Vue模块篇

(陈华旺-chenhuawang@itany.com)

目录 [Vue模块篇]

1、组件化与模块化

2、模块化环境构建

- 2.1、模块环境分类
- 2.2、Vue模块化独立环境搭建
  - 1、`@vue/cli` 脚手架
  - 2、基于`@vue/cli`创建模块项目
  - 3、项目启动

3、项目结构构成

- 3.1、文件结构
- 3.2、启动流程和组成文件
  - 1、index.html
  - 2、main.js
- 3.3、App.vue文件和单文件组件
  - 1、App.vue
  - 2、单文件组件
- 3.4、页面构建时的注意事项

4、单文件组件中的样式构建

- 4.1、基本样式规则
- 4.2、动态样式

5、模块化的开发规范

6、项目打包

7、模块化项目的数据交互

- 7.1、axios安装集成
- 7.2、axios的基本使用
  - 1、get类型请求
  - 2、post类型请求
- 7.3、跨域请求的分类和处理方式
  - 1、线上跨域（生产环境跨域）
  - 2、开发跨域（开发环境跨域）
  - 3、vue模块化项目开发跨域的代理服务器配置
- 7.4、AJAX请求的API模块化

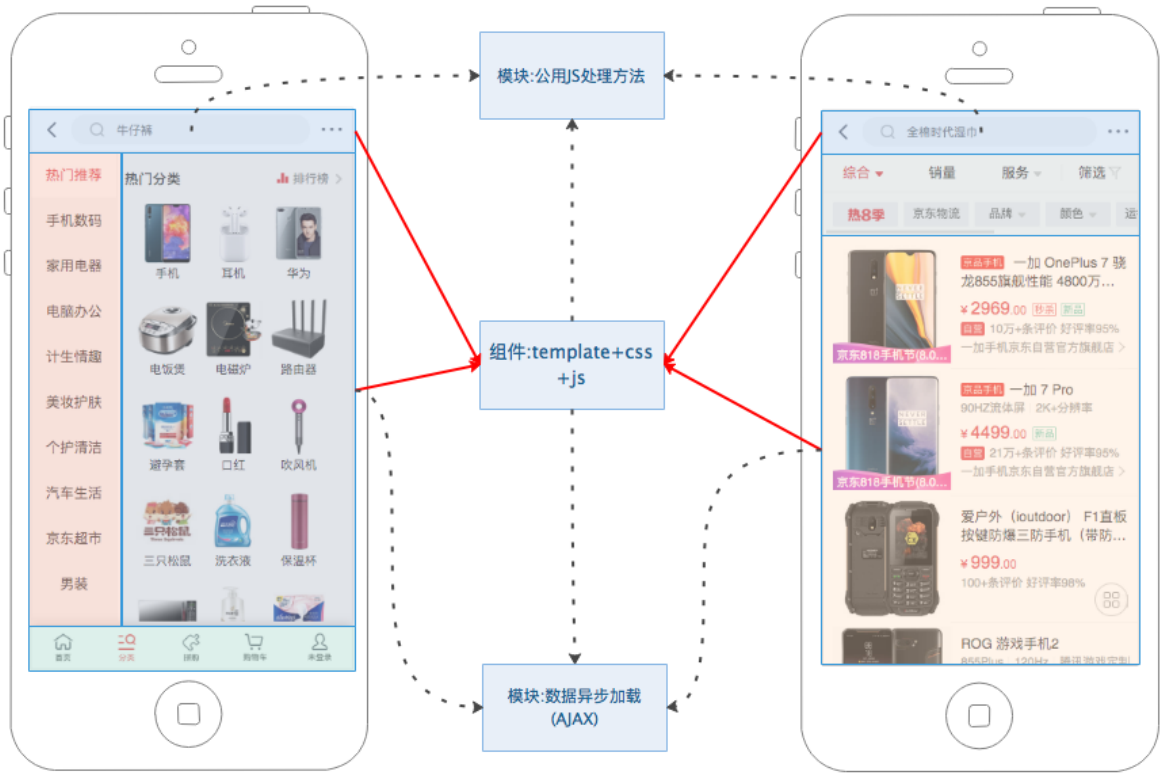
8、路由（vue-router）

- 8.1、添加路由模块

- 1、自动化安装
- 2、手动安装路由
- 8.2、路由配置和使用
- 1、基本配置和使用
- 2、地址定义
- 3、路由模式切换和定义
- 8.3、激活路由的控制
- 1、激活路由样式控制
- 2、激活路由信息对象
- 3、激活路由原信息
- 8.4、路由切换传参
- 1、get方式参数
- 2、rest方式
- 3、router-link的to属性定义
- 8.5、编程式导航
- 8.6、路由缓存
- 8.7、导航守卫
9、全局功能定义

1、组件化与模块化

- 什么是模块化？
  - 将一个复杂的程序依据一定的规则(规范)封装成几个块(文件), 并进行组合使用
  - 块(文件)的内部数据与实现是私有的, 只是向外部暴露一些接口(方法)与外部其它模块通信交换
- 什么是组件化？
  - 组件是以项目页面构成，将页面进行单元文件查分，用于复用重组页面结构
  - 组件包含页面构成(template)，样式(css)，功能(js+模块调用)
- 组件和模块



