Vue.js 从零开始

基础 & 进阶 & 实战

http://www.iwanli.me

目　录

入门

> By 晚黎

博客： http://www.iwanli.me

本教程从零开始一步步学习VueJs2，`Vue` 现在已经非常火了，但是网上的教程都是一些很零散的，没有一个系统完整的教程。大部分都是以文档为主，本教程是以视频的方式分享给大家。希望对大家有所帮助。

> 本人所学的知识和大家一样都是看网上的教程，互相借鉴。不喜勿喷~

教程代码地址

osChina：https://git.oschina.net/iwl/VueJs2

视频地址

哔哩哔哩：http://space.bilibili.com/11490447/#!/channel/detail?cid=8529

土豆：http://www.tudou.com/home/_428769813/playlist

优酷：http://i.youku.com/i/UMTcxNTA3OTI1Mg==/playlists?spm=a2hzp.8244740.0.0

Hello Vue

本教程直接独立版本安装，即直接下载并用

> Vue CDN : https://unpkg.com/vue@2.1.10/dist/vue.js

```
<!DOCTYPE html>
<html>
<head>
    <title>安装</title>
</head>
<body>
    <div id="app">
        <h1>{{title}}</h1>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                title: 'hello Vue'
            }
        });
    </script>
</body>
</html>
```

- `el`（Vue管理的作用域）
- `data`（属性，相当于后端语言 `MVC` 模式中的 `M`,即数据模型，提供数据）

> 作用域官方不建议挂载到 `HTML` 的根上

第一个Vue实例

- 创建一个Vue对象
- 模型属性显示
- 动态改变模型属性的值

创建一个Vue对象

创建一个Vue，只需要用关键词 `new Vue()` 即可，将会创建一个 `Vue` 对象

```
<script type="text/javascript" src="../js/vue.js"></script>
<script type="text/javascript">
    var app = new Vue({...});
</script>
```

模型属性显示

创建一个Vue对象后，所有的属性、方法等全部放在 `new Vue({...})` 对象中，页面显示属性用 `{{ ... }}` 显示

```
<!DOCTYPE html>
<html>
<head>
    <title>第一个Vue实例</title>
</head>
<body>
    <div id="app">
        <h1>{{title}}</h1>
        <input type="text" v-on:input="changeTitle">
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                title: 'hello Vue'
            },
            methods: {
                changeTitle: function (event) {
                    this.title = event.target.value;
                }
            }
        });
    </script>
</body>
</html>
```

> 属性的显示必须在Vue的作用域中，如果不在作用域中，属性值将不会显示。属性在的显示方法是由 `{{ ... }}` 的形式

动态改变模型属性的值

上面代码中 `<input type="text" v-on:input="changeTitle">` ，在input上加上一个 `changeTitle` 的事件。 `Vue` 实例中的 `methods` 相当于后端语言MVC模式中的 C （ controller ）。

模板

Vue.js 使用了基于 `HTML` 的模版语法，允许开发者声明式地将 `DOM` 绑定至底层 `Vue` 实例的数据。所有 Vue.js 的模板都是合法的 `HTML` ，所以能被遵循规范的浏览器和 `HTML` 解析器解析。
在底层的实现上， `Vue` 将模板编译成虚拟 `DOM` 渲染函数。结合响应系统，在应用状态改变时， `Vue` 能够智能地计算出重新渲染组件的最小代价并应用到 `DOM` 操作上。

文本显示

Vue.js 使用了基于 HTML 的模版语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。所有 Vue.js 的模板都是合法的 HTML ，所以能被遵循规范的浏览器和 HTML 解析器解析。

数据绑定最常见的形式就是使用 “Mustache” 语法（双大括号）的文本插值。

```
<div id="app">
    <h1>{{ title }}</h1>
</div>
```

