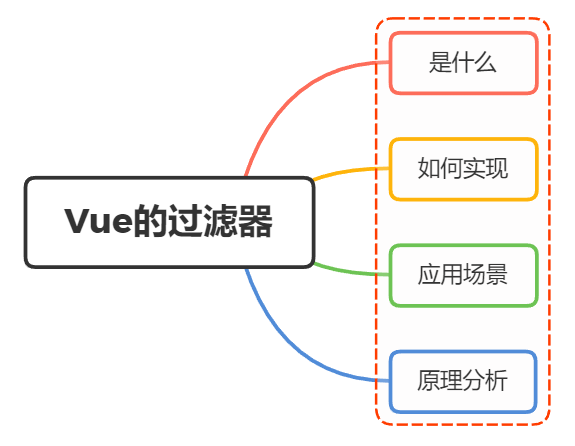
# 面试官：Vue中的过滤器了解吗？过滤器的应用场景有哪些？



## 一、是什么

过滤器（filter）是输送介质管道上不可缺少的一种装置

大白话，就是把一些不必要的东西过滤掉

过滤器实质不改变原始数据，只是对数据进行加工处理后返回过滤后的数据再进行调用处理，我们也可以理解其为一个纯函数

Vue 允许你自定义过滤器，可被用于一些常见的文本格式化

ps: Vue3中已废弃filter

## 二、如何用

vue中的过滤器可以用在两个地方：双花括号插值和 v-bind 表达式，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>

### 定义filter

在组件的选项中定义本地的过滤器

filters: {  
 capitalize: function (value) {  
 if (!value) return ''  
 value = value.toString()  
 return value.charAt(0).toUpperCase() + value.slice(1)  
 }  
}

定义全局过滤器：

Vue.filter('capitalize', function (value) {  
 if (!value) return ''  
 value = value.toString()  
 return value.charAt(0).toUpperCase() + value.slice(1)  
})  
  
new Vue({  
 // ...  
})

注意：当全局过滤器和局部过滤器重名时，会采用局部过滤器

过滤器函数总接收表达式的值 (之前的操作链的结果) 作为第一个参数。在上述例子中，capitalize 过滤器函数将会收到 message 的值作为第一个参数

过滤器可以串联：

{{ message | filterA | filterB }}

在这个例子中，filterA 被定义为接收单个参数的过滤器函数，表达式 message 的值将作为参数传入到函数中。然后继续调用同样被定义为接收单个参数的过滤器函数 filterB，将 filterA 的结果传递到 filterB 中。

过滤器是 JavaScript函数，因此可以接收参数：

{{ message | filterA('arg1', arg2) }}

这里，filterA 被定义为接收三个参数的过滤器函数。

其中 message 的值作为第一个参数，普通字符串 'arg1' 作为第二个参数，表达式 arg2 的值作为第三个参数

举个例子：

<div id="app">  
 <p>{{ msg | msgFormat('疯狂','--')}}</p>  
</div>  
  
<script>  
 // 定义一个 Vue 全局的过滤器，名字叫做 msgFormat  
 Vue.filter('msgFormat', function(msg, arg, arg2) {  
 // 字符串的 replace 方法，第一个参数，除了可写一个 字符串之外，还可以定义一个正则  
 return msg.replace(/单纯/g, arg+arg2)  
 })  
</script>

### 小结：

* 部过滤器优先于全局过滤器被调用
* 一个表达式可以使用多个过滤器。过滤器之间需要用管道符“|”隔开。其执行顺序从左往右

## 三、应用场景

平时开发中，需要用到过滤器的地方有很多，比如单位转换、数字打点、文本格式化、时间格式化之类的等

比如我们要实现将30000 => 30,000，这时候我们就需要使用过滤器

Vue.filter('toThousandFilter', function (value) {  
 if (!value) return ''  
 value = value.toString()  
 return .replace(str.indexOf('.') > -1 ? /(\d)(?=(\d{3})+\.)/g : /(\d)(?=(?:\d{3})+$)/g, '$1,')  
})

## 四、原理分析

使用过滤器

{{ message | capitalize }}

在模板编译阶段过滤器表达式将会被编译为过滤器函数，主要是用过parseFilters，我们放到最后讲

\_s(\_f('filterFormat')(message))

