

学习目标

- 会创建Vue实例，知道Vue的常见属性
- 会使用Vue的生命周期的钩子函数
- 会使用vue常见指令
- 会使用vue计算属性和watch监控
- 会编写Vue组件
- 掌握组件间通信
- 了解vue-router使用

0.前言

前几天我们已经对后端的技术栈有了初步的了解、并且已经搭建了整个后端微服务的平台。接下来要做的事情就是功能开发了。但是没有前端页面，我们肯定无从下手，因此今天我们就来了解一下前端的一些技术，完成前端页面搭建。

先聊一下前端开发模式的发展。

静态页面

最初的网页以HTML为主，是纯静态的网页。网页是只读的，信息流只能从服务端到客户端单向流通。开发人员也只关心页面的样式和内容即可。

异步刷新，操作DOM

1995年，网景工程师Brendan Eich 花了10天时间设计了JavaScript语言。

随着JavaScript的诞生，我们可以操作页面的DOM元素及样式，页面有了一些动态的效果，但是依然是以静态为主。

ajax盛行：

- 2005年开始，ajax逐渐被前端开发人员所重视，因为不用刷新页面就可以更新页面的数据和渲染效果。
- 此时的开发人员不仅仅要编写HTML样式，还要懂ajax与后端交互，然后通过JS操作Dom元素来实现页面动态效果。比较流行的框架如Jquery就是典型代表。

MVVM，关注模型和视图

2008年，google的Chrome发布，随后就以极快的速度占领市场，超过IE成为浏览器市场的主导者。

2009年，Ryan Dahl在谷歌的Chrome V8引擎基础上，打造了基于事件循环的异步IO框架：Node.js。

- 基于事件循环的异步IO
- 单线程运行，避免多线程的变量同步问题
- JS可以编写后台代码，前后台统一编程语言

node.js的伟大之处不在于让JS迈向了后端开发，而是构建了一个庞大的生态系统。

NodeJs Pakage Manage, NodeJs包管理工具, 类似于Maven

2010年, **NPM**作为node.js的包管理系统首次发布, 开发人员可以遵循Common.js规范来编写Node.js模块, 然后发布到NPM上供其他开发人员使用。目前已经是世界最大的包模块管理系统。

随后, 在node的基础上, 涌现出了一大批的前端框架:

框架	架构	最初发布时间	GitHub Stars
Knockout	MVVM	2010年7月	stars 8k
Backbone	MVP	2010年10月	stars 26k
Angular	MVC->MVVM	2010年10月	stars 24k
Ember	MVVM	2011年12月	stars 18k
Meteor	MVC	2012年1月	stars 38k
Vue	MVVM	2014年7月	stars 55k

MVVM模式

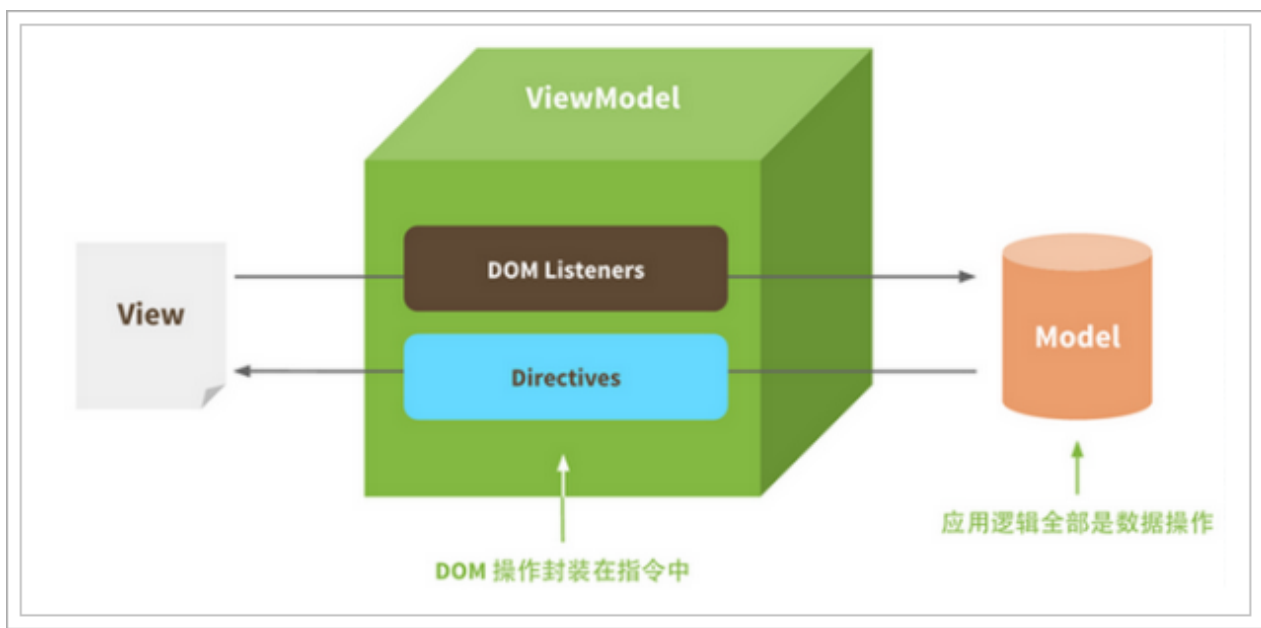
- M: 即Model, 模型, 包括数据和一些基本操作
- V: 即View, 视图, 页面渲染结果
- **VM**: 即**View-Model**, 模型与视图间的双向操作 (无需开发人员干涉)

在MVVM之前, 开发人员从后端获取需要的数据模型, 然后通过DOM操作Model渲染到View中。而后当用户操作视图, 我们还需要通过DOM获取View中的数据, 然后同步到Model中。

而MVVM中的VM要做的事情就是把DOM操作完全封装起来, 开发人员不用再关心Model和View之间是如何互相影响的:

- 只要我们Model发生了改变, View上自然就会表现出来。
- 当用户修改了View, Model中的数据也会跟着改变。

把开发人员从繁琐的DOM操作中解放出来, 把关注点放在如何操作Model上。



而我们今天要学习的，就是一款MVVM模式的框架：Vue

1.认识Vue

Vue (读音 /vju:/, 类似于 **view**) 是一套用于构建用户界面的**渐进式框架**。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与**现代化的工具链**以及各种**支持类库**结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

前端框架三巨头：Vue.js、React.js、AngularJS，vue.js以其轻量易用著称，vue.js和React.js发展速度最快，AngularJS还是老大。

官网：<https://cn.vuejs.org/>

参考：<https://cn.vuejs.org/v2/guide/>



渐进式 JavaScript 框架



WHY VUE.JS?

起步



GITHUB

Special Sponsors



Function as a Service Platform and Library



The fastest way to share code

易用

已经会了 HTML、CSS、JavaScript?
即刻阅读指南开始构建应用!

灵活

不断繁荣的生态系统, 可以在一个
库和一套完整框架之间自如伸缩。

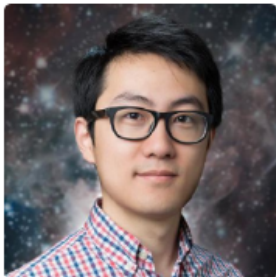
高效

20kB min+gzip 运行大小
超快虚拟 DOM
最省心的优化

Git地址: <https://github.com/vuejs>

<https://github.com/yyx990803>

Search GitHub Pull requests Issues Marketplace Explore



Evan You
yyx990803
Creator of @vuejs, previously @meteor & @google
[Follow](#)

Block or report user

New Jersey / China
<http://evanyou.me>

Overview Repositories 141 Stars 779 Followers 29.2k Following 90

Pinned repositories

- vuejs/vue**
A progressive, incrementally-adoptable JavaScript framework for building UI on the web.
JavaScript ★ 88.7k 🍴 13k
- vuejs/vue-router**
The official router for Vue.js.
JavaScript ★ 9.3k 🍴 2.4k
- vuejs/vuex**
Centralized State Management for Vue.js.
JavaScript ★ 13.5k 🍴 4.2k
- vuejs/vue-cli**
CLI for rapid Vue.js development
JavaScript ★ 10.5k 🍴 1.5k
- vuejs/vue-devtools**
Browser devtools extension for debugging Vue.js applications.
JavaScript ★ 7.6k 🍴 1k
- vuejs/vue-loader**
Webpack loader for Vue.js components
JavaScript ★ 2.9k 🍴 522

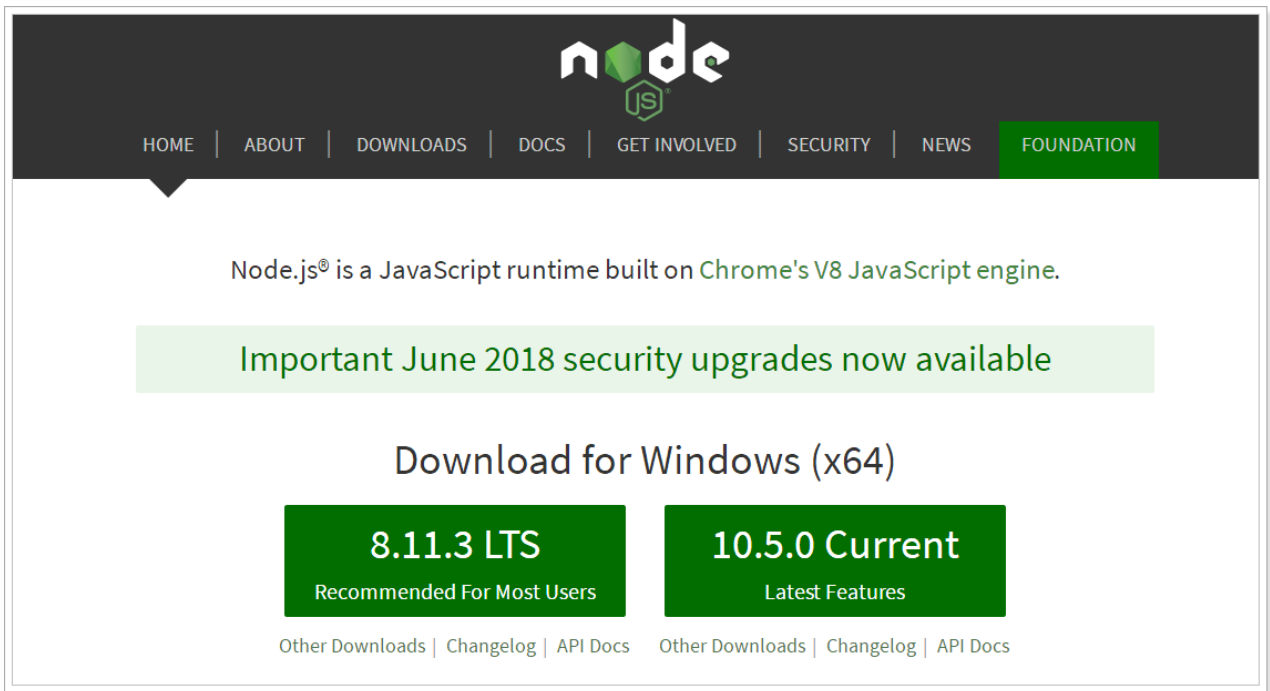
尤雨溪, Vue.js 创作者, Vue Technology创始人, 致力于Vue的研究开发。

2.Node和NPM

前面说过，NPM是Node提供的模块管理工具，可以非常方便的下载安装很多前端框架，包括jQuery、AngularJS、Vuejs都有。为了后面学习方便，我们先安装node及NPM工具。

2.1.下载Node.js

下载地址：<https://nodejs.org/en/>



推荐下载LTS版本。

课程中采用的是8.11.3版本。也是目前最新的。大家自行下载或者使用课前资料中提供的安装包。然后下一步安装即可。

完成以后，在控制台输入：

```
node -v
```

看到版本信息：

```
C:\Users\joedy>node -v
v8.11.3

C:\Users\joedy>
```

2.2.NPM

Node自带了NPM了，在控制台输入 `npm -v` 查看：

```
C:\Users\joedy>npm -v
5.6.0

C:\Users\joedy>
```

npm默认的仓库地址是在国外网站，速度较慢，建议大家设置到淘宝镜像。但是切换镜像是比较麻烦的。推荐一款切换镜像的工具：`nrm`

我们首先安装nrm，这里 `-g` 代表全局安装。可能需要一点儿时间

```
npm install nrm -g
```

```
C:\Users\joedy>npm install nrm -g
npm WARN deprecated coffee-script@1.7.1: CoffeeScript on NPM has moved to "coffeescript"
npm WARN notice [SECURITY] open has the following vulnerability: 1 critical. Go here for more info: https://nodesecurity.io/advisories?search=open&version=0.0.5 - Run `npm i npm@latest -g` to update your npm version, and then `npm audit` to get more info.
C:\Users\joedy\AppData\Roaming\npm\nrm -> C:\Users\joedy\AppData\Roaming\npm\node_modules\npm\node_modules\nrm
+ nrm@1.0.2
added 322 packages in 126.514s

C:\Users\joedy>
```

然后通过 `nrm ls` 命令查看npm的仓库列表，带*的就是当前选中的镜像仓库：

```
C:\Users\joedy>nrm ls

* npm ---- https://registry.npmjs.org/
  cnpm --- http://r.cnpmjs.org/
  taobao - https://registry.npm.taobao.org/
  nj ----- https://registry.nodejitsu.com/
  rednpm - http://registry.mirror.cqupt.edu.cn/
  npmMirror https://skimdb.npmjs.com/registry/
  edunpm - http://registry.enpmjs.org/

C:\Users\joedy>
```

通过 `nrm use taobao` 来指定要使用的镜像源：

```
C:\Users\joedy>nrm use taobao
verb config Skipping project config: C:\Users\joedy/.npmrc. (match fallback)
Registry has been set to: https://registry.npm.taobao.org/

C:\Users\joedy>
```

然后通过 `nrm test npm` 来测试速度：

```
C:\Users\joedy>nrm test npm  
npm ---- 823ms
```

注意：

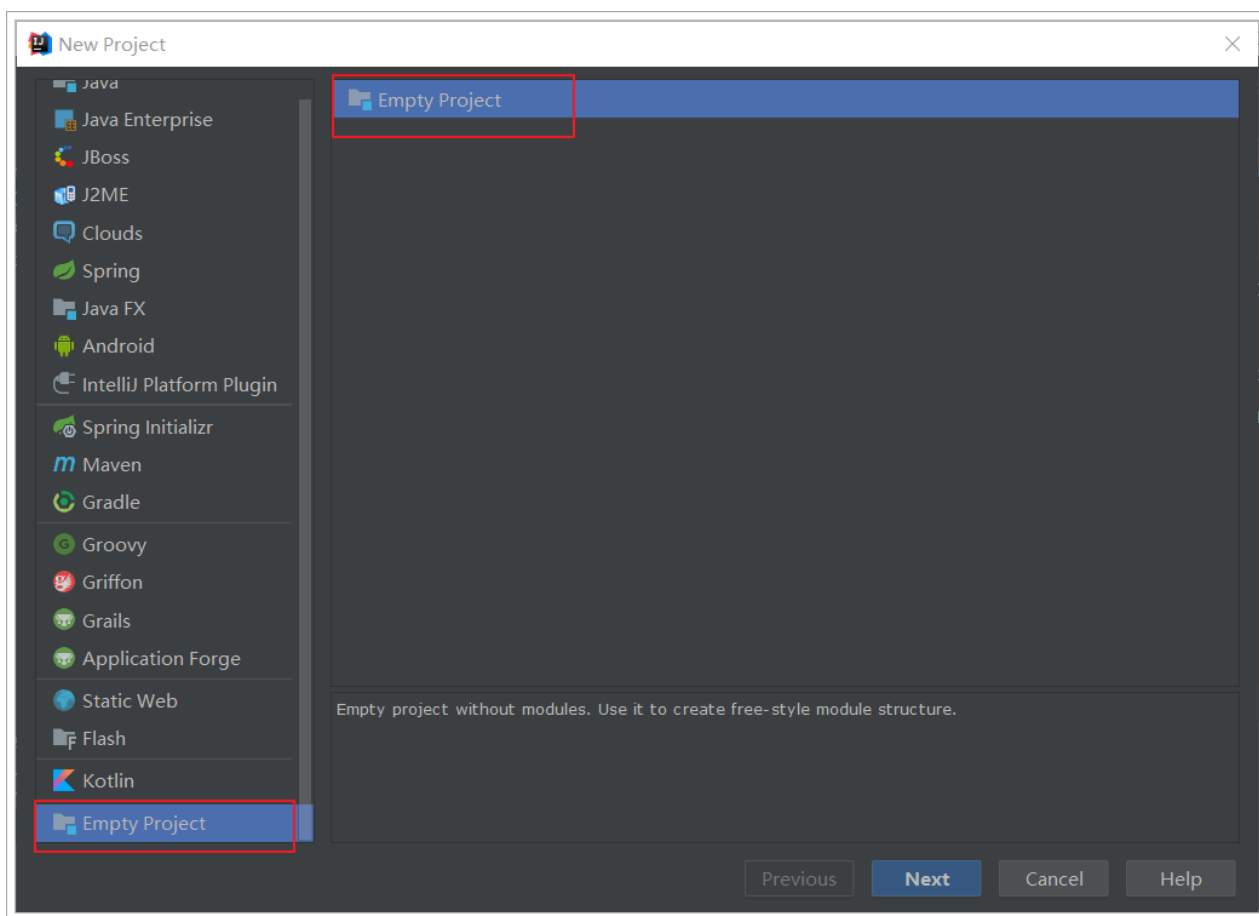
- 有教程推荐大家使用cnpm命令，但是使用发现cnpm有时会有bug，不推荐。
- 安装完成请一定要重启下电脑！！
- 安装完成请一定要重启下电脑！！
- 安装完成请一定要重启下电脑！！

