

## Vue是什么？为什么要学习他



### 易用

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

### 灵活

不断繁荣的生态系统，可以在一个库  
和一套完整框架之间自如伸缩。

### 高效

20kB min+gzip 运行大小  
超快虚拟 DOM  
最省心的优化

## Vue是什么？

Vue是前端优秀框架，是一套用于构建用户界面的**渐进式框架**

## 为什么要学习Vue

- ① Vue是目前前端最火的框架之一
- ② Vue是目前企业技术栈中要求的知识点
- ③ Vue可以提升开发体验
- ④ Vue学习难度较低
- ⑤ ...

## Vue开发前的准备



## 安装Vue工具 Vue CLI

Vue CLI Vue.js 开发的标准工具，Vue CLI 是一个基于 Vue.js 进行快速开发的完整系统

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

安装之后，你就可以在命令行中访问 vue 命令。你可以通过简单运行 vue，看看是否展示出了一份所有可用命令的帮助信息，来验证它是否安装成功。

```
1 | vue --version
```

## 创建一个项目

运行以下命令来创建一个新项目

```
1 | vue create vue-demo
```

### 温馨提示

在控制台中，可以用上下按键调整选择项

在控制台中，可以用空格(spacebar)选择是否选中和取消选中

可以选择默认项目模板，或者选“手动选择特性”来选取需要的特性。



我们选择 Babel 和 Progressive Web App (PWA) Support 两个选项即可

## 温馨提示

在学习期间，不要选中 **Linter / Formatter** 以避免不必要的错误提示

```
Vue CLI v5.0.4
? Please pick a preset: Manually select features
? Check the features needed for your project: (Press <space> to select, <a> to toggle all, <i> to invert selection, and <enter> to proceed)
> (*) Babel
  ( ) TypeScript
  ( ) Progressive Web App (PWA) Support
  ( ) Router
  ( ) Vuex
  ( ) CSS Pre-processors
  ( ) Linter / Formatter
  ( ) Unit Testing
  ( ) E2E Testing
```

Vue目前有两个主流大版本 **vue2** 和 **vue3**，我们本套课程选择 **vue3** 最新版本

```
Vue CLI v5.0.4
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, PWA
? Choose a version of Vue.js that you want to start the project with (Use arrow keys)
> 3.x
  2.x
```

配置放在哪里？**In dedicated config files** 专用配置文件或者 **In package.json** 在 package.json文件

```
Vue CLI v5.0.4
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, PWA
? Choose a version of Vue.js that you want to start the project with 3.x
? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)
> In dedicated config files
  In package.json
```

将其保存为未来项目的预置？**y** 代表保存，并添加名字，**n** 不保存

```
Vue CLI v5.0.4
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, PWA
? Choose a version of Vue.js that you want to start the project with 3.x
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? (y/N)
```

项目创建成功如下提示信息

```
Vue CLI v5.0.4
  Creating project in E:\itbaizhan_base\源码\5.第五章: Vue\vue-demo.
  Installing CLI plugins. This might take a while...

  added 933 packages in 19s
    Invoking generators...
    Installing additional dependencies...

  added 16 packages in 2s
    Running completion hooks...

  Generating README.md...
  Successfully created project vue-demo.
  Get started with the following commands:

  $ cd vue-demo
  $ npm run serve

PS E:\itbaizhan_base\源码\5.第五章: Vue> 
```

## 运行项目

第一步：进入项目根目录 `cd vue-demo`

第二步：运行 `npm run serve` 启动项目

## 安装Vue高亮插件

VSCode中安装 `vetur` 或者 `volar` 都可，前者针对Vue2版本，后者针对Vue3版本

## 模板语法

# 模板语法



## 文本

数据绑定最常见的形式就是使用“Mustache”(双大括号)语法的文本插值

```
1 <span>Message: {{ msg }}</span>
```

一般配合 `js` 中的 `data()` 设置数据

```
1 export default {
2   name: 'HelloWorld',
3   data(){
4     return{
5       msg:"消息提示"
6     }
7   }
8 }
```

## 原始 HTML

双大括号会将数据解释为普通文本，而非 HTML 代码。为了输出真正的 HTML，你需要使用 `v-html` 指令

```

1 <p>Using mustaches: {{ rawHtml }}</p>
2 <p>Using v-html directive: <span v-
  html="rawHtml"></span></p>

```

```

1 data(){
2   return{
3     rawHtml:<a
4       href='https://www.itbaizhan.com'>百战</a>
5   }

```

## 属性 Attribute

Mustache 语法不能在 HTML 属性中使用，然而，可以使用 `v-bind` 指令

```

1 <div v-bind:id="dynamicId"></div>

```

```

1 data(){
2   return{
3     dynamicId:1001
4   }
5 }

```

### 温馨提示

`v-bind:` 可以简写成 `:`

## 使用 JavaScript 表达式

在我们的模板中，我们一直都只绑定简单的 property 键值，Vue.js 都提供了完全的 JavaScript 表达式支持

```

1 {{ number + 1 }}
2
3 {{ ok ? 'YES' : 'NO' }}
4
5 {{ message.split(' ').reverse().join(' ') }}
```