首先分析一下\_f：

\_f 函数全名是：resolveFilter，这个函数的作用是从this.$options.filters中找出注册的过滤器并返回

// 变为  
this.$options.filters['filterFormat'](message) // message为参数

关于resolveFilter

import { indentity,resolveAsset } from 'core/util/index'   
  
export function resolveFilter(id){  
 return resolveAsset(this.$options,'filters',id,true) || identity  
}

内部直接调用resolveAsset，将option对象，类型，过滤器id，以及一个触发警告的标志作为参数传递，如果找到，则返回过滤器；

resolveAsset的代码如下：

export function resolveAsset(options,type,id,warnMissing){ // 因为我们找的是过滤器，所以在 resolveFilter函数中调用时 type 的值直接给的 'filters',实际这个函数还可以拿到其他很多东西  
 if(typeof id !== 'string'){ // 判断传递的过滤器id 是不是字符串，不是则直接返回  
 return   
 }  
 const assets = options[type] // 将我们注册的所有过滤器保存在变量中  
 // 接下来的逻辑便是判断id是否在assets中存在，即进行匹配  
 if(hasOwn(assets,id)) return assets[id] // 如找到，直接返回过滤器  
 // 没有找到，代码继续执行  
 const camelizedId = camelize(id) // 万一你是驼峰的呢  
 if(hasOwn(assets,camelizedId)) return assets[camelizedId]  
 // 没找到，继续执行  
 const PascalCaseId = capitalize(camelizedId) // 万一你是首字母大写的驼峰呢  
 if(hasOwn(assets,PascalCaseId)) return assets[PascalCaseId]  
 // 如果还是没找到，则检查原型链(即访问属性)  
 const result = assets[id] || assets[camelizedId] || assets[PascalCaseId]  
 // 如果依然没找到，则在非生产环境的控制台打印警告  
 if(process.env.NODE\_ENV !== 'production' && warnMissing && !result){  
 warn('Failed to resolve ' + type.slice(0,-1) + ': ' + id, options)  
 }  
 // 无论是否找到，都返回查找结果  
 return result  
}

下面再来分析一下\_s：

\_s 函数的全称是 toString,过滤器处理后的结果会当作参数传递给 toString函数，最终 toString函数执行后的结果会保存到Vnode中的text属性中，渲染到视图中

function toString(value){  
 return value == null  
 ? ''  
 : typeof value === 'object'  
 ? JSON.stringify(value,null,2)// JSON.stringify()第三个参数可用来控制字符串里面的间距  
 : String(value)  
}

最后，在分析下parseFilters，在模板编译阶段使用该函数阶段将模板过滤器解析为过滤器函数调用表达式

function parseFilters (filter) {  
 let filters = filter.split('|')  
 let expression = filters.shift().trim() // shift()删除数组第一个元素并将其返回，该方法会更改原数组  
 let i  
 if (filters) {  
 for(i = 0;i < filters.length;i++){  
 experssion = warpFilter(expression,filters[i].trim()) // 这里传进去的expression实际上是管道符号前面的字符串，即过滤器的第一个参数  
 }  
 }  
 return expression  
}  
// warpFilter函数实现  
function warpFilter(exp,filter){  
 // 首先判断过滤器是否有其他参数  
 const i = filter.indexof('(')  
 if(i<0){ // 不含其他参数，直接进行过滤器表达式字符串的拼接  
 return `\_f("${filter}")(${exp})`  
 }else{  
 const name = filter.slice(0,i) // 过滤器名称  
 const args = filter.slice(i+1) // 参数，但还多了 ‘)’  
 return `\_f('${name}')(${exp},${args}` // 注意这一步少给了一个 ')'  
 }  
}

### 小结：

* 在编译阶段通过parseFilters将过滤器编译成函数调用（串联过滤器则是一个嵌套的函数调用，前一个过滤器执行的结果是后一个过滤器函数的参数）
* 编译后通过调用resolveFilter函数找到对应过滤器并返回结果
* 执行结果作为参数传递给toString函数，而toString执行后，其结果会保存在Vnode的text属性中，渲染到视图

## 参考文献

* https://cn.vuejs.org/v2/guide/filters.html#ad
* https://blog.csdn.net/weixin\_42724176/article/details/105546684
* https://vue3js.cn