艾梯哎教育—芳芳带你学前端

资料获取WX：18040505058

vite2+vue3.2+elementplus快速上手

Vite

什么是Vite

Vite 是一个 web 开发构建工具，由于其原生 ES 模块导入方法，它允许快速提供代码。在开发环境下基于浏览器原生 ES imports 开发，在生产环境下基于 Rollup 打包。

通过在终端中运行以下命令，可以使用 Vite 快速构建 VUE 项目比webpack打包更加快速。

它主要具有以下特点：
1.快速的冷启动
2.即时的模块热更新
3.真正的按需编译

快速创建vite1项目步骤：

1、全局安装vite

```
D:\FF\files\course\public\vue3
λ cnpm i -g create-vite-app
Downloading create-vite-app to C:\Users\FF\AppData\Roaming\npm\node_modules\create-vite-app_tmp
Copying C:\Users\FF\AppData\Roaming\npm\node_modules\create-vite-app_tmp\_create-vite-app@1.21.0@create-vite-app to C:\Users\FF\AppData\Roaming\npm\node_modules\create-vite-app
Installing create-vite-app's dependencies to C:\Users\FF\AppData\Roaming\npm\node_modules\create-vite-app/node_modules
[1/2] minimist@^1.2.5 installed at node_modules\_minimist@1.2.5@minimist
[2/2] fs-extra@^9.0.0 installed at node_modules\_fs-extra@9.1.0@fs-extra
All packages installed (6 packages installed from npm registry, used 481ms(network 472ms), speed 159.18kB/s, json 6(15.62kB), tarball 59.51kB)
[create-vite-app@1.21.0] link C:\Users\FF\AppData\Roaming\npm\create-vite-app@ -> C:\Users\FF\AppData\Roaming\npm\node_modules\create-vite-app\index.js
[create-vite-app@1.21.0] link C:\Users\FF\AppData\Roaming\npm\cva@ -> C:\Users\FF\AppData\Roaming\npm\node_modules\create-vite-app\index.js
```

2、创建vite项目

```
D:\FF\files\course\public\vue3
λ create-vite-app vue3-demo
Scaffolding project in D:\FF\files\course\public\vue3\vue3-demo...

Done. Now run:

  cd vue3-demo
  npm install (or `yarn`)
  npm run dev (or `yarn dev`)
```

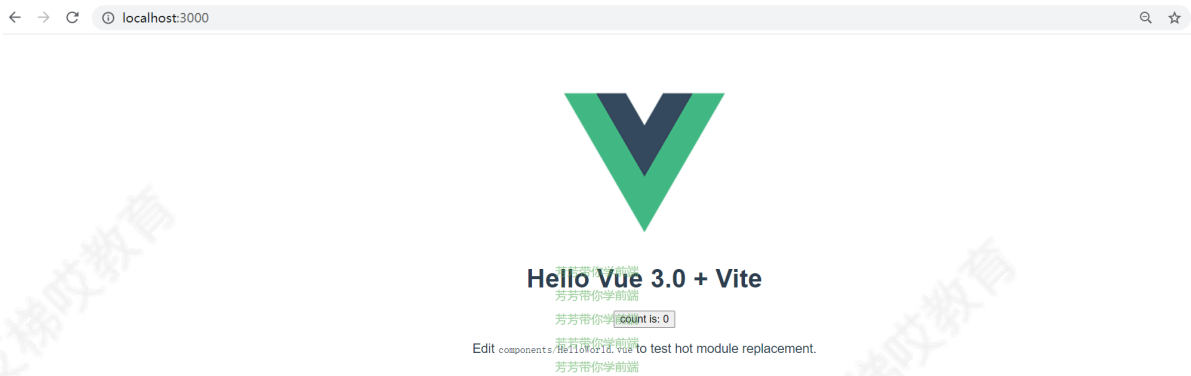
3、安装依赖

```
D:\FF\files\course\public\vue3\vue3-demo  (vue3-demo@0.0.0)
λ cnpm install
| [2/3] Installing path-is-absolute@^1.0.0platform unsupported vite@1.0.0-rc.13 >
@3.5.2 > fsevents@~2.3.2 Package require os(darwin) not compatible with your plat
2)
[fsevents@~2.3.2] optional install error: Package require os(darwin) not compatib
our platform(win32)
√ Installed 3 packages
√ Linked 250 latest versions
[1/1] scripts.postinstall vite@1.0.0-rc.13 > esbuild@^0.8.12 run "node install.js
"D:\\FF\\files\\course\\public\\vue3\\vue3-demo\\node_modules\\_esbuild@0.8.57@es
[1/1] scripts.postinstall vite@1.0.0-rc.13 > esbuild@^0.8.12 finished in 3s
```