这些表达式会在当前活动实例的数据作用域下作为 JavaScript 被解析。有个限制就是，每个绑定都只能包含**单个表达式**，所以下面的例子都**不会生效**。

```

1 <!-- 这是语句，不是表达式: -->
2 {{ var a = 1 }}
3
4 <!-- 流程控制也不会生效，请使用三元表达式 -->
5 {{ if (ok) { return message } }}
```

## 条件渲染



## v-if

`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` 值的时候被渲染。

```
1 | <p v-if="flag">我是孙猴子</p>
```

```
1 | data() {
2 |     return {
3 |         flag: true
4 |     }
5 | }
```

## v-else

你可以使用 `v-else` 指令来表示 `v-if` 的“else 块”

```
1 | <p v-if="flag">我是孙猴子</p>
2 | <p v-else>你是傻猴子</p>
```

```

1 data() {
2     return {
3         flag: false
4     }
5 }
```

## v-show

另一个用于条件性展示元素的选项是 `v-show` 指令

```

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

### `v-if` vs `v-show` 的区别

`v-if` 是“真正”的条件渲染，因为它会确保在切换过程中，条件块内的事件监听器和子组件适当地被销毁和重建。

`v-if` 也是**惰性的**：如果在初始渲染时条件为假，则什么也不做——一直到条件第一次变为真时，才会开始渲染条件块。

相比之下，`v-show` 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。

一般来说，`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，如果需要非常频繁地切换，则使用 `v-show` 较好；如果在运行时条件很少改变，则使用 `v-if` 较好

## 列表渲染



### 用 `v-for` 把一个数组映射为一组元素

我们可以用 `v-for` 指令基于一个数组来渲染一个列表。`v-for` 指令需要使用 `item in items` 形式的特殊语法，其中 `items` 是源数据数组，而 `item` 则是被迭代的数组元素的别名。

```
1 <ul>
2   <li v-for="item in items">{{ item.message
3 }}</li>
4 </ul>
```

```

1 data() {
2     return {
3         items: [{ message: 'Foo' }, {
4             message: 'Bar'
5         }]
6     }
7 }
```

## 维护状态

当 Vue 正在更新使用 `v-for` 渲染的元素列表时，它默认使用“就地更新”的策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序，而是就地更新每个元素，并且确保它们在每个索引位置正确渲染。

为了给 Vue 一个提示，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素，你需要为每项提供一个唯一的 `key` attribute：

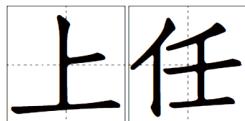
```

1 <div v-for="(item,index) in items"
2   :key="item.id|index">
3     <!-- 内容 -->
4   </div>
```

## 事件处理

上任

百战程序员



[拼音] [shàng rèn]

[释义] 1.指官吏就职：走马～。2.称前一任的官吏



## 监听事件

我们可以使用 `v-on` 指令 (通常缩写为 `@` 符号) 来监听 DOM 事件，并在触发事件时执行一些 JavaScript。用法为 `v-on:click="methodName"` 或使用快捷方式 `@click="methodName"`

```
1 <button @click="counter += 1">Add 1</button>
```

```
1 data() {
2   return {
3     counter: 0
4   }
5 }
```

## 事件处理方法

然而许多事件处理逻辑会更为复杂，所以直接把 JavaScript 代码写在 `v-on` 指令中是不可行的。因此 `v-on` 还可以接收一个需要调用的方法名称。

```
1 <button @click="greet">Greet</button>
```

```

1 methods: {
2     greet(event) {
3         // `event` 是原生 DOM event
4         if (event) {
5             alert(event.target.tagName)
6         }
7     }
8 }
```

## 内联处理器中的方法

这是官方的翻译称呼，其实我们可以直接叫他 "事件传递参数"

```

1 <button @click="say('hi')>Say hi</button>
2 <button @click="say('what')>Say
what</button>
```

```

1 methods: {
2     say(message) {
3         alert(message)
4     }
5 }
```

## 表单输入绑定



你可以用 `v-model` 指令在表单 `<input>`、`<textarea>` 及 `<select>` 元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素。尽管有些神奇，但 `v-model` 本质上不过是语法糖。它负责监听用户的输入事件来更新数据，并在某种极端场景下进行一些特殊处理。

```
1 <input v-model="message" placeholder="edit me" />
2 <p>Message is: {{ message }}</p>
```

```
1 data() {
2     return {
3         message:''
4     }
5 }
```

## 修饰符

`.lazy`

在默认情况下，`v-model` 在每次 `input` 事件触发后将输入框的值与数据进行同步。你可以添加 `lazy` 修饰符，从而转为在 `change` 事件之后进行同步

```

1 <input v-model.lazy="message" />
2 <p>Message is: {{ message }}</p>

```

```

1 data() {
2   return {
3     message:''
4   }
5 }

```

.trim

如果要自动过滤用户输入的首尾空白字符，可以给 `v-model` 添加 `trim` 修饰符

```

1 <input v-model.trim="message" />

```

```

1 data() {
2   return {
3     message:''
4   }
5 }

```

## 组件基础



## 单文件组件

Vue 单文件组件（又名 `*.vue` 文件，缩写为 **SFC**）是一种特殊的文件格式，它允许将 Vue 组件的模板、逻辑与样式封装在单个文件中

