Vue模块化环境搭建

1、`@vue/cli` 脚手架

- Vue模块化项目环境，可以通过官方提供的环境构建起 `vue-cli` 进行自动创建
- 环境依赖检测
  - node环境：`node -v`
  - npm环境：`npm -v`
- 脚手架安装
  - 提供通过命令行的方式快速 构建 vue 运行环境项目
  - cli ：(命令行接口工具 = 控制台使用的命令扩展)
  - 通过npm 方式安装创建 `npm install @vue/cli -g`
    - 系统命令行工具将创建一个主命令 `vue`，通过 `vue -V` 验证安装状态和版本
    - `vue --help` `vue -h` 查看vue环境的帮助手册

```
C:\Users\User>vue --help
Usage: vue <command> [options]

Options:
  -V, --version   查看当前vue\cli版本号
  -h, --help   在控制台输出帮助命令

Commands:
    命令行中 [] 表示可选命令    <> 表必须命令
  * create [options] <app-name>   根据开发者提供的项目名称创建项目，
                    项目名称 不能使用驼峰方式
                    项目名称 最好不要使用中文
  * add [options] <plugin> [pluginOptions]   为项目增加扩展插件功能 ==>npm
install <plugin>
                  自动识别 --save    --save-dev 自动安装依赖
  invoke [options] <plugin> [pluginOptions] 对项目中的插件进行 配置更新
  inspect [options] [paths...]   调整脚手架服务器的配置
  * serve [options] [entry]    驱动简易的vue环境
  * build [options] [entry]    基于webpack打包项目-将JS CSS html …… 语法转换为兼
容语法，
                             生成纯静态文件 vue build == npm run build
  * ui [options]    开启 vue-cli 管理器页面 -- 通过图形化管理电脑中的所有vue项目
  init [options] <template> <app-name> 根据外部模板创建功能
                                    vue-cli 2.0 版本项目构建旧项目结构
                                    依赖于 npm install -g @vue/cli-init
  config [options] [value] 配置文件的修改操作
  upgrade [semverLevel] vue项目的配置升级
  info 用于启动查看 调试信息
  Run vue <command> --help 查看子命令的帮助手册
```

