

# Vue 核心技术与实战

## 智慧商城项目

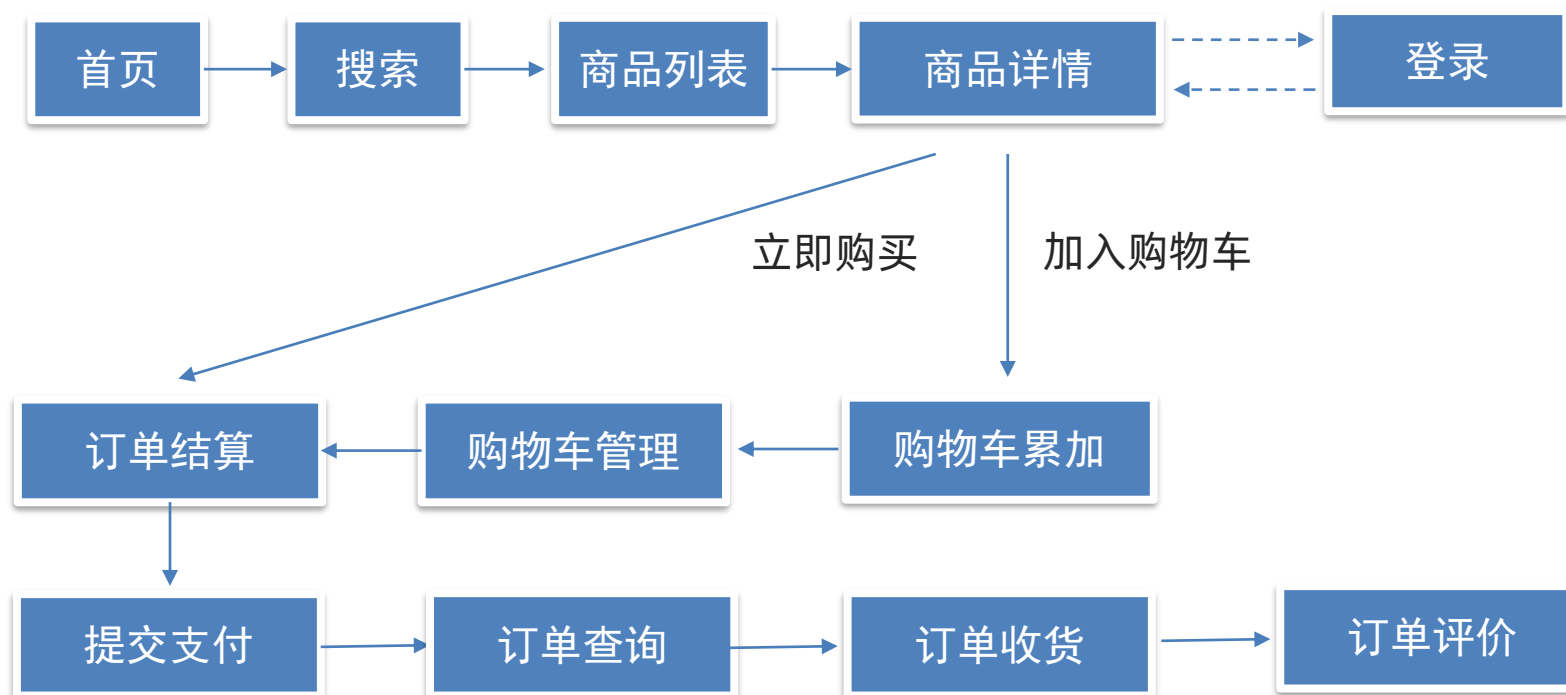


黑马程序员  
[www.itheima.com](http://www.itheima.com)

传智教育旗下  
高端IT教育品牌

## 项目演示

目标：查看项目效果，明确功能模块 → 完整的电商购物流程



## 项目收获

目标：明确做完本项目，能够收获哪些内容

完整电商购物的业务流

组件库vant (全部&按需导入)

