# Vue\_06

# 目标

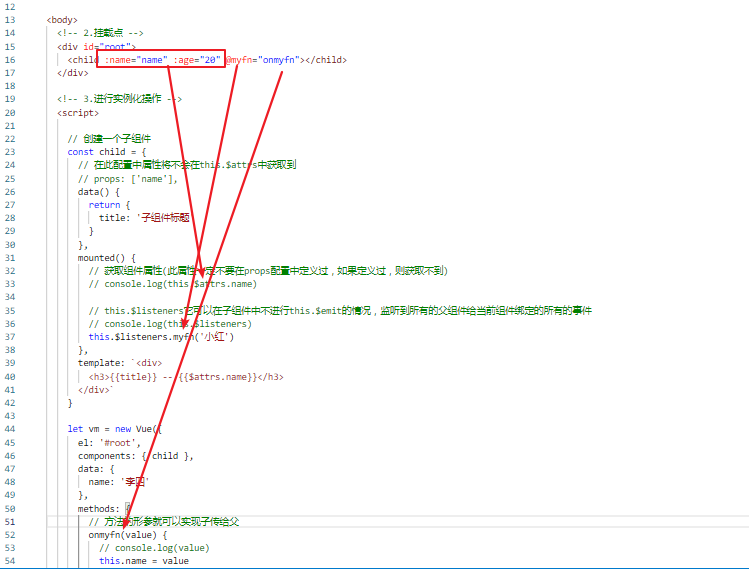
* 定义组件实现组件间传值
* 能够使用插槽对组件复用
* 使用vue-cli完成工程化项目创建
* 使用vue-router构建单页面应用

# 一、组件间传值

## 1.7、$attrs和$listeners

$attrs它把所有没有通过props去设置的属性，全都接受过来

$listeners它是接受所有父组件给子组件自定义的事件，你没有通过$emit抛发



# 二、动态组件

通过使用保留的 <component> 组件，动态地绑定到它的 is 属性，我们让多个组件可以使用同一个挂载点，并动态切换,多组件之间实现tab切换。

<component v-bind:is="currentView"></component>

<script>

var vm = new Vue({

el: '#root',

data: {

currentView: 'home'

},

components: {

home,

posts

}

})

</script>





* keep-alive

如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。

<keep-alive>

<component :is="currentView"></component>

include 写在此属性中的组件将缓存 白名单 包含

exclude 写在此属性中的组件将不缓存 黑名单 排除

</keep-alive>

<script>

var vm = new Vue({

el: '#root',

data: {

currentView: 'posts'

},

components: {

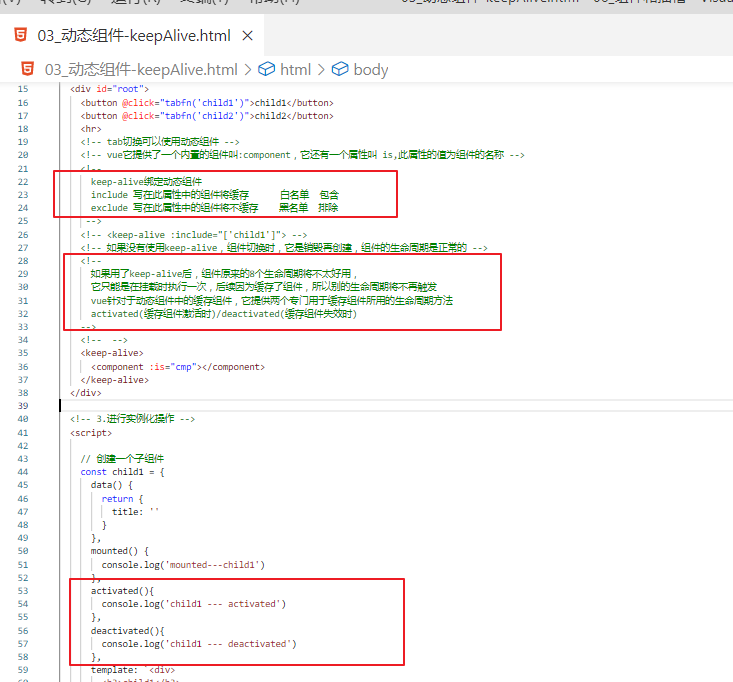
home,

posts

}

})

