# Vue学习笔记

## 介绍

### 1-1、Vue.js是什么？

Vue.Js是一套用于构建用户界面的**渐进式框架 - -** 来自官方介绍。

什么是渐进式？ 假设我们现有一台服务器端的应用，我们可以将vue作为该应用的一部分其纳入其中，来进行丰富的交互体验。或者如果我们希望将更多业务逻辑放到前端来实现，那么vue的核心库以及生态系统也可以满足我们各式的需求。

Vue同大多数的javascript框架一样，允许将网页分割成各种复用组件，每个组件包含自己的html 、css 、 javascript，以便渲染网页中相应的地方。如图-1



图-1

### 1-2、起步

我们可以创建一个html文件，然后引入vue服务器的js，

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

1、然后我们在body标签里面编写html和js，如下：

<body>  
 <div id="example">  
 <p v-bind:title="message">{{message}}</p>  
 <p>{{reverseMessage}}</p>  
 <p>{{test}}</p>  
 <p v-if="seen">你现在看到我1</p>  
 <p v-if="message">你现在看到我2</p>  
 <p v-if="test">你现在看到我2</p>  
 </div>  
</body>  
<!-- 开发环境版本，包含了用帮助的命令行警告 -->  
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
<script>  
 **var** vm = **new** Vue({  
 el:'#example',  
 data:{  
 message:'张三',  
 seen:**false** },  
 computed: {  
 reverseMessage:**function**(){  
 **return this**.message.split('').reverse().join('');  
 }  
 }  
 })  
</script>

1. 上面的代码，我们通过var vm = new Vue({}) 来创建叫“vm”一个vue的实例，图中的属性也很好理解，具体如下：

“el”属性，全拼代表“element”，也就是元素，我们要绑定的dom对象，也就是上面div标签的id。

“data”属性，vue的实例数据对象，vue会通过递归的方法将data属性转换为getter/setter(后面会解释)，从而让data的属性能够响应数据变化。**注意了，对象必须是纯粹的对象（含有零个或多个的key/value对），否则整个实例会创建失败.** 我们实例创建后，可以通过vm.$data访问原始数据对象。同时Vue代理了data对象上面的所有属性，因此vm.$data 等价于 vm.a。代码如下：

**var** data = { a: 1 }  
// 直接创建一个实例  
**var** vm = **new** Vue({  
 data: data  
})  
vm.a // => 1  
vm.$data === data // => true  
  
// Vue.extend() 中 data 必须是函数  
**var** Component = Vue.extend({  
 data: **function** () {  
 **return** { a: 1 }  
 }  
})

“computed”属性，也叫“计算属性”。官方的解释是这样的“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”。怎了理解嘞？我们知道了表达式{{ xxx }}这种写法，能够得到数据值。但如果我们想对数据做一些复杂逻辑且直接在表达式上面处理，这会增加我们对模板维护，具体看下面代码：

<body>  
 <div id="example">  
 <!--1、直接在模板中绑定表达式-->  
 <p>{{message.split('').reverse().join('')}}</p>  
 <!--2、运用计算属性-->  
 <p>{{reverseMessage}}</p>  
 <!--3、运用methods方式-->  
 <p>{{methodMessage()}}</p>  
 </div>  
</body>  
<!-- 开发环境版本，包含了用帮助的命令行警告 -->  
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
<script>  
 **var** vm = **new** Vue({  
 el:'#example',  
 data:{  
 message:'张三',  
 seen:**false** },  
 computed: {  
 reverseMessage:**function**(){  
 **return this**.message.split('').reverse().join('');  
 }  
 },  
 methods:{  
 methodMessage:**function**(){  
 **return this**.message.split('').reverse().join('');  
 }  
 }  
 })  
  