移动端vw适配

request请求方法封装

storage存储模块封装

api请求模块封装

请求响应拦截器

嵌套路由配置

路由导航守卫

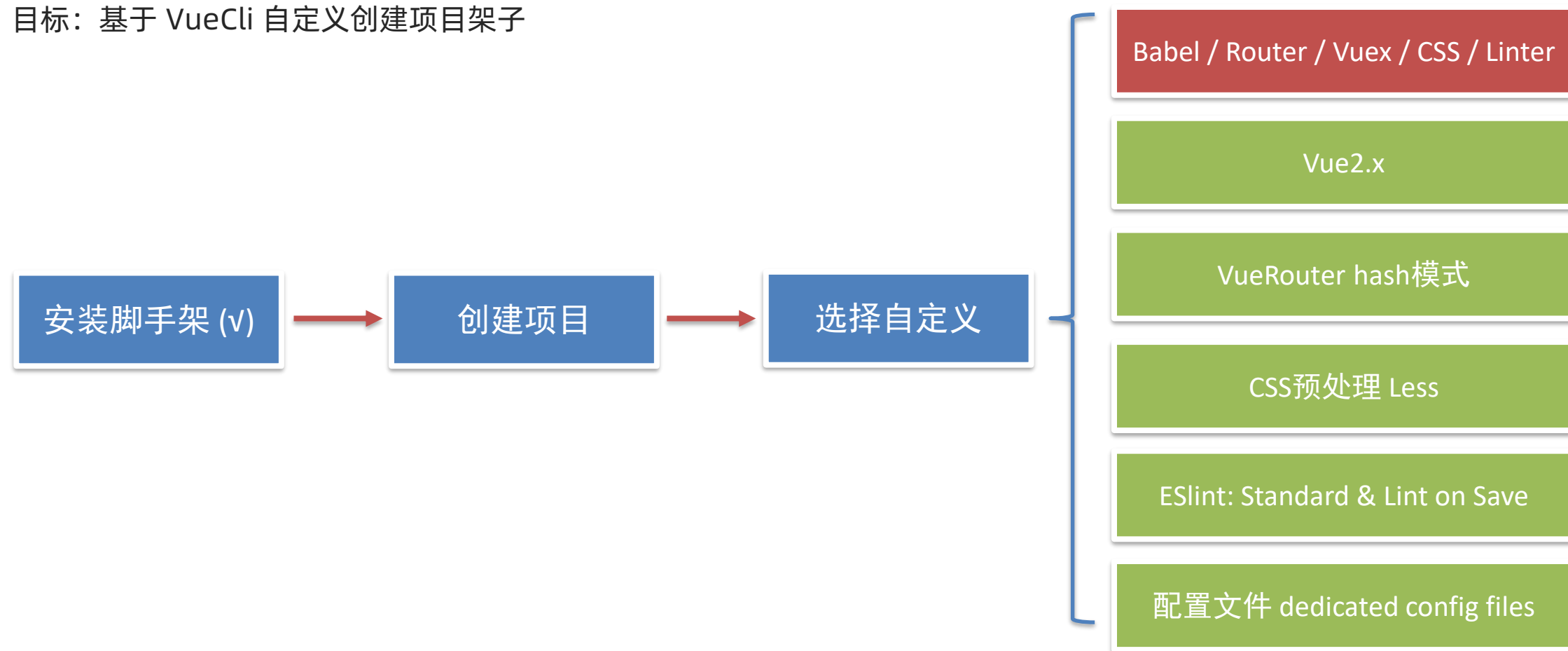
路由跳转传参

vuex分模块管理数据

项目打包&优化

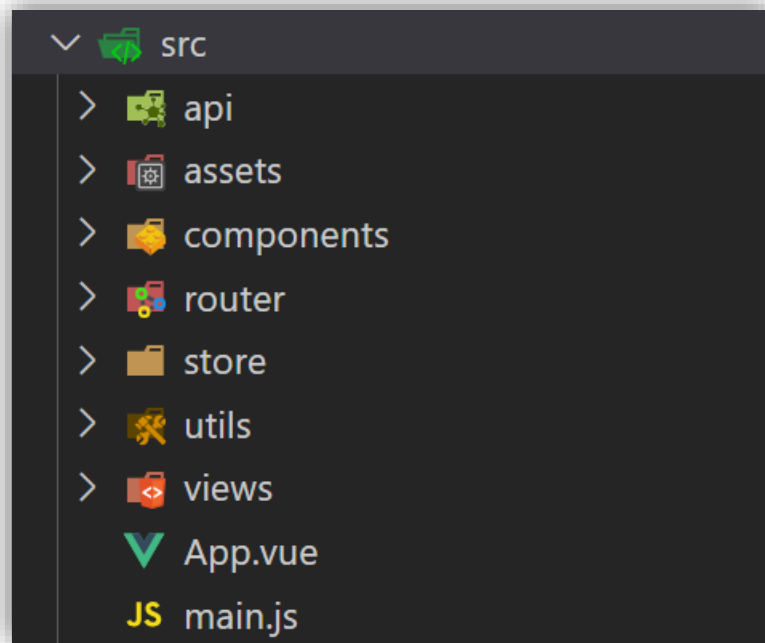
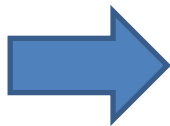
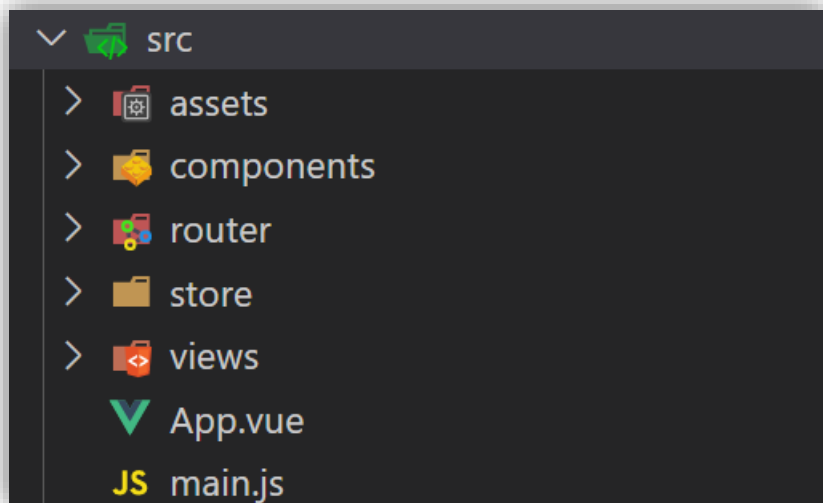
## 创建项目

目标：基于 VueCli 自定义创建项目架子



## 调整初始化目录

目标：将目录调整成符合企业规范的目录



1. 删除 多余的文件
2. 修改 路由配置 和 App.vue
3. 新增 两个目录 api / utils
  - ① api 接口模块：发送ajax请求的接口模块
  - ② utils 工具模块：自己封装的一些工具方法模块

## vant 组件库

目标：认识第三方 Vue组件库 vant-ui

组件库：第三方 **封装** 好了很多很多的 **组件**，整合到一起就是一个组件库。

<https://vant-contrib.gitee.io/vant/v2/#/zh-CN/>



## 其他 Vue 组件库

### 目标：了解其他 Vue 组件库

Vue的组件库并不是唯一的，vant-ui 也仅仅只是组件库的一种。

一般会按照不同平台进行分类：

- ① PC端：[element-ui](#) ([element-plus](#))    [ant-design-vue](#)
- ② 移动端：[vant-ui](#)    [Mint UI \(饿了么\)](#)    [Cube UI \(滴滴\)](#)

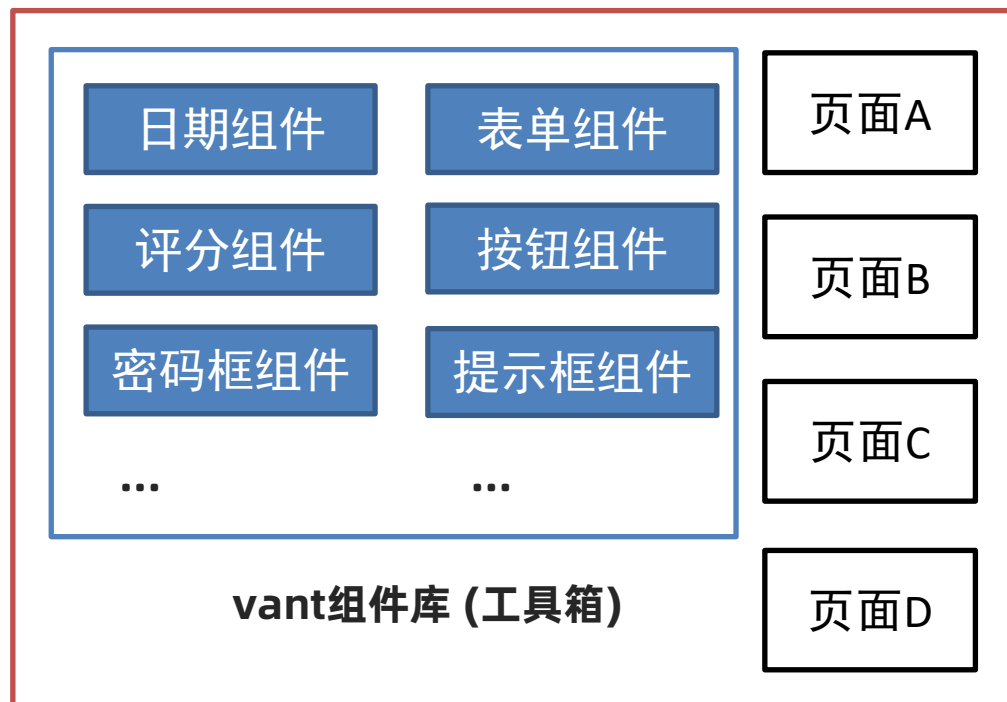
#### 维护状态

目前 Vant 各个版本的维护状态如下：

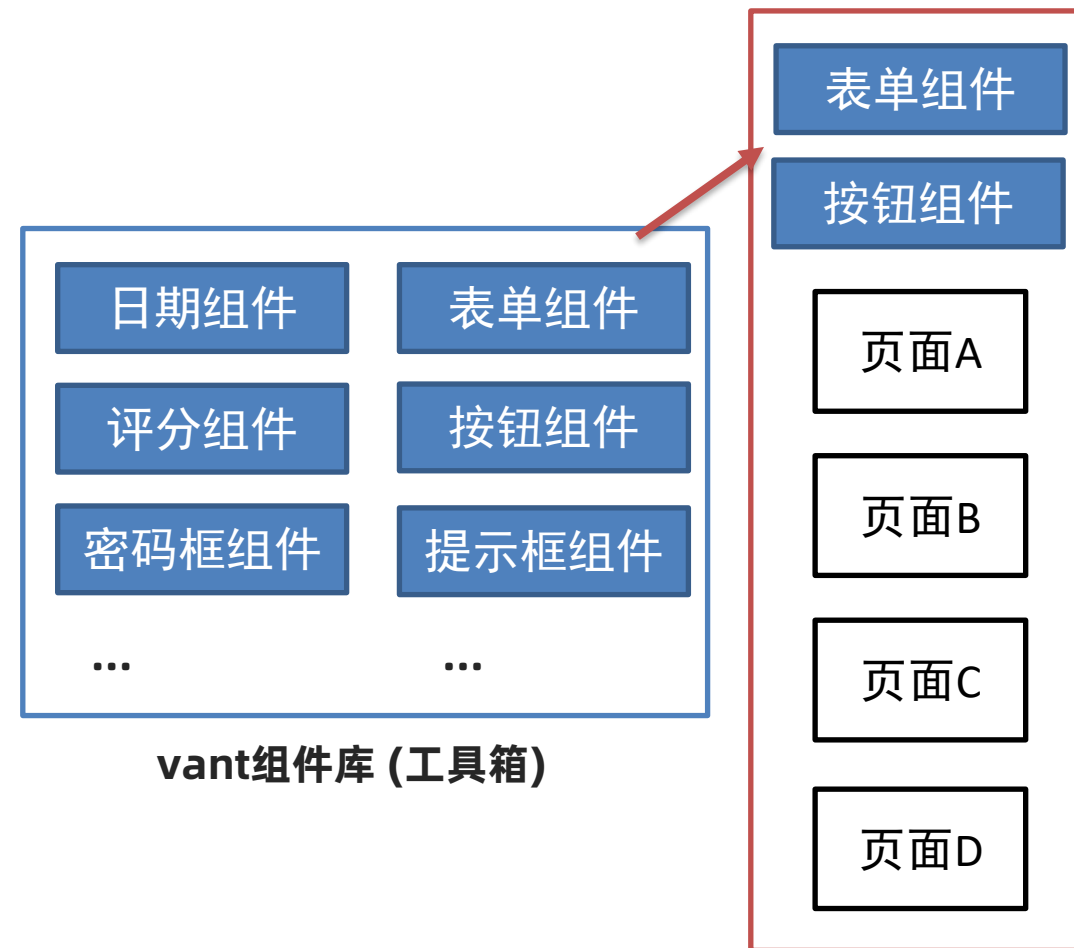
名称	框架	发布时间	最新版	维护状态
Vant 4	Vue 3	2022. 12	<code>npm@latest</code> <code>v4.1.2</code>	持续迭代新功能
Vant 3	Vue 3	2020. 12	<code>npm@latest-v3</code> <code>v3.6.11</code>	停止迭代新功能，bug 会被处理和修复
Vant 2	Vue 2	2019. 06	<code>npm@latest-v2</code> <code>v2.12.54</code>	停止迭代新功能，重要 bug 会被处理和修复
Vant 1	Vue 2	2018. 03	<code>npm@latest-v1</code> <code>v1.6.28</code>	停止维护，不再接受 PR

