初步认识vue

vue是一套构建用户界面的渐进式框架，是mvvm框架的一种。vue采用了自底向上增量开发的设计，其核心库只关注视图层，它不仅易于上手，还便于与第三方库或既有项目整合。

> vue有三个部分组成 : 视图-数据-视图模型。视图即HTML部分。 vue的引入：

```
<script src="https://unpkg.com/vue"></script>
//或者直接引入文件
<script src="js/vue.js"></script>
```

声明式渲染

vue的核心是一个允许采用简洁的模板语法来声明式的将数据渲染进 DOM：

```
<div id="app">
    {{message}}
</div>

<script>
    var app = new Vue({
    el: '#app',
    data: {
        message: 'Hello Vue!'
    }
    })
</script>
```

就这样，数据和DOM进行了绑定，视图将数据引入，并进行渲染，显示出来，而且元素是响应式的，打开控制台，修改 app.message 的值，就会发现视图的文本也会相应的更新。

除了文本插值，我们还可以使用指令的方式绑定DOM元素属性。

```
<div id="app-2">
  <span v-bind:title="message">
    鼠标悬停几秒钟查看此处动态绑定的提示信息!
  </span>
</div>
<script>
    var app2 = new Vue({
    el: '#app-2',
    data: {
        message: '页面加载于 ' + new Date().toLocaleString()
    }
```

```
    })
</script>
```

如上，v-bind 属性被称为指令，其前缀 v- 表示其为vue提供的特殊属性。这里该指令的作用是：“将这个元素节点的 title 属性和 Vue 实例的 message 属性保持一致”。

条件与循环

控制一个元素的显示与隐藏

```
<div id="app-3">
    <p v-if="seen">显示</p>
</div>
<script>
    var app3=new Vue({
        el:"#app-3",
        data:{
            seen:true
        }
    })
</script>
```

如果在控制台输入 app3.seen=false ， 你就发现上例的文本隐藏了。

利用 v-for 指令绑定数组的数据来渲染一个项目列表

```
<div id="app-4">
    <ol>
        <li v-for="todo in todos">
            {{todo.text}}
        </li>
    </ol>
</div>
<script>
    var app4=new Vue({
        el:"#app-4",
        data:{
            todos:[
                {text:"HTML"},
                {text:"JAVASCRIPT"},
                {text:"VUE.JS"},
            ]
        }
    })
</script>
```

```
1.HTML
2.JAVASCRIPT
3.VUE.JS
```

在控制台输入 app3.todos.push({text:"CSS"}) ， 列表中会增加一个新项

处理用户输入

利用 v-on 绑定一个事件监听器，使用户和应用之间进行互动

```
<div id="app-5">
    <p>{{message}}</p>
    <input type="button" value="逆转消息" v-on:click="reverse">
</div>
<script>
    var app5=new Vue({
        el:"#app-5",
        data({
            message:"Hello world!"
        }),
        methods:{
            reverse:function(){
                this.message=this.message.split("").reverse().join("")
            }
        }
    })
</script>
```

通过 v-model 指令，可以轻松实现表单输入和应用状态之间的双向绑定

```
<div id="app-6">
    <p>{{message}}</p>
    <input type="text" v-model="message">
</div>
<script>
    var app6=new Vue({
        el:"app-6",
        data:{
            message:"hello"
        }
    })
</script>
```

组件化应用构建

使用 v-bind 指令将todo传到每一个重复的组件中

```
<div id="app-7">
    <ol>
        <todo-item v-for="item in groceryLIst" v-bind:todo="item" v-
bind:key="item.id"></todo-item>
    </ol>
</div>
<script>
    Vue.component("todo-item",{
        props:["todo"],
        template:"<li>{{todo.text}}</li>"
    })
    var app7=new Vue({
        el:"#app-7",
        data:{
            groceryLIst:[
                {id:0,text:"JAVASCRIPT"},
                {id:1,text:"HTML"},
                {id:2,text:"VUE"}
            ]
        }
    })
</script>
```

```
1.JAVASCRIPT
2.HTML
3.VUE
```

在上面的例子中，我们已经设法将应用分割成了两个更小的单元，子单元通过 props 接口实现了与父单元很好的解耦。

To be continued.....

