# 项目需求

# 一: 任务

## 首页



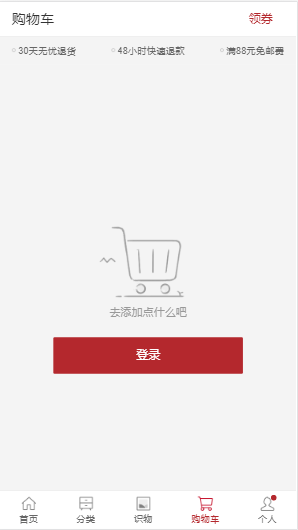
## 分类



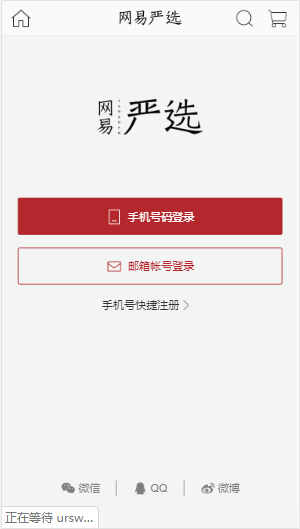
## 实物



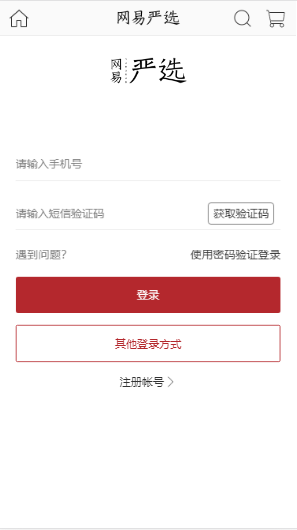
## 购物车



## 个人中心



## 登录



## 注册



# 二: 项目需求

## 2.1 底部导航组建封装

* 1. 一级路由搭建
  2. 路由传参（meta）

## 首页完成

1. 首页搜索框点击跳转到搜索页



b) 导航滑动效果实现

* + 1. 
    2. 导航列表宽度不能写死，个体宽度可写死，实际开发列表应该根据数据遍历生成
    3. 可使用flex布局解决问题
    4. 注意点击三角箭头显示全部频道
  1. 轮播图实现(swiper)
  2. 滑动效果实现(better-scroll)
     1. [](GIF.gif)
     2. 封装独立组件，实现复用
     3. 注意：new better-scroll的时候如果都使用一个class类名的情况下只有第一次new的实例会生效，可使用ref解决，或者动态传入组件数据配合动态class解决该问题
  3. 使用vuex管理状态数据，展示数据

## 分类页完成

### 2.3.1 布局（左右两侧）



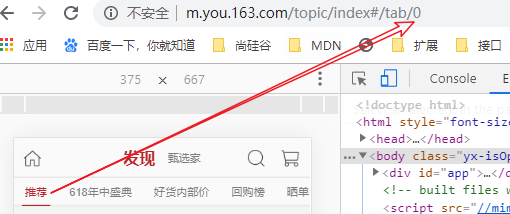
1. 左右布局可拆分组件(考虑组件通信)
2. 也可以不拆分组件，在一个组件中实现
3. 注意: 刷新页面的时候如何保证刷新之前的状态

## 实物页完成

### 发现 & 甑选家 路由设计

1. 可设计成以及路由（官网）
2. 也可以设计成二级路由(分类为以及，发现和甑选家分别为二级

### 发现页面导航切换



**思路: 可使用params或者query的形式实现**

### 发现推荐页面-组件封装

1. 仔细观察推荐页面显示数据方式有两种



1. 一种是图片是大图，一种是图片是小图，可在数据中找到显示大小图的标识

### 2.4.4 上拉加载功能实现

1. 推荐页面拉到底部后发送请求获取新的数据后继续展示在底部
2. 难点:
   1. 什么时候页面到底部
   2. 获取的数据怎么添加在原有数据基础上保证不重新加载整个页面
   3. 接口地址: 看接口文档

## 购物车页面完成

暂无实际交互功能

## 人中心页完成

### 页面切换

1. 手机号登录，邮箱登录及注册页面切换
2. 可使用路由跳转切换，也可以不使用路由通过控制显示隐藏切换
3. 注意: 完成前端验证，能够手写正则验证邮箱及手机号