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

> 属性显示直接写属性名称，像 `{{ this.title }}` 或 `{{ data.title }}` 都是错误的写法，只要属性在 `app` 作用域内，VueJs会自动解析属性

methods返回字符串

Vue模板除了能显示属性数据之外，还可以显示方法返回的字符串。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue模板</title>
</head>
<body>
    <div id="app">
        <h1>{{ title }}</h1>
        <h1>{{ sayHello() }}</h1>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                title: 'hello Vue'
            },
            methods: {
                sayHello(){
                    return 'Hello';
                }
            }
        });
    </script>
</body>
</html>
```

methods返回属性对象

在 `methods` 中除了返回字符串外，还可以返回当前作用域中的属性。

```
<h1>{{ say() }}</h1>
...
var app = new Vue({
    el: '#app',
    data: {
        title: 'hello Vue'
    },
    methods: {
       ...
        say(){
            return this.title;
        }
    }
});
```

属性绑定v-bind

Vue 在HTML 属性中使用时，不能使用 `Mustache` 语法，需要使用 `v-bind` 命令。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue模板-属性绑定</title>
</head>
<body>
    <div id="app">
        <a v-bind:href="link" target="_blank">百度</a>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                link: 'http://baidu.com'
            }
        });
    </script>
</body>
</html>
```

> 上面HTML属性 `v-bind:href` 可以简写为 `:href`

一次性属性v-once

通过使用 `v-once` 指令，你也能执行一次性地显示属性，当数据改变时，属性的内容不会更新。

> 注意这会影响到该节点上所有的数据绑定。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue模板v-once</title>
</head>
<body>
    <div id="app">
        <h1 v-once>{{ title }}</h1>
        <p>{{ sayHello() }} - <a :href="link" target="_blank">百度</a></p>

    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                title: 'Hello Vue',
                link: 'http://baidu.com'
            },
            methods: {
                sayHello(){
                    this.title = 'Hello !';
                    return this.title;
                }
            }
        });
    </script>
</body>
</html>
```

v-html

双大括号会将数据解释为纯文本，而非 HTML 。为了输出真正的 HTML ，你需要使用 `v-html` 命令。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue模板v-html</title>
</head>
<body>
    <div id="app">
        <p>{{ link }}</p>
        <p v-html="link"></p>

    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                link: 'hello - <a href="http://www.baidu.com" target="_blank">百度</a>'
            }
        });
    </script>
</body>
</html>
```

不能使用 v-html 来复合局部模板，因为 Vue 不是基于字符串的模板引擎。组件更适合担任 UI 重用与复合的基本单元。

> 你的站点上动态渲染的任意 HTML 可能会非常危险，因为它很容易导致 XSS 攻击。请只对可信内容使用 HTML 插值，绝不要对用户提供的内容插值。

事件处理器

本章节讲解在 `Vue` 中为 `DOM` 元素绑定事件的具体方法，通过v-on指令或事件语法糖 `@` 为 `DOM` 元素绑定事件。
`Vue` 解析组件模板后，在绑定更新 `v-on` 指令时会为 `DOM` 元素绑定事件(当然如果元素为 `iframe` ，会等到 `iframe` 加载完成后再为其绑定事件)。

监听事件

在 `Vue` 中可以用 `v-on` 指令监听 `DOM` 事件来触发一些 `JavaScript` 代码。一些常用的 `Javascript` 事件基本上都支持。

属性	当以下情况发生时，出现此事件
onabort	图像加载被中断
onblur	元素失去焦点
onchange	用户改变域的内容
onclick	鼠标点击某个对象
ondblclick	鼠标双击某个对象
onerror	当加载文档或图像时发生某个错误
onfocus	元素获得焦点
onkeydown	某个键盘的键被按下
onkeypress	某个键盘的键被按下或按住
onkeyup	某个键盘的键被松开
onload	某个页面或图像被完成加载
onmousedown	某个鼠标按键被按下
onmousemove	鼠标被移动
onmouseout	鼠标从某元素移开
onmouseover	鼠标被移到某元素之上
onmouseup	某个鼠标按键被松开
onreset	重置按钮被点击
onresize	窗口或框架被调整尺寸
onselect	文本被选定
onsubmit	提交按钮被点击
onunload	用户退出页面

