

0. 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

1. Vue概述

1.1. 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：

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

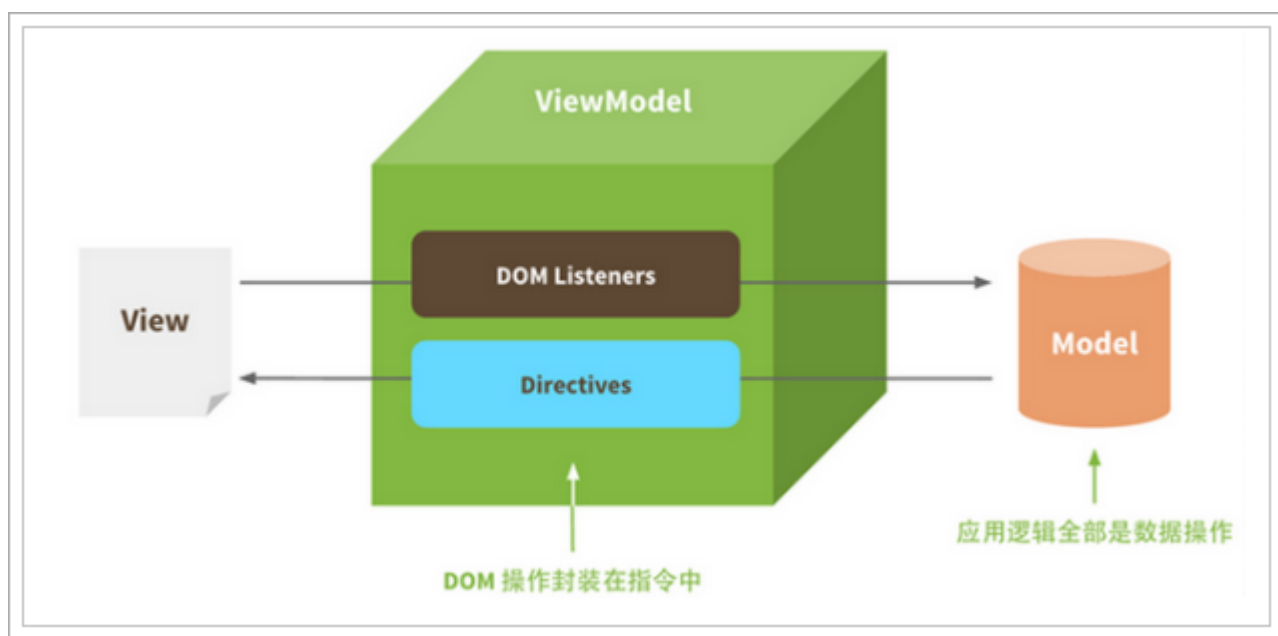
- M：即Model，模型，包括数据和一些基本操作
- V：即View，视图，页面渲染结果
- VM：即View-Model，模型与视图间的双向操作（无需开发人员干涉）

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

1.2. 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。

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/>



The image is a promotional banner for Vue.js. On the left is the large green and dark blue Vue logo. To its right, the text '渐进式 JavaScript 框架' (Progressive JavaScript Framework) is displayed in a large, dark blue font. Below this title are three buttons: a green button with a play icon and the text 'WHY VUE.JS?', a green button with the text '起步' (Getting Started), and a light gray button with the GitHub logo and the text 'GITHUB'. Further down, the section 'Special Sponsors' is shown, featuring logos for 'stdlib' (with the tagline 'Function as a Service Platform and Library') and 'bit' (with the tagline 'The fastest way to share code'). At the bottom, three key features are highlighted in green: '易用' (Easy to Use) with the text '已经会了 HTML、CSS、JavaScript? 即刻阅读指南开始构建应用!', '灵活' (Flexible) with the text '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。', and '高效' (Efficient) with the text '20kB min+gzip 运行大小, 超快虚拟 DOM, 最省心的优化'.

渐进式
JavaScript 框架

WHY VUE.JS? 起步 GITHUB

Special Sponsors

stdlib Function as a Service Platform and Library

bit The fastest way to share code

易用 灵活 高效

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


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。

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

 Overview Repositories 141 Stars 779 Followers 29.2k Following 90

Evan You
yyx990803
Creator of @vuejs, previously @meteor & @google
Follow
Block or report user
New Jersey / China
<http://evanyou.me>

Pinned repositories

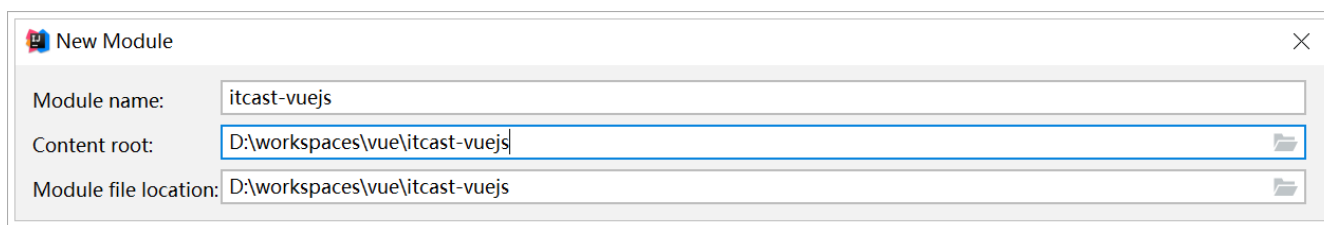
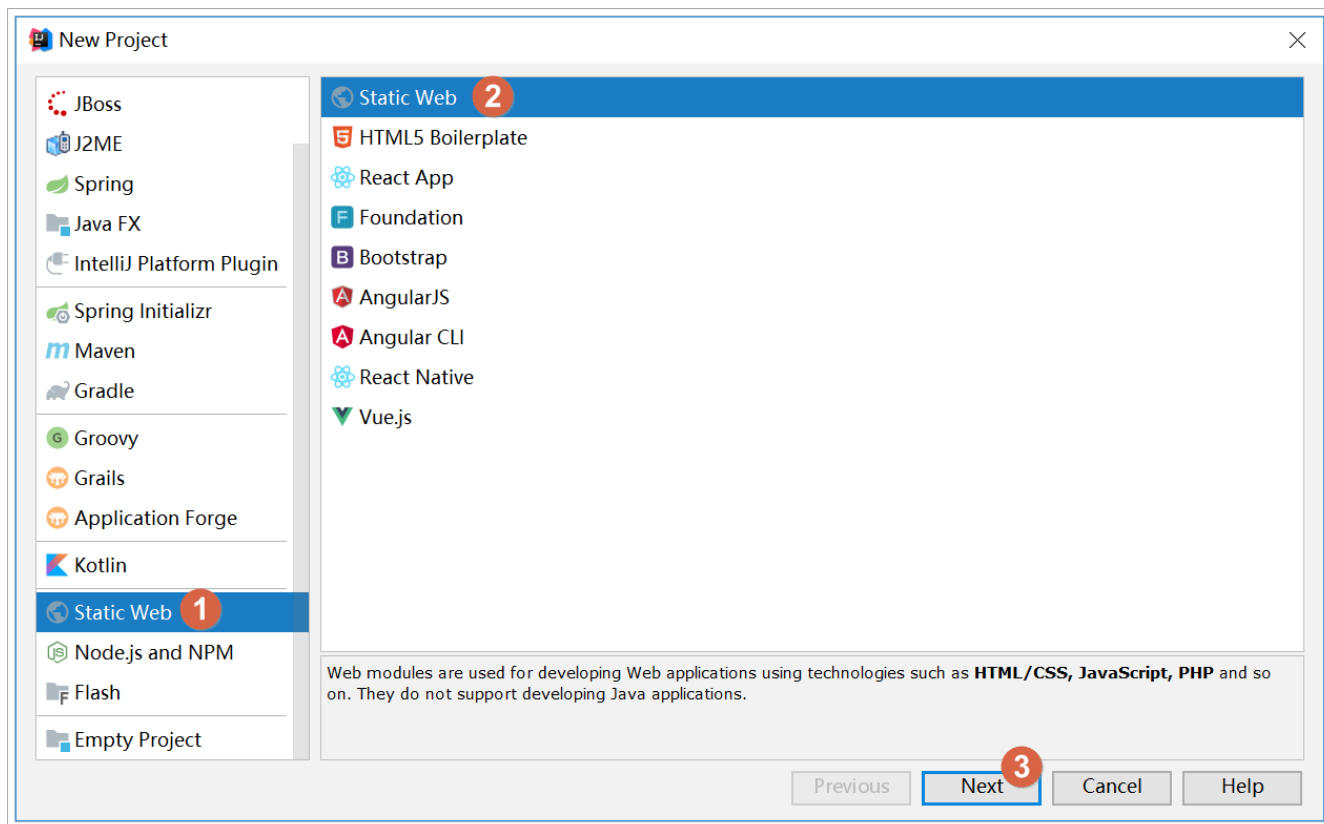
- vuejs/vue**
A progressive, incrementally-adoptable JavaScript framework for building UI on the web.
JavaScript 88.7k 13k
- vuejs/vue-router**
The official router for Vue.js.
JavaScript 9.3k 2.4k
- vuejs/vuex**
Centralized State Management for Vue.js.
JavaScript 13.5k 4.2k
- vuejs/vue-cli**
CLI for 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. 快速入门

2.1. 创建工程

创建一个新的工程；选择静态web类型工程：



2.2. 安装vue

2.2.1. 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。

2.2.2. 使用CDN

或者也可以直接使用公共的CDN (内容分发网络) 服务:

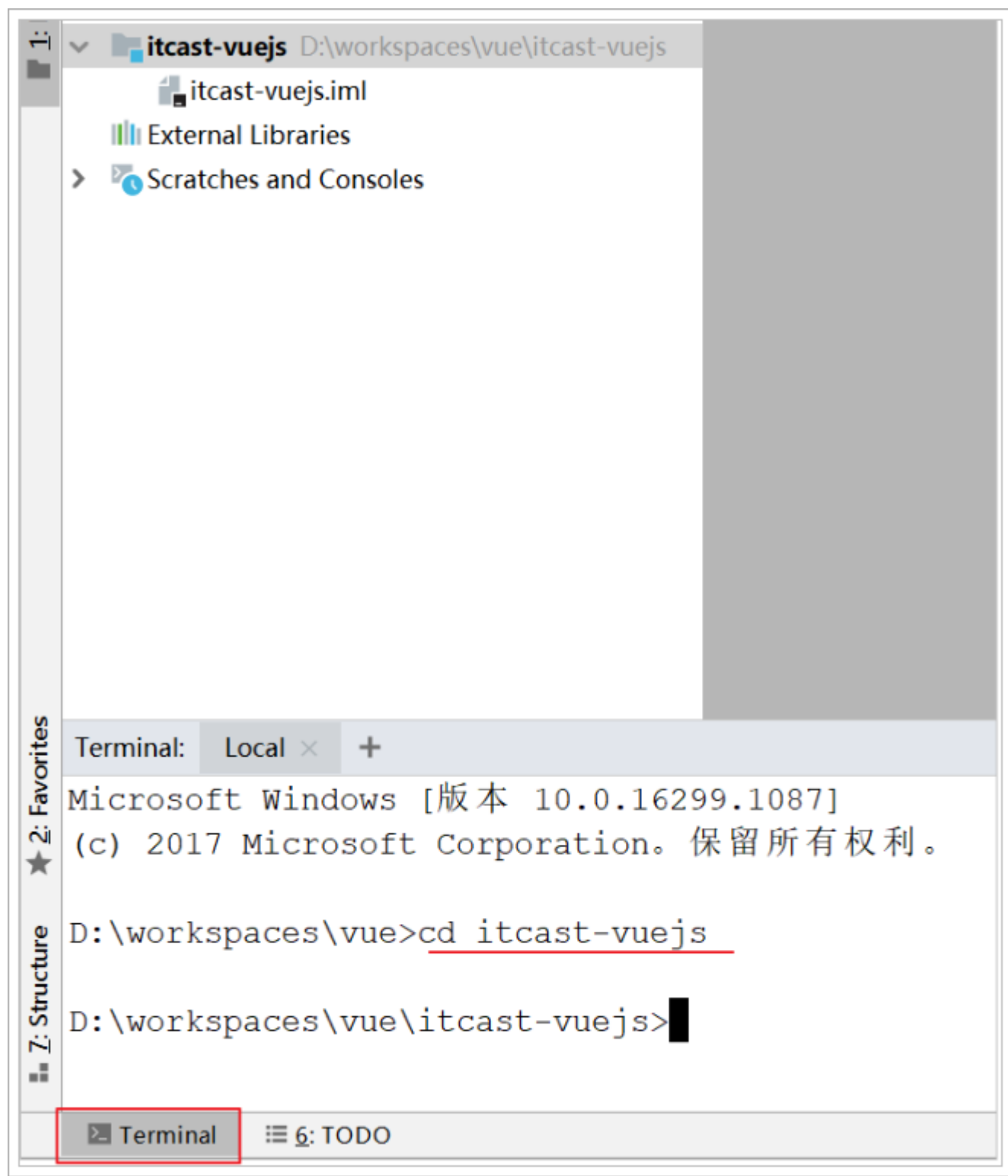
```
1 <!-- 开发环境版本, 包含了用帮助的命令警告 -->
2 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

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>
```

2.2.3. 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

```
Terminal: Local x +
D:\workspaces\vue\itcast-vuejs>npm init -y
Wrote to D:\workspaces\vue\itcast-vuejs\package.json:

{
  "name": "itcast-vuejs",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

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：

```
npm WARN registry Unexpected warning for https://registry.npm.taobao.org/: Miscellaneous Warning ERR_STREAM_DESTROYED: Cannot call
write after a stream was destroyed
npm WARN registry Using stale package data from https://registry.npm.taobao.org/ due to a request error during revalidation.
Unhandled rejection Error: EPERM: operation not permitted, open 'C:\Program Files\nodejs\npm_cache\_cacache\tmp\776a17ad'
```

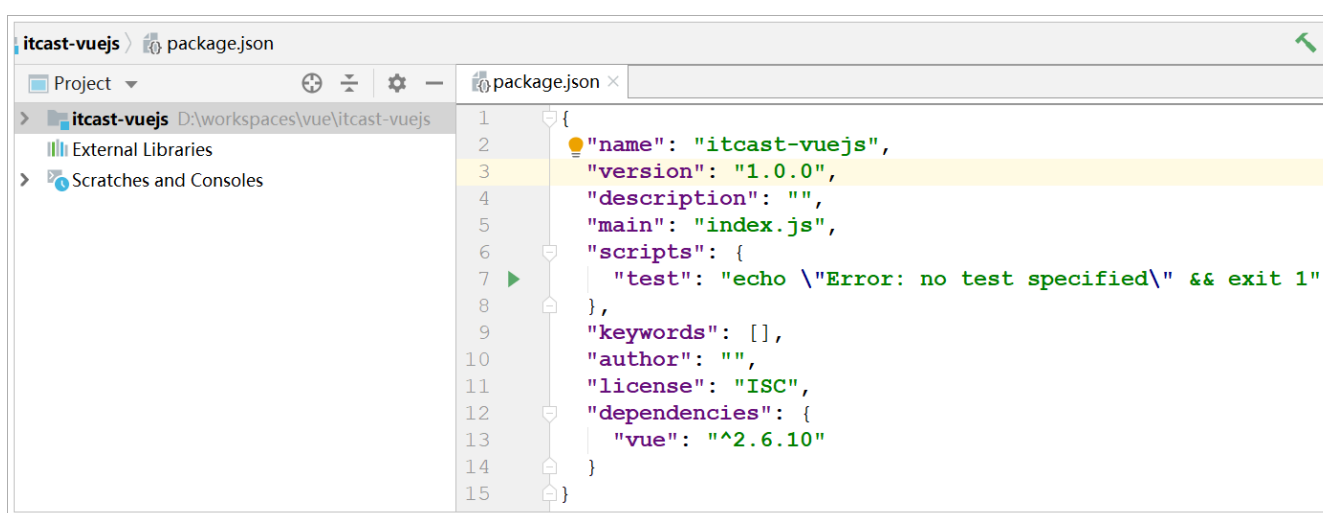
```
管理员: Windows PowerShell

PS D:\workspaces\vue\itcast-vuejs> npm install vue --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN itcast-vuejs@1.0.0 No description
npm WARN itcast-vuejs@1.0.0 No repository field.

+ vue@2.6.10
added 1 package from 1 contributor and audited 1 package in 1.943s
found 0 vulnerabilities

PS D:\workspaces\vue\itcast-vuejs>
```

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。

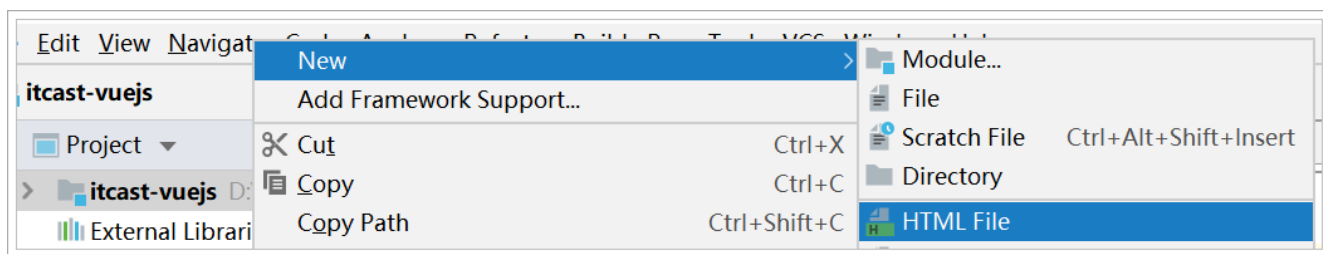
此时再查看package.json，会发现有了变化：

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？

2.3. vue入门案例

2.3.1. HTML模板

在项目目录新建一个HTML文件 01-demo.html



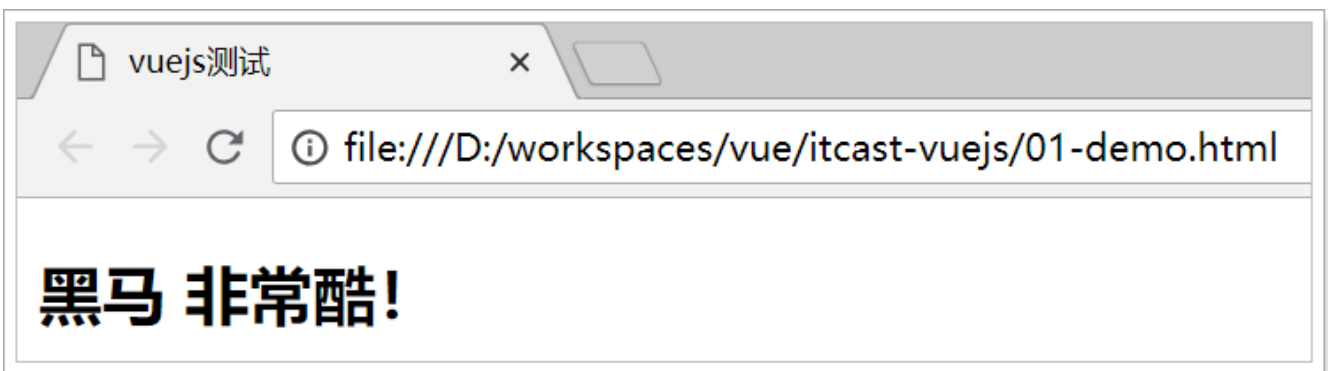
2.3.2. vue渲染

01-demo.html内容如下:

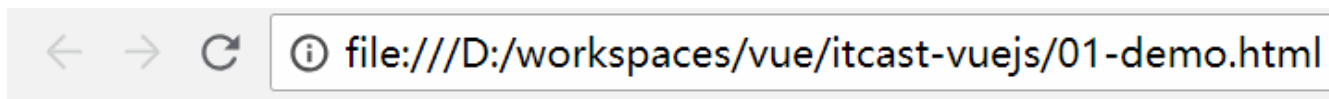
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>vuejs测试</title>
6 </head>
7 <body>
8   <div id="app">
9     <h2>{{name}} 非常酷! </h2>
10  </div>
11  <script src="node_modules/vue/dist/vue.js"></script>
12  <script>
13    var app = new Vue({
14      el: "#app", //el即element, 要渲染的页面元素
15      data: {
16        name: "黑马"
17      }
18    });
19  </script>
20
21 </body>
22 </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。

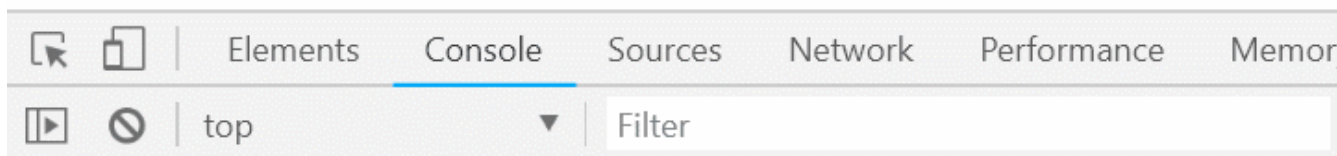
打开页面查看效果:



更神奇的在于, 当你修改name属性时, 页面会跟着变化:



黑马 非常酷!



2.3.3. 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>
11    {{name}} 非常酷!
12    有{{num}}个酷炫学科。
13  </h2>
14 </div>
15 <script src="node_modules/vue/dist/vue.js"></script>
16 <script>
17   var app = new Vue({
18     el: "#app", //el即element, 要渲染的页面元素
19     data: {
20       name: "黑马",
```

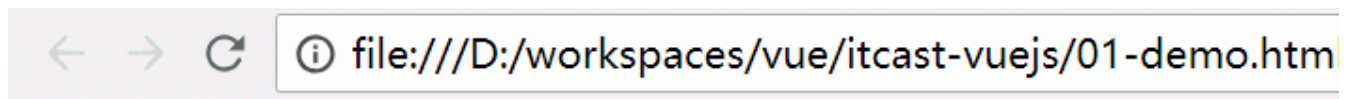
```

21         num: 1
22     }
23 });
24 </script>
25
26 </body>
27 </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:



1

黑马 非常酷! 有1个酷炫学科。

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。

2.3.4. 事件处理

在页面添加一个按钮:

```

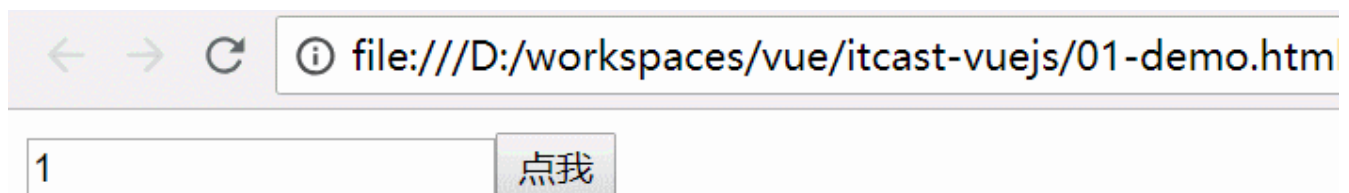
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>vuejs测试</title>
6  </head>
7  <body>
8      <div id="app">
9          <input type="text" v-model="num"><button v-on:click="num++">点我</button>
10         <h2>
11             {{name}} 非常酷!
12             有{{num}}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

```

```
17   var app = new Vue({
18     el: "#app", //el即element, 要渲染的页面元素
19     data: {
20       name: "黑马",
21       num: 1
22     }
23   });
24 </script>
25
26 </body>
27 </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：



黑马 非常酷！ 有1个酷炫学科。

3. Vue实例

3.1. 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：

```
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。

3.2. 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；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。

3.3. 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：

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

js:

```
1 var vm = new Vue({
2   el: "#app",
3   data: {
4     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

3.4. 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   },
5   methods:{
6     add:function(){
7       console.log("点我了...233")
8     }
9   }
10 })
```

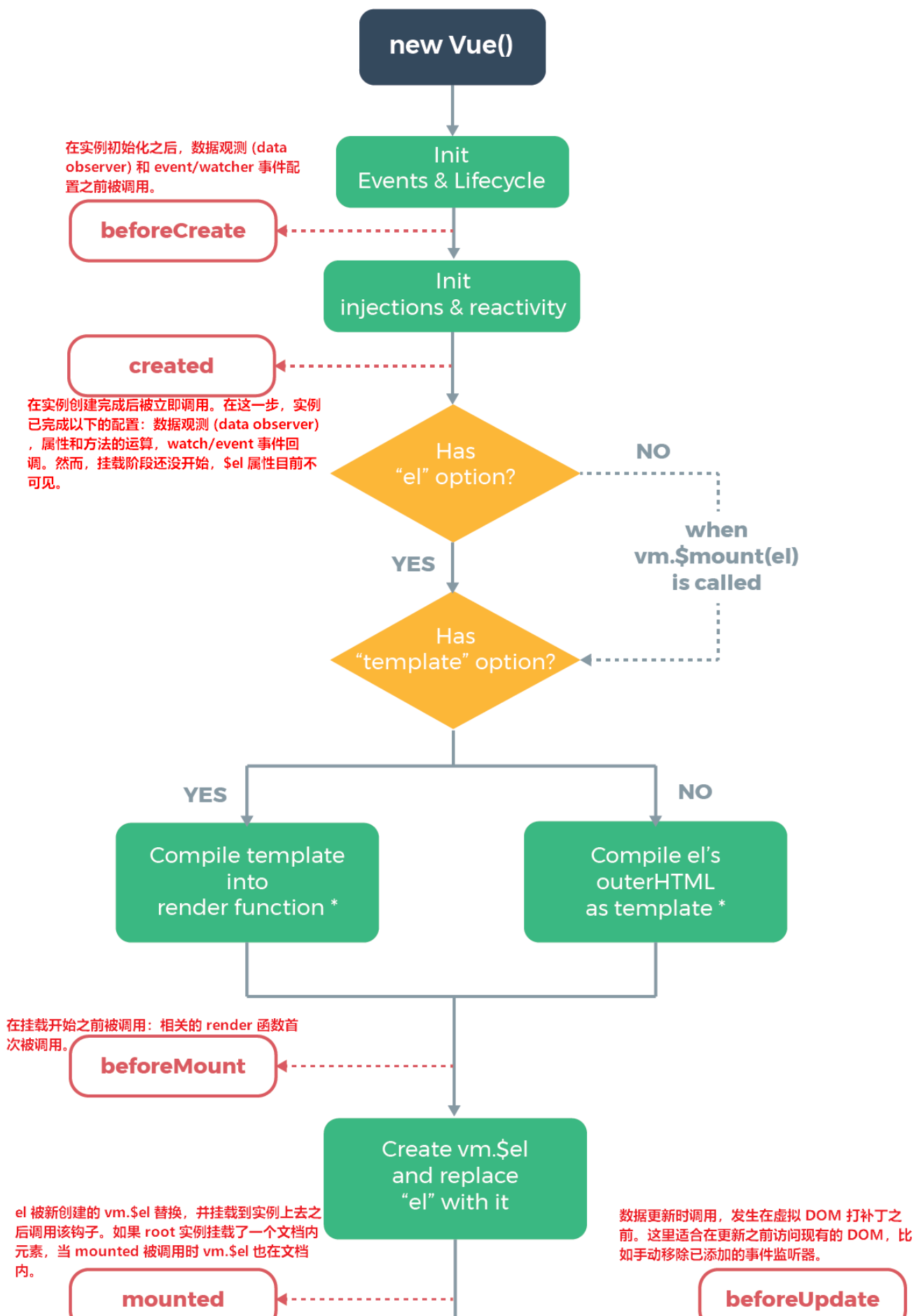
3.5. 生命周期钩子

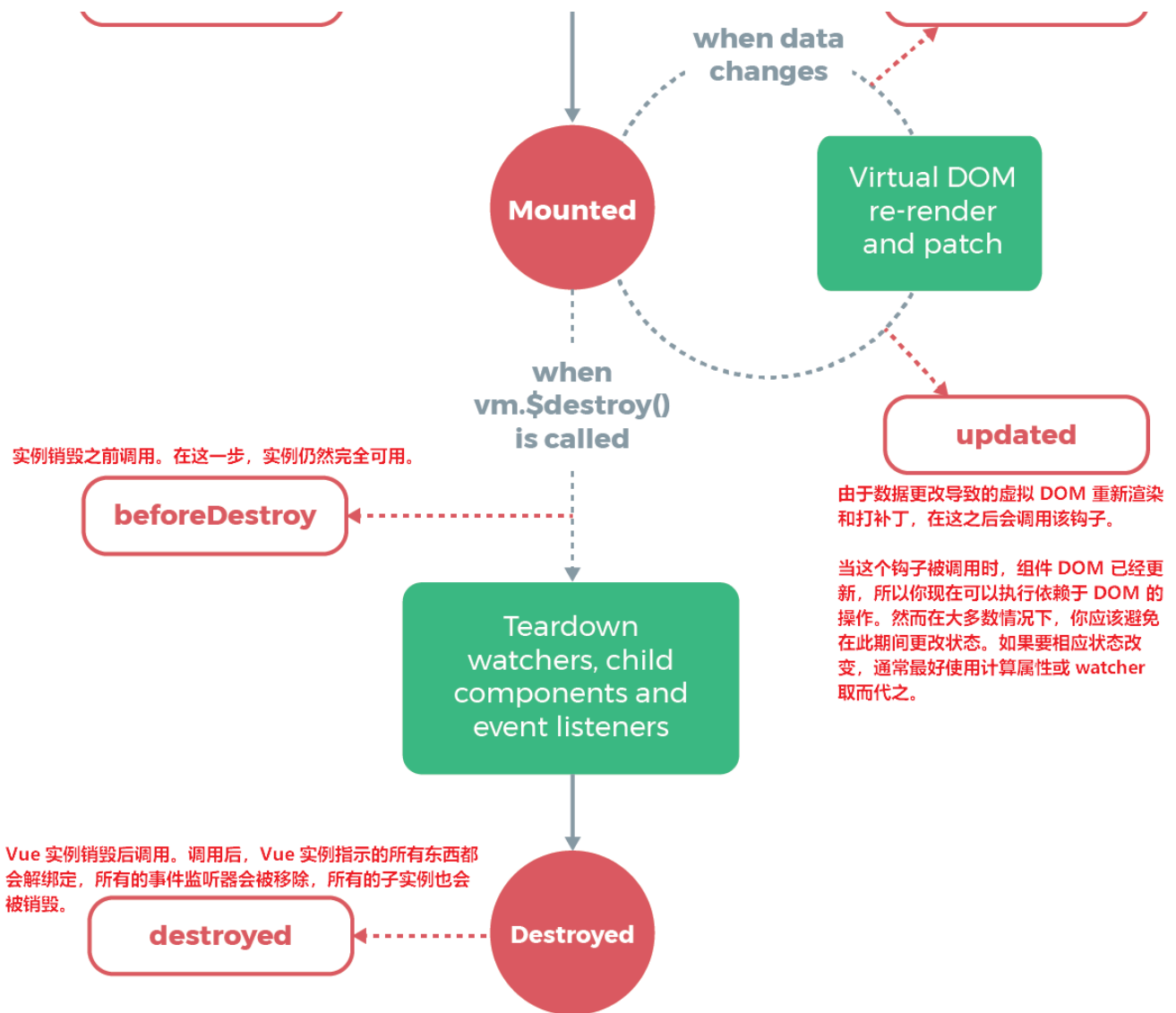
3.5.1. 生命周期

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。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 `this` 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()`）。这是因为箭头函数绑定了父上下文，因此 `this` 与你期待的 Vue 实例不同，`this.fetchTodos` 的行为未定义。

生命周期：





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

如果使用构建步骤，模板编译会提前执行；例如单文件组件

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。

3.5.2.钩子函数

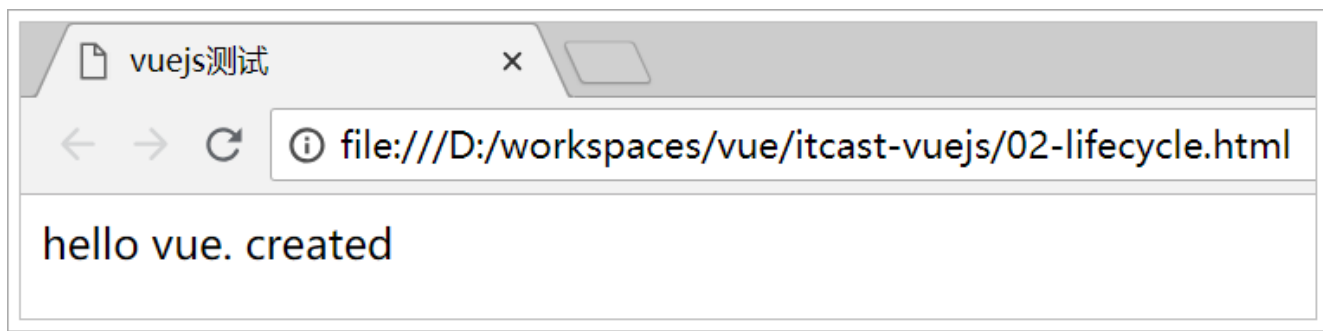
例如：created代表在vue实例创建后；

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}}
11   </div>
12   <script>
13     let app = new Vue({
14       el:"#app",
15       data:{
16         //初始化为空
17         msg:""
18       },
19       created(){
20         this.msg = "hello vue. created";
21       }
22     });
23   </script>
24 </body>
25 </html>
```

