# 过滤器

Vue.js 允许你自定义过滤器，可被用于一些常见的文本格式化。

过滤器可以用在两个地方：双花括号插值和 v-bind 表达式 (后者从 2.1.0+ 开始支持)。过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符号指示：

<!-- 在双花括号中 -->  
{{ message | capitalize }}

<!-- 在 `v-bind` 中 -->

<div v-bind:id="rawId | formatId"></div>

## 局部过滤器

补零，首字母大写，四舍五入，

你可以在一个组件的选项中定义本地的过滤器：

<**div id="app"**>  
  
 <**h1**>{{ 12.34567 | fixed }}</**h1**>  
</**div**>  
<**script type="text/javascript"**>  
 **let *vm*** = **new** Vue({  
 **el**: **'#app'**,  
 **data**: {  
 **message**: **'javascript'** },  
 **filters**: {  
   
 fixed(value) {  
 **if** (*isNaN*(value)) {  
 **return ''** } **else** {  
 **return** value.toFixed(2)  
 }  
 }  
 }  
 });  
</**script**>

## 全局过滤器

或者在创建 Vue 实例之前全局定义过滤器：

<**div id="app"**>  
  
 <**h1**>{{ 11 | addZero }}</**h1**>  
  
</**div**>  
<**script type="text/javascript"**>  
  
  
 Vue.filter(**'addZero'**, **function** (value) {  
 **if** (value < 10) {  
 **return '0'** + value;  
 } **else** {  
 **return ""** + value;  
 }  
 });  
  
  
  
 **let *vm*** = **new** Vue({  
 **el**: **'#app'**,  
 **data**: {  
 **message**: **'javascript'** }  
 });  
</**script**>

# 计算属性

模板内的表达式非常便利，但是设计它们的初衷是用于简单运算的。  
在模板中放入太多的逻辑会让模板过重且难以维护。例如：

<div id="example">

{{ message.split('').reverse().join('') }}

</div>

## 关于计算属性

计算属性定义在computed中，它不是方法，属性的值是函数的返回值。  
把对数据处理的逻辑抽离在计算属性中，使得模板更加轻量易读  
计算属性的值会被缓存，并根据依赖的数据变化而重新计算

get和set：  
只给计算属性一个函数，默认这个函数是在取值的时候触发的，这个函数是get()函数  
如果需要设置set()函数，那么就要把计算属性设置为一个对象

取值：触发get函数，默认设置的函数  
赋值：触发set函数

计算属性也是属性，挂载在实例上，可以使用vm实例直接访问  
计算属性对应的函数中的this都指向实例

定义了计算属性，没有使用，那么不会执行计算属性这个函数

## 计算属性的使用

所以，对于任何复杂逻辑，你都应当使用计算属性。这里我们声明了一个计算属性 newmsg。我们提供的函数将用作属性 vm.newmsg 的 getter 函数：

<**div id="box"**>  
 <**p**>原始信息{{ msg }}</**p**>  
 <**p**>计算后的反转信息：{{ newmsg }}</**p**>  
 <**p**>{{list}}</**p**>  
 <**p**>{{newList}}</**p**>  
</**div**>  
<**script**>  
 **var *vm*** = **new** Vue({  
 **el**: **"#box"**,  
 **data**: {  
 **msg**: **'hello vue.js'**,  
 **list**: [1,2,3,4,5]  
 },  
 **computed**: {  
 *// 计算属性的getter* newmsg: **function** () {  
 **return this**.**msg**.split(**""**).reverse().join(**""**);  
 },  
 newList(){  
 **return this**.**list**.map((item) => {  
 **return** item \* 2  
 }).filter((item) => {  
 **return** item < 8  
 })  
 }  
 },  
 });  
 ***console***.log(***vm***.**msg**);  
 ***console***.log(***vm***.newmsg);  
</**script**>

## 计算属性和方法的区别

你可能已经注意到我们可以通过在表达式中调用方法来达到同样的效果：

<**div id="box"**>  
 <**p**>反转信息：{{ reversemsg() }}</**p**>  
</**div**>  
<**script**>  
 **var *vm*** = **new** Vue({  
 **el**: **"#box"**,  
 **data**: {  
 **msg**: **'hello vue.js'** },  
 **methods**: {  
 reversemsg: **function**(){  
 **return this**.**msg**.split(**""**).reverse().join(**""**);  
 }  
 },  
 });  
</**script**>