监听属性

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue监听事件</title>
</head>
```

```
<body>
    <div id="app">
        <button v-on:click="counter ++"> 增加 1</button>
        <p>点击了 {{ counter }} 次</p>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                counter: 0
            }
        });
    </script>
</body>
</html>
```

> 监听属性一般用于最简单的情况。

方法事件处理器

许多事件处理的逻辑都很复杂，所以直接把 `JavaScript` 代码写在 `v-on` 指令中是不可行的。因此 `v-on` 可以接收一个定义的方法来调用。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue监听事件</title>
</head>
<body>
    <div id="app">
        <button v-on:click="counter ++"> 增加 1</button>
        <!-- `reduction` 是在下面定义的方法名 -->
        <button v-on:click="reduction">减少 1</button>
        <p>点击了 {{ counter }} 次</p>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                counter: 0
            },
            // 在 `methods` 对象中定义方法
            methods: {
                reduction(){
                    // `this` 在方法里指当前 Vue 实例
                    this.counter --;
                }
            }
        });
    </script>
</body>
</html>
```

内联处理方法

除了直接绑定到一个方法，也可以用内联 JavaScript 语句，传递参数。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue监听事件</title>
</head>
<body>
    <div id="app">
        <button v-on:click="add(2)"> 增加 2</button>
        <!-- `reduction` 是在下面定义的方法名 -->
        <button v-on:click="reduction">减少 1</button>
        <p>点击了 {{ counter }} 次</p>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                counter: 0
            },
            // 在 `methods` 对象中定义方法
            methods: {
                reduction(){
                    // `this` 在方法里指当前 Vue 实例
                    this.counter --;
                },
                add(step){
                    this.counter += step;
                }
            }
        });
    </script>
</body>
</html>
```

事件修饰符

在事件处理程序中调用 `event.preventDefault()` 或 `event.stopPropagation()` 是非常常见的需求。尽管我们可以在 `methods` 中轻松实现这点，但更好的方式是： `methods` 只有纯粹的数据逻辑，而不是去处理 `DOM` 事件细节。
为了解决这个问题， Vue.js 为 `v-on` 提供了 事件修饰符。通过由点(.)表示的指令后缀来调用修饰符。

- `.stop`
- `.prevent`
- `.capture`
- `.self`
- `.once`

> preventDefault() : 该方法将通知 Web 浏览器不要执行与事件关联的默认动作（如果存在这样的动作）。例如，如果 type 属性是 "submit"，在事件传播的任意阶段可以调用任意的事件句柄，通过调用该方法，可以阻止提交表单。注意，如果 Event 对象的 cancelable 属性是 fasle，那么就没有默认动作，或者不能阻止默认动作。无论哪种情况，调用该方法都没有作用。
>
> stopPropagation() : 该方法将停止事件的传播，阻止它被分派到其他 Document 节点。在事件传播的任何阶段都可以调用它。注意，虽然该方法不能阻止同一个 Document 节点上的其他事件句柄被调用，但是它可以阻止把事件分派到其他节点。
>
> .once Vue2.1.4 版本新增

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue事件修饰</title>
</head>
<body>
    <div id="app">
        <p v-on:mousemove="updateMove">
            坐标：{{ x }} / {{ y }}
            --
            <!--<span v-on:mousemove="stopMove">停止</span>-->
            <span v-on:mousemove.stop="stopMove">停止</span>
        </p>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                x: 0,
                y: 0,
            },
            // 在 `methods` 对象中定义方法
            methods: {
                updateMove(event){
```

```
                    this.x = event.clientX;
                    this.y = event.clientY;
                },
                stopMove(event){
                    //操作原生 DOM 事件
                    event.stopPropagation();
                }
            }
        });
    </script>
</body>
</html>
```

按键修饰符

当监听键盘事件时，我们常常需要判断常用的 `key code`。Vue.js 提供了一个特殊的只能用在 `v-on` 指令的过滤器：`key`。它接收一个表示 `key code` 的参数并完成判断。
记住所有的 keyCode 比较困难，所以 Vue 为最常用的按键提供了别名：

