# Vue 核心技术与实战

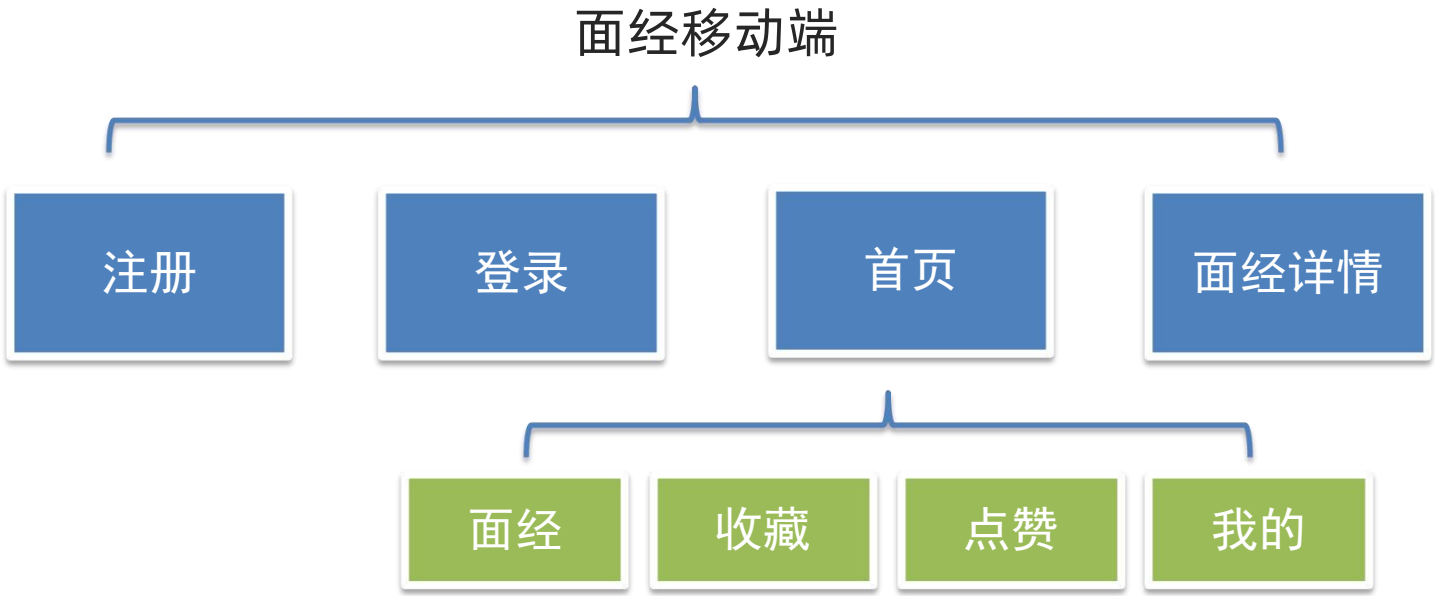
## 面经 H5

黑马程序员
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

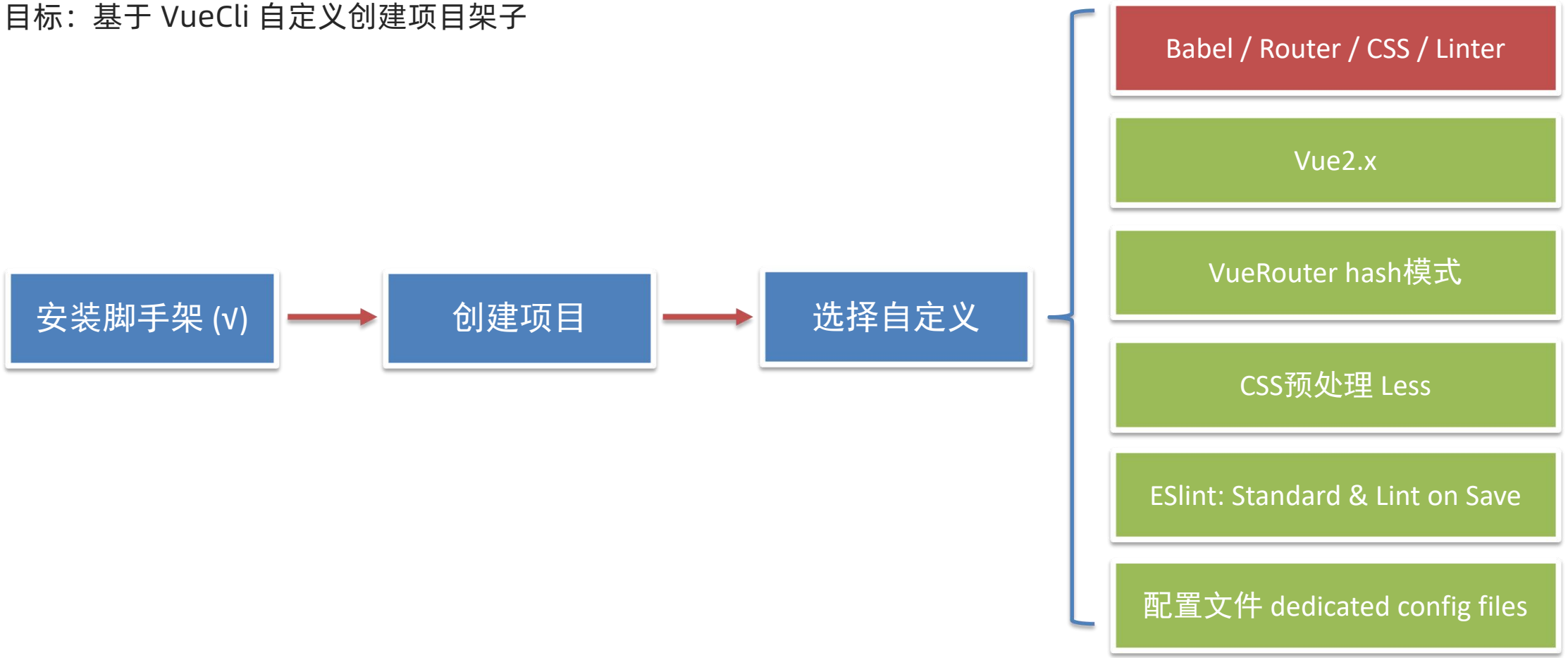
## 项目收获

目标：明确做完本项目，能够收获哪些内容

| | | |
|---|---|---|
| 脚手架自定义项目构建 | 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 |
| 嵌套路由配置 | 请求响应拦截器 | 路由导航守卫 |
| 动态路由传参 | 分页加载更多 | 项目打包&优化 |

# 创建项目

目标：基于 VueCli 自定义创建项目架子