```
1 <template>
2     <h3>单文件组件</h3>
3 </template>
4
5 <script>
6 export default {
7     name: "MyComponent"
8 }
9 </script>
10
11 <style scoped>
12 h3{
13     color: red;
14 }
15 </style>
```

## 加载组件

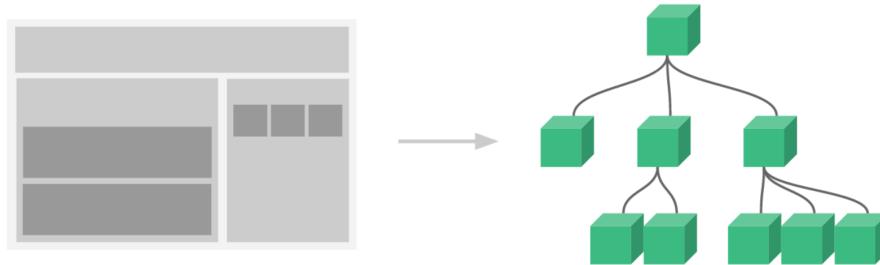
第一步：引入组件 `import MyComponentVue from './components/MyComponent.vue'`

第二步：挂载组件 `components: { MyComponentVue }`

第三步：显示组件 `<my-componentVue />`

## 组件的组织

通常一个应用会以一棵嵌套的组件树的形式来组织



#

## Props组件交互



组件与组件之间是需要存在交互的，否则完全没关系，组件的意义就很小了

**Prop** 是你可以在组件上注册的一些自定义 attribute

```
1 <my-component> title="标题" />
```

```
1 <template>
2   <h3>单文件组件</h3>
3   <p>{{ title }}</p>
4 </template>
5
6 <script>
7 export default {
8   name: "MyComponent",
9   props: {
10     title: {
11       type: String,
12       default: ""
13     }
14   }
15 }
16 </script>
```

## Prop 类型

Prop传递参数其实是没有类型限制的

```
1  props: {  
2      title: String,  
3      likes: Number,  
4      isPublished: Boolean,  
5      commentIds: Array,  
6      author: object,  
7      callback: Function  
8 }
```

### 温馨提示

数据类型为数组或者对象的时候，默认值是需要返回工厂模式

## 自定义事件组件交互



自定义事件可以在组件中反向传递数据，`prop` 可以将数据从父组件传递到子组件，那么反向如何操作呢，就可以利用自定义事件实现

`$emit`

```

1 <template>
2   <h3>单文件组件</h3>
3   <button @click="sendHandle">发送数据
4 </button>
5 </template>
6
6 <script>
7   export default {
8     name: "MyComponent",
9     methods: {
10       sendHandle(){
11         this.$emit("onCustom", "数据")
12       }
13     }
14   }
15 </script>
16
17 <style scoped>
18 h3 {
19   color: red;

```

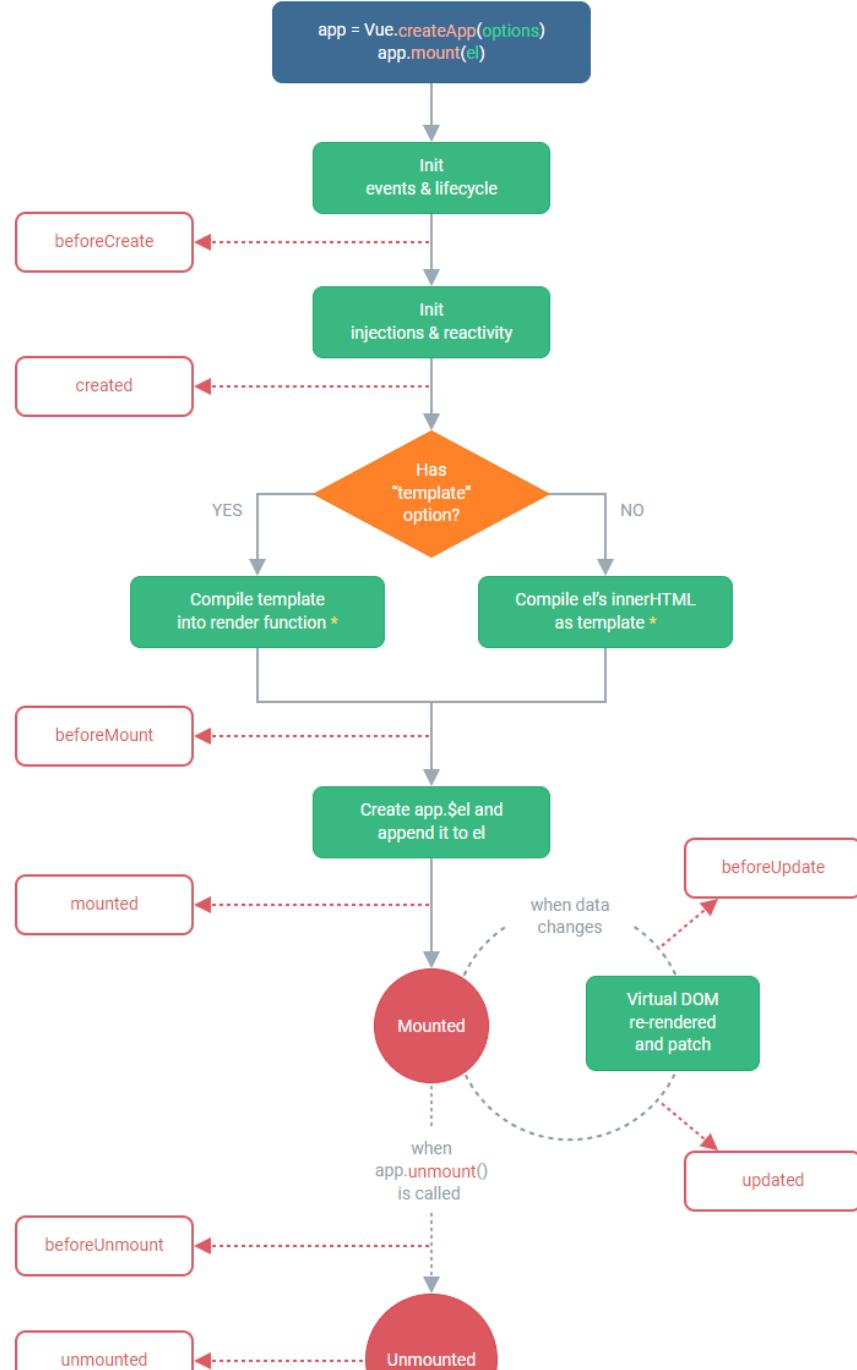
```
20  }
21 </style>
```

