



讲师：李振良（阿良）

今天课题：《Vue前端开发》

学院官网：[www.aliangedu.cn](http://www.aliangedu.cn)



阿良个人微信



DevOps技术栈公众号

# Vue前端开发（下篇）

- ❖ Vue常用指令之数据双向绑定
- ❖ 实例生命周期钩子
- ❖ Vue Cli脚手架
- ❖ Vue组件
- ❖ 前后端数据交互：Axios
- ❖ Vue路由：Vue Router
- ❖ Vue UI组件库：Element Plus

# Vue常用指令之数据双向绑定

- v-model
- 常用指令总结

双向数据绑定：通过前面学习知道Vue是数据驱动的，数据驱动有一个精髓之处是数据双向绑定，即当数据发生变化的时候，视图也就发生变化，当视图发生变化的时候，数据也会跟着同步变化。

## v-model: 表单输入

v-model指令提供表单输入绑定，可以在 `<input>`、`<textarea>` 及 `<select>` 元素上创建双向数据绑定。

```
<div id="hello-vue">
  <input type="text" v-model="msg">
  <p>{{ msg }}</p>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
      return {
        msg: "hello vue!"
      }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

- v-model指令其实是一个语法糖，背后本质上包含v-bind和v-on两个操作。

# v-model: 单选

单选框 (radio) :

```
<div id="hello-vue">
  <input type="radio" name="go" value="Go" v-model="msg">Go<br>
  <input type="radio" name="vue" value="Vue" v-model="msg">Vue
  <p>选择: {{ msg }}</p>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
      return {
        msg: ""
      }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

# v-model: 多选框

多选框 (select) :

```
<div id="hello-vue">
  <select v-model="selected">
    <option value="" disabled>请选择</option>
    <option value="go">Go</option>
    <option value="vue">Vue</option>
    <option value="k8s">K8s</option>
  </select>
  <p>选择: {{ selected }}</p>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
      return {
        selected: ""
      }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

# v-model: 登录案例

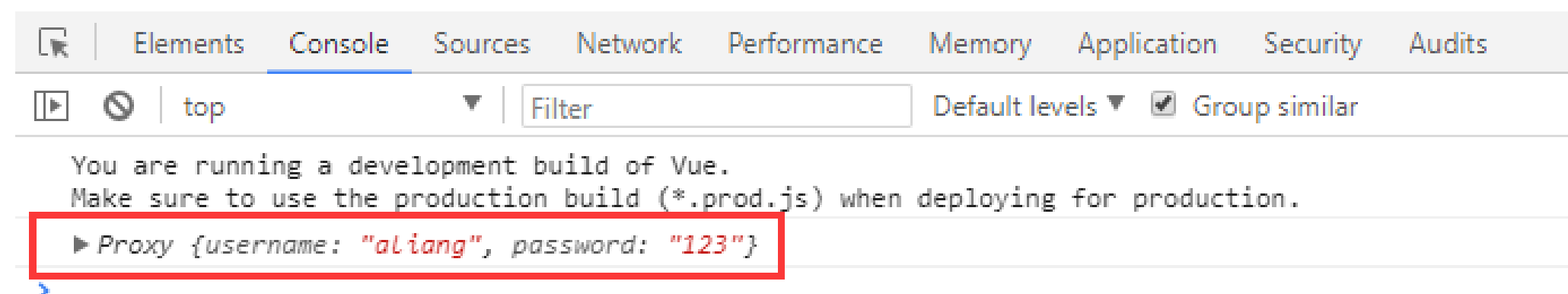
登录案例：获取用户输入用户名和密码

```
<div id="hello-vue">
  <h1>欢迎访问管理后台</h1>
  <form action="#">
    用户名: <input type="text" v-model="form.username">
    密 码: <input type="text" v-model="form.password">
    <button @click="loginBtn">登录</button>
  </form>
  <p style="color: red;" v-if="notice">用户名或者密码不能为空! </p>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
      return {
        form: {
          username: '',
          password: ''
        },
        notice: false
      }
    },
    methods: {
      loginBtn() {
        if (this.form.username == '' || this.form.password == '') {
          this.notice = true;
        } else {
          this.notice = false;
          console.log(this.form) // 获取输入用户名和密码 (提交到服务端)
        }
      }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

## 欢迎访问管理后台

用户名:  密 码:

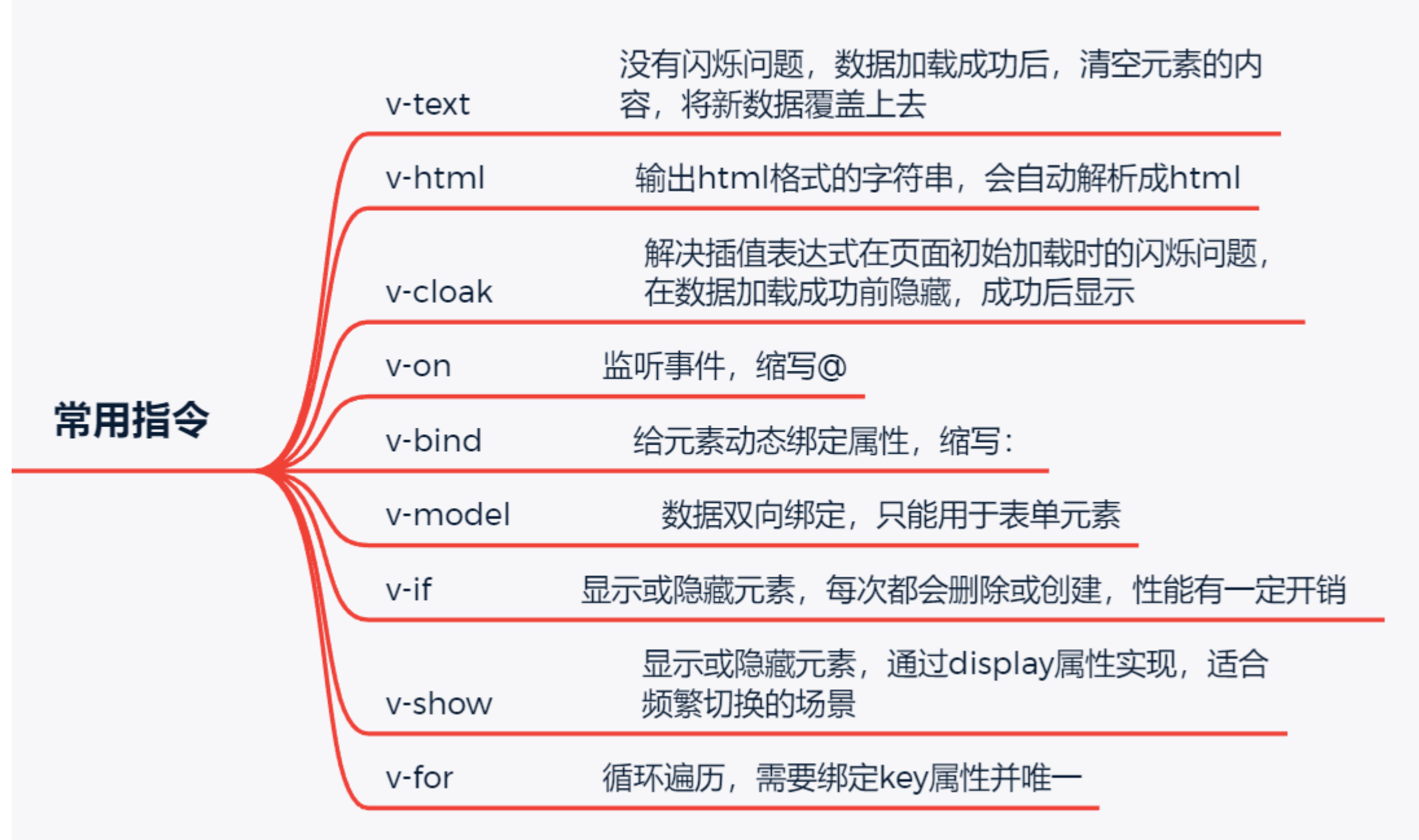
开发者工具 - file:///C:/Users/lizhenliang/Desktop/index.html