</script>



# 三、组件插槽

组件的最大特性就是复用性，而用好插槽能大大提高组件的可复用能力

## 3.1、单个插槽

当子组件模板只有一个没有属性的插槽时，父组件传入的整个内容片段将插入到插槽所在的 DOM 位置，并替换掉插槽标签本身。

# 子组件

<div>

<h2>我是子组件的标题</h2>

<slot>

只有在没有要分发的内容时才会显示。

</slot>

</div>

# 父组件

<div>

<h1>我是父组件的标题</h1>

<my-component>

<p>这是一些初始内容</p>

<p>这是更多的初始内容</p>

</my-component>

</div>



## 3.2、具名插槽

有时我们需要多个插槽，来完成对应的数据自定义显示。

一个不带 name 的 <slot> 出口会带有隐含的名字“default”。

自2.6.0 起有所更新。已废弃的使用 slot

# 子组件

<div class="container">

<header>

<slot name="header"></slot>

</header>

<main>

<slot></slot>

</main>

</div>

# 父组件

<app-layout>

// 老写法

<h1 slot="header">这里可能是一个页面标题</h1>

// 新写法

// v-slot 只能添加在 <template> 上

// 简写 v-slot:header == #header

<template v-slot:header>

<h1>这里可能是一个页面标题</h1>

</template>

<p>主要内容的一个段落。</p>

</app-layout>



## 3.3、作用域插槽

作用域插槽是一种特殊类型的插槽，用作一个 (能被传递数据的)可重用模板，来代替已经渲染好的元素。在子组件中，只需将数据传递到插槽，就像你将 prop 传递给组件一样

# 子组件中

Vue.component('child', {

template: `

<div class="child">

<slot name="default" text="我是子组件中的内容"></slot>

</div>

`

})

# 父组件中

<div class="parent">

<child>

// 老写法

<div name="default" slot-scope="props">

<div>父组件</div>

<h3>{{ props.text }}</h3>

</div>

// 新写法

<template #default="props"

<div>

<div>父组件</div>

<h3>{{ props.text }}</h3>

</div>

</child>

</div>



# 四、vue工程化

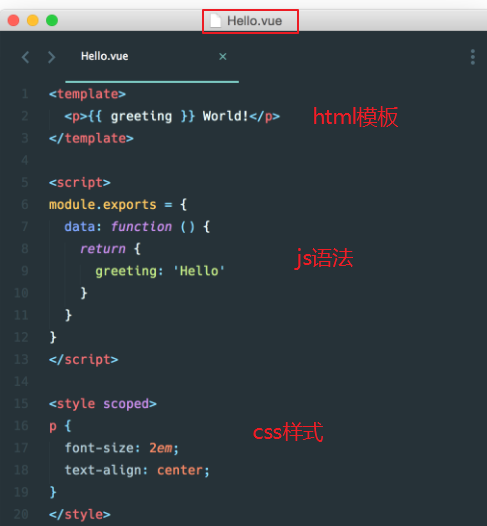
## 4.1、单文件组件

<https://cli.vuejs.org/zh/>

传统的网页引入vue.js使用有如下的问题

* 所有的组件都放同一个html文件中
* 没有构建步骤,不能使用npm来管理项目
* 缺乏语法高亮和提示
* 没有针对单个组件的css样式支持

针对于上述的问题，vue框架发布了vue-cli项目生成工具，Vue-cli是一个基于 Vue.js 进行快速开发的完整系统， 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。



vue-cli 3.x 它还没有在工程化中集成 vue3.0安装选项 dart-sass / node-sass

**vue-cli 4.3**之后 在vue-cli中就集成了 vue3.x安装选项

## 4.2、工具安装

Vue-cli3.x以上版本需要 Node.js 8.9 或更高版本(推荐 v10 以上)。

npm i -g @vue/cli

## 安装成功后，检查

vue --version

## 4.3、创建项目

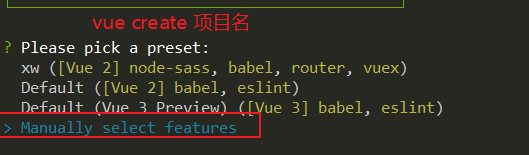
# 首先需要进入到对应的目录中(英文目录不要有空格),执行如下命令

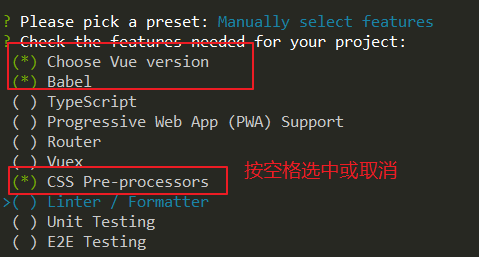
vue create 项目名称(创建时会自己以对应的项目名称生成目录)