```
1 <template>
2   <my-componentVue @onCustom="getData" />
3 </template>
4
5 <script>
6
7 import MyComponentVue from
8   './components/MyComponent.vue'
9
10 export default {
11   name: 'App',
12   components: {
13     MyComponentVue
14   },
15   methods: {
16     getData(data) {
17       console.log(data);
18     }
19   }
20 </script>
```

## 组件生命周期



每个组件在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做**生命周期钩子**的函数，这给了用户在不同阶段添加自己的代码的机会



为了方便记忆，我们可以将他们分类：

**创建时：** `beforeCreate`、`created`

**渲染时：** `beforeMount`、`mounted`

**更新时：** `beforeUpdate`、`updated`

**卸载时：** `beforeUnmount`、`unmounted`

## Vue引入第三方



**Swiper** 开源、免费、强大的触摸滑动插件

**Swiper** 是纯javascript打造的滑动特效插件，面向手机、平板电脑等移动终端

**Swiper** 能实现触屏焦点图、触屏Tab切换、触屏轮播图切换等常用效果

### 温馨提示

官方文档：<https://swiperjs.com/vue>

安装指定版本：`npm instal --save swiper@8.1.6`

## 基础实现

```

1 <template>
2   <div class="hello">
3     <swiper class="mySwiper">
4       <swiper-slide>slide 1</swiper-slide>
5       <swiper-slide>slide 2</swiper-slide>
6       <swiper-slide>slide 3</swiper-slide>
7     </swiper>
8   </div>
9 </template>
10
11 <script>
12 import { Swiper, SwiperSlide } from
13   'swiper/vue';
14 import 'swiper/css';
15
16 export default {
17   name: 'HelloWorld',
18   components: {
19     Swiper,
20     SwiperSlide,
21   }
22 }</script>

```

## 添加指示器

```

1 <template>
2   <div class="hello">

```

```
3     <swiper class="mySwiper"
4       :modules="modules" :pagination="{ clickable:
5         true }">
6       <swiper-slide>
7         
9       </swiper-slide>
10      <swiper-slide>
11        
13      </swiper-slide>
14    </swiper>
15  </div>
16
17  <script>
18  import { Pagination } from 'swiper';
19  import { Swiper, SwiperSlide } from
20  'swiper/vue';
21  import 'swiper/css';
22  import 'swiper/css/pagination';
23
24  export default {
25    name: 'HelloWorld',
26    data(){
27      return{
28        modules: [ Pagination ]
```

```
29  },
30  components: {
31    Swiper,
32    SwiperSlide,
33  }
34 }
35 </script>
```

## Axios网络请求



Axios 是一个基于 promise 的网络请求库

## 安装

Axios的应用是需要单独安装的 `npm install --save axios`

## 引入

组件中引入: `import axios from "axios"`

全局引用:

```

1 import axios from "axios"
2
3 const app = createApp(App);
4 app.config.globalProperties.$axios = axios
5 app.mount('#app')
6
7 // 在组件中调用
8 this.$axios

```

## 网络请求基本示例

get请求

```

1 axios({
2     method: "get",
3     url:
4         "http://iwenwiki.com/api/blueberrypai/getChen
5         gpinDetails.php"
6     }).then(res => {
7         console.log(res.data);
8     })

```

## post请求

### 温馨提示

post请求参数是需要额外处理的

- ① 安装依赖: `npm install --save querystring`
- ② 转换参数格式: `qs.stringify({})`

```

1 axios({
2     method:"post",
3
4     url:"http://iwenwiki.com/api/blueberrypai/1
5     ogin.php",
6     data:qs.stringify({
7         user_id:"iwen@qq.com",
8         password:"iwen123",
9         verification_code:"crfvw"
10    })
11 }).then(res =>{
12     console.log(res.data);
13 })

```

## 快捷方案

### get请求