## vant 全部导入 和 按需导入

目标：明确 全部导入 和 按需导入 的区别



项目A - 全部导入 (方便)



项目B - 按需导入(性能)

【推荐】



## vant 全部导入 和 按需导入

目标：阅读文档，掌握 **全部导入** 的基本使用

官网：[vant-ui](https://vant-ui.github.io/)

全部导入：

① 安装 vant-ui

```
yarn add vant@latest-v2
```

② main.js 中注册

```
import Vant from 'vant'  
import 'vant/lib/index.css'  
// 把vant中所有的组件都导入了  
Vue.use(Vant)
```

主要按钮

信息按钮

③ 使用测试

```
<van-button type="primary">主要按钮</van-button>  
<van-button type="info">信息按钮</van-button>
```

## vant 全部导入 和 按需导入

目标：阅读文档，掌握 **按需导入** 的基本使用

按需导入：

### ① 安装 vant-ui (已安装)

```
yarn add vant@latest-v2
```

### ② 安装插件

```
npm i babel-plugin-import -D
```

### ③ babel.config.js 中配置

```
module.exports = {
  presets: [
    '@vue/cli-plugin-babel/preset'
  ],
  plugins: [
    ['import', {
      libraryName: 'vant',
      libraryDirectory: 'es',
      style: true
    }, 'vant']
  ]
}
```

### ④ main.js 按需导入注册

```
import Vue from 'vue';
import { Button } from 'vant';

Vue.use(Button);
```

### ⑤ 测试使用

```
<van-button type="primary">主要按钮</van-button>
<van-button type="info">信息按钮</van-button>
```

### ⑥ 提取到 vant-ui.js 中，main.js 导入

```
// 导入按需导入的配置文件
import '@/utils/vant-ui'
```

## 项目中的 vw 适配

目标：基于 postcss 插件 实现项目 vw 适配

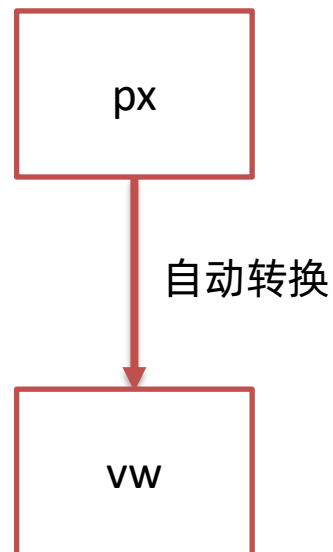
[官方配置](#)

### ① 安装插件

```
yarn add postcss-px-to-viewport@1.1.1 -D
```

### ② 根目录新建 postcss.config.js 文件，填入配置

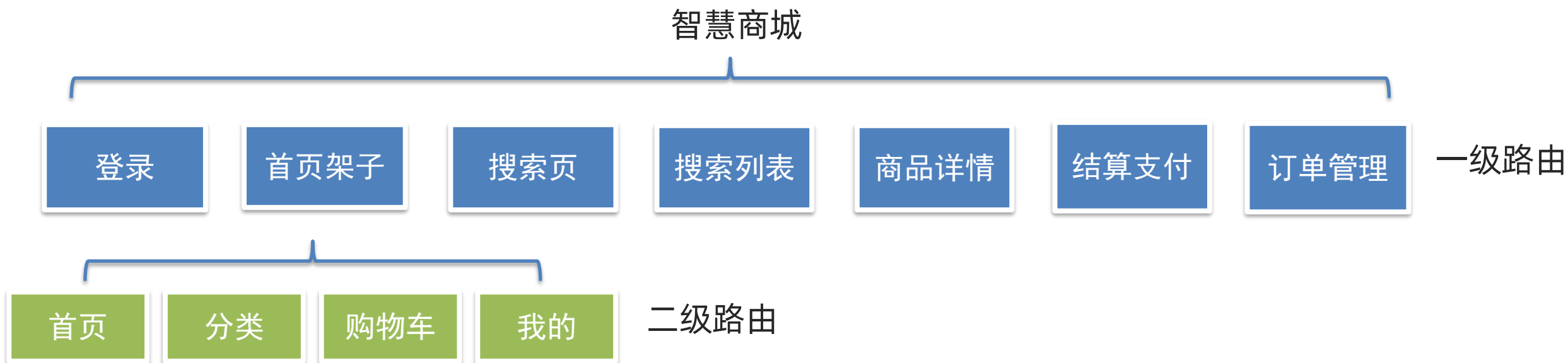
```
// postcss.config.js
module.exports = {
  plugins: {
    'postcss-px-to-viewport': {
      // 标准屏宽度
      viewportWidth: 375
    }
  }
}
```



## 路由设计配置

目标：分析项目页面，设计路由，配置一级路由

但凡是单个页面，独立展示的，都是一级路由



## 路由设计配置

目标：阅读vant组件库文档，实现**底部导航 tabbar**

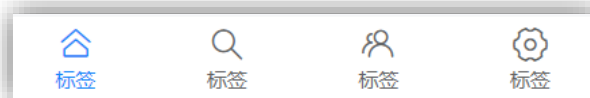


tabbar标签页：

① vant-ui.js 按需引入

```
import { Tabbar, TabbarItem } from 'vant'
Vue.use(Tabbar)
Vue.use(TabbarItem)
```

② layout.vue 粘贴官方代码测试



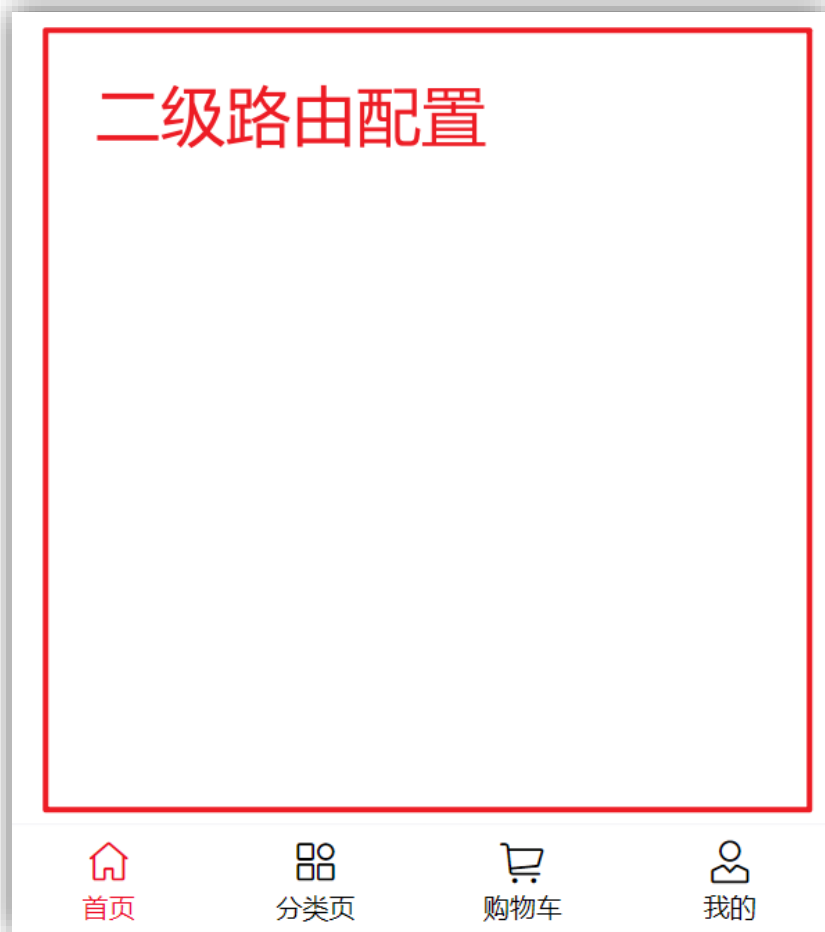
```
<van-tabbar>
  <van-tabbar-item icon="home-o">标签</van-tabbar-item>
  <van-tabbar-item icon="search">标签</van-tabbar-item>
  <van-tabbar-item icon="friends-o">标签</van-tabbar-item>
  <van-tabbar-item icon="setting-o">标签</van-tabbar-item>
</van-tabbar>
```

