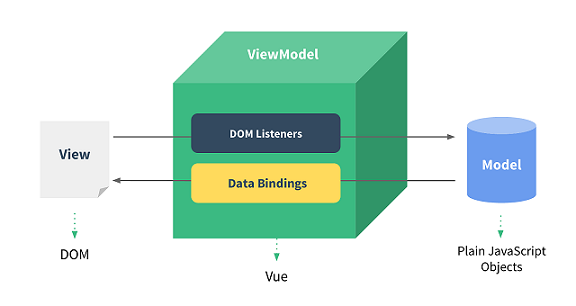
### what

　　Vue.js是一个构建数据驱动的web界面的库。技术上，它重点集中在MVVM模式的ViewModel层，因此它非常容易学习，非常容易与其它库或已有项目整合。

　　Vue.js的目标是通过尽可能简单的API实现响应的数据绑定和组合的视图组件。

　　Vue.js 的核心是一个响应的数据绑定系统，它让数据与DOM保持同步非常简单。在使用jQuery手工操作DOM时，我们的代码常常是命令式的、重复的与易错的。Vue.js拥抱数据驱动的视图概念。通俗地讲，它意味着我们在普通HTML模板中使用特殊的语法将DOM “绑定”到底层数据。



### 安装

**独立版本**

　　直接下载并用 < script > 标签引入，Vue会被注册为一个全局变量。如下代码，这样就可以在脚本中使用Vue.js了。

1. <script src="lib/vue.js"></script>

**CDN**

　　也可以在 [jsdelivr](http://cdn.jsdelivr.net/vue/1.0.10/vue.min.js" \t "http://www.hubwiz.com/class/_Blank)或 [cdnjs](http://cdnjs.cloudflare.com/ajax/libs/vue/1.0.10/vue.min.js" \t "http://www.hubwiz.com/class/_Blank)获取 (版本更新可能会略滞后)。

**NPM**

　　在用 Vue.js 构建大型应用时推荐使用 NPM 安装，NPM 能很好地和诸如 Webpack 或 Browserify 的 CommonJS 模块打包器配合使用。Vue.js 也提供配套工具来开发单文件组件。

1. $ npm install vue
2. `# 获取CSP兼容版本：
3. `$ npm install vue@csp
4. `# 获取最新开发版本(来自于GitHub):
5. $ npm install yyx990803/vue#dev

### Hello World

　　现在就让我们来写第一个vue.js的实例。如下代码：

html代码：

1. <div id="demo">
2. {{ message }}
3. </div>

JavaScript代码:

1. new Vue({
2. el: '#demo',
3. data: {
4. message: 'Hello World!'
5. }
6. })

　　上面代码中div中的部分 {{ message }}为数据绑定，我们将会在后面的学习中讲到。而vue.js代码是实例化一个Vue对象。在使用vue之前必须要实例化。

### 构造器

　　每个Vue.js应用的起步都是通过构造函数Vue创建一个Vue的根实例：

1. var vm = new Vue({
2. // 选项
3. })

　　一个Vue实例其实正是一个MVVM模式中所描述的 ViewModel - 因此在文档中经常会使用vm这个变量名。

### 属性与方法

　　每个Vue实例都会代理其data对象里所有的属性，如下代码：

1. var data = { a: 1 }
2. var vm = new Vue({
3. data: data
4. })
5. //vm.a === data.a -> true
6. // 设置属性也会影响到原始数据
7. vm.a = 2
8. // data.a -> 2
9. // ... 反之亦然
10. data.a = 3
11. //vm.a -> 3

　　注意只有这些被代理的属性是响应的。如果在实例创建之后添加新的属性到实例上，它不会触发视图更新。

　　除了前面这些数据属性，Vue 实例还有一些有用的实例属性与方法。这些属性与方法都有前缀 $，以便与代理的数据属性区分。例如：

1. var data = { a: 1 }
2. var vm = new Vue({
3. el: '#example',
4. data: data
5. })
7. vm.$data === data // -> true
8. vm.$el === document.getElementById('example') // -> true
10. // $watch 是一个实例方法
11. vm.$watch('a', function (newVal, oldVal) {
12. // 这个回调将在 `vm.a` 改变后调用
13. })

### 插值

　　数据绑定最基础的形式是文本插值，使用 {{}} 语法（双大括号）：

1. <span>Message: {{ msg }}</span>

　　{{ msg }} 标签会被相应数据对象的 msg 属性的值替换。每当这个属性变化时它也会更新。

　　也可以只处理单次插值，今后的数据变化就不会再引起插值更新了：

1. <span>This will never change: {{\* msg }}</span>

如下JavaScript代码:

1. var data={msg:'Hello Vue.js!'};
2. new Vue({
3. el: '#demo',
4. data: data
5. })
6. data.msg="Hello World!";

### 原始的HTML

　　双大括号标签将数据解析为纯文本而不是HTML。为了输出真的HTML字符串，需要用三大括号标签：

1. <div>{{{ msg }}}</div>

　　如下javascript代码：

1. var data={msg:'<p>Hello Vue.js!</p>'};
2. new Vue({
3. el: '#demo',
4. data: data
5. })

### HTML特性

　　双大括号标签也可以用在 HTML 特性 (Attributes) 内：

1. <div id="{{ id }}"></div>

javascript代码如下：

1. var data={id:'demo'};
2. new Vue({
3. el: 'div',
4. data: data
5. })

我们去查看HTML源码，是不是id已经变成我们设置的id了。

### JavaScript表达式

　　Vue.js 在数据绑定内支持全功能的JavaScript表达式：

1. {{ number + 1 }}
2. {{ ok ? 'YES' : 'NO' }}
3. {{ message.split('').reverse().join('') }}

### 过滤器

　　Vue.js 允许在表达式后添加可选的“过滤器 (Filter) ”，以“管道符(|)”指示。过滤器本质上是一个函数，这个函数会接收一个值，将其处理并返回。

1. {{ message | uppercase }}

　　这里我们将表达式 message 的值“管输（pipe）”到内置的 uppercase 过滤器，这个过滤器其实只是一个 JavaScript 函数，返回大写化的值。Vue.js 提供数个内置过滤器，在后面我们会谈到如何开发自己的过滤器。

　　可以串联多个过滤器：

1. {{ message | filterA | filterB }}

html代码：

1. <div id='demo'>
2. <span>{{msg | lowercase | capitalize}}</span>
3. </div>

javaScript代码：

1. var data={msg:'heLLO!'};
2. new Vue({
3. el: '#demo',
4. data: data
5. })

运行结果为：Hello

### 指令

　　Vue.js指令 (Directives) 是特殊的带有前缀 v- 的特性。本质是模板中出现的特殊标记，让处理模板的库知道需要对这里的DOM元素进行一些对应的处理。指令的职责就是当其表达式的值改变时把某些特殊的行为应用到 DOM 上。

1. <p v-if="msg">Hello!</p>

　　这里 v-if 指令将根据表达式 msg 值的真假删除/插入 < p > 元素。

　　在Vue.js中为我们提供了一些指令：v-text，v-html，v-model，v-on，v-else等等，同学们可以去查看Vue.js的指令api(cn.vuejs.org/api/#指令)。

javascript代码：

1. var data={msg:0};
2. new Vue({
3. el: '#demo',
4. data: data
5. })

### Class对象语法

　　可以传给 v-bind:class 一个对象，以动态地切换class。注意 v-bind:class 指令可以与普通的 class 特性共存。

html代码：

1. <div class="static" v-bind:class="{ 'class-a': isA, 'class-b': isB }"></div>

javascript代码：

1. var vm = new Vue({
2. el: 'div',
3. data: {
4. isA: true,
5. isB: false
6. }
7. })

渲染为：

1. <div class="static class-a">
2. </div>

　　当 isA 和 isB 变化时，class 列表将相应地更新。例如，如果 isB 变为 true，class 列表将变为 "static　class-a class-b"。

### Class数组语法

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

html代码：

1. <div v-bind:class="[classA, classB]">

js代码：

1. var vm = new Vue({
2. el: 'div',
3. data: {
4. classA: 'class-a',
5. classB: 'class-b'
6. }
7. })

渲染为：

1. <div class="class-a class-b"></div>

### Style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。

1. <div v-bind:style="[styleObjectA, styleObjectB]">Style 数组语法</div>

javascript代码：

1. var vm = new Vue({
2. el: 'div',
3. data: {
4. styleObjectA: {
5. color: 'red'
6. },
7. styleObjectB: {
8. fontSize: '30px'
9. }
10. }
11. })

渲染为：

1. <div style="color: red; font-size: 30px;">Style 数组语法</div>

### v-if

　　在字符串模板中，如Handlebars，我们得像这样写一个条件块：

1. <!-- Handlebars 模板 -->
2. {{#if ok}}
3. <h1>Yes</h1>
4. {{/if}}

　　在 Vue.js，我们使用 v-if 指令实现同样的功能：

1. <h1 v-if="ok">Yes</h1>

　　也可以用 v-else 添加一个 “else” 块：

1. <h1 v-if="ok">Yes</h1>
2. <h1 v-else>No</h1>

## template v-if

　　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。

1. <template v-if="ok">
2. <h1>Title</h1>
3. <p>Paragraph 1</p>
4. <p>Paragraph 2</p>
5. </template>