```

1 axios.get("http://iwenwiki.com/api/blueberryp
2     ai/getChengpinDetails.php")
3     .then(res =>{
4         console.log(res.data);
5     })

```

## post请求

```
1 axios.post("http://iwenwiki.com/api/blueberry  
2   pai/login.php", qs.stringify({  
3     user_id: "iwen@qq.com",  
4     password: "iwen123",  
5     verification_code: "crfvw"  
6   }))  
7   .then(res => {  
8     console.log(res.data);  
9   })
```

## Axios网络请求封装



在日常应用过程中，一个项目中的网络请求会很多，此时一般采取的方案是将网络请求封装起来

在 `src` 目录下创建文件夹 `utils`，并创建文件 `request`，用来存储网络请求对象 `axios`

```
1 import axios from "axios"
2 import qs from "querystring"
3
4
5 const errorHandle = (status,info) => {
6     switch(status){
7         case 400:
8             console.log("语义有误");
9             break;
10        case 401:
11            console.log("服务器认证失败");
12            break;
13        case 403:
14            console.log("服务器拒绝访问");
15            break;
16        case 404:
17            console.log("地址错误");
18            break;
19        case 500:
20            console.log("服务器遇到意外");
21            break;
22        case 502:
23            console.log("服务器无响应");
24            break;
25        default:
26            console.log(info);
27            break;
28    }
}
```

```
29 }
30
31
32 const instance = axios.create({
33   timeout:5000
34 })
35
36 instance.interceptors.request.use(
37   config =>{
38     if(config.method === "post"){
39       config.data =
40         qs.stringify(config.data)
41     }
42     return config;
43   },
44   error => Promise.reject(error)
45 )
46
47 instance.interceptors.response.use(
48   response => response.status === 200 ?
49     Promise.resolve(response) :
50     Promise.reject(response),
51   error =>{
52     const { response } = error;
53     errorHandle(response.status,response.info)
54   }
55 )
56
57
58 export default instance;
```

在 `src` 目录下创建文件夹 `api`，并创建文件 `index` 和 `path` 分别用来存放网络请求方法和请求路径

```

1 // path.js
2 const base = {
3     baseUrl:"http://iwenwiki.com",
4
5     chengpin:"/api/blueberrypai/getChengpinDetails.php"
6 }
7 export default base

```

```

1 // index.js
2 import path from "./path"
3 import axios from "../utils/request"
4
5 export default {
6     getChengpin(){
7         return axios.get(path.baseUrl +
8             path.chengpin)
9     }
10 }

```

在组件中直接调用网络请求

```

1 import api from "../api/index"
2
3 api.getChengpin().then(res =>{
4     console.log(res.data);
5 })

```

## 网络请求跨域解决方案



JS采取的是同源策略

同源策略是浏览器的一项安全策略，浏览器只允许js 代码请求和当前所在服务器域名,端口,协议相同的数据接口上的数据,这就是同源策略.

也就是说，当协议、域名、端口任意一个不相同时，都会产生跨域问题，所以又应该如何解决跨域问题呢

### 跨域错误提示信息

```

✖ Access to XMLHttpRequest at 'http://iwenwiki.com/api/FingerUnion/list.php' from origin 'http://localhost:8080' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
✖ ▶ GET http://iwenwiki.com/api/FingerUnion/list.php                               xhr.js?66c5:220 ⓘ
net::ERR_FAILED 200
✖ ▶ Uncaught (in promise)                                                        localhost:1
AxiosError {message: 'Network Error', name: 'AxiosError', code: 'ERR_NETWORK', config: {...}, request: XMLHttpRequest, ...}

```

## 目前主流的跨域解决方案有两种：

- ① 后台解决：cors
- ② 前台解决：proxy

```
1 devServer: {  
2   proxy: {  
3     '/api': {  
4       target: '<url>',  
5       changeOrigin: true  
6     }  
7   }  
8 }
```

### 温馨提示

解决完跨域配置之后，要记得重启服务器才行哦！

## Vue引入路由配置



在Vue中，我们可以通过 `vue-router` 路由管理页面之间的关系

Vue Router 是 Vue.js 的官方路由。它与 Vue.js 核心深度集成，让用 Vue.js 构建单页应用变得轻而易举

## 在Vue中引入路由

第一步：安装路由 `npm install --save vue-router`

第二步：配置独立的路由文件

```

1 // index.js
2 import { createRouter, createWebHashHistory
3 } from 'vue-router'
4 import Homeview from '../views/Homeview.vue'
5
6 const routes = [
7   {
8     path: '/',
9     name: 'home',
10    component: HomeView
11  },
12  {
13    path: '/about',
14    name: 'about',
15  }
16 ]
  
```

```

14     component: () =>
15       import('../views/AboutView.vue')
16     }
17   ]
18
19   const router = createRouter({
20     history: createWebHashHistory(),
21     routes
22   })
23
24   export default router

```

### 第三步：引入路由到项目

```

1 // main.js
2 import router from './router'
3 app.use(router)

```

### 第四步：指定路由显示入口 <router-view/>

### 第五步：指定路由跳转

