Vue的一些概念性运用

有点不理解：

**计算属性：**

这里我们声明了一个计算属性 reversedMessage。我们提供的函数将用作属性 vm.reversedMessage 的 getter 函数：

console.log(vm.reversedMessage) // => 'olleH'

vm.message = 'Goodbye'

console.log(vm.reversedMessage) // => 'eybdooG'

你可以打开浏览器的控制台，自行修改例子中的 vm。vm.reversedMessage 的值始终取决于 vm.message 的值。

Vue 知道 vm.reversedMessage 依赖于 vm.message，因此当 vm.message 发生改变时，所有依赖 vm.reversedMessage 的绑定也会更新。而且最妙的是我们已经以声明的方式创建了这种依赖关系：计算属性的 getter 函数是没有副作用 (side effect) 的，这使它更易于测试和理解。

**机损属性的缓存：methods和computed的区别**：

这里我们声明了一个计算属性 reversedMessage。我们提供的函数将用作属性 vm.reversedMessage 的 getter 函数：

console.log(vm.reversedMessage) // => 'olleH'

vm.message = 'Goodbye'

console.log(vm.reversedMessage) // => 'eybdooG'

你可以打开浏览器的控制台，自行修改例子中的 vm。vm.reversedMessage 的值始终取决于 vm.message 的值。

Methods和computed的结果是相同的：

不同的是**计算属性是基于它们的响应式依赖进行缓存的**。只在相关响应式依赖发生改变时它们才会重新求值。这就意味着只要 message 还没有发生改变，多次访问 reversedMessage计算属性会立即返回之前的计算结果，而不必再次执行函数。

Methods：是多次访问 reversedMessage计算属性会立即返回之前的计算结果，而不必再次执行函数。

Computed：每当触发重新渲染时，调用方法将**总会**再次执行函数。

2.class与style的绑定：

绑定html class：

1. 对象语法

<div v-bind:class="{ active: isActive }"></div>

1. 数组语法

<div v-bind:class="[activeClass, errorClass]"></div>

data: {

activeClass: 'active',

errorClass: 'text-danger'

}

1. 用在组件上

如果你声明了这个组件：

Vue.component('my-component', {

template: '<p class="foo bar">Hi</p>'

})

然后在使用它的时候添加一些 class：

<my-component class="baz boo"></my-component>

HTML 将被渲染为:

<p class="foo bar baz boo">Hi</p>

[**v-if vs v-show**](https://cn.vuejs.org/v2/guide/conditional.html#v-if-vs-v-show)

v-if 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。

v-if 也是**惰性的**：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

相比之下，v-show 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。

一般来说，v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。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。**

**当 v-if 与 v-for 一起使用时，v-for 具有比 v-if 更高的优先级，所以不推荐v-if和v-for同时使用，**可以将 v-if 置于外层元素 (或 [**<template>**](https://cn.vuejs.org/v2/guide/conditional.html#%E5%9C%A8-lt-template-gt-%E4%B8%AD%E9%85%8D%E5%90%88-v-if-%E6%9D%A1%E4%BB%B6%E6%B8%B2%E6%9F%93%E4%B8%80%E6%95%B4%E7%BB%84))上。

**尽可能在使用 v-for 时提供 key attribute，除非遍历输出的 DOM 内容非常简单，或者是刻意依赖默认行为以获取性能上的提升。**

常用的数组的方法：

数组的三个操作方法concat()、slice()和splice()。concat()和slice()方法都不会影响原数组，会在原数组上构建出一个新数组。其中concat()方法在原数组末尾添加所传的数组项（简单点看有类似于push()方法），构建一个新数组；slice()方法可以从指定的位置开始删除指定的数组项，并且将删除的数组项构建成一个新数组。splice()方法就更强大了，可以对一个数组做删除、插入和替换。而且splice()方法还会影响原数组，并且将返回的值构建出一个新数组。

[**事件修饰符**](https://cn.vuejs.org/v2/guide/events.html#%E4%BA%8B%E4%BB%B6%E4%BF%AE%E9%A5%B0%E7%AC%A6)

**Vue.js 为 v-on 提供了事件修饰符。之前提过，修饰符是由点开头的指令后缀来表示的。**

* .stop
* .prevent
* .capture
* .self
* .once
* .passive

<!-- 阻止单击事件继续传播 -->

<a v-on:click.stop="doThis"></a>

<!-- 提交事件不再重载页面 -->

<form v-on:submit.prevent="onSubmit"></form>

<!-- 修饰符可以串联 -->

<a v-on:click.stop.prevent="doThat"></a>

<!-- 只有修饰符 -->

<form v-on:submit.prevent></form>

<!-- 添加事件监听器时使用事件捕获模式 -->

<!-- 即元素自身触发的事件先在此处理，然后才交由内部元素进行处理 -->

<div v-on:click.capture="doThis">...</div>

<!-- 只当在 event.target 是当前元素自身时触发处理函数 -->

<!-- 即事件不是从内部元素触发的 -->

<div v-on:click.self="doThat">...</div>

使用修饰符时，顺序很重要；相应的代码会以同样的顺序产生。因此，用 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。

.exact 修饰符允许你控制由精确的系统修饰符组合触发的事件。

<!-- 即使 Alt 或 Shift 被一同按下时也会触发 -->

<button @click.ctrl="onClick">A</button>

<!-- 有且只有 Ctrl 被按下的时候才触发 -->

<button @click.ctrl.exact="onCtrlClick">A</button>

<!-- 没有任何系统修饰符被按下的时候才触发 -->

<button @click.exact="onClick">A</button>

表单的输入绑定：

用 v-model 指令在表单 <input>、<textarea> 及 <select> 元素上创建双向数据绑定。

### [.lazy](https://cn.vuejs.org/v2/guide/forms.html#lazy)

在默认情况下，v-model 在每次 input 事件触发后将输入框的值与数据进行同步 (除了[**上述**](https://cn.vuejs.org/v2/guide/forms.html#vmodel-ime-tip)输入法组合文字时)。你可以添加 lazy 修饰符，从而转变为使用 change 事件进行同步：

<!-- 在“change”时而非“input”时更新 -->

<input v-model.lazy="msg" >

### [.number](https://cn.vuejs.org/v2/guide/forms.html#number)

如果想自动将用户的输入值转为数值类型，可以给 v-model 添加 number 修饰符：

<input v-model.number="age" type="number">

这通常很有用，因为即使在 type="number" 时，HTML 输入元素的值也总会返回字符串。如果这个值无法被 parseFloat() 解析，则会返回原始的值。

### [.trim](https://cn.vuejs.org/v2/guide/forms.html#trim)

如果要自动过滤用户输入的首尾空白字符，可以给 v-model 添加 trim 修饰符：

<input v-model.trim="msg">