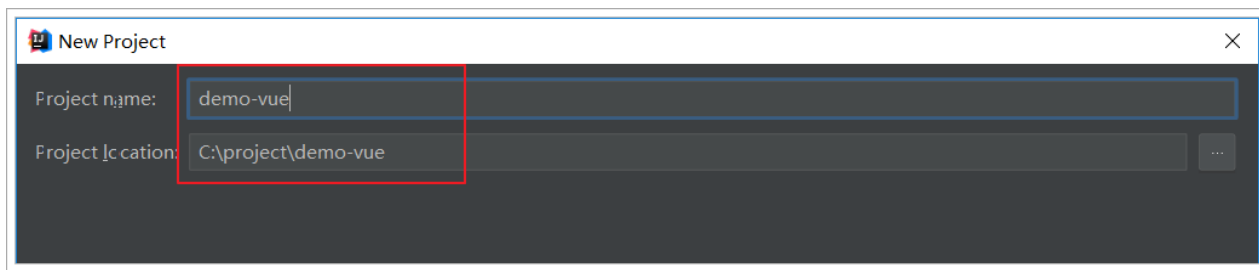
3.快速入门

接下来，我们快速领略下vue的魅力

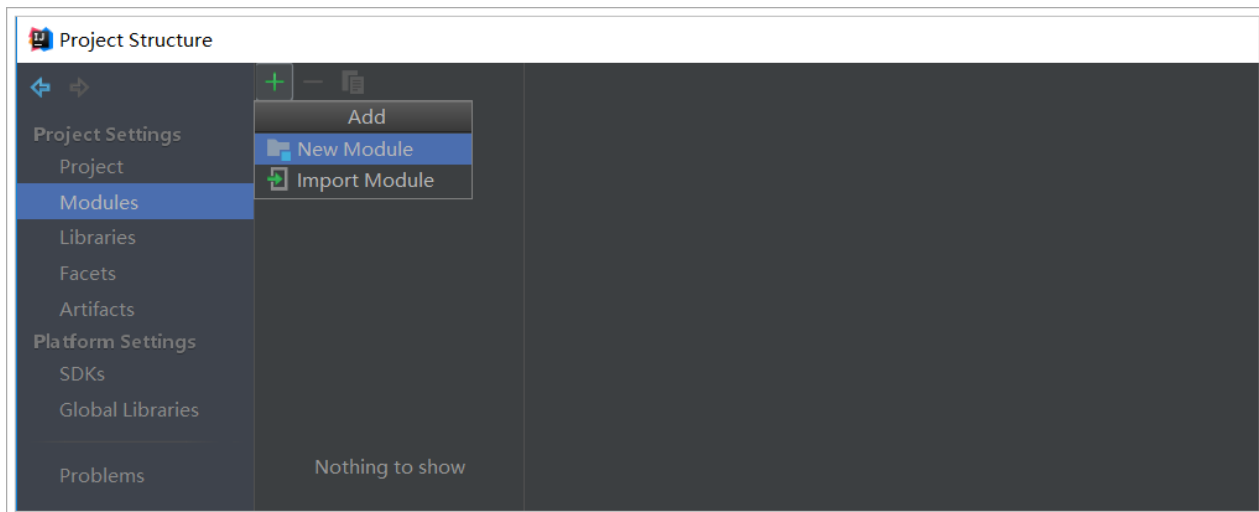
3.1.创建工程

创建一个新的空工程：

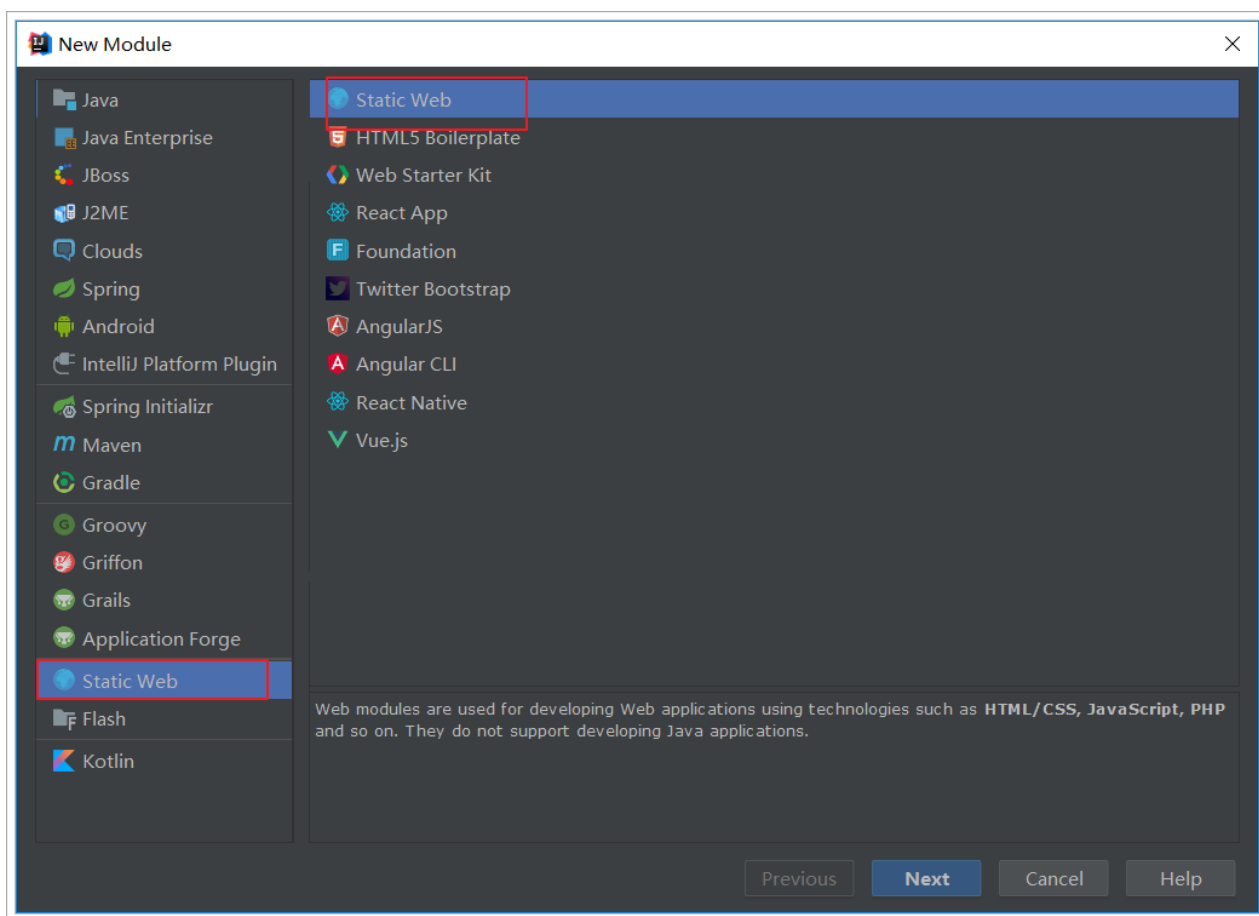




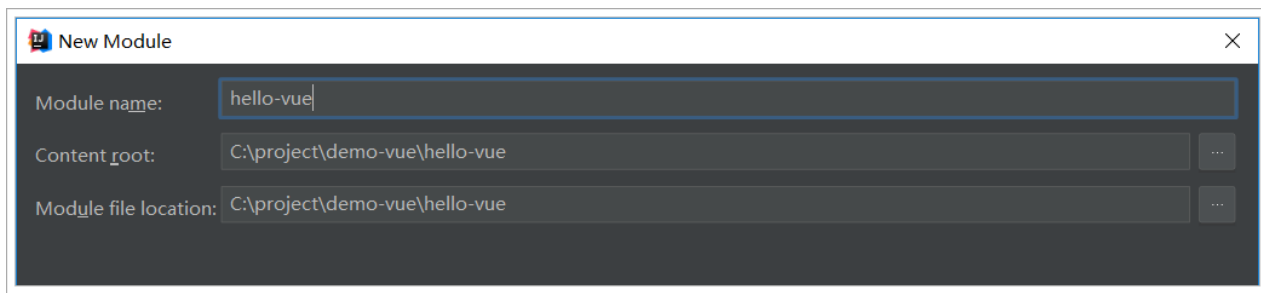
然后新建一个module:



选中static web, 静态web项目:



位置信息:



3.2.安装vue

3.2.1.下载安装

下载地址: <https://github.com/vuejs/vue>

可以下载2.5.16版本<https://github.com/vuejs/vue/archive/v2.5.16.zip>

下载解压, 得到vue.js文件。

3.2.2.使用CDN

本地可以不安装vue, 也可以使用vue
或者也可以直接使用公共的CDN服务:

```
<!-- 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>
```

或者:

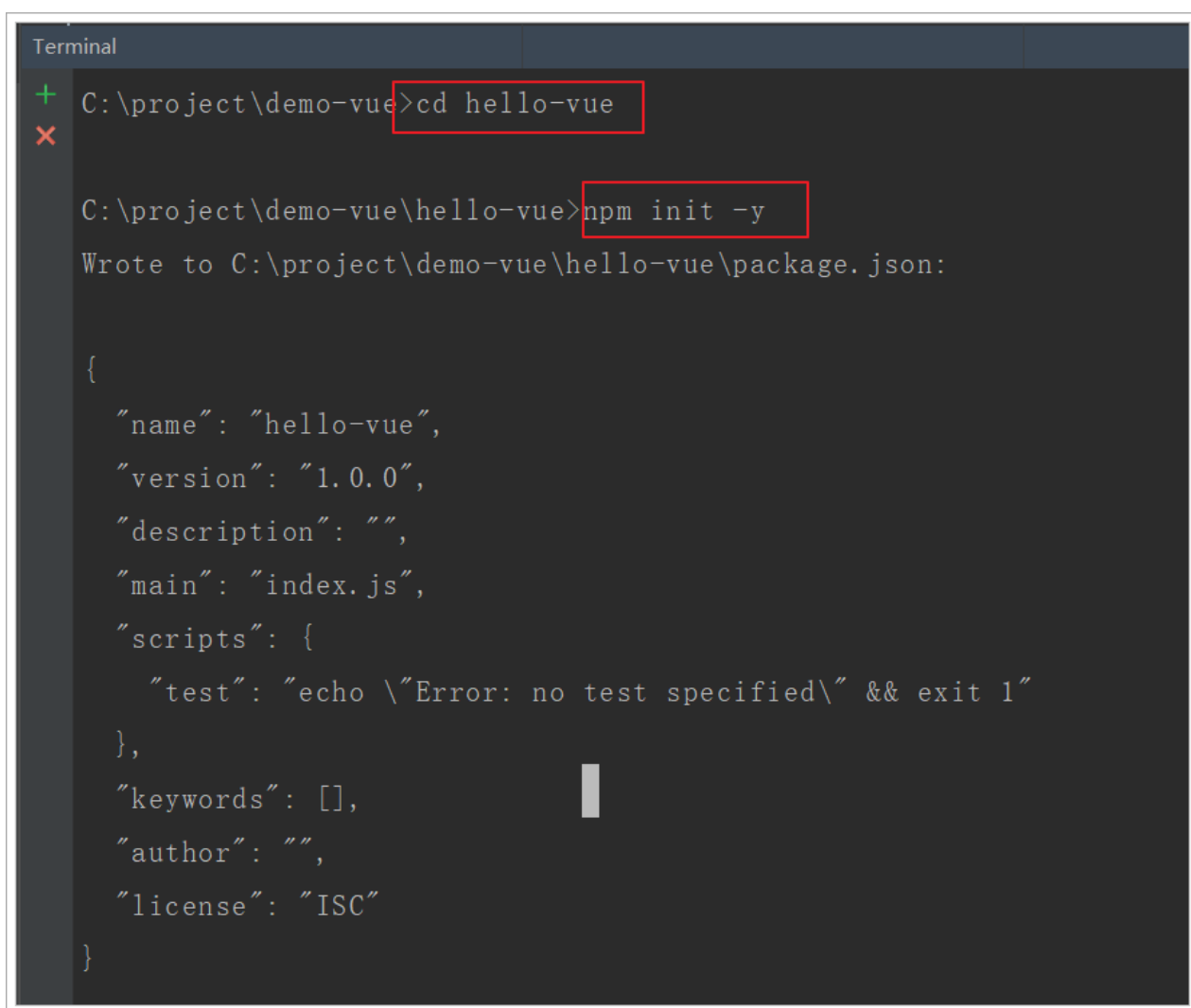
```
<!-- 生产环境版本, 优化了尺寸和速度 -->
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

3.2.3.推荐npm安装

在idea的左下角, 有个Terminal按钮, 点击打开控制台:



进入hello-vue目录，先输入：`npm init -y` 进行初始化



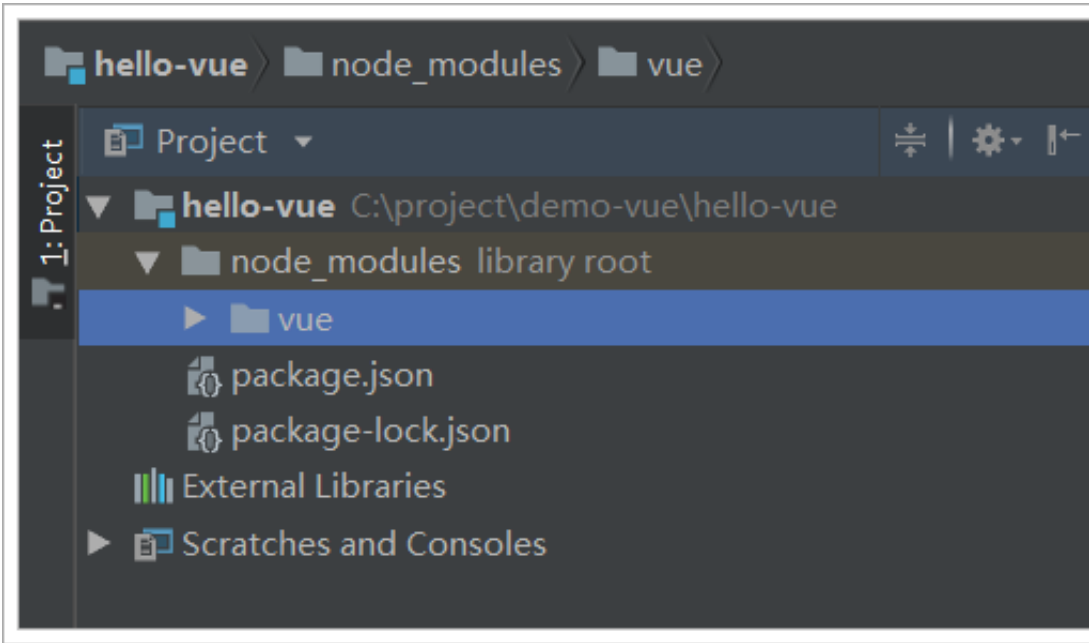
安装Vue，输入命令：`npm install vue --save`

安装到本地

```
C:\project\demo-vue\hello-vue>npm install vue --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN hello-vue@1.0.0 No description
npm WARN hello-vue@1.0.0 No repository field.