```

1 <router-link to="/">Home</router-link> |
2 <router-link to="/about">About</router-link>

```

## 路由传递参数



页面跳转过程中，是可以携带参数的，这也是很常见的业务

例如：在一个列表项，点击进入查看每个列表项的详情

第一步：在路由配置中指定参数的 `key`

```

1 {
2   path: "/list/:name",
3   name: "list",
4   component: () =>
5     import("../views/ListView.vue")

```

第二步：在跳转过程中携带参数

```

1 <li><router-link to="/list/内蒙">内蒙旅游十大景
  区</router-link></li>
2 <li><router-link to="/list/北京">北京旅游十大景
  区</router-link></li>
3 <li><router-link to="/list/四川">四川旅游十大景
  区</router-link></li>

```

第三步：在详情页面读取路由携带的参数

```

1 <p>{{ $route.params.name }}城市旅游景区详情</p>

```

## 嵌套路由配置

发放列表								
操作	优惠券发放ID	优惠券名称	发放时间	发放场景	发放数量	领取数量	有效时间	状态
1 编辑 停用 详情	30	测试活动	2020-02-17 21:13:4	活动领取	100	0	永久有效	启用
2 编辑 启用 详情	29	180元优惠券	2020-01-06 15:43:1	活动领取	1000	0	永久有效	停用
3 详情	28	已突破	2020-01-06 18:54:1	课程打开	2	2	永久有效	停用
4 编辑 启用 详情	27	预发布活动3	2020-01-06 18:59:4	活动领取	100	2	永久有效	停用
5 编辑 启用 详情	26	预发布活动2	2020-01-06 18:59:4	活动领取	100	2	永久有效	停用
6 编辑 启用 详情	25	预发布活动测试1	2020-01-06 18:28:4	活动领取	100	4	永久有效	停用
7 编辑 停用 详情	24	最新banner优惠券	2020-01-06 18:25:4	知药学院banner图	4	2	永久有效	启用
8 详情	23	7QWE	2020-01-06 00:00:1	活动领取	11	1	永久有效	停用
9 详情	22	SDAO	2020-01-05 00:00:1	消息推送	0	0	2020-01-05 - 2020-01-06	停用
10 详情	21	32019	2020-01-03 00:00:1	消息推送	0	4	2020-01-03 - 2020-01-04	停用
11 编辑 停用 详情	20	32019	2020-01-03 21:59:4	消息推送	10	4	2020-01-03 - 2020-01-04	启用
12 详情	19	恰恰游戏福利家	2020-01-03 21:52:4	活动领取	4	4	永久有效	停用
13 编辑 启用 详情	18	恰恰游戏福利	2020-01-03 21:49:4	知药学院banner图	14	3	永久有效	停用
14 编辑 启用 详情	17	王坤健	2020-01-03 21:02:4	活动领取	18	7	永久有效	停用
15 详情	16	王坤健	2020-01-03 21:03:4	活动领取	4	4	永久有效	停用

路由嵌套是非常常见的需求

第一步：创建子路由要加载显示的页面

第二步：在路由配置文件中添加子路由配置

```

1 {
2   path: '/news',
3   name: 'news',
4   redirect: '/news/baidu',

```

```

5   component: () =>
6     import("../views/NewsView.vue"),
7     children: [
8       {
9         path: "baidu",
10        component: () =>
11          import("../views/NewsList/BaiduNews.vue"),
12        },
13        {
14          path: "wangyi",
15          component: () =>
16            import("../views/NewsList/WangyiNews.vue"),
17          }
18      ]
19    }
20  }

```

第三步：指定子路由显示位置 `<router-view></router-view>`

第四步：添加子路由跳转链接

```

1 <router-link to="/news/baidu">百度新闻
2   </router-link> |
3 <router-link to="/news/wangyi">网易新闻
4   </router-link>

```

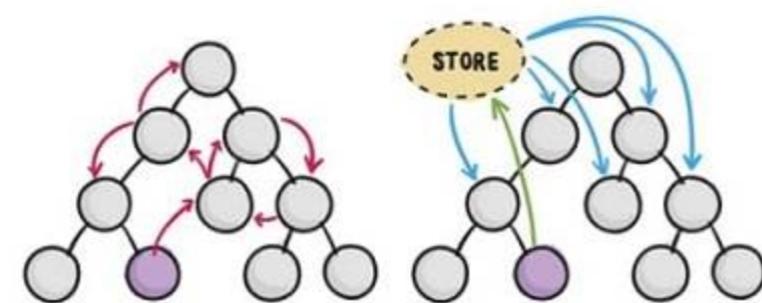
第五步：重定向配置 `redirect:"/news/baidu"`

## Vue状态管理(Vuex)



Vuex 是一个专为 Vue.js 应用程序开发的**状态管理模式 + 库**。它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。

简单来说，状态管理可以理解成为了更方便的管理组件之间的数据交互，提供了一个集中式的管理方案，任何组件都可以按照指定的方式进行读取和改变数据



## 引入Vuex的步骤

第一步：安装Vuex `npm install --save vuex`

第二步：配置Vuex文件

```

1 import { createStore } from 'vuex'
2
3 export default createStore({
4   state: {
5     counter: 0
6   }
7 })

```

第三步：在主文件中引入Vuex