2、基于`@vue/cli`创建模块项目

1. **切换到需要存放项目的目录**
2. **在正确的目录下执行** `vue create 项目名称`，注意：项目名称不能出现驼峰方式，不要定义中文字符

- 上述命令执行后会进入 REPL(可交互控制台) 环境

```
# 1、选择构建模式
Vue CLI v3.10.0
? Please pick a preset: (Use arrow keys)
❯ default (babel, eslint)  # 默认选项（只包含基础选项）
  Manually select features # 自定义环境选择
#2、选择 自定义环境 构建后
Vue CLI v3.10.0
? 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) # 选择环境支持语法
❯◉ Babel # ES6语法兼容转换器
 ◯ TypeScript # 使用 TS 语法
 ◯ Progressive Web App (PWA) Support # 构建 渐进式WEB应用
 ◯ Router # 集成路由功能
 ◯ Vuex # 集成统一数据状态管理器
 ◯ CSS Pre-processors # 启动 CSS 预编译功能（让项目支持使用 LESS SASS 等动态样式语言）
 ◉ Linter / Formatter # 启用语法校验和格式化检测插件
 ◯ Unit Testing # 启动单元测试（单文件测试）
 ◯ E2E Testing # 启动端到端测试（黑盒测试）
#3、配置完成后，如果选择了对应选项会进入固定的配置选项
#3.1、如果选择了 Router 选项，开启路由模式切换选项
 ? Use history mode for router? (Requires proper server setup for index fallback
in production) (Y/n)
#3.2、如果选择了 CSS Pre-processors ，开启动态语言选择
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported
by default): (Use arrow keys)
❯ Sass/SCSS (with dart-sass)
  Sass/SCSS (with node-sass)
  Less
  Stylus
#3.3、如果选择 Linter / Formatter，开启语言校验
? Pick a linter / formatter config: (Use arrow keys)
❯ ESLint with error prevention only # 仅检测错误
  ESLint + Airbnb config  # 使用 Airbnb 前端规范
  ESLint + Standard config # 使用标准规范
  ESLint + Prettier # 使用严格规范
#3.3.1、选择语法校验时间
? Pick additional lint features: (Press <space> to select, <a> to toggle all,
<i> to invert selection)
❯◉ Lint on save # 文件保存时
 ◯ Lint and fix on commit # 代码整理和提交时
#3.4、如果选择了 Unit Testing，进入单元测试工具选择
? Pick a unit testing solution: (Use arrow keys)
❯ Mocha + Chai
  Jest
#3.5、如果选择了 E2E Testing，进入端到端测试工具选择
? Pick a E2E testing solution: (Use arrow keys)
❯ Cypress (Chrome only)
  Nightwatch (Selenium-based)
#4、选择项目构建时，工具配置文件所定义位置
? Where do you prefer placing config for Babel, PostCSS, ESLint, etc.? (Use
arrow keys)
❯ In dedicated config files  # 独立文件定义
  In package.json # 集成与 package.json 文件中
#5、是否将上述配置 存储为 一个固定选项，提供下次使用
```

```
? Save this as a preset for future projects? Yes
? Save preset as:
#6、自动进入项目依赖安装--注意：该步骤不执行完成，只会构建文件夹和package.json文件
Vue CLI v3.10.0
✨  Creating project in /Users/appleuser/Desktop/aa-aa.
□  Initializing git repository...
⚙  Installing CLI plugins. This might take a while...

□ ░░░░░░░░░░░░░░░░░░░░░░░░░░░░□ ⠋ fetchMetadata: sill pacote range manifest for css-
loader@^1.0.1 fetched in 525ms
```

3、项目启动

- 切换到项目的开发目录，执行 `npm run serve` 启动项目
  - `npm run 命令名称` 是npm内置的脚本执行命令，该命令会自动搜索执行目录下 `package.json` 文件中的 `script` 对应的命令执行
- 启动完成后会在控制台提示访问地址

```
 DONE  Compiled successfully in 18063ms
                               9:04:39 PM
  App running at:
  - Local:   http://localhost:8080/
  - Network: http://192.168.0.160:8080/
  Note that the development build is not optimized.
  To create a production build, run npm run build.
```

3、项目结构构成

App.vue文件和单文件组件

```
<template>
    <!-- 组件模板区 => 定义当前组件的 页面结构 -->
    <div>

    </div>
</template>

<script>
    // 定义组件的 配置项
</script>

<style>
    /* 定义组件的 样式 */
</style>
```

- 组件开发时的图片路径问题
  - 页面中使用 img 标签方式加载图片，且图片为相对路径，写入的路径会被vue进行加工
    - 在**编译过程中会根据 组件的位置 寻找图片，编译后会使用 baseUrl的值为路径进行重新设置**
  - 在==行内样式==中使用 背景图片方式，图片地址 不会被vue解析编译
    - 图片地址以 index.html 文件作为参考文件
  - 单文件组件中 定义的 css 样式的图片地址 是会被vue进行加工处理的

- 如果图片的路径地址是以 vue 指令方式进行绑定的，vue在处理图片路径时 不会加工变量值
  - 图片地址以 index.html 文件作为参考文件

4、单文件组件中的样式构建

4.1、基本样式规则

- 在单文件组件中，提供 style 标签可以完成组件样式的定义 **默认是全局样式定义**
- 单文件组件，可以通过对 `<style scoped>` 标签定义 scoped 属性，完成将样式限制于当前组件的标签上
- 单文件组件在定义样式时，并不限制 style 标签的出现次数

```
<style>  全局样式定义
*{
  margin: 0px;
  padding: 0px;
}
</style>

<style scoped>   局部样式定义
  h1{
    color: red;
  }
</style>
```