浏览器开发工具验证



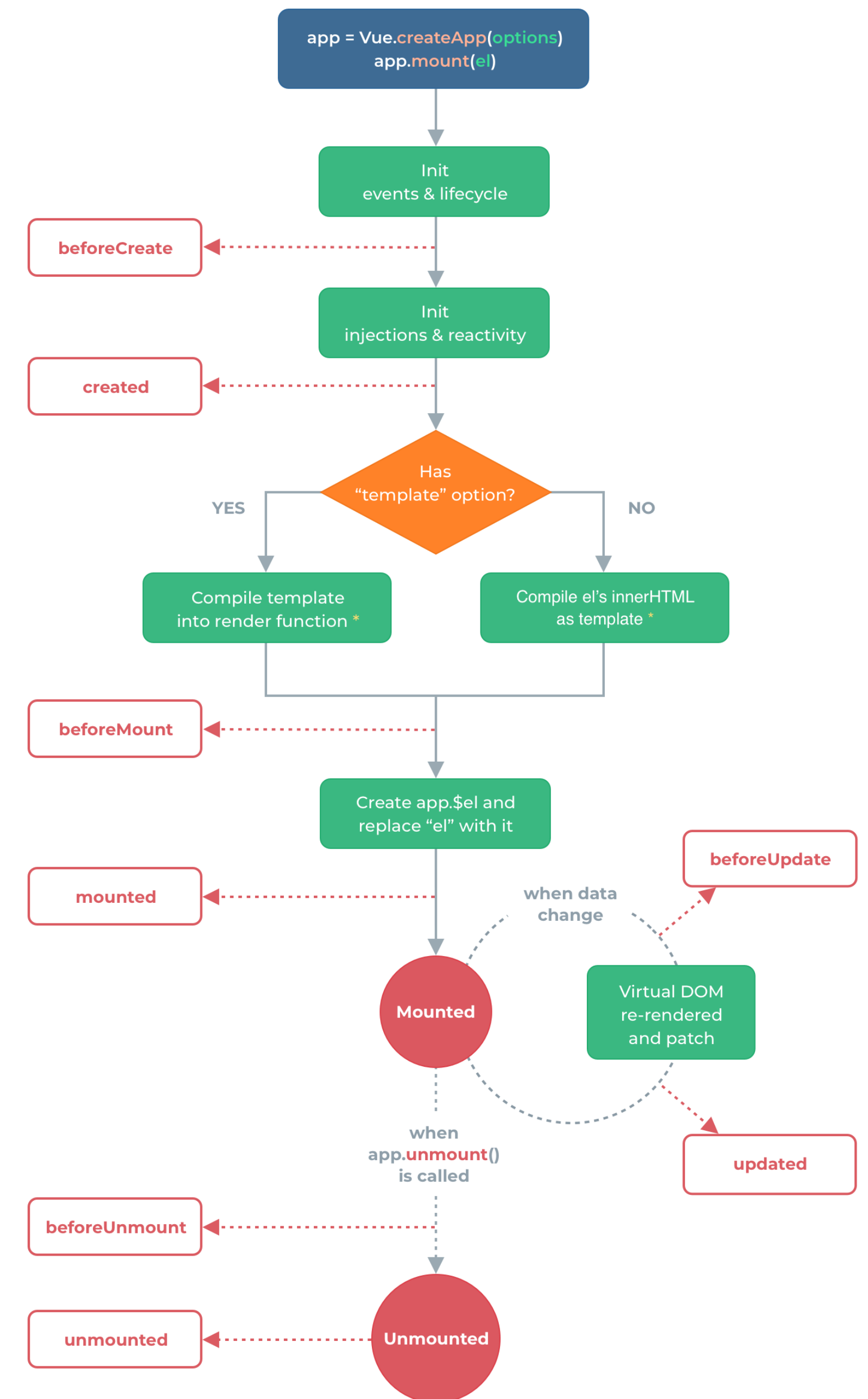
# 常用指令总结



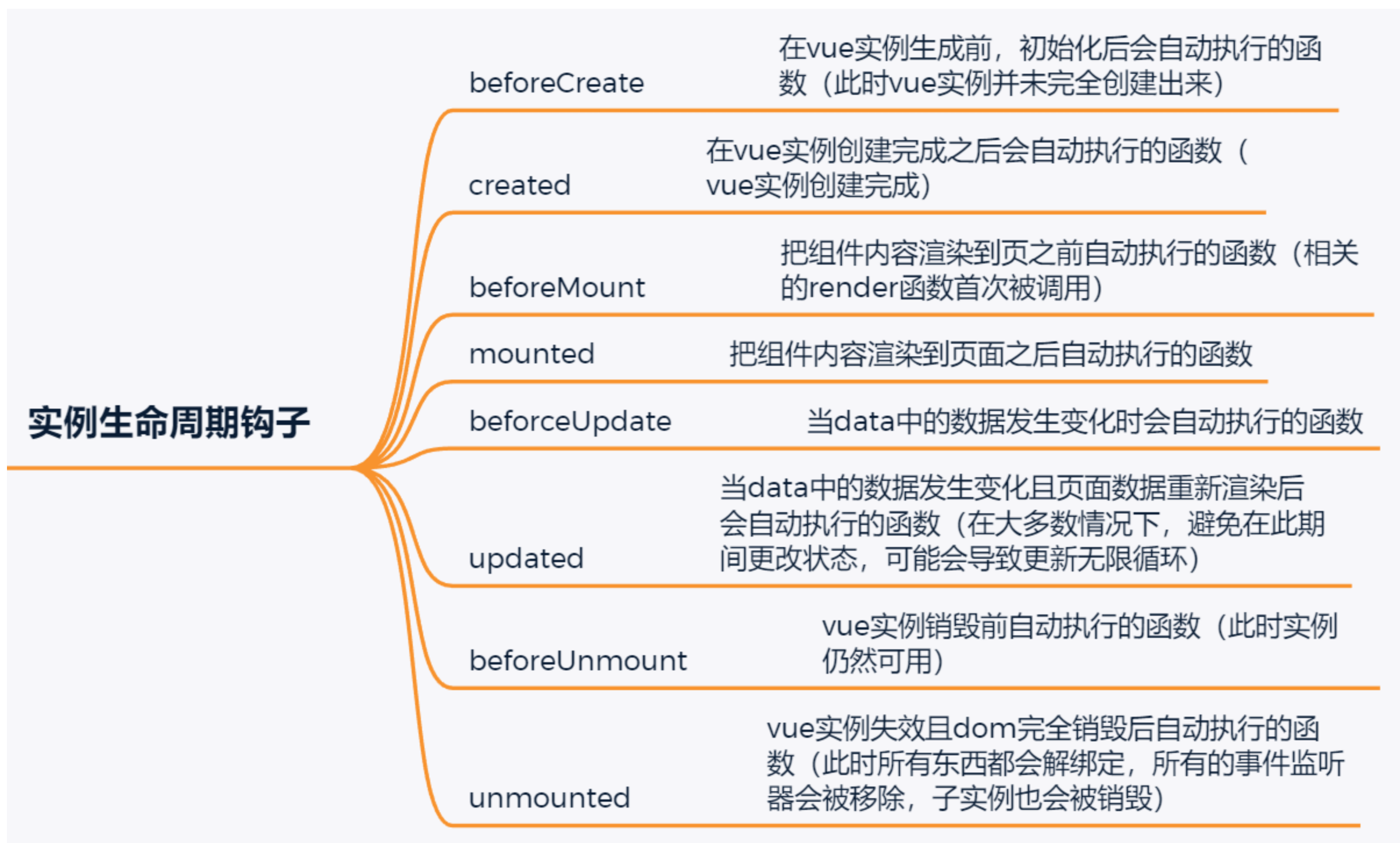
# 实例生命周期钩子

# 生命周期钩子

生命周期是指Vue实例从创建到销毁的过程。就是vue实例从开始创建、初始化数据、编译模板、挂载Dom、渲染->更新->渲染、卸载等一系列过程，在vue生命周期中提供了一系列的生命周期函数，如图所示。



# 生命周期钩子



# VueCli 脚手架

# Vue Cli 脚手架介绍

到目前为止，已经会了Vue基本使用，但这种在HTML引用Vue.js的方式，简单的页面还是没问题的，如果用Vue开发整个前端项目，组建Vue项目结构及配置还是比较复杂的，例如引入各种js文件、打包上线等。因此，为了提高开发效率，官方开发了VueCli脚手架快捷搭建开发环境。

# Vue Cli 脚手架介绍

Vue CLI 是一个基于 Vue.js 进行快速开发的完整系统，提供：

- 通过 @vue/cli 实现的交互式的项目脚手架。
- 通过 @vue/cli + @vue/cli-service-global 实现的零配置原型开发。
- 一个运行时依赖 (@vue/cli-service)，该依赖：
  - 可升级；
  - 基于 webpack 构建，并带有合理的默认配置；
  - 可以通过项目内的配置文件进行配置；
  - 可以通过插件进行扩展。
- 一个丰富的官方插件集合，集成了前端生态中最好的工具。
- 一套完全图形化的创建和管理 Vue.js 项目的用户界面。

Vue CLI 致力于将 Vue 生态中的工具基础标准化。它确保了各种构建工具能够基于智能的默认配置即可平稳衔接，这样你可以专注在撰写应用上，而不必花好几天去纠结配置的问题。

在使用Vue Cli之前，需先了解一些关于NPM的知识点：

- **NPM (Node Package Manager, Node包管理器)**，存放JavaScript代码共享中心，是目前最大的JavaScript仓库。类似于Linux yum仓库。
- 可能你会联想到Node.js，Node.js是服务端的JavaScript，类似于Gin、Django，NPM是基于Node.js开发的软件。
- 随着Node.js兴起，生态圈的JS库都纷纷向NPM官方仓库发布，所以现在，大都是使用npm install命令来安装JS库，而不必再去它们官网下载了。



# 认识NPM

安装Node.js，默认已经内置npm，下载对应软件包直接安装即可。<http://nodejs.cn/download/>

命令	描述
npm -v	查看版本
npm install <模块名>	安装模块
npm install -g <模块名>	可以直接在命令行里使用
npm list -g	查看所有全局安装的模块
npm list <模块名>	查看某个模块的版本号
npm install -g <模块名> @<版本号>	更新模块版本
npm install --save <模块名>	在package.json文件中写入依赖 (npm5版本之前需要指定，之后版本 无需再加--save选项)
npm config	管理npm的配置路径
npm run serve npm run build	运行项目 打包项目

常用命令列表

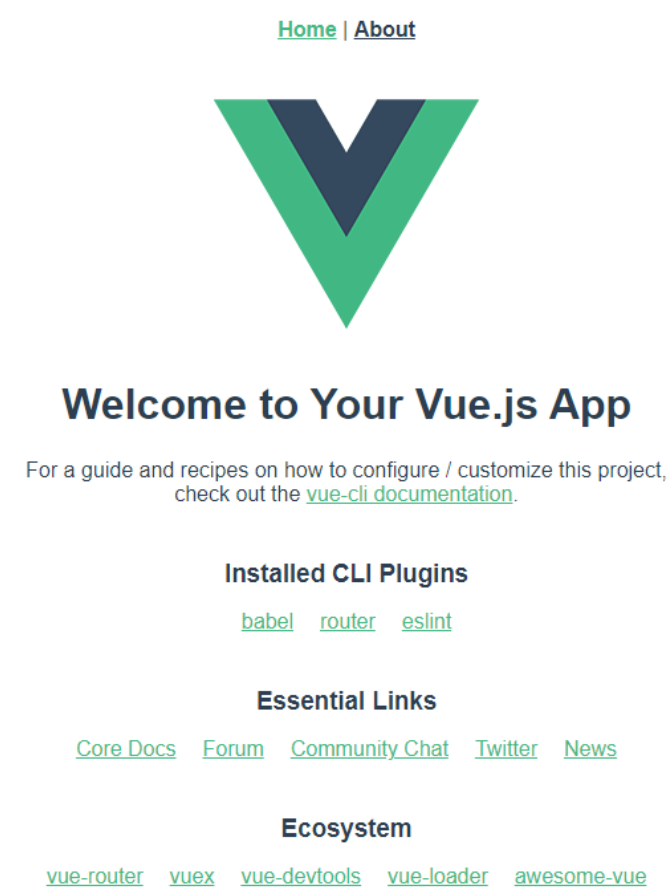
配置淘宝npm仓库：

```
npm config set registry https://registry.npm.taobao.org --global
npm config get registry
```

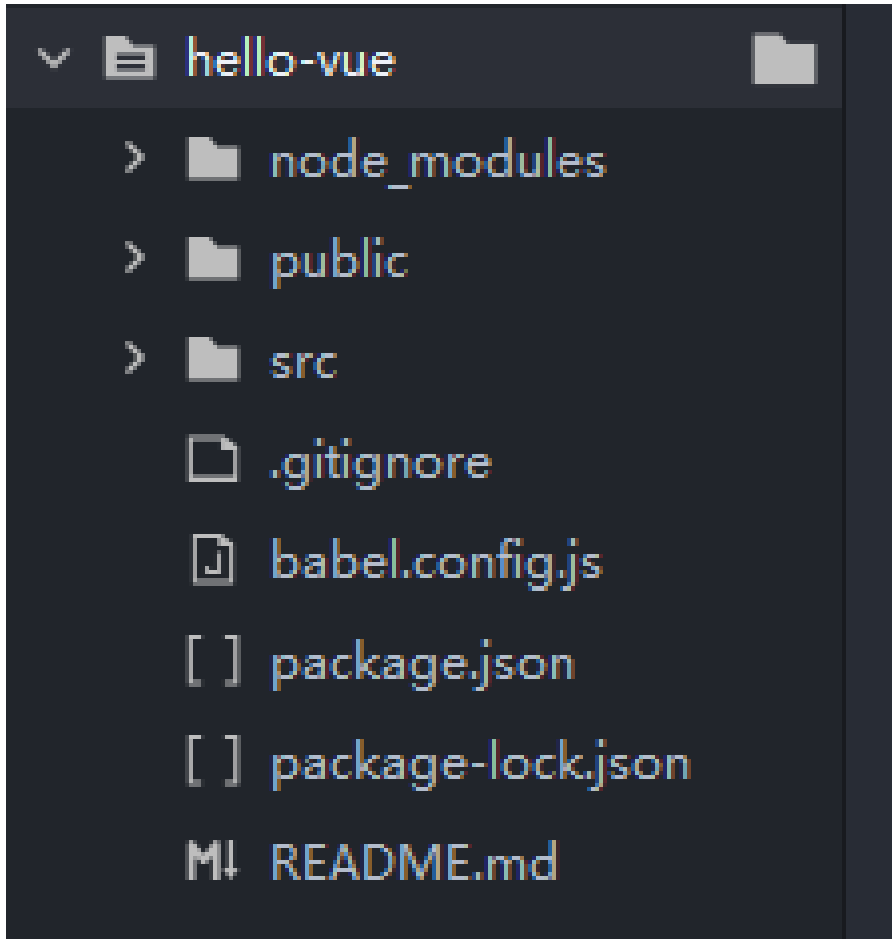
# Vue Cli 脚手架使用

## Vue Cli脚手架使用步骤:

1. 命令安装: `npm install -g @vue/cli`
2. 检查版本: `vue -V`
3. 创建项目: `vue create <项目名称>`
4. 运行项目, 访问