```

1 import store from './store'
2 app.use(store)

```

第四步：在组件中读取状态

```

1 <p>counter: {{ $store.state.counter }}</p>
2 // 或者
3 import { mapState } from 'vuex';
4 computed: {
5   ...mapState(['counter'])
6 }

```



最常用的核心概念包含: `State`、`Getter`、`Mutation`、`Action`

## Getter

对Vuex中的数据进行过滤

```
1 import { createStore } from 'vuex'  
2  
3 export default createStore({  
4   state: {  
5     counter: 0  
6   },  
7   getters: {  
8     getCount(state){  
9       return state.counter > 0 ?  
10      state.counter : "counter小于0, 不符合要求"  
11    }  
12  }  
13})
```

```

1 import { mapState, mapGetters } from 'vuex';
2 computed: {
3     ...mapGetters(['getCount'])
4 }

```

## Mutation

更改 Vuex 的 store 中的状态的唯一方法是提交 mutation。Vuex 中的 mutation 非常类似于事件：每个 mutation 都有一个字符串的事件类型 (type) 和一个回调函数 (handler)。这个回调函数就是我们实际进行状态更改的地方，并且它会接受 state 作为第一个参数

```

1 import { createStore } from 'vuex'
2
3 export default createStore({
4     state: {
5         counter: 0
6     },
7     getters: {
8     },
9     mutations: {
10         setCounter(state, num) {
11             state.counter += num
12         }
13     }
14 })

```

```

1 import { mapState, mapMutations } from
2   'vuex';
3
4 methods: {
5   ...mapMutations(["setCounter"]),
6   clickHandler() {
7     // this.$store.commit("setCounter", 20)
8     // 或者
9     // this.setCounter(10)
10 }
11 }

```

## Action

Action 类似于 mutation，不同在于：

- Action 提交的是 mutation，而不是直接变更状态
- Action 可以包含任意异步操作

```

1 import { createStore } from 'vuex'
2 import axios from "axios"
3
4 export default createStore({
5   state: {
6     counter: 0
7   },
8   getters: {
9     getCount(state) {
10       return state.counter > 0 ?
11         state.counter : "counter小于0, 不符合要求"
12     }
13   },
14   mutations: {
15     ...
16   }
17 })

```

```

14     setCounter(state, num) {
15         state.counter += num
16     }
17 },
18 actions: {
19     asyncSetCount({ commit }) {
20
21         axios.get("http://iwenwiki.com/api/generator/list.php")
22             .then(res =>{
23                 commit("setCounter", res.data[0])
24             })
25     }
26 }
27

```

```

1 import {
2     mapState, mapMutations, mapGetters, mapActions
3 } from 'vuex';
4
5 methods: {
6     ...mapActions(["asyncSetCount"]),
7     clickAsyncHandler() {
8         // ...
9         this.$store.dispatch("asyncSetCount")
10        // 或者
11        // this.asyncSetCount()
12    }
13 }

```

## Vue3新特性1



### 易用

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

### 灵活

不断繁荣的生态系统，可以在一个库  
和一套完整框架之间自如伸缩。

### 高效

20kB min+gzip 运行大小  
超快虚拟 DOM  
最省心的优化

Vue3是目前Vue的最新版本，自然也是新增了很多新特性

## 六大亮点

- Performance: 性能更比Vue 2.0强。
- Tree shaking support: 可以将无用模块“剪辑”，仅打包需要的。
- **Composition API: 组合API**
- Fragment, Teleport, Suspense: “碎片”，Teleport即Portal传送门，“悬念”
- Better TypeScript support: 更优秀的Ts支持
- Custom Renderer API: 暴露了自定义渲染API

## ref或者reactive

在2.x中通过组件data的方法来定义一些当前组件的数据

```

1  data() {
2      return {
3          name: 'iwen',
4          list: [],
5      }
6  }

```

在3.x中通过ref或者reactive创建响应式对象

```

1 import { ref,reactive } from "vue"
2 export default {
3     name: 'HelloWorld',
4     setup(){
5         const name = ref("iwen")
6         const state = reactive({
7             list:[]
8         })
9
10        return{
11            name,
12            state
13        }
14    }
15 }

```

## methods中定义的方法写在setup()

在2.x中methods来定义一些当前组件内部方法

```
1 | methods: {
2 |     http() {}
3 | }
```

在3.x中直接在setup方法中定义并return

```
1 | setup() {
2 |     const http = ()=>{
3 |         // do something
4 |     }
5 |     return {
6 |         http
7 |     };
8 | }
```

## setup()中使用props和context

在2.x中，组件的方法中可以通过this获取到当前组件的实例，并执行data变量的修改，方法的调用，组件的通信等等，但是在3.x中，setup()在beforeCreate和created时机就已调用，无法使用和2.x一样的this，但是可以通过接收setup(props,ctx)的方法，获取到当前组件的实例和props

```

1 export default {
2   props: {
3     name: String,
4   },
5   setup(props, ctx) {
6     console.log(props.name)
7     ctx.emit('event')
8   },
9 }

```

## Vue3新特性2



### 易用

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

### 灵活

不断繁荣的生态系统，可以在一个库  
和一套完整框架之间自如伸缩。

### 高效

20kB min+gzip 运行大小  
超快虚拟 DOM  
最省心的优化

## 在setup中使生命周期函

你可以通过在生命周期钩子前面加上“on”来访问组件的生命周期钩子。

下表包含如何在 setup () 内部调用生命周期钩子

Options API	Hook inside setup
beforeCreate	Not needed*
created	Not needed*
beforeMount	onBeforeMount
mounted	onMounted
beforeUpdate	onBeforeUpdate
updated	onUpdated
beforeUnmount	onBeforeUnmount
unmounted	onUnmounted