当ok为真值时，渲染为：

1. <h1>Title</h1>
2. <p>Paragraph 1</p>
3. <p>Paragraph 2</p>

## v-show

　　另一个根据条件展示元素的选项是 v-show 指令。用法大体上一样：

1. <h1 v-show="ok">Hello!</h1>

　　不同的是有 v-show 的元素会始终渲染并保持在 DOM 中。v-show 是简单的切换元素的 CSS 属性 display。

## v-else

　　可以用 v-else 指令给 v-if 或 v-show 添加一个 “else 块”：

1. <div v-if="Math.random() > 0.5">
2. Sorry
3. </div>
4. <div v-else>
5. Not sorry
6. </div>

　　v-else 元素必须立即跟在 v-if 或 v-show元素的后面，否则它不能被识别。

## v-for

　　可以使用 v-for 指令基于一个数组渲染一个列表。这个指令使用特殊的语法，形式为 item in items，items 是数据数组，item 是当前数组元素的别名：

1. <ul id="example">
2. <li v-for="item in items">
3. {{ item.message }}
4. </li>
5. </ul>

js代码：

1. var example1 = new Vue({
2. el: '#example',
3. data: {
4. items: [
5. { message: 'one' },
6. { message: 'two' }
7. ]
8. }
9. })

　　在 v-for 块内我们能完全访问父组件作用域内的属性，另有一个特殊变量 $index，正如你猜到的，它是当前数组元素的索引：

1. <ul id="example">
2. <li v-for="item in items">
3. {{ parentMessage }} - {{ $index }} - {{ item.message }}
4. </li>
5. </ul>

js代码：

1. var example2 = new Vue({
2. el: '#example',
3. data: {
4. parentMessage: 'Parent',
5. items: [
6. { message: 'one' },
7. { message: 'two' }
8. ]
9. }
10. })

## template v-for

　　类似于 template v-if，也可以将 v-for 用在 template 标签上，以渲染一个包含多个元素的块。例如：

1. <ul>
2. <template v-for="item in items">
3. <li>{{ item.msg }}</li>
4. <li class="divider"></li>
5. </template>
6. </ul>

## Text

　　可以在表单的input 元素上使用v-model 指令来创建双向数据绑定。它会根据input元素的类型自动选取正确的绑定模式。

1. <span>Message is:{{msg}}</span>
2. <input type="text" v-model="msg" placeholder="edit me">

js代码：

1. var vm = new Vue({
2. el: 'div',
3. data: {
4. msg:"hello world"
5. }
6. })

其中placeholder为默认显示提示。

## Checkbox

当我们要绑定值时可以用v-bind来绑定，如下代码：

1. <div>
2. <input
3. type="checkbox"
4. v-model="toggle"
5. v-bind:true-value="a"
6. v-bind:false-value="b">
7. </div>

当checkbox为true时，值为a；当为fals时，值为b。

var vm = new Vue({

el:'div',

data:{

a:'a',

b:'b'

}

})

## Radio

1. <input type="radio" id="one" value="One" v-model="picked">
2. <label for="one">One</label>
3. <br>
4. <input type="radio" id="two" value="Two" v-model="picked">
5. <label for="two">Two</label>
6. <br>
7. <span>Picked: {{ picked }}</span>

我们可以用v-bind指令来绑定数据。如下代码：

1. <input type="radio" v-model="pick" v-bind:value="a">

## select

单选，如下代码：

1. <select v-model="selected">
2. <option selected>A</option>
3. <option>B</option>
4. <option>C</option>
5. </select>
6. <span>Selected: {{ selected }}</span>

多选，如下代码：

1. <select v-model="selected" multiple>
2. <option selected>A</option>
3. <option>B</option>
4. <option>C</option>
5. </select>
6. <br>
7. <span>Selected: {{ selected | json }}</span>

动态选项，用 v-for 渲染，如下代码：

1. <select v-model="selected">
2. <option v-for="option in options" v-bind:value="option.value">
3. {{ option.text }}
4. </option>
5. </select>
6. <span>Selected: {{ selected }}</span>

js代码如下：

1. new Vue({
2. el: 'div',
3. data: {
4. selected: 'A',
5. options: [
6. { text: 'One', value: 'A' },
7. { text: 'Two', value: 'B' },
8. { text: 'Three', value: 'C' }
9. ]
10. }
11. })

　　对象的格式应为 {text:'', value:''}。这允许你把展示的文字和其背后对应的值区分开来。

也可以用v-bind来绑定数据，如我们要绑定一个数值型的数据，如下代码：

1. <select v-model="selected">
2. <!-- 对象字面量 -->
3. <option v-bind:value="{ number: 123 }">123</option>
4. </select>