App.vue这个东西，就是嵌套在index.html中。

App.vue里其实也差不多是空的，我们暂时只关注最上边的<template>，有一行是<router-view></router-view>

**App.vue调用了Hello.vue这个组件，是通过router来控制的显示**

打开router文件夹下的index.js:

1. 上述的小实验已经充分证明了，index.html, App.vue, Hello.vue之间的关系。一般来说，index.html是不用动的（顶多增加一些css头文件）。所以我们开始修改的其实文件就是App.vue，然后鉴于我们想要添加文件，所以需要修改router里边的index.js文件，最后鉴于我们要添加新页面和组件，我们还需要增加一些components文件夹里边的东西。
2. 如果按照目前的index.js的结构，如果你怎么修改那个Hello.vue文件，或者增加其他组件，它们只有一个归宿：就是躺在App.vue里的那个Logo图像的下面。因为App.vue是父组件，而Hello.vue只不过是它的子组件。但是这并不意味着谁重要谁难写，因为如果你把App.vue里边那个logo那一行注释掉，App.vue就什么都无法呈现，仅仅是为Hello.vue以及未来的一些页面提供了一个承载框架而已。
3. 未来开发完的网站里，用户感觉上可能是一个页面和一个url的单独对应，就像是曾经的每一个PHP文件对应每一个url一样，但实际上，可能很多页面，都像Hello.vue一样，仅仅是App.vue的子组件而已。
4. 目前在App.vue里，有Hello组件；以后我们在Hello组件里，还可以插入其他的组件，当然不一定是这种使用router的办法，因为用这个模式，每一个组件都会有一个对应的url，如果你的页面上有三个tab，可能没有必要每一个tab对应一个url吧？但目前要理解的就是：**我个人觉得Vue最核心的设计理念，就是把一个看起来很完整的页面，分散到了每一个组件，这个组件可能大道一个页面，Hello.vue，也可能小到一个按钮，最终我们看到的页面，是这一群组件的堆砌的结果，这样做的一大好处就是，每一个组件都可以单独进行与数据库和用户的交互，开发更为清晰。**

安装elementUI

npm i element-ui –S

合并开发版和生产版插件

Npm install webpack-merge –save-dev

**Vue CLI 脚手架**

**安装CLI3：**

npm install -g @vue/cli

**安装CLI2（模板）：**

npm install -g @vue/cli-init

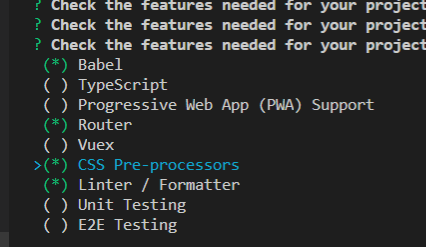
**初始化项目 vuecli2**

vue init webpack vuecli2test



**初始化项目 vuecli3**

vue create [项目名称]



**路由：**

首先在src下创建一个router文件夹 在下面创建index.js

//配置路由相关信息

**router-link**

to :要跳转的路径

eg. to=”/home”

tag:要改变成的标签

eg. tag="button"

replace :不允许退回和前进

active-class:改变活跃的标签

eg.active-class=”active”

**用方法来实现跳转(每个组件都会有一个$router)**

methods: {

homeClick(){

this.$router.push("/home")

},

aboutClick(){

this.$router.push("/about")

}

}

**Vue懒加载**

const Home = () => import('../components/Home.vue')

**URL:**

协议://主机:端口/路径?查询

Scheme://host:port/path?query

**用方法来实现传值：**

profileClick(){

this.$router.push({

path:'/profile',

query:{

name:'acc',

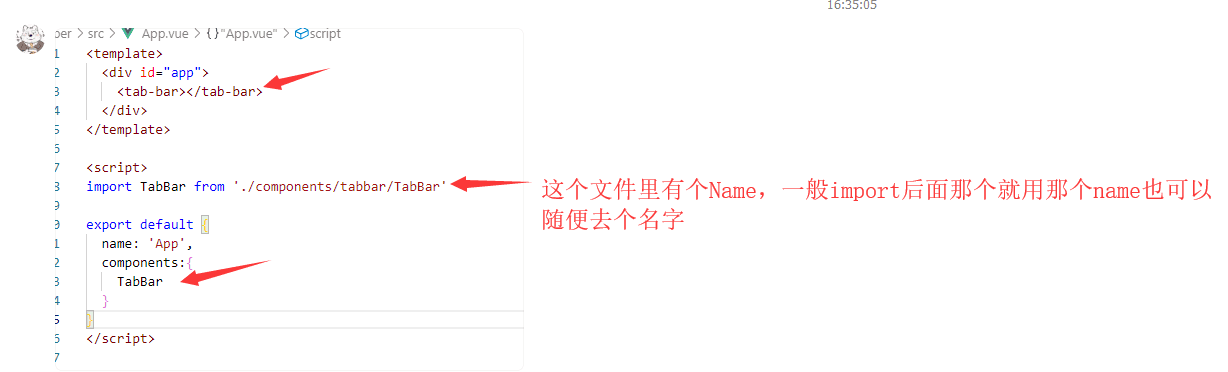
age:'16'