- 项目中less语法的使用

1. 全局变量定义:在项目的src目录下，任选位置 定义一个 专门存放变量的 less 文件即可

```
<style lang="less" scoped>
@import "../less/var.less";  // less语法的 装载 外部less文件
.content{
    background-color: #dedede;
    border: 1px solid black;
    .title{
        color: @blue;
    }
    .list{
        font-weight: bolder;
        border: 3px solid @blue;
    }
}
</style>
```

1. **修改项目全局启动配置文件 vue.config.js 文件**
   - vue/cli3.0 脚手架 生成的默认项目时没有项目的配置文件vue.config.js
   - 可以在项目目录下手动创建 vue.config.js 文件

```
// node语法自动导出 模块module.exports
module.exports = {
  // 三方插件的 加载配置项
  pluginOptions: {
    'style-resources-loader': {
      preProcessor: "less",  // 需要解析的 样式资源后缀
```

```
      patterns: [
         // 定义 less 全局样式的文件地址,以当前文件作为参考
         "./src/less/var.less"
      ]  // 定义语法规则
    }
  }
}
//配置服务器端口
    devServer:{
        // 代理服务器配置,实现在开发环境解决开发跨域问题
        // proxy:Object
        // Object 定义代理服务器的相关代理规则
        port:8080,//修改端口
        proxy:{
            "/api":{
                target:"http://127.0.0.1:8080", // 定义被拦截请求的真实访问服务
器
                // 拦截前缀的重写规则
                pathRewrite:{
                    // key 匹配的重写 正则 规则
                    "^/api":""
                },
                // 在项目开发服务器的启动控制台进行 日志输出
                logLevel:"debug"
            }
        }
    },
```

项目打包

- 构建脱离环境依赖和开发服务器的纯静态文件构成的项目，**让项目可以在任意服务器上运行**
- **在项目目录下执行 ：**`npm run build`
  - **上述命令在当前项目目录下生成一个特定文件夹(默认dist)，该文件夹中存放打包后的项目静态文件**
    - 将public文件夹下的资源文件直接拷贝到 dist 目录中
    - 会将 src/assets 目录中 **使用但没有被转换**的 静态文件 拷贝到 dist 目录中，并存于对应文件夹中，例如 css 存放于css文件夹，js存放于js文件，图片存于 img文件，font存放于font文件夹.....
    - 会将 src 下以 main.js 文件引导的所有 文件代码 ，分别打包成 app.js 和 app.css 文件进行定义
  - **打包后会提供产品报表，记录了生成的文件信息，和大小**

7.1、axios安装集成

- 通过npm 进行模块安装`npm install axios --save`，`npm install axios -S`
- 在项目需要使用的位置，通过ES6 的模块导入语法`import.....from.....`进行模块加载
- 该模块加载完成后，提供异步请求对象 `axios`，该对象包含相关的请求方法
  - get 类型请求

```
- axios.get(url[, config])
- axios.delete(url[, config])
- axios.head(url[, config])
- axios.options(url[, config])
```

- post 类型请求

```
- axios.post(url[, data[, config]])
- axios.put(url[, data[, config]])
- axios.patch(url[, data[, config]])
```

7.2、axios的基本使用

1、get类型请求

```
// 为给定 ID 的 user 创建请求
axios.get('/user?ID=12345')
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });

// 可选地，上面的请求可以这样做
axios.get('/user', {
    params: {
      ID: 12345
    }
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

post请求

```
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

8、路由

- 在页面中提供操作接口，让用户可以通过简单操作方式，完成组件的切换展示

- 路由是一种组件动态化分发机制，通过**模拟 URL** 路径变换 ，寻找对应的组件并将组件渲染页面显示到对应的位置

8.1、添加路由模块

1、自动安装

- 在创建项目时 `vue create 项目名称` 通过选择路由功能自动添加
- vue/cli 3.0 版本中对以创建的项目，通过下述方式进行自动添加(**创建完项目未做代码修改时可用**)
  - 1、通过命令 `vue add router` 进行插件安装的同时配置路由功能
  - 2、启动 GUI 页面 `vue ui` 通过添加路由按钮进行自动换添加