+ vue@2.5.16
added 1 package in 2.049s
```

然后就会在hello-vue目录发现一个node_modules目录，并且在下面有一个vue目录。

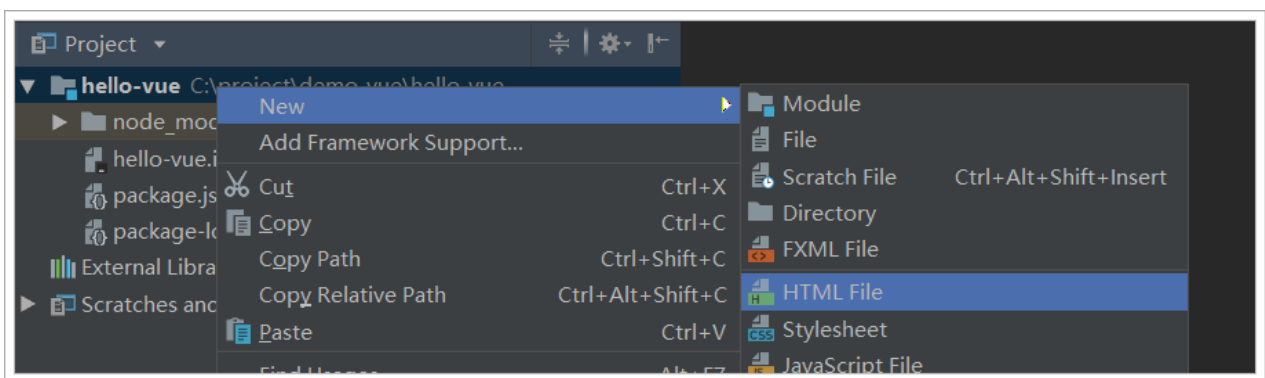


node_modules是通过npm安装的所有模块的默认位置。

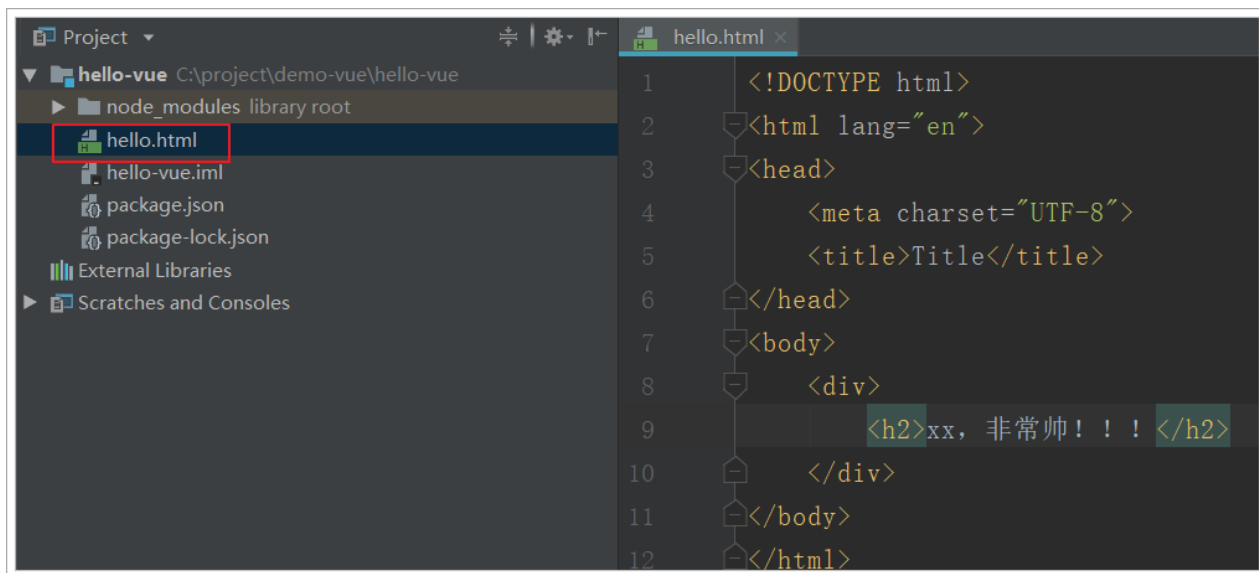
3.3.vue入门案例

3.3.1.HTML模板

在hello-vue目录新建一个HTML



在hello.html中，我们编写一段简单的代码：



h2中要输出一句话：xx 非常帅。前面的xx是要渲染的数据。

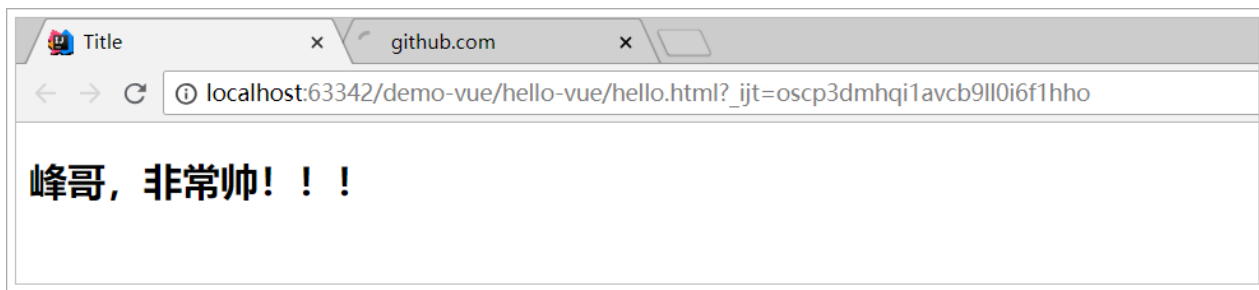
3.3.2.vue声明式渲染

然后通过Vue进行渲染：

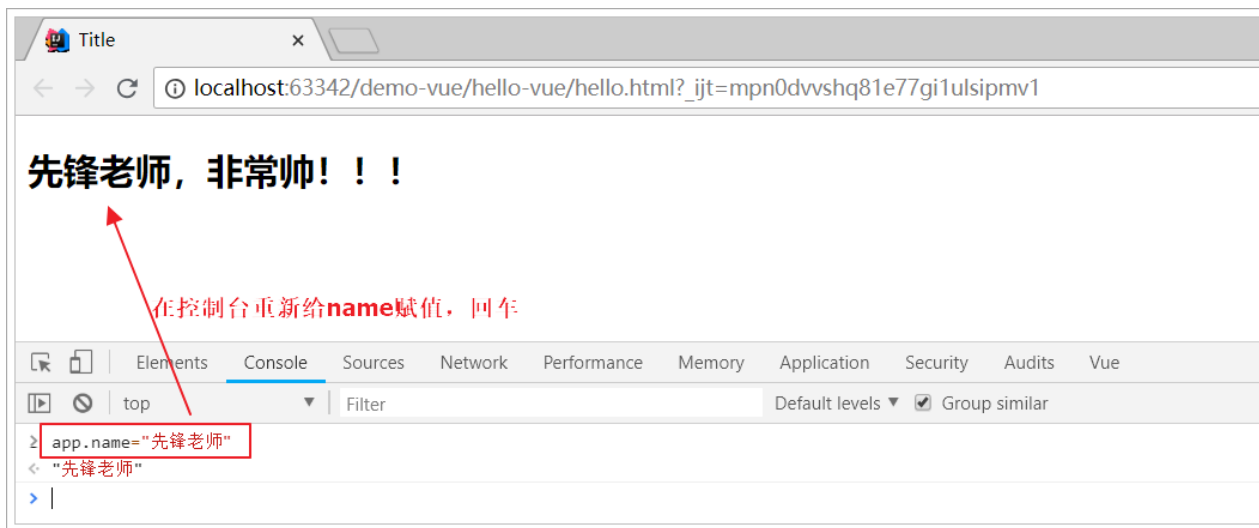
```
<body>
  <div id="app">
    <h2>{{name}}, 非常帅!!! </h2>
  </div>
</body>
<script src="node_modules/vue/dist/vue.js" ></script>
<script>
  // 创建vue实例
  var app = new Vue({
    el:"#app", // el即element, 该vue实例要渲染的页面元素
    data:{ // 渲染页面需要的数据
      name: "峰哥"
    }
  });
</script>
```

- 首先通过 `new Vue()` 来创建Vue实例
- 然后构造函数接收一个对象，对象中有一些属性：
 - `el`：是 `element` 的缩写，通过id选中要渲染的页面元素，本例中是一个div
 - `data`：数据，数据是一个对象，里面有很多属性，都可以渲染到视图中
 - `name`：这里我们指定了一个name属性
- 页面中的 `h2` 元素中，我们通过 `{{name}}` 的方式，来渲染刚刚定义的name属性。

打开页面查看效果：



更神奇的在于，当你修改name属性时，页面会跟着变化：



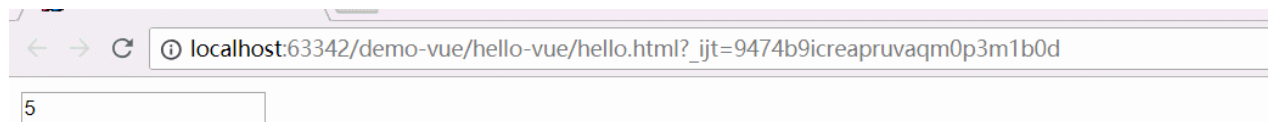
3.3.3.双向绑定

我们对刚才的案例进行简单修改：

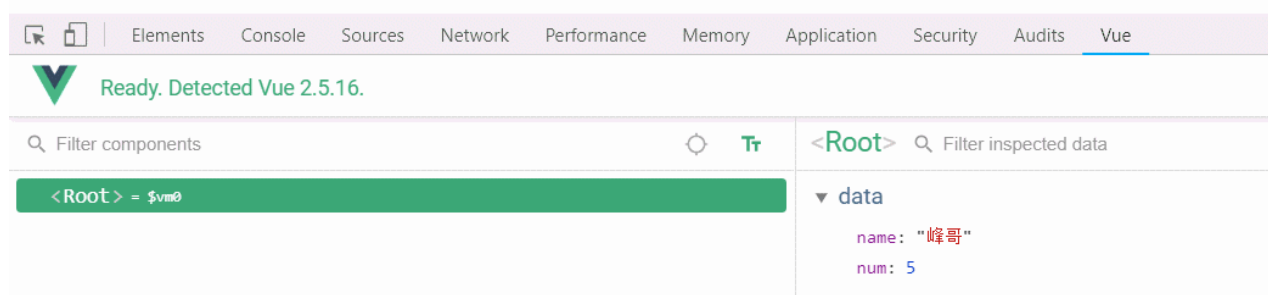
```
<body>
  <div id="app">
    <input type="text" v-model="num">
    <h2>
      {{name}}, 非常帅!!! 有{{num}}位女神为他着迷。
    </h2>
  </div>
</body>
<script src="node_modules/vue/dist/vue.js" ></script>
<script>
  // 创建vue实例
  var app = new Vue({
    el: "#app", // el即element, 该vue实例要渲染的页面元素
    data: { // 渲染页面需要的数据
      name: "峰哥",
      num: 5
    }
  });
</script>
```

- 我们在data添加了新的属性：`num`
- 在页面中有一个 `input` 元素，通过 `v-model` 与 `num` 进行绑定。
- 同时通过 `{{num}}` 在页面输出

效果：



峰哥，非常帅！！！有5位女神为他着迷。



我们可以观察到，输入框的变化引起了data中的num的变化，同时页面输出也跟着变化。

- input与num绑定，input的value值变化，影响到了data中的num值
- 页面 `{{num}}` 与数据num绑定，因此num值变化，引起了页面效果变化。

没有任何dom操作，这就是双向绑定的魅力。

3.3.4.事件处理

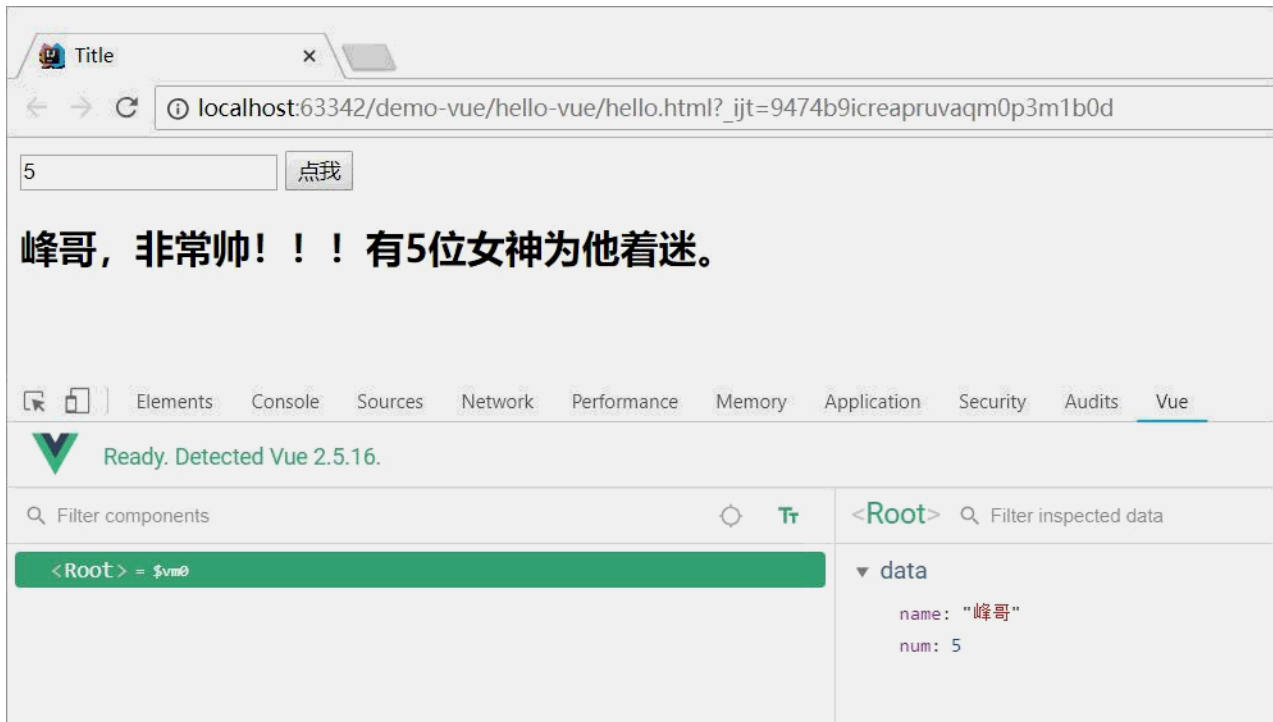
我们在页面添加一个按钮：

双引号里面可以写js表达式，也可以直接调用函数

```
<button v-on:click="num++">点我</button>
```

- 这里用 `v-on` 指令绑定点击事件，而不是普通的 `onclick`，然后直接操作num
- 普通click是无法直接操作num的。

效果：



4.Vue实例

4.1.创建Vue实例

每个 Vue 应用都是通过用 `Vue` 函数创建一个新的 **Vue 实例**开始的：

```
var vm = new Vue({  
  // 选项  
})
```

在构造函数中传入一个对象，并且在对象中声明各种Vue需要的数据和方法，包括：

- el
- data
- methods

等等

接下来我们一一介绍。

4.2.模板或元素

每个Vue实例都需要关联一段Html模板，Vue会基于此模板进行视图渲染。

我们可以通过el属性来指定。

例如一段html模板：

```
<div id="app">

</div>
```

然后创建Vue实例，关联这个div

```
var vm = new Vue({
  el: "#app"
})
```

这样，Vue就可以基于id为 `app` 的div元素作为模板进行渲染了。在这个div范围以外的部分是无法使用vue特性的。

4.3.数据

当Vue实例被创建时，它会尝试获取在data中定义的所有属性，用于视图的渲染，并且监视data中的属性变化，当data发生改变，所有相关的视图都将重新渲染，这就是“响应式”系统。

html:

```
<div id="app">
  <input type="text" v-model="name"/>
</div>
```

js:

```
var vm = new Vue({
  el: "#app",
  data: {
    name: "刘德华"
  }
})
```

- name的变化会影响到 input 的值
- input中输入的值，也会导致vm中的name发生改变

4.4.方法

Vue实例中除了可以定义data属性，也可以定义方法，并且在Vue实例的作用范围内使用。

html:


```
<div id="app">
  {{num}}
  <button v-on:click="add">加</button>
</div>
```

js:

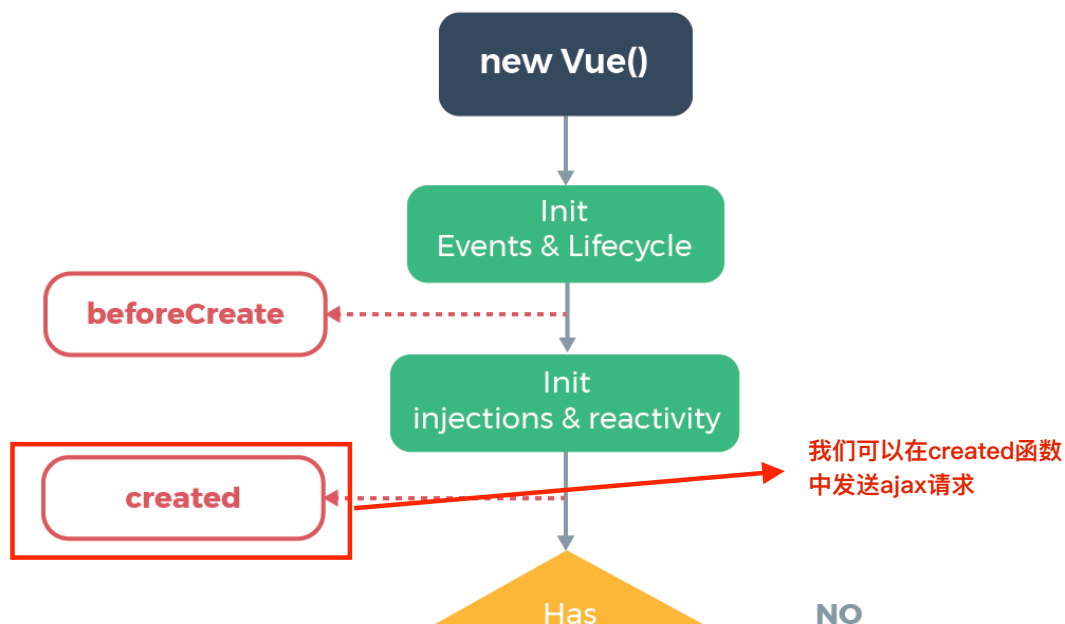
```
var vm = new Vue({
  el: "#app",
  data: {
    num: 0
  },
  methods: {
    add: function() {
      // this代表的当前vue实例
      this.num++;
    }
  }
})
```

4.5.生命周期钩子

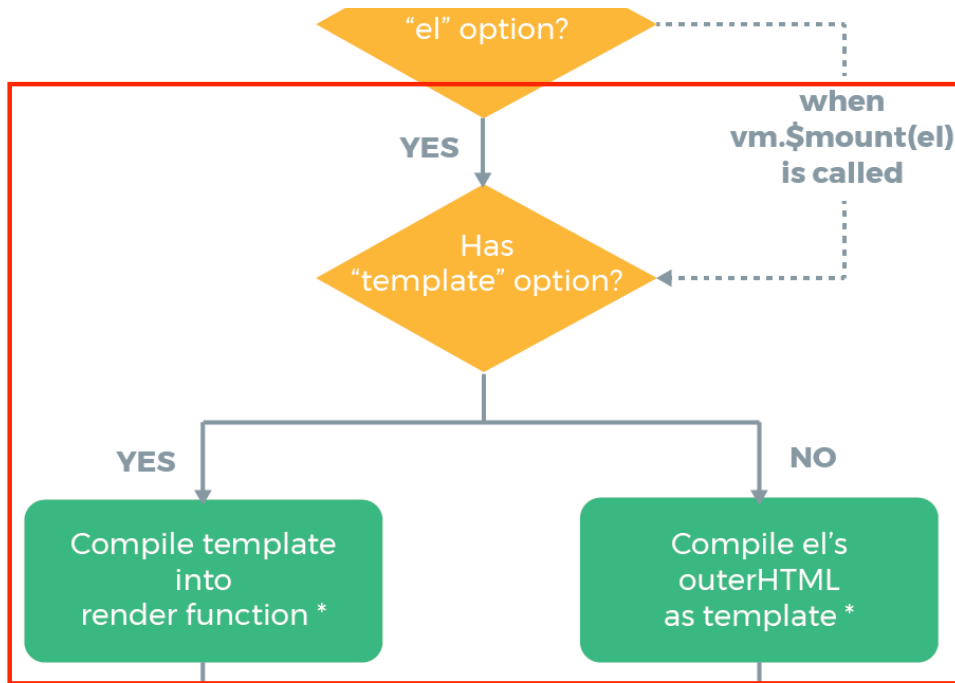
4.5.1.生命周期

每个 Vue 实例在被创建时都要经过一系列的初始化过程：创建实例，装载模板，渲染模板等等。Vue 为生命周期中的每个状态都设置了钩子函数（监听函数）。每当 Vue 实例处于不同的生命周期时，对应的函数就会被触发调用。

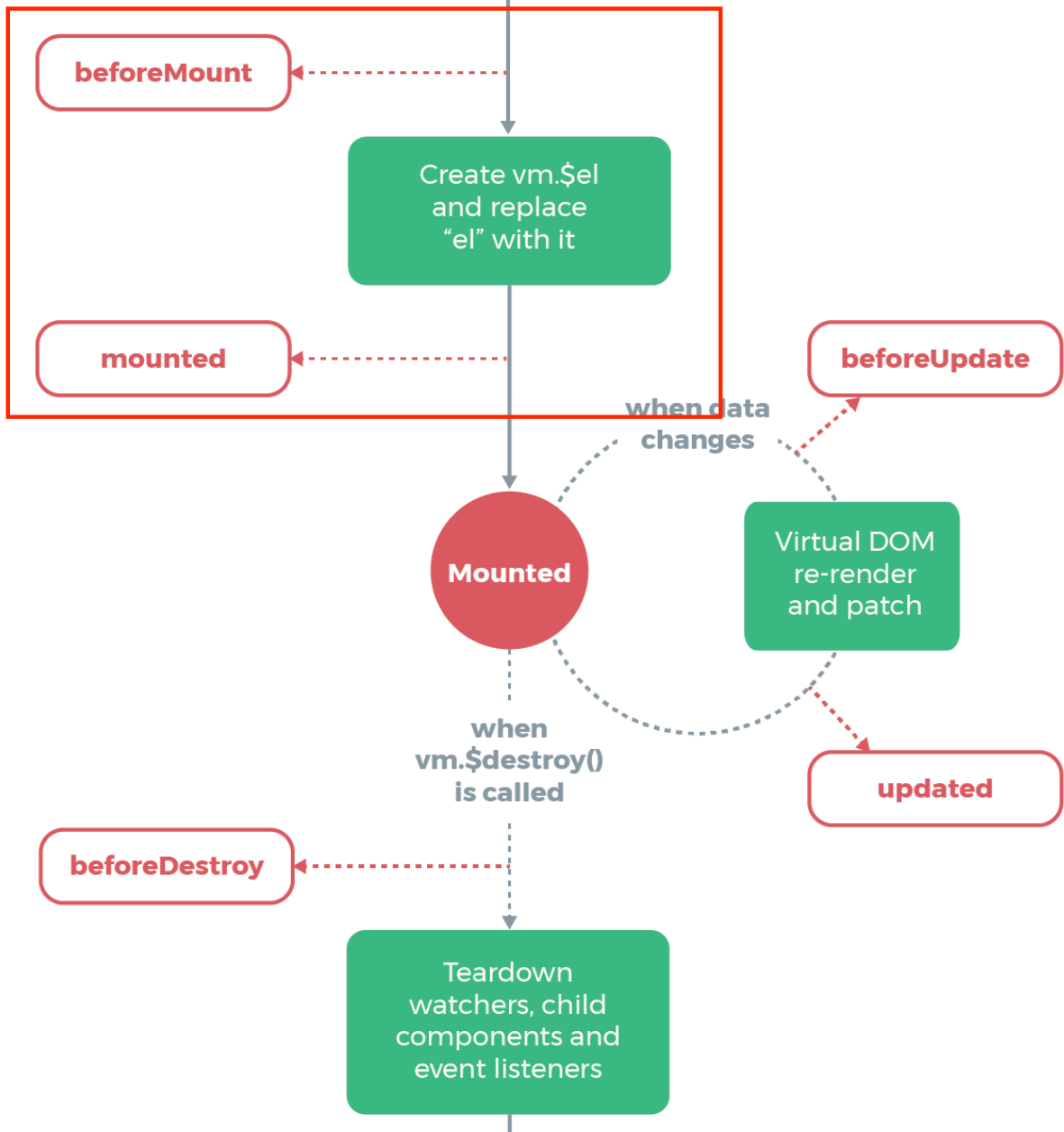
生命周期：



装载模板
将vue实例和html标签
关联起来



渲染模板
将data中的数据同步到html
中





* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

4.5.2.钩子函数

beforeCreated：我们在用Vue时都要进行实例化，因此，该函数就是在Vue实例化时调用，也可以将他理解为初始化函数比较方便一点，在Vue1.0时，这个函数的名字就是init。

created：在创建实例之后进行调用。

beforeMount：页面加载完成，没有渲染。如：此时页面还是{{name}}

mounted：我们可以将他理解为原生js中的window.onload=function({,}),或许大家也在用jquery，所以也可以理解为jquery中的\$(document).ready(function){...}，他的功能就是：在dom文档渲染完毕之后将要执行的函数，该函数在Vue1.0版本中名字为compiled。此时页面中的{{name}}已被渲染成峰哥

beforeDestroy：该函数将在销毁实例前进行调用。

destroyed：改函数将在销毁实例时进行调用。

beforeUpdate：组件更新之前。

updated：组件更新之后。

例如：created代表在vue实例创建后；

我们可以在Vue中定义一个created函数，代表这个时期的钩子函数：

```
// 创建vue实例
var app = new Vue({
  el: "#app", // el即element, 该vue实例要渲染的页面元素
  data: { // 渲染页面需要的数据
    name: "峰哥",
    num: 5
  },
  methods: {
    add: function(){
      this.num--;
    }
  }
})
```

```
    },  
    created: function () {  
      this.num = 100;  
    }  
  }  
});
```

结果：

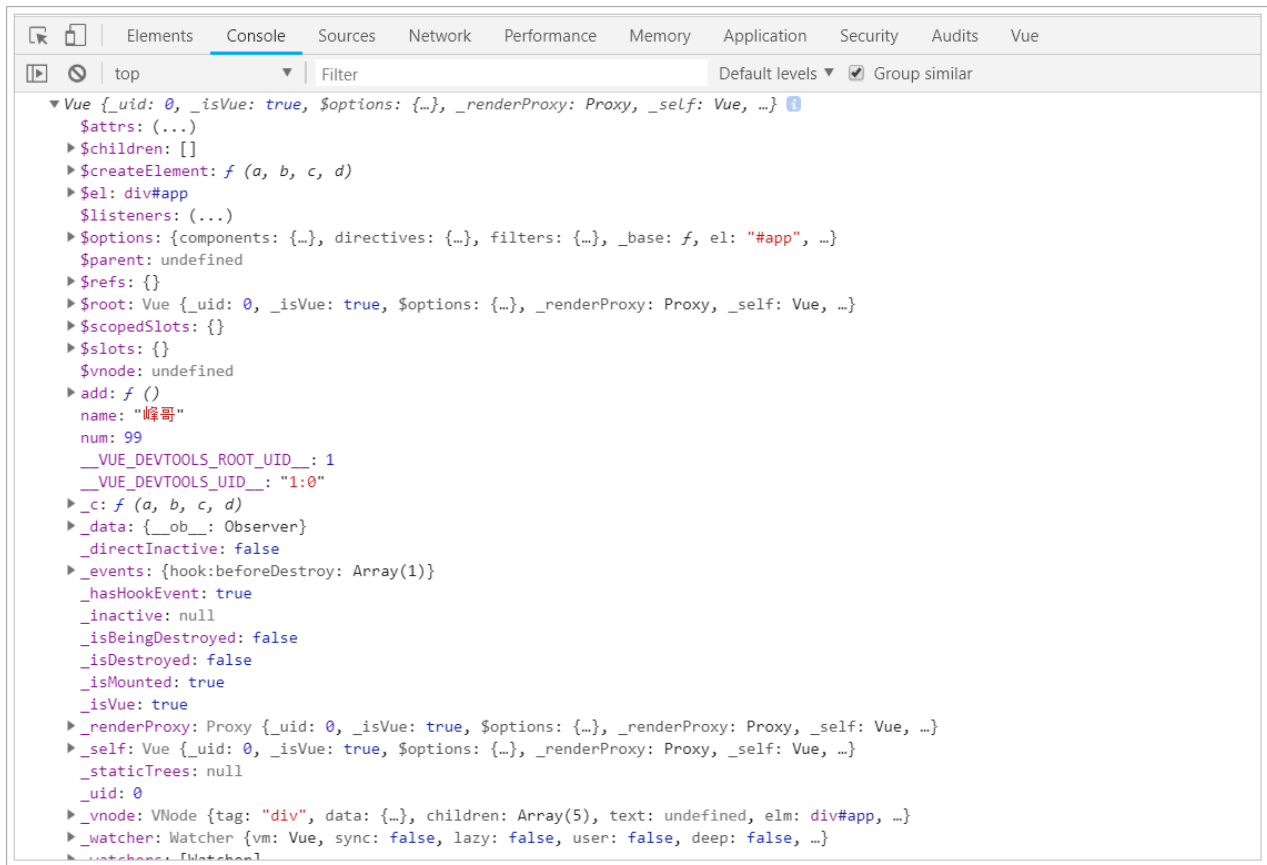


4.5.3.this

我们可以看下在vue内部的this变量是谁，我们在created的时候，打印this

```
methods: {  
  add: function(){  
    this.num--;  
    console.log(this);  
  }  
},
```

控制台的输出：



5.指令

什么是指令？

指令 (Directives) 是带有 `v-` 前缀的特殊特性。指令特性的预期值是：**单个 JavaScript 表达式**。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。

例如我们在入门案例中的 `v-on`，代表绑定事件。

5.1.插值表达式

5.1.1.花括号

格式：

```
{{表达式}}
```

说明：

- 该表达式支持JS语法，可以调用js内置函数（必须有返回值）
- 表达式必须有返回结果。例如 `1 + 1`，没有结果的表达式不允许使用，如：`var a = 1 + 1;`
- 可以直接获取Vue实例中定义的数据或函数

示例：

HTML:

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

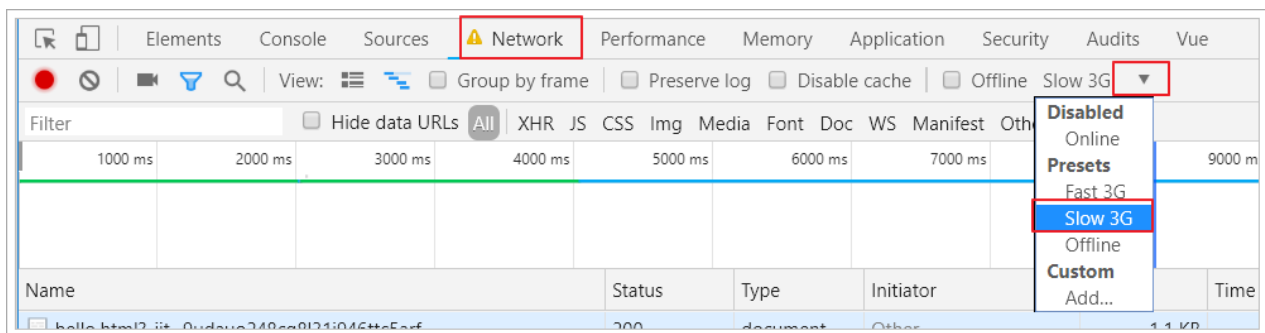
JS:

```
var app = new Vue({
  el: "#app",
  data: {
    name: "Jack"
  }
})
```

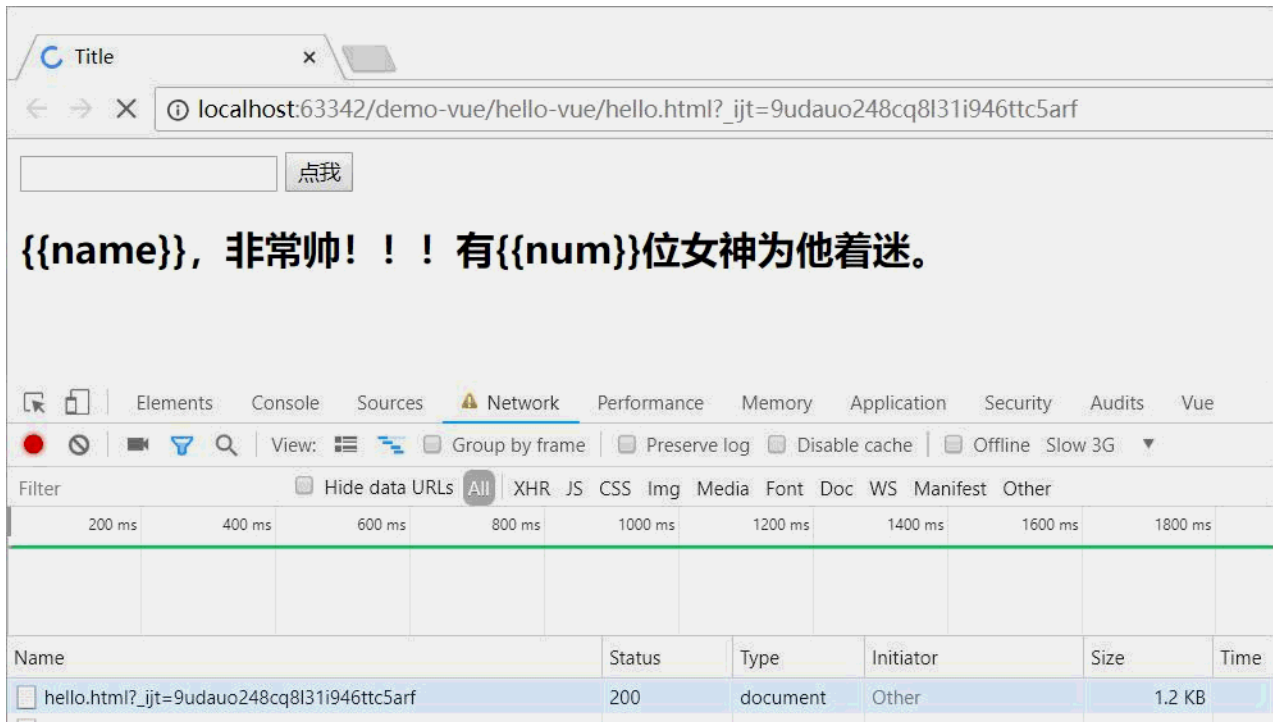
5.1.2. 插值闪烁

使用`{{}}`方式在网速较慢时会出现问题。在数据未加载完成时，页面会显示出原始的`{{}}`，加载完毕后才显示正确数据，我们称为插值闪烁。

我们将网速调慢一些，然后试试看刚才的案例：



刷新页面：



5.1.3.v-text和v-html

使用v-text和v-html指令来替代 {{ }}

说明：

一般情况下我们还是使用{{}}, 只有出现了插值闪烁才会使用v-text

- **v-text**：将数据输出到元素内部，如果输出的数据有HTML代码，会作为普通文本输出 通常使用这个
- **v-html**：将数据输出到元素内部，如果输出的数据有HTML代码，会被渲染 一般不用，会存在脚本侵入

示例：

HTML:

可以在标签里面填上初始值

```
<div id="app">
  v-text:<span v-text="hello"></span> <br/>
  v-html:<span v-html="hello"></span>
</div>
```

JS:

```
var vm = new Vue({
  el:"#app",
  data:{
    hello: "<h1>大家好，我是峰哥</h1>"
  }
})
```

效果：



并且不会出现插值闪烁，当没有数据时，会显示空白。

5.2.v-model

刚才的v-text和v-html可以看做是单向绑定，数据影响了视图渲染，但是反过来就不行。接下来学习的v-model是双向绑定，视图（View）和模型（Model）之间会互相影响。

既然是双向绑定，一定是在视图中可以修改数据，这样就限定了视图的元素类型。目前v-model的可用元素有：

- input
 - select
 - textarea
 - checkbox
 - radio
 - components (Vue中的自定义组件)
- 只有这几个标签可以使用双向model

基本上除了最后一项，其它都是表单的输入项。

举例：

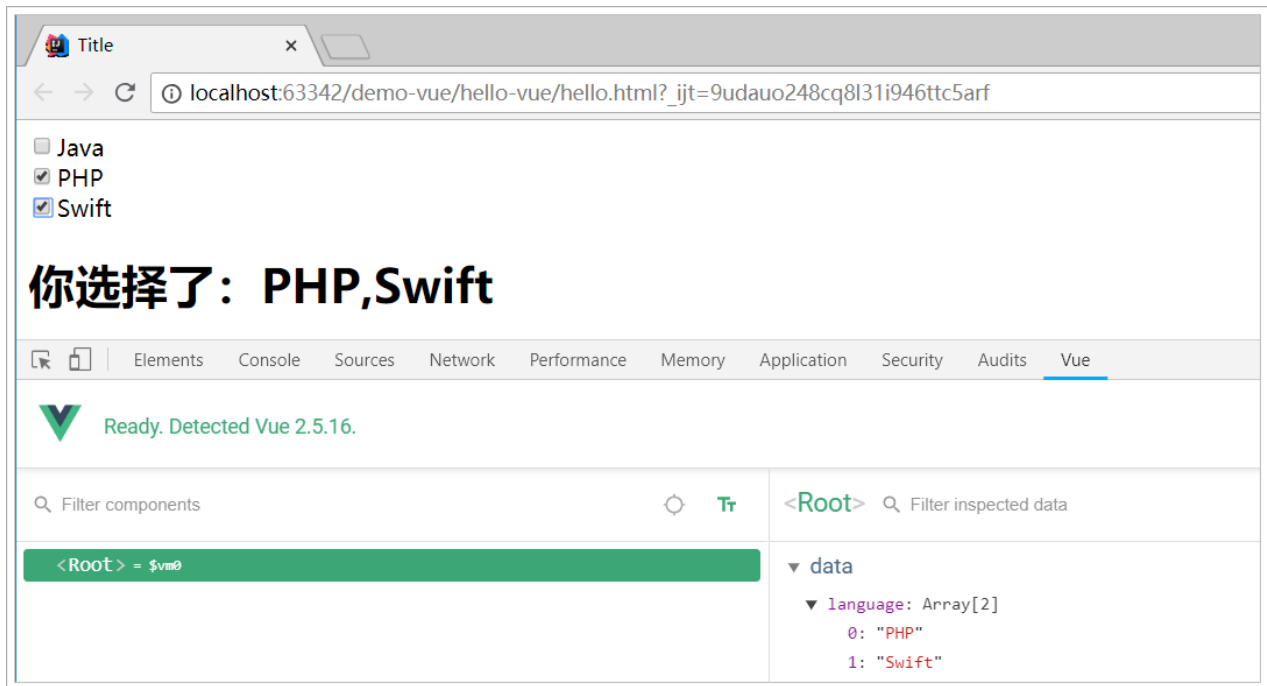
html：

```
<div id="app">
  <input type="checkbox" v-model="language" value="Java" />Java<br/>
  <input type="checkbox" v-model="language" value="PHP" />PHP<br/>
  <input type="checkbox" v-model="language" value="Swift" />Swift<br/>
  <h1>
    你选择了：{{language.join(',')}}
  </h1>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  var vm = new Vue({
    el:"#app",
    data:{
      language: []
    }
  })
</script>
```

输出结果用‘，’隔开

- 多个 `CheckBox` 对应一个model时，model的类型是一个数组，单个checkbox值默认是boolean类型
- radio对应的值是input的value值
- `text` 和 `textarea` 默认对应的model是字符串
- `select` 单选对应字符串，多选对应也是数组

效果：



5.3.v-on

5.3.1.基本用法

v-on指令用于给页面元素绑定事件。

语法：

v-on:事件名="js片段或函数名"

示例：

```
<div id="app">
  <!--事件中直接写js片段-->
  <button v-on:click="num++">增加一个</button><br/>
  <!--事件指定一个回调函数，必须是Vue实例中定义的函数-->
  <button v-on:click="decrement">减少一个</button><br/>
  <h1>有{{num}}个女神迷恋峰哥</h1>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
```

```
var app = new Vue({
  el: "#app",
  data: {
    num: 100
  },
  methods: {
    decrement() {
      this.num--;
    }
  }
})
</script>
```

效果：



另外，事件绑定可以简写，例如 `v-on:click='add'` 可以简写为 `@click='add'`

5.3.2. 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。尽管我们可以在方法中轻松实现这点，但更好的方式是：方法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**。修饰符是由点开头的指令后缀来表示的。

- `.stop`：阻止事件冒泡到父元素
- `.prevent`：阻止默认事件发生*
- `.capture`：使用事件捕获模式
- `.self`：只有元素自身触发事件才执行。（冒泡或捕获的都不执行）

- `.once`：只执行一次

阻止默认事件

```
<div id="app">
  <!-- 右击事件，并阻止默认事件发生 -->
  <button v-on:contextmenu.prevent="num++">增加一个</button>
  <br/>
  <!-- 右击事件，不阻止默认事件发生 -->
  <button v-on:contextmenu="decrement($event)">减少一个</button>
  <br/>
  <h1>有{{num}}个女神迷恋峰哥</h1>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  var app = new Vue({
    el: "#app",
    data: {
      num: 100
    },
    methods: {
      decrement(ev) {
        // ev.preventDefault();
        this.num--;
      }
    }
  })
</script>
```

效果：（右键“增加一个”，不会触发默认的浏览器右击事件；右键“减少一个”，会触发默认的浏览器右击事件）



5.3.3. 按键修饰符

在监听键盘事件时，我们经常需要检查常见的键值。Vue 允许为 `v-on` 在监听键盘事件时添加按键修饰符：

```
<!-- 只有在 `keyCode` 是 13 时调用 `vm.submit()` -->
<input v-on:keyup.13="submit">
```

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：

```
<!-- 同上 -->
<input v-on:keyup.enter="submit">

<!-- 缩写语法 -->
<input @keyup.enter="submit">
```

全部的按键别名：

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

- `.left`
- `.right`

5.3.4. 组合按钮 web项目很少使用组合按钮

可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器。

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

例如：

```
<!-- Alt + C -->
<input @keyup.alt.67="clear">
    需要用的组合后缀，只需要不停的在后面加后缀即可
<!-- Ctrl + Click -->
<div @click.ctrl="doSomething">Do something</div>
```

5.4. v-for

遍历数据渲染页面是非常常用的需求，Vue中通过v-for指令来实现。

5.4.1. 遍历数组

语法：

```
v-for="item in items"
```

- `items`：要遍历的数组，需要在vue的data中定义好。
- `item`：迭代得到的数组元素的别名

示例

```
<div id="app">
  <ul>
    <li v-for="user in users">
      {{user.name}} - {{user.gender}} - {{user.age}}
    </li>
  </ul>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  var app = new Vue({
    el: "#app",
```

```
data: {
  users:[
    {name:'柳岩', gender:'女', age: 21},
    {name:'峰哥', gender:'男', age: 18},
    {name:'范冰冰', gender:'女', age: 24},
    {name:'刘亦菲', gender:'女', age: 18},
    {name:'古力娜扎', gender:'女', age: 25}
  ]
},
})
</script>
```

效果：



5.4.2.数组角标

在遍历的过程中，如果我们需要知道数组角标，可以指定第二个参数：

语法

当遍历的字段超过一个时，需要用括号
v-for="(item,index) in items"

- items：要迭代的数组
- item：迭代得到的数组元素别名
- index：迭代到的当前元素索引，从0开始。

示例

```
<ul>
  <li v-for="(user, index) in users">
    {{index + 1}}. {{user.name}} - {{user.gender}} - {{user.age}}
  </li>
</ul>
```

效果：



5.4.3. 遍历对象

v-for除了可以迭代数组，也可以迭代对象。语法基本类似

语法：

```
v-for="value in object"
v-for="(value,key) in object"
v-for="(value,key,index) in object"
需要注意：value是放在前面的，后面是key
```

- 1个参数时，得到的是对象的属性值
- 2个参数时，第一个是属性值，第二个是属性名
- 3个参数时，第三个是索引，从0开始

示例：

```
<div id="app">
  <ul>
    <li v-for="(value, key, index) in user">
      {{index + 1}}. {{key}} - {{value}}
    </li>
  </ul>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  var vm = new Vue({
    el:"#app",
    data:{
      user:{name:'峰哥', gender:'男', age: 18}
    }
  })
</script>
```

效果：



5.4.4.key 使用v-for时就记得指定一个唯一的key来提高渲染效率

当 Vue.js 用 `v-for` 正在更新已渲染过的元素列表时，它默认用“就地复用”策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序，而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素。

这个功能可以有效的提高渲染的效率。

但是要实现这个功能，你需要给Vue一些提示，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素，你需要为每项提供一个唯一 `key` 属性。理想的 `key` 值是每项都有的且唯一的 `id`。

示例：

```
<ul>
  <li v-for="(item,index) in items" :key=index></li>
</ul>
```

- 这里使用了一个特殊语法：`:key=""` 我们后面会讲到，它可以让你读取vue中的属性，并赋值给 `key` 属性
- 这里我们绑定的key是数组的索引，应该是唯一的

5.5.v-if和v-show

5.5.1.基本使用

`v-if`，顾名思义，条件判断。当得到结果为true时，所在的元素才会被渲染。

语法：

```
v-if="布尔表达式"
```

示例：

```
<div id="app">
  <button v-on:click="show = !show">点我呀</button>
  <br>
  <h1 v-if="show">
    看到我啦？！
```

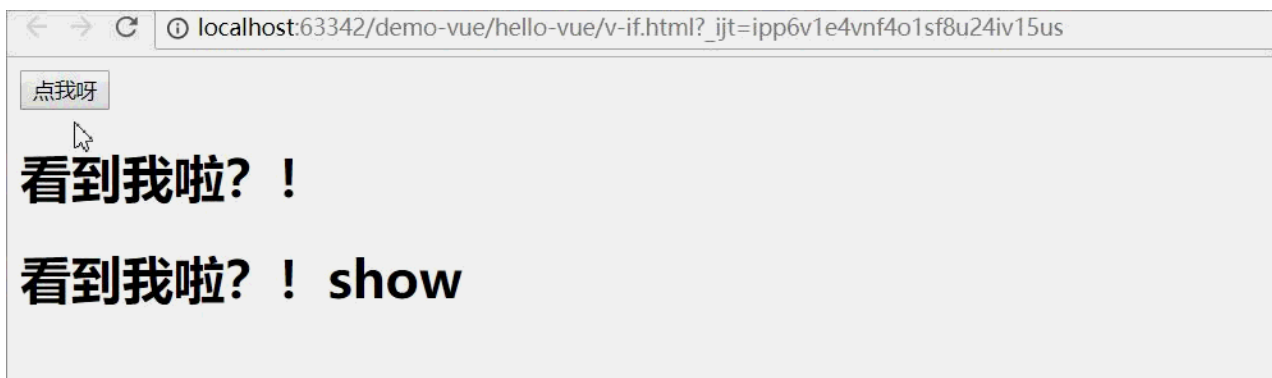