结果:



3.5.3.this

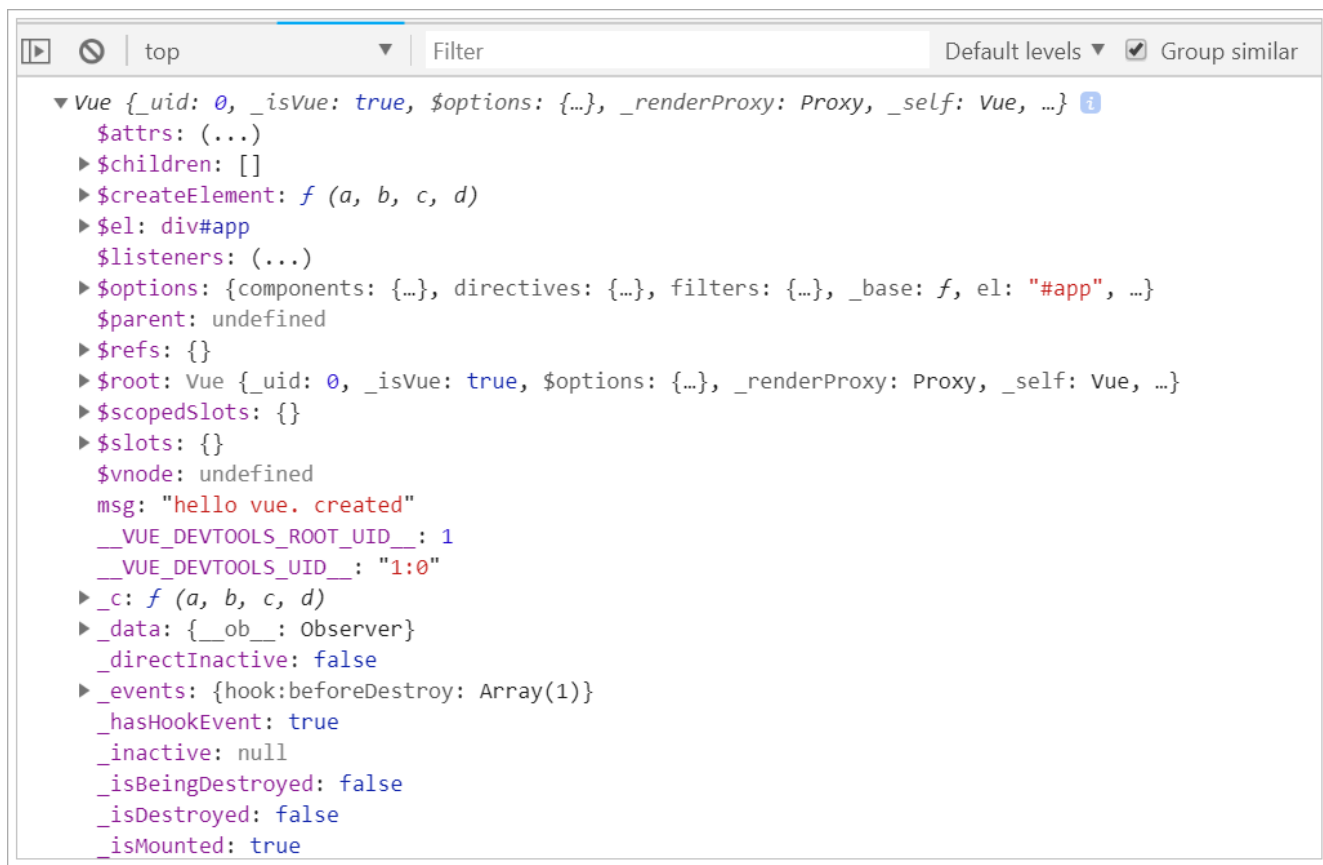
可以看下在vue内部的this变量是谁, 在created的时候, 打印this

```

1  let vm = new Vue({
2    el:"#app",
3    data:{
4      //初始化为空
5      msg:""
6    },
7    created(){
8      this.msg = "hello vue. created";
9      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。

4. 指令

什么是指令？

指令 (Directives) 是带有 `v-` 前缀的特殊属性。例如在入门案例中的v-model，代表双向绑定。

4.1. 插值表达式

4.1.1. 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}}
11 |   </div>
12 |   <script>
13 |     let app = new Vue({
14 |       el: "#app",
15 |       data: {
16 |         msg: "hello vue"
17 |       }
18 |     });
19 |   </script>
20 | </body>
21 | </html>
```

4.1.2. 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，称为插值闪烁。类似如下的效果（最新vue是几乎没有此问题）：



4.1.3. v-text和v-html

使用v-text和v-html指令来替代 {{ }}

说明：

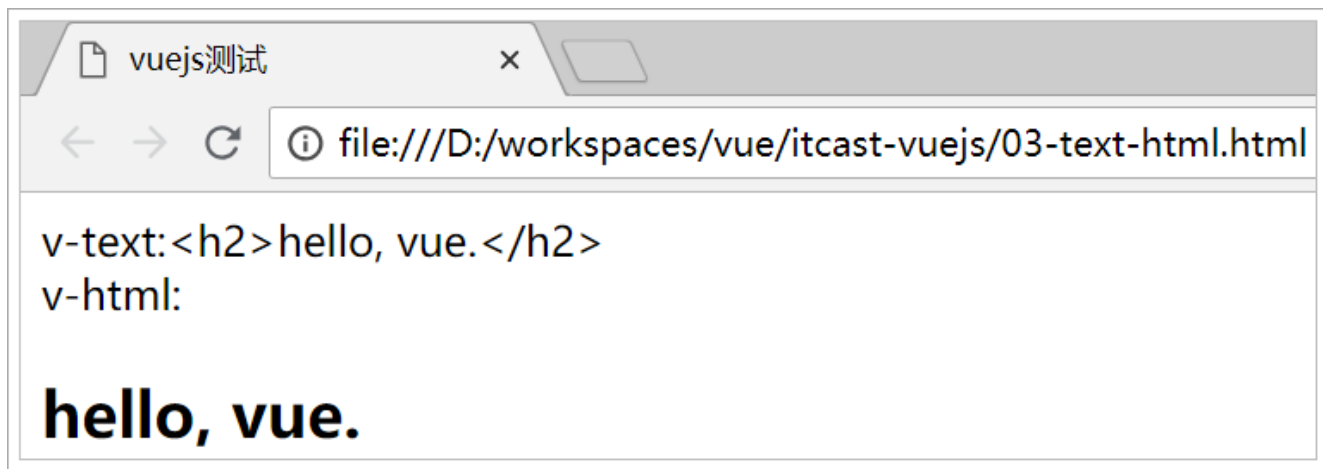
- v-text：将数据输出到元素内部，如果输出的数据有HTML代码，会作为普通文本输出
- v-html：将数据输出到元素内部，如果输出的数据有HTML代码，会被渲染

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      });
20    </script>
21  </body>
22  </html>
```

效果：



并且不会出现插值闪烁，当没有数据时，会显示空白。

4.2. 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>
```

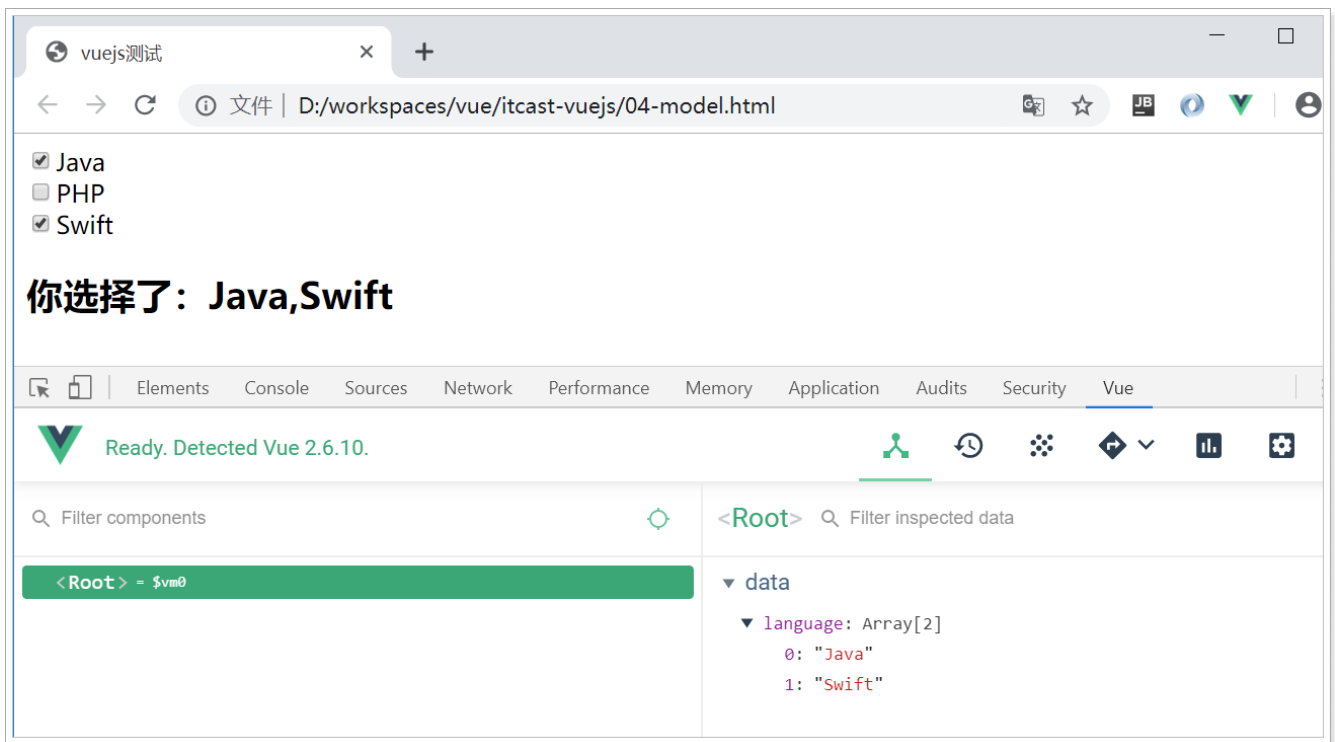
```

17 <script>
18   let app = new Vue({
19     el:"#app",
20     data:{
21       language: []
22     }
23   });
24 </script>
25 </body>
26 </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
- input 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：



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](#) 查看并安装。

4.3. v-on

4.3.1. 基本用法

v-on指令用于给页面元素绑定事件。

语法：