模板语法

> Vue 入门，Vue属性和指令 : https://segmentfault.com/a/1190000010917625

插值

文本

最常用的文本插值方法是使用"Mustache"语法

```
<p>Message:{{msg}}</p>
```

Mustache 标签将会被替代为对应数据对象上 msg 属性的值。无论何时，绑定的数据对象上 msg 属性发生了改变，插值处的内容都会更新。

通过使用 v-once 指令，可以执行一次性的插值，当数据改变时，插值处的内容便不会再更新

```
<p v-once>这个数据将不会改变{{msg}}</p>
```

纯HTMl

{{}}会将数据解释为纯文本，为了能够输出HTML，可以使用 v-html 指令

```
<div class="exp" v-html="rewHtml">{{msg}}</div>
<script>
    var exp=new Vue({
        el:".exp",
        data:{
            msg:"big world",
            rewHtml:"<p>hello world</p>"
        }
    })
</script>
```

属性 rewHtml 会将这个 div 内的内容替换，也就是说原来绑定的 msg 属性被忽略，rewHtml 将作为 HTML 被直接添加

```
hello world
```

特性

Mustache 语法不能用在设置 HTML 特性上，这时候就可以使用 v-bind 指令：

```
<div class="exp" v-bind:title="tit">哈哈哈哈</div>
<script>
    var exp=new Vue({
        el:".exp",
        data:{
            tit:"2333"
        }
    })
</script>
```

布尔类特性可以这样设置，值为 false ，则该特性就会被删除

```
<button class="exp" v-bind:disabled="tit">按钮</button>
<script>
    var exp=new Vue({
        el:".exp",
        data:{
            tit:true
        }
    })
</script>
```

如上，当我们把 tit的值改为 false 时，则 button 中的 disabled 会被删除，按钮也就可以点击了

javascript 表达式

Vue 也提供了对js表达式的支持

```
<div class="exp">{{ message.split('').reverse().join('')}}</div>
<script>
    var exp=new Vue({
        el:".exp",
        data:{
            message:"abcdefg"
        }
    })
</script>
```

```
gfedcba
```

这些表达式会在所属 Vue 实例的数据作用域下作为 JavaScript 被解析。有个限制就是，每个绑定都只能包含单个表达式，所以下面的例子都不会生效。

```
<!-- 这是语句，不是表达式 -->
{{ var a = 1 }}
<!-- 流控制也不会生效，请使用三元表达式 -->
{{ if (ok) { return message } }}
```

指令

指令（Directives）是带有 v- 前缀的特殊属性。指令属性的值预期是单个 JavaScript 表达式（v-for 是例外情况，稍后我们再讨论）。指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于

DOM。我上一篇文章中也提到过了

```
<div id="app-3">
    <p v-if="seen">显示</p>
</div>
```

这里， v-if 指令将根据表达式 seen 的值的真假来插入/移除

元素。

参数

一些指令能够接收一个“参数”，在指令名称之后以冒号表示。例如，v-bind 指令可以用于响应式地更新 HTML 属性：

```
<a v-bind:href="url"></a>
```

在这里 href 是参数，告知 v-bind 指令将该元素的 href 属性与表达式 url 的值绑定。

另一个例子是 v-on 指令，它用于监听 DOM 事件，也在我的上一篇文章中也提到了

```
<input type="button" value="逆转消息" v-on:click="reverse">
```

修饰符

修饰符（Modifiers）是以半角句号 . 指明的特殊后缀，用于指出一个指令应该以特殊方式绑定。例如，.prevent 修饰符告诉 v-on 指令对于触发的事件调用 event.preventDefault()：