4、运行vite项目

```
D:\FF\files\course\public\vue3\vue3-demo  (vue3-demo@0.0.0)
λ npm run dev

> vue3-demo@0.0.0 dev D:\FF\files\course\public\vue3\vue3-demo
> vite

[vite] Optimizable dependencies detected:
vue

  Dev server running at:
  > Network:  http://192.168.1.3:3000/
  > Local:    http://localhost:3000/
```

5、访问项目



vue-cli默认是8080端口，vite默认是3000端口

vite项目默认不装有vue-router和vuex，如需使用需要手动导入

package.json

package.json

```
{
  "name": "vue3-demo",
  "version": "0.0.0",
  "scripts": {
    "dev": "vite",
    "build": "vite build"
  },
  "dependencies": {
    "vue": "^3.0.4"
  },
  "devDependencies": {
    "vite": "^1.0.0-rc.13",
    "@vue/compiler-sfc": "^3.0.4"
  }
}
```

Vite2构建Vue3项目

环境：node.js版本>=12.0.0; npm 6.x

1、创建vite2项目

```
D:\FF\files\course\public\vue3
λ npm init vite
npx: 6 安装成功，用时 1.59 秒
√ Project name: ... vite-project
? Select a framework: » - Use arrow-keys. Return to submit.
>   vanilla
    vue
    react
    preact
    lit
    svelte
```

选择框架

```
D:\FF\files\course\public\vue3
λ npm init vite
npx: 6 安装成功，用时 1.59 秒
√ Project name: ... vite-project
√ Select a framework: » vue
? Select a variant: » - Use arrow-keys. Return to submit.
>   vue
    vue-ts
```

选择类别

```
D:\FF\files\course\public\vue3
λ npm init vite
npx: 6 安装成功，用时 1.59 秒
√ Project name: ... vite-project
√ Select a framework: » vue
√ Select a variant: » vue

Scaffolding project in D:\FF\files\course\public\vue3\vite-project...

Done. Now run:

  cd vite-project
  npm install
  npm run dev
```