```
1 v-on:事件名="js片段或函数名"
```

简写语法：

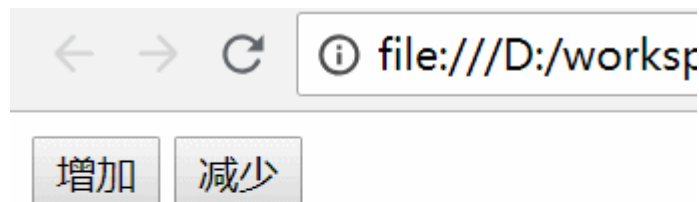
1 | @事件名="js片段或函数名"

例如 `v-on:click='add'` 可以简写为 `@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>
17 </div>
18 <script>
19   let app = new Vue({
20     el:"#app",
21     data:{
22       num:1
23     },
24     methods:{
25       decrement(){
26         this.num--;
27       }
28     }
29   });
30 </script>
31 </body>
32 </html>
```

效果:



num = 1

4.3.2. 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**。之前提过，修饰符是由点开头的指令后缀来表示的。

- `.stop`：阻止事件冒泡
- `.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 <!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>
17     <hr>
18     事件冒泡测试：<br>
19     <div style="background-color: lightblue;width: 100px;height: 100px"
20       @click="print('你点击了div')">
21       <button @click.stop="print('点击了button')">点我试试</button>
22     </div>
23     <br>阻止默认事件：<br>
24     <a href="http://www.itcast.cn" @click.prevent="print('点击超链接')">传智播客</a>
25
```



```

26 </div>
27 <script>
28   let app = new Vue({
29     el: "#app",
30     data: {
31       num: 1
32     },
33     methods: {
34       decrement() {
35         this.num--;
36       },
37       print(msg) {
38         console.log(msg)
39       }
40     }
41   });
42 </script>
43 </body>
44 </html>

```

4.4. 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。

4.4.1. 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>
12 </div>
13 <script src="node_modules/vue/dist/vue.js"></script>
14 <script>
15   var app = new Vue({

```

```

16     el:"#app", //el即element, 要渲染的页面元素
17     data: {
18         users:[
19             {"name":"黑马","age":8,"gender":"男"},
20             {"name":"传智播客","age":12,"gender":"女"},
21             {"name":"酷丁鱼","age":4,"gender":"男"}
22         ]
23     }
24 });
25 </script>
26
27 </body>
28 </html>

```

效果:

- 黑马--8--男
- 传智播客--12--女
- 酷丁鱼--4--男

4.4.2. 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>
3         <li v-for="(user, index) in users">
4             {{index}}--{{user.name}}--{{user.age}}--{{user.gender}}
5         </li>
6     </ul>
7 </div>

```

效果:

- 0--黑马--8--男
- 1--传智播客--12--女
- 2--酷丁鱼--4--男

4.4.3. 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：

```
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>
10      <li v-for="(user, index) in users" :key="index">
11        {{index}}--{{user.name}}--{{user.age}}--{{user.gender}}
12      </li>
13    </ul>
14    <hr>
15    <ul>
16      <li v-for="(value,key,index) in person">
17        {{index}}--{{key}}--{{value}}
18      </li>
19    </ul>
20  </div>
21  <script src="node_modules/vue/dist/vue.js"></script>
22  <script>
23    var app = new Vue({
24      el:"#app", //el即element, 要渲染的页面元素
25      data: {
26        users:[
27          {"name":"黑马", "age":8, "gender":"男"},
```

```

28         {"name": "传智播客", "age": 12, "gender": "女"},
29         {"name": "酷丁鱼", "age": 4, "gender": "男"}
30     ],
31     person: {"name": "传智汇", "age": 3, "address": "中国"}
32 }
33 });
34 </script>
35
36 </body>
37 </html>

```

效果：

- 0--黑马--8--男
- 1--传智播客--12--女
- 2--酷丁鱼--4--男

- 0--name--传智汇
- 1--age--3
- 2--address--中国

4.4.4. key

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。

如果使用key这个功能可以有效的提高渲染的效率；key一般使用在遍历完后，又增、减集合元素的时候更有意义。

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。也就是key是该项的唯一标识。

示例：

```

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

```

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

4.5.v-if和v-show

4.5.1. 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:

```
1 | v-if="布尔表达式"
```

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>
13 |  </div>
14 |  <script src="node_modules/vue/dist/vue.js"></script>
15 |  <script>
16 |    var app = new Vue({
17 |      el: "#app", //el即element, 要渲染的页面元素
18 |      data: {
19 |        show: true
20 |      }
21 |    });
22 |  </script>
23 |
24 | </body>
25 | </html>
```

效果:



Hello VueJS.

4.5.2. 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。

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>
13    <hr>
14    <ul>
15      <li v-for="(user,index) in users" v-if="user.gender=='女'">
16        {{index}}--{{user.name}}--{{user.age}}--{{user.gender}}
17      </li>
18    </ul>
19  </div>
20  <script src="node_modules/vue/dist/vue.js"></script>
21  <script>
22    var app = new Vue({
23      el:"#app",//el即element, 要渲染的页面元素
24      data: {
25        show:true,
26        users:[
27          {"name":"黑马","age":8,"gender":"男"},
28          {"name":"传智播客","age":12,"gender":"女"},
29          {"name":"酷丁鱼","age":4,"gender":"男"},
30          {"name":"传智大学","age":2,"gender":"女"}
31        ]
32      }
33    });
34  </script>
35
36 </body>
37 </html>
```

效果：



4.5.3. v-else

可以使用 `v-else` 指令来表示 `v-if` 的“else 块”:

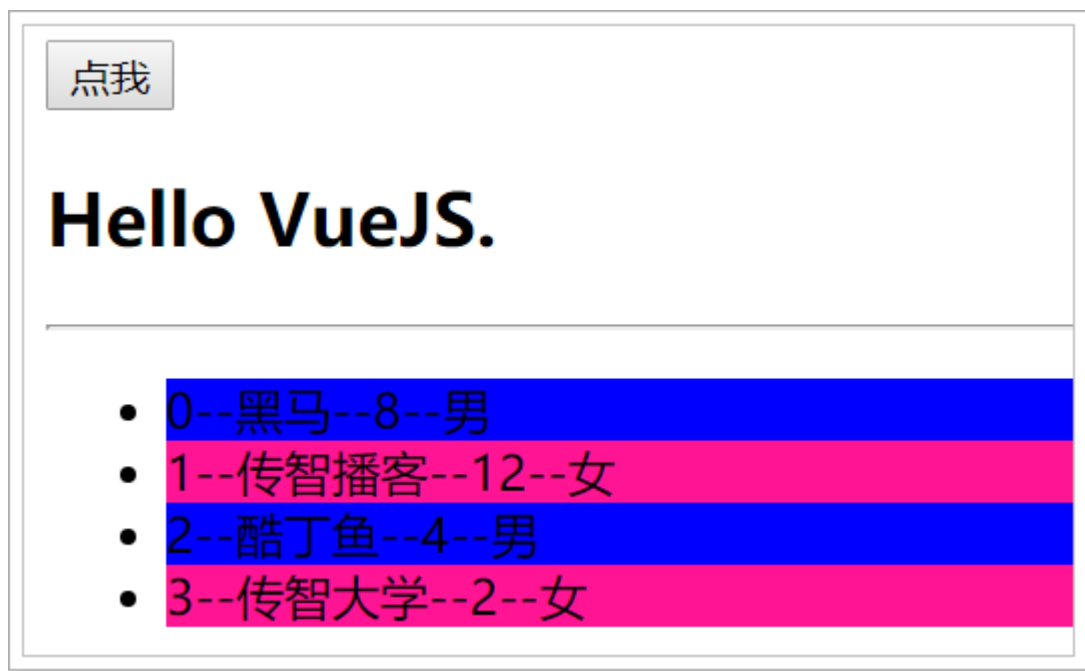
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>vuejs测试</title>
6 </head>
7 <body>
8   <div id="app">
9     <button @click="show = !show">点我</button>
10    <h2 v-if="show">
11      Hello VueJS.
12    </h2>
13    <hr>
14    <ul v-if="show">
15      <li v-for="(user,index) in users" v-if="user.gender=='女'" style="background-
16        color: deeppink">
17        {{index}}--{{user.name}}--{{user.age}}--{{user.gender}}
18      </li>
19      <li v-else style="background-color: blue">
20        {{index}}--{{user.name}}--{{user.age}}--{{user.gender}}
21      </li>
22    </ul>
23  </div>
24  <script src="node_modules/vue/dist/vue.js"></script>
25  <script>
26    var app = new Vue({
27      el:"#app",//el即element, 要渲染的页面元素
28      data: {
29        show:true,
30        users:[
31          {"name":"黑马","age":8,"gender":"男"},
32          {"name":"传智播客","age":12,"gender":"女"},
33          {"name":"酷丁鱼","age":4,"gender":"男"},
```

```

33     {"name": "传智大学", "age": 2, "gender": "女"}
34   ]
35 }
36 });
37 </script>
38
39 </body>
40 </html>

```

`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。



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

```

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 `v-if` 或者 `v-else-if` 的元素之后。

4.5.4. 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：


```
1 | <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:

```
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>
13 |    <hr>
14 |    <ul v-if="show">
15 |      <li v-for="(user,index) in users" v-if="user.gender=='女'" style="background-
16 |        color: deeppink">
17 |        {{index}}--{{user.name}}--{{user.age}}--{{user.gender}}
18 |      </li>
19 |      <li v-else style="background-color: blue">
20 |        {{index}}--{{user.name}}--{{user.age}}--{{user.gender}}
21 |      </li>
22 |    </ul>
23 |    <hr>
24 |    <h2 v-show="show">
25 |      你好; 黑马!
26 |    </h2>
27 |  </div>
28 |  <script src="node_modules/vue/dist/vue.js"></script>
29 |  <script>
30 |    var app = new Vue({
31 |      el:"#app", //el即element, 要渲染的页面元素
32 |      data: {
33 |        show:true,
34 |        users:[
35 |          {"name": "黑马", "age": 8, "gender": "男"},
36 |          {"name": "传智播客", "age": 12, "gender": "女"},
37 |          {"name": "酷丁鱼", "age": 4, "gender": "男"},
38 |          {"name": "传智大学", "age": 2, "gender": "女"}
39 |        ]
40 |      }
41 |    });
42 |  </script>
43 | </body>
44 | </html>
```

代码:

点我

Hello VueJS.