# 项目目录



目录/文件	说明
node_modules	项目开发依赖的一些模块，不用管
public	主要存放首页、favicon
src	源码目录，这里是我们要开发的目录，基本上要做的事情都在这个目录里。 里面包含了几个目录及文件： <ul style="list-style-type: none"><li>● assets: 放入资源，例如图片、CSS等</li><li>● components: 公共组件目录</li><li>● routes: 前端路由</li><li>● store: 应用级数据（state）Vuex</li><li>● views: 单页面组件目录</li><li>● App.vue: 项目入口文件（根组件）</li><li>● main.js: 项目的全局配置，在任意一个文件中都有效的</li></ul>
.gitignore文件	git提交忽略文件
babel.config.js	babel配置，例如es5转es6
package.json	项目配置文件。 npm包配置文件，里面定义了项目的npm脚本，依赖包等信息
README.md	项目的说明文档， markdown 格式

# Vue 组件

- 介绍
- 文件格式
- 使用
- 注册
- 传参

# 组件：介绍

**组件：**一段独立的，能代表页面某一个部分的代码片段，拥有自己独立的数据、JavaScript脚本、以及CSS样式。

组件是可复用的Vue实例，在开发过程中可以把经常重复的功能，封装为组件，达到快捷开发的目的。

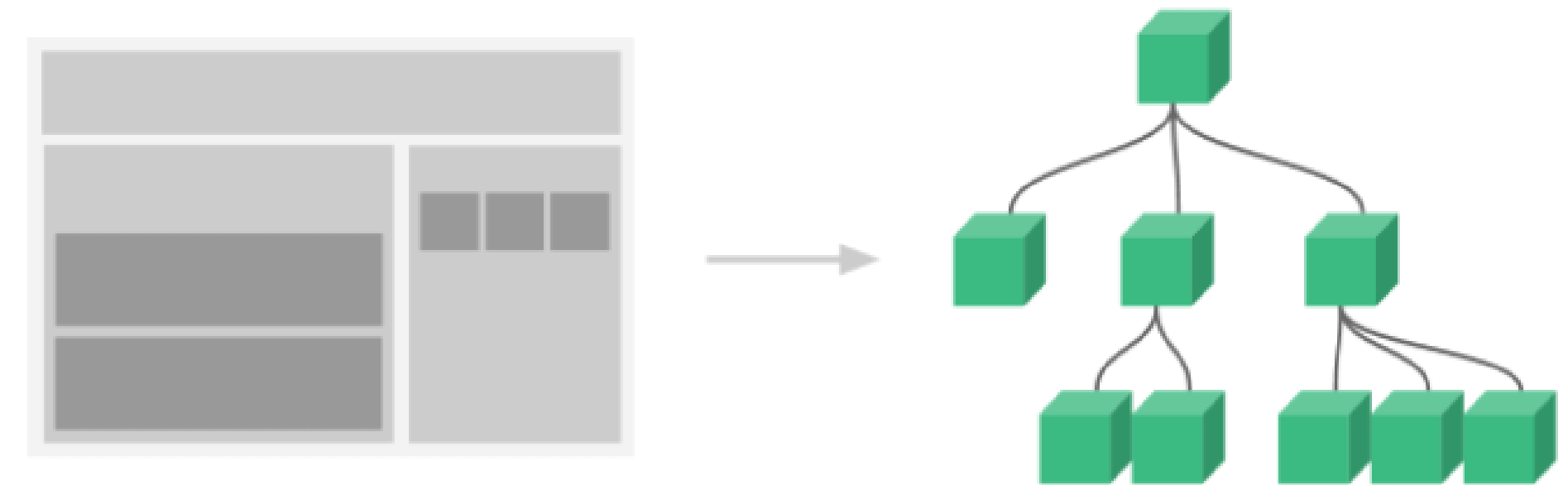
## 组件的好处：

- 提高开发效率
- 方便重复使用
- 易于管理和维护

## 组件：介绍

通常一个应用会以一棵嵌套的组件树的形式来组织，如图所示。

例如，你可能会有页头、侧边栏、内容区等组件，每个组件又包含了其它的像导航链接、博文之类的组件。



## 组件：文件格式

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，它允许讲Vue组件的模板、逻辑与样式封装在单个文件中。

正如所见，Vue SFC 是经典的 HTML、CSS 与 JavaScript 三个经典组合的自然延伸。每个 \*.vue 文件由三种类型的顶层代码块组成：<template>、<script> 与 <style>：

- <template> 部分定义了组件的模板。
- <script> 部分是一个标准的 JavaScript 模块。它应该导出一个 Vue 组件定义作为其默认导出。
- <style> 部分定义了与此组件关联的 CSS。

```
<template>
  <p class="greeting">{{ greeting }}</p>
</template>

<script>
  export default {
    data() {
      return {
        greeting: 'Hello World!'
      }
    }
  }
</script>

<style>
  .greeting {
    color: red;
    font-weight: bold;
  }
</style>
```

## 组件：使用

使用具体流程：

- 1、在src/components目录里开发一个组件文件（首字母大写）
- 2、在父组件里引用子组件 `import xxx from 'xxx'`
- 3、在默认导出里注册组件
- 4、在template模板里使用组件

开发组件



注册组件



使用组件



## 组件：注册组件

为了能在模板中使用，这些组件必须先注册以便 Vue 能够识别。这里有两种组件的注册类型：全局注册和局部注册。上述是局部注册，只能在当前模板中使用。

- **全局注册：**声明一次，在任何vue文件模板中使用，一般使用该组件的地方多时使用
- **局部注册：**在使用组件的vue文件中声明和使用，一般只需要解耦代码时使用

# 组件：注册组件

全局注册：在main.js文件

```
import { createApp } from "vue";
import App from "./App.vue";
import router from "./router";
import Test from './components/Test.vue' // 导入组件

const app = createApp(App)

app.use(router).mount("#app");
app.component('Test', Test) // 注册组件
```

全局注册后，在任意.vue文件里可使用该组件：  
views/Home.vue

```
<template>
  <div class="home">
    
    <!--引用组件-->
    <HelloWorld msg="Welcome to Your Vue.js App" />
  </div>
  <!--引用组件-->
  <Test></Test>
</template>

<script>
// @ is an alias to /src
import HelloWorld from "@components/HelloWorld.vue";

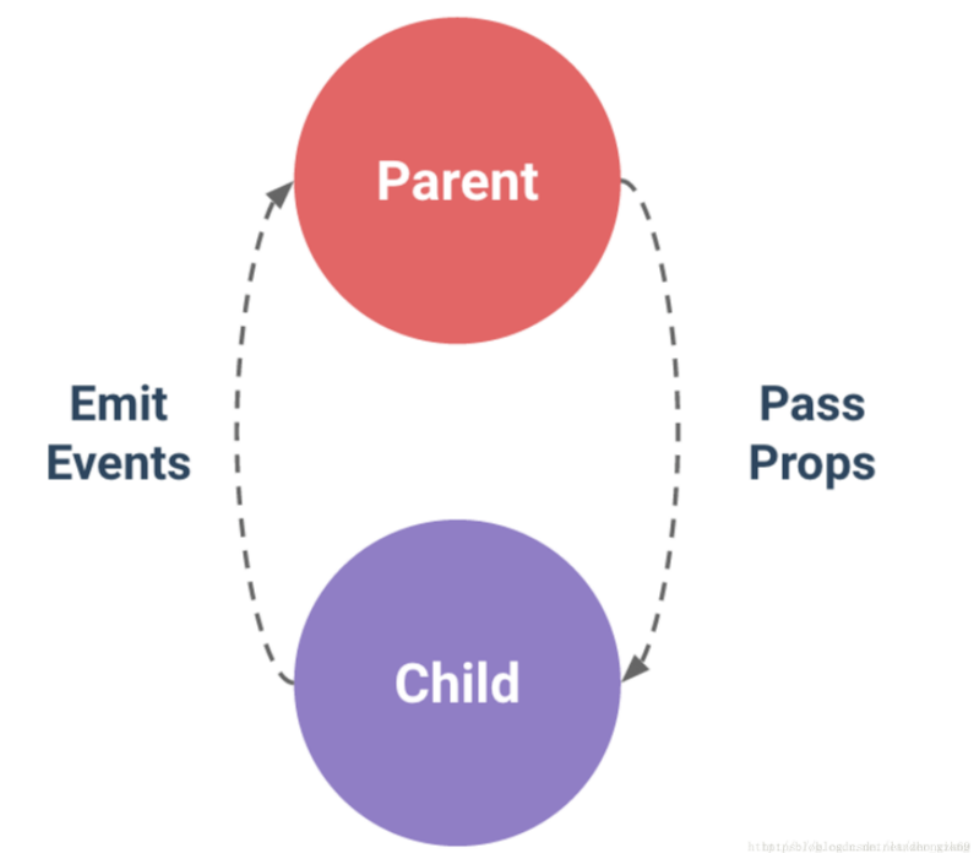
export default {
  name: "Home",
  components: {
    HelloWorld,
  },
};
</script>
```