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

过滤器

过滤器可以被用作一些常见的文本格式化，规定好过滤器的方法，便可以在模板里调用了。过滤器可以用在两个地方：mustache 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”符指示：

```
<!-- in mustaches -->
{{ message | capitalize }}
<!-- in v-bind -->
<div v-bind:id="rawId | formatId"></div>
```

```
<!-- 依旧是反转这个字符串 -->
<div class="exp">{{ message|reverse}}</div>
<script>
    var exp=new Vue({
        el:".exp",
        data:{
            message:"abcdefg"
        },
        filters:{
            reverse:function(value){
                return value.split("").reverse().join("")
            }
        }
    })
</script>
```

过滤器也可以串联：

```
<!-- 反转这个字符串，然后再把翻转后的字符串转换成数组 -->
<div class="exp">{{message|reverse|split}}</div>
<script>
    var exp=new Vue({
        el:".exp",
        data:{
            message:"abcdefg"
        },
        filters:{
            reverse:function(value){
                return value.split("").reverse().join("")
            },
            split:function(value1){
                return value1.split("")
            }
        }
    })
</script>
```

```
[ "g", "f", "e", "d", "c", "b", "a" ]
```

在这个例子中，reverse 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 split，将 reverse 的结果传递到 split 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 取值后的值作为第三个参数。

缩写

Vue.js 为 v-bind 和 v-on 这两个最常用的指令，提供了特定简写：

v-bind 缩写

```
<!-- 完整语法 -->
<a v-bind:href="url"></a>
<!-- 缩写 -->
<a :href="url"></a>
```

v-on 缩写

```
<!-- 完整语法 -->
<a v-on:click="doSomething"></a>
<!-- 缩写 -->
<a @click="doSomething"></a>
```

计算属性

gatter函数

```
<!-- 使用了N次的字符串反转 -->
<div class="exp">
    <p>正向{{message}}</p>
    <p>反向{{reMessage}}</p>
</div>
<script>
    var exp = new Vue({
        el:".exp",
        data:{
            message:"abcdefg"
        },
        computed:{
            reMessage:function(){
                return this.message.split("").reverse().join("")
            }
        }
    })
</script>
```

结果如下

```
正向abcdefg

反向gfedcba
```

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

```
console.log(exp.reMessage) // --> "gfedcba"
exp.message="Hello"
console.log(exp.reMessage) // --> eybdooG
```

当我们在控制台修改exp.message的值时，exp.reMessage 的值也会更新

计算属性 vs method 方法

其实针对上面的例子，使用method方法也可以达到

```
<p>反向{{reMessage}}</p>
<script>
    methods: {
        reMessage: function () {
            return this.message.split('').reverse().join('')
        }
    }
</script>
```

事实上，在计算结果上，method方法和compute方法是一致的，然而，不同的是计算属性是基于它们的依赖进行缓存的。计算属性只有在它的相关依赖发生改变时才会重新求值。这就意味着只要 message 还没有发生改变，多次访问 reMessage 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。

而使用method方法时，每次重新渲染时，都要再次执行函数

计算属性 vs Watched 属性

```
<div class="exp">
    {{fullName}}
    <input value="text" v-model="firstName">
    <input value="text" v-model="lastName">
</div>
<script>
var exp=new Vue({
```

```
    el:".exp",
    data:{
        firstName:"Pure",
        lastName:"View",
        fullName:"PureView"
    },
    watch:{
        firstName:function(val){
            this.fullName=val+this.lastName
        },
        lastName:function(val){
            this.fullName=this.firstName+val1
        }
    }
})
```

如上，我们要改变fullName的值，使用watch方法会有比较多的重复代码，如果要改变的数据很多，则需要写很多的watch方法。其实可以使用计算属性：

```
var exp=new Vue({
    el:".exp",
    data:{
        firstName:"Pure",
        lastName:"View"
    },
    computed:{
        fullName:function(){
            return this.firstName+this.lastName
        }
    }
})
```

