# 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

能够使用vue常见指令

能够使用vue计算属性和watch监控能够编写Vue组件

能够使用axios发送异步请求获取数据

# Vue概述

## 前言

先了解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**。

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。

基于时间循环的异步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diamante，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。

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：



MVVM模式

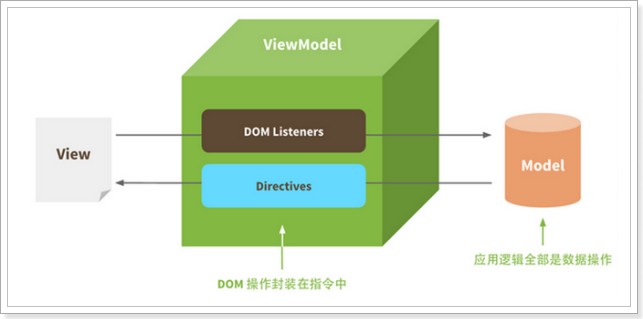
M：即Model，模型，包括数据和一些基本操作V：即View，视图，页面渲染结果

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，然后要通过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。

而今天要学习的，就是一款MVVM模式的框架：Vue

## 认识Vue

Vue (读音 /vjuː/，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](https://cn.vuejs.org/v2/guide/single-file-components.html)以及各种[支持类库](https://github.com/vuejs/awesome-vue#libraries--plugins)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。

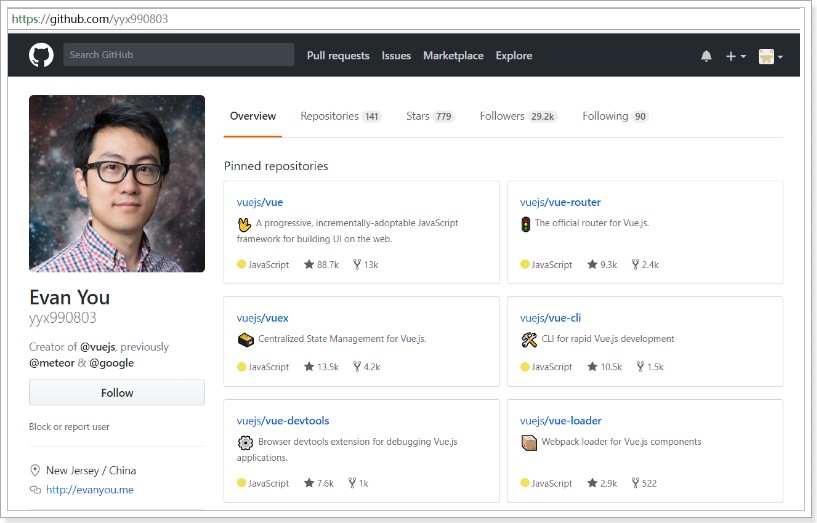
渐进式：可以选择性的使用该框架的一个或一些组件，这些组件的使用也不需要将框架全部组件都应用；而且 用了这些组件也不要求你的系统全部都使用该框架。

官网：<https://cn.vuejs.org/>

参考：<https://cn.vuejs.org/v2/guide/>



Git地址：<https://github.com/vuejs>

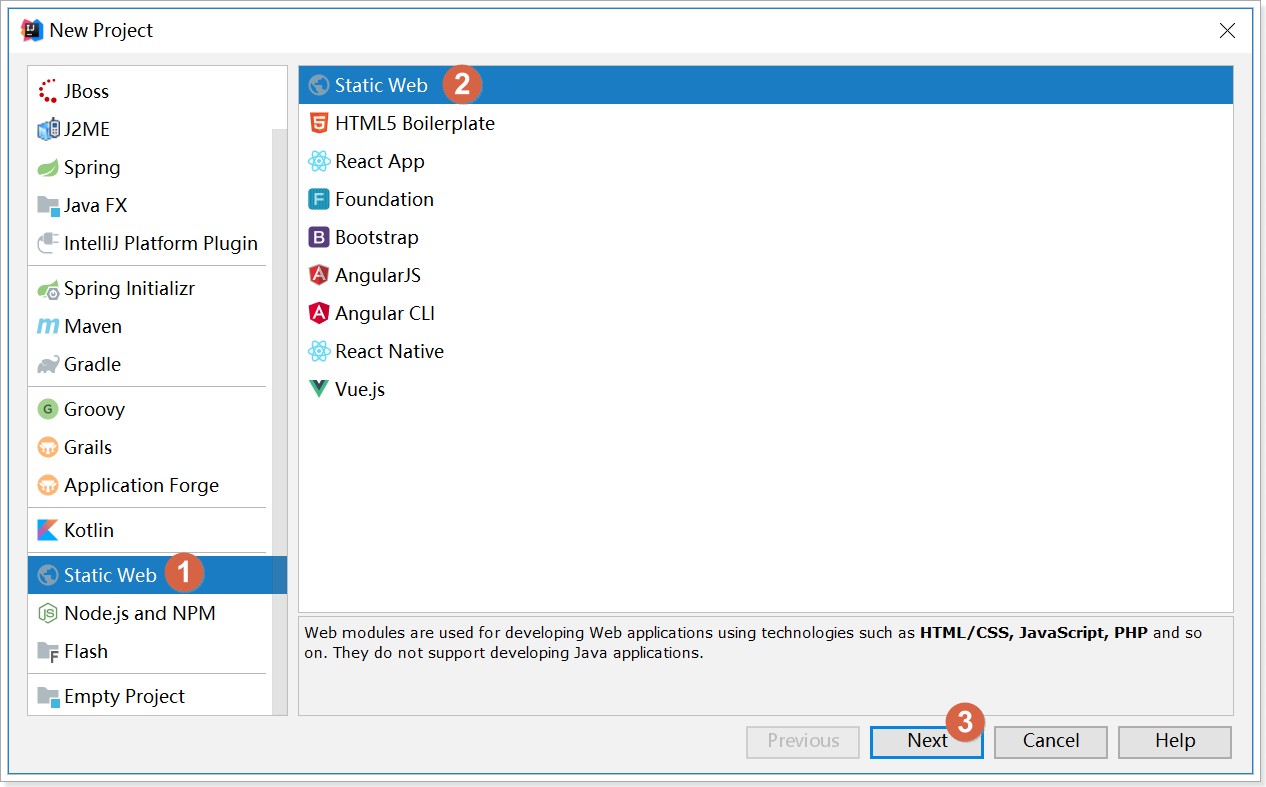


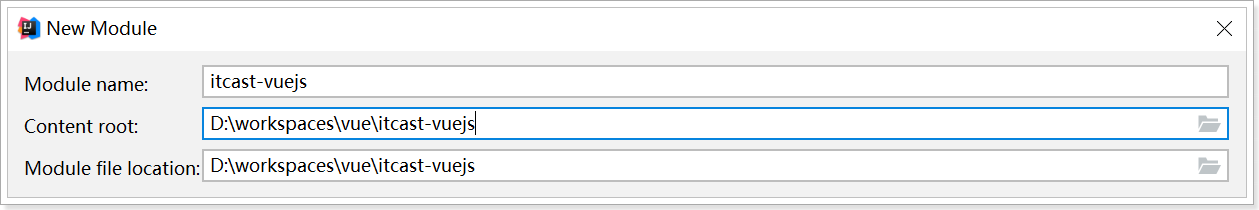
**尤雨溪**，Vue.js 创作者，Vue Technology创始人，致力于Vue的研究开发。

# 快速入门

## 创建工程

创建一个新的工程；选择静态web类型工程：





## 安装vue

### 下载安装

下载地址：<https://github.com/vuejs/vue>

可以下载2.6.10版本<https://github.com/vuejs/vue/archive/v2.6.10.zip>或资料 文件夹中也已下载下载解压，在dist 可得到vue.js文件。

### 使用CDN

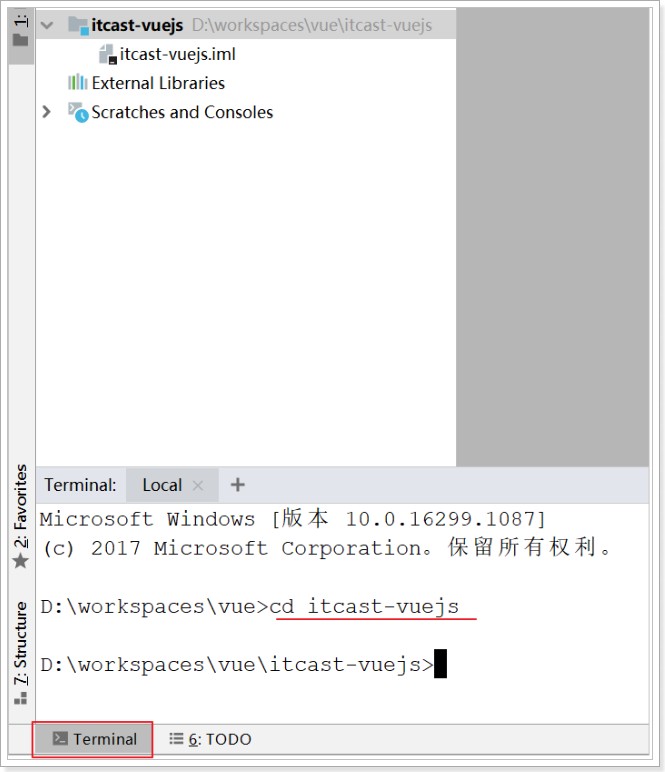
或者也可以直接使用公共的CDN（内容分发网络）服务：

1. <!-- 开发环境版本，包含了用帮助的命令行警告 -->
2. <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

或者：

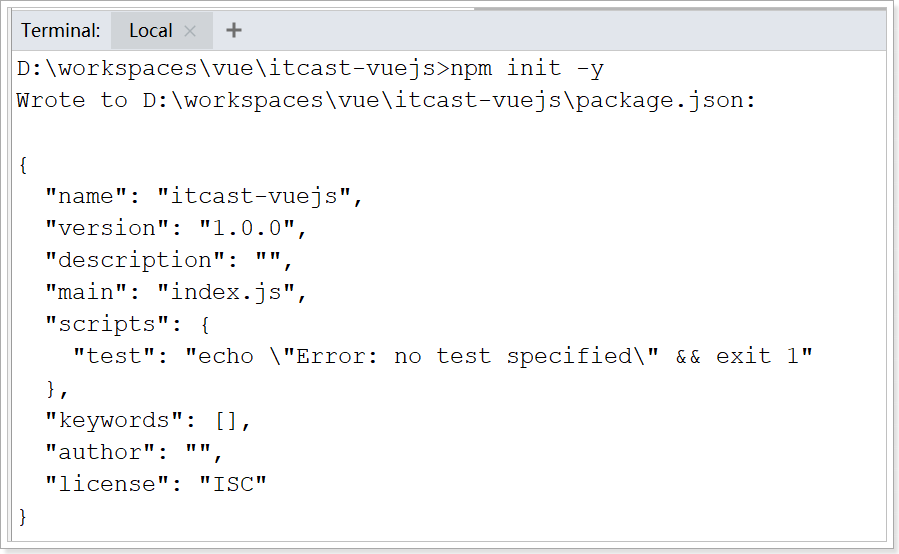
1. <!-- 生产环境版本，优化了尺寸和速度 -->
2. <script src="https://cdn.jsdelivr.net/npm/vue"></script>