# 组件：传参

学习了组件用法，就像一种嵌套引用关系，在这个关系中，经常会涉及相互传数据的需求，即父组件传子组件，子组件传父组件。

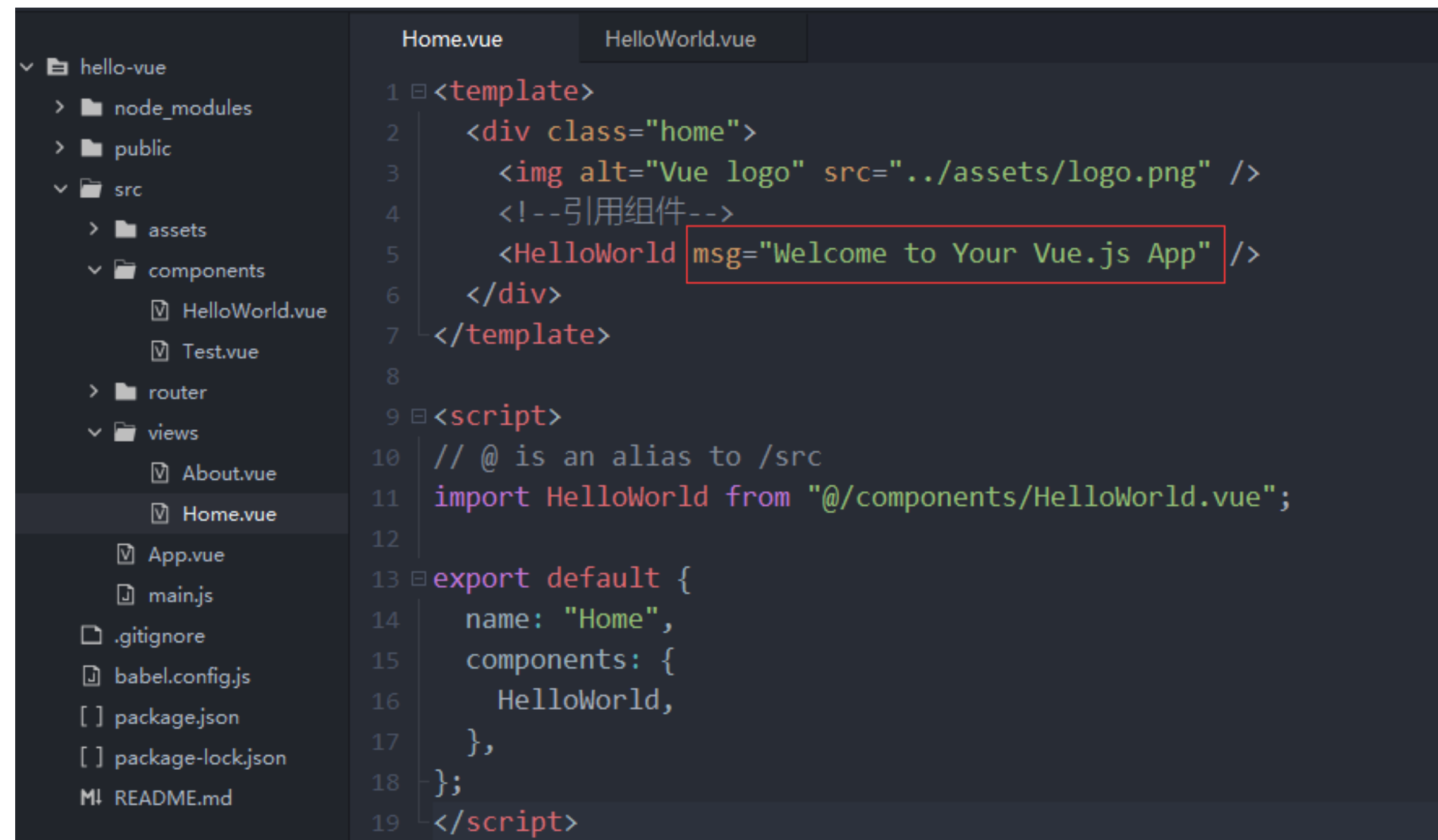
父、子组件的关系可以总结为：**prop 向下传递，事件向上传递。**

父组件通过 prop 给子组件下发数据，子组件通过事件给父组件发送消息，如右图所示：



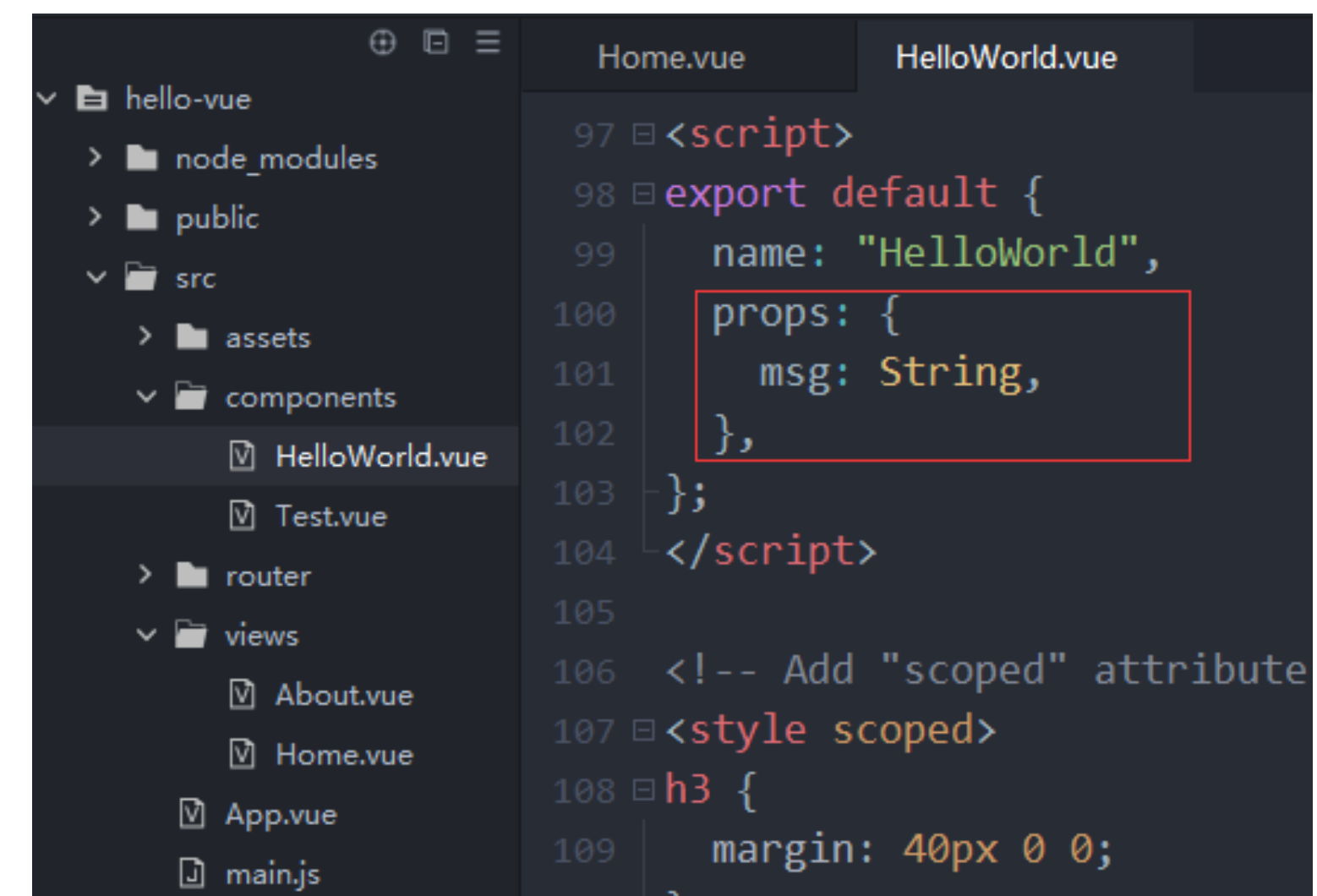
# 组件：传参

**父传子：**在默认页面中，也用到了父传子，在父组件Home.vue中给引用的组件传入一个静态的值，子组件通过props属性接收，并在模板中使用。



```
1 <template>
2   <div class="home">
3     
4     <!--引用组件-->
5     <HelloWorld msg="Welcome to Your Vue.js App" />
6   </div>
7 </template>
8
9 <script>
10  // @ is an alias to /src
11  import HelloWorld from "@components/HelloWorld.vue";
12
13  export default {
14    name: "Home",
15    components: {
16      HelloWorld,
17    },
18  };
19 </script>
```

父组件

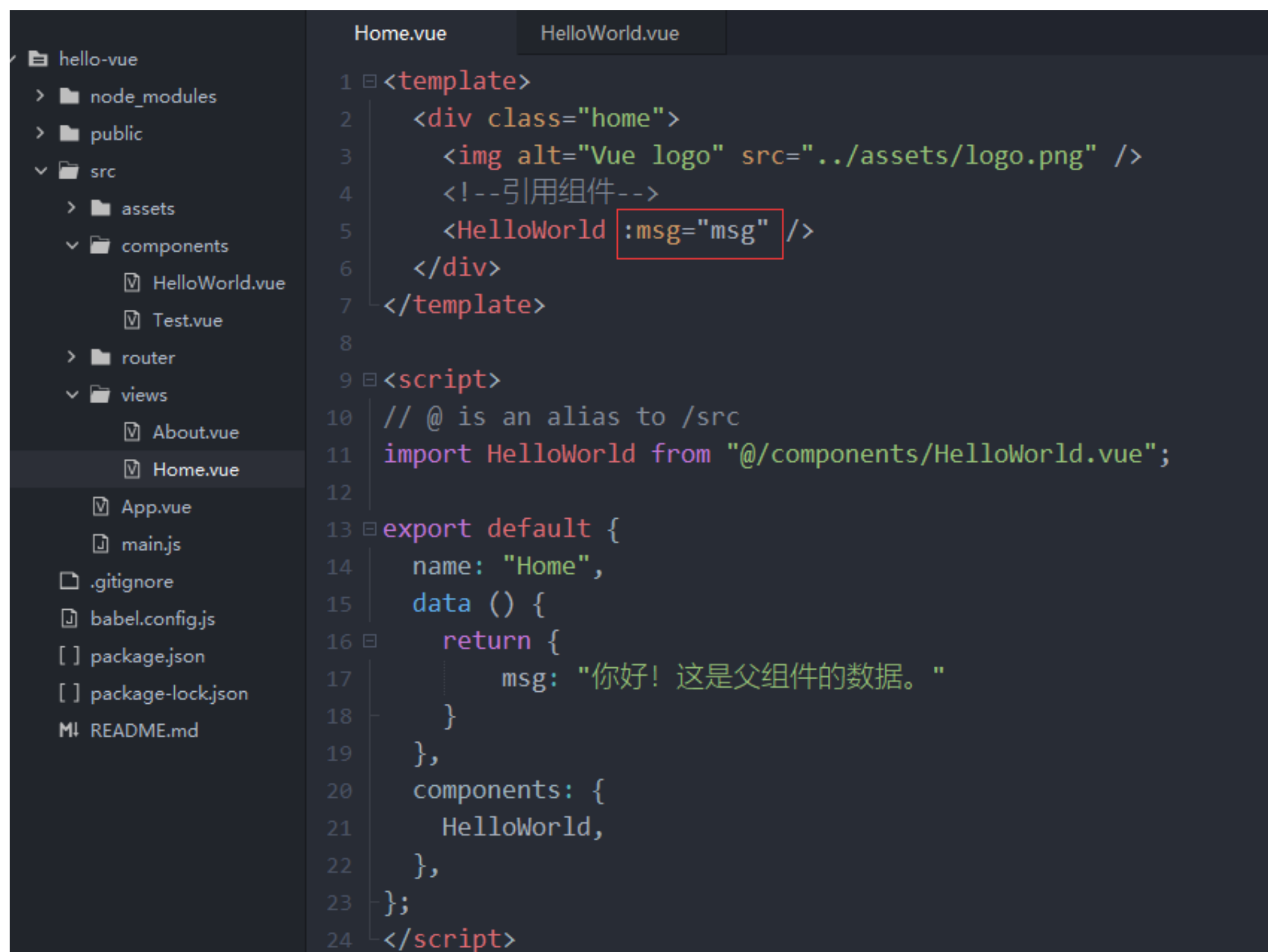


```
97 <script>
98  export default {
99    name: "HelloWorld",
100    props: {
101      msg: String,
102    },
103  };
104 </script>
105
106 <!-- Add "scoped" attribute
107 <style scoped>
108  h3 {
109    margin: 40px 0 0;
```