# 三: 项目实战应巩固知识点

## 3.1 路由

### 3.1.1 路由跳转

1. 生成路由器 router
2. 注册路由 routes = {path:’路由路径’, components: ‘路由组件’}
3. 使用路由器
   1. new Vue({router})
4. 请求路由
5. <router-link :to="{name: 'home', params: {id: 1}}">home</router-link>

### 3.1.2 路由传参

**1. params:**



**2. query:**



**Params和query区别**

****

**3. meta:**



**4. props:**

**1) 布尔值模式: 解耦路由组件**

****

****

**2) 对象模式：props静态传参**

****

**3) 函数模式: props根据不同的路由对象动态传参**

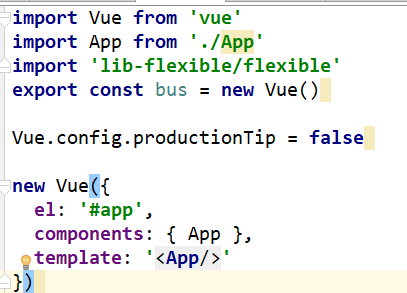
****

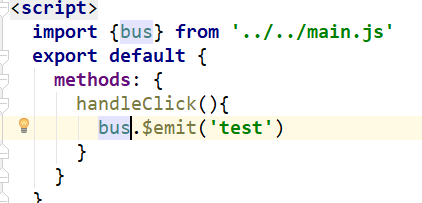
## 3.2 生命周期函数

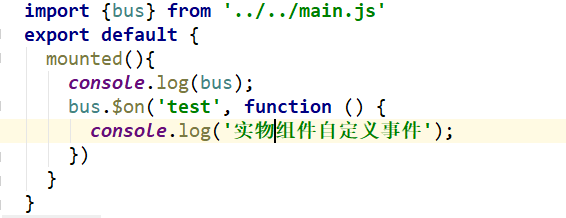
|  |
| --- |
| <**template**>  <**div**>  <**button @click="**handleClick**"**>更新状态按钮</**button**>  <**p**>{{msg}}</**p**>  <**button @click="**$emit(**'test'**)**"**>测试解绑自定义事件</**button**>  <**button @click="**$destroy()**"**>测试组件销毁</**button**>  </**div**> </**template**> |
| **export default** {  data(){  **return** {  msg: **'测试初始化数据'** }  },  methods:{  handleClick(){  console.log(**'点击事件'**);  **this**.msg = **'修改之后的数据'**;  }  },  beforeCreate(){  *// 实例初始化* console.log(**'------------- beforeCreate -------------'**);  console.log(**this**.$el); *// $el不可见* console.log(**this**.msg); *// 数据观测 (data observer) 和 event/watcher 事件配置没有完成* console.log(**this**.handleClick);  },  created(){  console.log(**'------------- created -------------'**);  console.log(**this**.$el); *// $el不可见* console.log(**this**.msg); *// 数据观测 (data observer) 和 event/watcher 事件配置完成* console.log(**this**.handleClick);  },  beforeMount(){  console.log(**'------------- beforeMount -------------'**);  console.log(**this**.$el); *// $el不可见* console.log(**this**.msg); *// 数据观测 (data observer) 和 event/watcher 事件配置完成* console.log(**this**.handleClick);  },  mounted(){  console.log(**'------------- mounted -------------'**);  console.log(**this**.$el); *// $el创建完成，替代页面的el元素* console.log(**this**.msg); *// 数据观测 (data observer) 和 event/watcher 事件配置完成* console.log(**this**.handleClick);   **this**.msg = 111;   **this**.$on(**'test'**, **function** () {  console.log(**'自定义事件'**);  })  },  beforeUpdate(){  console.log(**'------------- beforeUpdate -------------'**);  *// 数据更新时调用，发生在虚拟 DOM 打补丁之前。这里适合在更新之前访问现有的 DOM，比如手动移除已添加的事件监听器。* **this**.$off(**'test'**) *// 移除自定义事件* },  updated(){  console.log(**'------------- updated -------------'**);  *// 数据更新时调用，发生在虚拟 DOM 打补丁之前。这里适合在更新之前访问现有的 DOM，比如手动移除已添加的事件监听器。* },  beforeDestroy(){  console.log(**'------------- beforeDestory -------------'**);  *// 实例销毁之前调用。在这一步，实例仍然完全可用。通常在这关闭定时器* console.log(**this**.$el);  console.log(**this**.msg);  console.log(**this**.handleClick);  console.log(**this**);  },  destroyed(){  **debugger** console.log(**'------------- destoryed -------------'**);  *// 实例vm已经销毁，虽然可以可以通过this获取对应的数据，但不是当前组件的数据是当前钩子函数的context* console.log(**this**);  console.log(**this**.$el);  console.log(**this**.msg);  console.log(**this**.handleClick);   } } |