### npm安装(推荐)

在idea的左下角，有个Terminal按钮，点击打开控制台：

先输入：

1 npm init -y

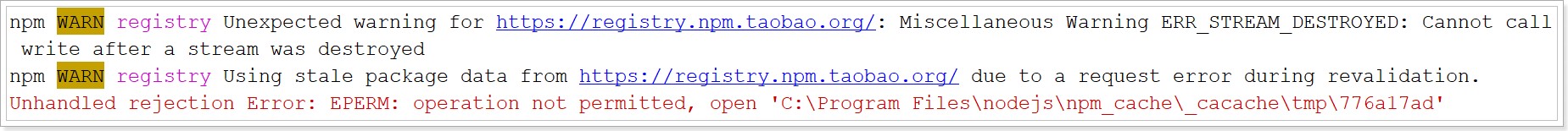
对项目进行初始化

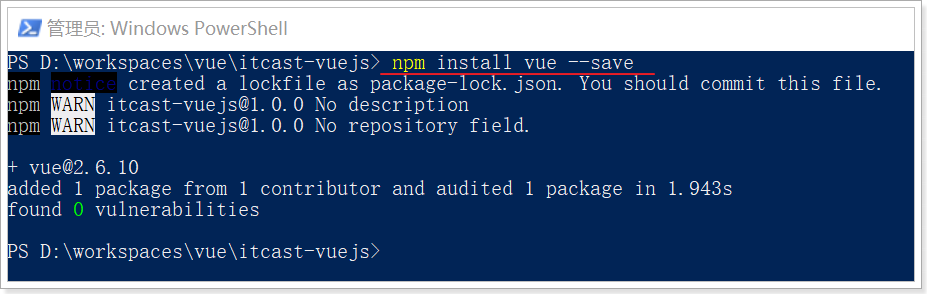
此时，会在项目目录下出现一个package.json文件。

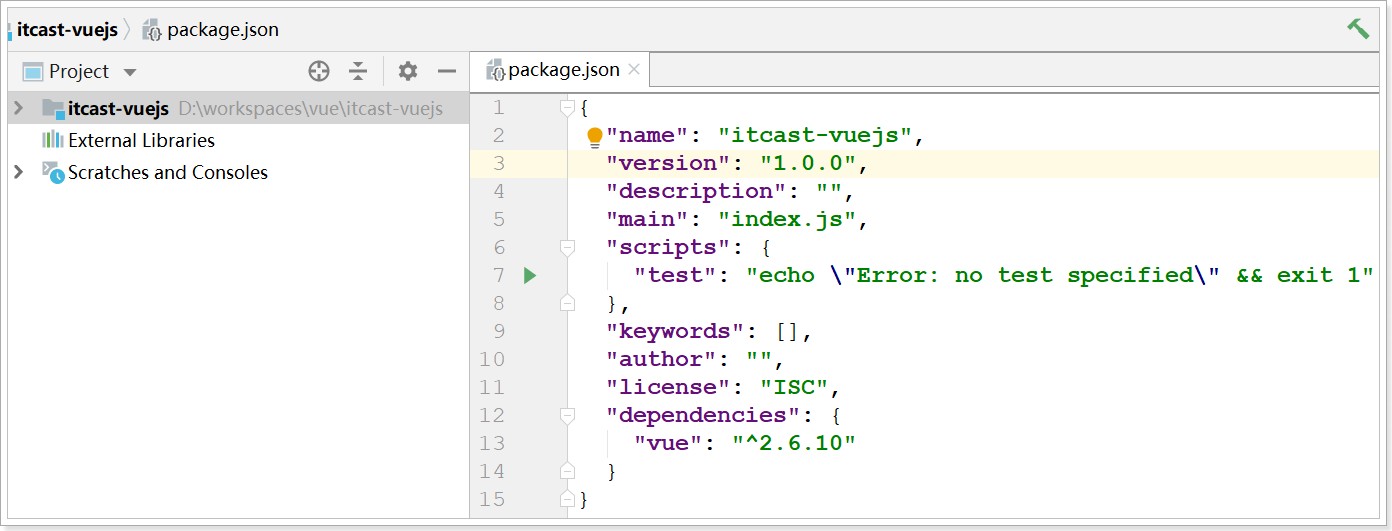
这是对项目的基本描述信息。例如名称、版本等，有点类似java中的pom文件。安装Vue，输入命令：

1. # save 的意思是将模块安装到项目目录下，并在package文件的dependencies节点写入依赖
2. npm install vue --save

如果执行的时候出现如下错误；那么使用**管理员身份**进入到项目路径下执行上述命令：





然后就会在项目目录发现一个node\_modules目录，并且在下面有一个vue目录。

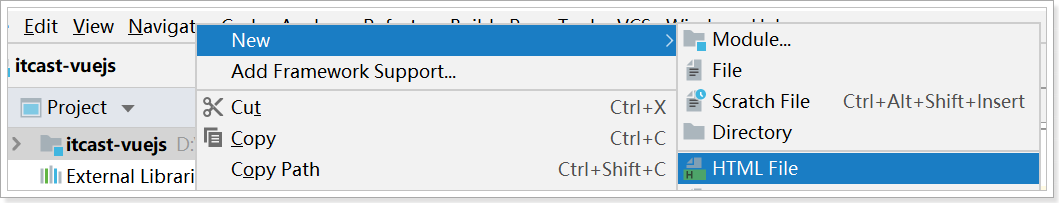
node\_modules是通过npm安装的所有模块的默认位置。 此时再查看package.json，会发现有了变化：

会发现，刚刚安装的vue依赖再这里出现了描述信息。是不是跟pom文件很像？

## vue入门案例

### HTML模板

在项目目录新建一个HTML文件 01-demo.html



### vue渲染

01-demo.html内容如下：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>vuejs测试</title>

</head>

<body>

<div id="app">

<h2>{{name}} 非常酷！</h2>

</div>

<script src="node\_modules/vue/dist/vue.js"></script>

<script>

var app = new Vue({

el:"#app",//el即element，要渲染的页面元素data: {

name:"黑马"

}

});

</script>

</body>

</html>

首先通过 new Vue()来创建Vue实例

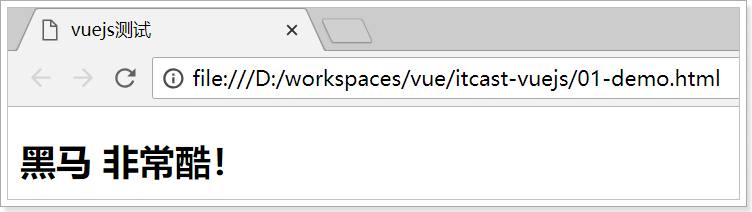
然后构造函数接收一个对象，对象中有一些属性：

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

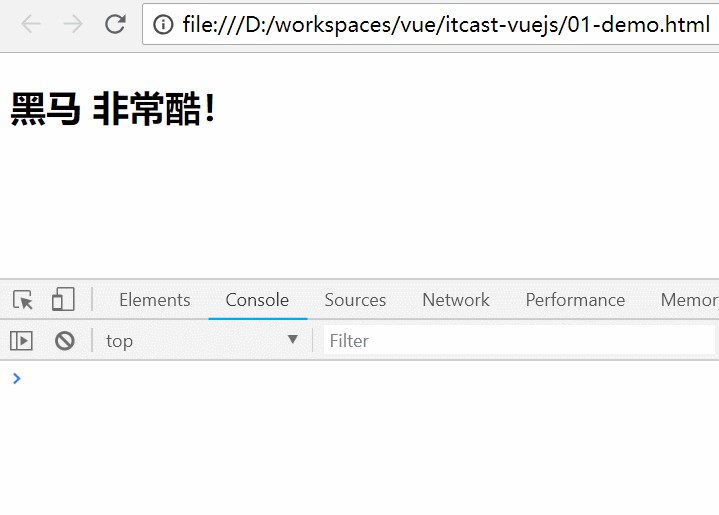
name：这里指定了一个name属性

页面中的h2 元素中，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：



### 双向绑定

对刚才的案例进行简单修改：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. </head>
7. <body>
8. <div id="app">
9. <input type="text" v-model="num"> 10 <h2>
10. {{name}} 非常酷！
11. 有{{num}}个酷炫学科。

13 </h2>

1. </div>
2. <script src="node\_modules/vue/dist/vue.js"></script>
3. <script>
4. var app = new Vue({
5. el:"#app",//el即element，要渲染的页面元素
6. data: {
7. name:"黑马",

21

22

23

24

25

26

27

num: 1

}

});

</script>

</body>

</html>

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：



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。

### 事件处理

在页面添加一个按钮：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. </head>
7. <body>
8. <div id="app">
9. <input type="text" v-model="num"><button v-on:click="num++">点我</button> 10 <h2>
10. {{name}} 非常酷！
11. 有{{num}}个酷炫学科。

13 </h2>

1. </div>
2. <script src="node\_modules/vue/dist/vue.js"></script>
3. <script>

17

18

19

20

21

22

23

24

25

26

27

var app = new Vue({

el:"#app",//el即element，要渲染的页面元素data: {

name:"黑马",

num: 1

}

});

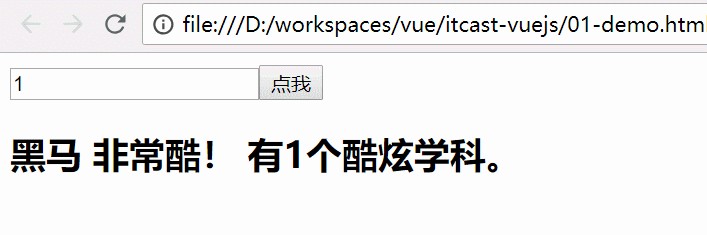
</script>

</body>

</html>

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onclick是无法直接操作num的。

效果：



# Vue实例

## 创建Vue实例

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

Vue

1 var vm = new Vue({ 2 // 选 项

3 })

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

...