- 模块化优点(可维护性)
  1. 灵活架构，焦点分离（单一职能）
  2. 方便模块间组合、分解
  3. 方便单个模块功能调试、升级
  4. 多人协作互不干扰
- 模块化缺点(性能损耗)
  1. 系统分层，调用链会很长
  2. 模块间通信,模块间发送消息会很耗性能

2、模块化环境构建

2.1、模块环境分类

- 页面环境：传统页面构建模块化环境，主要基于模块化插件 `RequireJS` 、`CommonJS` 和 `SeaJS`
  - AMD (Asynchronous Module Definition-异步模块定义):
    - **是RequireJS提出的** 一个**依赖前置、异步定义**的模块化加载框架，在构建功能的同时如果需要用到别的模块，需在最前面定义好模块文件的引入。
  - CMD (Common Module Definition通用模块定义):
    - 是淘宝团队开发的 **SeaJS提出的** 一个**依赖就近，同步加载**的模块框架，在构建功能时什么地方使用到模块，就在什么地方加载模块，即用即返。
  - CommonJS (通用JS)
    - CommonJS是一个JS开发的统一规范定义，该规范致力于提供一个类似java、python等语言的标准库，使JS可以开发服务器端应用程序、命令行工具、桌面图形界面应用程序
    - NodeJS 就是基于 CommonJS 规范构建的一套基于服务器端的模块系统架构
- 独立环境构建(开发环境构建)
  - 多数前端模块框架都需要基于一个固定的开发环境，构建独立的模块开发系统，这些系统多数都基于 NodeJS 和 webpack 进行环境搭建
    - NodeJS 独立JS运行环境和模块系统支持（基于commonJS语法开发的）
    - webpack 前端构成工作流工具库

2.2、Vue模块化独立环境搭建

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`
  - nrm环境：`nrm ls`
    - nrm 安装 `npm install nrm -g` （要使用管理员权限 ：windows 系统右键 ，mac|liunx `sudo npm install nrm -g` ）
- 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 ： commond line interface 命令行接口
  - 通过npm 方式安装创建 `[sudo] 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

命令行中 [] 表示可选命令    <> 表必须命令
Commands:
    根据开发者提供的项目名称创建项目  ==> 会在指定目录下构建项目文件夹
        项目名称 不能使用驼峰方式
        项目名称 最好不要使用中文
  * create [options] <app-name>
    为项目增加扩展插件功能 （只会按照脚手架能够识别的模块）
        自动识别 --save 和 --save-dev 环境 ==> 自动执行 npm install
        自动安装依赖，自动添加默认配置 ==> 自动完成项目 vue.config.js 文件配置

  * add [options] <plugin> [pluginOptions]
    对项目中的插件进行 配置更新
  invoke [options] <plugin> [pluginOptions]
    项目配置输出
  inspect [options] [paths...]
    驱动简易的vue环境
  serve [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
  build [options] [entry]
    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
  * ui [options]
    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
        依赖于额外全局模块@vue/cli-init ==> npm install -g @vue/cli-init
  init [options] <template> <app-name>
    配置文件的修改操作
  config [options] [value]
```

```
    vue项目的配置升级
  upgrade [semverLevel]
    用于启动查看 调试信息
  info
    查看子命令的帮助手册
  Run vue <command> --help
```

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） == 直接执行第6步
  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
```

> vue/cli3.0 特色功能,ui页面构成
>
> `vue ui` 开启图形化管理界面 http://localhost:33861/project/select