## 3.3 组件通信方式

1. **props**
   1. **多用于‘父传子’， 通过标签属性传递，在子组件通过props接收**
   2. **也可用于跨组件通信，逐层传递，但无形中增大的复杂度，容易出问题**
   3. **加分项: 通过props传值是一个给props对象赋值的过程，如果传递的数据中包含引用数据类型的时候会出现引用传递，即在子组件中修改数据会影响父组件的状态值**
2. **自定义函数传递**
   1. **多用于‘子传父’，在父组件中定义函数并声明形参，将函数通过props传递给子组件，在子组件中调用该函数并传入实参**
3. **插槽slot**
   1. **多用于‘向组件内部传递标签数据’**
   2. **场景: 封装组件用于复用，将组件内不同的标签部分提取出来用插槽传递提高组件的复用性**
   3. **分类: 普通插槽，具名插槽**
      1. **当组件内只有一个插槽的时候可使用普通插槽**
      2. **当组件内有多个插槽的时候需要根据插槽的名字使用具名插槽**
4. **$emit $on**
   1. **使用范围较广: ‘父传子’，‘子传父’，‘跨组件’通信**
   2. **加分项: 首先要注意当在不同的组件中使用$emit,$on的自定义事件机制是不能进行通信的，如果在不同的组件中使用$emit,$on需要绑定事件和触发事件的对象是同一个对象，即在vue的同一个‘事件总线**EventBus**’中**







1. **自定义消息机制(pubsub, eventbus)**
   1. **订阅消息: PubSub.subscribe(‘消息名称’, callback)**
   2. **发布消息: PubSub.publish(‘消息名称’, ‘发布数据’)**
2. **Vuex状态集中管理**
   1. **多用于中大型项目，组件嵌套层级较多，数据较为复杂的场景**
   2. **核心:** 
      1. **store： 状态管理对象**
      2. **mutations: 本质是函数，负责为store提供修改后的最新数据**
      3. **actions: 本质是函数， 负责调用mutations并为mutations提供参与修改状态的最新数据**
      4. **dispatch: 本质是函数，负责分发调用指定的action，并且为action提供数据**

# 四: 接口文档

## 4.1 关键字搜索接口

1. methods: GET
2. url: http://m.you.163.com/xhr/search/searchAutoComplete.json
3. data: keywordPrefix=value
4. 返回数据: json
5. 示例url: <http://m.you.163.com/xhr/search/searchAutoComplete.json?keywordPrefix=a>

## 4.2 实物发现页上拉加载接口

1. methods: GET
2. url: http://m.you.163.com/topic/v1/find/recAuto.json
3. data: page=value&size=value
   1. page代表页数
   2. size代表请求的数据个数
4. 返回数据: json
5. 示例url: <http://m.you.163.com/topic/v1/find/recAuto.json?page=1&size=5>

## 4.3 搜索页面默认显示keyword && hotkeyword接口

1. methods: GET || POST
2. url: http://m.you.163.com/xhr/search/init.json
3. data: 无
4. 返回数据: json
5. 示例url: <http://m.you.163.com/xhr/search/init.json>

# 五: 项目开发中常遇到的问题及解决方案

## 5.1 fastClick的300ms延迟解决方案

### 5.1.1 问题剖析:

**双击缩放(double tap to zoom)**，这也是会有上述 300 毫秒延迟的主要原因。双击缩放，即用手指在屏幕上快速点击两次，iOS 自带的 Safari 浏览器会将网页缩放至原始比例。

假定这么一个场景。用户在 iOS Safari 里边点击了一个链接。由于用户可以进行双击缩放或者单击跳转的操作，当用户一次点击屏幕之后，浏览器并不能立刻判断用户是确实要打开这个链接，还是想要进行双击操作。因此，iOS Safari 就等待 300 毫秒，以判断用户是否再次点击了屏幕。

### 5.1.2 解决方案

1. 引入fastClick库
2. 声明使用



## 5.2 UI库按需加载问题

### 5.2.1 问题剖析:

当开发的项目需要引入第三方的UI组件库的时候，需要引入其样式文件及js文件，但是对于我们的项目来说不需要整个UI组件库的样式，只需要我们使用的UI组件的样式，这个时候如果能按需加载需要的样式就会提高项目的加载运行效率，避免不必要代码的引入