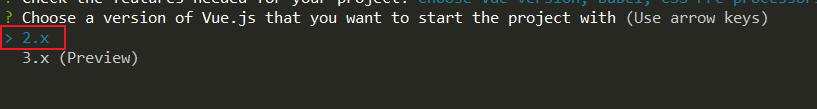
## 例

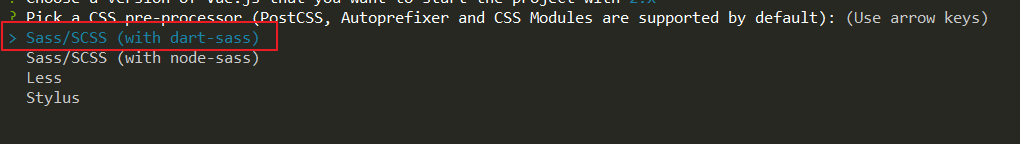
vue create hello-world

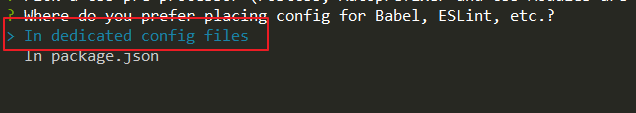
创建步骤



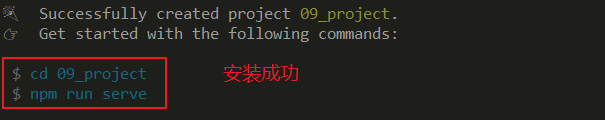












## 4.4、重要目录

public

index.html



src

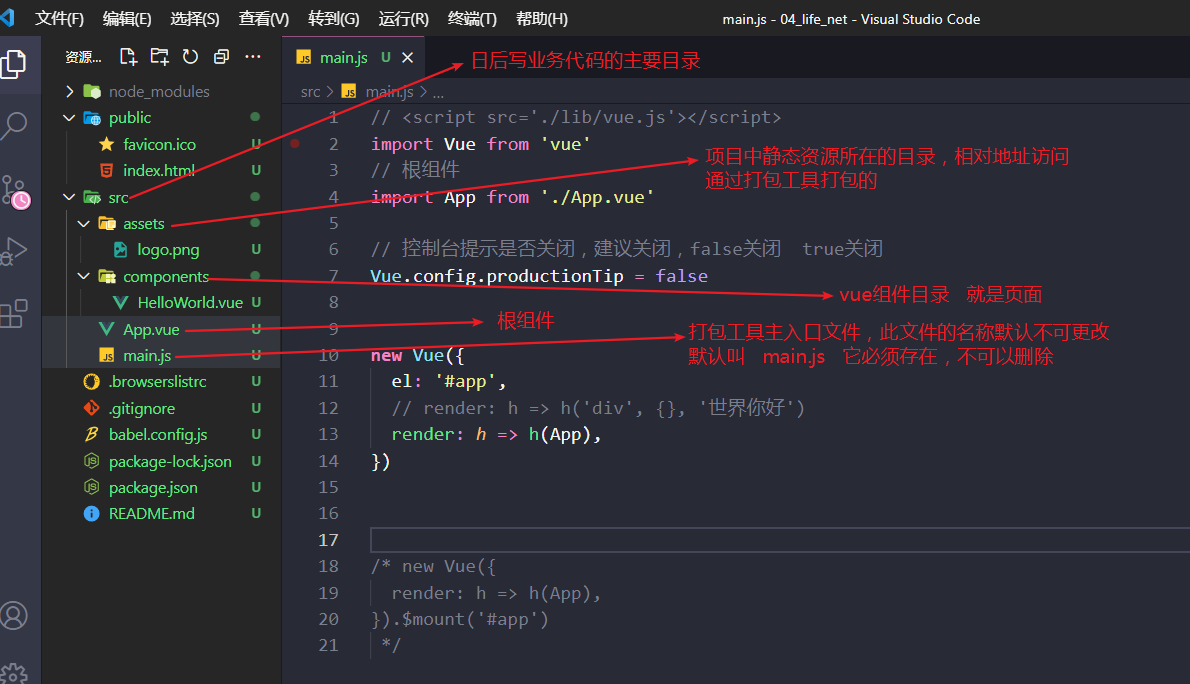
main.js 项目入口文件

App.vue 根组件

components 自定义的功能组件

views 展示视图组件

assets 静态资源目录



# 五、路由

## 5.1、路由概念

路由的本质就是一种对应关系，根据不同的URL请求，返回对应不同的资源。那么url地址和真实的资源之间就有一种对应的关系，就是路由。

路由分为：后端路由和前端路由

* 后端路由：由服务器端进行实现并实现资源映射分发(路由与路由之前切换会刷新页面)
* 前端路由：根据不同的事件来显示不同的页面内容，取事件与事件处理函数之间的对应关系(前端路由切换可能不会有网络请求存在，也不会有刷新页面)