- 0--黑马--8--男
- 1--传智播客--12--女
- 2--酷丁鱼--4--男
- 3--传智大学--2--女

你好; 黑马!

```

<!doctype html>
<html lang="en">
  <head>...</head>
  <body>
    <div id="app">
      <button>点我</button>
      <h2>
        Hello VueJS.
      </h2>
      <hr>
      <ul>...</ul>
      <hr>
      <h2>
        你好; 黑马!
      </h2>
    </div>
  </body>
</html>

```

4.6. v-bind

4.6.1. 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   <title>vuejs测试</title>
6   <style type="text/css">
7     div {
8       width: 100px;
9       height: 100px;
10      color: white;
11    }
12    .red {
13      background-color: red;
14    }
15    .blue {
16      background-color: blue;
17    }
18  </style>
19 </head>
20 <body>
21 <div id="app">
22   <button @click="color='red'">红色</button>
23   <button @click="color='blue'">蓝色</button>
24   <div class="">
25     点击按钮改变背景颜色。
26   </div>
27 </div>
28 <script src="node_modules/vue/dist/vue.js"></script>
29 <script>
30   var app = new Vue({
31     el: "#app",
32     data: {
33       color: "red"
34     }
35   });
36 </script>
37
38 </body>
39 </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"
```

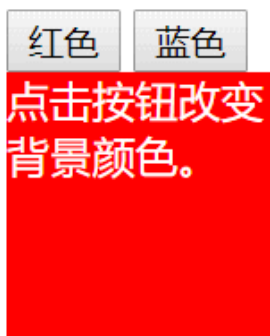
例如，在这里我们可以写成：

```
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：



4.6.2. 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 `:class` 一个对象，以动态地切换 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;
```

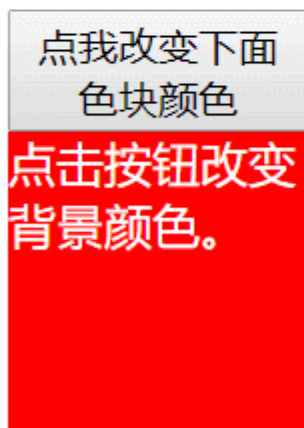
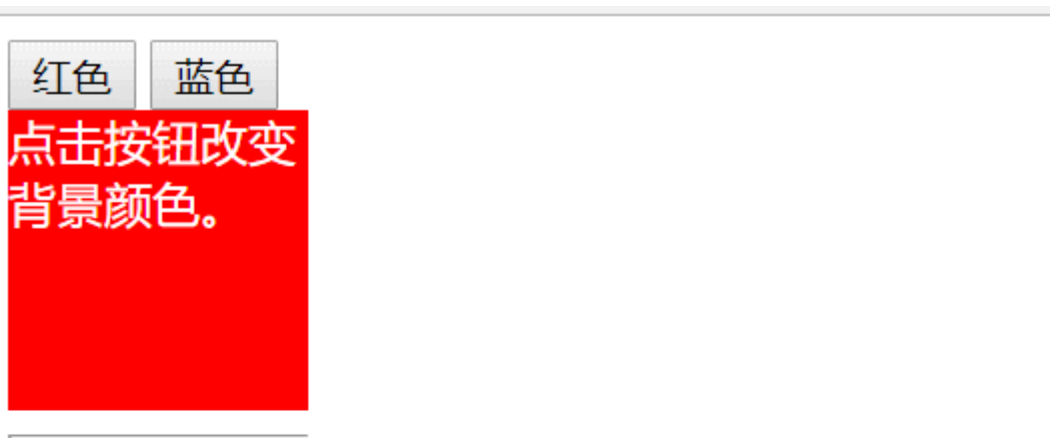
```

9         height: 100px;
10        color: white;
11    }
12    .red {
13        background-color: red;
14    }
15    .blue {
16        background-color: blue;
17    }
18    </style>
19 </head>
20 <body>
21 <div id="app">
22     <button @click="color='red'">红色</button>
23     <button @click="color='blue'">蓝色</button>
24     <div :class="color">
25         点击按钮改变背景颜色。
26     </div>
27     <hr><br>
28     <button @click="bool=!bool">点我改变下面色块颜色</button>
29     <div :class="{red:bool,blue:!bool}">
30         点击按钮改变背景颜色。
31     </div>
32 </div>
33 <script src="node_modules/vue/dist/vue.js"></script>
34 <script>
35     var app = new Vue({
36         el: "#app",
37         data: {
38             color: "red",
39             bool: true
40         }
41     });
42 </script>
43
44 </body>
45 </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：



4.7. 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：

```
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 </div>
13 <script src="node_modules/vue/dist/vue.js"></script>
14 <script>
15     var app = new Vue({
16         el:"#app",//el即element, 要渲染的页面元素
17         data: {
18             birthday:1429032123201
19         }
20     });
21 </script>
22
23 </body>
24 </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>
16         </div>
17     <script src="node_modules/vue/dist/vue.js"></script>
18     <script>
19         var app = new Vue({
20             el:"#app",//el即element, 要渲染的页面元素
21             data: {
22                 birthday:1429032123201
23             },
24             computed:{
25                 birth(){
26                     const date = new Date(this.birthday);
27                     return date.getFullYear() + "-" + (date.getMonth()+1) + "-" +
date.getDay();
28                 }
29             }
30         });

```

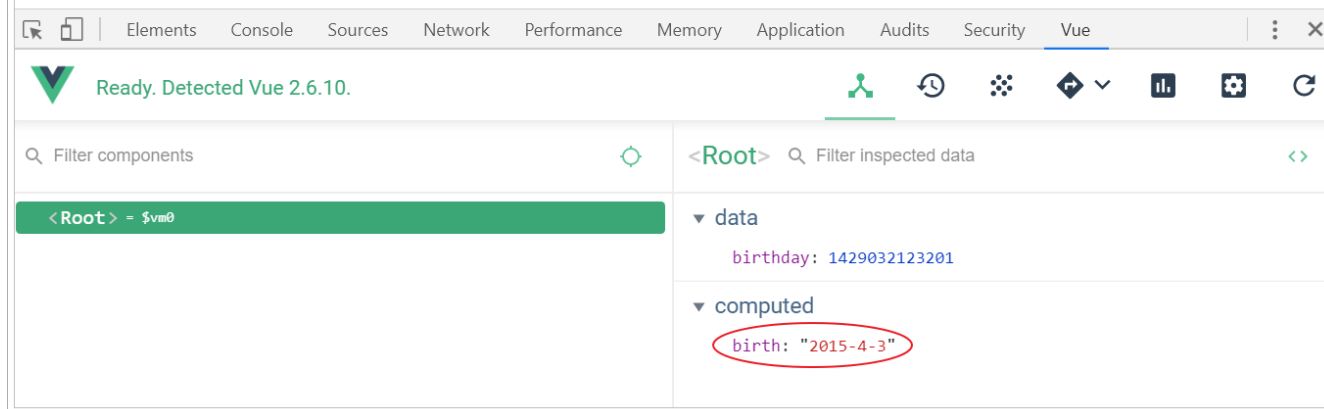
```
31 </script>
32
33 </body>
34 </html>
```

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：

你的生日为：2015-4-3

computed计算方式；你的生日为：2015-4-3



4.8.watch

4.8.1. 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"
```

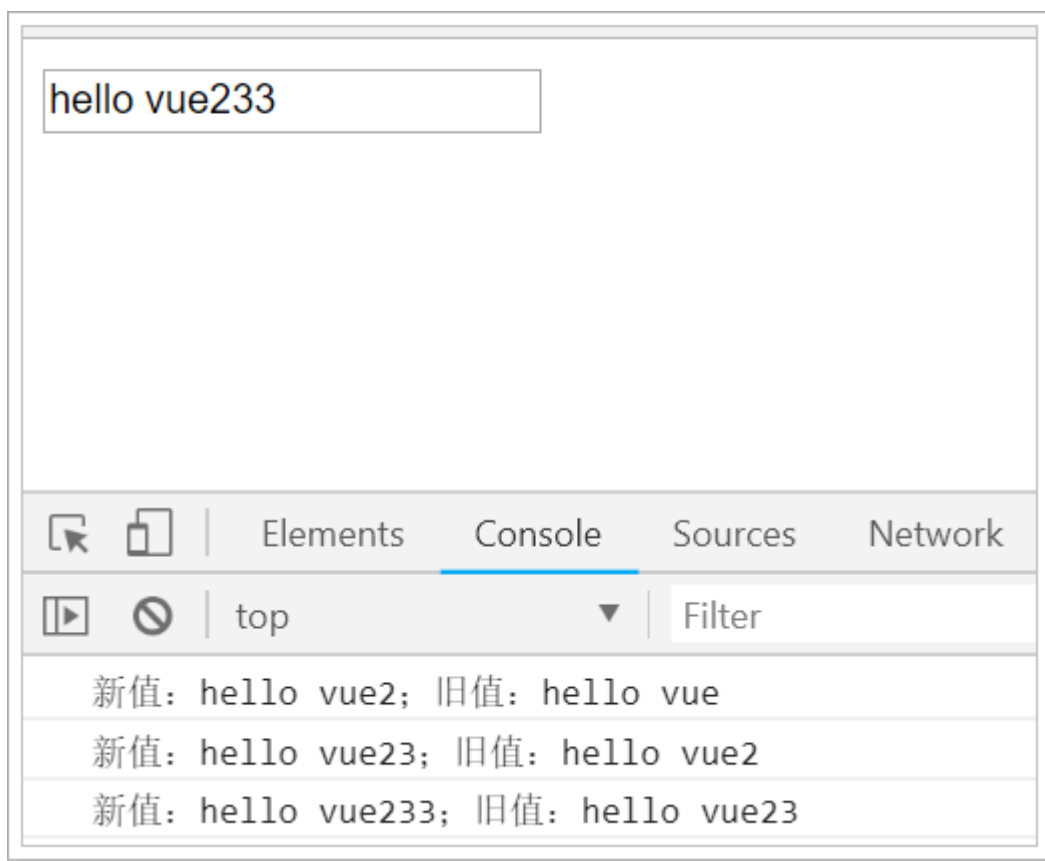


```

17     },
18     watch:{
19         message(newValue, oldValue){
20             console.log("新值: " + newValue + "; 旧值: " + oldValue);
21         }
22     }
23 });
24 </script>
25
26 </body>
27 </html>

```

效果：



4.8.2. 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>