2、安装依赖

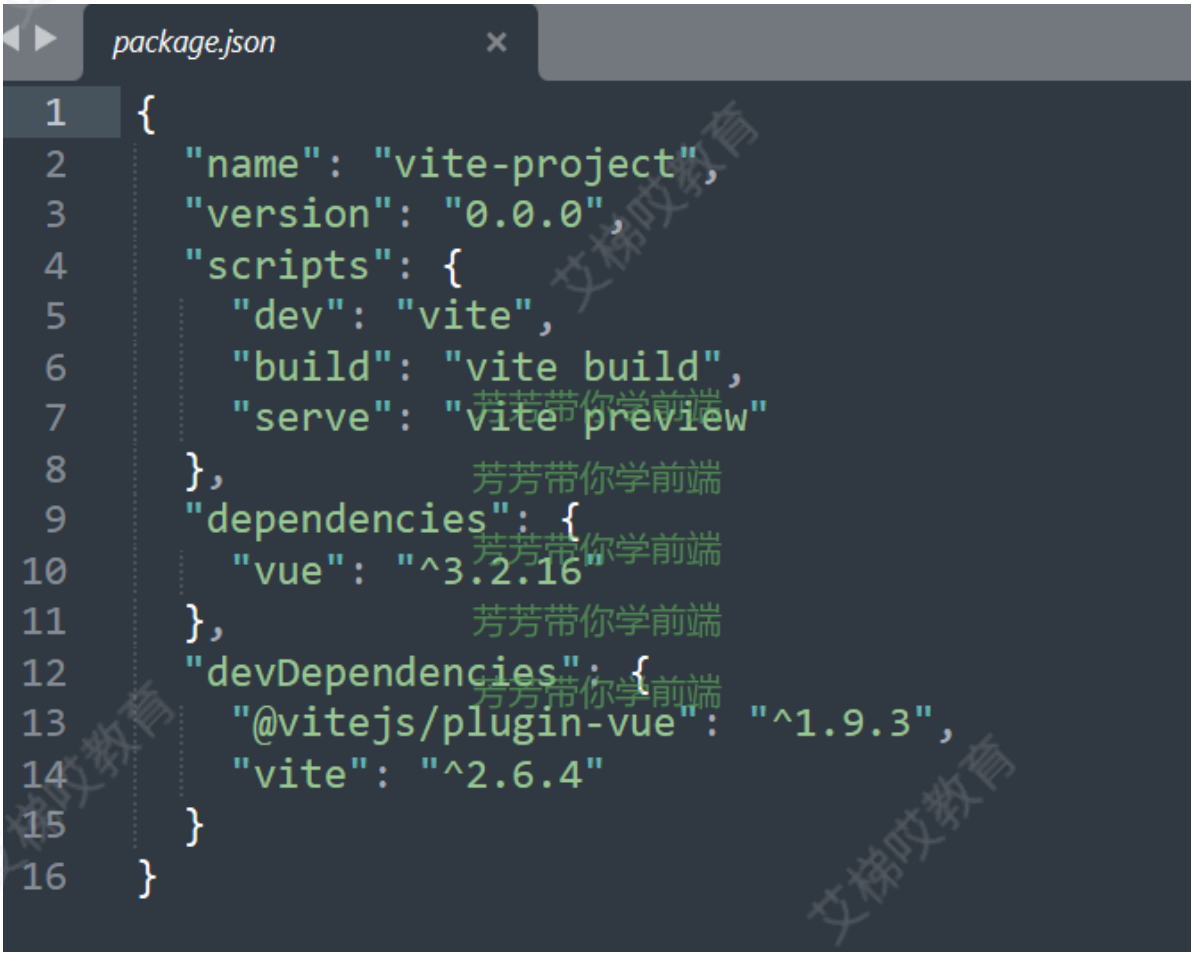
```
D:\FF\files\course\public\vue3
λ cd vite-project

D:\FF\files\course\public\vue3\vite-project  (vite-project@0.0.0)
λ cnpm install
/ [1/3] Installing source-map-js@^0.6.2platform unsupported vite@2.6.13 › fsevents@~2.3.2
Package require os(darwin) not compatible with your platform(win32)
[fsevents@~2.3.2] optional install error: Package require os(darwin) not compatible with y
our platform(win32)
\ [1/3] Installing function-bind@^1.1.1platform unsupported vite@2.6.13 › esbuild@0.13.12
› esbuild-android-arm64@0.13.12 Package require os(android) not compatible with your platf
orm(win32)
platform unsupported vite@2.6.13 › esbuild@0.13.12 › esbuild-linux-32@0.13.12 Package requ
ire os(linux) not compatible with your platform(win32)
platform unsupported vite@2.6.13 › esbuild@0.13.12 › esbuild-darwin-arm64@0.13.12 Package
require os(darwin) not compatible with your platform(win32)
```

3、运行vite项目

```
D:\FF\files\course\public\vue3\vite-project  (vite-project@0.0.0)
λ npm run dev

> vite-project@0.0.0 dev D:\FF\files\course\public\vue3\vite-project
> vite

Pre-bundling dependencies:
  vue
(this will be run only when your dependencies or config have changed)

  vite v2.6.13 dev server running at:

  > Local: http://localhost:3000/
  > Network: use `--host` to expose

  ready in 575ms.
```