### 5.2.2 解决方案:

## 5.3 外链样式页面刷新页面样式丢失问题

### 5.3.1 问题剖析:

项目通常需要一份重置样式文件reset.css, 而该文件通常放在static中通过link引入，

<link rel="stylesheet" href="./static/css/reset.css">， 但在子路由组件中刷新页面会发现重 置样式文件失效，原因是加载路劲错误

### 5.3.2 解决方案:

将相对路径改成绝对路径

## 5.4 scoped下修改第三方样式数据(深度选择器)

### 5.4.1 问题剖析:

在Vue组件化开发中为了避免组件之间样式冲突问题，可在style标签中加上scoped属性，等同于设置样式作用域，只在当前的组件生效，其原理就是给当前的选择器后添加一个随机生成的唯一字段（swiper-container[**data-v-7c66a6a2**]），但如果需要在有scoped的组件中修改外部组件的样式(想修改swiper库的样式),因为scoped的限制问题就无法修改

### 5.4.2 解决方案:

1. 使用深度选择器: 原生css样式--🡪 >>>, 在stylus，sass，less中使用 /deep/
2. 语法示例: 外层选择器 /deep/ 修改样式的选择器

|  |
| --- |
| <**style lang="stylus" rel="stylesheet/stylus" scoped**>  **#homeContainer** .**swiper-container  width** 500**px  height** 300**px  border** 1**px solid** .**swiper-wrapper  width** 100%  **height** 100%  .**swiper-slide  width** 100%  **height** 100%   /deep/ .**swiper-pagination-bullet-active  width** 20**px  height** 20**px  background red** </**style**> |

## 5.5 如何缓存路由组件

### 5.5.1 问题剖析:

1. Vue开发中使用路由跳转页面的时候，通常会注册多个路由，对应的有多个路由组件
2. 在Vue打包后文件非常之大，如果没有路由懒加载的话，一上来加载所有页面的文件
3. 如果同时加载所有页面的文件内容的话会导致首屏加载显示过慢，甚至白屏，导致用户体验差

### 5.5.2 解决方案:

Table 没有使用懒加载的方式

|  |
| --- |
| import Category from '../pages/CategoryList/Category/Category.vue';  import RecoFind from '../pages/Recommend/RecoFind/RecoFind.vue';  import RecoSelect from '../pages/Recommend/RecoSelect/RecoSelect.vue';  import Login from '../pages/Profile/Login/Login.vue';  import Register from '../pages/Profile/Register/Register.vue'; |

Table 使用懒加载的方式

|  |
| --- |
| const Home = () => import('../pages/Home/Home.vue');  const Search = () => import('../pages/Search/Search.vue');  const CategoryList = () => import('../pages/CategoryList/CategoryList.vue');  const Recommend = () => import('../pages/Recommend/Recommend.vue');  const ShopCart = () => import('../pages/ShopCart/ShopCart.vue');  const Profile = () => import('../pages/Profile/Profile.vue'); |

Table 公共代码

|  |
| --- |
| export default [  {  path: '/home',  component: Home,  meta: {  isShow: true  }  },  {  path: '/search',  component: Search  },  {  path: '/categorylist',  component: CategoryList,  meta: {  isShow: true  },  children: [  {  path: '/categorylist/category',  component: Category,  meta: {  isShow: true  }  },  {  path: '',  redirect: '/categorylist/category'  }  ]  },  {  path: '/recommend',  component: Recommend,  meta: {  isShow: true  },  children: [  {  path: '/recommend/find',  component: RecoFind,  meta: {  isShow: true  }  },  {  path: '/recommend/select',  component: RecoSelect,  meta: {  isShow: true  }  },  {  path: '',  redirect: '/recommend/find'  }  ]  },  {  path: '/shopcart',  component: ShopCart,  meta: {  isShow: true  }  },  {  path: '/profile',  component: Profile,  children: [  {  path: '/profile/login',  component: Login  },  {  path: '/profile/register',  component: Register  },  {  path: '',  redirect: '/profile/login'  }  ]  },  {  path: '/',  redirect: '/home'  }  ] |