- `.enter`
- `.tab`
- `.delete` (捕获 “删除” 和 “退格” 键)
- `.esc`
- `.space`
- `.up`
- `.down`
- `.left`
- `.right`

> Vue2.10 新增按键修饰符

- `.ctrl`
- `.alt`
- `.shift`
- `.meta`

可以通过全局 `config.keyCodes` 对象自定义按键修饰符别名：

```
// 可以使用 v-on:keyup.f1
Vue.config.keyCodes.f1 = 112
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue键盘修饰</title>
</head>
<body>
    <div id="app">
        <!--<input type="text" v-on:keyup.enter="alert($event)">-->
        <input type="text" v-on:keyup.alt.67="alert($event)">
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            // 在 `methods` 对象中定义方法
            methods: {
                alert(event){
```

```
                    alert(event.target.value);
                }
            }
        });
    </script>
</body>
</html>
```

为什么要在 HTML 中写监听器？

你可能注意到这种事件监听的方式违背了关注点分离（separation of concern）传统理念。不必担心，因为所有的 Vue.js 事件处理方法和表达式都严格绑定在当前视图的 `ViewModel` 上，它不会导致任何维护上的困难。实际上，使用 `v-on` 有几个好处：

- 扫一眼 HTML 模板便能轻松定位在 JavaScript 代码里对应的方法。
- 因为你无须在 JavaScript 里手动绑定事件，你的 ViewModel 代码可以是非常纯粹的逻辑，和 DOM 完全解耦，更易于测试。
- 当一个 ViewModel 被销毁时，所有的事件处理器都会自动被删除。你无须担心如何自己清理它们。

计算属性

Vue.js 的内联表达式非常方便，但它最合适的使用场景是简单的布尔操作或字符串拼接。如果涉及更复杂的逻辑，你应该使用计算属性。

基础栗子

为什么需要计算属性

在模板中绑定表达式是非常便利的，但是它们实际上只用于简单的操作。在模板中放入太多的逻辑会让模板过重且难以维护。在这种情况下，模板不再简单和清晰。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue计算属性</title>
</head>
<body>
    <div id="app">
        <button @click="counter++">增加 1</button>
        <button @click="counter--">减少 1</button>
        <button @click="counter++">计算属性增加 1</button>
        <p>Counter: {{ counter }}</p>
        <p>Result: {{ result() }}</p>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
              counter: 0
            },
            computed: {
                output(){
                    return this.counter > 5 ? 'Counter大于5' : 'Counter小于5';
                }
            },
             // 在 `methods` 对象中定义方法
            methods: {
              result(){
                 return this.counter > 5 ? 'Counter大于5' : 'Counter小于5';
              }
            }
        });
    </script>
</body>
</html>
```

计算属性和methods区别

你可能已经注意到我们可以通过调用表达式中的method来达到同样的效果。不经过计算属性，我们可以在 `methods` 中定义一个相同的函数来替代它。对于最终的结果，两种方式确实是相同的。然而，**不同的是计算属性是基于它的依赖缓存**。计算属性只有在它的相关依赖发生改变时才会重新取值。这就意味着只要 `counter` 没有发生改变，多次访问 `output` 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。所以在巨大的数组遍历和做大量的计算的情况下，使用计算属性是最佳的选择。

watch属性

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 `watcher` 。这是为什么 `Vue` 提供一个更通用的方法通过 `watch` 选项，来响应数据的变化。当你想要在**数据变化响应时，执行异步操作或开销较大的操作**，这是很有用的。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue计算属性之watch</title>
</head>
<body>
    <div id="app">
        <button @click="counter++">增加 1</button>
        <button @click="counter--">减少 1</button>
        <button @click="counter++">计算属性增加 1</button>
        <p>Counter: {{ counter }}</p>
        <p>Result: {{ result() }}</p>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                counter: 0
            },
            computed: {
                output() {
                    return this.counter > 5 ? 'Counter大于5' : 'Counter小于5';
                }
            },
            watch: {
                // counter: function(value){
                //  var vm = this;
                //  console.log(value);
                //  if (value > 5) {
                //      vm.counter = 0;
                //  }
                // },
                // ES6简写模式
                counter(value){
                    var vm = this;
                    console.log(value);
                    if (value > 5) {
                        vm.counter = 0;
                    }
                }
            },
            // 在 `methods` 对象中定义方法
            methods: {
                result() {
                    return this.counter > 5 ? 'Counter大于5' : 'Counter小于5';
                }
            }
        });
    </script>
</body>
```

```
</html>
```