# ESlint 代码规范

**目标：认识代码规范**

代码规范：一套写代码的约定规则。例如："赋值符号的左右是否需要空格" "一句结束是否是要加;" ...

老话说："没有规矩不成方圆" → 正规的团队 需要 统一的编码风格

JavaScript Standard Style 规范说明 https://standardjs.com/rules-zhcn.html

下面是这份规则中的一小部分：

- 字符串使用单引号 'abc'
- 无分号 const name = 'zs'
- 关键字后加空格 if (name = 'ls') { ... }
- 函数名后加空格 function name (arg) { ... }
- 坚持使用全等 === 摒弃 ==

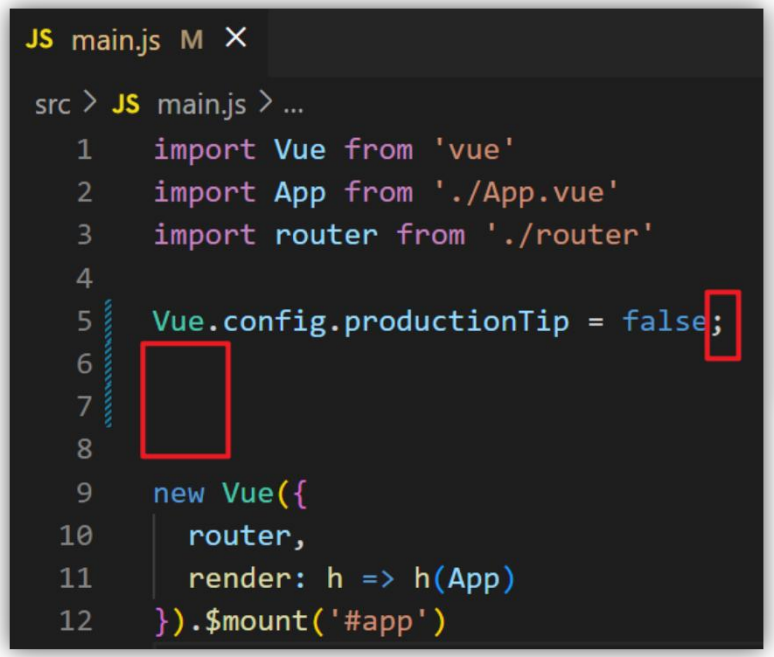
...

