

目录 Contents

- ◆ 侦听器
- ◆ 计算属性
- ◆ vue-cli
- ◆ vue 组件

watch 侦听器

1. 什么是 watch 侦听器

watch 侦听器允许开发者监视数据的变化，从而针对数据的变化做特定的操作。

语法格式如下：

```
1 const vm = new Vue({
2   el: '#app',
3   data: { username: '' },
4   watch: {
5     // 监听 username 值的变化
6     // newVal 是“变化后的新值”，oldVal 是“变化之前的旧值”
7     username(newVal, oldVal) {
8       console.log(newVal, oldVal)
9     }
10  }
11 })
```

watch 侦听器

2. 使用 watch 检测用户名是否可用

监听 username 值的变化，并使用 axios 发起 Ajax 请求，检测当前输入的用户名是否可用：

```
watch: {  
  // 监听 username 值的变化  
  async username(newVal) {  
    if (newVal === '') return  
    // 使用 axios 发起请求，判断用户名是否可用  
    const { data: res } = await axios.get('https://www.escook.cn/api/finduser/' + newVal)  
    console.log(res)  
  }  
}
```

watch 侦听器

3. immediate 选项

默认情况下，组件在初次加载完毕后不会调用 watch 侦听器。如果想让 watch 侦听器立即被调用，则需要使用 **immediate** 选项。示例代码如下：

```
watch: {
  username: {
    // handler 是固定写法，表示当 username 的值变化时，自动调用 handler 处理函数
    handler: async function (newVal) {
      if (newVal === '') return
      const { data: res } = await axios.get('https://www.escook.cn/api/finduser/' + newVal)
      console.log(res)
    },
    // 表示页面初次渲染好之后，就立即触发当前的 watch 侦听器
    immediate: true
  }
}
```

4. deep 选项

如果 **watch** 侦听的是一个对象，如果对象中的属性值发生了变化，则无法被监听到。此时需要使用 **deep** 选项，代码示例如下：

```
1 const vm = new Vue({
2   el: '#app',
3   data: {
4     info: { username: 'admin' }
5   },
6   watch: {
7     info: {
8       handler(newVal) {
9         console.log(newVal.username)
10      },
11      deep: true
12    }
13  }
14 })
```

5. 监听对象单个属性的变化

如果只想监听对象中单个属性的变化，则可以按照如下的方式定义 watch 侦听器：

```
1 const vm = new Vue({
2   el: '#app',
3   data: {
4     info: { username: 'admin' }
5   },
6   watch: {
7     'info.username': {
8       handler(newVal) {
9         console.log(newVal)
10      }
11    }
12  }
13 })
```

目录 Contents

- ◆ 侦听器
- ◆ 计算属性
- ◆ vue-cli
- ◆ vue 组件

■ ■ ■ 计算属性

1. 什么是计算属性

计算属性指的是通过一系列运算之后，最终得到一个属性值。

这个动态计算出来的属性值可以被模板结构或 methods 方法使用。示例代码如下：

```
1 var vm = new Vue({
2   el: '#app',
3   data: {
4     r: 0, g: 0, b: 0
5   },
6   computed: {
7     rgb() { return `rgb(${this.r}, ${this.g}, ${this.b})` }
8   },
9   methods: {
10    show() { console.log(this.rgb) }
11  },
12 })
```

■ 计算属性

2. 计算属性的特点

- ① 虽然计算属性在声明的时候被定义为方法，但是计算属性的本质是一个属性
- ② 计算属性会缓存计算的结果，只有计算属性依赖的数据变化时，才会重新进行运算

目录 Contents

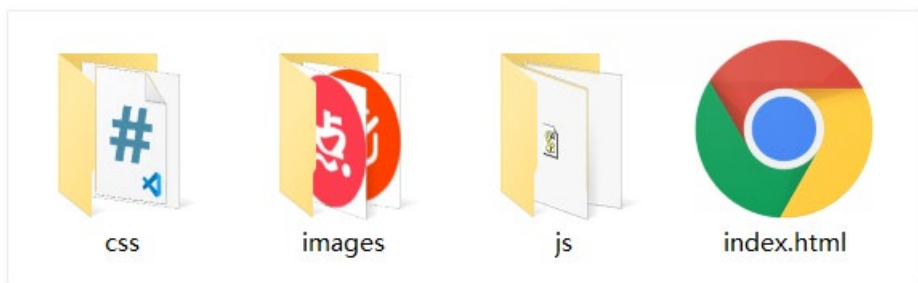
- ◆ 侦听器
- ◆ 计算属性
- ◆ vue-cli
- ◆ vue 组件

■ 单页面应用程序

1. 什么是单页面应用程序

单页面应用程序（英文名：Single Page Application）简称 SPA，顾名思义，指的是一个 Web 网站中只有唯一的一个 HTML 页面，所有的功能与交互都在这唯一的一个页面内完成。