③ 修改文字、图标、颜色

```
<van-tabbar active-color="#ee0a24" inactive-color="#000">
  <van-tabbar-item icon="wap-home-o">首页</...>
  <van-tabbar-item icon="apps-o">分类页</...>
  <van-tabbar-item icon="shopping-cart-o">购物车</...>
  <van-tabbar-item icon="user-o">我的</...>
</van-tabbar>
```

## 路由设计配置

目标：基于底部导航，完成**二级路由配置**



配置二级路由  
(规则 & 组件)

children

配置导航链接

```
<van-tabbar route>
  <van-tabbar-item to="/home" ...>
    首页
  </van-tabbar-item>
  ...
</van-tabbar>
```

配置路由出口

```
<div class="layout-page">
  <router-view></router-view>
  ... (导航部分)
</div>
```

## 登录页静态布局

目标：基于笔记，快速实现登录页静态布局

### 1. 准备工作

- (1) 新建 `styles/common.less` 重置默认样式
- (2) main.js 导入 common.less
- (3) 图片素材拷贝到 assets 目录【备用】

### 2. 登录页静态布局编写

- (1) 头部组件说明 (NavBar)
- (2) 通用样式覆盖
- (3) 其他静态结构编写



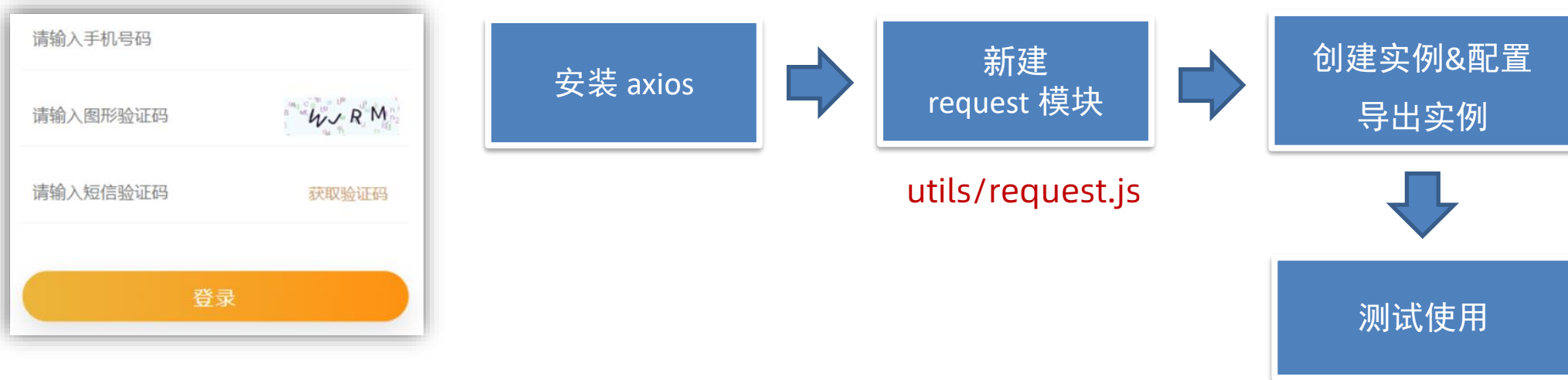
The image shows a mobile application login screen. At the top, there is a navigation bar with a back arrow on the left and the text '会员登录' (Member Login) on the right. Below the navigation bar, the main heading is '手机号登录' (Login with Mobile Number). Underneath the heading, a subtitle reads '未注册的手机号登录后将自动注册' (After logging in with an unregistered mobile number, it will automatically register). There are three input fields: '请输入手机号码' (Please enter mobile number), '请输入图形验证码' (Please enter graphical verification code), and '请输入短信验证码' (Please enter SMS verification code). To the right of the graphical verification code input is a small image of a verification code. To the right of the SMS verification code input is a button labeled '获取验证码' (Get verification code). At the bottom of the form is a large orange button labeled '登录' (Login).

## request模块 - axios 封装

**目标：将 axios 请求方法，封装到 request 模块**

使用 axios 来请求后端接口，一般都会对 axios 进行一些配置（比如：配置基础地址，请求响应拦截器等）

所以项目开发中，都会对 axios 进行基本的二次封装，单独封装到一个 request 模块中，便于维护使用



接口文档地址：

<https://apifox.com/apidoc/shared-12ab6b18-adc2-444c-ad11-0e60f5693f66/doc-2221080>

基地址：

<http://cba.itlike.com/public/index.php?s=/api/>



## 图形验证码功能完成

目标：基于请求回来的 base64 图片，实现图形验证码功能

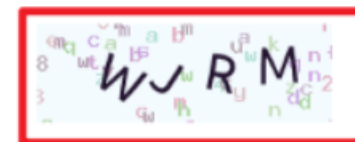
说明：

1. 图形验证码，本质就是一个请求回来的图片
2. 用户将来输入图形验证码，用于强制人机交互，可以抵御机器自动化攻击 (例如：避免批量请求获取短信)

需求：

1. 动态将请求回来的 base64 图片，解析渲染出来
2. 点击验证码图片盒子，要刷新验证码

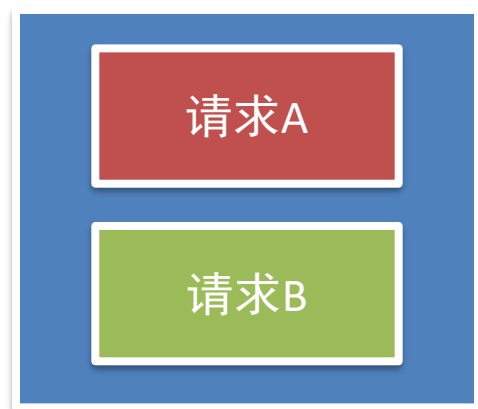
请输入图形验证码



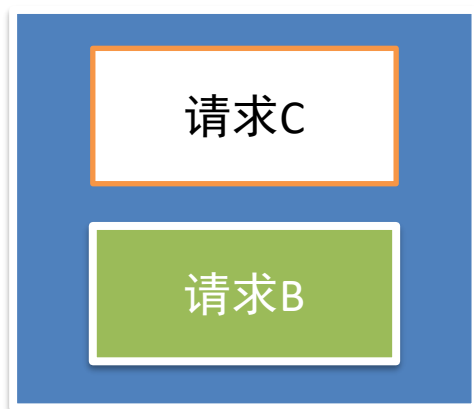
```
// 获取图形验证码
async getPicCode () {
  const { data: { base64, key } } = await getPicCode()
  this.picUrl = base64 设置给 img src
  this.picKey = key 图片的唯一标识 (将来验证需要携带)
},
```

## api 接口模块 - 封装图片验证码接口

目标：将请求封装成方法，统一存放到 **api 模块**，与页面分离



页面1



页面2



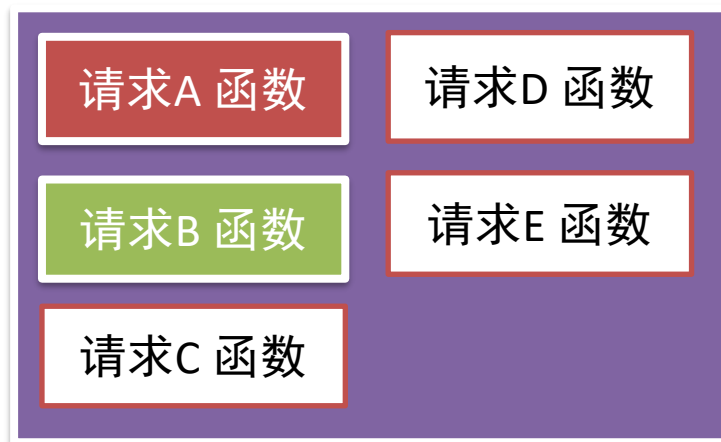
页面3

以前的模式：

1. 页面中充斥着请求代码，可读性不高
2. 相同的请求没有复用
3. 请求没有统一管理

## api 接口模块 - 封装图片验证码接口

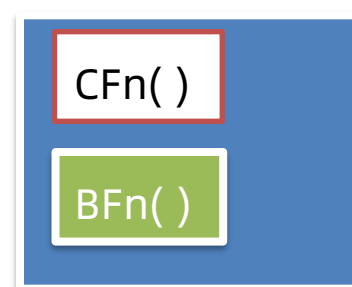
目标：将请求封装成方法，统一存放到 **api 模块**，与页面分离



API模块 (存放封装好的请求函数)