</script>

假设我们想对反向显示“message”，我们需要分割字符，然后反向通过“join”方法加入，上述代码中，对于方法1，我们是直接在表达式中做处理，看起来是不是比较复杂，模板不在清晰和简单，那如果我们想要多处用到这个反向显示，模板就显得很拥挤了，这就是为什么会有“computed”属性的存在。然后上述代码还出现了“methods”属性，他与“computed”属性调用的区别就是后面多了个“()”，原理区别：computed方法处理复杂逻辑，它是基于依赖缓存，当依赖发生改变时会重新取值。methods也可以是想实现的效果，但methods在重新渲染的时候会重新调用执行，在性能上computed比methods好，当不需要缓存时可用methods。

### 1-3、声明式渲染

Vue.Js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的系统，如1-2中的{{message}}。这看起来跟渲染一个字符串模板非常相似，但是vue却在背后做了大量工作，比如现在的数据和DOM已经被建立了关联，所有东西都是响应式的。我们可以通过vm.message的值来进行测试（注：此处的“vm”是上面的vm实例），测试结果如图1-2、1-3：

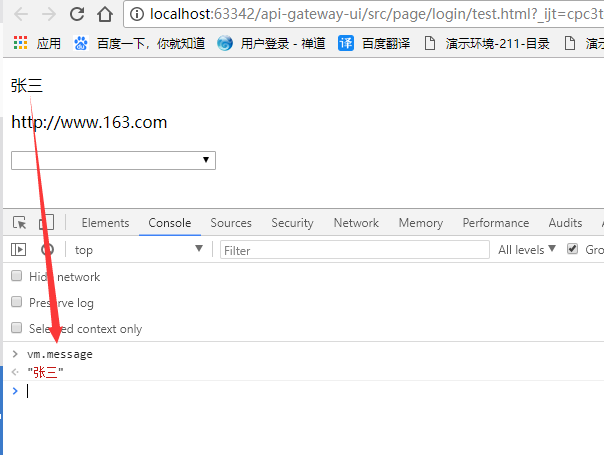


图 1-2

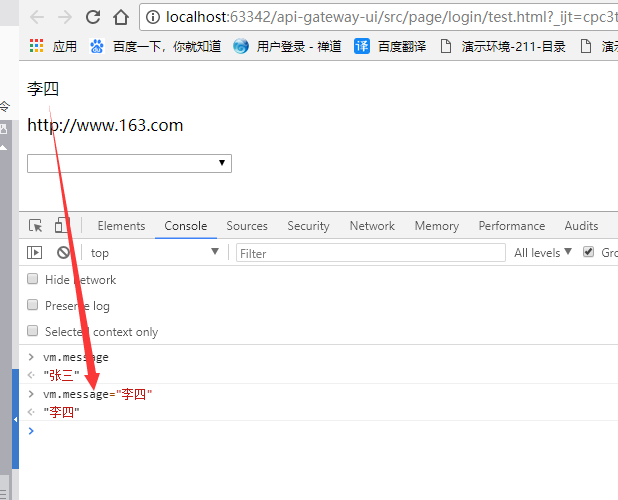


图 1-3

### 1-4、指令小接触

<body>  
 <div id="example">  
 <p v-if="seen">你现在看到我1</p>  
 <p v-if="message" v-bind:title="message">你现在看到我2</p>

<ul>  
 <li v-for="list in eachData">{{list.text}}</li>  
 </ul>  
 </div>  
</body>  
<!-- 开发环境版本，包含了用帮助的命令行警告 -->  
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
<script>  
 **var** vm = **new** Vue({  
 el:'#example',  
 data:{  
 message:'google',  
 seen:**false**,  
 name:'http://www.google.com',  
 eachData: [  
 { text: '学习 JavaScript' },  
 { text: '学习 Vue' },  
 { text: '整个牛项目' }  
 ]  
 },

computed: {  
 },  
 methods:{  
 }  
 })  
</script>

上述代码中，我们观察到一共出现了3种指令，“v-bind”、“v-if”、“v-for”。

“v-bind”指令意思是绑定，用于绑定dom元素的属性，如上面对p标签绑定“title”，这样我们就可以实现鼠标经过p标签显示提示文本内容这一交互。

“v-if”指令用于条件渲染，与jsp的“c-if”标签的功能类似，区别在于语法，v-if它是一个指令，通过绑定data的某个属性来进行条件渲染，而且我们可以对任意的html标签直接绑定使用。