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

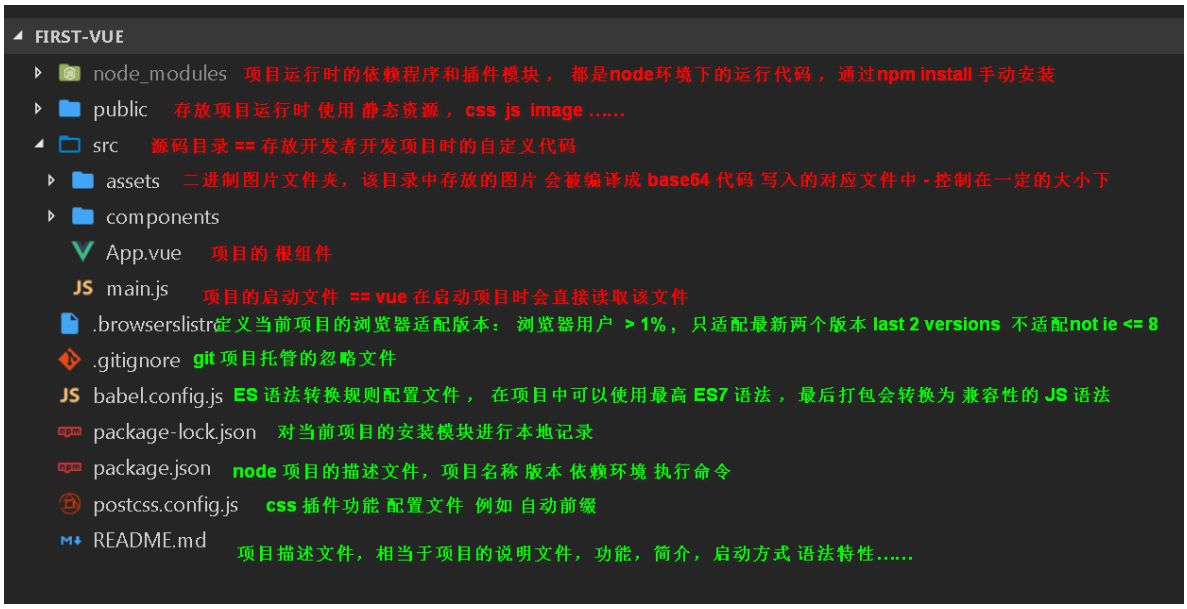
```
DONE  Compiled successfully in 18063ms                    9:04:39 PM

 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

3.1、文件结构

- **除public目录以外的其它目录和文件，都是项目构成时的依赖关联文件**
  - 通过多个关联文件构成组件化和模块化开发环境，最终由webpack工具将独立文件合并



- **public文件夹，描述当前模块化项目的对外公开资源==>启动服务器的根目录**
  - 定义一些被地址直接可以访问的静态资源，js文件，css文件，图片文件，视频文件……

3.2、启动流程和组成文件

- `npm run serve` 命令执行后，会从当前启动目录寻找 vue 项目的主要引导启动文件 `index.html` 和 `main.js` 文件

1、index.html

- index.html 为用户页面入口文件，为项目通过地址访问时，需要展示的HTML静态主页
- 该文件也是vue项目的容器定义文件

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <!--
      描述当前网站的图标文件
      index.html 中的路径定义语法 <%= BASE_URL %>
        => <%= BASE_URL %> 描述项目所在的服务器目录的
    -->
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">

    <title>film-project</title>
  </head>
  <body>
    <noscript>
```

```
      <strong>We're sorry but film-project doesn't work properly without JavaScript enabled. Please enable it to continue.</strong>
    </noscript>
    <div id="app"></div>
    <!--
        构建的文件将被装载到该注释的后面
        装载项目目录下除去 public 目录中的静态资源以外的其他文件，打包后的JS整合文件（app.js）
    -->
    <!-- built files will be auto injected -->
  </body>
</html>
```

2、main.js

- vue项目的组件、模块整合入口文件，项目启动依赖于当前文件，加载整合和编译最终运行的js代码

```
// ES6 模块导入语法 == 等效于 node   require（） 功能
import Vue from 'vue'
import App from './App.vue'

// 全局配置 关闭产品控制台提示
Vue.config.productionTip = false

// 创建当前项目运行的 根实例组件
new Vue({
  // template:"",
  render: h => h(App),
  // render:(h)=>{
  //   return h(App);
  // },
}).$mount('#app')
```

3.3、App.vue文件和单文件组件

1、App.vue

- App.vue文件为模块化项目的基础构成组件，该组件用于统一管理vue组件的构成和页面结构

2、单文件组件

- App.vue 为vue提供的一种**特殊的 组件定义文件**，该文件用于描述模块化项目中，组件的基本构成代码
  - 包含页面结构 template
  - 组件功能 script
  - 组件样式 style
- 单文件组件在模块化项目运行时，通过`import`方法时装载前，**会被vue构造成 组件创建时的构成配置**

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

    </div>
</template>
```

```
7
8  <script>
9      // 定义组件的 配置项
10 </script>
11
12 <style>
13     /* 定义组件的 样式 */
14 </style>
```

3.4、页面构建时的注意事项

- 错误信息的展示
  - 1、系统控制台（cmd）==> 编译错误 模块找不到，标签不对应，环境读取错误，资源不存在
  - 2、HTML页面 ==> HTML 语法错误、行内样式的文件加载错误
  - 3、浏览器的控制台 ==> 项目的逻辑代码错误

> CMD 错误 & 页面错误 & 浏览器控制台错误 ==>环境错误
>
> CMD错误 & 页面错误 ==> 模板错误
>
> CMD错误 & 浏览器控制台错误 ==> vue逻辑代码错误
>
> CMD错误 ==> 服务器无法启动 ==> 项目结构功能错误

- 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**
    - 对于参与vue单文件组件编译的标签图片地址，以**当前文件的位置作为图片路径定义参考**
  - 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 style标签样式的图片地址 **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>
```

4.2、动态样式

- **为vue项目的单文件组件增加动态语言支持：less (sass sacc) stylus**
  - 组件中的 `style` 标签默认识别css语法
  - 在组件定义时，可以使用 `<style lang="动态样式语言">` 描述当前样式使用何种语言
  - 单文件组件中，一个组件是可以定义多种样式语言特性
  - 项目中需要使用 动态语法，必须为项目安装 动态语法加载器
    - `less` **核心语法包，** `less-loader` **less的加载器** `npm install less less-loader -D（--save-dev）`
      - 生成环境模块：`--save` `-S`
      - 开发环境模块：`--save-dev` `-D`
- 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>
```

2. **通过框架配置可以实现,一次加载整个项目都可以使用**
   - 方式1：

> 添加全局动态样式变量加载器 **`style-resources-loader`**
>
> 上述插件依赖于 **`vue-cli-plugin-style-resources-loader`**
>
> `npm install style-resources-loader vue-cli-plugin-style-resources-loader -D`

> 手动添加 `vue.config.js` 文件 ==> 定义在项目目录下

- 方式2

  > 也可以直接使用命令`vue add style-resources-loader`，进行组合安装

> **自动在项目下生成vue.config.js，如果存在该文件，添加配置项**

3. **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
   - **当vue.config.js文本被修改后，必须重启服务器才可生效**