页面1



页面2



页面3

封装api模块的好处:

1. 请求与页面逻辑分离
2. 相同的请求可以直接复用
3. 请求进行了统一管理



## Toast 轻提示

目标：阅读文档，掌握 toast 轻提示

注册安装：

```
import { Toast } from 'vant'
Vue.use(Toast)
```

两种使用方式

① 导入调用（组件内 或 非组件中均可）

```
import { Toast } from 'vant'
Toast('提示内容')
```

② 通过this直接调用（必须组件内）

本质：将方法，注册挂载到了Vue原型上 `Vue.prototype.$toast = xxx`

```
this.$toast('提示内容')
```



请输入手机号码

请输入图形验证码

请输入正确的手机号

请输入短信验证码

15751776629

l6aj

发送成功，请注意查收

登录

请输入短信验证码

59秒后重新发送

登录

## 短信验证倒计时

目标：实现短信验证倒计时功能

请输入短信验证码

重新发送(55)秒

步骤分析：

1. 点击按钮，实现 **倒计时** 效果
2. 倒计时之前的 **校验处理** (手机号、验证码)
3. 封装**短信验证请求接口**，发送请求添加提示

```
if (!this.validFn()) {  
    return ② 请求前的校验  
}  
  
if (!this.timer && this.second === 60) {  
    // 发送请求，获取验证码  
    await getMsgCode(this.picCode, this.picKey, this.mobile)  
    this.$toast('发送成功，请注意查收') ③ 封装接口，调用发送请求  
  
    // 开启倒计时  
    this.timer = setInterval(() => {  
        this.second--  
  
        ① 开启倒计时  
  
        if (this.second < 1) {  
            clearInterval(this.timer)  
            this.timer = null  
            this.second = 60  
        }  
    }, 1000)  
}
```

## 登录功能

目标：封装api登录接口，实现登录功能

步骤分析：

1. 阅读接口文档，**封装登录接口**
2. 登录前的校验 (**手机号**，**图形验证码**，**短信验证码**)
3. **调用方法**，发送请求，成功添加提示并跳转



A login form with three input fields: '请输入手机号码' (Please enter mobile number), '请输入图形验证码' (Please enter graphical verification code), and '请输入短信验证码' (Please enter SMS verification code). To the right of the second field is a small image of a graphical verification code. To the right of the third field is a button labeled '获取验证码' (Get verification code). At the bottom is a large orange button labeled '登录' (Login).

```
// 验证码登录
export const codeLogin = (mobile, smsCode) => {
  return request.post('/passport/login', {
    form: {
      isParty: false,
      mobile,
      partyData: {},
      smsCode
    }
  })
}
```

```
async login () {
  if (!this.validFn()) {
    return
  }
  if (!/^\\d{6}$/.test(this.msgCode)) {
    this.$toast('请输入正确的手机验证码')
    return
  }
  await codeLogin(this.mobile, this.msgCode)
  this.$router.push('/')
  this.$toast('登录成功')
}
```

## 响应拦截器统一处理错误提示

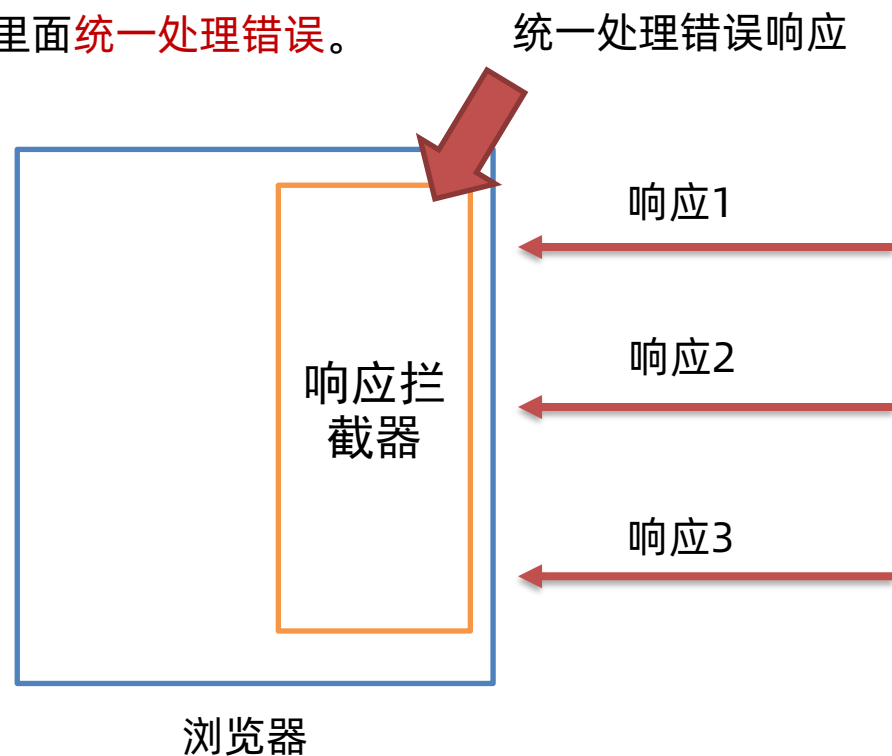
**目标：**通过响应拦截器，统一处理接口的错误提示

**问题：**每次请求，都会有可能会错误，就都需要错误提示

**说明：**响应拦截器是咱们拿到数据的第一个数据流转站，可以在里面**统一处理错误**。

只要不是 200，就给默认提示，抛出错误

```
// 添加响应拦截器
request.interceptors.response.use(function (response) {
  const res = response.data
  if (res.status !== 200) {
    Toast(res.message)
    return Promise.reject(res.message)
  }
  // 对响应数据做点什么
  return res
}, function (error) {
  // 对响应错误做点什么
  return Promise.reject(error)
})
```



## 登录权证信息存储

目标：vuex 构建 user 模块存储登录权证 (token & userId)

补充说明：

1. token 存入 vuex 的好处，易获取，响应式
2. vuex 需要分模块 => user 模块

```
▼ data: {userId: 10034, token: "c1c079695f414a71b9903444e882259c"}  
  token: "c1c079695f414a71b9903444e882259c"  
  userId: 10034  
message: "登录成功"  
status: 200
```





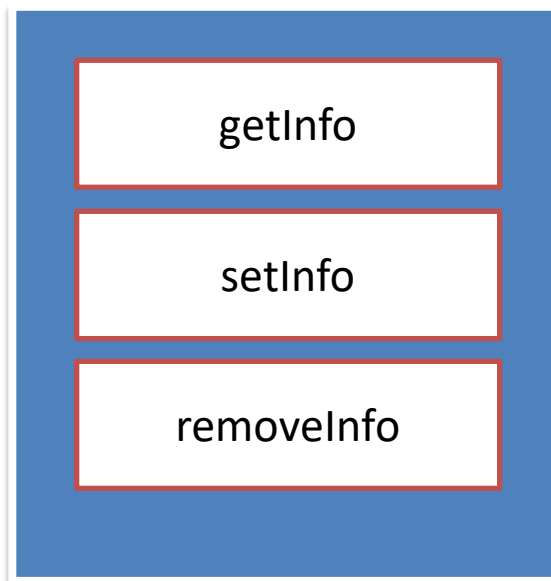
## storage存储模块 - vuex 持久化处理

目标：封装 storage 存储模块，利用本地存储，进行 vuex 持久化处理

问题1：vuex 刷新会丢失，怎么办？

```
// 将token存入本地
localStorage.setItem('hm_shopping_info', JSON.stringify(xxx))
```

问题2：每次存取操作太长，太麻烦？



storage模块

```
const INFO_KEY = 'hm_shopping_info'
// 获取个人信息
export const getInfo = () => {
  const result = localStorage.getItem(INFO_KEY)
  return result ? JSON.parse(result) : { token: '', userId: '' }
}
// 设置个人信息
export const setInfo = (info) => {
  localStorage.setItem(INFO_KEY, JSON.stringify(info))
}
// 移除个人信息
export const removeInfo = () => {
  localStorage.removeItem(INFO_KEY)
}
```