Class 与 Style 绑定

数据绑定一个常见需求是操作元素的 `class` 列表和它的内联样式。因为它们都是属性 ，我们可以用 `v-bind` 处理它们：只需要计算出表达式最终的字符串。不过，字符串拼接麻烦又易错。因此，在 `v-bind` 用于 `class` 和 `style` 时， Vue.js 专门增强了它。表达式的结果类型除了字符串之外，还可以是对象或数组。

绑定对象形式

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

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue绑定class</title>
    <style>
        .demo{
            width: 100px;
            height: 100px;
            background-color: gray;
            display: inline-block;
        }
        .red{
            background-color: red;
        }

        .purple{
            background-color: purple;
        }
        .pink{
            background-color: pink;
        }
    </style>
</head>
<body>
    <div id="app">
        <div class="demo" @click="attachRed = !attachRed" :class="{red : attachRed}"></
div>
        <div class="demo"></div>
        <div class="demo"></div>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                attachRed: false
            }

        });
    </script>
</body>
</html>
```

上面的语法表示 `red` 的更新将取决于数据属性 `attachRed` 是否为真值 。我们也可以在对象中传入更多属性用来动态切换多个 class 。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue绑定class</title>
    <style>
        .demo{
```

```
            width: 100px;
            height: 100px;
            background-color: gray;
            display: inline-block;
        }
        .red{
            background-color: red;
        }

        .purple{
            background-color: purple;
        }
        .pink{
            background-color: pink;
        }
    </style>
</head>
<body>
    <div id="app">
        <div class="demo" @click="attachRed = !attachRed" :class="{red : attachRed}"></
div>
        <div class="demo" @click="attachRed = !attachRed" :class="divClass"></div>
        <div class="demo"></div>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                attachRed: false
            },
            computed: {
                divClass(){
                    return {
                        red: this.attachRed,
                        purple: !this.attachRed
                    }
                }
            }

        });
    </script>
</body>
</html>
```

> 对象中 `key` 为class样式名称，`value` 为判断是否应用这个class的条件

绑定数组形式

我们可以把一个数组传给 `v-bind:class` ，以应用多个class样式。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue绑定class</title>
    <style>
        .demo{
            width: 100px;
            height: 100px;
            background-color: gray;
            display: inline-block;
        }
        .red{
            background-color: red;
        }

        .purple{
            background-color: purple;
        }
        .pink{
            background-color: pink;
        }
    </style>
</head>
<body>
    <div id="app">
        <div class="demo" @click="attachRed = !attachRed" :class="{red : attachRed}"></
div>
        <div class="demo" @click="attachRed = !attachRed" :class="divClass"></div>
        <div class="demo" :class="[color, {purple: attachRed}]"></div>
        <hr>
        <input type="text" v-model="color">
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                attachRed: false,
                color: 'pink'
            },
            computed: {
                divClass(){
                    return {
                        red: this.attachRed,
                        purple: !this.attachRed
                    }
                }
            }

        });
    </script>
</body>
</html>
```

绑定style对象形式

`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）或短横分隔命名（kebab-case）：

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue绑定class</title>
    <style>
        .demo{
            width: 100px;
            height: 100px;
            background-color: gray;
            display: inline-block;
            margin: 20px;
        }
    </style>
</head>
<body>
    <div id="app">
        <div class="demo" :style="{backgroundColor: color}"></div>
        <div class="demo"></div>
        <div class="demo"></div>
        <hr>
        <input type="text" v-model="color">
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                color: 'pink'
            }
        });
    </script>
</body>
</html>
```