# 代码规范错误

**目标：学会解决代码规范错误**

如果你的代码不符合 standard 的要求，ESlint 会跳出来刀子嘴，豆腐心地提示你。

比如：在main.js中随意做一些改动，添加一些分号，空行。




# 代码规范错误

**目标：学会解决代码规范错误**

两种解决方案：

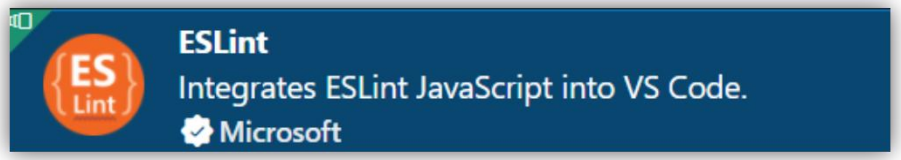
① 手动修正

根据错误提示来一项一项手动修改纠正。

如果你不认识命令行中的语法报错是什么意思，根据错误代码去 [ESLint 规则表] 中查找其具体含义。

② 自动修正

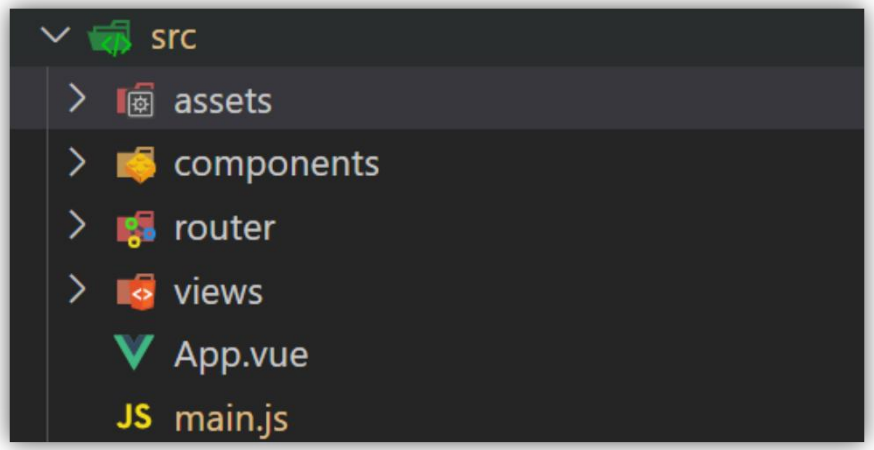
基于 vscode 插件 ESLint 高亮错误，并通过配置 自动 帮助我们修复错误。

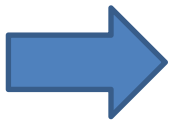


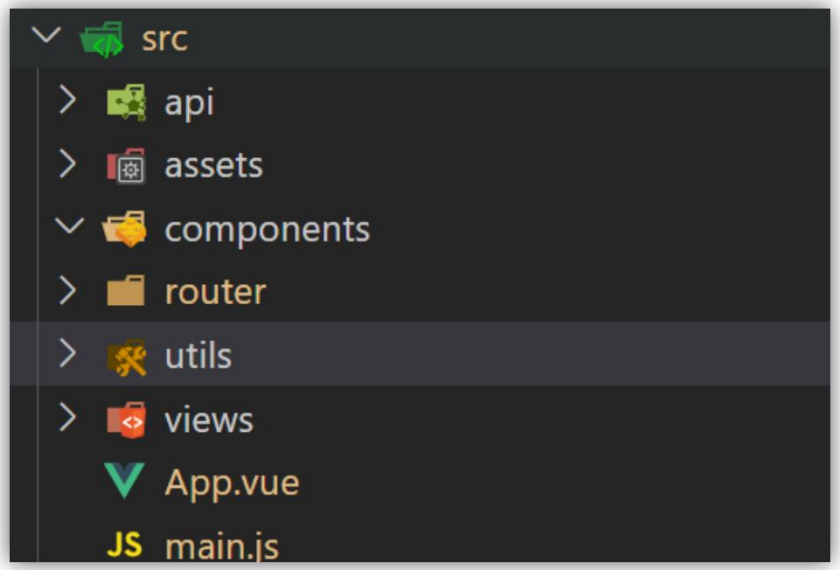

```
// 当保存的时候，eslint自动帮我们修复错误
"editor.codeActionsOnSave": {
  "source.fixAll": true
},
// 保存代码，不自动格式化
"editor.formatOnSave": false
```