这样就舒服多了。

计算 setter

```
<div class="exp">
    {{fullName}}
    <input value="text" v-model="firstName">
    <input value="text" v-model="lastName">
</div>
<script>
    var exp=new Vue({
        el:".exp",
        data:{
            firstName:"Pure",
            lastName:"View"
```

```
        },
        computed:{
            fullName:{
                get:function(){
                    return this.firstName+this.lastName
                },
                set:function(newValue){
                    var name=newValue.split("")
                    this.firstName=name[0]
                    this.lastName = name[name.length - 1]
                }
            }
        }
    })
</script>
```

在控制台修改 exp.fullName的值，那么 firstName 和 lastName 的值也会相应的更新

Vue 方法

> 事件

```
methods:{

}
```

> 过滤器

```
filters:{

}
```

> 计算

```
conputed:{

}
```

> 观察

```
watch:{

}
```

> 钩子函数

```
created function(){

}
mountde function(){

}
updated function(){

}
destoryed function(){

}
```

Class 与 Style 绑定

绑定 HTML Class

对象语法

我们可以传给 v-bind:class 一个对象，以动态地切换 class：

```
<style>
    .exp{
        border: 1px solid #ccc;
    }
    .forExp{
        background: blue;
    }
</style>
<div class="exp" v-bind:class="{newExp:isNewExp}"></div>
<script>
    var exp=new Vue({
        el:".exp".
        data:{
            isForExp:false
        }
    })
</script>
```

如上，我们先给 .exp 一个边框，我们利用 v-bind 方法传入一个新的 class 属性 .newExp，设置一个蓝色的背景颜色。当我们在控制台修改 .newExp 的属性为 true 时，会给 div 添加一个蓝色的背景颜色。

我们也可以传入更多的属性来切换多个 class 。

```
<div class="exp" v-bind:class="{newExp:isExp,npc:isNpc}"></div>
<script>
    data:{
        isExp:false,
        isNpc:true
    }
</script>
```

在模板里的渲染结果为：

```
<div class="exp isNpc"></div>
```

我们也可以使用对象的方法来切换属性

```
<div class="exp" v-bind:class="obj"></div>
<script>
    data:{
        obj:{
            newExp:false,
            npc:true
        }
    }
</script>
```

渲染结果和上面的一样

数组语法

我们可以把一个数组传给 v-bind:class，以应用一个 class 列表：

```
<div class="exp" v-bind:class="[newExp,oldExp]"></div>
<script>
    data:{
        newExp:"new",
        oldExp:"old"
    }
</script>
```

渲染为：

```
<div class="exp new old"></div>
```

要切换class，使用三元运算符：

```
<div class="exp" v-bind:class="[act? newExp:oldExp]"></div>
<script>
    data{
        newExp:"new",
        oldExp:"old",
        act:true
    }
<script>
```

act 为 true 时，添加 new ，为 false 时添加 old。

用在组件上

声明一个组件：

```
Vue.component("my", {
    template: '<p class="foo bar">Hi</p>'
})
```

然后在使用它的时候添加一些 class：

```
<my class="tip"></my>
```

最终渲染为

```
<p class="foo bar tip">Hi</p>
```

同样的适用于绑定 HTML class:

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

当active为true时，HTML 将被渲染成为：

```
<p class="foo bar active">Hi</p>
```

绑定内联样式

对象语法

v-bind:style 的对象语法十分直观——看着非常像 CSS，其实它是一个 JavaScript 对象。 CSS 属性名可以用驼峰式 (camelCase) 或 (配合引号的) 短横分隔命名 (kebab-case)：

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
<script>
    data:{
        activeColor:"blue",
        fontSize:20
    }
</script>
```

使用对象语法的话，会看起来更加清晰

```
<div v-bind:style="obj"></div>
<script>
    data:{
        obj:{
            color:"#FFF",
            fontSize:"20px"
        }
    }
</script>
```

数组语法

v-bind:style 的数组语法可以将多个样式对象应用到一个元素上：

