# computed

计算属性，有缓存，如果依赖的值不变，返回结果不变

|  |
| --- |
| <div id="example">  <p>Original message: "{{ message }}"</p>  <p>Computed reversed message: "{{ reversedMessage }}"</p> </div> |

|  |
| --- |
| var vm = new Vue({  el: '#example',  data: {  message: 'Hello'  },  computed: {  // 计算属性的 getter  reversedMessage: function () {  // `this` 指向 vm 实例  return this.message.split('').reverse().join('')  }  } }) |

# methods

方法，实时计算

|  |
| --- |
| <p>Reversed message: "{{ reversedMessage() }}"</p> |

|  |
| --- |
| // 在组件中 methods: {  reversedMessage: function () {  return this.message.split('').reverse().join('')  } } |

# watch

侦听，动态侦听数据的变化，不要滥用

|  |
| --- |
| <div id="demo">{{ fullName }}</div> |

|  |
| --- |
| var vm = new Vue({  el: '#demo',  data: {  firstName: 'Foo',  lastName: 'Bar',  fullName: 'Foo Bar'  },  watch: {  firstName: function (val) {  this.fullName = val + ' ' + this.lastName  },  lastName: function (val) {  this.fullName = this.firstName + ' ' + val  }  } }) |

上面代码是命令式且重复的。将它与计算属性的版本进行比较：

|  |
| --- |
| var vm = new Vue({  el: '#demo',  data: {  firstName: 'Foo',  lastName: 'Bar'  },  computed: {  fullName: function () {  return this.firstName + ' ' + this.lastName  }  } }) |

好得多了，不是吗？

虽然计算属性在大多数情况下更合适，但有时也需要一个自定义的侦听器。这就是为什么 Vue 通过 watch 选项提供了一个更通用的方法，来响应数据的变化。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：

|  |
| --- |
| <div id="watch-example">  <p>  Ask a yes/no question:  <input v-model="question">  </p>  <p>{{ answer }}</p> </div> |

|  |
| --- |
| <!-- 因为 AJAX 库和通用工具的生态已经相当丰富，Vue 核心代码没有重复 --> <!-- 提供这些功能以保持精简。这也可以让你自由选择自己更熟悉的工具。 --> <script src="https://cdn.jsdelivr.net/npm/axios@0.12.0/dist/axios.min.js"></script> <script src="https://cdn.jsdelivr.net/npm/lodash@4.13.1/lodash.min.js"></script> <script> var watchExampleVM = new Vue({  el: '#watch-example',  data: {  question: '',  answer: 'I cannot give you an answer until you ask a question!'  },  watch: {    // 如果 `question` 发生改变，这个函数就会运行  question: function (newQuestion, oldQuestion) {  this.answer = 'Waiting for you to stop typing...'  this.getAnswer()  }  },  methods: {    // `\_.debounce` 是一个通过 Lodash 限制操作频率的函数。    // 在这个例子中，我们希望限制访问 yesno.wtf/api 的频率    // AJAX 请求直到用户输入完毕才会发出。想要了解更多关于  // `\_.debounce` 函数 (及其近亲 `\_.throttle`) 的知识，  // 请参考：https://lodash.com/docs#debounce  getAnswer: \_.debounce(  function () {  if (this.question.indexOf('?') === -1) {  this.answer = 'Questions usually contain a question mark. ;-)'  return  }  this.answer = 'Thinking...'  var vm = this  axios.get('https://yesno.wtf/api')  .then(function (response) {  vm.answer = \_.capitalize(response.data.answer)  })  .catch(function (error) {  vm.answer = 'Error! Could not reach the API. ' + error  })  },  // 这是我们为判定用户停止输入等待的毫秒数  500  )  } }) </script> |

在这个示例中，使用 watch 选项允许我们执行异步操作 (访问一个 API)，限制我们执行该操作的频率，并在我们得到最终结果前，设置中间状态。这些都是计算属性无法做到的。

除了 watch 选项之外，您还可以使用命令式的 [vm.$watch API](https://cn.vuejs.org/v2/api/#vm-watch)

# [setter](https://cn.vuejs.org/v2/guide/computed.html#计算属性的-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 ：

|  |
| --- |
| // ... computed: {  fullName: {  // getter  get: function () {  return this.firstName + ' ' + this.lastName  },  // setter  set: function (newValue) {  var names = newValue.split(' ')  this.firstName = names[0]  this.lastName = names[names.length - 1]  }  } } // ... |

现在再运行 vm.fullName = 'John Doe' 时，setter 会被调用，vm.firstName 和 vm.lastName 也会相应地被更新。