子组件

# 组件：传参

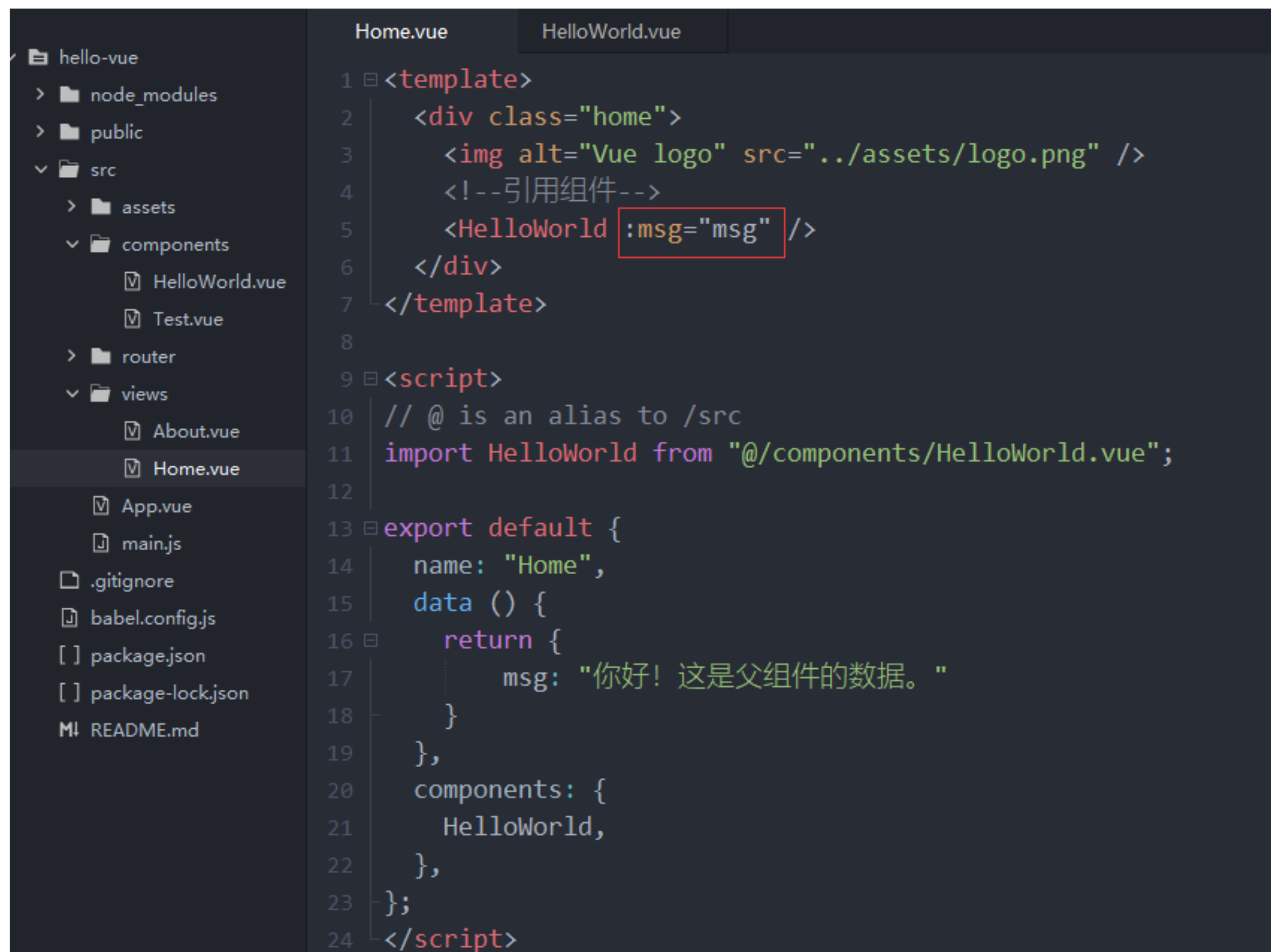
也可以通过v-bind或者简写：动态赋值，例如：



```
1 <template>
2   <div class="home">
3     
4     <!-- 引用组件 -->
5     <HelloWorld :msg="msg" />
6   </div>
7 </template>
8
9 <script>
10  // @ is an alias to /src
11  import HelloWorld from "@components/HelloWorld.vue";
12
13  export default {
14    name: "Home",
15    data () {
16      return {
17        msg: "你好! 这是父组件的数据。"
18      }
19    },
20    components: {
21      HelloWorld,
22    },
23  };
24 </script>
```

# 组件：传参

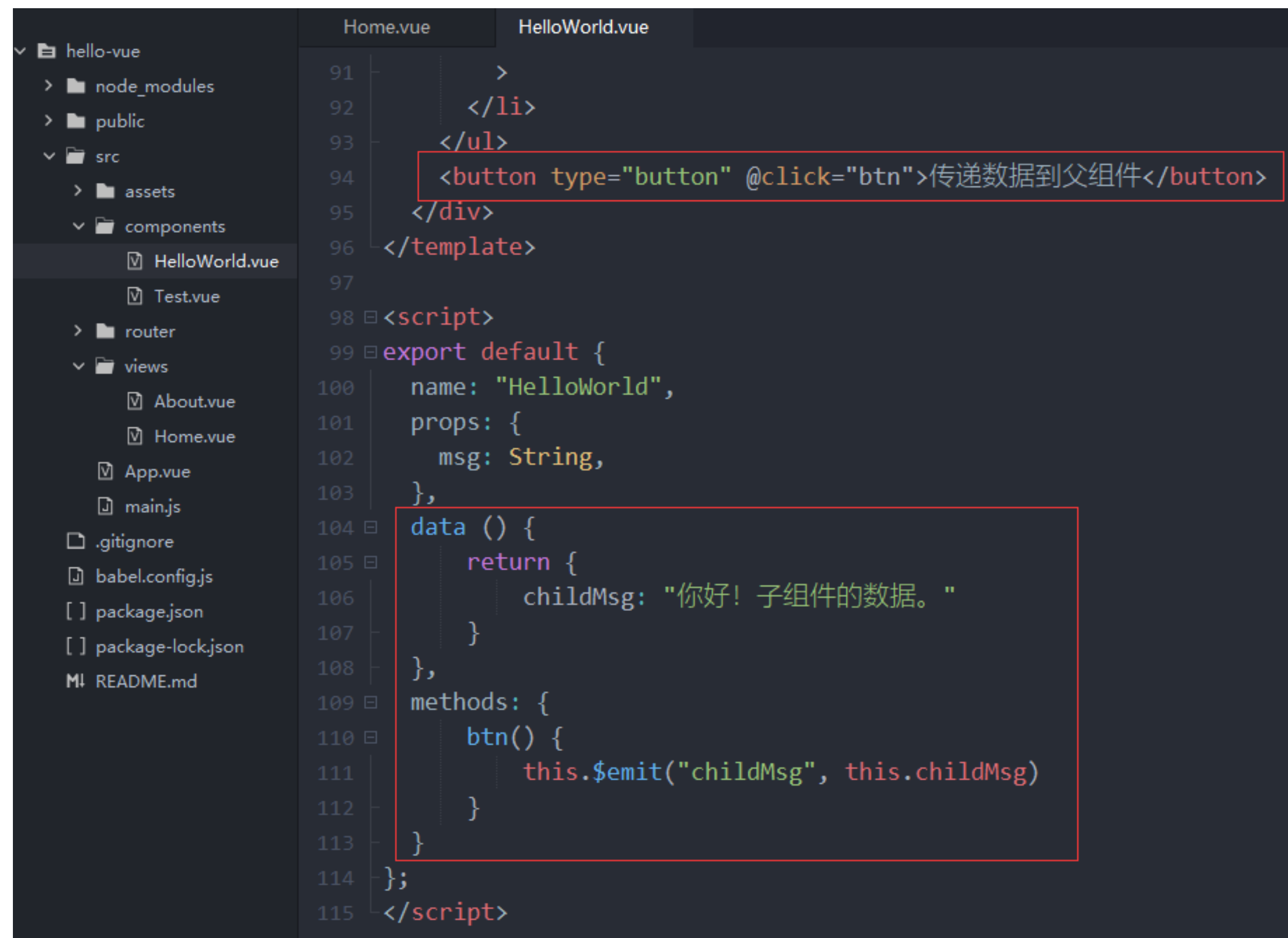
也可以通过v-bind或者简写：动态赋值，例如：



```
1 <template>
2   <div class="home">
3     
4     <!-- 引用组件 -->
5     <HelloWorld :msg="msg" />
6   </div>
7 </template>
8
9 <script>
10  // @ is an alias to /src
11  import HelloWorld from "@components/HelloWorld.vue";
12
13  export default {
14    name: "Home",
15    data () {
16      return {
17        msg: "你好！这是父组件的数据。"
18      }
19    },
20    components: {
21      HelloWorld,
22    },
23  };
24 </script>
```

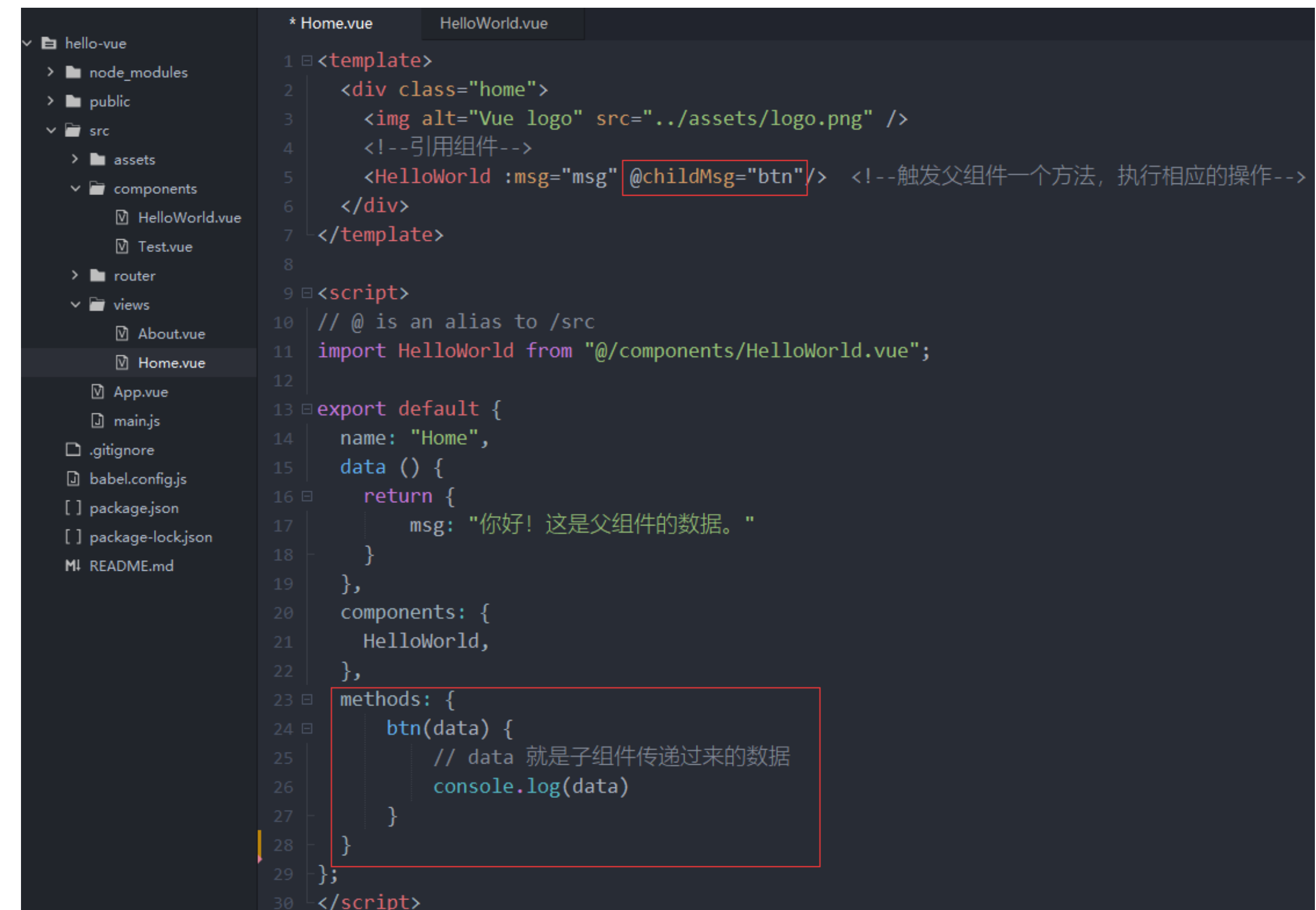
# 组件：传参

## 子传父：



```
91 |     >
92 |   </li>
93 | </ul>
94 |   <button type="button" @click="btn">传递数据到父组件</button>
95 | </div>
96 | </template>
97 |
98 | <script>
99 | export default {
100 |   name: "HelloWorld",
101 |   props: {
102 |     msg: String,
103 |   },
104 |   data () {
105 |     return {
106 |       childMsg: "你好！子组件的数据。"
107 |     }
108 |   },
109 |   methods: {
110 |     btn() {
111 |       this.$emit("childMsg", this.childMsg)
112 |     }
113 |   }
114 | };
115 | </script>
```

子组件



```
1 | <template>
2 |   <div class="home">
3 |     
4 |     <!--引用组件-->
5 |     <HelloWorld :msg="msg" @childMsg="btn"/> <!--触发父组件一个方法，执行相应的操作-->
6 |   </div>
7 | </template>
8 |
9 | <script>
10 | // @ is an alias to /src
11 | import HelloWorld from "@components/HelloWorld.vue";
12 |
13 | export default {
14 |   name: "Home",
15 |   data () {
16 |     return {
17 |       msg: "你好！这是父组件的数据。"
18 |     }
19 |   },
20 |   components: {
21 |     HelloWorld,
22 |   },
23 |   methods: {
24 |     btn(data) {
25 |       // data 就是子组件传递过来的数据
26 |       console.log(data)
27 |     }
28 |   }
29 | };
30 | </script>
```