接下来一 一介绍。

## 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；可以通过el属性来指定。例如一段html模板：

1 <div id="app"> 2

3 </div>

然后创建Vue实例，关联这个div

1. var vm = new Vue({
2. el:"#app" 3 })

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。

## 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：

1. <div id="app">
2. <input type="text" v-model="name"/> 3 </div>

js:

1 var vm = new Vue({ 2 el:"#app",

1. data:{
2. name:"黑马" 5 }

6 })

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

$: new Vue 里初始化的了的，都可以用$+name 来调用,以便与用户定义的属性区分开来

|  |
| --- |
| <script>  var ***data***={  a:10,  b:"ox"  }  var ***app*** = new Vue({  el:"#app",  data:***data*** });  ***console***.log(***app***.a);  ***console***.log(***data***.a);  ***console***.log(***app***.a==***data***.a);//app.a app可以直接访问数据  ***console***.log(***app***.$el);  ***console***.log(***app***.$data)  ***console***.log(***app***.$el==***document***.getElementById("app")) //app.$el 即得到整个<div>元素  ***console***.log(***app***) //得到了Vue对象  ***console***.log(***app***.el) //未定义  ***console***.log(***app***.$el==***app***.el); </script> |

## 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的作用范围内使用。html:

1 <div id="app">

2

<button v-on:click="add">点我</button>

3 </div>

js:

1 var vm = new Vue({ 2 el:"#app",

3 data:{

4 },

1. methods:{
2. add:function(){
3. console.log("点我了...233") 8 }

9 }

10 })

## 生命周期钩子

### 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。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。

所有的生命周期钩子自动绑定 上下文到实例中，因此你可以访问数据，对属性和方法进行运算。这意味

this

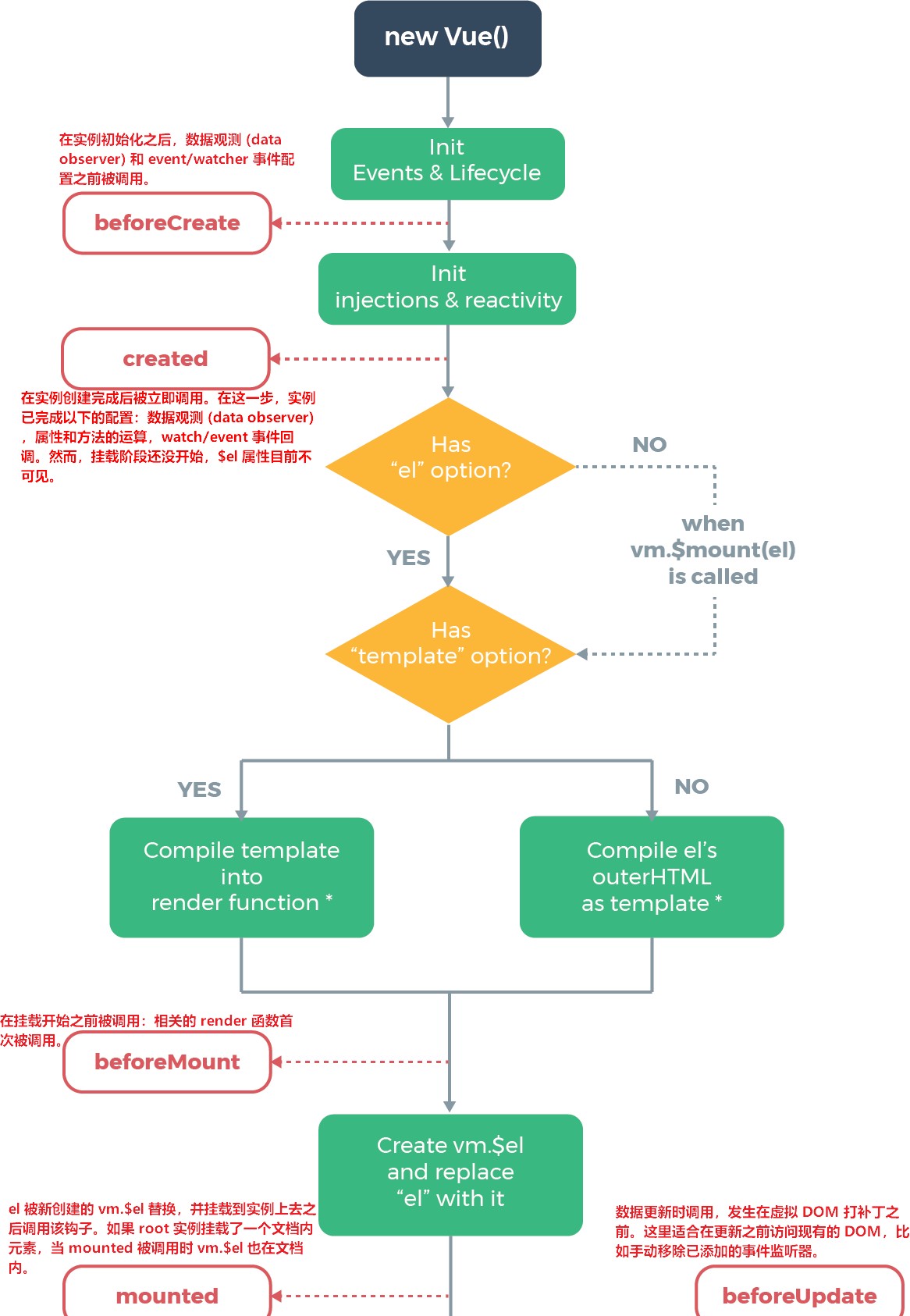
着**你不能使用箭头函数来定义一个生命周期方法** (例如 created: () => this.fetchTodos() )。这是因为箭

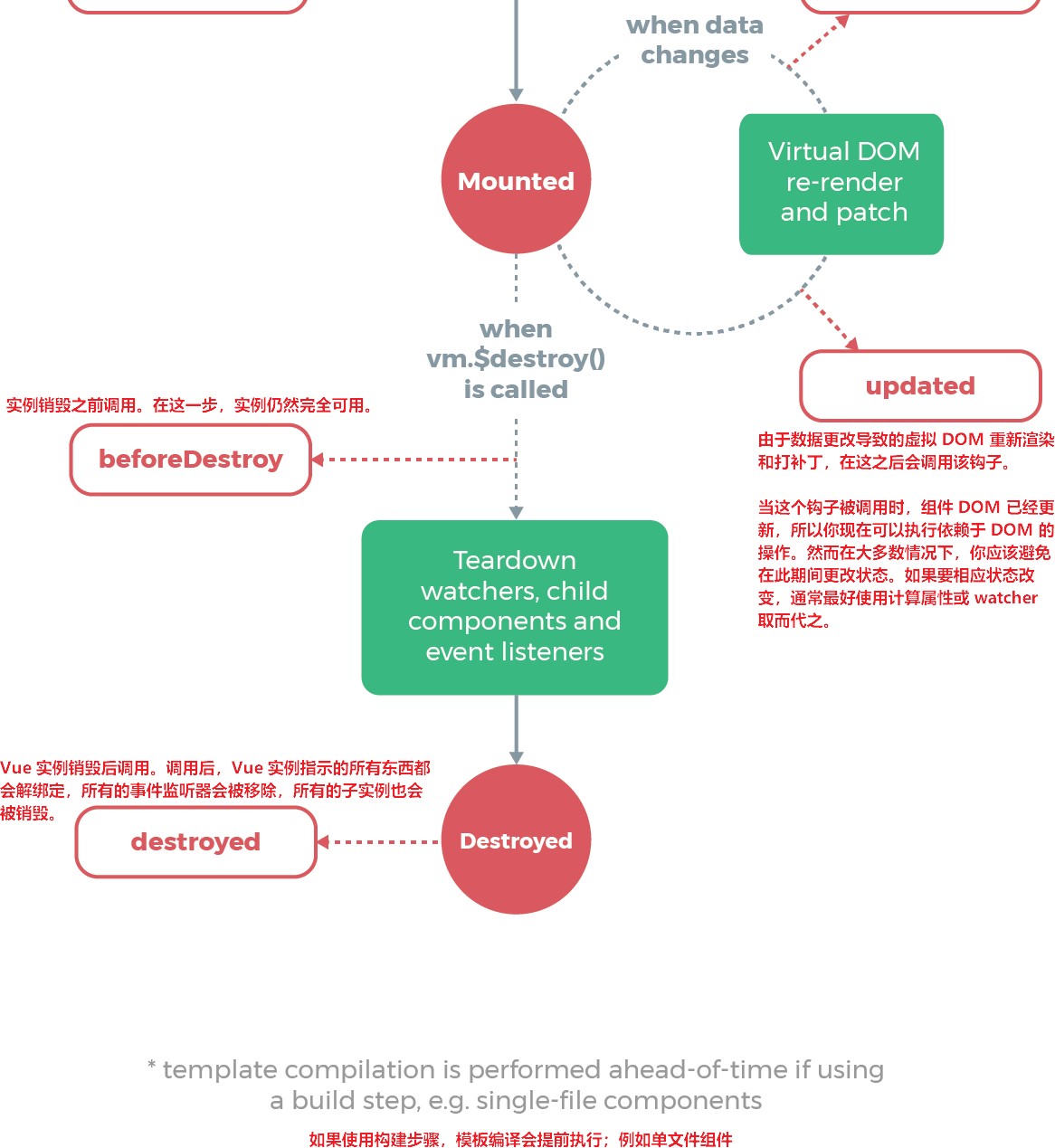
this

头函数绑定了父上下文，因此

生命周期：

与你期待的 Vue 实例不同， this.fetchTodos 的行为未定义。





vm.$el ：Vue 实例使用的根 DOM 元素

vm.$root ：当前组件树的根 Vue 实例。如果当前实例没有父实例，此实例将会是其自己。

### 钩子函数

例如：created代表在vue实例创建后；

可以在Vue中定义一个created函数，代表这个时期的构造函数：

创建示例html页面02-lifecycle.html如下：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. <script src="node\_modules/vue/dist/vue.js"></script>
7. </head>
8. <body>
9. <div id="app"> 10 {{msg}}
10. </div>
11. <script>
12. let app = new Vue({
13. el:"#app",
14. data:{
15. //初始化为空
16. msg:""