```
// 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
    }
  }
}
```

> assets用于存放静态资源的，和public目录功能相同
>
> -> 对于项目中使用图片，脚手架在编译组件时，会自动读取图片大小，当文件大小小于一个固定值，vue会自动转换为base64

5、模块化的开发规范

- 文件分层：（采用官方建议）组件分层，静态代码分层
  - 组件分层：页面组件，构成组件
    - 页面组件单独存放于一个独立文件夹 views pages
    - 构成组件 单独存放于一个独立文件夹 components
      - 也可以细分组件文件夹，例如：baseComponents ,viewsComponents.....
  - 静态分层：全局样式定义文件夹，统一存放于 assets 目录下
  - 插件分层：过滤器插件，指令插件，组件插件，.....
- 组件名称的定义
  - **组件名应该始终是多个单词的，根组件 App 除外。**
  - 单文件组件文件的大小写强烈推荐

    > **单文件组件的文件名应该要么始终是单词大写开头 (PascalCase)，要么始终是横线连接 (kebab-case)。**
- 构成方法时，方法名称应该以方法功能进行描述 **笔者要求**

6、项目打包

- `npm run serve` 启动开发服务器，提供前端工程师在项目开发时完成的一些特殊功能定义个执行
  - 开发过程中错误提示，语法错误，结构错误.....
  - 开发过程提供相关的调试功能

- 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**
    - **打包后会提供产品报表，记录了生成的文件信息，和大小**

- 打包后可将目录中的静态资源存放于任意服务器目录下进行运行，**但如果未进行开发和生产环境地址区分，会造成资源访问失败的情况**
- 通过配置和修改 `vue.config.js` 配置文件可以进行生产环境和开发环境调整

```
module.exports = {
    // publicPath:"/film/",
    // 根据环境区分 生成服务器的地址 和 开发服务器的地址
    // publicPath:process.env.NODE_ENV === "production"?"/film/":"/",
    // 以访问服务器地址的 index.html 存放的位置作为参考目录
    publicPath:"./",
    // outputDir:"filmProject",
    // 定义vue项目的页面入库
    indexPath:"index.html",
    // 改变开发服务器的特性
    devServer:{
        // 定义开发服务器端口的
        // host:"192.168.19.8",
        port:"8080"
    },
    // 打包时(生成环境下)关闭JS源代码调试功能
    productionSourceMap:false,
    // 开发环境下，是否开启css的调试功能。不会影响项目的打包
    css:{
        sourceMap:true
    },
    pluginOptions: {
        'style-resources-loader': {
            preProcessor: 'less',
            patterns: [
                "./src/less/var.less"
            ]
        }
    }
}
```

7、模块化项目的数据交互

- 模块化项目多数基于SPA+MVVM设计模式构成，因此数据交互被限制为AJAX技术上

> 1、原生AJAX技术 XMLHttpRequest
>
> 2、原生AJAX技术 fetch
>
> 3、jquery 中 ajax `ajax({ success(){},error(){} })`

- Vue模块化项目数据交互，同样基于异步请求模块完成
- 因为MVVM设计模式**不再需要额外的DOM操作功能，因此对于ajax请求开始面向于专门完 异步请求的插件**

- 可用于模块化的异步请求模块很多，官方推荐的使用 `axios` 模块
  - axios 插件是标准的基于 **ES6 promise** 对象封装的，能够**配合后端实现RESFULL请求**的 ajax 独立插件
    - 独立插件：任何场景任何页面均可使用的插件
    - RESFULL 请求：根据用户行为区分请求功能
      - GET == 描述当前请求完成的是数据查找
      - POST == 描述当前请求完成是数据添加
      - PUT == 描述当前请求完成原始数据的更新
      - DELETE == 描述当前请求完成 数据的 删除

```
例子
    后端提供一个接口
            http: //127.0.0.1:8080/product  ==>  该请求可以完成数据查找 添加 修改 删除
  后端根据请求类型，判断需要执行的方法
        get  =>  执行的是查询操作
        post =>  执行的添加操作
        put  =>  执行的是修改操作
        delelet  =>  执行删除操作
```

- **特性：不支持JSONP格式请求**

7.1、axios安装集成

- 通过npm 进行模块安装 `npm install axios --save` ，`npm install axios -S`

> - 判断模块提供的功能项目上线后是否需要使用
>
> 开发环境依赖包：--save-dev 或者 -D => 模块功能上线后不需要使用
>
> 生成环境依赖包：--save 或者 -S => 模块功能上线后需要继续使用

- 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

- `axios.get(url [,config])`
  - url-string:定义请求地址

- config-object:定义请求的额外配置
  - params:描述get请求的相关参数
- 方法调用后，会返回一个 Promise 对象对请求功能进行后续处理