父组件

# 前后端数据交互：Axios

- 介绍
- 使用
- 异常处理
- 全局默认值
- 自定义实例默认值
- 拦截器



# Axios 介绍

在前端页面展示的数据大多数都是通过访问一个API获取的，做这件事的方法有好几种，例如jquery ajax、vue-resource、axios，而vue-resource是vue插件，但3版本不再更新，目前官方推荐主流的axios，aixos是一个http请求库。

官方文档： <http://www.axios-js.com/zh-cn/docs/>

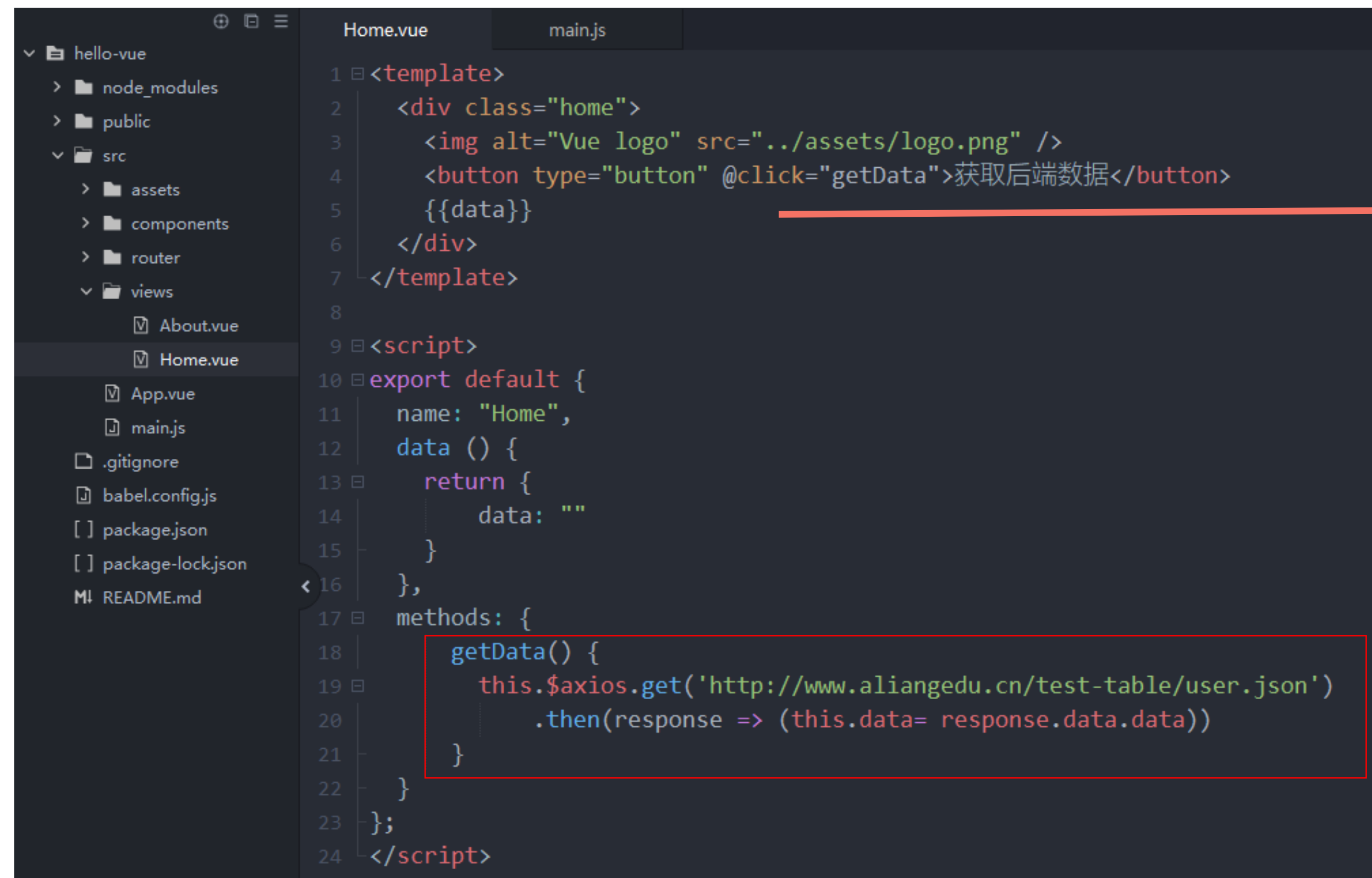
# Axios 使用

## 使用流程：

- 1、安装axios： `npm install axios`
- 2、在main.js文件中全局注册
- 3、在组件中使用

# Axios 使用: GET

从API获取用户数据，在前端展示：



```
1 <template>
2   <div class="home">
3     
4     <button type="button" @click="getData">获取后端数据</button>
5     {{data}}
6   </div>
7 </template>
8
9 <script>
10 export default {
11   name: "Home",
12   data () {
13     return {
14       data: ""
15     }
16   },
17   methods: {
18     getData() {
19       this.$axios.get('http://www.aliangedu.cn/test-table/user.json')
20         .then(response => (this.data= response.data.data))
21     }
22   }
23 };
24 </script>
```

数据表格展示数据：

```
<table border="1">
  <thead>
    <tr>
      <td>ID</td>
      <td>用户名</td>
      <td>性别</td>
      <td>城市</td>
    </tr>
  </thead>
  <tbody v-for="row in data" :key="row.id">
    <tr>
      <td>{{ row.id }}</td>
      <td>{{ row.username }}</td>
      <td>{{ row.sex }}</td>
      <td>{{ row.city }}</td>
    </tr>
  </tbody>
</table>
```

# Axios 使用: GET

如果打开页面就自动渲染, 可以放到mounted生命周期钩子中获取数据并赋值:

```
<script>
export default {
  name: "Home",
  data () {
    return {
      data: ""
    }
  },
  mounted: function () {
    this.getData();
  },
  methods: {
    getData() {
      this.$axios.get('http://www.aliangedu.cn/test-table/user.json')
        .then(response => (this.data= response.data.data))
    }
  }
};
</script>
```

# Axios 使用: POST

登录时提交表单数据案例:

```
* Home.vue      main.js
1 <template>
2   <div class="home">
3     
4
5     <h1>欢迎访问管理后台</h1>
6   <form action="#">
7     用户名: <input type="text" v-model="form.username">
8     密 码: <input type="text" v-model="form.password">
9     <button @click="loginBtn">登录</button>
10  </form>
11    <p style="color: red;" v-if="notice">用户名或者密码不能为空! </p>
12
13  </div>
14 </template>
15
16 <script>
17 export default {
18   name: "Home",
19   data () {
20     return {
21       form: {
22         username: "",
23         password: ""
24       },
25       notice: false
26     }
27   },
28   methods: {
29     loginBtn() {
30       if (this.form.username == '' || this.form.password == '') {
31         this.notice = true;
32       } else {
33         this.notice = false;
34         // console.log(this.form) // 获取输入用户名和密码 (提交到服务端)
35         this.$axios.post('http://127.0.0.1:8000/api', this.form)
36       }
37     }
38   }
39 };
40 </script>
```

# Axios 异常处理

很多时候我们可能并没有从 API 获取想要的`数据`。这可能是由于很多种因素引起的，比如 axios 调用可能由于多种原因而失败，包括但不限于：

- API 不工作了；
- 请求发错了；
- API 没有按我们预期的格式返回信息。

可以使用`catch`异常处理这些问题。

# Axios 异常处理

模拟连接一个未存在的地址，前端给出提示：

```
<template>
  <div class="home">
    
    <button type="button" @click="getData">获取后端数据</button>
    {{data}}
    <p v-if="error" style="color: red;">连接服务器失败，请稍后再试！ </p>
  </div>
</template>

<script>
export default {
  name: "Home",
  data () {
    return {
      data: "",
      error: false
    }
  },
  methods: {
    getData() {
      this.$axios.get('http://www.aliangedu.cn/test-table/user1.json')
        .then(response => (this.data= response.data.data))
        .catch(error => { // error是请求失败反馈数据
          console.log(error)
          this.error = error;
        })
    }
  }
};
</script>
```

# Axios 全局默认值

在实际开发中，几乎每个组件都会用到axios发起数据请求，如果每次都填写完整的请求路径，不利于后期维护。这时可以设置全局axios默认值。

```
Home.vue  ×  main.js
1  import { createApp } from "vue";
2  import App from "./App.vue";
3  import router from "./router";
4  import Test from './components/Test.vue' // 导入组件
5  import axios from 'axios'
6
7  const app = createApp(App)
8
9  axios.defaults.baseURL = 'http://www.aliangedu.cn/';
10 axios.defaults.timeout = 5000;
11 app.config.globalProperties.$axios=axios
12 app.use(router).mount("#app");
13
14 app.component('Test', Test) // 注册组件
```

在main.js里定义

```
Home.vue  main.js
1 <template>
2   <div class="home">
3     
4     <button type="button" @click="getData">获取后端数据</button>
5     {{data}}
6     <p v-if="error" style="color: red;">连接服务器失败，请稍后再试! </p>
7   </div>
8 </template>
9
10 <script>
11 export default {
12   name: "Home",
13   data () {
14     return {
15       data: "",
16       error: false
17     }
18   },
19   methods: {
20     getData() {
21       this.$axios.get('/test-table/user.json')
22         .then(response => (this.data= response.data.data))
23         .catch(error => { // error是请求失败反馈数据
24           console.log(error)
25           this.error = error;
26         })
27     }
28   }
29 };
30 </script>
```