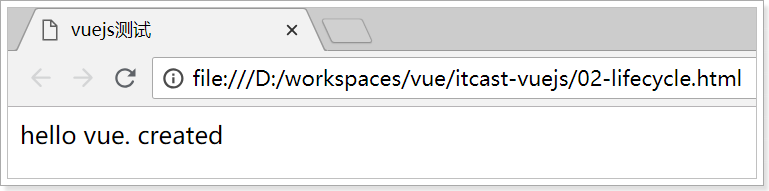
18 },

1. created(){
2. this.msg = "hello vue. created"; 21 }

22 });

1. </script>
2. </body>
3. </html>

结果：



### 3.5.3.this

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

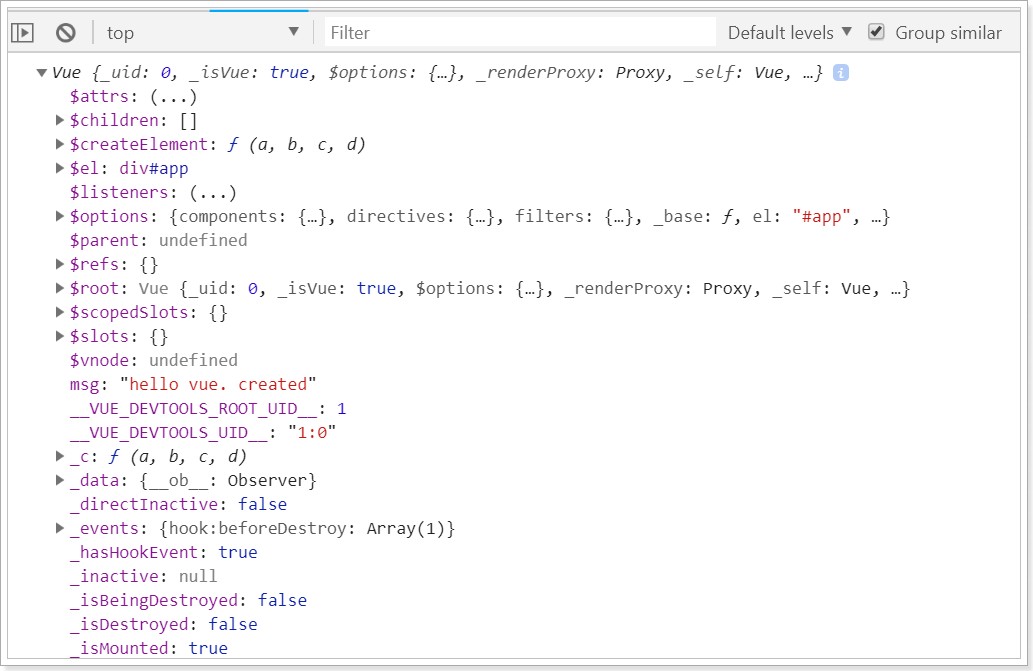
1 let vm = new Vue({ 2 el:"#app",

1. data:{
2. //初始化为空
3. msg:""

6 },

1. created(){
2. this.msg = "hello vue. created";
3. console.log(this); 10 }

11 })

控制台的输出：

总结： this 就是当前的Vue实例，在Vue对象内部，必须使用 this 才能访问到Vue中定义的data内属性、方法等。

# 指令

什么是指令？

指令 (Directives) 是带有

前缀的特殊属性。例如在入门案例中的v-model，代表双向绑定。



v-

## 插值表达式

### 花括号

格式：

1 {{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：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. <script src="node\_modules/vue/dist/vue.js"></script>
7. </head>
8. <body>
9. <div id="app"> 10 {{msg}}
10. </div>
11. <script>
12. let app = new Vue({
13. el:"#app",
14. data:{
15. msg:"hello vue" 17 }

18 });

1. </script>
2. </body>
3. </html>

### 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，称为插值闪烁。类似如下的效果（最新vue是几乎没有此问题）：

### v-text和v-html

使用v-text和v-html指令来替代 {{}}

说明：

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

示例：

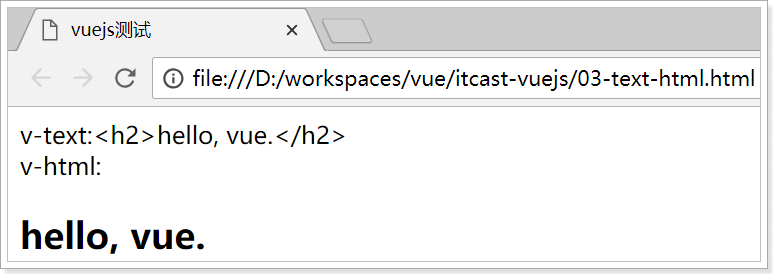
改造原页面内容为:

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. <script src="node\_modules/vue/dist/vue.js"></script>
7. </head>
8. <body>
9. <div id="app">
10. v-text:<span v-text="msg"></span><br>
11. v-html:<span v-html="msg"></span><br>
12. </div>
13. <script>
14. let app = new Vue({
15. el:"#app",
16. data:{
17. msg:"<h2>hello, vue.</h2>" 18 }

19 });

1. </script>
2. </body>
3. </html>

效果：



并且不会出现插值闪烁，当没有数据时，会显示空白。

## 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的可使用元素有：

input select textarea checkbox radio

components（Vue中的自定义组件）

基本上除了最后一项，其它都是表单的输入项。示例：

html：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. <script src="node\_modules/vue/dist/vue.js"></script>
7. </head>
8. <body>
9. <div id="app">
10. <input type="checkbox" value="Java" v-model="language">Java<br>
11. <input type="checkbox" value="PHP" v-model="language">PHP<br>
12. <input type="checkbox" value="Swift" v-model="language">Swift<br> 13 <h2>

14 你选择了：{{language.join(",")}} 15 </h2>

16 </div>

1. <script>
2. let app = new Vue({
3. el:"#app",
4. data:{
5. language:[]

22 }

23 });

1. </script>
2. </body>
3. </html>

多个checkbox 对应一个model时，model的类型是一个数组，单个checkbox值是boolean类型

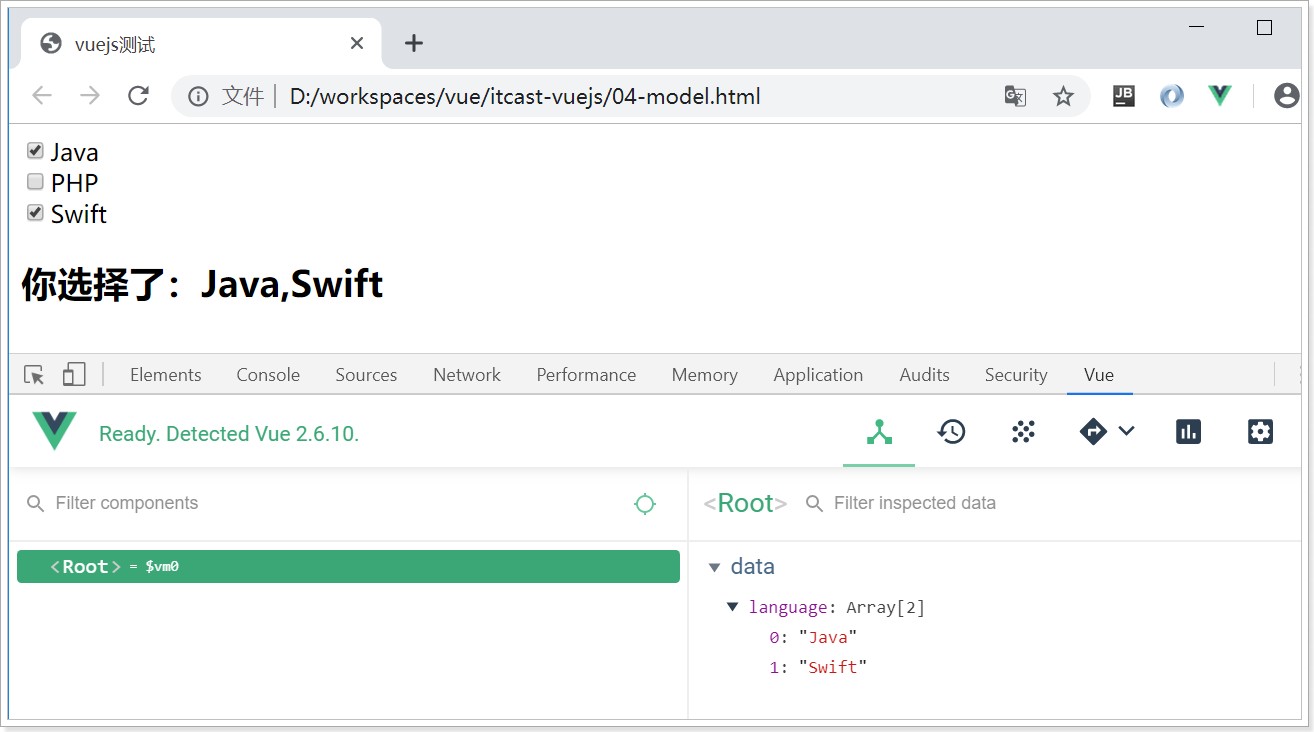
radio对应的值是input的value值

和textarea 默认对应的model是字符串

input

select 单选对应字符串，多选对应也是数组

效果：



要使用上图中的chrome浏览器的 vue插件的话；可以找

资料\chrome vue插件\安装说明

## v-on

### 基本用法

v-on指令用于给页面元素绑定事件。语法：

查看并安装。

1 v-on:事件名="js片段或函数名"

简写语法：

1 @事件名="js片段或函数名"

例如 可以简写为@click='add'

v-on:click='add'

示例：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. <script src="node\_modules/vue/dist/vue.js"></script>
7. </head>
8. <body>
9. <div id="app">
10. <!--直接写js片段-->
11. <button @click="num++">增加</button>
12. <!--使用函数名，该函数必须要在vue实例中定义-->
13. <button @click="decrement">减少</button>

14 <h2>

15 num = {{num}} 16 </h2>

1. </div>
2. <script>
3. let app = new Vue({
4. el:"#app",
5. data:{
6. num:1

23 },

1. methods:{
2. decrement(){
3. this.num--;

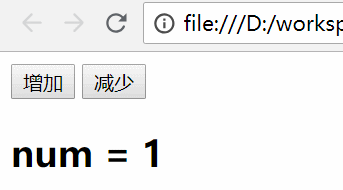
27 }

28 }

29 });

1. </script>
2. </body>
3. </html>

效果：



### 事件修饰符

在事件处理程序中调用 或

event.preventDefault()

是非常常见的需求。尽管我们

event.stopPropagation()