“v-for”指令用于循环渲染，与js的“for( var i in data)”语法一致。同样也可以对任意html标签直接绑定使用。附上代码效果图1-4：



图 1-4

同样的，这些指令也是响应式的，看图1-5：

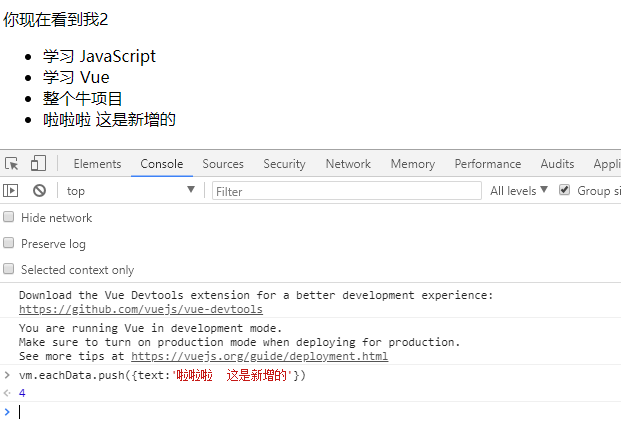


图 1-5

### 1-5、组件化应用构建

组件系统是vue的一个重要概念，它是一种抽象，允许我们使用小型、独立和通常复用的组件构建大型应用。在vue中几乎任意类型的应用界面都可以抽象为一个组件树（图1-6）：

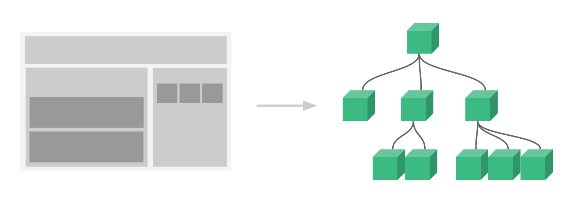


图 1-6（来自vue官网）

那么怎么使用注册并使用组件呢？

在vue里，一个组件本质上是一个拥有预定义选项的一个vue实例。它的注册方法如下：

Vue.component('test-item',{  
 props:['todo'],  
 template:'<li>这是某某代办项</li>'  
})

**var** vm = **new** Vue({  
 el:'#example'  
})

上述代码，我们注册了一个名为“test-item”的组件，然后可以像标签一样的写法，去调用这个组件：

<body>  
 <div id="example">  
 <ol>  
 <test-item></test-item>  
 </ol>  
 </div>  
</body>

附上效果图-1-7：

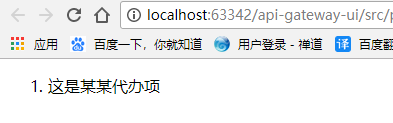


图1-7

但是呢，这个组件太死板了，每次调用这个组件为渲染相同的文本，不够灵活，因此我们可以做下如下修改，先看看代码：

<body>  
 <div id="example">  
 <ol>  
 <test-item v-for="index in dataList" v-bind:todo="index">

</test-item>  
 </ol>  
 </div>  
</body>  
<!-- 开发环境版本，包含了用帮助的命令行警告 -->  
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
<script>  
 Vue.component('test-item',{  
 props:['todo'],  
 template:'<li>{{todo.text}}</li>'  
 })  
 **var** vm = **new** Vue({  
 el:'#example',  
 data:{  
 dataList:[  
 {id:0,text:'张三'},  
 {id:1,text:'李四'},  
 {id:2,text:'王五'}  
 ]  
 }  
 })  
</script>