**SPA**(Single Page Application)单页面应用程序,基于前端路由而起：整个网站只有一个页面，通过监听地址栏中的变化事件，来通过局部更新内容信息显示、同时支持浏览器地址栏的前进和后退操作。

## 5.2、前端路由实现

前端浏览器路由也有2种模式:

* hash模式(锚链接)

hash模式兼容性很强，刷新浏览器，页面还会存在

hash模式，缺点：地址栏不优雅，有#存在，不利于seo，记忆困难

hash路由模式是这样的：http://xxx.abc.com/**#**/xx。 有带#号，后面就是hash值的变化。改变后面的hash值，它不会向服务器发出请求，因此也就不会刷新页面。并且每次hash值发生改变的时候，会触发hashchange事件。因此我们可以通过监听该事件，来知道hash值发生了哪些变化。

window.addEventListener('hashchange', ()=>{

// 通过 location.hash 获取到最新的 hash 值

console.log(location.hash);

});

* history模式

history模式，兼容性较差，刷新页面，页面会404，需要服务器端配置支持

优点：地址栏更优雅，方便记忆，有利于seo

HTML5的History API为浏览器的全局history对象增加了该扩展方法。它是一个浏览器（bom）的一个接口，在window对象中提供了onpopstate事件来监听历史栈的改变，只要历史栈有信息发生改变的话，就会触发该事件。

history.pushState({},title,url); // 向历史记录中追加一条记录

history.replaceState({},title,url); // 替换当前页在历史记录中的信息。

window.addEventListener('popstate', function(event) {

console.log(event)

})

注：浏览器地址没有#， 比如(http://localhost:3001/a); 它也一样不会刷新页面的。但是url地址会改变。但它在服务器没有配置的情况下，不能手动刷新，否则返回404页面

## 5.3、Vue Router

网址：<https://router.vuejs.org/zh/>

vue-router通过hash与history两种方式实现前端路由，更新视图但不重新请求页面”是前端路由原理的核心之一，目前在浏览器环境中这一功能的实现主要有两种方式。

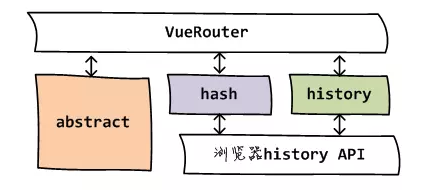
### 5.3.1、介绍

Vue Router 是 Vue.js 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建单页面应用变得易如反掌。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
* 带有自动激活的 CSS class 的链接
* HTML5 历史模式或 hash 模式，在 IE9 中自动降级
* 自定义的滚动条行为

**vue-router路由模式:**

* hash: 使用 URL hash 值来作路由。支持所有浏览器，包括不支持 HTML5 History的浏览器。
* history: 依赖 HTML5 History API 和服务器配置。查看 HTML5 History 模式。
* abstract: 支持所有 JavaScript 运行环境，如 Node.js 服务器端。如果发现没有浏览器的 API，路由会自动强制进入这个模式。



### 5.3.2、安装

如果在vue-cli创建项目时没有勾选上vue-router选项，此时就需要手动的来安装它

npm i -S vue-router

### 5.3.3、Vue Router的基本使用

Vue Router的基本使用步骤：

* 引入相关库文件
* VueRouter引入到Vue类中
* 定义路由组件规则并创建路由实例
* 把路由挂载到Vue根实例中
* 添加路由组件渲染容器

# 路由文件

import Vue from 'vue'

import VueRouter from 'vue-router'

Vue.use(VueRouter)

# 定义路由规则

const routes = [

{ path: '/foo', component: Foo },

{ path: '/bar', component: Bar }

]

# 创建路由实例

const router = new VueRouter({

routes

})

# 挂载根实例

// 记得要通过 router 配置参数注入路由，

// 从而让整个应用都有路由功能

const app = new Vue({

router

}).$mount('#app')

# html

<div>

<router-view></router-view>

</div>

### 5.3.4、声明式导航

它就是先在页面中定义好跳转的路由规则，vueRouter中通过 router-link组件来完成

<router-link to="path"></router-link>

# to 要跳转到的路由规则 string|object

to=”users”