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。

v-on

为了解决这个问题，Vue.js 为

：阻止事件冒泡

.stop

提供了**事件修饰符**。之前提过，修饰符是由点开头的指令后缀来表示的。

.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

示例：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>vuejs测试</title>

<script src="node\_modules/vue/dist/vue.js"></script>

</head>

<body>

<div id="app">

<!--直接写js片段-->

<button @click="num++">增加</button>

<!--使用函数名，该函数必须要在vue实例中定义-->

<button @click="decrement">减少</button>

<h2>

num = {{num}}

</h2>

<hr>

事件冒泡测试：<br>

<div style="background-color: lightblue;width: 100px;height: 100px" @click="print('你点击了div')">

<button @click.stop="print('点击了button')">点我试试</button>

</div>

<br>阻止默认事件：<br>

<a href="[http://www.itcast.cn](http://www.itcast.cn/)" @click.prevent="print('点击超链接')">传智播客</a>

1. </div>
2. <script>
3. let app = new Vue({
4. el:"#app",
5. data:{
6. num:1

32 },

1. methods:{
2. decrement(){
3. this.num--;

36 },

1. print(msg){
2. console.log(msg)

39 }

40 }

41 });

1. </script>
2. </body>
3. </html>

## 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。

### 遍历数组

语法：

1 v-for="item in items"

items：要遍历的数组，需要在vue的data中定义好。item：循环变量

示例：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. </head>
7. <body>
8. <div id="app"> 9 <ul>

10 <li v-for="user in users">{{user.name}}--{{user.age}}--{{user.gender}}</li> 11 </ul>

1. </div>
2. <script src="node\_modules/vue/dist/vue.js"></script>
3. <script>
4. var app = new Vue({

16

17

18

19

20

21

22

23

24

25

26

27

28

el:"#app",//el即element，要渲染的页面元素

data: {

users:[

{"name":"黑马","age":8,"gender":"男"},

{"name":"传智播客","age":12,"gender":"女"},

{"name":"酷丁鱼","age":4,"gender":"男"}

]

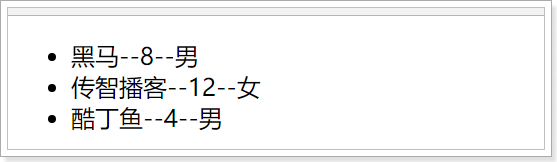
}

});

</script>

</body>

</html>

效果：

### 数组角标

在遍历的过程中，如果需要知道数组角标，可以指定第二个参数： 语法

1 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

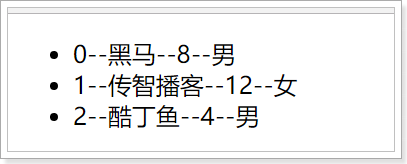
1 <div id="app"> 2 <ul>

1. <li v-for="(user, index) in users">
2. {{index}}--{{user.name}}--{{user.age}}--{{user.gender}} 5 </li>

6 </ul>

7 </div>

效果：



### 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：

1. v-for="value in object"
2. v-for="(value,key) in object"
3. v-for="(value,key,index) in object"

1个参数时，得到的是对象的值

2个参数时，第一个是值，第二个是键

3个参数时，第三个是索引，从0开始

示例：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. </head>
7. <body>
8. <div id="app"> 9 <ul>
9. <li v-for="(user, index) in users" :key="index">
10. {{index}}--{{user.name}}--{{user.age}}--{{user.gender}} 12 </li>

13 </ul>

1. <hr>
2. <ul>
3. <li v-for="(value,key,index) in person"> 17 {{index}}--{{key}}--{{value}}

18 </li>

19 </ul>

1. </div>
2. <script src="node\_modules/vue/dist/vue.js"></script>
3. <script>
4. var app = new Vue({
5. el:"#app",//el即element，要渲染的页面元素
6. data: {
7. users:[

27 {"name":"黑马","age":8,"gender":"男"},

28

29

30

31

32

33

34

35

36

37

{"name":"传智播客","age":12,"gender":"女"},

{"name":"酷丁鱼","age":4,"gender":"男"}

],

person:{"name":"传智汇", "age":3, "address":"中国"}

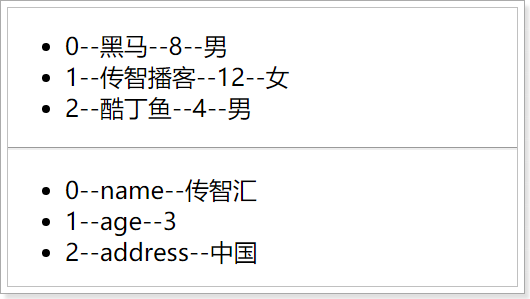
}

});

</script>

</body>

</html>

效果：

### 遍历数字

|  |
| --- |
| 打印乘法表 <!DOCTYPE html> <html lang="en"> <head>  <meta charset="UTF-8">  <title>Title</title> </head> <body>  <div id="app">  <table border="1">  <tr v-for="i in num">  <td v-for="j in i">{{i}}\*{{j}}={{i\*j}}</td>  </tr>  </table>  </div>  <script src="node\_modules/vue/dist/vue.js"></script>  <script>  new Vue({  el:"#app",  data:{  num:9  }  });  </script> </body> </html> |

效果：

### 

## 4.5.v-if和v-show

### 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：

1 v-if="布尔表达式"

示例：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>vuejs测试</title>

</head>

<body>

<div id="app">

<button @click="show = !show">点我</button>

<h2 v-if="show"> Hello VueJS.

</h2>

</div>

<script src="node\_modules/vue/dist/vue.js"></script>

<script>

var app = new Vue({

el:"#app",//el即element，要渲染的页面元素data: {

show:true

}

});

</script>

</body>

</html>

效果：



### 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。示例：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>vuejs测试</title>

</head>

<body>

<div id="app">

<button @click="show = !show">点我</button>

<h2 v-if="show"> Hello VueJS.

</h2>

<hr>

<ul>

<li v-for="(user,index) in users" v-if="user.gender=='女'">

{{index}}--{{user.name}}--{{user.age}}--{{user.gender}}

</li>

</ul>

</div>

<script src="node\_modules/vue/dist/vue.js"></script>

<script>

var app = new Vue({

el:"#app",//el即element，要渲染的页面元素data: {

show:true,

users:[

{"name":"黑马","age":8,"gender":"男"},

{"name":"传智播客","age":12,"gender":"女"},

{"name":"酷丁鱼","age":4,"gender":"男"},

{"name":"传智大学","age":2,"gender":"女"}

]

}

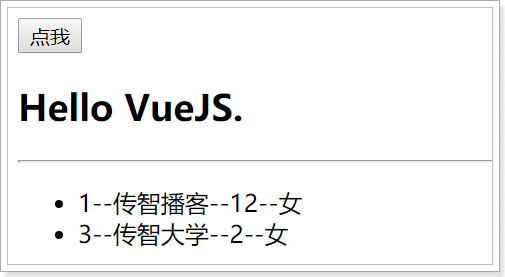
});

</script>

</body>

</html>

效果：



### v-else

v-if

可以使用

v-else

指令来表示

的“else 块”：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. </head>
7. <body>
8. <div id="app">
9. <button @click="show = !show">点我</button>
10. <h2 v-if="show">
11. Hello VueJS. 12 </h2>
12. <hr>
13. <ul v-if="show">
14. <li v-for="(user,index) in users" v-if="user.gender=='女'" style="background- color: deeppink">
15. {{index}}--{{user.name}}--{{user.age}}--{{user.gender}} 17 </li>
16. <li v-else style="background-color: blue">
17. {{index}}--{{user.name}}--{{user.age}}--{{user.gender}} 20 </li>

21 </ul>

1. </div>
2. <script src="node\_modules/vue/dist/vue.js"></script>
3. <script>
4. var app = new Vue({
5. el:"#app",//el即element，要渲染的页面元素
6. data: {
7. show:true,
8. users:[

30 {"name":"黑马","age":8,"gender":"男"},

31 {"name":"传智播客","age":12,"gender":"女"},

32 {"name":"酷丁鱼","age":4,"gender":"男"},

33

34

35

36

37

38

39

40

{"name":"传智大学","age":2,"gender":"女"}

]

}

});

</script>

</body>

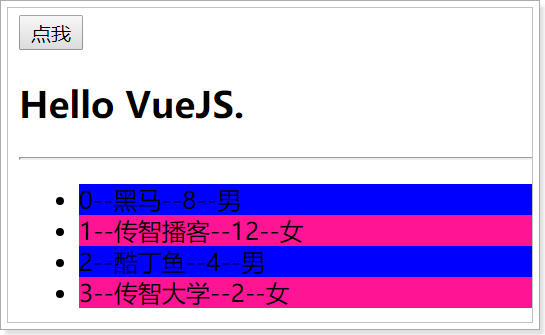
</html>

元素必须紧跟在带 或者 的元素的后面，否则它将不会被识别。

v-else

v-if

v-else-if



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

v-if

1. <div v-if="type === 'A'">
2. A
3. </div>
4. <div v-else-if="type === 'B'">
5. B
6. </div>
7. <div v-else-if="type === 'C'">
8. C
9. </div>
10. <div v-else>
11. Not A/B/C
12. </div>

类似于 v-else ， v-else-if 也必须**紧跟**在带 或者

v-if

v-else-if

### v-show

的元素之后。

另一个用于根据条件展示元素的选项是 指令。用法大致一样：

v-show

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

不同的是带有 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 。

v-show

示例：

* + - 1. <!DOCTYPE html>
      2. <html lang="en">
      3. <head>
      4. <meta charset="UTF-8">
      5. <title>vuejs测试</title>
      6. </head>
      7. <body>
      8. <div id="app">
      9. <button @click="show = !show">点我</button>
      10. <h2 v-if="show">
      11. Hello VueJS. 12 </h2>

1. <hr>
2. <ul v-if="show">
3. <li v-for="(user,index) in users" v-if="user.gender=='女'" style="background- color: deeppink">
4. {{index}}--{{user.name}}--{{user.age}}--{{user.gender}} 17 </li>
5. <li v-else style="background-color: blue">
6. {{index}}--{{user.name}}--{{user.age}}--{{user.gender}} 20 </li>

21 </ul>

1. <hr>
2. <h2 v-show="show">
3. 你好；黑马！

25 </h2>

1. </div>
2. <script src="node\_modules/vue/dist/vue.js"></script>
3. <script>
4. var app = new Vue({
5. el:"#app",//el即element，要渲染的页面元素
6. data: {
7. show:true,
8. users:[

34 {"name":"黑马","age":8,"gender":"男"},

35 {"name":"传智播客","age":12,"gender":"女"},

36 {"name":"酷丁鱼","age":4,"gender":"男"},

37 {"name":"传智大学","age":2,"gender":"女"}

38 ]