```
// console.log(axios)
let pro = axios.get("http://127.0.0.1:8080/data/home.json");
// console.log(pro);
pro.then((response)=>{
    // 表述 axios 对象的 异步请求执行成功 （XMLHttpRequest 200 && 4）
    // axios 的请求成功后，会返回一个包装有后台数据的 完成响应对象
    //    response：地址，参数，模式，时间，状态，后台响应数据 ……
    console.log(response);
    if(response.statusText==="OK"&&response.data.resultState){
        console.log(response.data,this)
        this.list = response.data.result;
    }
});
pro.catch(function(){
    // 异步请求失败时所执行的方法
    //   ajax的请求成功和失败取决于 请求后台是否接收且存在返回结果
});
pro.finally(function(){
    // 无论请求成功还是失败都会执行的方法
});
```

2、post类型请求

- HTTP 参数协议中的三种方式

> 1、参数形式：Request Payload 表示以纯文本方式将参数定义到请求体中
>
> 提供何种格式的参数，就传递何种格式的参数
>
> 2、参数形式：Query String Parameters 表示参数地址后 以 ？和&符号进行定义
>
> 会根据 ？、&、= 作为参数格式化
>
> 3、参数形式：Form Data 表示将 以 & 和= 构成的字符串参数，定义到请求体中
>
> 会根据 &、= 作为参数格式化

```
// axios.post(url[, data[, config]])
//      url:请求地址
//      data：请求时 post 方式所携带的参数
//            + 默认定义的数据类型如果是 非 string 类型，请求将以 Request Payload
//              需要后台代码进行配合解析的
//            + 如果定义的数据类型为 string 类型，请求将以 Form Data
//              将 key：value 常规方式定义参数
//              string 参数必须使用 HTTP参数格式  key=value&key=value……
//      config:请求配置
//            => baseURL 为请求地址主动添加 请求前缀

// axios 可以对一个特殊配置项进行全局的默认配置定义
// axios.defaults.baseURL 完成整个项目的全局配置 = 定义在mian.js

// let baseUrl = process.env.NODE_ENV === "production"? "http://127.0.0.1:80" :
"/api";
// this.$axios.post("/filmApi/loadFilms.php",null,{
```

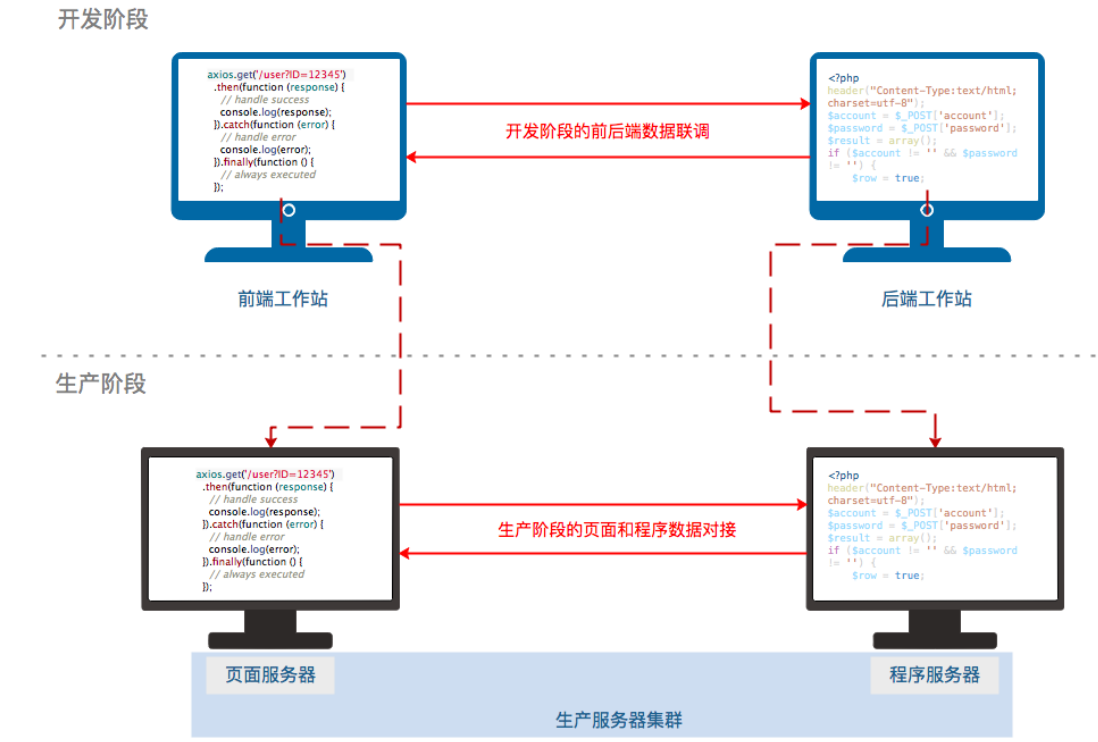
```
//     baseURL:process.env.NODE_ENV === "production"? "http://127.0.0.1:80" :
"/api"
// })
// this.$axios.post("/filmApi/loadFilms.php?aaa=aaa&ccc=ddd",{
//     size:12
// })
// this.$axios.post("/filmApi/loadFilms.php","size=12&aa=aa")
this.$axios.post("/filmApi/loadFilms.php",QS.stringify({
    size:12,
    aa:123
}))
.then(( { statusText,data } )=>{
    // if(statusText=="OK"&&data.resultState){
    // }else{
    //     return Promise.reject("请求失败");
    // }
    if(statusText!="OK"||!data.resultState){
        return Promise.reject("请求失败");
    }
    this.films = data.result;
}).catch(()=>{