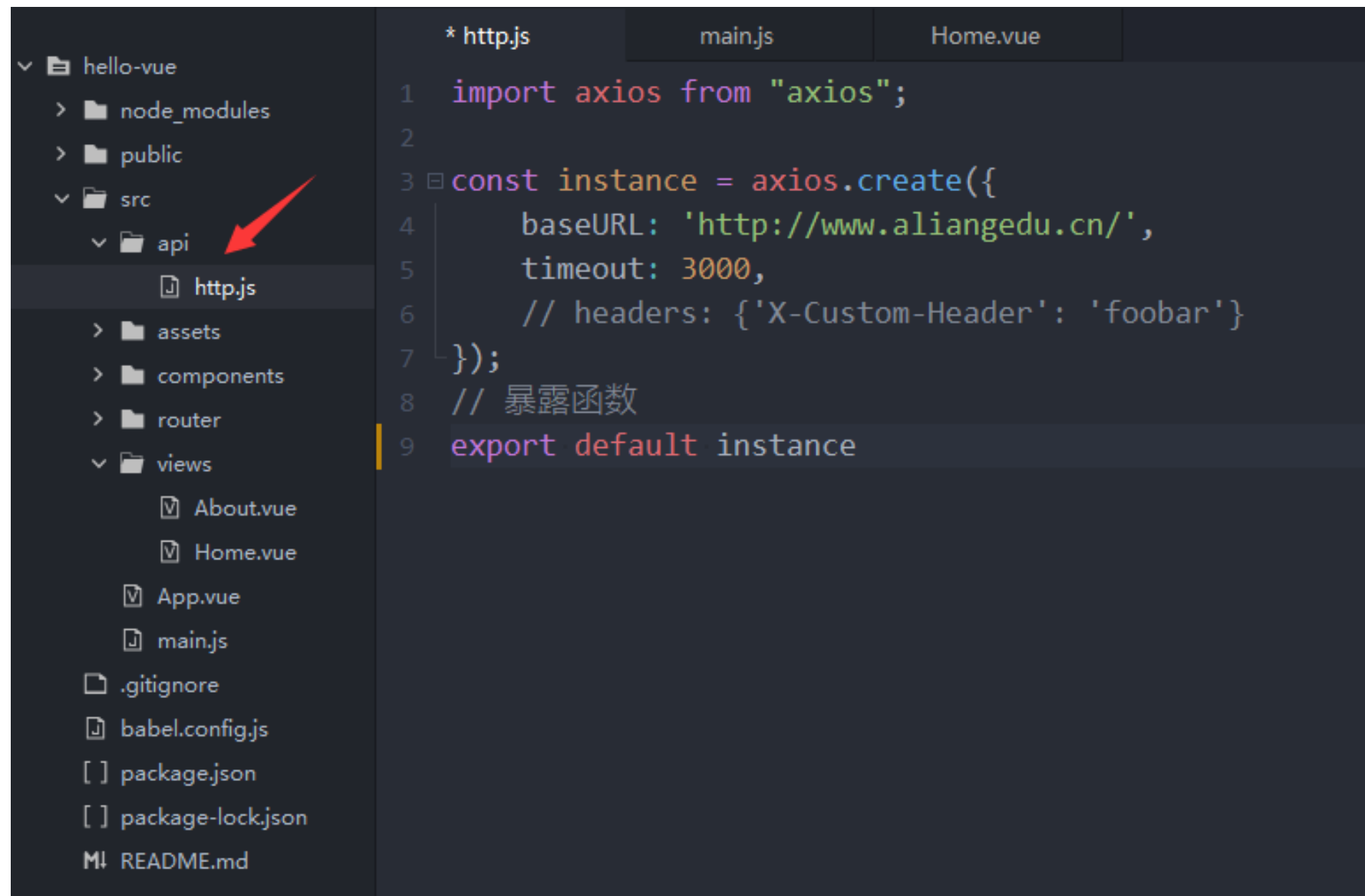
直接写接口地址即可



# Axios 自定义实例默认值

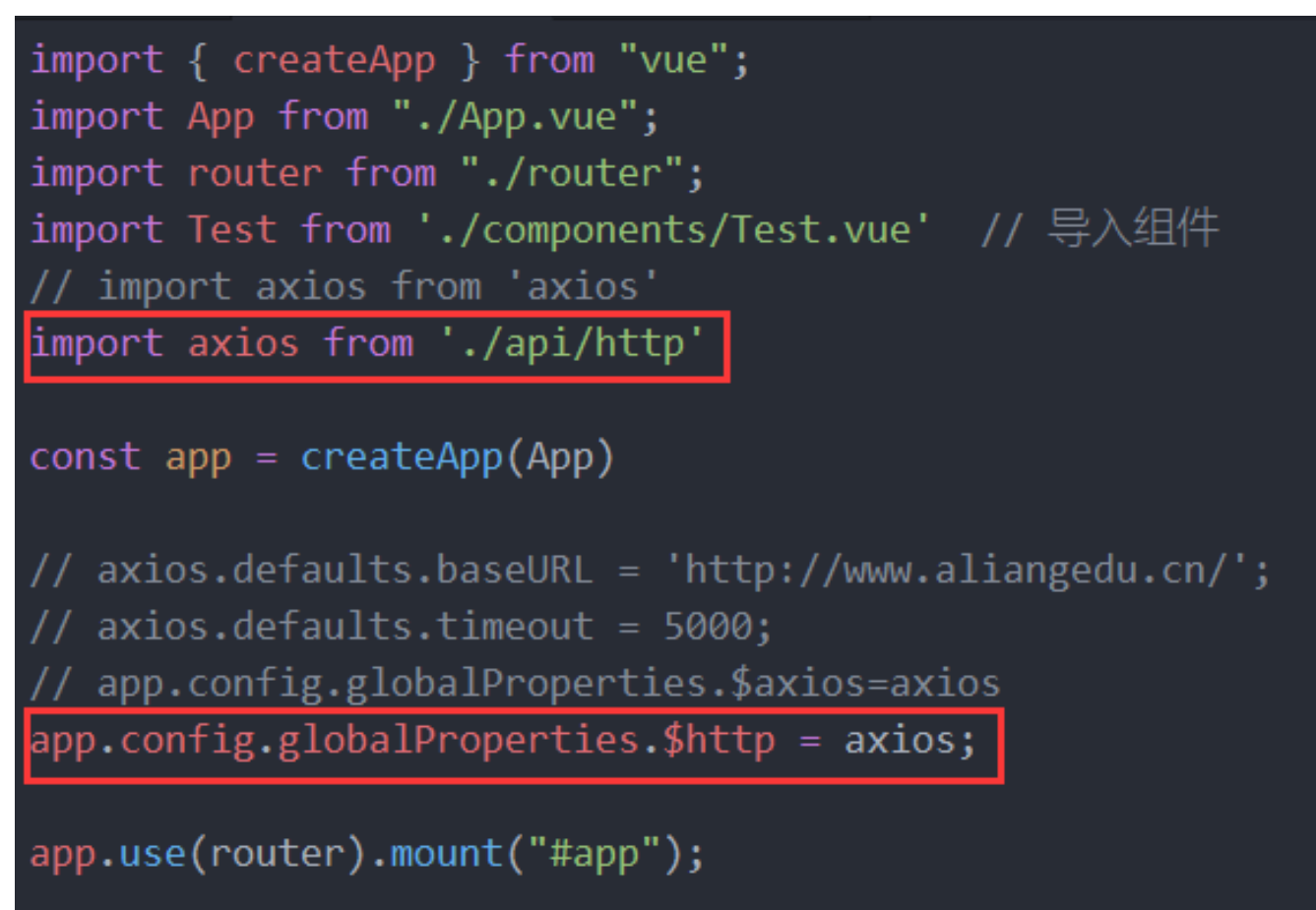
有时候服务端接口有多个地址，就会涉及请求的域名不同、配置不同等，这时自定义实例可以很好解决。

## 1、创建src/api/http.js文件，定义实例



```
* http.js
1 import axios from "axios";
2
3 const instance = axios.create({
4   baseURL: 'http://www.aliangedu.cn/',
5   timeout: 3000,
6   // headers: {'X-Custom-Header': 'foobar'}
7 });
8 // 暴露函数
9 export default instance
```

## 2、全局注册



```
import { createApp } from "vue";
import App from "./App.vue";
import router from "./router";
import Test from './components/Test.vue' // 导入组件
// import axios from 'axios'
import axios from './api/http'

const app = createApp(App)

// axios.defaults.baseURL = 'http://www.aliangedu.cn/';
// axios.defaults.timeout = 5000;
// app.config.globalProperties.$axios=axios
app.config.globalProperties.$http = axios;

app.use(router).mount("#app");
```

## 3、组件使用



```
<script>
export default {
  name: "Home",
  data () {
    return {
      data: "",
      error: false
    }
  },
  methods: {
    getData() {
      this.$http.get('/test-table/user.json')
        .then(response => (this.data= response.data.data))
        .catch(error => { // error是请求失败反馈数据
          console.log(error)
          this.error = error;
        })
    }
  }
};
</script>
```

# Axios 拦截器

拦截器可以拦截每一次请求和响应，然后进行相应的处理。

请求拦截应用场景：

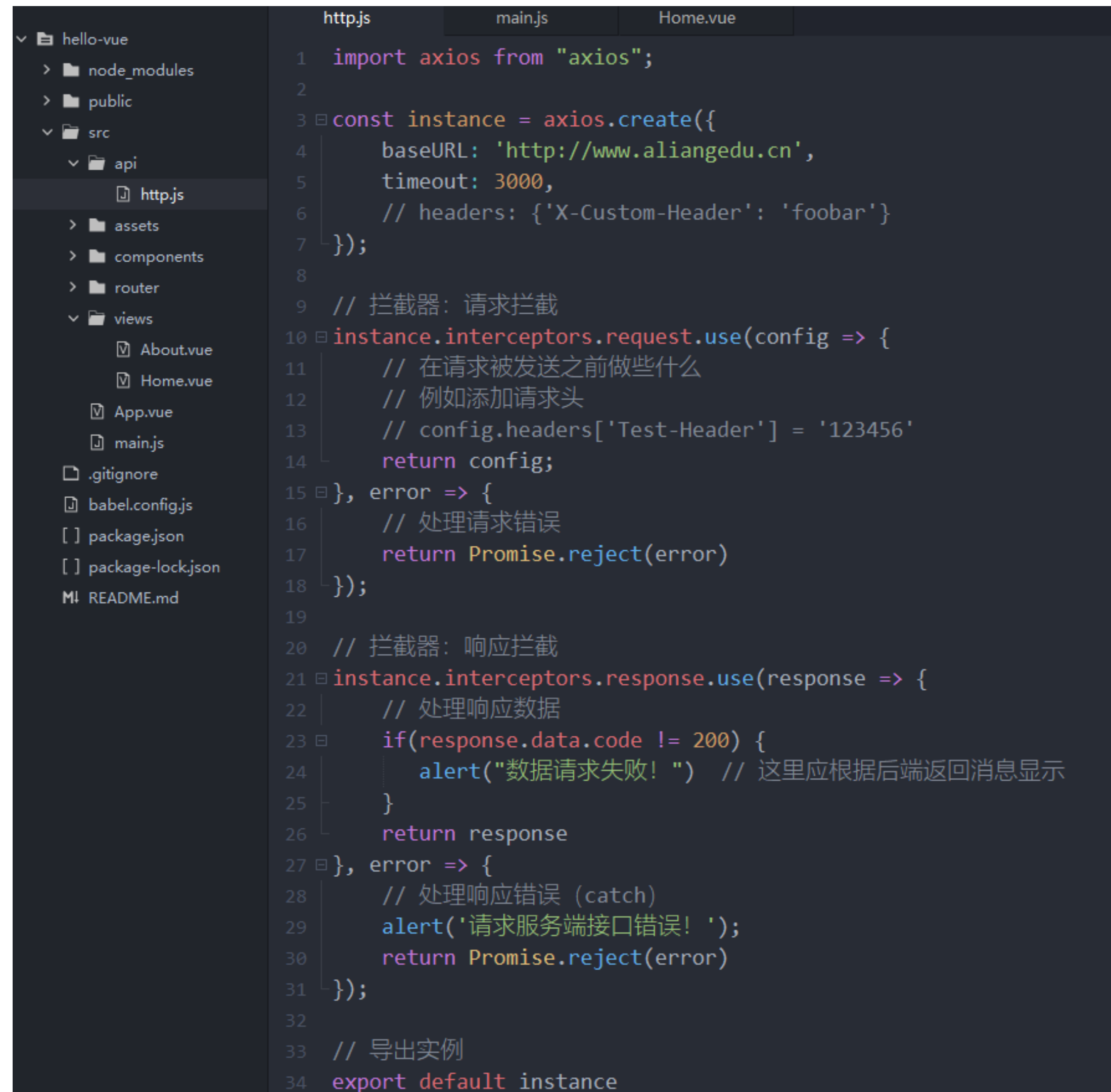
- 发起请求前添加header

响应拦截应用场景：

- 统一处理API响应状态码200或非200的提示消息
- 统一处理catch异常提示信息



# Axios 拦截器



```
1 import axios from "axios";
2
3 const instance = axios.create({
4   baseURL: 'http://www.aliangedu.cn',
5   timeout: 3000,
6   // headers: {'X-Custom-Header': 'foobar'}
7 });
8
9 // 拦截器：请求拦截
10 instance.interceptors.request.use(config => {
11   // 在请求被发送之前做点什么
12   // 例如添加请求头
13   // config.headers['Test-Header'] = '123456'
14   return config;
15 }, error => {
16   // 处理请求错误
17   return Promise.reject(error)
18 });
19
20 // 拦截器：响应拦截
21 instance.interceptors.response.use(response => {
22   // 处理响应数据
23   if(response.data.code !== 200) {
24     alert("数据请求失败！") // 这里应根据后端返回消息显示
25   }
26   return response
27 }, error => {
28   // 处理响应错误 (catch)
29   alert('请求服务端接口错误！');
30   return Promise.reject(error)
31 });
32
33 // 导出实例
34 export default instance
```