```
<div v-bind:style="[style1,style2]"></div>
<script>
    data:{
        style1:{
            color:"#666"
        },
        style2:{
            background:"#b1b1b1"
        }
    }
</script>
```

自动添加前缀

当 v-bind:style 使用需要特定前缀的 CSS 属性时，如 transform，Vue.js 会自动侦测并添加相应的前缀。

```
Chrome 和 Safari : -webkit-
IE :               -ms-
Firfox :           -moz-
Opera :            -o-
```

条件渲染

v-if

在 < template > 中配合 v-if 渲染一整组

在使用 v-if 控制元素的时候，我们需要将它添加到这个元素上去。然而如果要切换很多元素的时候，一个个的添加就太麻烦了。这时候就可以使用 < template > 将一组元素进行包裹，并在上面使用 v-if。最终的渲染结果不会包含 < template > 元素。

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>

<script>
    data:{
        ok:true
    }
</script>
```

我们更改 ok 的值，就可以控制整组的元素了

v-else

你可以使用 v-else 指令来表示 v-if 的"else 块"：

```
<div v-if="ok">
    Now you see me
</div>
<div v-else>
    Now you don't
</div>
```

v-else 元素必须紧跟在 v-if 或者 v-else-if 元素的后面——否则它将不会被识别。

v-else-if

v-else-if，顾名思义，充当 v-if 的“else-if 块”。可以链式地使用多次：

```
<div>
    <p v-if="sc>=90">优秀</p>
    <p v-else-if="sc>=60">及格</p>
    <p v-else="sc<60">不及格</p>
</div>
```

类似于 v-else，v-else-if 必须紧跟在 v-if 或者 v-else-if 元素之后。

可复用元素

Vue 会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染。这么做，除了使 Vue 变得非常快之外，还有一些有用的好处。例如，如果你允许用户在不同的登录方式之间切换:

```
<div class="exp">
    <template v-if="loginType === 'username'">
        <label>Username</label>
        <input placeholder="Enter your username">
    </template>

    <template v-else>
        <label>Email</label>
        <input placeholder="Enter your email address">
    </template>
    <input type="button" @click="btn" value="切换"/>
</div>

<script>
    var exp=new Vue({
        el:".exp",
        data:{
            loginType:"username"
        },
        methods:{
            btn:function(){
                if(this.loginType==="username"){
                    this.loginType="email"
                }else{
                    this.loginType="username"
                }
            }
        }
    })
</script>
```

那么在上面的代码中切换 loginType 将不会清除用户已经输入的内容。因为两个模板使用了相同的元素，<

input > 不会被替换掉——仅仅是替换了它的 placeholder。

复制上面的代码，在自己的浏览器中试一试。

有时候我们不希望浏览器保留我们输入的内容，所以 Vue 为你提供了一种方式来声明“这两个元素是完全独立的——不要复用它们”。只需添加一个具有唯一值的 key 属性即可：

```
<template v-if="loginType === 'username'">
    <label>Username</label>
    <input placeholder="Enter your username" key="username">
</template>

<template v-else>
    <label>Email</label>
    <input placeholder="Enter your email address" key="email">
</template>
```

v-show

另一个用于根据条件展示元素的选项是 v-show 指令。用法大致一样：

```
<h1 v-show="ok">Hello!</h1>
<script>
    data:{
        ok:false
    }
</script>
```

不同的是带有 v-show 的元素始终会被渲染并保留在 DOM 中。v-show 是简单地切换元素的 CSS 属性 display 。

渲染如下

```
<div style="display:none;"></div>
```

列表渲染

使用 v-for 把一个数组对应为一组元素

我们用 v-for 指令根据一组数组的选项列表进行渲染。 v-for 指令需要以 item in items 形式的特殊语法， items 是源数据数组并且 item 是数组元素迭代的别名。

```
<div class="exp">
    <ul>
        <li v-for="item in items">{{item.text}}</li>
    </ul>
</div>

<script>
    data:{
        items:[
            {text:"eat"},
            {text:"play"},
            {text:"game"}
        ]
    }
</script>
```