})
```

7.3、跨域请求的分类和处理方式

1、线上跨域（生产环境跨域）

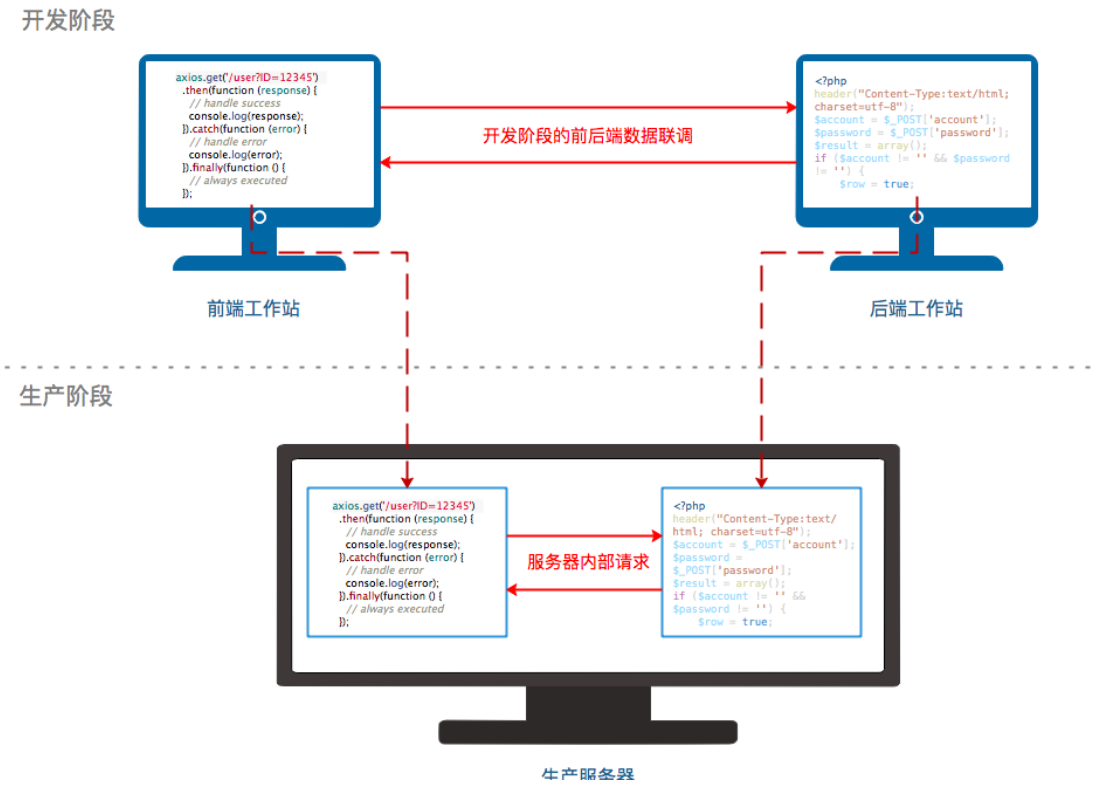
- 前后端代码已经部署商用：项目完成开发，已经交付使用，所有的代码运行与真实的商用服务器上
  - **后端处理：代码上增加跨域请求头 == Access-Control-Allow-Origin**
  - **前端后端配置处理： 采用 JSONP 模式，前端完成JSONP请求发送（callback）**
  - **nginx：代理服务器跨域**



2、开发跨域（开发环境跨域）

- **项目处于开发阶段：（完成前后端数据联调对接），但上线后跨域消失**

- 纯前端处理：由前端构建代理服务器完成跨域（代理服务器:请求转发器）

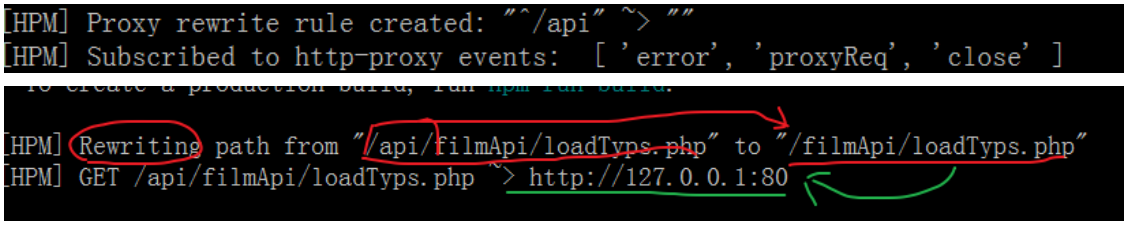


3、vue模块化项目开发跨域的代理服务器配置

- **通过修改 vue.config.js 文件完成环境配置，环境配置需要重启开发服务器生效配置**
- **代理服务器属于隐式代码执行，无法被浏览器进行监视和调测，可通过配置开启debug功能，在控制记录代理日志**
- **注意事项：代理服务器仅在开发环境下生效==>只用于开发跨域**
- vue.config.js 配置项中devServer中存在`proxy`开发服务器配置选项