请求/响应拦截示例

# Vue 路由: Vue Router

- 介绍
- 使用
- 路由传参
- 导航守卫

# Vue Router介绍

**Vue Router** 是 [Vue.js \(opens new window\)](#)官方的路由管理器。它和 Vue.js 的核心深度集成，包含的功能有：

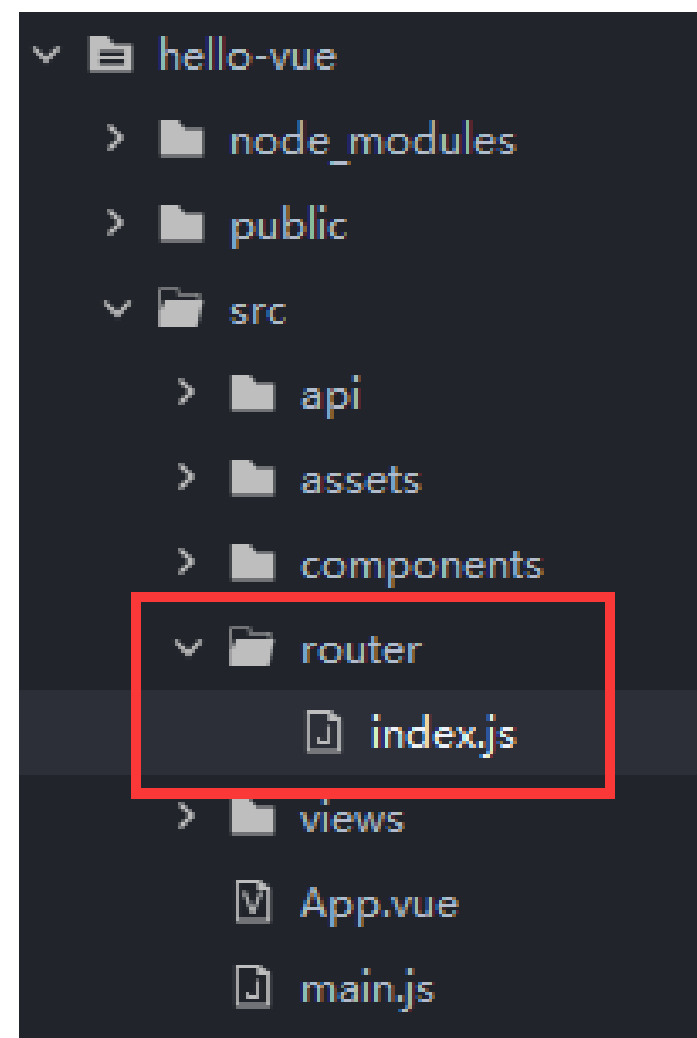
- 嵌套的路由/视图表
- 模块化的、基于组件的路由配置
- 路由参数、查询、通配符
- 基于 Vue.js 过渡系统的视图过渡效果
- 细粒度的导航控制

# Vue Router安装

在用脚手架创建项目时已经选择安装上了，如果刚开始没有，通过npm安装：

`npm install vue-router@4`

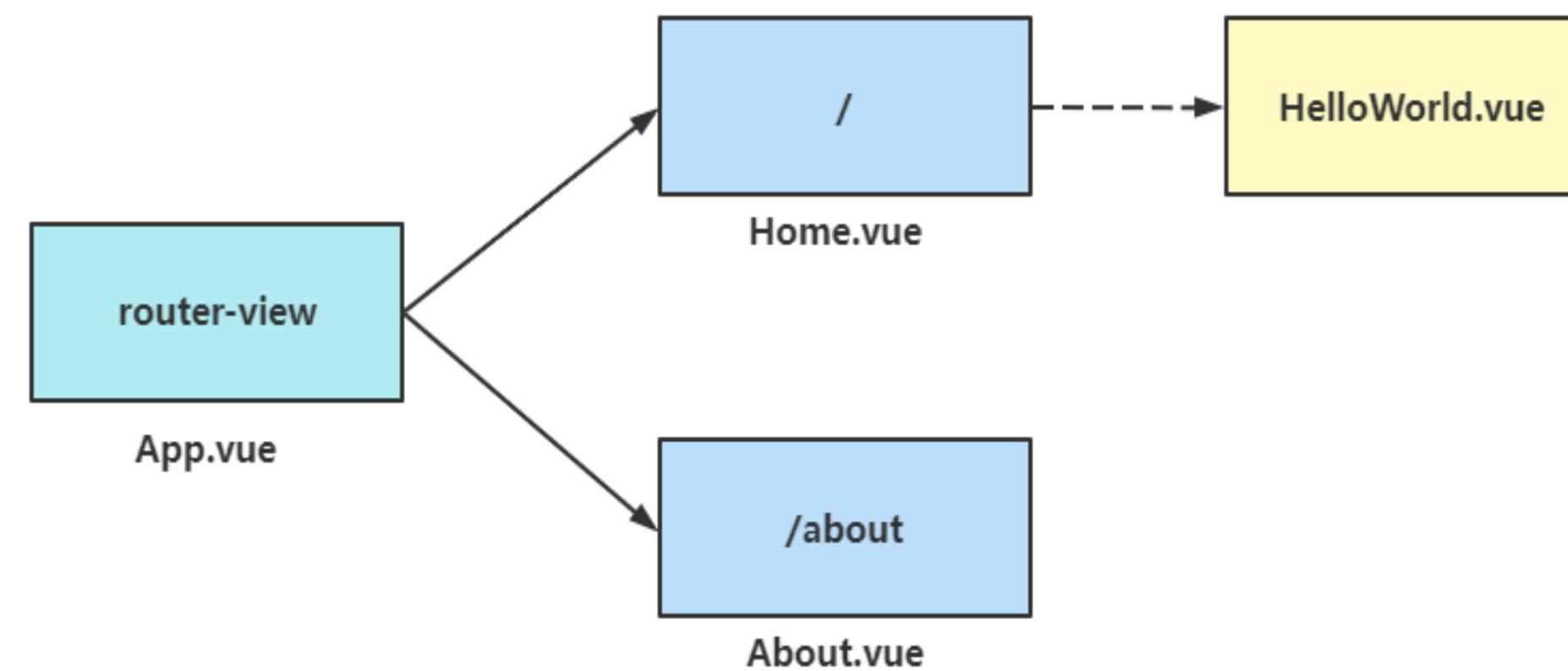
安装完后会有router目录：



# Vue Router介绍

## 使用流程:

1. 开发页面 (组件)
2. 定义路由
3. 组件使用路由



项目已有示例



# Vue Router介绍

hello-vue

node\_modules

public

src

api

assets

components

router

views

App.vue

main.js

.gitignore

babel.config.js

package.json

Home.vue

```
1 <template>
2   <div class="home">
3     
4     <HelloWorld msg="Welcome to Your Vue.js App"></HelloWorld>
5   </div>
6 </template>
7
8 <script>
9   import HelloWorld from "@components/HelloWorld.vue"
10
11 export default {
12   name: "Home",
13   components: {
14     HelloWorld,
15   }
16 };
17 </script>
```

页面组件

hello-vue

node\_modules

public

src

api

assets

components

router

views

App.vue

main.js

.gitignore

babel.config.js

package.json

package-lock.json

README.md

index.js

```
1 import { createRouter, createWebHashHistory } from "vue-router";
2 import Home from "../views/Home.vue"; // 导入组件方式1: 先导入, 下面引用
3
4 // 1.定义路由对象
5 const routes = [
6   {
7     path: "/", // 访问路径
8     name: "Home", // 路由名称
9     component: Home, // 引用组件
10   },
11   {
12     path: "/about",
13     name: "About",
14     // 导入组件方式2: 当路由被访问时才会加载组件 (惰性)
15     component: () =>
16       import("../views/About.vue"),
17   },
18 ];
19
20 // 2.创建路由实例并传递上面路由对象 routes
21 const router = createRouter({
22   history: createWebHashHistory(), // 路由模式
23   routes,
24 });
25
26 export default router;
```

定义路由

hello-vue

node\_modules

public

src

api

assets

components

router

views

App.vue

main.js

.gitignore

babel.config.js

package.json

package-lock.json

README.md

index.js

App.vue

```
1 <template>
2   <div id="nav">
3     <!-- 使用 router-link 组件来导航 -->
4     <!-- 通过传入 to 属性指定链接 -->
5     <!-- <router-link> 默认会被渲染成一个 a 标签 -->
6     <router-link to="/">Home</router-link> |
7     <router-link to="/about">About</router-link>
8   </div>
9   <!-- 路由出口, 相当于占位符, 路由匹配到的组件就会展示在这里 -->
10   <router-view />
11 </template>
12
13 <style>
14 #app {
15   font-family: Avenir, Helvetica, Arial, sans-serif;
16   -webkit-font-smoothing: antialiased;
17   -moz-osx-font-smoothing: grayscale;
```

组件使用路由



# Vue Router 路由传参

URL传参：一般用于页面跳转，将当前数据传递到新页面，例如详情页

## params传参

- 配置路由：{path: '/user/:id' , component: about}
- 传递方式：<router-link to=" /user/6/" ></router-link>
- 传递后路径：/user/6
- 接收参数：\$route.params.id

## query传参

- 配置路由：{path: '/user/' , component: about}
- 传递方式：<router-link to=" {path: '/about ' , query:{id:6}}" ></router-link>
- 传递后路径：/user?id=6
- 接收参数：\$route.query.id

# Vue Router 导航守卫

正如其名，vue-router 提供的导航守卫主要用来通过跳转或取消的方式守卫导航。简单来说，就是在路由跳转时候的一些钩子，当从一个页面跳转到另一个页面时，可以在跳转前、中、后做一些事情。

# Vue Router 导航守卫

你可以使用 `router.beforeEach` 注册一个全局前置守卫：

```
const router = createRouter({ ... })

// 添加全局导航守卫（回调函数）
router.beforeEach((to, from) => {
  // 这里执行具体操作
  console.log(to)
  console.log(from)
});
```

当一个导航触发时，就会异步执行这个回调函数。

**每个守卫方法接收参数：**

- `to`：即将要进入的目标，是一个路由对象
- `from`：当前导航正要离开的路由，也是一个路由对象
- `next`：可选，是一个方法

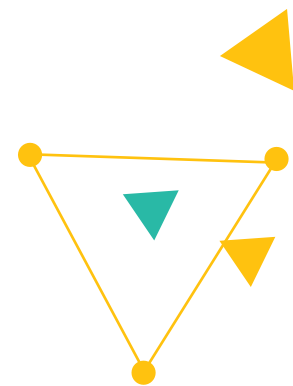
**可以返回的值如下：**

- `false`：取消当前的导航。如果浏览器的 URL 改变了(可能是用户手动或者浏览器后退按钮)，那么 URL 地址会重置到 `from` 路由对应的地址。
- 一个路由地址：通过一个路由地址跳转到一个不同的地址。

# Vue Router 导航守卫

在网站开发中，使用导航守卫一个普遍需求：**登录验证**，即在没  
有登录的情况下，访问任何页面都跳转到登录页面。

```
router.beforeEach((to, from, next) => {  
  // 如果用户访问登录页，直接放行  
  if(to.path == '/login') {  
    return next();  
  }  
  const token = '666666'; // 模拟token，正常是从本地session存储获取  
  if(token) {  
    next() // 如果有token，则正常跳转访问  
  } else {  
    return next('/login') // 如果没有token，跳转到登录页  
  }  
});
```



# 谢谢

---



阿良个人微信



DevOps技术栈公众号

阿良教育: [www.aliangedu.cn](http://www.aliangedu.cn)