## 添加请求 loading 效果

**目标：**统一在每次请求后台时，添加 loading 效果

**背景：**有时候因为网络原因，一次请求的结果可能需要一段时间后才能回来，此时，需要给用户 **添加 loading 提示**。

添加 loading 提示的好处：

1. 节流处理：防止用户在一次请求还没回来之前，多次进行点击，发送无效请求
2. 友好提示：告知用户，目前是在加载中，请耐心等待，用户体验会更好

实操步骤：

1. **请求拦截器**中，每次请求，**打开 loading**
2. **响应拦截器**中，每次响应，**关闭 loading**

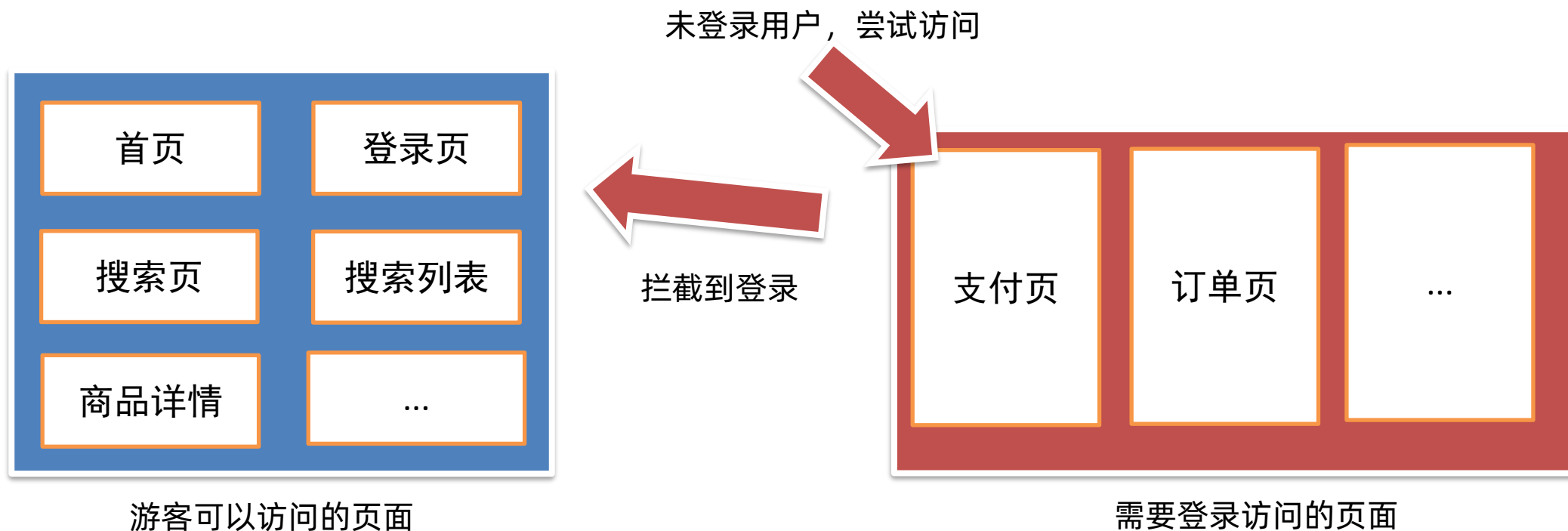


## 页面访问拦截

**目标：基于全局前置守卫，进行页面访问拦截处理**

说明：智慧商城项目，大部分页面，**游客都可以直接访问**，如遇到需要登录才能进行的操作，提示并跳转到登录

但是：对于支付页，订单页等，必须是登录的用户才能访问的，游客不能进入该页面，**需要做拦截处理**



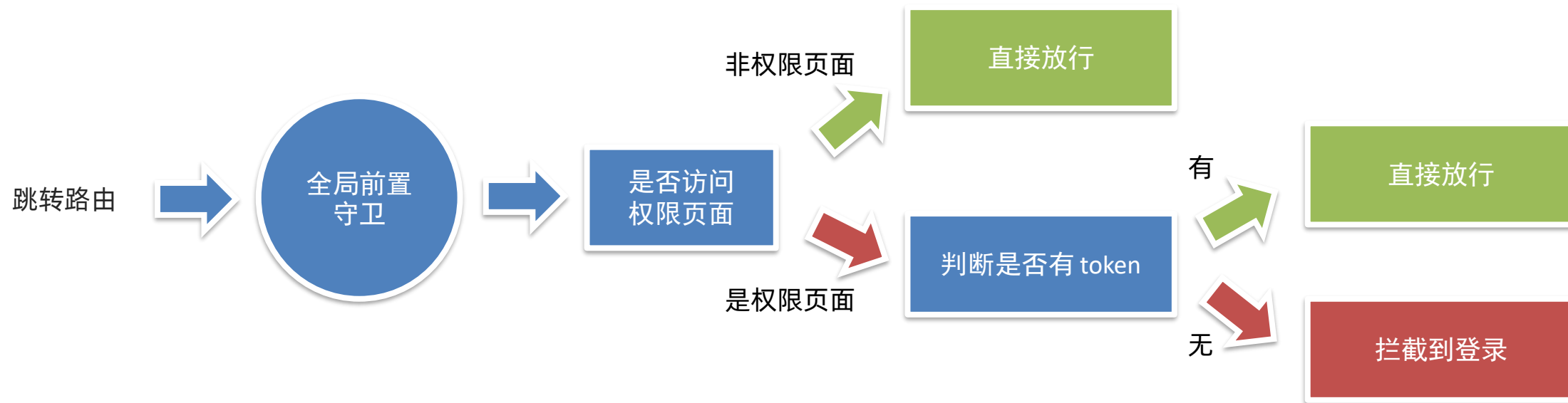
## 页面访问拦截

目标：基于全局前置守卫，进行页面访问拦截处理

路由导航守卫 - [全局前置守卫](#)

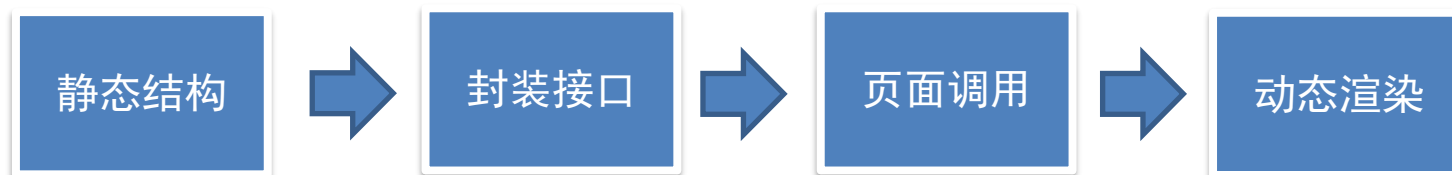
1. 所有的路由一旦被匹配到，都会先经过全局前置守卫
  2. 只有全局前置守卫放行，才会真正解析渲染组件，才能看到页面内容
- 访问权限页面时，拦截或放行的关键点？ → 用户是否有登录权证 token

```
router.beforeEach((to, from, next) => {  
  // 1. to  去哪里去， 到哪去的路由信息对象  
  // 2. from 从哪里来， 从哪来的路由信息对象  
  // 3. next() 是否放行  
  //    如果next()调用，就是放行  
  //    next(路径) 拦截到某个路径页面  
})
```



## 首页 - 静态结构准备 & 动态渲染

目标：实现首页静态结构，封装接口，完成首页动态渲染



## 搜索 - 历史记录管理

**目标：构建搜索页的静态布局，完成历史记录的管理**

需求：

1. 搜索历史基本渲染
2. 点击搜索 (添加历史)

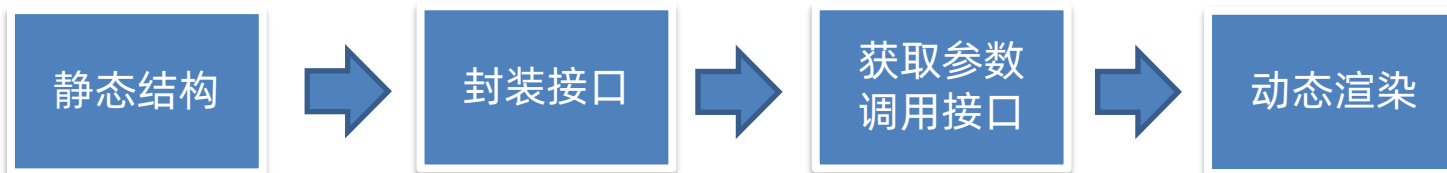
点击 搜索按钮 或 底下历史记录，都能进行搜索

- ① 若之前 **没有** 相同搜索关键字，则直接**追加到最前面**
  - ② 若之前 **已有** 相同搜索关键字，将该**原有关键字移除**，再**追加**
3. 清空历史：添加清空图标，可以清空历史记录
  4. 持久化：搜索历史需要持久化，刷新历史不丢失