}

})

}

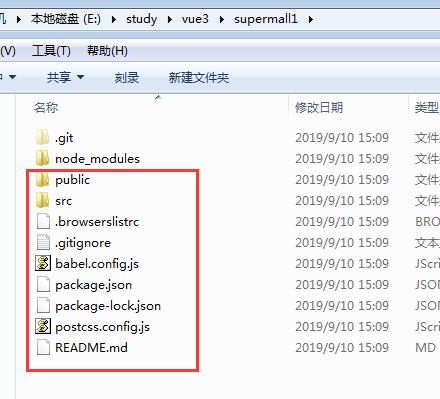
**Keep-alive（116集）**

****

**Git使用（前端）**

1.先复制git上的地址，然后在终端处 git clone 【地址】

2.接着讲原来的文件夹 红框内的文件复制到 clone 下来的 文件夹内



3.终端内：git status（查看当前状态）

git add .

git commit –m ‘初始化项目’ （提交到本地缓存）

git push （提交到服务器）

**划分目录结构**

**src**

|- assets 资源

|- img 图片

|- css 样式

| - normalize.css 第三方基础样式（市面上流行）

| - base.css 自己写的基础样式

|- components 组件

|- common 公共的组件（可在其他项目中使用）

|- content公共的组件（可在其他项目中使用）

|- views 大视图

|- router路由

| - index.js

|- store 公共状态管理（vuex）

|- network 网络相关

|- common 公共的js文件

|- const.js常量

**vue.config.js 自己新建的配置**

样式

v-bind 数据绑定（缩写：）

eg.v-bind:href=””

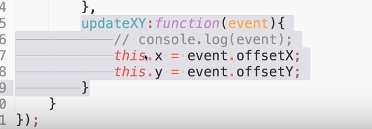
v-html 标签绑定

v-on 事件绑定（缩写@）

eg.v-on:click=”add”

dbclick 双击触发

mousemove鼠标移动

自动获取当前坐标

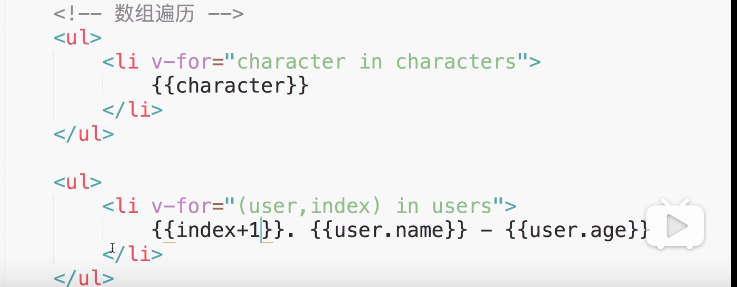
双向数据绑定： input / select / textarea

v-model

eg.v-model:[属性]

computed 计算属性

v-for 数组遍历



index 下标

<template>可以防止过多的div



直接遍历对象的key和val

组件

Vue.component(“[组件命名]”,{[组件结构]})

组件结构：

{

template:’[标签]’,

data:function(){

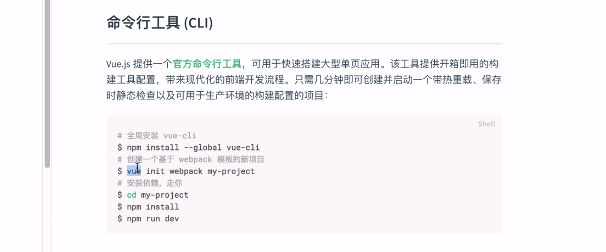
return {[对象]}

}

}

**重点：为什么组件内的data要return对象？因为如果在外面写会变成公共的data**

vue 脚手架



每个模板里：1.html结构 2.处理逻辑 3.解决样式

全局注册组件：

在main.js里

import [对应组件] from ‘./components/Users’

Vue.component(“users”,[对应组件])

css作用域问题： style scoped

**props传值**

v-bind:usersa=”usersb”

props:{

usersa:{

type:Array, -----传递的值的类型

传值：String number boolean 引用：Array Object

required:true ----是不是规格的

}

}

usersb是父组件里的属性

注册事件：

this.$emit(“titleChanged”,”子向父组件传值（传的内容）”)