```

    </h1>
    <h1 v-show="show">
      看到我啦? ! show
    </h1>
  </div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  var app = new Vue({
    el: "#app",
    data: {
      show: true
    }
  })
</script>

```

效果：



5.5.2. 与v-for结合

当v-if和v-for出现在一起时，v-for优先级更高。也就是说，会先遍历，再判断条件。

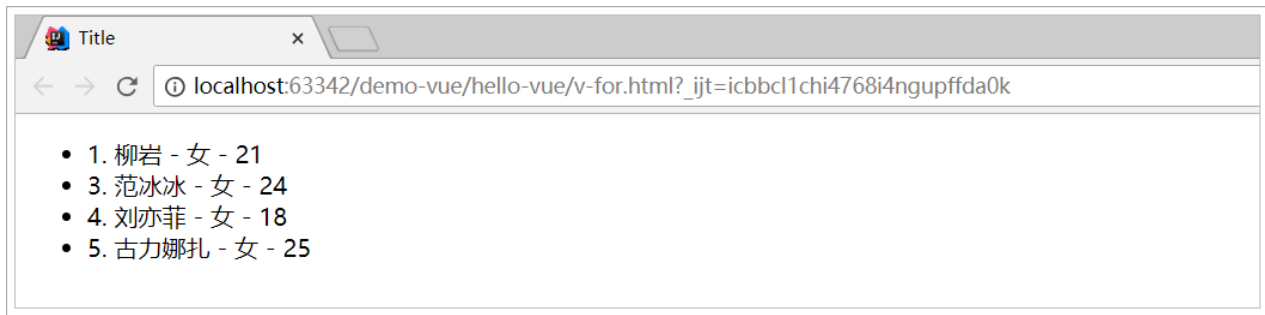
修改v-for中的案例，添加v-if：

```

<ul>
  <li v-for="(user, index) in users" v-if="user.gender == '女'">
    {{index + 1}}. {{user.name}} - {{user.gender}} - {{user.age}}
  </li>
</ul>

```

效果：



只显示女性用户信息

5.5.3.v-else

你可以使用 `v-else` 指令来表示 `v-if` 的“else 块”：

```
<div id="app">生成[0,1)的随机数
  <h1 v-if="Math.random() > 0.5">
    看到我啦? ! if
  </h1>
  <h1 v-else>
    看到我啦? ! else
  </h1>
</div>
```

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

意思就是，这两个标签里面不能有别的标签

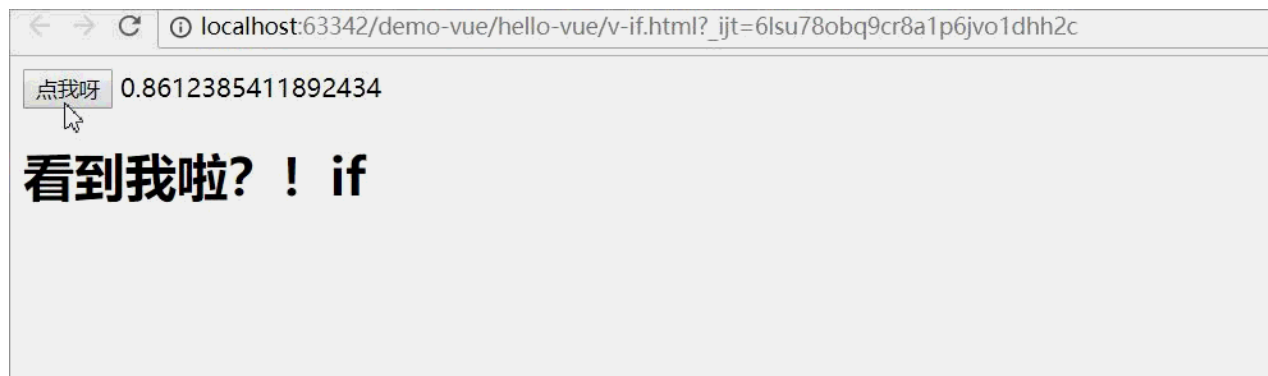
`v-else-if`，顾名思义，充当 `v-if` 的“else-if 块”，可以连续使用：

```
<div id="app">
  <button v-on:click="random=Math.random()">点我呀</button><span>{{random}}</span>
  <h1 v-if="random >= 0.75">
    看到我啦? ! if
  </h1>
  <h1 v-else-if="random > 0.5">
    看到我啦? ! if 0.5
  </h1>
  <h1 v-else-if="random > 0.25">
    看到我啦? ! if 0.25
  </h1>
  <h1 v-else>
    看到我啦? ! else
  </h1>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  var app = new Vue({
```

```
    el: "#app",
    data: {
      random: 1
    }
  })
</script>
```

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

演示：



5.5.4.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>
```

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` 只是简单地切换元素的 CSS 属性 `display`。

示例：

```
<div id="app">
  <!-- 事件中直接写js片段-->
  <button v-on:click="show = !show">点击切换</button><br/>
  <h1 v-if="show">
    你好
  </h1>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  var app = new Vue({
    el: "#app",
    data: {
      show: true
    }
  })
})
```

```
</script>
```

代码：



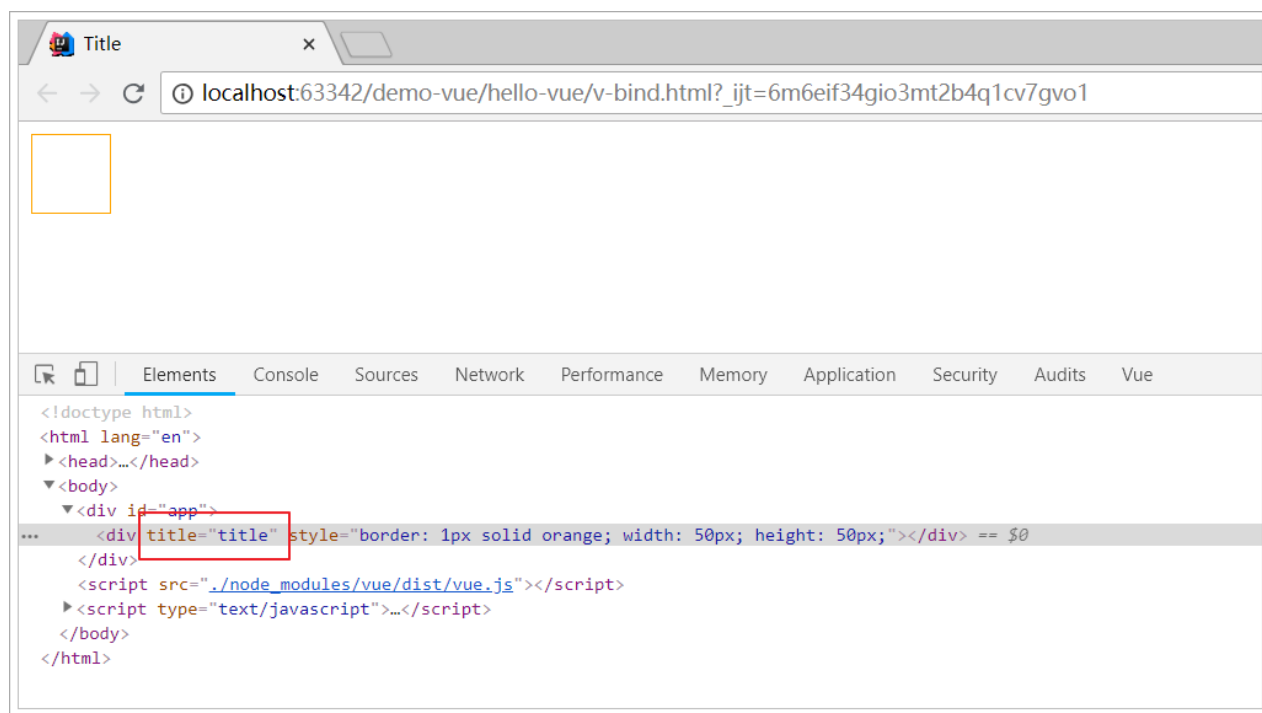
5.6.v-bind

html属性不能使用双大括号形式绑定，只能使用v-bind指令。

在将 `v-bind` 用于 `class` 和 `style` 时，Vue.js 做了专门的增强。表达式结果的类型除了字符串之外，还可以是对象或数组。

```
<div id="app">
  <!--可以是数据模型，可以是具有返回值的js代码块或者函数-->
  <div v-bind:title="title" style="border: 1px solid red; width: 50px; height:
50px;"></div>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  var app = new Vue({
    el: "#app",
    data: {
      title: "title",
    }
  })
</script>
```

效果：



在将 `v-bind` 用于 `class` 和 `style` 时，Vue.js 做了专门的增强。表达式结果的类型除了字符串之外，还可以是对象或数组。

5.6.1.绑定class样式

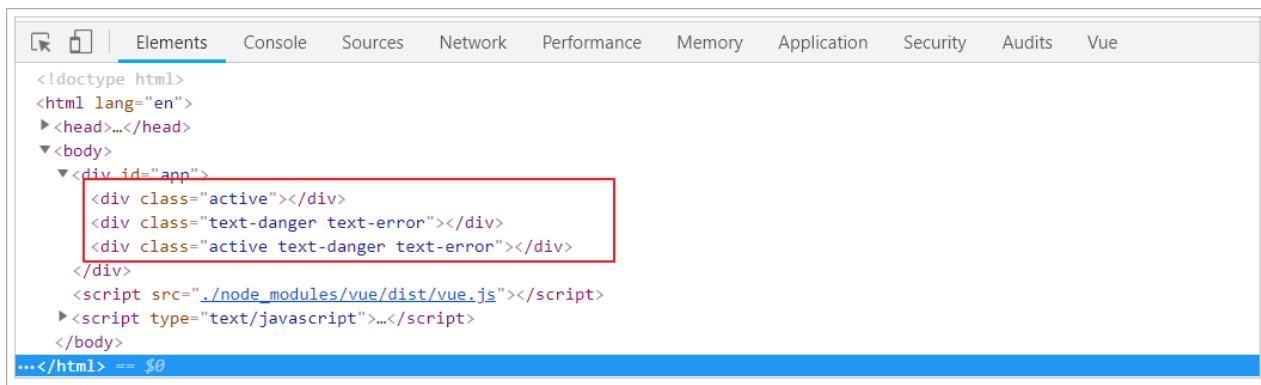
数组语法

我们可以借助于 `v-bind` 指令来实现：

HTML：

```
<div id="app">
  <div v-bind:class="activeClass"></div>
  <div v-bind:class="errorClass"></div>
  <div v-bind:class="[activeClass, errorClass]"></div>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  var app = new Vue({
    el: "#app",
    data: {
      activeClass: 'active',
      errorClass: ['text-danger', 'text-error']
    }
  })
</script>
```

渲染后的效果：（具有active和hasError的样式）



对象语法

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

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

如果isActive的值是true，那么active就会生效

上面的语法表示 `active` 这个 **class** 存在与否将取决于数据属性 `isActive` 的 [truthiness](#)（所有的值都是真实的，除了false,0,"",null,undefined和NaN）。

你可以在对象中传入更多属性来动态切换多个 class。此外，`v-bind:class` 指令也可以与普通的 class 属性共存。如下模板：

```
<div class="static" 静态属性
```

```
  v-bind:class="{ active: isActive, 'text-danger': hasError }"> 动态属性
```

```
</div>
```

动态属性最后会解析到静态属性里面

和如下 data：

```
data: {
  isActive: true,
  hasError: false
}
```

结果渲染为：

```
<div class="static active"></div>
```

active样式和text-danger样式的存在与否，取决于isActive和hasError的值。本例中isActive为true，hasError为false，所以active样式存在，text-danger不存在。

5.6.2. 绑定style样式

数组语法

数组语法可以将多个样式对象应用到同一个元素上：

```
<div v-bind:style="[baseStyles, overridingStyles]"></div>
```

数据：

```
data: {  
  baseStyles: {'background-color': 'red'},  
  overridingStyles: {border: '1px solid black'}  
}
```

渲染后的结果：

```
<div style="background-color: red; border: 1px solid black;"></div>
```

对象语法

`v-bind:style` 的对象语法十分直观——看着非常像 CSS，但其实是一个 JavaScript 对象。CSS 属性名可以用驼峰式 (camelCase) 或短横线分隔 (kebab-case，记得用单引号括起来) 来命名：

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

数据：

```
data: {  
  activeColor: 'red',  
  fontSize: 30  
}
```

效果：

```
<div style="color: red; font-size: 30px;"></div>
```

5.6.3. 简写

`v-bind:class` 可以简写为 `:class`

5.7. 计算属性

在插值表达式中使用js表达式是非常方便的，而且也经常用到。

但是如果表达式的内容很长，就会显得不够优雅，而且后期维护起来也不方便，例如下面的场景，我们有一个日期的数据，但是是毫秒值：

```
data:{
  birthday:1529032123201 // 毫秒值
}
```

我们在页面渲染，希望得到yyyy-MM-dd的样式：

```
<h1>您的生日是：{{
  new Date(birthday).getFullYear() + '-' + new Date(birthday).getMonth()+ '-' + new
  Date(birthday).getDay()
}}
</h1>
```

虽然能得到结果，但是非常麻烦。

Vue中提供了计算属性，来替代复杂的表达式：

```
var vm = new Vue({
  el:"#app",
  data:{
    birthday:1429032123201 // 毫秒值
  },
  computed:{
    birth(){// 计算属性本质是一个方法，但是必须返回结果
      const d = new Date(this.birthday);
      return d.getFullYear() + "-" + d.getMonth() + "-" + d.getDay();
    }
  }
})
```

- 计算属性本质就是方法，但是一定要返回数据。然后页面渲染时，可以把这个方法当成一个变量来使用。

页面使用：

```
<div id="app">
  <h1>您的生日是：{{birth}} </h1>
</div>
```

效果：



我们可以将同一函数定义为一个方法而不是一个计算属性。两种方式最终结果确实是完全相同的。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。这意味着只要 `birthday` 还没有发生改变，多次访问 `birthday` 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。 **可以提高渲染速度**

5.8.watch

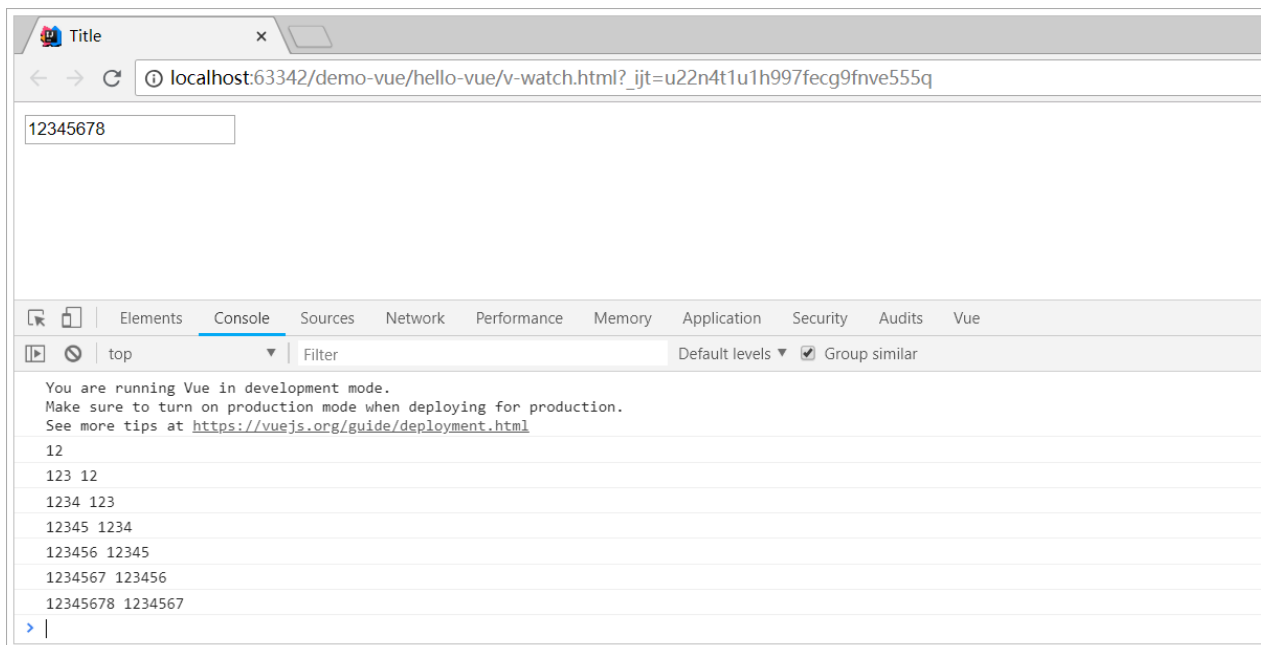
watch可以让我们监控一个值的变化。从而做出相应的反应。

示例： **可以实现搜索框的实时响应**

```
<div id="app">
  <input type="text" v-model="message">
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  var vm = new Vue({
    el:"#app",
    data:{
      message:""
    },
    watch:{
      message(新值, 旧值){
        console.log(newVal, oldVal);
      }
    }
  })
</script>
```

方法名和变量名保持一致 **message**(**newVal**, **oldVal**){
console.log(newVal, oldVal);

