
* // watch vm.e.f's value: {g: 5}
* 'e.f': function (val, oldVal) { /\* ... \*/ }
* }
* })

vm.a = 2 // => new: 2, old: 1

注意，**不应该使用箭头函数来定义 watcher 函数** (例如 searchQuery: newValue => this.updateAutocomplete(newValue))。理由是箭头函数绑定了父级作用域的上下文，所以 this 将不会按照期望指向 Vue 实例，this.updateAutocomplete 将是 undefined。

### [el](https://cn.vuejs.org/v2/api/#el)

* **类型**：string | Element
* **限制**：只在由 new 创建的实例中遵守。
* **详细**：

提供一个在页面上已存在的 DOM 元素作为 Vue 实例的挂载目标。可以是 CSS 选择器，也可以是一个 HTMLElement 实例。

在实例挂载之后，元素可以用 vm.$el 访问。

如果在实例化时存在这个选项，实例将立即进入编译过程，否则，需要显式调用 vm.$mount() 手动开启编译。

提供的元素只能作为挂载点。不同于 Vue 1.x，所有的挂载元素会被 Vue 生成的 DOM 替换。因此不推荐挂载 root 实例到 <html> 或者 <body> 上。

如果 render 函数和 template 属性都不存在，挂载 DOM 元素的 HTML 会被提取出来用作模板，此时，必须使用 Runtime + Compiler 构建的 Vue 库。