## 5.6 params传参刷新参数丢失问题

### 5.6.1 问题剖析:

1. Vue-router进行路由跳转的时候可通过params传参
2. 如果在注册路由的时候没有使用占位符进行注册: ‘/home/:id’,首次路由跳转可以获取params参数，再次刷新页面params数据丢失

### 5.6.2 解决方案:

**注册路由的时候写好占位符**

|  |
| --- |
| **export default** [  {  **path**: **'/home/:id'**,  **component**: Home,  **name**: **'home'**,  **meta**: {  **isShow**: **true** }  } ]  <**router-link :to="**{name: **'home'**, params: {id: 1}}**"**>home</**router-link**> |

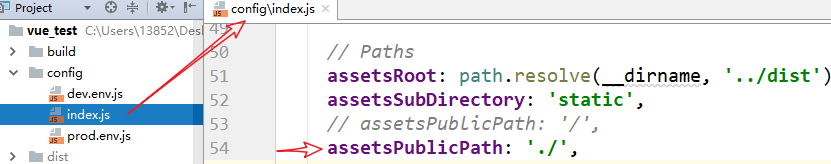
## 5.7 打包后访问页面404问题

### 5.7.1 问题剖析:

1. 打包后生成的dist文件交给后台以后通常会部署到服务器项目的静态资源根目录下
2. 如果没有部署到服务器项目静态资源的根目录下访问就是404,

### 5.7.2 解决方案:

1. 打包之前修改打包的公共资源路径: assetsPublicPath: './'
2. 将绝对路径该成相对路径
3. 但大多情况不需要修改



## 5.8 .map文件过大问题

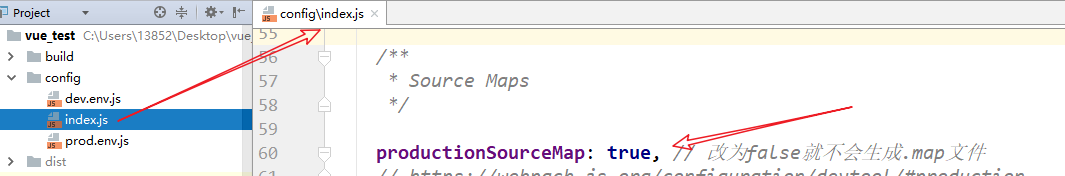
### 5.8.1 问题剖析:

1. 打包生成的dist文件中的文件是压缩文件，如果发生错误不能很好的提示是哪里出了问题
2. .map文件就是为了提示用户哪里出了问题



### 5.8.2 解决方案:

1. 实际项目上线文件是不需要.map文件的，如果没有去掉无形导致项目体积增大
2. 设置修改: config/index ---🡪 productionSourceMap: false



## 5.9 生产环境 & 开发环境解决跨域问题

### 5.9.1 问题剖析:

1. 跨域: 浏览器为了安全起见推出了同源策略（协议，域名，端口号三者完全一样就是同源，否则就是跨域）
2. 因为现在开发的项目都是前后端完全分离(前端有自己的服务器，如：vue脚手架服务器，同时还有服务器端服务器)的项目，必然存在跨域问题