首先，我们实例化一个叫vm的实例，然后嘞绑定了一个叫dataList的json数组的data属性，接下来为“test-item”组件绑定v-for指令循环输出文本值，这里我们用到了“props”属性，他的意思是继承自父级的自定义属性，如图中的“todo”。这点，我觉得跟jquery的prop和attr一样。“todo.text”===“$(xxx).attr(‘xxx’)”;附上代码效果图1-8：

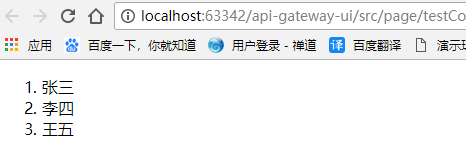


图 1-8

## Vue实例

### 2-1、创建一个vue实例

每个Vue应用都是通过Vue函数创建一个新的Vue实例开始的：

**var** vm = **new** Vue({})

每当我们创建一个实例时，可以传入一个选项对象。我们只需要明白所有的Vue组件都是Vue实例，并且接受相同的选项对象。

### 2-2、数据与方法

当一个Vue实例被创建时，它向Vue的响应式系统加入了其data对象中能找到的所有的属性。当这些属性的值发生改变时，试图将会产生“响应”,既匹配更新为新的值。

<script>  
 **var** data = {a:1};  
 **var** vm = **new** Vue({  
 data:data  
 });  
 console.log("vm.a="+vm.a);  
 console.log("data.a="+data.a);  
</script>

当数据改变时，视图会重新渲染。代码效果如图2-1：



图 2-1

值得我们注意的是，只有当实例被创建时data中存在的属性才是响应式的，也就是说如果我们添加一个新的属性，比如：

vm.b = 'lala';

那么对b的改动将不会触发任何视图的更新。为了避免这种情况，我们可以预先定义一些属性，只需要对他们赋空或者给个初始值就可以了。

**var** vm = **new** Vue({  
 data:{  
 newTodoText: '',  
 visitCount: 0,  
 hideCompletedTodos: **false**,  
 todos: [],  
 error: **null** }  
});

如果我们想阻止系统的响应式，可以只用“Object.freeze( )”,比如：

**var** vm = **new** Vue({  
 el: '#app',  
 data:{  
 foo: 'bar'  
 }  
})  
Object.freeze(vm);

代码效果图2-2：

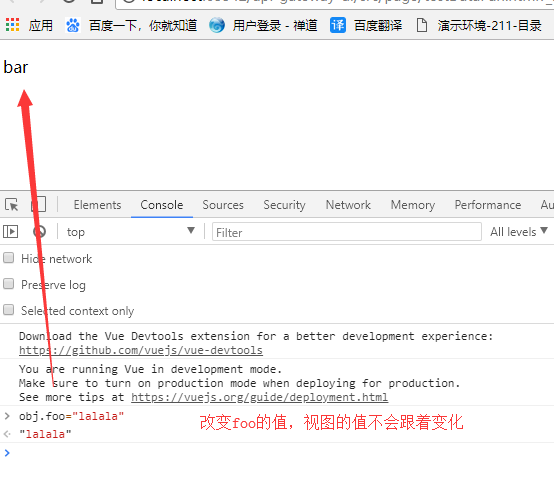


图 2-2

除了数据属性，Vue实例还暴露了一些有用的实力属性与方法。它们都有前缀“$”，以便与用户定义的属性区分开来，例如：

<script>  
 **var** data = {a:1};  
 **var** vm = **new** Vue({  
 el:'#app',  
 data:data  
 })  
</script>