# 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)

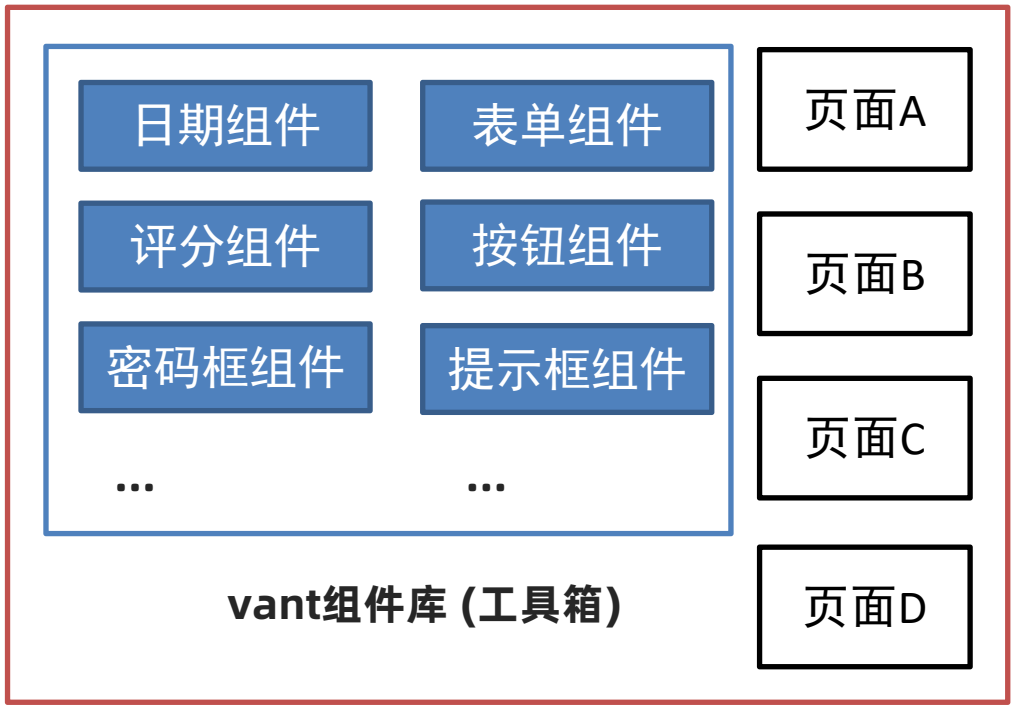
维护状态

目前 Vant 各个版本的维护状态如下：

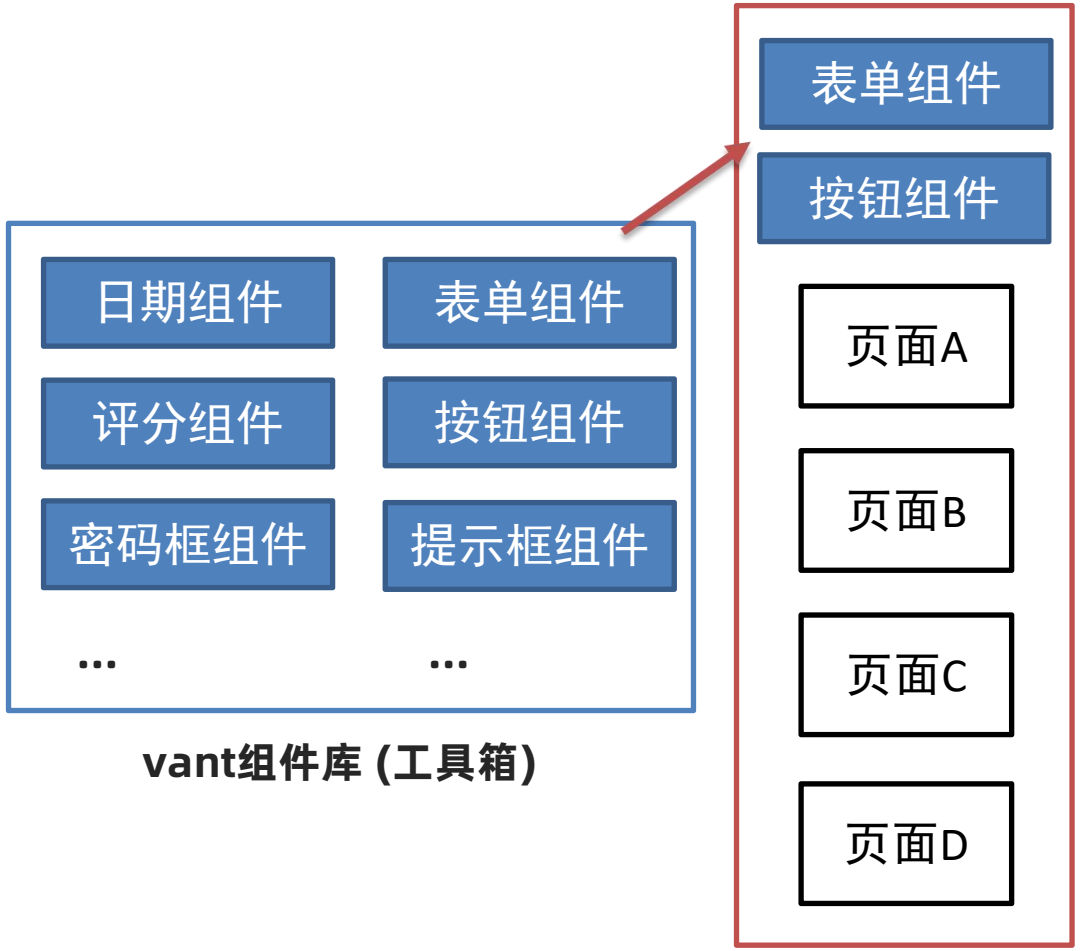
| 名称 | 框架 | 发布时间 | 最新版 | 维护状态 |
| --- | --- | --- | --- | --- |
| Vant 4 | Vue 3 | 2022. 12 | npm@latest v4.1.2 | 持续迭代新功能 |
| Vant 3 | Vue 3 | 2020. 12 | npm@latest-v3 v3.6.11 | 停止迭代新功能，bug 会被处理和修复 |
| Vant 2 | Vue 2 | 2019. 06 | npm@latest-v2 v2.12.54 | 停止迭代新功能，重要 bug 会被处理和修复 |
| Vant 1 | Vue 2 | 2018.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