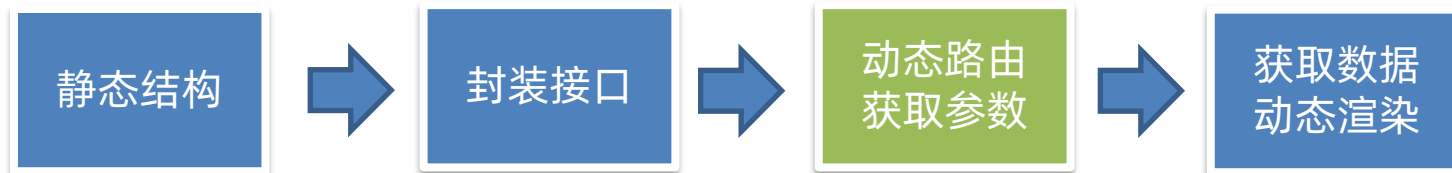
## 搜索列表 - 静态布局 & 动态渲染

目标：实现搜索列表页静态结构，封装接口，完成搜索列表页的渲染



## 商品详情- 静态布局 & 渲染

目标：实现商品详情静态结构，封装接口，完成商品详情页渲染





## 加入购物车 - 唤起弹层

目标：点击加入购物车，唤起弹层效果



## 加入购物车 - 封装数字框组件

### 目标：封装弹层中的数字框组件

分析：组件名 CountBox

1. 静态结构，左中右三部分
2. 数字框的数字，应该是外部传递进来的 (父传子)
3. 点击 + - 号，可以修改数字 (子传父)
4. 使用 **v-model** 实现封装 (:value 和 @input 的简写)
5. 数字不能减到小于 1
6. 可以直接输入内容，输入完成判断是否合法

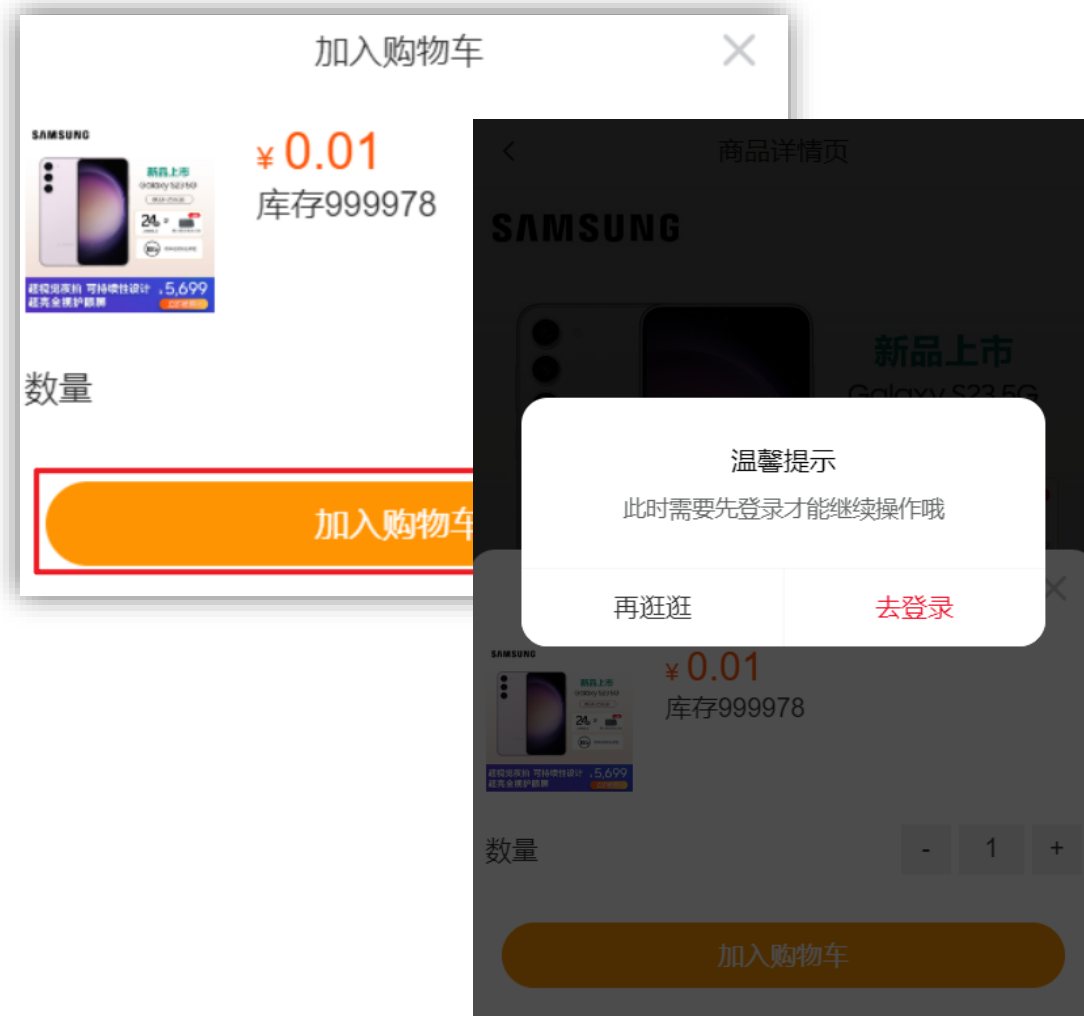


## 加入购物车 - 判断 token 添加登录提示

**目标：给未登录的用户，添加登录提示**

说明：加入购物车，是一个 **登录后的用户** 才能进行的操作  
所以需要进行鉴权判断，判断用户 token 是否存在

1. 若存在：继续加入购物车操作
2. 不存在：**提示** 用户未登录，引导到登录页，**登录完回跳**



## 加入购物车 - 封装接口进行请求

目标：封装接口，进行加入购物车的请求

1. api/cart.js 中封装接口
2. 页面中调用接口
3. 遇到问题：接口需要传递 token
4. 解决问题：请求拦截器统一携带 token
5. 小图标定制



```
✖ [Vue warn]: Error in v-on handler (Promise/async): "缺少必要的参数token, 请先登录"
found in
---> <ProDetail> at src/views/prodetail/index.vue
      <App> at src/App.vue
      <Root>
✖ 缺少必要的参数token, 请先登录
```

## 购物车模块

说明：购物车 **数据联动关系** 较多，且通常会封装一些 **小组件**，  
所以为了便于维护，一般都会将购物车的数据基于 **vuex 分模块管理**

需求分析：

1. 基本静态结构 (快速实现)
2. 构建 **vuex cart 模块**，获取数据存储
3. 基于 数据 **动态渲染** 购物车列表
4. 封装 **getters** 实现动态统计
5. 全选反选功能
6. 数字框修改数量功能
7. 编辑切换状态，删除功能
8. 空购物车处理



## 订单结算台

说明：所有的结算，本质上就是 **跳转到 "订单结算台"**，并且，跳转的同时，需要 **携带上对应的订单相关参数**，具体需要哪些参数，基于 "订单结算台" 的需求来定。



① 确认收货地址

② 确认订单信息

## 订单结算台 - 确认订单信息

目标：封装通用的订单信息确认接口

说明：这里的订单信息确认结算，有两种情况

1. 购物车结算
2. 立即购买结算

订单信息确认，可以共用同一个接口(参数不同)



## 订单结算台 - 购物车结算

目标：购物车结算跳转，**传递参数**，调用接口渲染订单结算台

核心步骤：

1. 跳转传递查询参数 `mode="cart"` 和 `cartIds`
2. 页面中 `$route.query` 接收参数
3. 调用接口，获取数据
4. 基于数据渲染





## 订单结算台 - 立即购买结算

目标：购物车结算跳转，**传递参数**，调用接口渲染订单结算台

核心步骤：

1. 跳转传递查询参数

`mode="buyNow", goodsId, goodsSkuld, goodsNum`

2. 页面中 `$route.query` 接收参数

3. 调用接口，获取数据

4. 基于数据渲染

5. 未登录时，确认框的复用 (**mixins混入**)