代码效果图2-3：

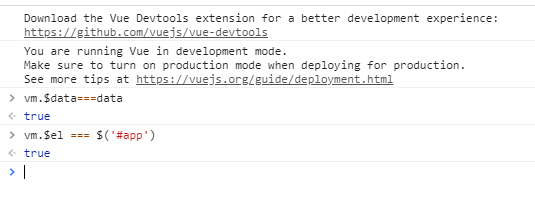


图 2-3

### 2-3、实例生命周期钩子（重）

钩子？又是一个新概念，其实很好理解，与ajax的success、complete、error相似。每个Vue实例在被创建的时候都要经过一系列的初始化过程 - - 例如需要设置数据监听、请求数据、编译模板，将实例挂载到DOM并在数据变化时更新DOM等。Vue在这个过程中为我们定义好了不同阶段的生命周期钩子函数。

比如“created”钩子，英语翻译下就是创建，这个钩子可以用来在一个实例被创建完成后执行调用。

<script>  
 **new** Vue({  
 data: {  
 a: 1  
 },  
 created: **function** () {  
 // `this` 指向 vm 实例  
 console.log('a is: ' + **this**.a)  
 }  
 })  
</script>

结果输出为：图2-4

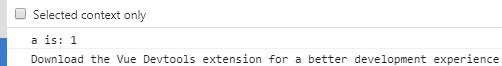
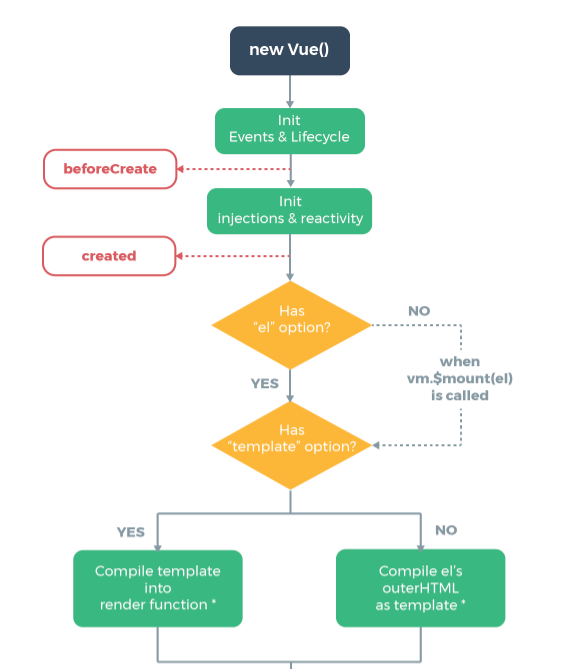


图2-4

也有一些其他的钩子，在实例生命周期的不同阶段被调用，如“mounted”、“updated”和“destroyed”。生命周期钩子的this上下文指向调用它的Vue实例。

为了有个大体的概念，Vue提供了生命周期阶段图2-5：



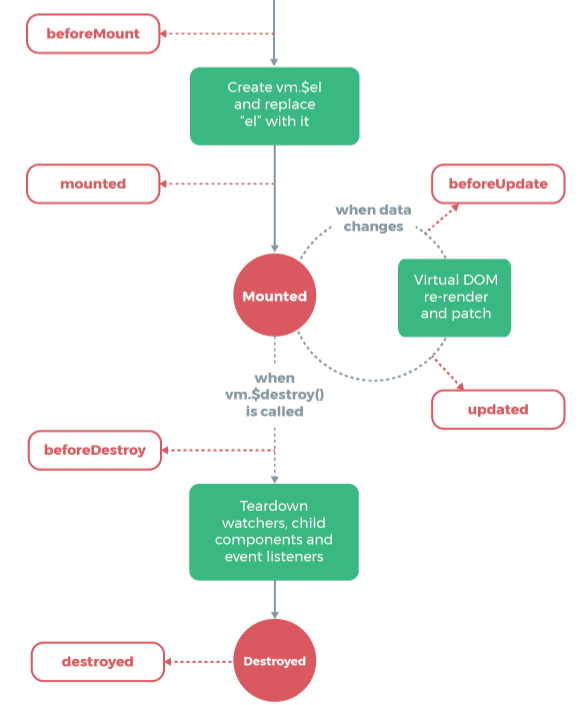


图 2-5