```
devServer: {
    port: 8080,
    // 代理服务器配置
    proxy:{
        // key:value
        // key=描述项目中哪些请求需要被代理服务器代理
        //   以路径前缀方式描述key，定义被拦截请求的 前缀
        //   value 配置代理服务器的行为
        "/api":{
            // 拦截项目中所有以 /api 开头的请求
            target:"http://127.0.0.1:80", // 定义被拦截的请求需要访问的真实服务器
                // 地址转发时默认作为 请求地址和目标地址的拼接
                // 例：/api/filmApi/loadTyps.php
                //    转发到 http://127.0.0.1:80/filmApi/loadTyps.php
                // 通过前缀替换实现地址的重写和定义
                pathRewrite:{
                    // key:value  ==> 是JS的正则表达式使用配置 replace(匹配字符或者正则（key），替换后的值(value))
                    "^/api":""
                }
            }
        }
},
```

- 项目代理启动时，控制台显示下述信息，表示代理服务器开启成功



- 生产环境和开发环境切换时的地址动态切换
  - 通过vue提供环境监测 `process.env.NODE_ENV` 进行环境判断

```
let baseUrl = process.env.NODE_ENV==="production"?"生产环境地址":"开发环境地址";
```

7.4、AJAX请求的API模块化

- 请求方法的复用
- 请求地址的统一管理
- 请求数据的统一处理
- 请求的权限统一处理
- 所谓的功能模块化实际上就是**将功能进行JS文件单独定义**，然后以模块方式进行加载调用
- **高内聚，低耦合**

8、路由（vue-router）

- vue全家桶构成：vue + vue-router + vuex (**不在简历上写vue全家桶**)
- 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、自动化安装

- 在**创建项目**时 `vue create 项目名称` 通过选择路由功能自动添加
- 在**已经存在的项目中使用**，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
  - 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添加
  - **注意事项：添加方式只适合于未被修改的项目结构**

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 route.js 文件
  2. 在main.js 文件中导入并装载路由功能

```
// 1、加载路由语法构建模块 vue-router，Vue
import VueRouter from 'vue-router';
import Vue from 'vue';

/*
    vue-router 模块在完成装载和定义时，需要为整个项目提供路由功能
    在路由功能装载后，可以为vue提供一个特殊配置 router
```

```
    vue实例生命周期执行的 beforeCreate 时，路由会读取 router 配置

    router 就是 VueRouter 提供的 路由实例对象

    项目需要实现上述功能，必须让VueRouter.install 方法执行
*/

// 2、为项目安装路由 环境
Vue.use(VueRouter);

// 5、构成自定义路由对象，提供 路由配置
const config = {
    // 路由表模块
    // 定义组件和地址关系的配置文件
    routes:[]
}

// 3、创建VueRouter的实例对象
const router = new VueRouter(config);
export default router;
// 4、main.js 文件中进行路由对象导入
//    import router from './router.js';
//    在new Vue({ router }) 增加路由配置项
```

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",
  // 路由表数组
  // 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，**完成路由组件在项目中的位置定义**
2. `<router-link></router-link>` 用于切换和跳转组件，该组件功能类似于 HTML `<a href="">` 标签
    - 该组件是用于模拟地址变化的组件
    - 该组件需要定义一个必须属性 `to` ，定义当前组件被点击后所要完成 URL切换路径，取值为路由配置时组件唯一地址
    - 该组件本身不具有标签特性（默认会转换为 a 标签），可以模拟**任何已知的HTML标签**
    - 该组件可以通过定义 `tag` 属性，描述需要渲染为标签元素
    - 该组件可以包含任意 元素结构

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**
            - 绝对路径：服务器环境中 以 `/` 开头路径为绝对路径
                - 所有地址的方法必须从**端口后开始**
            - 相对路径：服务器中不以 `/` 开头的路径为相对路径
                - 在现有的地址上进行路径拼接访问
- 默认地址定义
    - 提供项目初次方式，默认组件的展示行为
    - **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}`

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
        - 实现方式：通过JS对history的方法访问，完成历史记录的读取，但该对象再访问历史记录时不会刷新页面，vue在此时进行历史功能的拦截，转为 vue 路由切换
        - **开发环境使用该模式不会产生任何负面效果**
        - 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
- 可以通过路由配置重写激活样式名
  - linkActiveClass:"class名称"
  - linkExactActiveClass:"class名称"

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
}
```

> 构成源码
>
> ```
> Object.defineProperty(Vue.prototype, '$route', {
>  get: function get () { return this._routerRoot._route }
> });
> ```

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**
- 原信息（元信息）：**在定义路由时为当前路由固定的信息数据**

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

> URL 统一资源定位符
>
> 统一资源定位符是统一资源标志符的一个下种。
>
> 统一资源标志符确定一个资源，而统一资源定位符不但确定一个资源，而且还表示出它在哪里。
>
> 一个完整的URL组成：
>
> protocol :// hostname[:port] / path / resource / [;parameters][?query]#anchor/auth
>
> protocol : 协议（HTTP、HTTPS、FTP、FILE ..... ）
>
> hostname：主机域名 或 主机IP地址
>
> port： 端口号 （HTTP协议默认端口 80 ）
>
> path：访问路径
>
> resource：访问的资源
>
> parameters：请求地址特殊参数
>
> query：查询字符串
>
> anchor：锚点（用于页面跳转的快速定位）
>
> auth：资源请求身份验证

- 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

```
// 只用于获取get方式的参数
vm.$route.query
vm.$route.query.参数名
```

2、rest方式

- Resfull 风格请求出现后，提出一种新的**基于URL地址栏** 参数传递方式
  1. 参数本身还是存在于 URL地址栏上
  2. 通过伪装URL地址，实现隐藏参数