> **执行上述命令进行路由添加，都会直接覆盖 main.js 和 App.vue 文件，会创建view目录生成两个页面文件Home.vue,About.vue**

2、手动安装路由

- **路由功能依赖于模块 vue-router，安装路由插件** `npm install vue-router -S`
- 配置项目路由功能
  1. 在当前项目的src 目录下创建 **router/index.js** 文件 ==> 完成路由模块的功能配置
  2. 在main.js 文件中导入并装载路由功能 ==> 为项目加载路由功能

8.2、路由配置和使用

1、基本配置和使用

- 路由模块在Vue原型配置增加了一个新的配置 `router` 接收一个 `VueRouter`对象，用于记录路由表
- 路由功能通过`new VueRouter(options)` 配置完成定义

```
options={
  // 激活路由样式配置
  linkActiveClass:"class名称",
  linkExactActiveClass:"class名称",
  // 路由模式切换
  mode:"history|hash",
  // 描述组件和地址的关系 ==> 当URL地址变换后，vue知道该地址应该渲染那个组件
    // 路由中定义的组件 以及被路由进行统一管理，不需要在使用页面中定义为局部组件
  routes:[
    {
      path:"定义组件地址",
      name:"路由别名"
      component: "定义组件",
      redirect: "目标地址",
      meta:"定义自定义数据",
      children:"定义子路由"
    }
  ]
}
```

- **Vue.use(VueRouter)**提供两个用于路由显示和切换的全局组件
  1. `<router-view></router-view>` 提供页面组件占位符号

2. `<router-link></router-link>` 用于切换和跳转组件

2、地址定义

- 基本地址定义
  - 通过配置path属性描述组件和地址关系，一级路由定义时必须使用`/`开头，子级路由可选择绝对路径或相对路径
- 默认地址定义
  - **vue路由默认展示地址为 `/` 的组件，**因此以`path:"/"`描述默认组件
  - 默认地址一般使用路由定向方式跳转到其它组件上：`{path:"/"，redirect："目标地址"}`
- 通配地址定义
  - **对没有在路由表中定义的路径，vue会进行统一的地址处理**
  - 通过配置**path通配符** `*`，描述路由表中不存在的地址默认访问的组件
    - `{path:"*"，redirect："目标地址"}`
    - `{path:"*"，component：组件}`
- 多级路由定义
  - 对于具有多级导航的项目而言，需要使用到多级路由功能
  - **router-view 和多级路由的关系，只依赖路由组件的关系，不受其他组件影响**

3、路由模式切换和定义

- 通过配置 mode 取值，可修改路由模式
  - hash模式 ： 以 html 中 锚点技术实现的 路由匹配切换
    - 锚点 可以更改 url地址的同时 ，不刷新页面的方式实现
    - 通过 window.location.hash 获取 锚点值，通过锚点值 匹配组件，进行渲染展示
  - history 模式 ： 历史地址模式
    - 该模式不能单独使用，必须需要后台通过代码进行 配合使用
    - 将地址转换为正常的 URL地址，**通过后端代码的拦截处理时实现页面不刷新组件更新的效果**

8.3、激活路由的控制

- 激活路由：当前URL地址展示的组件 对应的路由配置 叫做激活路由

1、激活路由样式控制

- **Vue.use(VueRouter)**提供两个用于描述激活路由样式的class样式名
  - `.router-link-active` 样式
    - 该样式被自动定义在`<router-link>`标签上
    - 样式随着激活路由的变化，选择出现在**路径中包含当前路径**的`<router-link>`标签上
  - `.router-link-exact-active` 样式
    - 该样式被自动定义在`<router-link>`标签上
    - 样式随着激活路由的变化，选择出现在**路径完全相同**的`<router-link>`标签上

2、激活路由信息对象

- **Vue.use(VueRouter)**提供组件全局数据仓库 `$route`
  - 路由模块通过对 Vue 原型的修改，为 ==所有的 vue实例的数据仓库 增加一个 特殊变量`$route`,==