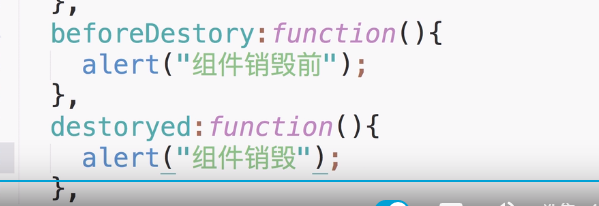
父组件：v-on:titleChanged=”updateTitle($event)”

钩子函数：

beforeCreate:可以弄动画

created:网络请求的一些数据、结束动画





**配置路由：**

安装：npm install vue-router --save -dev

import VueRouter from ‘vue-router’

Vue.use(VueRouter)

const router = new VueRouter({

routes:[

{path:”/”,component:Home},

{path:”/helloworld”,component:HelloWorld},

],

(#号的问题)

mode:”history”

})

底下new Vue 中 加router

app.vue中写

<router-view></router-view>

导航：

<router-link to=”/”></router-link>

<router-link to=”/helloworld”></router-link>

**网络请求：**

安装：npm install vue-resource –-save -dev

import VueResource from ‘vue-resource’

Vue.use(VueResource)

使用钩子函数

created(){

this.$http.get(“”).then((data) =>{console.log(data)})

}

**插槽**slot  
  
**缓存**<keep-alive>  包裹起组件

**axios**

**为什么要用：因为Resource不更新了，所以用axios**

created(){

        axios.get('/',{

        params:{id:12}

        }).then(res =>{

    })

**axios请求方法：get,post,put,patch,delete**

get:获取数据

post:提交数据（表单提交，文件上传）

put:更新数据（所有数据推送到后端）

patch:更新数据（只将修改的数据推送到后端）

delete：删除数据

get:

axios({

  method:'get',

  url:'',

  params:{

    id:12

  }

}).then(res =>{

  console.log(res);

})

//懒加载

component:() => import('./views/About.vue')

post:

let data = {

      id:13

    }

    axios({

      method:'post',

      url:'/post',

      data:data

    }).then(res =>{

      console.log(res);

    })

put:

patch:

delete:

axios.delete('/delete',{

      //在url地址上拼接

      params:{

        id:12

      }

    }).then(res =>{

      console.log(res);

    })

 axios.delete('/delete',{

      //用对象传输

      data:{

        id:12

      }

    }).then(res =>{

      console.log(res);

    })

**并发请求**

**//axios.all() //axios.spread()**

  //并发请求

  axios.all(

    [

      axios.get('/a.json'),

      axios.get('/b.json')

    ]

    ).then(axios.spread((dataA,dataB) =>{

      console.log(dataA,dataB);

    })

  )

**axios实例**

//创建axios实例,之后就可以用instance去做操作

  let instance = axios.create({

    baseURL:'http://localhost:8080',

    timeout:1000

  })

//1.axios 全局配置

  axios.defaults.timeout =1000

  axios.defaults.baseURL='http://localhost:8080'

  //2.axios 实例配置

  let instance = axios.create()

  instance.defaults.timeout = 3000

  //3.axios 请求配置

  instance.get('/data.json',{

    timeout:5000

  })

**拦截器**

//请求拦截器

    axios.interceptors.request.use(config =>{

      //在发送请求前做些什么

      return config

    },err=>{

      //在请求错误的时候做些什么

      return Promise.reject(err)

    })

    //响应拦截器

    axios.interceptors.response.use(res=>{

      //请求成功对响应数据做处理

      return res

    },err=>{

      //响应错误做些什么

      return Promise.reject(err)

    })

//例子 登录状态（token:''）   需要登录的就调用拦截器

    let instance = axios.create({})

    instance.interceptors.request.use(config =>{

      config.headers.token = ''

      return config

    })

    //不需要登录的就不调用拦截器

    let newInstance = axios.create({})

**错误处理**

//错误处理

    axios.interceptors.request.use(config =>{

      return config

    },err=>{

      return Promise.reject(err)

    })

    axios.get('/data.json').then((res) =>{

      console.log(res);

      //错误会被catch

    }).catch(err =>{

      console.log(err);

    })

 //例子：实际开发过程中，一般添加统一错误处理

    let instance = axios.create({})

    instance.interceptors.request(config =>{

      return config

    },err=>{

      //请求错误 一般http状态码以4开头，常见：401超时，404 not found

      $('#model').show()

      setTimeout(() =>{

        $('model').hide()

      },2000)

      return Promise.reject(err)

    })

    instance.interceptors.response(config =>{

      return config

    },err=>{

      //响应错误处理 一般http状态码以5开头，常见：500 系统错误，502 系统重启

      $('#model').show()

      setTimeout(() =>{

        $('model').hide()

      },2000)

      return Promise.reject(err)

    })

  instance.get('/data.json').then(res =>{

    console.log(res);

  }).catch(err =>{

    console.log(err);

  })

**'define' is not defined.'$' is not defined.解决**

