Vue 计算属性和侦听属性

作者：lingyundu
原文链接：https://ld246.com/article/1621220441940
来源网站：链滴

计算属性

在模板内可以使用表达式，但是如果在模板中放入太多的逻辑会让模板变的复杂且难以维护。

为此，Vue 提供了 **计算属性。**对于大部分复杂逻辑，都应该使用 **计算属性。**

简单示例

计算属性的简单示例:

```
<!DOCTYPE html>
<html>

<head>
    <title>example</title>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>

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

</body>
<script>
    var vm = new Vue({
        el: '#example',
        data: {
            message: 'Hello'
        },
```

```
        // 计算属性在 `computed` 的对象中声明
        computed: {
            // 声明计算属性并指定一个函数
            reversedMessage: function () {
                // `this` 指向 vm 实例
                return this.message.split('').reverse().join('')
            }
        }
    })
</script>

</html>
```

该示例声明了一个计算属性：reversedMessage，并为该属性指定了一个函数。该函数将用作该属性 getter 函数，当通过 vm.reversedMessage 访问该属性时，调用的就是这个 getter 函数。

另外，这个vm.reversedMessage 依赖于 vm.message，当 vm.message 发生变化时，模板中所有赖 vm.reversedMessage 的绑定也会随之更新。

计算属性 vs 方法

上面的示例，通过在表达式中调用方法也可以达到相同的效果。

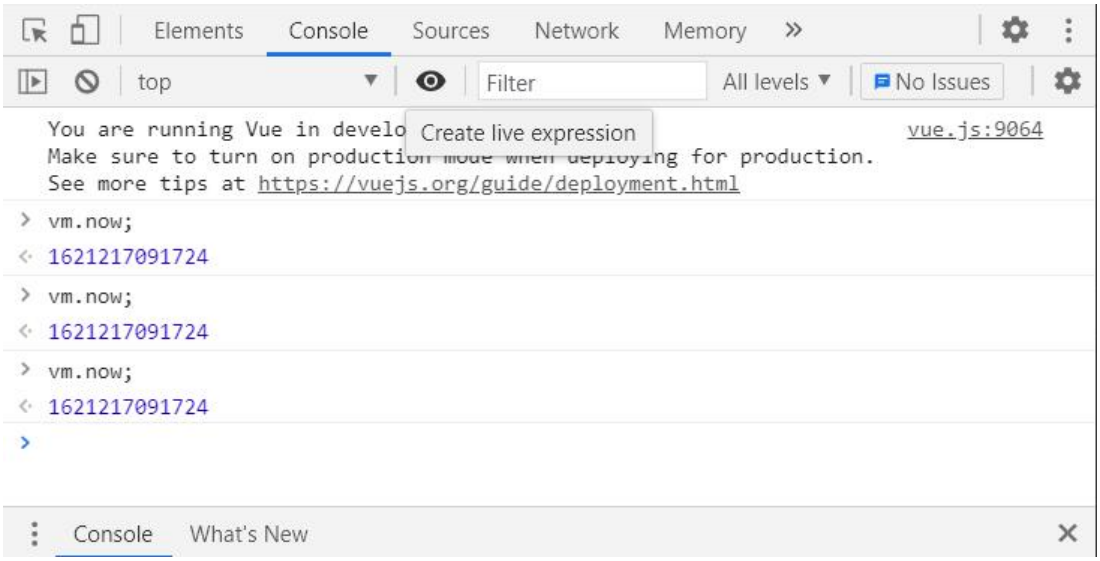
```
<!DOCTYPE html>
<html>

<head>
    <title>example</title>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>

<body>
    <div id="example">
        <p>Original message: "{{ message }}"</p>
        <p>Computed reversed message: "{{ reversedMessage() }}"</p>
    </div>

</body>
<script>
    var vm = new Vue({
        el: '#example',
        data: {
            message: 'Hello'
        },
        methods: {
            reversedMessage: function () {
                return this.message.split('').reverse().join('')
            }
        }
    })
</script>

</html>
```

二者的效果虽然相同，但是计算属性是基于响应式依赖进行缓存的。只有相关响应式依赖发生改变时计算属性才会重新求值。

对于非响应式依赖，计算属性将不再更新。比如：

```
computed: {
  now: function () {
    return Date.now()
  }
}
```

该示例中，Date.now() 不是响应式依赖，now 一旦被赋值，将不会再更新。



计算属性 vs 侦听属性

Vue 提供了一种更通用的方式来观察和响应 Vue 实例上的数据变动：**侦听属性**。如下面的例子：

```
<!DOCTYPE html>
<html>

<head>
    <title>demo</title>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>

<body>
    <div id="demo">{{ fullName }}</div>

</body>
<script>
    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
            }
        }
    })
</script>

</html>
```

上面的方式比较繁琐，改成计算属性版本会好很多：

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

计算属性的 setter

计算属性默认只有 getter，不过在需要时我们也可以给它提供一个 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]
    }
  }
}
// ...
```

侦听器

计算属性适用于大多数情况，但有时也需要定义一个侦听器。

示例:

```
<!DOCTYPE html>
<html>

<head>
    <title>watch example</title>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>

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

</body>
<!-- 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.debouncedGetAnswer()
            }
        },
        created: function () {
            // `_.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
            this.debouncedGetAnswer = _.debounce(this.getAnswer, 500)
        },
        methods: {
            getAnswer: 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
                })
        }
    }
})
</script>

</html>
```

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，还可以使用命令式的 vm.$watch API。