渲染结果

```
<div class="exp">
    <ul>
        <li>eat</li>
        <li>play</li>
        <li>game</li>
    </ul>
</div>
```

v-for 还支持一个可选的第二个参数为当前项的索引。

```
<div class="exp">
    <ul>
        <li v-for="item,index in items">{{index}}-{{item.text}}</li>
    </ul>
</div>
<script>
    var exp=new Vue({
        el:".exp",
        data:{
            items:[
                {text:'eat'},
                {text:'paly'},
                {text:'game'}
            ]
        }
    })
</script>
```

结果

```
0-eat
1-paly
2-game
```

一个对象的 v-for

你也可以用 v-for 通过一个对象的属性来迭代。

```
<div class="exp">
    <ul>
        <li v-for="value in obj">{{value}}</li>
    </ul>
</div>
<script>
    var exp=new Vue({
        el:".exp",
        data:{
            obj:{
                firstname:"PureView",
                lastname:"一个安静的美男子",
                age:18
            }
        }
    })
</script>
```

结果

```
PureView
一个安静的美男子
18
```

你一共可以提供三个参数，第二个参数为键名，第三个为索引：

```
<li v-for="value,key,index in obj">{{index+1}}. {{key}}: {{value}}</li>
```

结果

```
1. firstname: PureView
2. lastname: 一个安静的美男子
```

```
3. age: 18
```

数组更新检测

变异方法

Vue 包含一组观察数组的变异方法，所以它们也将会触发视图更新。这些方法如下：

- push()
- pop()
- shift()
- unshift()
- splice()
- sort()
- reverse()

例如

```
<div class="exp">
    <ul>
        <li v-for="item in items">{{item.text}}</li>
    </ul>
</div>

<script>
    var exp=new Vue({
        el:".exp",
        data:{
            items:[
                {text:"eat"},
                {text:"play"},
                {text:"game"}
            ]
    }
    })
    exp.items.push({text:'watch TV'})
</script>
```

重塑数组

变异方法(mutation method)，顾名思义，会改变被这些方法调用的原始数组。相比之下，也有非变异(non-mutating method)方法，例如： filter(), concat() 和 slice() 。这些不会改变原始数组，但总是返回一个新数组。当使用非变异方法时，可以用新数组替换旧数组：

```
data:{
    items:[
```

```
            {text:"eat"},
            {text:"play"},
            {text:"game"},
            {text:"gaming"},
            {text:"wot"},
            {text:"wows"},
            {text:"wt"}
        ]
    }
}
exp.items.slice(0,5)
```

利用上一节的例子,返回的值不会改变原数据，在控制台打印我们就能看到了。

注意事项

由于 JavaScript 的限制， Vue 不能检测以下变动的数组：

- 当你利用索引直接设置一个项时，例如： vm.items[indexOfItem] = newValue
- 当你修改数组的长度时，例如： vm.items.length = newLength

为了解决第一类问题，以下两种方式都可以实现和 vm.items[indexOfItem] = newValue 相同的效果， 同时也将触发状态更新：

```
// Vue.set
Vue.set(exp.items, indexOfItem, newValue)
```

```
// Array.prototype.splice
exp.items.splice(indexOfItem, 1, newValue)
```

为了解决第二类问题，你可以使用 splice：

```
exp.items.splice(newLength)
```

对象更新检测

由于现代JavaScript的限制，Vue无法检测属性添加或删除。 例如：

```
var exp=new Vue({
    data:{
        a:1
    }
})
```

```
vm.b=2 //模板内无响应
```

Vue是不允许动态地向已创建的实例添加新的根级属性的。这时候 Vue 提供了一个方法用来对对象添加属性：

```
Vue.set(object,key,value)
```