39 }

40 });

41 </script> 42

1. </body>
2. </html>

代码：



## v-bind

### 属性上使用vue数据

看这样一个案例；

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

<title>vuejs测试</title>

<style type="text/css"> div {

width: 100px; height: 100px; color: white;

}

.red {

background-color: red;

}

.blue {

background-color: blue;

}

</style>

</head>

<body>

<div id="app">

<button @click="color='red'">红色</button>

<button @click="color='blue'">蓝色</button>

<div class="">

点击按钮改变背景颜色。

</div>

</div>

<script src="node\_modules/vue/dist/vue.js"></script>

<script>

var app = new Vue({ el:"#app", data: {

color:"red"

}

});

</script>

</body>

</html>

解读：

页面有两个按钮，点击时，会改变Vue实例中的color值，这个值与前面定义的CSS样式一致。 目前div的class为空，希望实现点击按钮后，div的class样式会在.red和.blue之间切换

该如何实现？

大家可能会这么想，既然color值会动态变化为不同的class名称，那么我们直接把color注入到class属性就好了，于 是就这样写：

1 <div class="{{color}}"></div>

这样写是错误的！因为插值表达式不能用在标签的属性中。

此时，Vue提供了一个新的指令来解决：v-bind，语法：

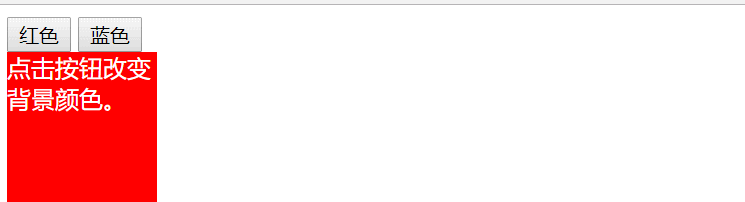
1 v-bind:属性名="Vue中的变量"

例如，在这里我们可以写成：

1 <div v-bind:class="color"></div>

不过，v-bind太麻烦，因此可以省略，直接写成： :属性名='属性值' ，即：

1 <div :class="color"></div>

效果：

### class属性的特殊用法

上面虽然实现了颜色切换，但是语法却比较啰嗦。

Vue对class属性进行了特殊处理，可以接收数组或对象格式 对象语法

可以传给 一个对象，以动态地切换 class：

:class

1 <div :class="{ red: true,blue:false }"></div>

对象中，key是已经定义的class样式的名称，如本例中的： red 和blue

对象中，value是一个布尔值，如果为true，则这个样式会生效，如果为false，则不生效。

之前的案例可以改写成这样：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. <style type="text/css">
7. div {
8. width: 100px;

9

10

11 }

height: 100px; color: white;

1. .red {
2. background-color: red; 14 }
3. .blue {
4. background-color: blue; 17 }
5. </style>
6. </head>
7. <body>
8. <div id="app">
9. <button @click="color='red'">红色</button>
10. <button @click="color='blue'">蓝色</button>
11. <div :class="color">
12. 点击按钮改变背景颜色。
13. </div>
14. <hr><br>
15. <button @click="bool=!bool">点我改变下面色块颜色</button>
16. <div :class="{red:bool,blue:!bool}">
17. 点击按钮改变背景颜色。
18. </div>
19. </div>
20. <script src="node\_modules/vue/dist/vue.js"></script>
21. <script>
22. var app = new Vue({
23. el:"#app",
24. data: {
25. color:"red",
26. bool: true

40 }

41 });

42 </script> 43

1. </body>
2. </html>

首先class绑定的是一个对象： {red:bool, blue: !bool}

red和blue两个样式的值分别是bool和!bool，也就是说这两个样式的生效标记恰好相反，一个生效，另一个失效。

bool默认为true，也就是说默认red生效，blue不生效

现在只需要一个按钮即可，点击时对bool取反，自然实现了样式的切换

效果：



## 计算属性

在插值表达式中使用js表达式是非常方便的，而且也经常被用到。

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，有一个日期的数 据，但是是毫秒值：

1 data:{

2 birthday:1429032123201 // 毫秒值

3 }

在页面渲染，希望得到yyyy-MM-dd的样式则需要如下处理：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>vuejs测试</title>

</head>

<body>

<div id="app">

<h2>

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

你的生日为：{{new Date(birthday).getFullYear()}}-{{new

Date(birthday).getMonth()+1}}-{{new Date(birthday).getDay()}}

</h2>

</div>

<script src="node\_modules/vue/dist/vue.js"></script>

<script>

var app = new Vue({

el:"#app",//el即element，要渲染的页面元素data: {

birthday:1429032123201

}

});

</script>

</body>

</html>

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：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. </head>
7. <body>
8. <div id="app"> 9 <h2>

10 你的生日为：{{new Date(birthday).getFullYear()}}-{{new Date(birthday).getMonth()+1}}-{{new Date(birthday).getDay()}}

11 </h2>

12 <hr>

13 <h2>

14 computed计算方式；你的生日为：{{birth}} 15 </h2>

1. </div>
2. <script src="node\_modules/vue/dist/vue.js"></script>
3. <script>
4. var app = new Vue({
5. el:"#app",//el即element，要渲染的页面元素
6. data: {

22 birthday:1429032123201

23 },

1. computed:{
2. birth(){
3. const date = new Date(this.birthday);
4. return date.getFullYear() + "-" + (date.getMonth()+1) + "-" + date.getDay();

28 }

29 }

30 });

31

32

33

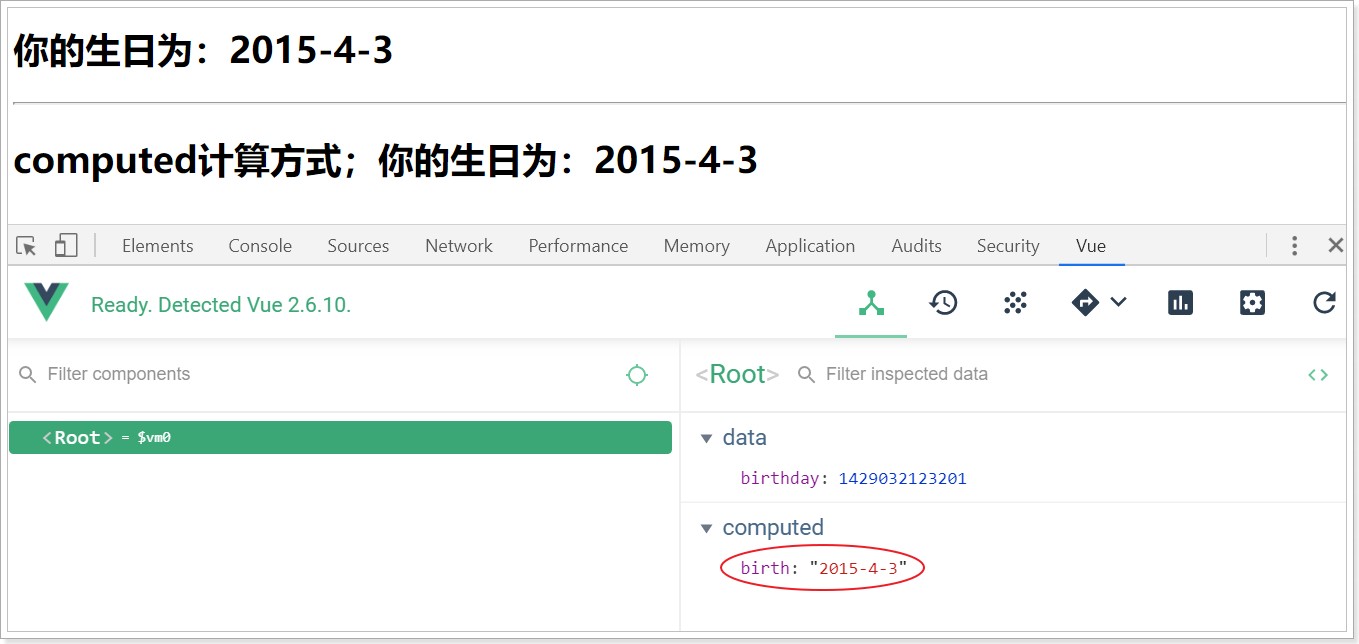
34

</script>

</body>

</html>

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。

效果：

## 4.8.watch

### 监控

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：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. </head>
7. <body>
8. <div id="app">
9. <input v-model="message">
10. </div>
11. <script src="node\_modules/vue/dist/vue.js"></script>
12. <script>
13. var app = new Vue({
14. el:"#app",//el即element，要渲染的页面元素
15. data: {
16. message:"hello vue"

17

18

19

20

21

22

23

24

25

26

27

},

watch:{

message(newValue, oldValue){

console.log("新值：" + newValue + "；旧值：" + oldValue);

}

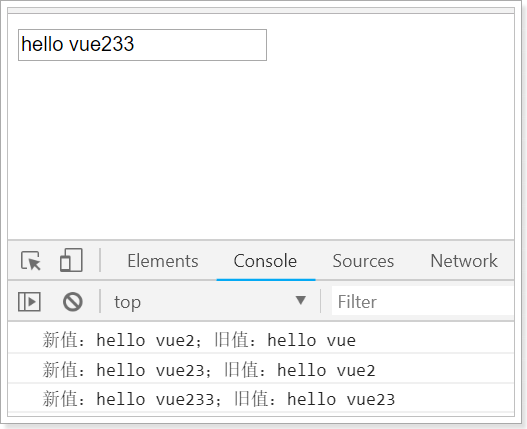
}

});

</script>

</body>

</html>

效果：

### 深度监控

如果监控的是一个对象，需要进行深度监控，才能监控到对象中属性的变化，例如：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. </head>
7. <body>
8. <div id="app">
9. <input v-model="message"/>
10. <hr><br>

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

<input v-model="person.name"><br>

<input v-model="person.age"> <button @click="person.age++">+</button>

<h2>

姓名为：{{person.name}}；年龄为：{{person.age}}

</h2>

</div>

<script src="node\_modules/vue/dist/vue.js"></script>

<script>

var app = new Vue({

el:"#app",//el即element，要渲染的页面元素data: {

message:"hello vue", person:{"name":"itcast", "age":12}

},

watch:{

message(newValue, oldValue){

console.log("新值：" + newValue + "；旧值：" + oldValue);

},

person: {

//开启深度监控，可以监控到对象属性值的变化deep: true,

//监控的处理方法

handler(obj){

console.log("name = " + obj.name + ", age=" + obj.age);