例如资料中的这个 Demo 项目：





2. 什么是 vue-cli

vue-cli 是 Vue.js 开发的标准工具。它简化了程序员基于 webpack 创建工程化的 Vue 项目的过程。

引用自 vue-cli 官网上的一句话：

程序员可以专注在撰写应用上，而不必花好几天去纠结 webpack 配置的问题。

中文官网：<https://cli.vuejs.org/zh/>



3. 安装和使用

vue-cli 是 npm 上的一个全局包，使用 `npm install` 命令，即可方便的把它安装到自己的电脑上：

```
npm install -g @vue/cli
```

基于 vue-cli 快速生成工程化的 Vue 项目：

```
vue create 项目的名称
```

4. vue 项目的运行流程

在工程化的项目中，vue 要做的事情很单纯：通过 `main.js` 把 `App.vue` 渲染到 `index.html` 的指定区域中。

其中：

- ① `App.vue` 用来编写待渲染的模板结构
- ② `index.html` 中需要预留一个 `el` 区域
- ③ `main.js` 把 `App.vue` 渲染到了 `index.html` 所预留的区域中

目录 Contents

- ◆ 侦听器
- ◆ 计算属性
- ◆ vue-cli
- ◆ vue 组件

1. 什么是组件化开发

组件化开发指的是：根据封装的思想，把页面上可重用的 UI 结构封装为组件，从而方便项目的开发和维护。

2. vue 中的组件化开发

vue 是一个支持组件化开发的前端框架。

vue 中规定：组件的后缀名是 `.vue`。之前接触到的 `App.vue` 文件本质上就是一个 vue 的组件。

3. vue 组件的三个组成部分

每个 .vue 组件都由 3 部分构成，分别是：

- `template` -> 组件的模板结构
- `script` -> 组件的 JavaScript 行为
- `style` -> 组件的样式

其中，每个组件中必须包含 `template` 模板结构，而 `script` 行为和 `style` 样式是可选的组成部分。

3.1 template

vue 规定：每个组件对应的模板结构，需要定义到 `<template>` 节点中。

```
1 <template>
2   <!-- 当前组件的 DOM 结构，需要定义到 template 标签的内部 -->
3 </template>
```

注意：

- template 是 vue 提供的容器标签，只起到包裹性质的作用，它不会被渲染为真正的 DOM 元素
- template 中只能包含唯一的根节点

3.2 script

vue 规定：开发者可以在 `<script>` 节点中封装组件的 JavaScript 业务逻辑。

`<script>` 节点的基本结构如下：

```
1 <script>
2 // 今后，组件相关的 data 数据、methods 方法等，
3 // 都需要定义到 export default 所导出的对象中。
4 export default {}
5 </script>
```

vue 组件

.vue 组件中的 data 必须是函数

vue 规定：.vue 组件中的 data 必须是一个函数，不能直接指向一个数据对象。

因此在组件中定义 data 数据节点时，下面的方式是错误的：

```
1 data: { // 组件中，不能直接让 data 指向一个数据对象（会报错）
2   count: 0
3 }
```