4、访问项目

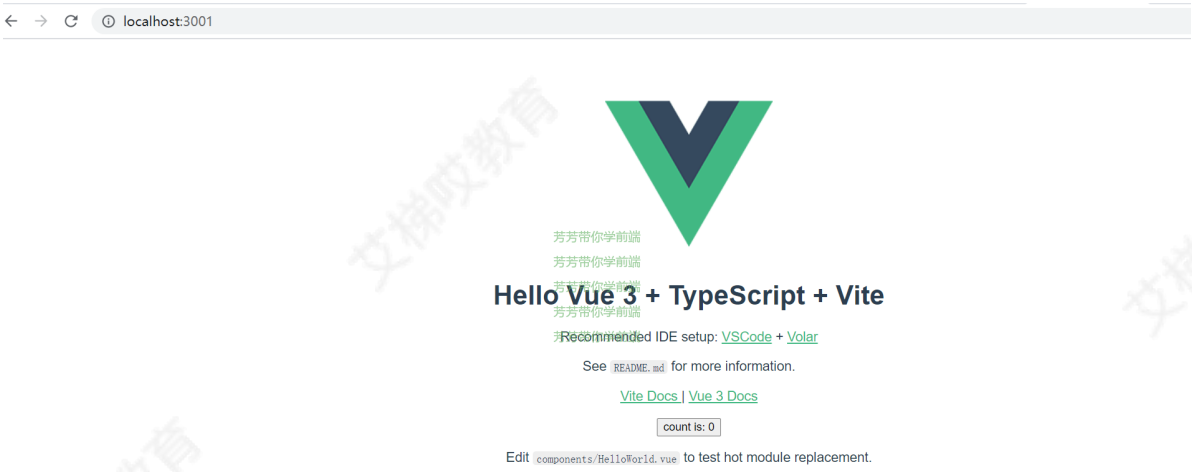
package.json



vue3+typescript+vite

创建步骤和前面一样

访问项目



package.json

```
package.json
{
  "name": "vite-ts",
  "version": "0.0.0",
  "scripts": {
    "dev": "vite",
    "build": "vue-tsc --noEmit && vite build",
    "serve": "vite preview"
  },
  "dependencies": {
    "vue": "^3.2.16"
  },
  "devDependencies": {
    "@vitejs/plugin-vue": "^1.9.3",
    "typescript": "^4.4.3",
    "vite": "^2.6.4",
    "vue-tsc": "^0.3.0"
  }
}
```

路由router

```
D:\FF\files\course\public\vue3\vite-project  (vite-project@0.0.0)
λ cnpm i vue-router@4 -S
√ Installed 1 packages
√ Linked 2 latest versions
√ Run 0 scripts
√ All packages installed (2 packages installed from npm registry, used 310ms(network
s), speed 204.11kB/s, json 3(61.85kB), tarball 0B)
```

src目录下，添加router/index.js

```
import { createRouter, createWebHashHistory } from 'vue-router'
import Home from '../views/Home.vue'
```

```
const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/about',
    name: 'About',
    component: () => import('../views/About.vue')
  }
]

const router = createRouter({
  history: createWebHashHistory(),
  routes
})

export default router
```

main.js导入router

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router/index'

createApp(App).use(router).mount('#app')
```

vuex4

src目录下，添加store/index.js

```
import { createStore } from 'vuex'

export default createStore({
  state: {
    num:0
  },
  mutations: {
  },
  actions: {
  },
  modules: {
  }
})
```

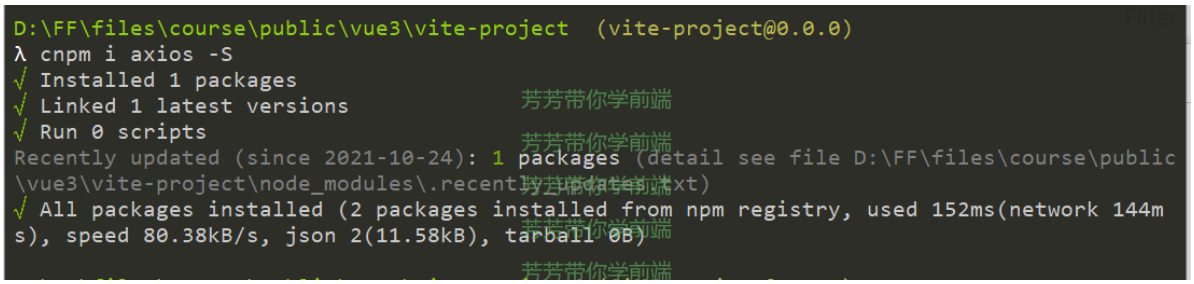
main.js导入vuex

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router/index'
import store from './store/index'

createApp(App).use(router).use(store).mount('#app')
```

vuex组件内测试

```
<template>
  <div class="home">
    store: {{$store.state.num}}
    {{msg}}
  </div>
</template>

<script setup>
import {useStore} from "vuex"
const store = useStore;
let msg = "hello!"
</script>
```

axios



axios配置

```
import axios from "axios";

const service = axios.create({
    baseURL:"http://127.0.0.1:3333/",
    timeout:1000
})
service.interceptors.request.use(config =>{
    return config;
})
service.interceptors.response.use(response =>{
    return response
}, err=>{
    return Promise.reject(err);
})
export default service;
```

引入element-plus

1、安装

```
# NPM
$ npm install element-plus --save

# Yarn
$ yarn add element-plus
```

2、引入

2-1、完整引入

```
import ElementPlus from 'element-plus'
import 'element-plus/dist/index.css'

const app = createApp(App)

app.use(ElementPlus)
app.mount('#app')
```