> 在 CSS 样式中 `backgroud-color` 是以短横分隔命名，在 Vue 中统一用驼峰式，即 `backgroundColor`

同样的，对象语法可以结合返回对象的计算属性使用。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue绑定class</title>
    <style>
        .demo{
            width: 100px;
            height: 100px;
            background-color: gray;
```

```
            display: inline-block;
            margin: 20px;
        }
    </style>
</head>
<body>
    <div id="app">
        <div class="demo" :style="{backgroundColor: color}"></div>
        <div class="demo" :style="myStyle"></div>
        <div class="demo"></div>
        <hr>
        <input type="text" v-model="color">
        <input type="text" v-model="width">
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                color: 'pink',
                width: 100,
            },
            computed: {
                myStyle(){
                    return {
                        backgroundColor: this.color,
                        width: this.width  + 'px'
                    }
                }
            }
        });
    </script>
</body>
</html>
```

绑定style数组形式

`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。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue绑定class</title>
    <style>
        .demo{
            width: 100px;
            height: 100px;
            background-color: gray;
            display: inline-block;
            margin: 20px;
        }
    </style>
</head>
<body>
    <div id="app">
        <div class="demo" :style="{backgroundColor: color}"></div>
        <div class="demo" :style="myStyle"></div>
        <div class="demo" :style="[myStyle,{height: width + 'px'}]"></div>
        <hr>
        <input type="text" v-model="color">
        <input type="text" v-model="width">
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                color: 'pink',
                width: 100,
            },
            computed: {
                myStyle(){
                    return {
                        backgroundColor: this.color,
                        width: this.width  + 'px'
                    }
                }
            }
        });
    </script>
</body>
</html>
```

> 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。

条件渲染

v-if和v-else

条件判断是程序中最常见的，在 `Vue.js` ，我们使用 `v-if` 指令实现同样的功能。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue条件</title>
</head>
<body>
    <div id="app">
        <p v-if="show">you can see me !</p>
        <button @click="show = !show">Switch</button>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                show: false,
            },
        });
    </script>
</body>
</html>
```

有 `v-if` 当然也有 `v-else` 添加一个 “else” 块：

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue条件</title>
</head>
<body>
    <div id="app">
        <p v-if="show">you can see me ! <span>Hello</span></p>
        <p v-else>Hello Vue</p>
        <button @click="show = !show">Switch</button>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                show: false,
            },
        });
    </script>
</body>
</html>
```

> if 条件判断时，不管该DOM元素内有多少子元素，都会被一起隐藏掉

v-else-if

`v-else-if`，顾名思义，用作 `v-if` 的 `else-if` 块。可以链式的多次使用，在 Vue 2.10 版本中新增。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue条件</title>
</head>
<body>
    <div id="app">
        <div v-if="type === 'A'">
            A
        </div>
        <div v-else-if="type === 'B'">
            B
        </div>
        <div v-else-if="type === 'C'">
            C
        </div>
        <div v-else>
            Not A/B/C
        </div>
        <hr>
        <input type="text" v-model="type">
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                type: 'A',
            },
        });
    </script>
</body>
</html>
```

<template>中v-if条件组

因为 `v-if` 是一个指令，需要将它添加到一个元素上。但是如果我们想切换多个元素呢？此时我们可以把一个 `<template>` 元素当做包装元素，并在上面使用 `v-if` ，最终的渲染结果不会包含它。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue条件</title>
</head>
<body>
    <div id="app">
        <p v-if="show">you can see me ! <span>Hello</span></p>
        <p v-else>Hello Vue</p>
        <template v-if="show">
            <h1>title</h1>
            <p>template 模块组</p>
        </template>
        <template v-else>
            <h1>title</h1>
            <p>template 模块组1111</p>
        </template>
        <button @click="show = !show">Switch</button>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                show: false,
            },
        });
    </script>
</body>
</html>
```