会导致多个组件实例共用同一份数据的问题，请参考官方给出的示例：

<https://cn.vuejs.org/v2/guide/components.html#data-必须是一个函数>

3.3 style

vue 规定：组件内的 `<style>` 节点是可选的，开发者可以在 `<style>` 节点中编写样式美化当前组件的 UI 结构。

`<script>` 节点的基本结构如下：



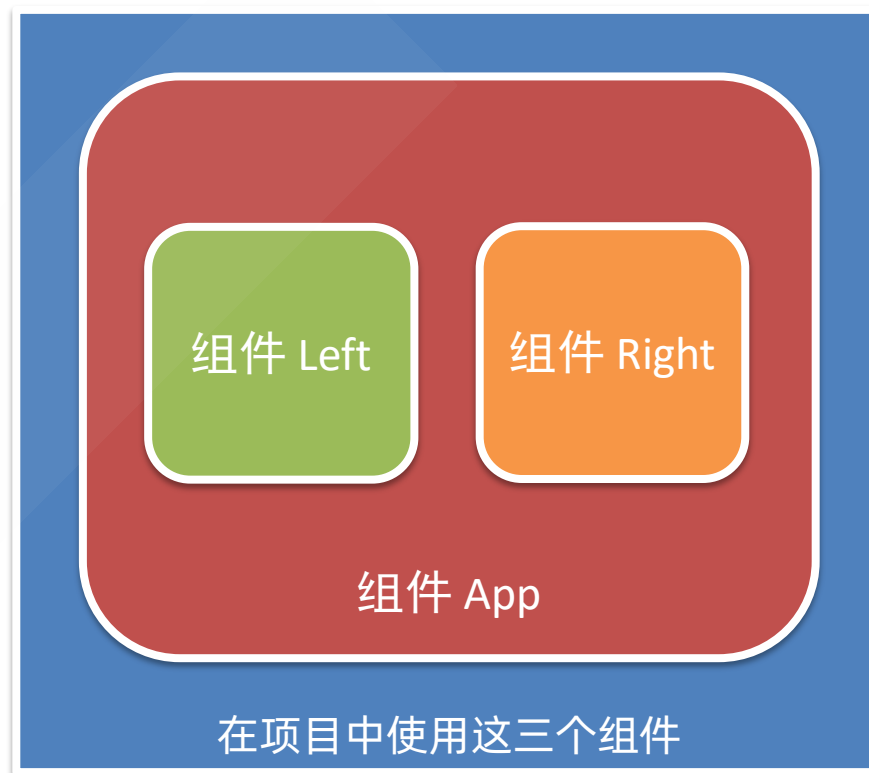
```
1 <style>
2 h1 {
3   font-weight: normal;
4 }
5 </style>
```

让 style 中支持 less 语法

在 `<style>` 标签上添加 `lang="less"` 属性，即可使用 less 语法编写组件的样式：

```
1 <style lang="less">
2 h1 {
3   font-weight: normal;
4   span {
5     color: red;
6   }
7 }
8 </style>
```


4. 组件之间的父子关系



组件在被封装好之后，彼此之间是相互独立的，不存在父子关系

在使用组件的时候，根据彼此的嵌套关系，形成了父子关系、兄弟关系

vue 组件

4.1 使用组件的三个步骤

```
<div class="box">  
  <Left></Left>  
</div>
```

```
import Left from '@components/Left.vue'
```

```
export default {  
  components: {  
    Left  
  }  
}
```

App.vue 组件

步骤3: 以标签形式使用刚才注册的组件

步骤1: 使用 import 语法导入需要的组件

步骤2: 使用 components 节点注册组件

4.2 通过 components 注册的是私有子组件

例如：

在组件 A 的 components 节点下，注册了组件 F。

则组件 F 只能用在组件 A 中；不能被用在组件 C 中。

请大家思考两个问题：

- ① 为什么 F 不能用在组件 C 中？
- ② 怎样才能能在组件 C 中使用 F？

4.3 注册全局组件

在 vue 项目的 `main.js` 入口文件中，通过 `Vue.component()` 方法，可以注册全局组件。示例代码如下：

```
1 // 导入需要全局注册的组件
2 import Count from '@/components/Count.vue'
3
4 // 参数1: 字符串格式，表示组件的“注册名称”
5 // 参数2: 需要被全局注册的那个组件
6 Vue.component('MyCount', Count)
```