:to=”{path:’path’}”

### 5.3.5、编程式导航

编程式导航就是通过js来实现路由跳转

this.$router.push("/login");

this.$router.replace({ name:'user' , params: {id:123} });

this.$router.push({ path:"/login" });

this.$router.push({ path:"/login",query:{username:"jack"} });

this.$router.go( n );//n为数字 负数为回退

### 5.3.6、路由重定向/404

用户在访问地址A的时候，强制用户跳转到地址C ，从而展示特定的组件页面。

var router = new VueRouter({

// routes是路由规则数组

routes: [

// 每个路由规则都是个配置对象，至少有path和component两个属性

// path表示当前路由规则匹配的hash地址

// component表示当前路由规则对应要展示的组件

{ path: '/', redirect: '/user' },

{ path: '/user', component: User }

]

})

### 5.3.7、嵌套路由

嵌套路由最关键在于理解子级路由的概念：

比如我们有一个/users的路由，那么/users下面还可以添加子级路由，如:

/users/index、/users/add等等，这样的路由情形称之为嵌套路由。

routes: [

{

path: "/user",

component: User,

//通过children属性为/user添加子路由规则

children:[

{ path: "/user/index", component: Index },

{ path: "/user/add", component: Add },

]

}

]

# 需要在 User组件中定义一个router-view 用于嵌套路由的渲染显示

<router-view></router-view>

### 5.3.8、动态路由匹配

所谓动态路由就是路由规则中有部分规则是动态变化的，不是固定的值，需要去匹配取出数据(即路由参数)。

// 传递参数id

var router = new VueRouter({

// routes是路由规则数组

routes: [

{ path: '/user/:id', component: User },

]

})

// 组件中获取id值

const User = {

template: '<div>User ID is {{$route.params.id}}</div>'

}

### 5.3.9、命名路由

通过一个名称来标识一个路由显得更方便一些，特别是在链接一个路由，或者是执行一些跳转的时候。

# 路由

const router = new VueRouter({

routes: [

{

path: '/user/:id',

name: 'user',

component: User

}

]

})

# 声明路由

<router-link :to="{ name: 'user', params: { id: 123 }}">User</router-link>

### 5.3.10、路由守卫

路由守卫就是路由跳转过程中的一些钩子函数，这个过程中触发的这些函数能让你操作一些其他的事儿，这就是路由守卫。

* **全局守卫**

# 全局前置守卫

//当一个路由触发时,触发前置守卫，

to: Route: 即将要进入的目标 路由对象

from: Route: 当前导航正要离开的路由

next: Function: 一定要调用该next方法，否则路由不向下执行。

router.beforeEach((to, from, next) => {

// next函数必须要调用，否则路由不会向下执行

**next()**

})

//全局解析守卫

router.beforeResolve((to, from, next) => {

next()

})

# 全局后置守卫

// 此钩子不会接受 next 函数也不会改变导航本身

router.afterEach((to, from) => {

})

* 局部路由守卫

它的执行的顺序比beforeEach要晚，但早于beforeResolve

const router = new VueRouter({

routes: [

{

path: '/foo',

component: Foo,

beforeEnter: (to, from, next) => {

// ...

next()

}

}

]

})

* 组件内守卫

它比全局前置要慢，比路由独享要慢，快于全局解析路由守卫

你可以在路由组件内直接定义以下路由导航守卫

const Foo = {

template: `...`,

beforeRouteEnter (to, from, next) {

// 在渲染该组件的对应路由被 confirm 前调用

// 不！能！获取组件实例 `this`

// 因为当守卫执行前，组件实例还没被创建

},

beforeRouteUpdate (to, from, next) {

// 在当前路由改变，但是该组件被复用时调用

// 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，

// 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。

// 可以访问组件实例 `this`

},

beforeRouteLeave (to, from, next) {

// 导航离开该组件的对应路由时调用

// 可以访问组件实例 `this`

}

}

### 5.3.11、路由元信息

定义路由的时候可以配置 meta 字段：

const router = new VueRouter({

routes: [

{

path: '/admin',

component: Admin,

meta: {login:true}

}

]

})

### 5.3.12、路由懒加载代码拆分

当打包构建应用时，js包会变得非常大，影响页面加载。如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。

const Foo = () => import(/\* webpackChunkName: "group-foo" \*/ './Foo.vue')

const router = new VueRouter({

routes: [{ path: '/foo', component: Foo }]

})