}

}

}

});

</script>

</body>

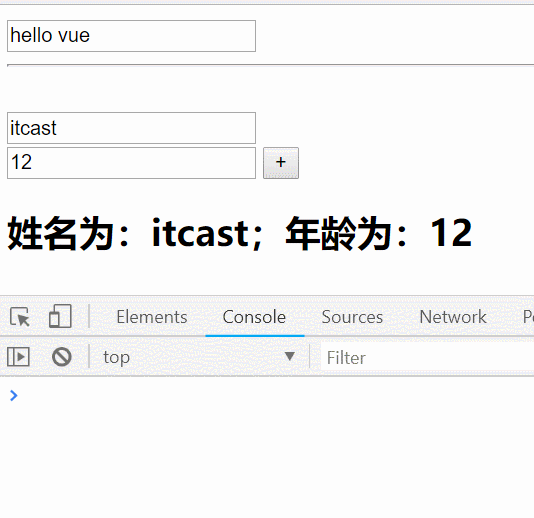
</html>

变化：

以前定义监控时，person是一个函数，现在改成了对象，并且要指定两个属性：

deep:代表深度监控，不仅监控person变化，也监控person中属性变化handler：就是以前的监控处理函数

效果：



# 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，这无疑增加了开发的成本。所以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。

## 定义全局组件

通过Vue的component方法来定义一个全局组件。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>vuejs测试</title>

</head>

<body>

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

<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>

//定义组件

const counter = {

template: "<button @click='num++'>你点击了{{num}}次；我记住了</button>", data(){

return {num: 0}

}

};

//全局注册组件；参数1：组件名称，参数2：组件

Vue.component("counter", counter);

var app = new Vue({ el:"#app"

});

</script>

</body>

</html>

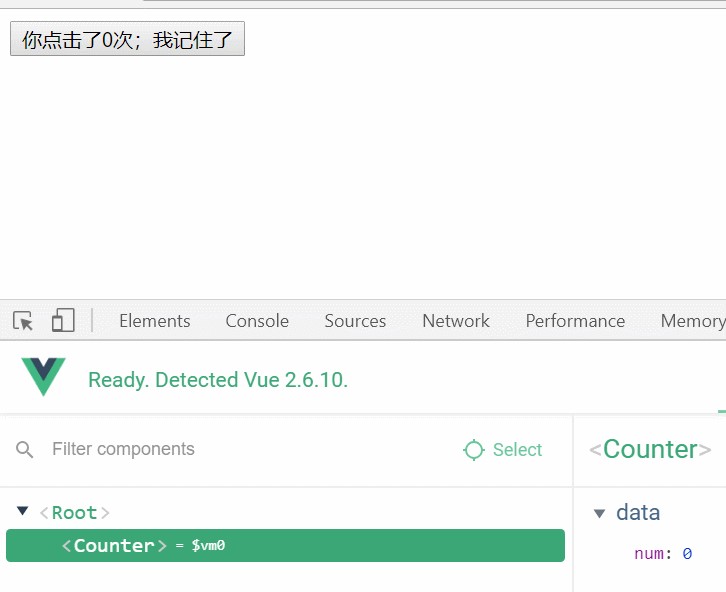
组件其实也是一个Vue实例，因此它在定义时也会接收：data、methods、生命周期函数等 不同的是组件不会与页面的元素绑定，否则就无法复用了，因此没有el属性。

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的定义方式比较特殊，必须是一个函数。

效果：

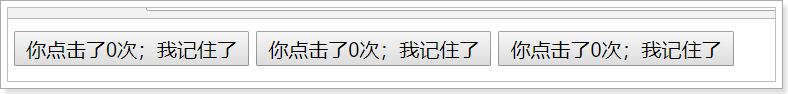


## 组件的复用

定义好的组件，可以任意复用多次：

1. <div id="app">
2. <!--使用定义好的全局组件-->
3. <counter></counter>
4. <counter></counter>
5. <counter></counter> 6 </div>

效果：



你会发现每个组件互不干扰，都有自己的num值。怎么实现的？

##### 组件的data属性必须是函数！

当定义这个 组件时，它的data 并不是像这样直接提供一个对象：

<counter>

1. data: {
2. num: 0 3 }

4

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：

1. data: function () {
2. return {
3. num: 0 4 }

5 }

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！

## 局部注册

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，会采用局部注册。

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： 然后在Vue中使用它：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. </head>
7. <body>
8. <div id="app">
9. <!--使用定义好的全局组件-->
10. <counter></counter>
11. <counter></counter>
12. <counter></counter>
13. </div>
14. <script src="node\_modules/vue/dist/vue.js"></script>
15. <script>
16. //定义组件
17. const counter = {
18. template: "<button @click='num++'>你点击了{{num}}次；我记住了</button>",
19. data(){
20. return {num: 0} 21 }

22 };

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

//全局注册组件；参数1：组件名称，参数2：组件

//Vue.component("counter", counter);

var app = new Vue({

el:"#app",

//局部注册组件components:{

counter: counter

}

});

</script>

</body>

</html>

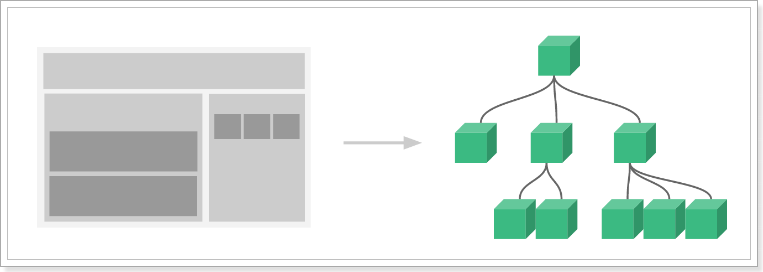
components就是当前vue对象子组件集合。其key就是子组件名称

其值就是组件对象的属性

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

## 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。

### 父向子传递props

比如有一个子组件：

1. Vue.component("introduce",{
2. // 直接使用props接收到的属性来渲染页面
3. template:'<h3>{{title}}</h3>',
4. props:[title] // 通过props来接收一个父组件传递的属性

5 })

这个子组件中要使用title属性渲染页面，但是自己并没有title属性 通过props来接收父组件属性，名为title

父组件使用子组件，同时传递title属性：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>vuejs测试</title>

</head>

<body>

<div id="app">

<!--使用定义好的全局组件-->

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

</div>

<script src="node\_modules/vue/dist/vue.js"></script>

<script>

//定义组件

const introduce = {

//使用props属性title的值渲染模版template: "<h2>{{title}}</h2>",

//定义接收来自父组件的属性

props:["title"]

};

//全局注册组件；参数1：组件名称，参数2：组件

Vue.component("introduce", introduce);

var app = new Vue({ el:"#app", data:{

msg: "父组件中的msg属性的内容"

}

});

</script>

</body>

</html>

效果：



### 传递复杂数据

定义一个子组件：

1. const myList = {
2. template:'\ 3 <ul>\

4 <li v-for="item in items" :key="item.id">{{item.id}} : {{item.name}}</li>\ 5 </ul>\

6 ',

1. props:{ // 通过props来接收父组件传递来的属性
2. items:{// 这里定义items属性
3. type:Array,// 要求必须是Array类型
4. default:[] // 如果父组件没有传，那么给定默认值是[]

11 }

12 }

13 }

这个子组件可以对 items 进行迭代，并输出到页面。但是组件中并未定义items属性。

通过props来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，否则报错；type的值可以是Array或者Object（传递对象的 时候使用）

default： 默 认 值 ， default，如果是对象则需要写成方法的方式返回默认值。如： default(){

return {"xxx":"默认值"};

}

页面内容：

* + - 1. <!DOCTYPE html>
      2. <html lang="en">
      3. <head>
      4. <meta charset="UTF-8">
      5. <title>vuejs测试</title>
      6. </head>
      7. <body>
      8. <div id="app">
      9. <h2>传智播客开设的课程有：</h2>
      10. <!-- 接受来自父组件的属性值，使用v-bind指向父组件的属性lessons；注意使用my-list -->
      11. <my-list :items="lessons"></my-list>
      12. </div>
      13. <script src="node\_modules/vue/dist/vue.js"></script>
      14. <script>
      15. //定义组件
      16. const myList = {
      17. //可以使用双引号、单引号或者如下使用的 ` 飘号
      18. template: `
      19. <ul>
      20. <li v-for="item in items" :key="item.id">{{item.id}}--{{item.name}}</li> 21 </ul>

22 `,

1. //定义接收来自父组件的属性
2. props: {
3. //定义模版中使用的属性
4. items: {
5. //必须为数组类型
6. type: Array,
7. //默认为空数组
8. default: []

31 }

32 }

33 };

34

1. var app = new Vue({
2. el: "#app",
3. data: {
4. msg: "父组件中的msg属性的内容",
5. lessons:[

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

{"id":1, "name":"Java"},

{"id":2, "name":"PHP"},

{"id":3, "name":"前端"}

]

},

//注册组件components: {

//如果组件key和value一致可以简写如下

myList

}

});

</script>

</body>

</html>

效果：



### 子向父的通信

来看这样的一个案例：

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. </head>
7. <body>
8. <div id="app">

9 <h2>num = {{num}}</h2>

1. <!--使用定义好的全局组件-->
2. <counter :snum="num"></counter>
3. </div>
4. <script src="node\_modules/vue/dist/vue.js"></script>
5. <script>

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

//定义组件

const counter = {

//组件只能是一个元素里面包裹其他元素；如下面，一个div包含两个按钮template: `

<div>

<button @click="snum++">+</button>

<button @click="snum--">-</button>

</div>

`, props:["snum"]

};

//全局注册组件；参数1：组件名称，参数2：组件

Vue.component("counter", counter);

var app = new Vue({ el:"#app", data:{

num:0

}

});