效果：



6. 组件化

在大型应用开发的时候，页面可以划分成很多部分。往往不同的页面，也会有相同的部分。例如可能会有相同的头部导航。

但是如果每个页面都独自开发，这无疑增加了我们开发的成本。所以我们会把页面的不同部分拆分成独立的组件，然后在不同页面就可以共享这些组件，避免重复开发。

在vue里，所有的vue实例都是组件

6.1. 全局组件

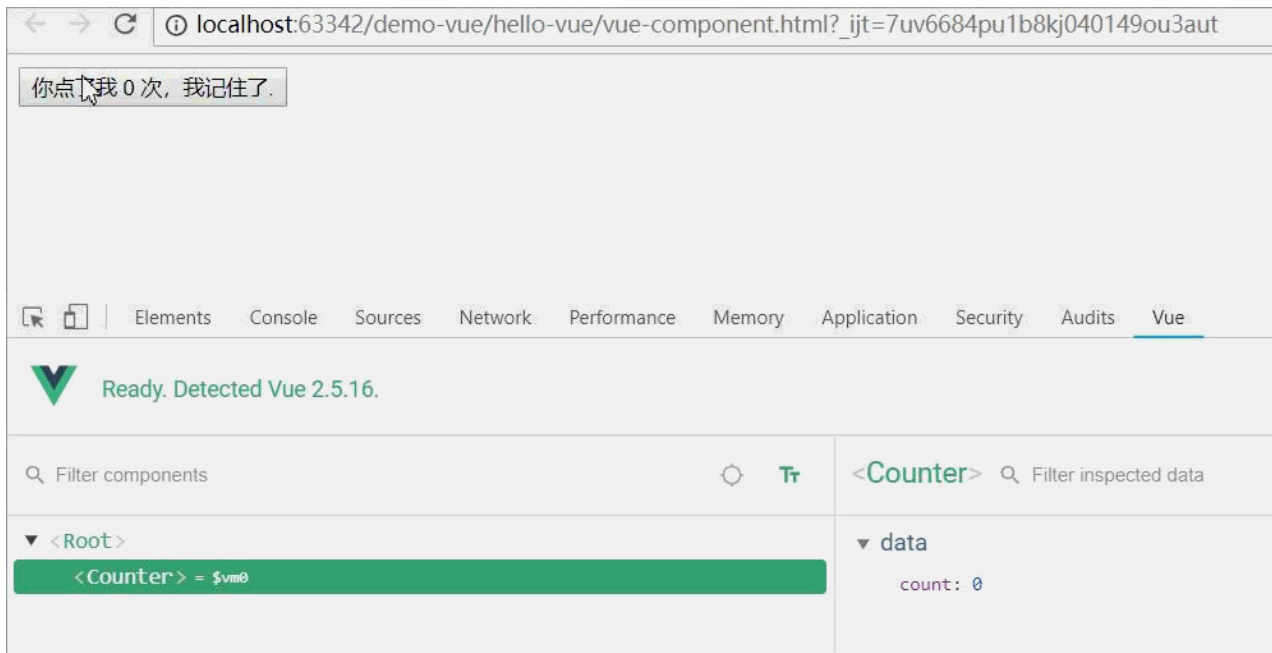
我们通过Vue的component方法来定义一个全局组件。

```
<div id="app">
  <!--使用定义好的全局组件-->
  <counter></counter>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  // 定义全局组件，两个参数：1，组件名称。2，组件参数
  Vue.component("counter",{
    template:'<button v-on:click="count++">你点了我 {{ count }} 次，我记住了。
    </button>',
    data(){
      return {
        count:0
      }
    }
  })
  var app = new Vue({
```

```
    el: "#app"  
  })  
</script>
```

- 组件其实也是一个Vue实例，因此它在定义时也会接收：data、methods、生命周期函数等
- 不同的是组件不会与页面的元素绑定，否则就无法复用了，因此没有el属性。
- 但是组件渲染需要html模板，所以增加了template属性，值就是HTML模板
- 全局组件定义完毕，任何vue实例都可以直接在HTML中通过组件名称来使用组件了。
- data必须是一个函数，不再是一个对象。

效果：



6.2.组件的复用

定义好的组件，可以任意复用多次：

```
<div id="app">  
  <!--使用定义好的全局组件-->  
  <counter></counter>  
  <counter></counter>  
  <counter></counter>  
</div>
```

效果：



你会发现每个组件互不干扰，都有自己的count值。怎么实现的？

组件的data属性必须是函数！

当我们定义这个 `<counter>` 组件时，它的data 并不是像之前直接提供一个对象：

```
data: {  
  count: 0  
}
```

取而代之的是，一个组件的 data 选项必须是一个函数，因此每个实例可以维护一份被返回对象的独立的拷贝：

```
data: function () {  
  return {  
    count: 0  
  }  
}
```

如果 Vue 没有这条规则，点击一个按钮就会影响到其它所有实例！

6.3.局部组件

一旦全局注册，就意味着即便以后你不再使用这个组件，它依然会随着Vue的加载而加载。

因此，对于一些并不频繁使用的组件，我们会采用局部注册。

我们先在外部定义一个对象，结构与创建组件时传递的第二个参数一致：

```
const counter = {  
  关键词 template: '<button v-on:click="count++">你点了我 {{ count }} 次, 我记住了.</button>',  
  data() {  
    return {  
      count: 0  
    }  
  }  
};
```

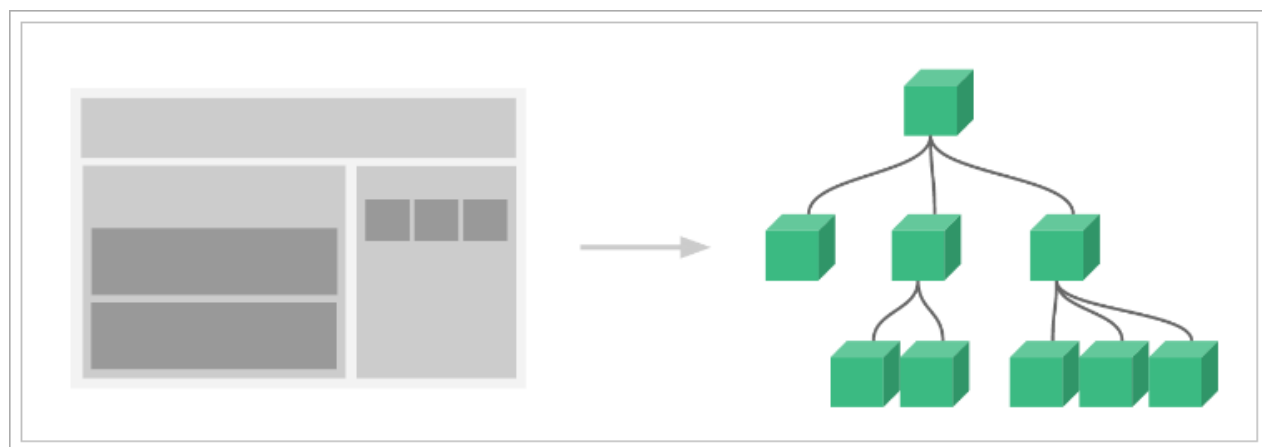
然后在Vue中使用它：

```
var app = new Vue({
  el: "#app",
  components: {
    counter: counter // 将定义的对象注册为组件
  }
})
```

- **components**就是当前vue对象子组件集合。就是标签名
 - 其key就是子组件名称
 - 其值就是组件对象名
- 效果与刚才的全局注册是类似的，不同的是，这个counter组件只能在当前的Vue实例中使用

6.4.组件通信

通常一个单页应用会以一棵嵌套的组件树的形式来组织：



- 页面首先分成了顶部导航、左侧内容区、右侧边栏三部分
- 左侧内容区又分为上下两个组件
- 右侧边栏中又包含了3个子组件

各个组件之间以嵌套的关系组合在一起，那么这个时候不可避免的会有组件间通信的需求。

6.4.1.props（父向子传递）

1. 父组件使用子组件时，自定义属性（属性名任意，属性值为要传递的数据）
2. 子组件通过props接收父组件数据，通过自定义属性的属性名

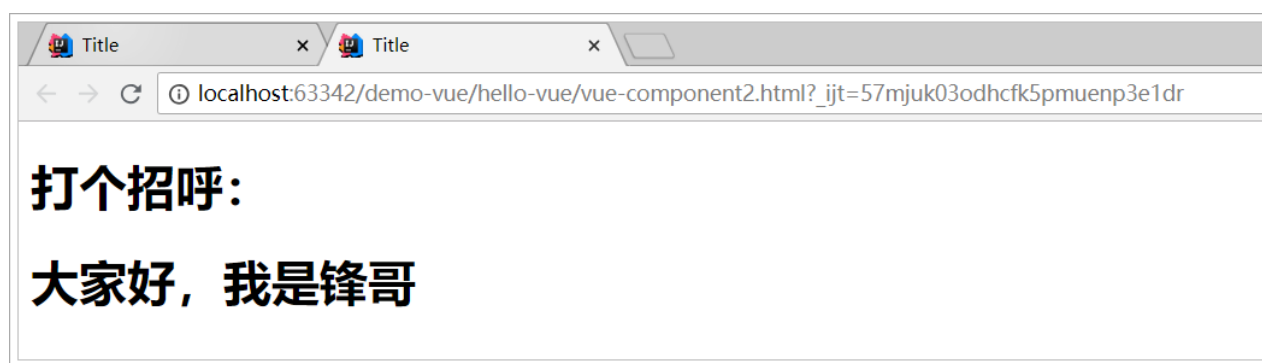
父组件使用子组件，并自定义了title属性：

```
<div id="app">
  <h1>打个招呼： </h1>
  <!--使用子组件，同时传递title属性-->
```

自定义属性

```
<introduce title="大家好，我是锋哥"/>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  Vue.component("introduce",{
    // 直接使用props接收到的属性来渲染页面
    template:'<h1>{{title}}</h1>',
    props:['title'] // 通过props来接收一个父组件传递的属性
  })
  var app = new Vue({
    el:"#app"
  })
</script>
```

效果：



6.4.2.props验证

我们定义一个子组件，并接收复杂数据：

```
const myList = {
  template: '\
<ul>\
  <li v-for="item in items" :key="item.id">{{item.id}} : {{item.name}}</li>\
</ul>\
',
  props: {
    items: {
      type: Array,
      default: [],
      required: true
    }
  }
};
```

- 这个子组件可以对 items 进行迭代，并输出到页面。
- props: 定义需要从父组件中接收的属性

- items: 是要接收的属性名称
 - type: 限定父组件传递来的必须是数组
 - default: 默认值
 - required: 是否必须

当 prop 验证失败的时候, (开发环境构建版本的) Vue 将会产生一个控制台的警告。

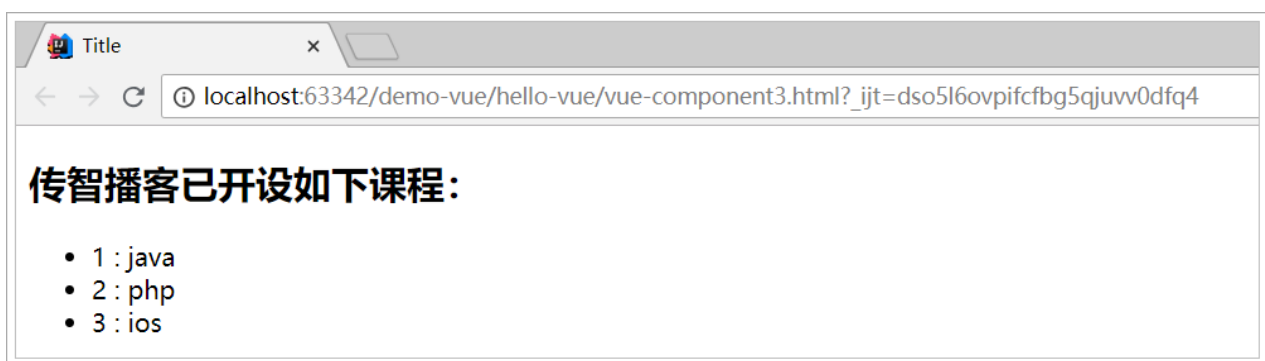
仅仅起到一个警告作用, 没有实际的约束力

我们在父组件中使用它:

```
<div id="app">
  <h2>传智播客已开设如下课程: </h2>
  <!-- 使用子组件的同时, 传递属性, 这里使用了v-bind, 指向了父组件自己的属性lessons -->
  <my-list :items="lessons"/>
</div>
```

```
var app = new Vue({
  el: "#app",
  components: {
    myList // 当key和value一样时, 可以只写一个
  },
  data: {
    lessons: [
      {id: 1, name: 'java'},
      {id: 2, name: 'php'},
      {id: 3, name: 'ios'},
    ]
  }
})
```

效果:



type类型, 可以有:

`type` 可以是下列原生构造函数中的一个：

- String
- Number
- Boolean
- Array
- Object
- Date
- Function
- Symbol

注意：子组件模板有且只有一个根标签

6.4.3.动态静态传递

给 prop 传入一个静态的值：

```
<introduce title="大家好，我是锋哥"/>
```

给 prop 传入一个动态的值：（通过v-bind从数据模型中，获取title的值）

```
<introduce :title="title"/>
```

静态传递时，我们传入的值都是字符串类型的，但实际上任何类型的值都可以传给一个 props。

```
<!-- 即便 `42` 是静态的，我们仍然需要 `v-bind` 来告诉 Vue -->
<!-- 这是一个JavaScript表达式而不是一个字符串。-->
<blog-post v-bind:likes="42"></blog-post>

<!-- 用一个变量进行动态赋值。-->
<blog-post v-bind:likes="post.likes"></blog-post>
```

6.4.4.子向父的通信：\$emit

来看这样的一个案例： 子组件只能操作父组件传递过来的值，无法操作父组件的值
我们的做法是：用父组件给子组件传递一个方法，而这个方法可以操作父组件里面的值

```
<div id="app">
  <h2>num: {{num}}</h2>
  <!--使用子组件的时候，传递num到子组件中-->
  <counter :num="num"></counter>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
```



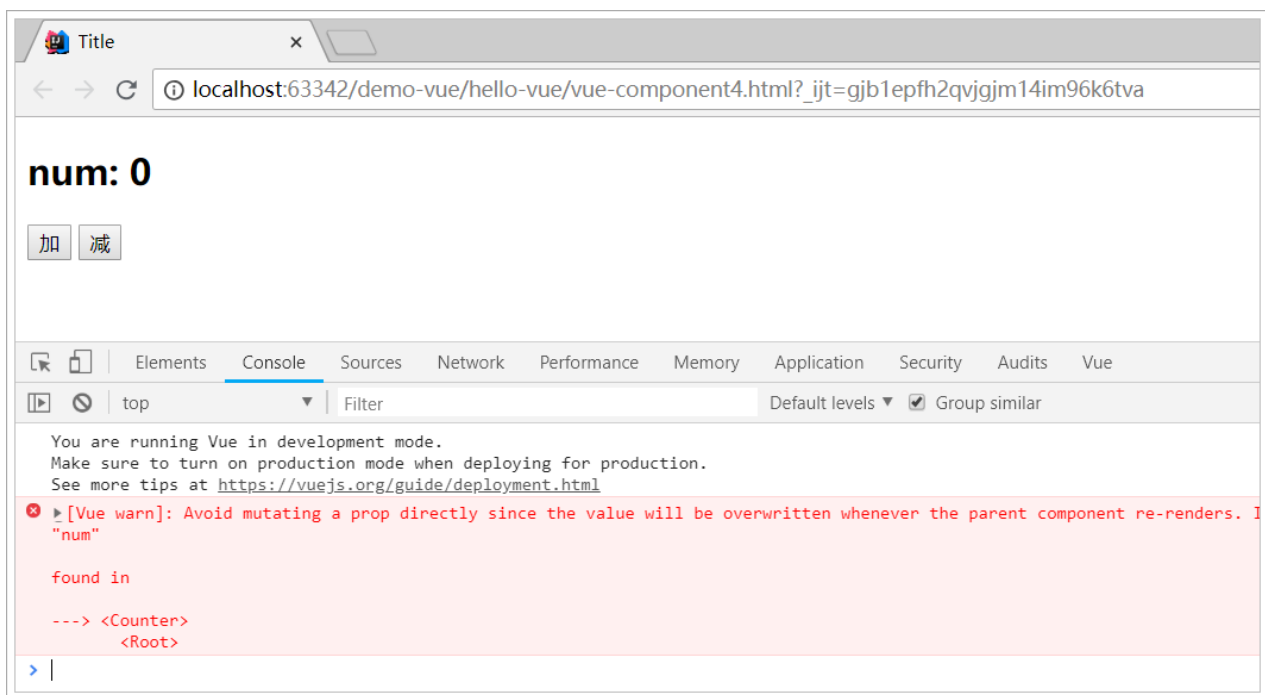
```

Vue.component("counter", { // 子组件，定义了两个按钮，点击数字num会加或减
  template: `
    <div>
      <button @click="num++">加</button> \
      <button @click="num--">减</button> \
    </div>`,
  props: ['num'] // count是从父组件获取的。
})
var app = new Vue({
  el: "#app",
  data: {
    num: 0
  }
})
</script>