- `$route`是激活路由信息==对象==，存储当前激活路由的相关信息
- `$route`为组件共享数据仓库，==所有组件的route 都是同一个对象==
- `$route`为只读对象，无法被修改

```
{
    path:"/detail"    // 描述当前激活路由的 路径地址
    query:Object (empty)    //  存放是当前激活路由  get 形式传递的参数
    params:Object (empty)   //  存放是当前激活路由  rest 形式传递的参数
    fullPath:"/detail"    // 路由地址和参数地址形成的完整路径
    meta:Object (empty)   //  激活路由的原始信息
}
```

3、激活路由原信息

- meta 属性在构建路由时 ，为当前路由定义的 自定义配置项，**用于页面特殊构成的配置**

8.4、路由切换传参

- 路由参数常用与对相同组件传入不同数据，实现组件页面展示切换
- 注意事项：
  - **1、路由参数不要过长或过多**
  - **2、路由参数传递关键参数，用于后台数据查询**

1、get方式参数

- 以 HTTP get请求传递参数的方式，将参数传递下一个组件上
  - get 方式传递参数，在页面特定路径情况下，不会刷新页面
  - 在vue的路由构成中，get参数虽然会导致路径变化，但不会影响路径对组件指定
- 1、在页面中对路由的跳转地址 **以 ？和 & 方式定义动态参数**
- 2、在目标组件中以激活路由对象 `$route` 进行参数的获取
- **vue路由项目中，如果需要实现 rest 参数传递，需要配合路由的相关配置**

实现方式

- 1、在页面定义请求路径时，以 URL的地址定义规范拼接参数的**值**...
- 2、在vue的路由定义中，需要在path属性构建时，明确表示地址存在几个部分且那些部分为参数
  - 路由地址配置属性 `path` ，可借助关键字 `:` 描述地址那些部分为参数
  - `:`后所定义的名称，将成为该参数的变量名

```
{ path:"/detail/:id" }
```

- 3、在目标组件中以激活路由对象 `$route` 进行参数的获取
  - 配置路由path时，`:`后定义的变量名，将作为参数名写入 `params`对象中，用于指代参数

```
vm.$route.params.参数名
```

3、router-link的to属性定义

- to 作用是用于完成组件切换指向，定义携带参数
- 当to属性以 v-bind 方式进行数据绑定时，可用于拆分地址和参数

- 取值string 类型 ：
  - get : 传统的 string路径定义方式 **to="路径?参数名=参数值&参数名=参数值……"**
  - rest : **to=" ……路径/参数…… "**
- 取值object类型 ：以对象方式拆分 路径的 组件地址 和 参数
  - get ：**to=" { path:'组件的路径地址'， query:{ key:value,key:value } } "**
  - get ：**to=" { name:'组件的路径地址'， query:{ key:value,key:value } } "**
  - rest ：**to=" { name:'路由组件的别名'， params:{ key:value,key:value } } "**

8.5、编程式导航

- 通过JS代码控制路由切换，统称为编程式导航
- **Vue.use(VueRouter)**提供组件全局实例属性 `$router`
  - $router 对象中，存放编程式导航的执行方法，因此该对象被称为 路由控制对象
- $router 的常用方法
  - $router.push(`location`) 跳转到指定页面
    - location 参数 以string 方式定义跳转目标组件的地址和参数
    - location 参数 以object 方式定义跳转目标组件的地址和参数
  - $router.go(n) 模拟浏览器的 前进按钮
  - $router.back(n) 模拟浏览器的 后退按钮

8.6、路由缓存

- Vue为了优化组件在浏览器中使用率和内存占用量上，做了路由组件的 ==缓存操作==
  - 如果发现跳转的地址指向的组件 ，为当前激活路由组件（**组件相同**），vue将不再对当前组件进行重新渲染绑
  - 因为组件缓存导致组件不重新渲染，因此组件mounted钩子函数之前的生命周期将不再被执行