```

```

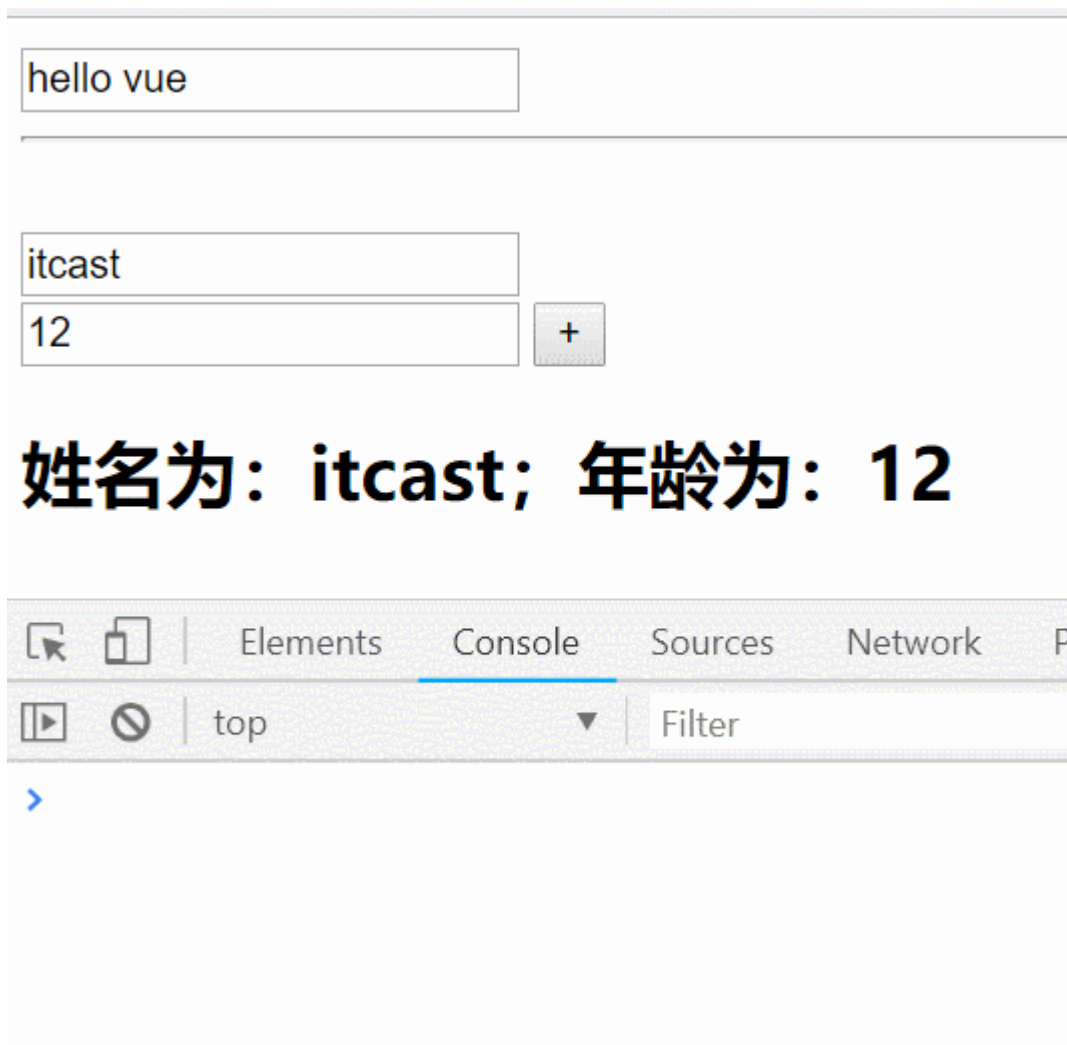
11     <input v-model="person.name"><br>
12     <input v-model="person.age"> <button @click="person.age++">+</button>
13     <h2>
14         姓名为: {{person.name}}; 年龄为: {{person.age}}
15     </h2>
16 </div>
17 <script src="node_modules/vue/dist/vue.js"></script>
18 <script>
19     var app = new Vue({
20         el:"#app", //el即element, 要渲染的页面元素
21         data: {
22             message:"hello vue",
23             person:{ "name": "itcast", "age": 12 }
24         },
25         watch: {
26             message(newValue, oldValue) {
27                 console.log("新值: " + newValue + "; 旧值: " + oldValue);
28             },
29             person: {
30                 //开启深度监控, 可以监控到对象属性值的变化
31                 deep: true,
32                 //监控的处理方法
33                 handler(obj) {
34                     console.log("name = " + obj.name + ", age=" + obj.age);
35                 }
36             }
37         }
38     });
39 </script>
40
41 </body>
42 </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:



5. 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。

5.1. 定义全局组件

通过Vue的component方法来定义一个全局组件。

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

```

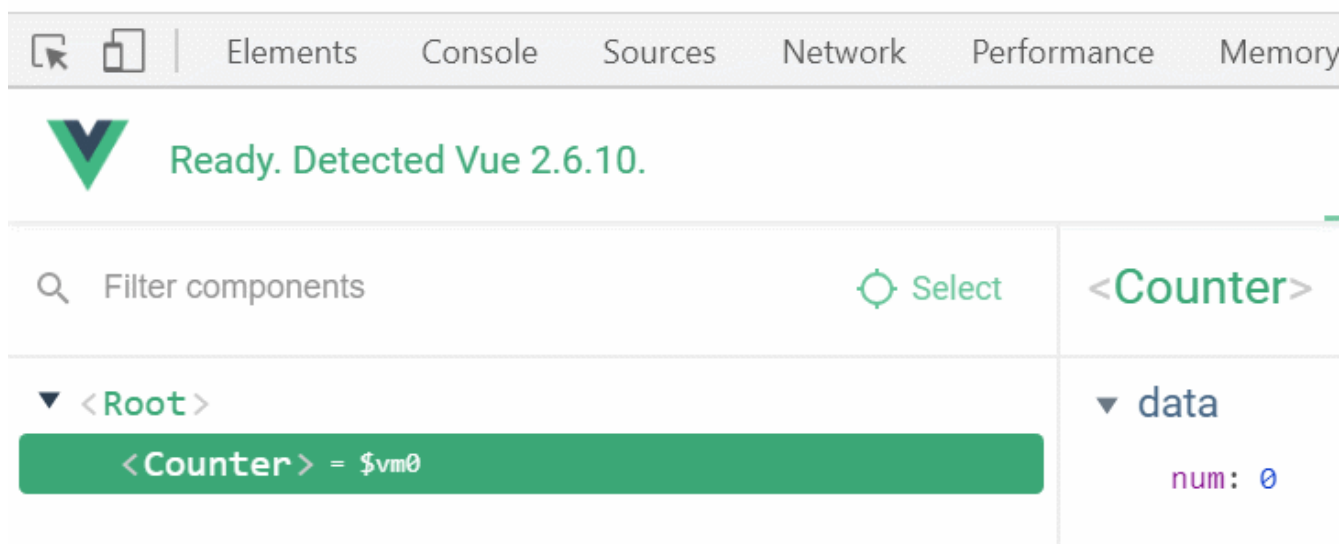
8   <div id="app">
9       <!--使用定义好的全局组件-->
10      <counter></counter>
11  </div>
12  <script src="node_modules/vue/dist/vue.js"></script>
13  <script>
14      //定义组件
15      const counter = {
16          template: "<button @click='num++'>你点击了{{num}}次; 我记住了</button>",
17          data(){
18              return {num: 0}
19          }
20      };
21
22      //全局注册组件; 参数1: 组件名称, 参数2: 组件
23      Vue.component("counter", counter);
24
25      var app = new Vue({
26          el: "#app"
27      });
28
29  </script>
30
31  </body>
32  </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：

你点击了0次；我记住了

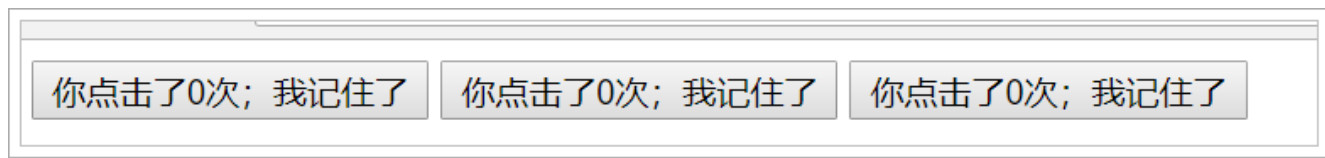


5.2. 组件的复用

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 `<counter>` 组件时, 它的data 并不是像这样直接提供一个对象:

```
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!

5.3. 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   };
```

```

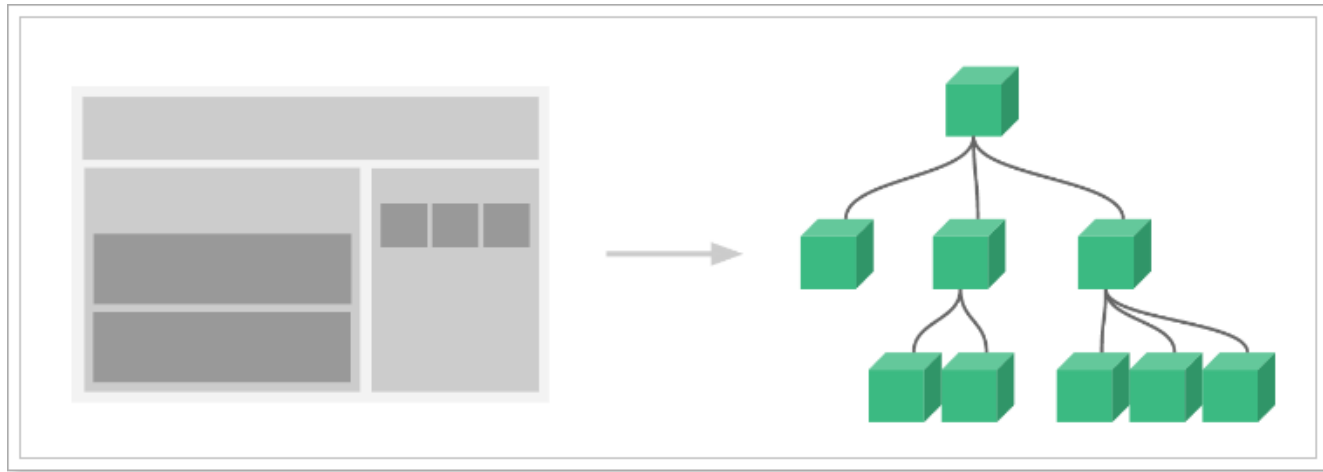
23
24 //全局注册组件; 参数1: 组件名称, 参数2: 组件
25 //Vue.component("counter", counter);
26
27 var app = new Vue({
28   el:"#app",
29   //局部注册组件
30   components:{
31     counter: counter
32   }
33 });
34
35 </script>
36
37 </body>
38 </html>