2-2、按需导入

需要使用额外的插件来导入要使用的组件。

2-2-1、自动导入

首先需要安装 unplugin-vue-components

```
npm install unplugin-vue-components
```

然后将以下代码添加到 `Vite` 或 `Webpack` 的配置文件中

vite

```
// vite.config.ts
import Components from 'unplugin-vue-components/vite'
import { ElementPlusResolver } from 'unplugin-vue-components/resolvers'

export default {
  plugins: [
    // ...
    Components({
      resolvers: [ElementPlusResolver()],
    }),
  ],
}
```

webpack

```
// webpack.config.js
const Components = require('unplugin-vue-components/webpack')
const { ElementPlusResolver } = require('unplugin-vue-components/resolvers')

module.exports = {
  // ...
  plugins: [
    Components({
      resolvers: [ElementPlusResolver()],
    }),
  ],
}
```

2-2-2、手动导入

Element Plus 提供了基于 ES Module 开箱即用的Tree Shaking 功能。

app.vue

```
<template>
  <el-button>I am ElButton</el-button>
</template>
<script>
  import { ElButton } from 'element-plus'
  export default {
    components: { ElButton },
  }
</script>
```

```
// vite.config.ts
import ElementPlus from 'unplugin-element-plus/vite'

export default {
  plugins: [ElementPlus()],
}
```

vue3.2新特性

单文件组件 (SFC) 的新特性

单文件组件（SFC，又称作`.vue`文件）的两项实验特性已毕业，现已提供稳定版本：

-
-
  - 更少的样板内容，更简洁的代码。
  - 能够使用纯 Typescript 声明 props 和抛出事件。
  - 更好的运行时性能 (其模板会被编译成与其同一作用域的渲染函数，没有任何的中间代理)。
  - 更好的 IDE 类型推断性能 (减少语言服务器从代码中抽离类型的工作)。

```
<script setup>
import { ref } from 'vue'

const color = ref('red')
</script>
```

```
<template>
  <button @click="color = color === 'red' ? 'green' : 'red'">
    Color is: {{ color }}
  </button>
</template>

<style scoped>
button {
  color: v-bind(color);
}
</style>
```

基本用法：

要使用这个语法，需要将 `setup` attribute 添加到 `<script>` 代码块上：

```
<script setup>
    console.log('hello script setup')
</script>
```

里面的代码会被编译成组件 `setup()` 函数的内容。这意味着与普通的 `<script>` 只在组件被首次引入的时候执行一次不同，`<script setup>` 中的代码会在**每次组件实例被创建的时候执行。**

顶层的绑定会被暴露给模板

当使用 `<script setup>` 的时候，任何在 `<script setup>` 声明的顶层的绑定 (包括变量，函数声明，以及 import 引入的内容) 都能在模板中直接使用：

```
<script setup>
// 变量
const msg = 'Hello!'

// 函数
function log() {
  console.log(msg)
}
</script>

<template>
  <div @click="log">{{ msg }}</div>
</template>
```

import 导入的内容也会以同样的方式暴露。意味着可以在模板表达式中直接使用导入的 helper 函数，并不需要通过 `methods` 选项来暴露它：

```
<script setup>
import { capitalize } from './helpers'
</script>

<template>
  <div>{{ capitalize('hello') }}</div>
</template>
```

响应式

响应式状态需要明确使用响应式APIs 来创建。和从 `setup()` 函数中返回值一样，ref 值在模板中使用的时候会自动解包：

```
<script setup>
import { ref } from 'vue'

const count = ref(0)
</script>

<template>
  <button @click="count++">{{ count }}</button>
</template>
```

使用组件

```
<script setup>
import MyComponent from './MyComponent.vue'
</script>

<template>
  <MyComponent />
</template>
```

动态组件

由于组件被引用为变量而不是作为字符串键来注册的，在 `<script setup>` 中要使用动态组件的时候，就应该使用动态的 `:is` 来绑定：

```
<script setup>
import Foo from './Foo.vue'
import Bar from './Bar.vue'
</script>

<template>
  <component :is="Foo" />
  <component :is="someCondition ? Foo : Bar" />
</template>
```

递归组件

一个单文件组件可以通过它的文件名被其自己所引用。例如：名为 `FooBar.vue` 的组件可以在其模板中用 `<FooBar/>` 引用它自己。

请注意这种方式相比于 import 导入的组件优先级更低。如果有命名的 import 导入和组件的推断名冲突了，可以使用 import 别名导入：

```
import { FooBar as FooBarChild } from './components'
```