## 提交订单并支付

目标：封装 API 请求方法，提交订单并支付

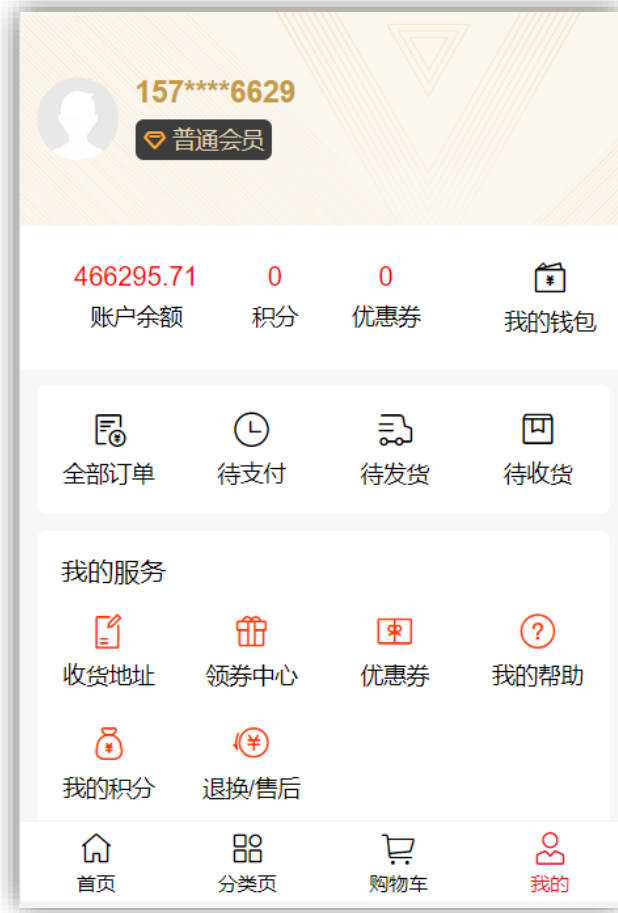
核心步骤：

1. 封装通用请求方法
2. 买家留言绑定
3. 注册事件，调用方法提交订单并支付



## 订单管理 & 个人中心 (快速实现)

目标：基于笔记，快速实现 订单管理 和 个人中心 跑通流程



## 订单管理 & 个人中心 (快速实现)

目标：基于笔记，快速实现 订单管理 和 个人中心 跑通流程



首页



选品



购物车



结算订单



订单管理

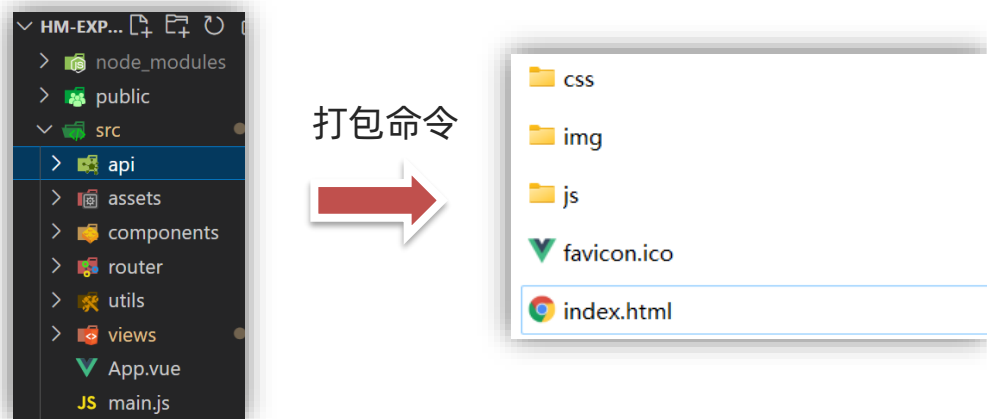
## 打包发布

### 目标：明确打包的作用

说明：vue脚手架只是开发过程中，协助开发的工具，当真正开发完了 => 脚手架不参与上线

打包的作用：

- ① 将多个文件压缩合并成一个文件
- ② 语法降级
- ③ less sass ts 语法解析
- ④ ....



打包后，可以生成，浏览器能够直接运行的网页 => 就是需要上线的源码！

## 打包发布

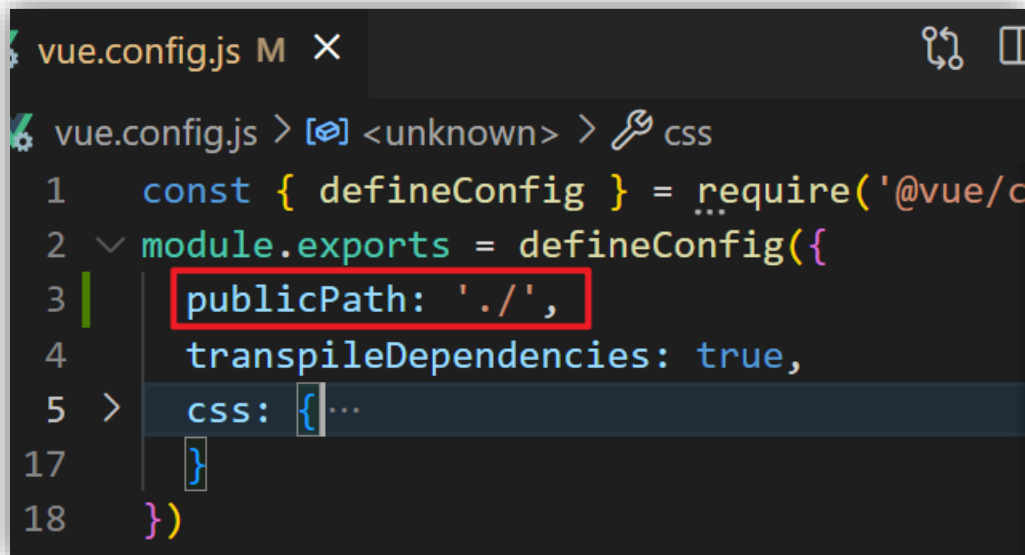
### 目标：打包的命令 和 配置

说明：vue脚手架工具已经提供了打包命令，直接使用即可。

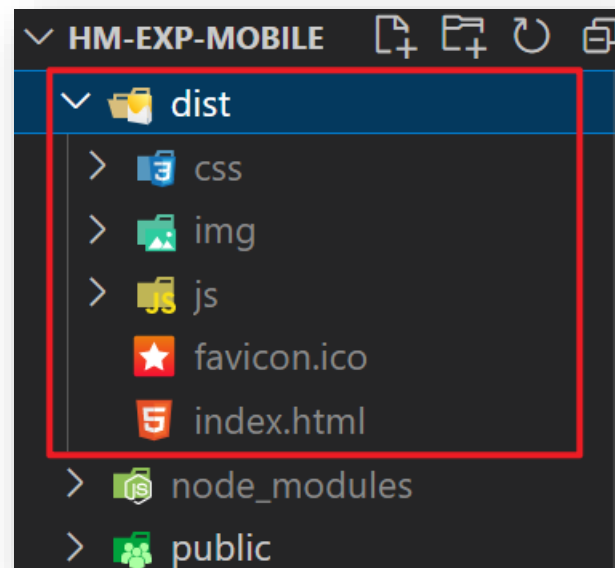
命令：yarn build

结果：在项目的根目录会自动创建一个文件夹`dist`，dist中的文件就是打包后的文件，只需要放到服务器中即可。

配置：默认情况下，需要放到服务器根目录打开，如果希望双击运行，需要配置publicPath 配成相对路径



```
vue.config.js M x
vue.config.js > [?] <unknown> > [?] css
1  const { defineConfig } = require('@vue/c
2  module.exports = defineConfig({
3    publicPath: './',
4    transpileDependencies: true,
5  >  css: { ...
17  }
18  })
```



## 打包优化：路由懒加载

**目标：**配置路由懒加载，实现打包优化

**说明：**当打包构建应用时，JavaScript 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。

[官方链接](#)

步骤1：异步组件改造

```
const ProDetail = () => import('@/views/prodetail')
const Pay = () => import('@/views/pay')
...
```

步骤2：路由中应用

```
const router = new VueRouter({
  routes: [
    ...
    { path: '/prodetail/:id', component: ProDetail },
    { path: '/pay', component: Pay },
    ...
  ]
})
```



传智教育旗下高端IT教育品牌