```

- 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

5.4. 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。

5.4.1. 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：

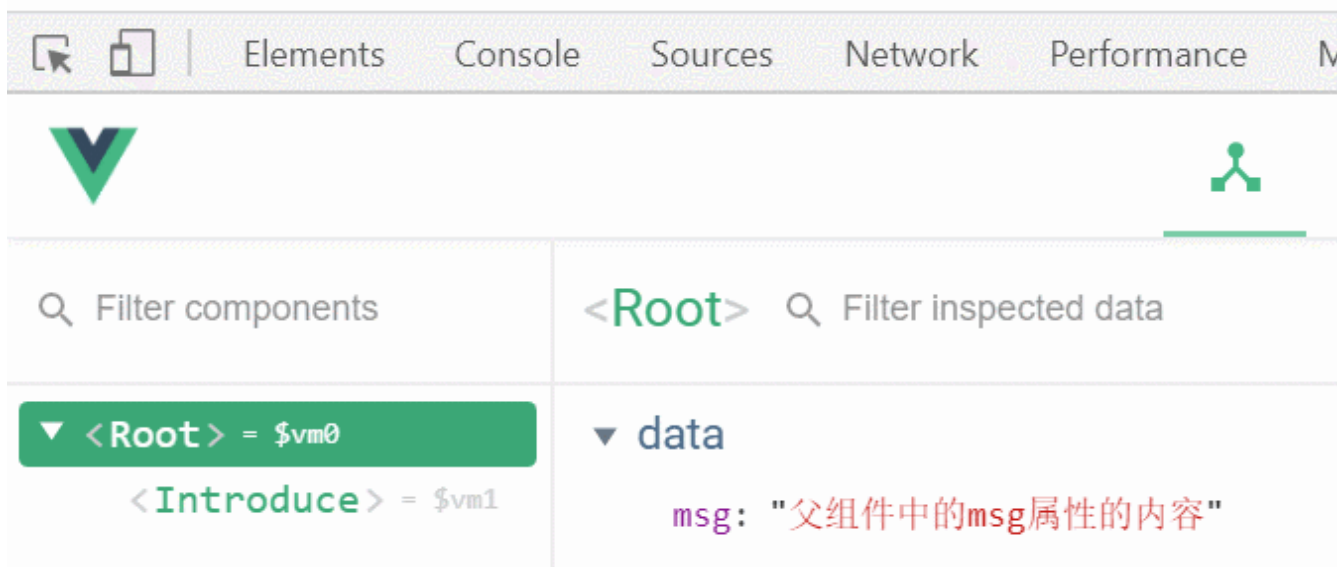
```

1 <!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    <introduce :title="msg"></introduce>
11  </div>
12  <script src="node_modules/vue/dist/vue.js"></script>
13  <script>
14    //定义组件
15    const introduce = {
16      //使用props属性title的值渲染模版
17      template: "<h2>{{title}}</h2>",
18      //定义接收来自父组件的属性
19      props:["title"]
20    };
21
22    //全局注册组件；参数1：组件名称，参数2：组件
23    Vue.component("introduce", introduce);
24
25    var app = new Vue({
26      el:"#app",
27      data:{
28        msg: "父组件中的msg属性的内容"
29      }
30    });
31
32  </script>
33
34 </body>
35 </html>

```

效果：

父组件中的msg属性的内容



5.4.2. 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    ',
7    props: { // 通过props来接收父组件传递来的属性
8      items: { // 这里定义items属性
9        type: Array, // 要求必须是Array类型
10       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     `,  
23     //定义接收来自父组件的属性  
24     props: {  
25       //定义模版中使用的属性  
26       items: {  
27         //必须为数组类型  
28         type: Array,  
29         //默认为空数组  
30         default: []  
31       }  
32     }  
33   };  
34  
35   var app = new Vue({  
36     el: "#app",  
37     data: {  
38       msg: "父组件中的msg属性的内容",  
39       lessons: [
```

```

40         {"id":1, "name":"Java"},
41         {"id":2, "name":"PHP"},
42         {"id":3, "name":"前端"}
43     ]
44 },
45     //注册组件
46     components: {
47         //如果组件key和value一致可以简写如下
48         myList
49     }
50 });
51
52 </script>
53
54 </body>
55 </html>

```

效果：



5.5.3. 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>
10         <!--使用定义好的全局组件-->
11         <counter :snum="num"></counter>
12     </div>
13     <script src="node_modules/vue/dist/vue.js"></script>
14     <script>

```

```

15 //定义组件
16 const counter = {
17   //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
18   template: `
19     <div>
20       <button @click="snum++">+</button>
21       <button @click="snum--">-</button>
22     </div>
23   `,
24   props: ["snum"]
25 };
26
27 //全局注册组件；参数1：组件名称，参数2：组件
28 vue.component("counter", counter);
29
30 var app = new Vue({
31   el: "#app",
32   data: {
33     num: 0
34   }
35 });
36
37 </script>
38
39 </body>
40 </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：

The screenshot shows a web application with a large text display showing "num = 0" and two buttons, "+" and "-", for incrementing and decrementing the value. Below the application is a browser's developer console. The console shows a warning message: "[Vue warn]: Avoid mutating a prop directly since the value will be overwritten whenever the parent component re-renders. Instead, use a data or computed property based on the prop's value. Prop being mutated: \"snum\"". The warning is attributed to "vue.js:634". Below the warning, the component stack is shown: "found in" followed by "----> <Counter>" and "<Root>".

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  var app = new Vue({
2    el:"#app",
3    data:{
4      num:0
5    },
6    methods:{
7      //父组件中定义操作num的方法
8      numPlus(){
9        this.num++;
10     },
11     numReduce(){
12       this.num--;
13     }
14   }
15 });

```

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>
3    <!--使用定义好的全局组件-->
4    <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  //定义组件
2  const counter = {
3    //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
4    template: `
5      <div>
6        <button @click="incrNum">+</button>
7        <button @click="decrNum">-</button>
8      </div>
9    `,
10   props: ["snum"],
11   methods: {
12     //点击模板中使用的方法
13     incrNum(){
14       return this.$emit("plus");
15     },
16     decrNum(){
17       return this.$emit("reduce");
18     }
19   }
20 };

```

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>
10    <!--使用定义好的全局组件-->
11    <counter @plus="numPlus" @reduce="numReduce" :snum="num"></counter>
12  </div>
13  <script src="node_modules/vue/dist/vue.js"></script>
14  <script>
15    //定义组件
16    const counter = {
17      //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
18      template: `
19        <div>
20          <button @click="incrNum">+</button>
21          <button @click="decrNum">-</button>
22        </div>
23      `,
24      props: ["snum"],
25      methods: {
26        //点击模板中使用的方法
27        incrNum(){
28          return this.$emit("plus");
29        },
30        decrNum(){
31          return this.$emit("reduce");
32        }
33      }
34    };
35
36    //全局注册组件；参数1：组件名称，参数2：组件
37    Vue.component("counter", counter);
38
39    var app = new Vue({
40      el: "#app",
41      data: {
42        num: 0
43      },
44      methods: {
45        //父组件中定义操作num的方法
46        numPlus(){
47          this.num++;
48        },
49        numReduce(){
50          this.num--;
51        }
52      }
53    });
```

```
52     }  
53   });  
54  
55 </script>  
56  
57 </body>  
58 </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：

num = 0



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。

6. 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>

6.2 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>
```

6.3 axios应用

6.3.1 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]])

1) config请求配置

这些是创建请求时可以用的配置选项。只有 `url` 是必需的。如果没有指定 `method`，请求将默认使用 `get` 方法。

```
1 {
2   // `url` 是用于请求的服务器 URL
3   url: '/user',
4
5   // `method` 是创建请求时使用的方法
6   method: 'get', // 默认是 get
7
8   // `baseUrl` 将自动加在 `url` 前面, 除非 `url` 是一个绝对 URL。
9   // 它可以通过设置一个 `baseUrl` 便于为 axios 实例的方法传递相对 URL
10  baseUrl: 'https://some-domain.com/api/',
11
12  // `transformRequest` 允许在向服务器发送前, 修改请求数据
13  // 只能用在 'PUT', 'POST' 和 'PATCH' 这几个请求方法
14  // 后面数组中的函数必须返回一个字符串, 或 ArrayBuffer, 或 Stream
15  transformRequest: [function (data) {
16    // 对 data 进行任意转换处理
17
18    return data;
19  }],
20
21  // `transformResponse` 在传递给 then/catch 前, 允许修改响应数据
22  transformResponse: [function (data) {
23    // 对 data 进行任意转换处理
24
25    return data;
26  }],
27
28  // `headers` 是即将被发送的自定义请求头
29  headers: {
30    'X-Requested-With': 'XMLHttpRequest',
```



```

31     'Content-Type': 'application/json'
32 },
33
34 // `params` 是即将与请求一起发送的 URL 参数
35 // 必须是一个无格式对象(plain object)或 URLSearchParams 对象
36 params: {
37     ID: 12345
38 },
39
40 // `data` 是作为请求主体被发送的数据
41 // 只适用于这些请求方法 'PUT', 'POST', 和 'PATCH'
42 // 在没有设置 `transformRequest` 时, 必须是以下类型之一:
43 // - string, plain object, ArrayBuffer, ArrayBufferView, URLSearchParams
44 // - 浏览器专属: FormData, File, Blob
45 // - Node 专属: Stream
46 data: {
47     firstName: 'Fred'
48 },
49
50 // `timeout` 指定请求超时的毫秒数(0 表示无超时时间)
51 // 如果请求话费了超过 `timeout` 的时间, 请求将被中断
52 timeout: 1000,
53
54 // `withCredentials` 表示跨域请求时是否需要使用凭证
55 withCredentials: false, // 默认的
56
57 // `responseType` 表示服务器响应的数据类型, 可以是 'arraybuffer', 'blob', 'document',
58 // 'json', 'text', 'stream'
59 responseType: 'json', // 默认的
60
61 // `maxContentLength` 定义允许的响应内容的最大尺寸
62 maxContentLength: 2000,
63
64 // `validateStatus` 定义对于给定的HTTP 响应状态码是 resolve 或 reject promise 。如果
65 // `validateStatus` 返回 `true` (或者设置为 `null` 或 `undefined`), promise 将被 resolve; 否
66 // 则, promise 将被 rejecte
67 validateStatus: function (status) {
68     return status >= 200 && status < 300; // 默认的
69 },
70
71 // `maxRedirects` 定义在 node.js 中 follow 的最大重定向数目
72 // 如果设置为0, 将不会 follow 任何重定向
73 maxRedirects: 5 // 默认的
74 }

```

2) 响应结构

```

1  {
2    // `data` 由服务器提供的响应
3    data: {},
4
5    // `status` 来自服务器响应的 HTTP 状态码

```

```

6   status: 200,
7
8   // `statusText` 来自服务器响应的 HTTP 状态信息
9   statusText: 'OK',
10
11  // `headers` 服务器响应的头
12  headers: {},
13
14  // `config` 是为请求提供的配置信息
15  config: {}
16 }
17

```

使用 `then` 时，你将接收下面这样的响应：

```

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](#) 作为 `then` 的第二个参数时，响应可以通过 `error` 对象可被使用。

6.3.2 axios方法示例

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` 传递相关配置来创建请求

axios(config)

```

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     <ul>
12         <li v-for="(user,index) in users" :key="index">
13             {{index}}--{{user.name}}--{{user.age}}--{{user.gender}}
14         </li>
15     </ul>
16 </div>
17 <script>
18     var app = new Vue({
19         el: "#app",
20         data: {
21             users: []
22         },
23         created(){
24             //加载数据
25             axios({
26                 method: "get",
27                 url: "data.json"
28             }).then((res)=>{
29                 console.log(res);
30                 //将获取数据设置到users属性
31                 app.users = res.data;
32             }).catch(error=>{
33                 alert(error);
34             });
35         }
36     });
37 </script>
38
39 </body>
40 </html>

```

6.3.3 get方法示例

将上述示例中的axios操作部分修改为如下：

```

1         axios.get("data.json")
2             .then( res => {
3                 console.log(res);
4                 //将获取数据设置到users属性
5                 app.users = res.data;
6             }).catch(error =>{
7                 console.log(error)
8             });

```

6.3.4 post方法示例

将示例中的axios操作部分修改为如下：

```
1      axios.post("data.json")
2      .then( res => {
3          console.log(res);
4          //将获取数据设置到users属性
5          app.users = res.data;
6      }).catch(error =>{
7          console.log(error)
8      });
9  }
10
```