```

- 子组件接收父组件的num属性
- 子组件定义点击按钮，点击后对num进行加或减操作

我们尝试运行，好像没问题，点击按钮试试：



子组件接收到父组件属性后，默认是不允许修改的。怎么办？

既然只有父组件能修改，那么加和减的操作一定是放在父组件：

```

var app = new Vue({
  el: "#app",
  data: {
    num: 0
  },
  methods: { // 父组件中定义操作num的方法
    increment(){

```

```

        this.num++;
    },
    decrement(){
        this.num--;
    }
}
})

```

但是，点击按钮是在子组件中，那就是说需要子组件来调用父组件的函数，怎么做？

我们可以通过v-on指令将父组件的函数绑定到子组件上：

```

<div id="app">
  <h2>num: {{num}}</h2>
  <counter :count="num" @inc="increment" @dec="decrement"></counter>
</div>

```

自定义事件，传递父组件的函数

在子组件中定义函数，函数的具体实现调用父组件的实现，并在子组件中调用这些函数。当子组件中按钮被点击时，调用绑定的函数：

```

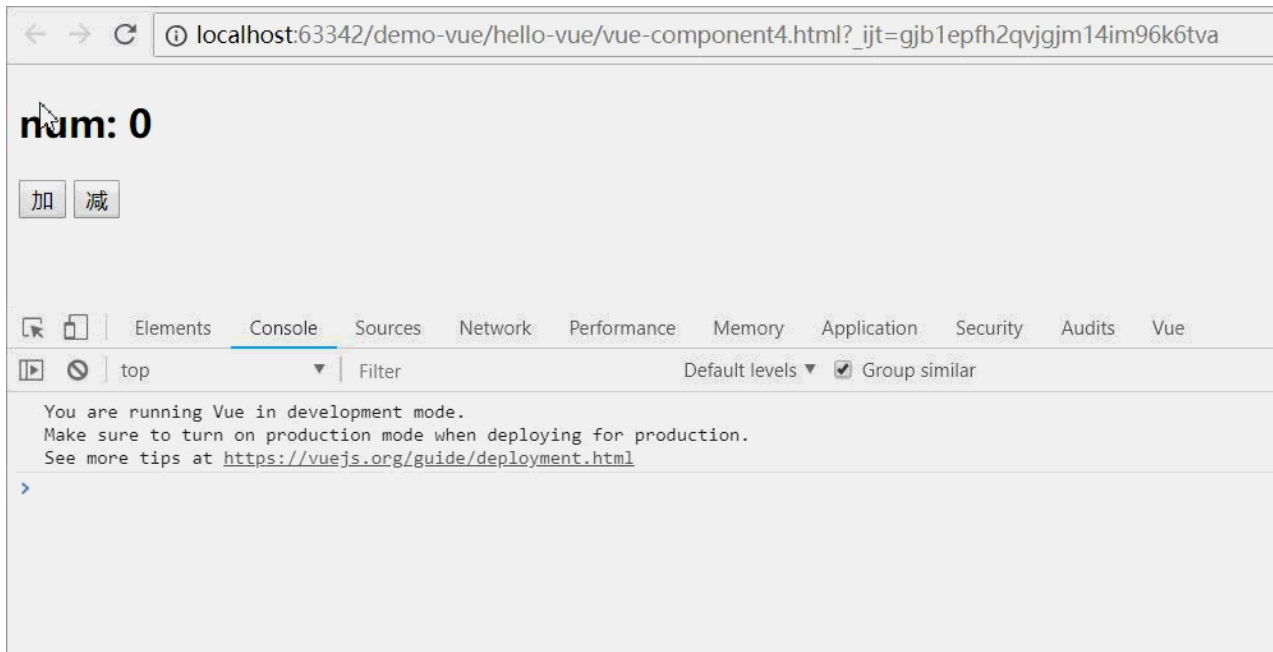
Vue.component("counter", {
  template: `
    <div>
      <button @click="plus">加</button> \
      <button @click="reduce">减</button> \
    </div>`,
  props: ['count'],
  methods: {
    plus(){
      this.$emit("inc");
    },
    reduce(){
      this.$emit("dec");
    }
  }
})

```

固定写法，参数是：父组件方法名

- vue提供了一个内置的this.\$emit()函数，用来调用父组件绑定的函数

效果：



7.路由vue-router

7.1.场景模拟

现在我们来实现这样一个功能：

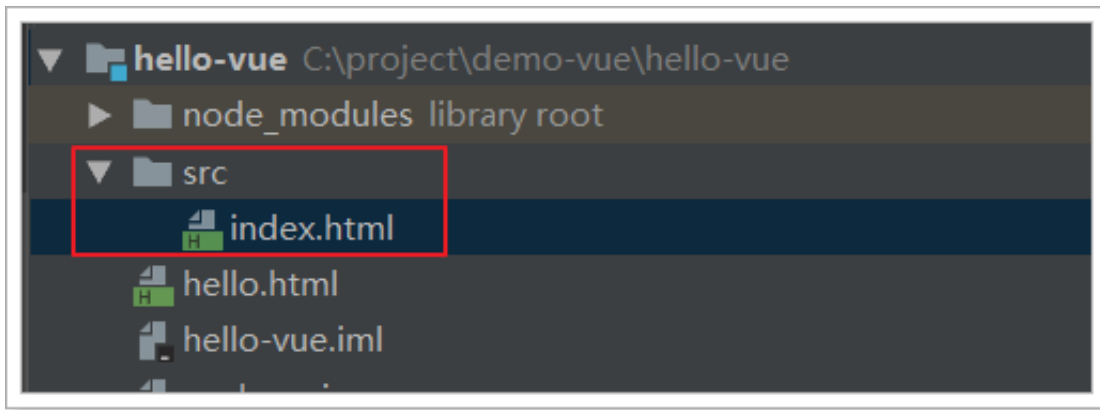
一个页面，包含登录和注册，点击不同按钮，实现登录和注册页切换：



7.1.1.编写父组件

为了让接下来的功能比较清晰，我们先新建一个文件夹：src

然后新建一个HTML文件，作为入口：index.html



然后编写页面的基本结构：

```
<div id="app">
  <span>登录</span>
  <span>注册</span>
  <hr/>
  <div>
    登录页/注册页
  </div>
</div>
```

引入vue的代码不要写在head标签里面，否则不起作用，
尽量下载下面

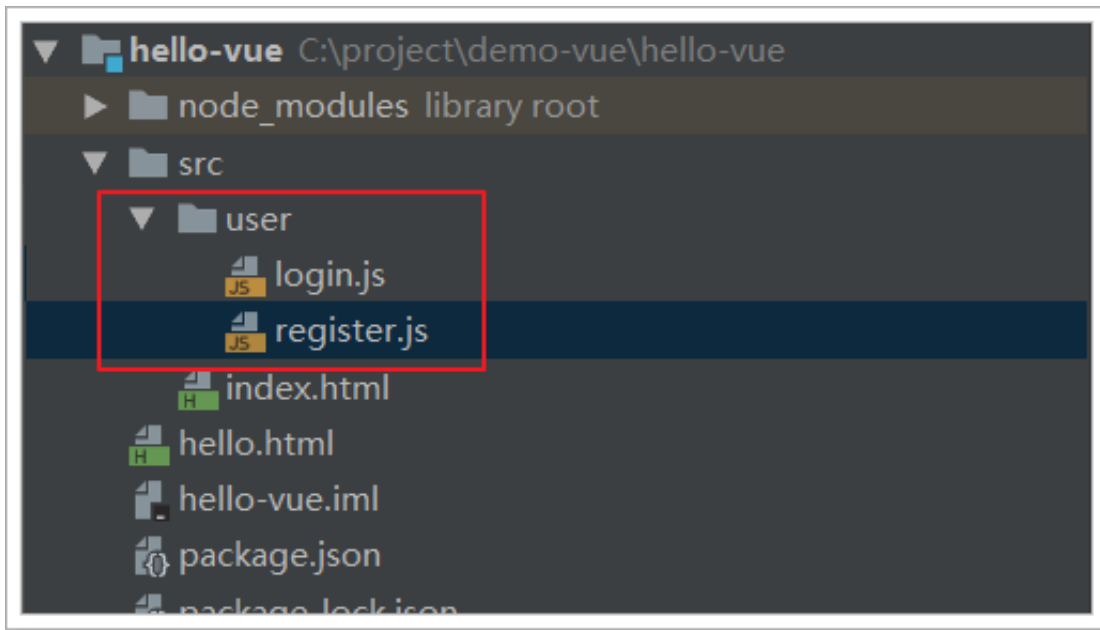
```
<script src="../../node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  var vm = new Vue({
    el:"#app"
  })
</script>
```

样式：



7.1.2.编写登录及注册组件

接下来我们来实现登录组件，以前我们都是写在一个文件中，但是为了复用性，开发中都会把组件放入独立的JS文件中，我们新建一个user目录以及login.js及register.js：



编写组件，这里我们只写模板，不写功能。

login.js内容如下：

```
const loginForm = {  
  template: '\n      <div>\n        <h2>登录页</h2> \n        用户名: <input type="text"><br/>\n        密码: <input type="password"><br/>\n      </div>\n    ',  
}
```

template里面只能由一个根标签，如果有两个根标签的话就会无法识别 报错，所以一般建议先写一个div，然后所有的内容都写在里面

这里并不是一个单引号，而是位于数字键盘最左边的那个按键

register.js内容：

```
const registerForm = {  
  template: '\n      <div>\n        <h2>注册页</h2> \n        用&ensp;户&ensp;名: <input type="text"><br/>\n        密&emsp;&emsp;码: <input type="password"><br/>\n        确认密码: <input type="password"><br/>\n      </div>\n    ',  
}
```

占据半个汉字空间

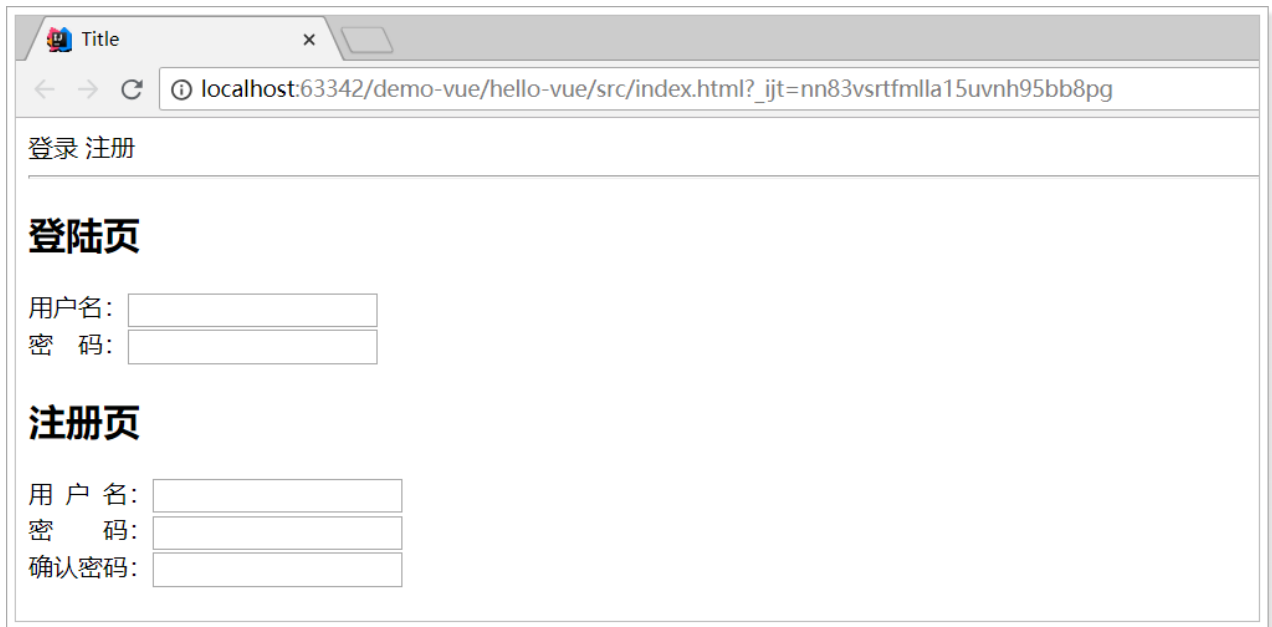
占据一个汉字空间

7.1.3.在父组件中引用

```
<div id="app">
  <span>登录</span>
  <span>注册</span>
  <hr/>
  <div>
    <!--<loginForm></loginForm>-->
    <!--
      疑问：为什么不采用上面的写法？
      由于html是大小写不敏感的，如果采用上面的写法，则被认为是<loginform></loginform>
      所以，如果是驼峰形式的组件，需要把驼峰转化为“-”的形式
    -->
    <login-form></login-form>
    <register-form></register-form>
  </div>
</div>
<script src="../../node_modules/vue/dist/vue.js"></script>
<script src="user/login.js"></script>
<script src="user/register.js"></script>
<script type="text/javascript">
  var vm = new Vue({
    el: "#app",
    components: {
      loginForm,
      registerForm
    }
  })
</script>
```

可以自动将短横线转换为驼峰形式

效果：



7.1.5.问题

我们期待的是，当点击登录或注册按钮，分别显示登录页或注册页，而不是一起显示。

但是，如何才能动态加载组件，实现组件切换呢？

虽然使用原生的Html5和JS也能实现，但是官方推荐我们使用vue-router模块。

7.2.vue-router简介和安装

使用vue-router和vue可以非常方便的实现 复杂单页应用的动态路由功能。

官网：<https://router.vuejs.org/zh-cn/>

使用npm安装：`npm install vue-router --save`

```
C:\project\demo-vue\hello-vue>npm install vue-router --save
npm WARN hello-vue@1.0.0 No description
npm WARN hello-vue@1.0.0 No repository field.

+ vue-router@3.0.1
added 1 package in 0.656s
```

在index.html中引入依赖：

```
<script src="../../node_modules/vue-router/dist/vue-router.js"></script>
```

7.3.快速入门

新建vue-router对象，并且指定路由规则：

```
// 创建VueRouter对象
const router = new VueRouter({
  routes: [ // 编写路由规则 是一个数组，可以写许多个路由
    {
      path: "/login", // 请求路径，以"/"开头
      component: loginForm // 组件名称
    },
    {
      path: "/register",
      component: registerForm
    }
  ]
})
```

- 创建VueRouter对象，并指定路由参数
- routes: 路由规则的数组，可以指定多个对象，每个对象是一条路由规则，包含以下属性：
 - path: 路由的路径
 - component: 组件名称

在父组件中引入router对象：

```
var vm = new Vue({
  el: "#app",
  components: { // 引用登录和注册组件
    loginForm,
    registerForm
  },
  router // 引用上面定义的router对象
})
```

页面跳转控制：


```

<div id="app">
  <!--router-link来指定跳转的路径-->
  <span><router-link to="/login">登录</router-link></span>
  <span><router-link to="/register">注册</router-link></span>
  <hr/>
  <div>
    <!--vue-router的锚点-->
    <router-view></router-view>
  </div>
</div>

```

- 通过 `<router-view>` 来指定一个锚点，当路由的路径匹配时，vue-router会自动把对应组件放到锚点位置进行渲染
- 通过 `<router-link>` 指定一个跳转链接，当点击时，会触发vue-router的路由功能，路径中的hash值会随之改变

效果：



注意：单页应用中，页面的切换并不是页面的跳转。仅仅是地址最后的hash值变化。

事实上，我们总共就一个HTML：index.html