5. 组件的 props

props 是组件的自定义属性，在封装通用组件的时候，合理地使用 props 可以极大的提高组件的复用性！

它的语法格式如下：

```
1 export default {  
2   // 组件的自定义属性  
3   props: ['自定义属性A', '自定义属性B', '其它自定义属性...'],  
4  
5   // 组件的私有数据  
6   data() {  
7     return { }  
8   }  
9 }
```

vue 组件

5.1 props 是只读的

vue 规定：组件中封装的自定义属性是只读的，程序员不能直接修改 props 的值。否则会直接报错：

```
[HMR] Waiting for update signal from WDS... log.js?1afd:24
vue-devtools Detected Vue v2.6.14 backend.js:2237
✖ [Vue warn]: Avoid mutating a prop directly since the value will be overwritten whenever the parent component re-renders. Instead, use a data or computed property based on the prop's value. Prop being mutated: "init"
    vue.runtime.esm.js?2b0e:619
found in
  ---> <MyCount> at src/components/Count.vue
    <Left> at src/components/Left.vue
      <App> at src/App.vue
        <Root>
```

要想修改 props 的值，可以把 props 的值转存到 data 中，因为 data 中的数据都是可读可写的！

```
1 props: ['init'],
2 data() {
3   return {
4     count: this.init // 把 this.init 的值转存到 count
5   }
6 }
```

5.2 props 的 **default** 默认值

在声明自定义属性时，可以通过 **default** 来定义属性的默认值。示例代码如下：

```
1 export default {  
2   props: {  
3     init: {  
4       // 用 default 属性定义属性的默认值  
5       default: 0  
6     }  
7   }  
8 }
```

5.3 props 的 **type** 值类型

在声明自定义属性时，可以通过 **type** 来定义属性的值类型。示例代码如下：

```
1 export default {
2   props: {
3     init: {
4       // 用 default 属性定义属性的默认值
5       default: 0,
6       // 用 type 属性定义属性的值类型，
7       // 如果传递过来的值不符合此类型，则会在终端报错
8       type: Number
9     }
10  }
11 }
```


5.4 props 的 **required** 必填项

在声明自定义属性时，可以通过 **required** 选项，将属性设置为**必填项**，强制用户必须传递属性的值。示例代码如下：

```
1 export default {  
2   props: {  
3     init: {  
4       // 值类型为 Number 数字  
5       type: Number,  
6       // 必填项校验  
7       required: true  
8     }  
9   }  
10 }
```

6. 组件之间的样式冲突问题

默认情况下，**写在 .vue 组件中的样式会全局生效**，因此很容易造成**多个组件之间的样式冲突问题**。

导致组件之间样式冲突的根本原因是：

- ① 单页面应用程序中，所有组件的 DOM 结构，都是基于**唯一的 index.html 页面**进行呈现的
- ② 每个组件中的样式，都会**影响整个 index.html 页面**中的 DOM 元素

■ 组件的基本使用

6.1 思考：如何解决组件样式冲突的问题

为每个组件分配唯一的自定义属性，在编写组件样式时，通过属性选择器来控制样式的作用域，示例代码如下：

```
1 <template>
2   <div class="container" data-v-001>
3     <h3 data-v-001>轮播图组件</h3>
4   </div>
5 </template>
6
7 <style>
8   /* 通过中括号"属性选择器"，来防止组件之间的"样式冲突问题"，
9      因为每个组件分配的自定义属性是"唯一的" */
10  .container[data-v-0001] {
11    border: 1px solid red;
12  }
13 </style>
```

■ 组件的基本使用

6.2 style 节点的 scoped 属性

为了提高开发效率和开发体验，vue 为 **style 节点** 提供了 **scoped** 属性，从而防止组件之间的样式冲突问题：

```
1 <template>
2   <div class="container">
3     <h3>轮播图组件</h3>
4   </div>
5 </template>
6
7 <style scoped>
8   /* style 节点的 scoped 属性，用来自动为每个组件分配唯一的“自定义属性”，
9      并自动为当前组件的 DOM 标签和 style 样式应用这个自定义属性，防止组件的样式冲突问题 */
10  .container {
11    border: 1px solid red;
12  }
13 </style>
```

■ 组件的基本使用

6.3 /deep/ 样式穿透

如果给当前组件的 style 节点添加了 scoped 属性，则当前组件的样式对其子组件是不生效的。如果想让某些样式对子组件生效，可以使用 /deep/ 深度选择器。

```
1 <style lang="less" scoped>
2 .title {
3   color: blue; /* 不加 /deep/ 时，生成的选择器格式为 .title[data-v-052242de] */
4 }
5
6 /deep/ .title {
7   color: blue; /* 加上 /deep/ 时，生成的选择器格式为 [data-v-052242de] .title */
8 }
9 </style>
```