举个例子

```
var exp=new Vue({
    el:".exp",
    data:{
        obj:{
            me:"pureview",
            pet1:"dog",
            pet2:"cat",
            hobby:"games"
        }
    }
})
```

我们在控制台输入下面的代码，就可以看到模板内的数据进行了更新

```
Vue.set(exp.obj,"todo","eating")
```

除了添加属性，我们也可以进行删除操作

```
Vue.delete(exp.obj,"pet2")
```

显示过滤/排序结果

有时，我们想要显示一个数组的过滤或排序副本，而不实际改变或重置原始数据。在这种情况下，可以创建返回过滤或排序数组的计算属性。

比如我们在一个数组中取其偶数

```
<div class="exp">
    <ul>
        <li v-for="n in numbers">{{n}}</li>
    </ul>
</div>
```

```
<script>
    var exp=new Vue({
        el:".exp",
        data:{
            num:[1,2,3,4,5,6,7,8,9,10]
        },
        computed:{
            numbers:function(){
                return this.num.filter(function(num){
                    return num%2===0
                })
            }
        }
    })
</script>
```

模板显示结果：

```
2
4
6
8
10
```

在计算属性不适用的情况下 (例如，在嵌套 v-for 循环中) 你可以使用一个 method 方法：

```
<div class="exp">
    <ul>
        <li v-for="n in even(num)">{{n}}</li>
    </ul>
</div>

<script>
    var exp=new Vue({
        el:".exp",
        data:{
            num:[1,2,3,4,5,6,7,8,9,10]
        },
        methods:{
            even:function(num){
                return num.filter(function(num){
                    return num%2===0
                })
            }
        }
    })
</script>
```

结果是一样的

一段取值范围的 v-for

v-for 也可以取整数。在这种情况下，它将重复多次模板。

```
<div>
  <span v-for="n in 10">{{ n }} </span>
</div>
```

结果

```
1 2 3 4 5 6 7 8 9 10
```

v-for 在 < template > 上

与模板v-if类似，您还可以使用带有 v-for 的< template >标签来呈现多个元素的块。

```
<ul>
  <template v-for="item in items">
    <li>{{ item.msg }}</li>
    <li class="divider"></li>
  </template>
</ul>
```

v-for 和 v-if

当 v-for 与 v-if 一起使用时，v-for 具有比 v-if 更高的优先级，这意味着 v-if 将分别重复运行于每个 v-for 循环中。当我们仅为某些项目渲染节点时，这可能很有用：

```
<li v-for="todo in todos" v-if="!todo.isComplete">
  {{ todo }}
</li>
```

而如果我们的目的是有条件地跳过循环的执行，那么可以将 v-if 置于外层元素 (或 < template >)上。如:

```
<ul v-if="todos.length">
  <li v-for="todo in todos">
    {{ todo }}
  </li>
</ul>
```

```
<p v-else>No todos left!</p>
```

组件的 v-for

在 Vue 的 2.2.0 以上的版本中，我们要在组件中使用 v-for 时，必须使用 key

```
<my-component v-for="(item,index) in itmes" v-bind:key="index"></my-component>
```

虽然在自定义组件里可以使用 v-for ，但是，他不能自动传递数据到组件里，因为组件有自己独立的作用域。为了传递迭代数据到组件里，我们要用 props ：

```
<my-component v-for="(item,index) in items" v-bind:key="index"
:lie="item.text"l></my-component>

<script>
    Vue.component('mycom', {
        template: "<p>{{this.lie}}</p>",
        props:["lie"]
    })
    var exp=new Vue({
        el:".exp",
        data:{
            items:[
                {text:'从前有座山'},
                {text:'山上有座庙'},
                {text:'庙里有个老和尚'},
                {text:'正在玩王者荣耀'}
            ]
        }
    })
</script>
```

结果

```
从前有座山

山上有座庙

庙里有个老和尚

正在玩王者荣耀
```