### 5.9.2 解决方案之开发环境跨域:

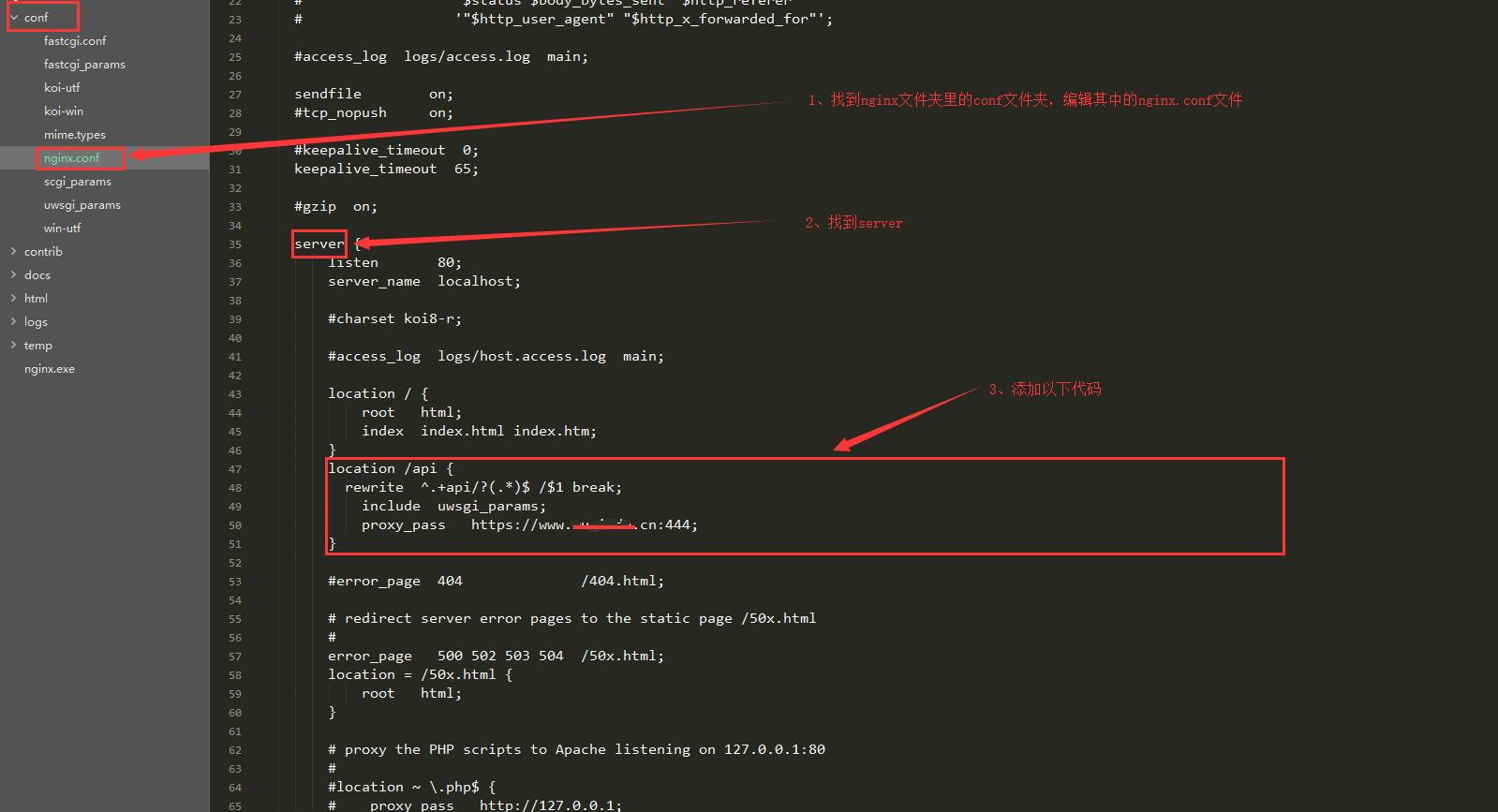
1. 配置正向代理
2. 思路: 页面请求本地服务器，本地服务器通过配置手动转换请求的服务器地址，请求服务器： /api/home --🡪 http://m.you.163.com/home

Table config/index.js文件中配置

|  |
| --- |
| **proxyTable**: {  **'/api'**: { *// 匹配所有以 '/api'开头的请求路径* **target**: **'http://m.you.163.com'**, *// 代理目标的基础路径* **changeOrigin**: **true**, *// 支持跨域* **pathRewrite**: {*// 重写路径: 去掉路径中开头的'/api'* **'^/api'**: **''** }  } }, |

### 5.9.3 解决方案之生产环境跨域:

1. 后台使用CORS解决跨域，存在风险，容易被人恶意攻击
2. 使用nginx反向代理
3. 思路: 页面请求发送给nginx服务器，nginx根据配置动态匹配要请求的服务器地址，最终的请求由nginx服务器发出解决跨域



|  |
| --- |
| location /api {  add\_header **'Access-Control-Allow-Origin' '\*'**;  proxy\_pass http:*//fanyi.baidu.com/v2transapi;* } |

## 5.10 使用swiper & better-scroll问题

### 5.3.1 问题剖析:

1. 使用swiper或者better-scroll的时候需要实例化，一下以swiper为例说明，在new Swiper的时候通常传递一个class类名，其目的是为了让swiper找到页面中对应的DOM对象从而在该DOM对象上产生作用
2. 当一个页面中同时new多个swiper或者better-scroll的时候，使用的class类名一样的话则最先new的那个swiper生效，其他的无效
3. 场景: 当封住一个swiper或者better-scroll组件的时候在一个页面复用，最先使用的那个swiper组件能够使用，其他的失效

### 5.3.2 解决方案:

**将class换成ref**