- rest就是将参数伪装成请求地址的一部分进行URL的模拟操作
- **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**

```
请求地址  /detail
携带参数  filmId=234567
                 aa=1234

   get:   /detail?filmId=234567&aa=1234
   rest:  /detail/234567/1234  或 234567/detail/1234 或 1234/detail/234567  ……
```

- 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** `:` 描述地址中**那些部分为rest参数**
  - `:` 后所定义的名称，将成为该参数的变量名

```
{ path:"/detail/:filmId/:aa" }
```

  - rest风格参数定义后，可通过 `?` **描述该参数是否为可选参数**

```
{ path:"/detail/:filmId?/:aa?"},
```

- 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.$route.params
vm.$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（ **可以直接定义为 $router.push(opt)的参数** ）
    - 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 } } "**
      - **rest方式必须使用具名路由传递参数**
      - **此时路由切换将不受 path 属性的影响，只要保证 name 属性的值不变即可**
- **具名路由对于参数的影响**
  - 因具名路由不受path属性的影响，因此在切换时会导致路径的特殊变化

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**

```
Object.defineProperty(Vue.prototype, '$router', {
  get: function get () { return this._routerRoot._router }
});
```

- $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
  - $router.go(n) 模拟浏览器的 前进按钮
    - n > 0，表示从历史记录中向前前进多少个页面
    - n < 0，等效于$router.back()，可以通过n控制退回的层级
  - $router.back() 模拟浏览器的 后退按钮

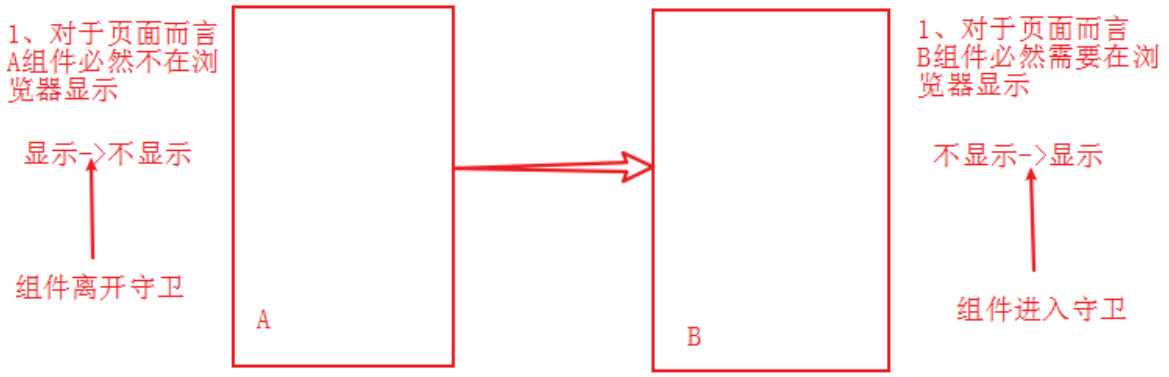
8.6、路由缓存

- 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

8.7、导航守卫

- 进行路由切换过程中，执行的相关**拦截**方法，可以路由的访问地址和数据进行读取操作（**路由生命周期**）
- 守卫的拦截，主要用于拦截页面切换请求，通过相关代码控制该组件能不能被显示



- 路由切换守卫（导航守卫）
  - 组件内守卫：和组件的生命周期的钩子函数使用方法一致
    - `beforeRouteEnter（to, from, next）` 组件由不显示到显示过程中执行的方法
      - 默认会阻止一切路由组件的切换功能
      - next 为一个方法，控制路由的执行行为
        - next() : 直接放行路由地址的访问
        - next(path) : 根据path的地址将路由切换到 指定组件中
      - to : 路由对象（路由地址、路由信息、参数、元数据），路由切换时目标路由信息对象
      - from : 路由对象（路由地址、路由信息、参数、元数据），路由切换时当前的路由信息对象
    - `beforeRouteUpdate` (2.2 新增)：在路由地址不变的情况化，监控参数的变化（**路由组件缓存导致的组件不重新发送请求的问题**）等效于组件中监控 $route 的功能
    - `beforeRouteLeave` 组件由显示到不显示过程中执行的方法，在组件离开时，情况内存中一些相关数据的
  - 全局守卫
    - 全局前置守卫 router.beforeEach：所有路由组件在进入路由前都必须执行的守卫方法
    - 全局后置守卫 router.afterEach((to,form)=>{})：所有路由组件在离开路由前都必须执行的守卫方法
      - 该方法只能用于记录路由访问数据，但无法拦截路由的请求

    > 循环+mixin混合

  - 独享守卫
    - beforeEnter ,和组件内守卫 beforeRouteEnter 一样
    - 独享守卫定义在全局路由配置，只针对于一个特点的组件

9、全局功能定义

- 用于统一定义和设置全局方法，全局属性，全局过滤器，全局指令，全局组件 ……
- 构成方法依赖于插件安装方法 **Vue.use()**，该方式将装载功能作为当模块安装至Vue全局功能上
  - `Vue.use()` 方法一般被定义在 main.js 文件中
  - `Vue.use(options)` options 为一个固定构建对象，**Vue.use实际是时读取提供参数的install属性进行执行的**

```
{
  install:function(Vue){

  }
}
```