命名空间组件

可以使用带点的组件标记，例如 `<Foo.Bar>` 来引用嵌套在对象属性中的组件。这在需要从单个文件中导入多个组件的时候非常有用：

```
<script setup>
import * as Form from './form-components'
</script>

<template>
  <Form.Input>
    <Form.Label>label</Form.Label>
  </Form.Input>
</template>
```

defineProps和 defineEmits

在 `<script setup>` 中必须使用 `defineProps` 和 `defineEmits` API 来声明 `props` 和 `emits` ，它们具备完整的类型推断并且在 `<script setup>` 中是直接可用的：

```
<script setup>
const props = defineProps({
  foo: String
})

const emit = defineEmits(['change', 'delete'])
// setup code
</script>
```

- `defineProps` 和 `defineEmits` 都是只在 `<script setup>` 中才能使用的**编译器宏**。他们不需要导入且会随着 `<script setup>` 处理过程一同被编译掉。
- `defineProps` 接收与 `props` 选项相同的值，`defineEmits` 也接收 `emits` 选项相同的值。
- `defineProps` 和 `defineEmits` 在选项传入后，会提供恰当的类型推断。
- 传入到 `defineProps` 和 `defineEmits` 的选项会从 setup 中提升到模块的范围。因此，传入的选项不能引用在 setup 范围中声明的局部变量。这样做会引起编译错误。但是，它*可以*引用导入的绑定，因为它们也在模块范围内。

与普通的 `<script>` 一起使用

```
<script>
// 普通 <script>, 在模块范围下执行(只执行一次)
runSideEffectOnce()

// 声明额外的选项
export default {
  inheritAttrs: false,
  customOptions: {}
}
</script>

<script setup>
// 在 setup() 作用域中执行 (对每个实例皆如此)
</script>
```

顶层 await

```
<script setup>
const post = await fetch(`/api/post/1`).then(r => r.json())
</script>
```

实例扩展效果图

所需组件汇总

1、登录页

`Form` 组件提供了表单验证的功能，只需要通过 `rules` 属性传入约定的验证规则，并将 `form-Item` 的 `prop` 属性设置为需 form 绑定值对应的字段名即可。

```
<div class="login">
    <p class="login-title">芳芳带你学前端</p>
    <div class="login-content">
      <el-form
        ref="ruleForm"
        :model="ruleForm"
        :rules="rules"
        label-width="120px"
        class="demo-ruleForm">
        <el-form-item label="用户名" prop="name">
          <el-input v-model="ruleForm.name"></el-input>
        </el-form-item>
        <el-form-item label="密码" prop="password">
          <el-input type="password" v-model="ruleForm.password"></el-input>
        </el-form-item>
        <el-form-item >
          <el-button type="primary" @click="login()">登录</el-button>
        </el-form-item>
      </el-form>
    </div>
  </div>
```

2、Menu 菜单

垂直菜单，可内嵌子菜单。

通过 el-menu-item-group 组件可以实现菜单进行分组，分组名可以通过 title 属性直接设定，也可以通过具名 slot 来设定。

```
<template>
  <el-aside width="200px" class="aside">
    <el-menu
        :default-active="$route.path"
        class="el-menu-vertical-demo" router>
        <el-menu-item :index="v.url" v-for="v in items" :key="v.url">
          <el-icon><icon-menu /></el-icon>
          <span>{{v.name}}</span>
        </el-menu-item>
    </el-menu>
  </el-aside>
</template>
```

3、Breadcrumb 面包屑

显示当前页面的路径，快速返回之前的任意页面。

在 `el-breadcrumb` 中使用 `el-breadcrumb-item` 标签表示从首页开始的每一级。 该组件接受一个 `String` 类型的参数 `separator` 来作为分隔符。 默认值为 '/'。

```
<div class="breadcrumb">
      <el-breadcrumb separator="/">
          <el-breadcrumb-item v-for="v in lists" :key="v.path">
            <router-link :to="v.path">{{v.meta.title}}</router-link>
          </el-breadcrumb-item>
      </el-breadcrumb>
</div>
```

接口文档

1、登录页

资源URL：http://127.0.0.1:3333/get_login		
HTTP协议：POST		
参数名	**说明**	**是否必须**
user	名称最大255字符	必须
password	类型 String	必须

2、统计管理

资源URL：http://127.0.0.1:3333/tablist		
HTTP协议：GET		
参数名	**说明**	**是否必须**

result:[{各字段},{各字段}]

参数名	说明	
code	200成功	
msg	名称 String 最大255字符	
result	[]	