</script>

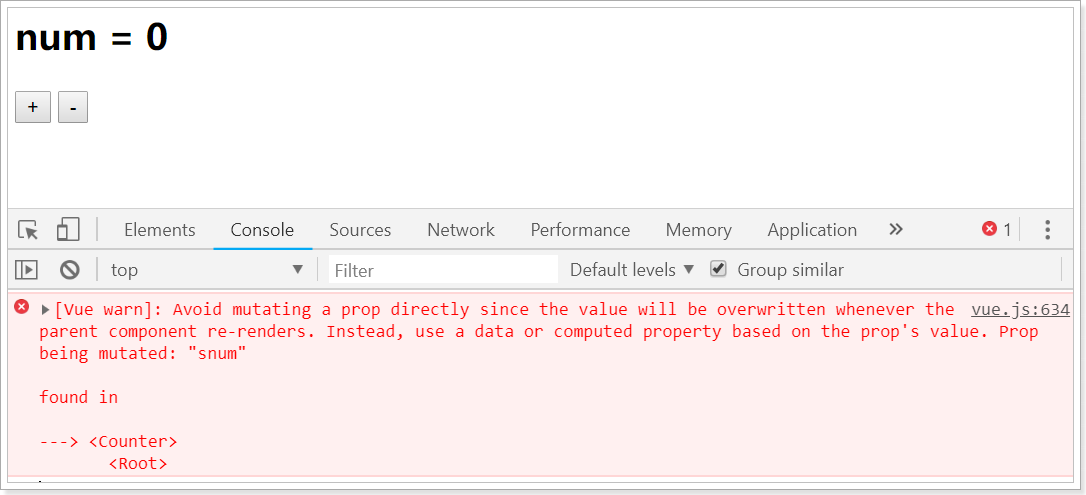
</body>

</html>

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

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：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

var app = new Vue({ el:"#app", data:{

num:0

},

methods:{

//父组件中定义操作num的方法numPlus(){

this.num++;

},

numReduce(){

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？ 可以**通过v-on指令将父组件的函数绑定到子组件**上：

1 <div id="app">

2 <h2>num = {{num}}</h2>

1. <!--使用定义好的全局组件-->
2. <counter @plus="numPlus" @reduce="numReduce" :snum="num"></counter> 5 </div>

6

然后，当子组件中按钮被点击时，调用绑定的函数：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

//定义组件

const counter = {

//组件只能是一个元素里面包裹其他元素；如下面，一个div包含两个按钮template: `

<div>

<button @click="incrNum">+</button>

<button @click="decrNum">-</button>

</div>

`, props:["snum"], methods: {

//点击模板中使用的方法incrNum(){

return this.$emit("plus");

},

decrNum(){

return this.$emit("reduce");

}

}

};

完成页面如下：

* + - 1. <!DOCTYPE html>
      2. <html lang="en">
      3. <head>
      4. <meta charset="UTF-8">
      5. <title>vuejs测试</title>
      6. </head>
      7. <body>
      8. <div id="app">

9 <h2>num = {{num}}</h2>

1. <!--使用定义好的全局组件-->
2. <counter @plus="numPlus" @reduce="numReduce" :snum="num"></counter>
3. </div>
4. <script src="node\_modules/vue/dist/vue.js"></script>
5. <script>
6. //定义组件
7. const counter = {
8. //组件只能是一个元素里面包裹其他元素；如下面，一个div包含两个按钮
9. template: `
10. <div>
11. <button @click="incrNum">+</button>
12. <button @click="decrNum">-</button>
13. </div>

23 `,

1. props:["snum"],
2. methods: {
3. //点击模板中使用的方法
4. incrNum(){
5. return this.$emit("plus"); 29 },
6. decrNum(){
7. return this.$emit("reduce"); 32 }

33 }

34 };

35

1. //全局注册组件；参数1：组件名称，参数2：组件
2. Vue.component("counter", counter); 38
3. var app = new Vue({
4. el:"#app",
5. data:{
6. num:0

43 },

1. methods:{
2. //父组件中定义操作num的方法
3. numPlus(){
4. this.num++;

48 },

1. numReduce(){
2. this.num--;

51 }

52

53

54

55

56

57

58

}

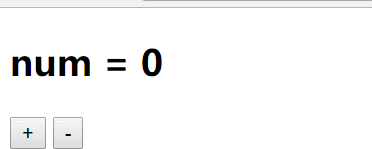
});

</script>

</body>

</html>

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：



子组件不能直接修改父组件传递参数的引用或者基本类型参数值。

# Vuejs ajax

Vuejs 并没有直接处理ajax的组件，但可以使用axios或vue-resource组件实现对异步请求的操作。

## 6.1 vue-resource

vue-resource是Vue.js的插件提供了使用XMLHttpRequest或JSONP进行Web请求和处理响应的服务。 当vue更新 到

2.0之后，作者就宣告不再对vue-resource更新，而是推荐axios。

vue-resource的github地址： <https://github.com/pagekit/vue-resource>

## axios简介

Axios 是一个基于 promise 的 HTTP 库，可以用在浏览器和 node.js 中。

axios的github地址：<https://github.com/axios/axios>

1. # 如果使用npm则可以如下安装
2. npm install axios

或者也可以直接使用公共的CDN（内容分发网络）服务：

1. <!-- 开发环境版本，包含了用帮助的命令行警告 -->
2. <script src="https://unpkg.com/axios/dist/axios.min.js"></script>

## axios应用

### 方法说明

axios可以使用的方法有：

axios(conﬁg) axios.get(url[, conﬁg]) axios.delete(url[, conﬁg]) axios.head(url[, conﬁg])

axios.post(url[, data[, conﬁg]]) axios.put(url[, data[, conﬁg]]) axios.patch(url[, data[, conﬁg]])

#### conﬁg请求配置

这些是创建请求时可以用的配置选项。只有

是必需的。如果没有指定 method ，请求将默认使用

url

方法。

get

1 {

1. // `url` 是用于请求的服务器 URL
2. url: '/user',

4

1. // `method` 是创建请求时使用的方法
2. method: 'get', // 默认是 get

7

1. // `baseURL` 将自动加在 `url` 前面，除非 `url` 是一个绝对 URL。
2. // 它可以通过设置一个 `baseURL` 便于为 axios 实例的方法传递相对 URL
3. baseURL: 'https://some-domain.com/api/',

11

1. // `transformRequest` 允许在向服务器发送前，修改请求数据
2. // 只能用在 'PUT', 'POST' 和 'PATCH' 这几个请求方法
3. // 后面数组中的函数必须返回一个字符串，或 ArrayBuffer，或 Stream
4. transformRequest: [function (data) {
5. // 对 data 进行任意转换处理

17

18 return data; 19 }],

20

1. // `transformResponse` 在传递给 then/catch 前，允许修改响应数据
2. transformResponse: [function (data) {
3. // 对 data 进行任意转换处理

24

25 return data; 26 }],

27

1. // `headers` 是即将被发送的自定义请求头
2. headers: {
3. 'X-Requested-With': 'XMLHttpRequest',

31

32 },

33

'Content-Type': 'application/json'

1. // `params` 是即将与请求一起发送的 URL 参数
2. // 必须是一个无格式对象(plain object)或 URLSearchParams 对象
3. params: { 37 ID: 12345 38 },

39

1. // `data` 是作为请求主体被发送的数据
2. // 只适用于这些请求方法 'PUT', 'POST', 和 'PATCH'
3. // 在没有设置 `transformRequest` 时，必须是以下类型之一：
4. // - string, plain object, ArrayBuffer, ArrayBufferView, URLSearchParams
5. // - 浏览器专属：FormData, File, Blob
6. // - Node 专属： Stream
7. data: {
8. firstName: 'Fred' 48 },

49

1. // `timeout` 指定请求超时的毫秒数(0 表示无超时时间)
2. // 如果请求话费了超过 `timeout` 的时间，请求将被中断
3. timeout: 1000,

53

1. // `withCredentials` 表示跨域请求时是否需要使用凭证
2. withCredentials: false, // 默认的

56

1. // `responseType` 表示服务器响应的数据类型，可以是 'arraybuffer', 'blob', 'document', 'json', 'text', 'stream'
2. responseType: 'json', // 默认的

59

1. // `maxContentLength` 定义允许的响应内容的最大尺寸
2. maxContentLength: 2000,

62

1. // `validateStatus` 定义对于给定的HTTP 响应状态码是 resolve 或 reject promise 。如果

`validateStatus` 返回 `true` (或者设置为 `null` 或 `undefined`)，promise 将被 resolve; 否则，promise 将被 rejecte

1. validateStatus: function (status) {
2. return status >= 200 && status < 300; // 默认的

66 },

67

1. // `maxRedirects` 定义在 node.js 中 follow 的最大重定向数目
2. // 如果设置为0，将不会 follow 任何重定向
3. maxRedirects: 5 // 默认的

71

72 }

#### 响应结构

1 {

2

3

4

5

// `data` 由服务器提供的响应

data: {},

// `status` 来自服务器响应的 HTTP 状态码

6

7

8

9

10

11

12

13

14

15

16

17

status: 200,

// `statusText` 来自服务器响应的 HTTP 状态信息

statusText: 'OK',

// `headers` 服务器响应的头

headers: {},

// `config` 是为请求提供的配置信息

config: {}

}

使用 时，你将接收下面这样的响应：

then

1. axios.get('/user/12345')
2. .then(function(response) {
3. console.log(response.data);
4. console.log(response.status);
5. console.log(response.statusText);
6. console.log(response.headers);
7. console.log(response.config); 8 });

在使用

catch

时，或传递 [rejection callback](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then) 作为

的第二个参数时，响应可以通过

对象可被使用。

### axios方法示例

then

error

注意：如果使用axios访问跨域数据的时候，只需要在服务提供方中，在方法上面使用SpringMVC的跨域注解即 可解决数据跨域问题。如在方法上添加：@CrossOrigin(origins = "http://localhost:10000")

如果请求的地址是使用了网关，那么在网关服务器上配置跨域就可以了；不能同时在网关服务器和服务提供服 务工程中同时配置。

可以通过向

axios

##### axios(conﬁg)

传递相关配置来创建请求

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>vuejs测试</title>
6. <script src="js/vue-2.6.10.js"></script>
7. <script src="js/axios.min.js"></script>
8. </head>
9. <body>
10. <div id="app">

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

<ul>

<li v-for="(user,index) in users" :key="index">

{{index}}--{{user.name}}--{{user.age}}--{{user.gender}}

</li>

</ul>

</div>

<script>

var app = new Vue({ el:"#app", data: {

users:[]

},

created(){

//加载数据axios({

method:"get", url: "data.json"

}).then((res)=>{

console.log(res);

//将获取数据设置到users属性app.users = res.data;

}).catch(error=>{ alert(error);

});

}

});

</script>

</body>

</html>

### get方法示例

将上述示例中的axios操作部分修改为如下：

1

2

3

4

5

6

7

8

axios.get("data.json")

.then( res => { console.log(res);

//将获取数据设置到users属性app.users = res.data;

}).catch(error =>{ console.log(error)

});

### post方法示例

将示例中的axios操作部分修改为如下：

1

2

3

4

5

6

7

8

9

10

axios.post("data.json")

.then( res => { console.log(res);

//将获取数据设置到users属性app.users = res.data;

}).catch(error =>{ console.log(error)

});

}