> `<template>` 元素当做包装元素，最终不会再浏览器中渲染出来

v-show

另一个根据条件展示元素的选项是 `v-show` 指令。用法大体上和 `v-if` 一样。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue条件</title>
</head>
<body>
    <div id="app">
        <p v-if="show">you can see me ! <span>Hello</span></p>
        <p v-show="show">Hello Vue</p>
        <button @click="show = !show">Switch</button>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                show: true,
            },
        });
    </script>
</body>
</html>
```

v-if VS v-show

`v-if` 是真实的条件渲染，因为它会确保条件块在切换当中适当地销毁与重建条件块内的事件监听器和子组件。

`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——在条件第一次变为真时才开始局部编译（编译会被缓存起来）。

相比之下，`v-show` 简单得多——元素始终被编译并保留，只是简单地基于 CSS 切换。

v-if 和 v-show 怎么选择？

一般来说，`v-if` 有更高的切换消耗而 `v-show` 有更高的初始渲染消耗。因此，如果需要频繁切换使用 `v-show` 较好，如果在运行时条件不大可能改变则使用 `v-if` 较好。

列表渲染

v-for

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。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue条件</title>
</head>
<body>
    <div id="app">
        <ul>
            <li v-for="fruit in fruits">{{ fruit }}</li>
        </ul>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                fruits:['香蕉','苹果','芒果','火龙果'],
                website:[
                    {name: '百度' , url: 'http://www.baidu.com'},
                    {name: '晚黎' , url: 'http://www.iwanli.me'},
                    {name: 'Google' , url: 'http://www.google.com'},
                ]
            },
        });
    </script>
</body>
</html>
```

`v-for` 还支持一个可选的第二个参数为当前项的索引。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue条件</title>
</head>
<body>
    <div id="app">
        <ul>
            <li v-for="(fruit,i) in fruits">{{i}}-{{ fruit }}</li>
        </ul>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                fruits:['香蕉','苹果','芒果','火龙果'],
                website:[
                    {name: '百度' , url: 'http://www.baidu.com'},
                    {name: '晚黎' , url: 'http://www.iwanli.me'},
                    {name: 'Google' , url: 'http://www.google.com'},
```

```
                ]
            },
        });
    </script>
</body>
</html>
```

> 第二个参数名称可以任意取名，但要保证在模板渲染的时候要跟索引名称保持一致

Template v-for

如同 `v-if` 模板，你也可以用带有`v-for` 的 `<template>` 标签来渲染多个元素块。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue条件</title>
</head>
<body>
    <div id="app">
        <template v-for="(fruit,index) in fruits">
            <h1>{{fruit}}({{index}})</h1>
            <hr>
        </template>
    </div>
    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                fruits:['香蕉','苹果','芒果','火龙果'],
                website:[
                    {name: '百度' , url: 'http://www.baidu.com'},
                    {name: '晚黎' , url: 'http://www.iwanli.me'},
                    {name: 'Google' , url: 'http://www.google.com'},
                ]
            },
        });
    </script>
</body>
</html>
```

对象迭代-v-for

`v-for` 通过一个对象的属性来迭代，同时支持当前项的索引。

```
<!DOCTYPE html>
<html>
<head>
    <title>Vue条件</title>
</head>
<body>
    <div id="app">
        <ul>
            <li v-for="web in website">
                <div v-for="(value, key, index) in web">{{key}}:{{value}}({{index}})</d
iv>
            </li>
        </ul>
    </div>

    <script type="text/javascript" src="../js/vue.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: '#app',
            data: {
                fruits:['香蕉','苹果','芒果','火龙果'],
                website:[
                    {name: '百度' , url: 'http://www.baidu.com'},
                    {name: '晚黎' , url: 'http://www.iwanli.me'},
                    {name: 'Google' , url: 'http://www.google.com'},
                ]
            },
        });
    </script>
</body>
</html>
```