我们可以将同一函数定义为一个方法而不是一个计算属性。两种方式的最终结果确实是完全相同的。然而，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。

相比之下，每当触发重新渲染时，调用方法将总会再次执行函数。

我们为什么需要缓存？假设我们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！如果你不希望有缓存，请用方法来替代。

<**div id="box"**>  
 <**p**>计算后的反转信息：{{newmsg}}</**p**>  
 <**p**>计算后的反转信息：{{newmsg}}</**p**>  
</**div**>  
<**script**>  
 **let *cnt*** = 1;  
 **let *vm*** = **new** Vue({  
 **el**: **'#box'**,  
 **data**: {  
 **msg**: **'hello'** },  
 **computed**: {  
 newmsg: **function**() {  
 ***cnt***++;  
 **return *cnt*** + **this**.**msg**.split(**''**).reverse().join(**''**);  
 }  
 },  
 });  
</**script**>

结果：  
2olleh  
2olleh

<**div id="box"**>  
 <**p**>计算后的反转信息：{{newmsg()}}</**p**>  
 <**p**>计算后的反转信息：{{newmsg()}}</**p**>  
</**div**>  
<**script**>  
 **let *cnt*** = 1;  
 **let *vm*** = **new** Vue({  
 **el**: **'#box'**,  
 **data**: {  
 **msg**: **'hello'** },  
 **methods**: {  
 newmsg: **function**() {  
 ***cnt***++;  
 **return *cnt*** + **this**.**msg**.split(**''**).reverse().join(**''**);  
 }  
 },  
 });  
</**script**>

结果：  
2olleh  
3olleh

可以看到同样是 return ，计算属性调用不用括号，方法调用要加小括号

<**div id="app"**>  
 <**input type="text" v-model="message"** />  
 <**p**>{{message}}</**p**>  
 <**p**>{{reverserMessage}}</**p**>  
</**div**>  
<**script**>  
  
 **let *vm*** = **new** Vue({  
 **el**: **'#app'**,  
 **data**:{  
 **message**: **'hello vue'**,  
 **list**: [1,2,3,4,5]  
 },  
 **computed** : {  
 *// 计算属性中依赖了data中的数据，data中的数据发生变化，计算属性会重新计算* reverserMessage(){  
 **return this**.**message**.split(**''**).reverse().join(**''**)  
 }  
 }  
 })  
  
 *//console.log(vm);*</**script**>

## 计算属性和过滤器的区别

计算属性定义在vue实例中，只能在当前vue实例中使用；  
过滤器可以定义在vue实例中，也可以定义在vue实例外（全局过滤器），在多个vue实例中调用。

计算属性的结果有缓存，；过滤器的结果没有缓存

共同点：都必须有返回值

## 计算属性和watch的区别

fullName依赖于firstName和lastName，如果我们要用watch，那么就要监听两个数据

<**div id="box"**>  
 <**p**>姓：<**input type="text" v-model="firstName"**></**p**>  
 <**p**>名：<**input type="text" v-model="lastName"**></**p**>  
 <**p**>{{fullName}}</**p**>  
</**div**>  
<**script**>  
 **let *vm*** = **new** Vue({  
 **el**: **'#box'**,  
 **data**: {  
 **firstName**: **'张'**,  
 **lastName**: **'三'**,  
 **fullName**: **"张三"** },  
 **watch**: {  
 *// watch监控data中的firstName和lastName  
 // 当firstName和lastName值改变时就会把新值和旧值传递给回调函数* **'firstName'**:**function**(newval,oldval){  
 *// 参数newval代表新值，参数oldval代表旧值* ***console***.log(newval,oldval);  
 **this**.**fullName** = **this**.firstName + **this**.lastName;  
 },  
 **'lastName'**:**function**(newval,oldval){  
 ***console***.log(newval,oldval);  
 **this**.**fullName** = **this**.firstName + **this**.lastName;  
 }  
 },  
 });  
</**script**>

# 监听器

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。

<**div class="box"**>  
 千米： <**input type="text" v-model="kilometers"**>  
 米： <**input type="text" v-model="meters"**>  
</**div**>  
<**script**>  
 **new** Vue({  
 **el**: **'.box'**,  
 **data**: {  
 **kilometers**: 0,  
 **meters**: 0  
 },  
 **watch**: {  
 kilometers: **function**(val){  
 **this**.meters = val \* 1000  
 },  
 meters: **function**(val,oldval){  
 **this**.kilometers = val / 1000  
 }  
 },  
 })  
</**script**>