```

1 export default {
2   setup() {
3     // mounted
4     onMounted(() => {
5       console.log('Component is mounted!')
6     })
7   }
8 }
```

## Provide / Inject

- provide() 和 inject() 可以实现嵌套组件之间的数据传递。
- 这两个函数只能在 setup() 函数中使用。
- 父级组件中使用 provide() 函数向下传递数据。
- 子级组件中使用 inject() 获取上层传递过来的数据。
- 不限层级

```

1 // 父组件
2 import { provide } from "vue"
3
4 setup() {
5     provide("customVal", "我是父组件向子组件传递
6         的值");
7 }

```

```

1 // 子组件
2 import { inject } from "vue"
3
4 setup() {
5     const customVal = inject("customVal");
6     return {
7         customVal
8     }
9 }

```

## Fragment

Fragment翻译为：“碎片”

- 不再限于模板中的单个根节点

```

1 <template>
2     
4     <HelloWorld msg="welcome to Your Vue.js
5         App" />
6 </template>

```

## Vue3加载Element-plus



Element，一套为开发者、设计师和产品经理准备的基于 **Vue 2.0** 的桌面端组件库

Element Plus 基于 **Vue 3**，面向设计师和开发者的组件库

## 安装Element-Plus

```
1 | npm install element-plus --save
```

## 完整引用

如果你对打包后的文件大小不是很在乎，那么使用完整导入会更方便

```
1 import { createApp } from 'vue'
2 import ElementPlus from 'element-plus'
3 import 'element-plus/dist/index.css'
4 import App from './App.vue'
5
6 const app = createApp(App)
7
8 app.use(ElementPlus)
9 app.mount('#app')
```

## 按需导入

按需导入才是我们的最爱，毕竟在真实的应用场景中并不是每个组件都会用到，这会造成不小的浪费

首先你需要安装 `unplugin-vue-components` 和 `unplugin-auto-import` 这两款插件

```
1 | npm install -D unplugin-vue-components
  unplugin-auto-import
```

然后修改 `vue.config.js` 配置文件

```

1 const { defineConfig } = require('@vue/cli-service')
2 const AutoImport = require('unplugin-auto-import/webpack')
3 const Components = require('unplugin-vue-components/webpack')
4 const { ElementPlusResolver } =
require('unplugin-vue-components/resolvers')

5
6 module.exports = defineConfig({
7   transpileDependencies: true,
8   configureWebpack: {
9     plugins: [
10       AutoImport({
11         resolvers: [ElementPlusResolver()]
12       }),
13       Components({
14         resolvers: [ElementPlusResolver()]
15       })
16     ]
17   }
18 })

```

最后，可以直接在组件中使用

```

1 <template>
2   <el-button>Default</el-button>
3   <el-button type="primary">Primary</el-
button>
4 </template>

```

## 实时效果反馈

1. 在Vue3项目中引入饿了么UI组件库，下列命令正确的是：

- A `npm install --save element-iu`
- B `vue add element`
- C `npm install element-plus --save`
- D `vue add element-plus`

## 答案

1=>C

Vue3加载Element-plus的字体图标

⊕ AddLocation	⊕ Aim	⌚ AlarmClock	🍏 Apple	⌄ ArrowDownBold	⌄ ArrowDown	◀ ArrowLeftBold
< ArrowLeft	> ArrowRightBold	> ArrowRight	^K ArrowUpBold	^K ArrowUp	👤 Avatar	⬅ Back
⚾ Baseball	🏀 Basketball	🔔 BellFilled	⚠ Bell	🚲 Bicycle	↙ BottomLeft	↘ BottomRight
⬇ Bottom	🍽 Bowl	📦 Box	💼 Briefcase	🖌 BrushFilled	🖌 Brush	🍔 Burger
📅 Calendar	📷 CameraFilled	📸 Camera	▼ CaretBottom	◀ CaretLeft	▶ CaretRight	▲ CaretTop

Element-plus 不仅仅是提供了各种组件，同时还提供了一整套的字体图标方便开发者使用

## 安装 icons 字体图标

```
1 | npm install @element-plus/icons-vue
```

## 全局注册

在项目根目录下，创建 `plugins` 文件夹，在文件夹下创建文件 `icons.js` 文件

```

1 import * as components from "@element-
plus/icons-vue";
2 export default {
3     install: (app) => {
4         for (const key in components) {
5             const componentConfig =
6                 components[key];
7
8             app.component(componentConfig.name,
9                 componentConfig);
10        }
11    },
12 };

```

## 引入文件

在 `main.js` 中引入 `icons.js` 文件

```

1 import elementIcon from "./plugins/icons";
2 app.use(elementIcon)

```

## 使用方式

接下来就可以直接在组件中引入使用了

```

1 <el-icon class="expand" color="#409EFC"
2   :size="30">
3     <expand />
4   </el-icon>

```