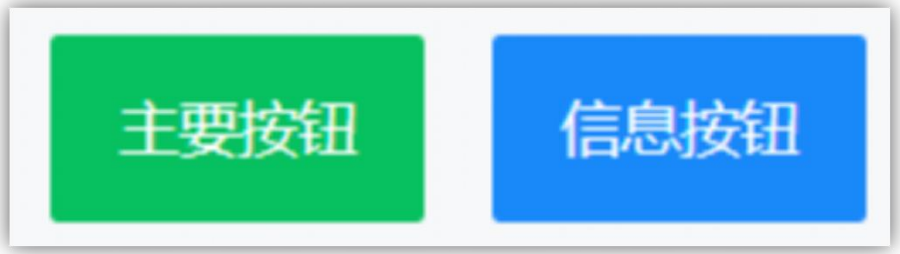
全部导入：

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)
```




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**

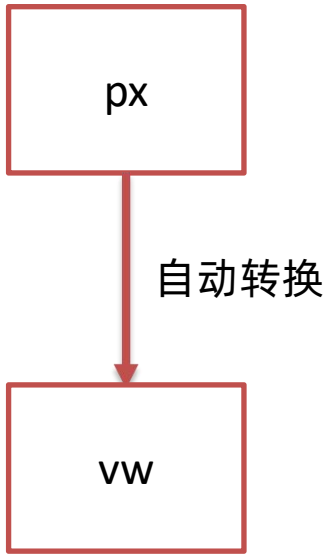
官方配置

① 安装插件

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

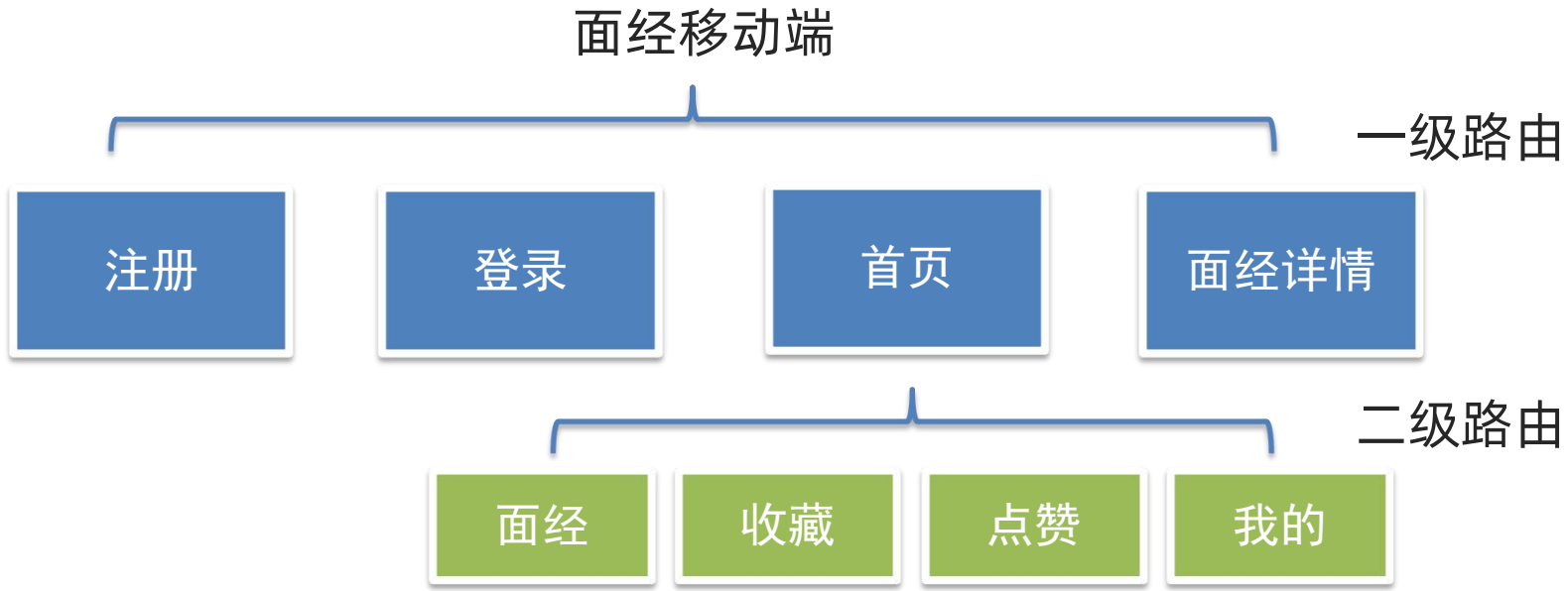
② 根目录新建 postcss.config.js 文件，填入配置

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



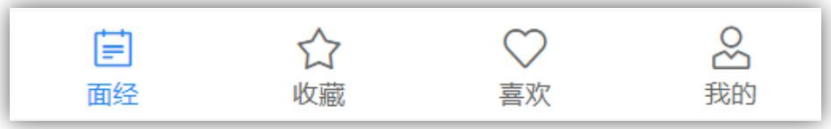

# 路由设计配置

**目标：分析项目页面，设计路由，配置一级路由**

# 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，进行定制

```
<van-tabbar-item icon="notes-o">面经</van-tabbar-item>
<van-tabbar-item icon="star-o">收藏</van-tabbar-item>
<van-tabbar-item icon="like-o">喜欢</van-tabbar-item>
<van-tabbar-item icon="user-o">我的</van-tabbar-item>
```

# 路由设计配置

**目标：通过 配置主题色，实现底部导航定制**



配置主题色：

① babel.config.js 指定样式路径

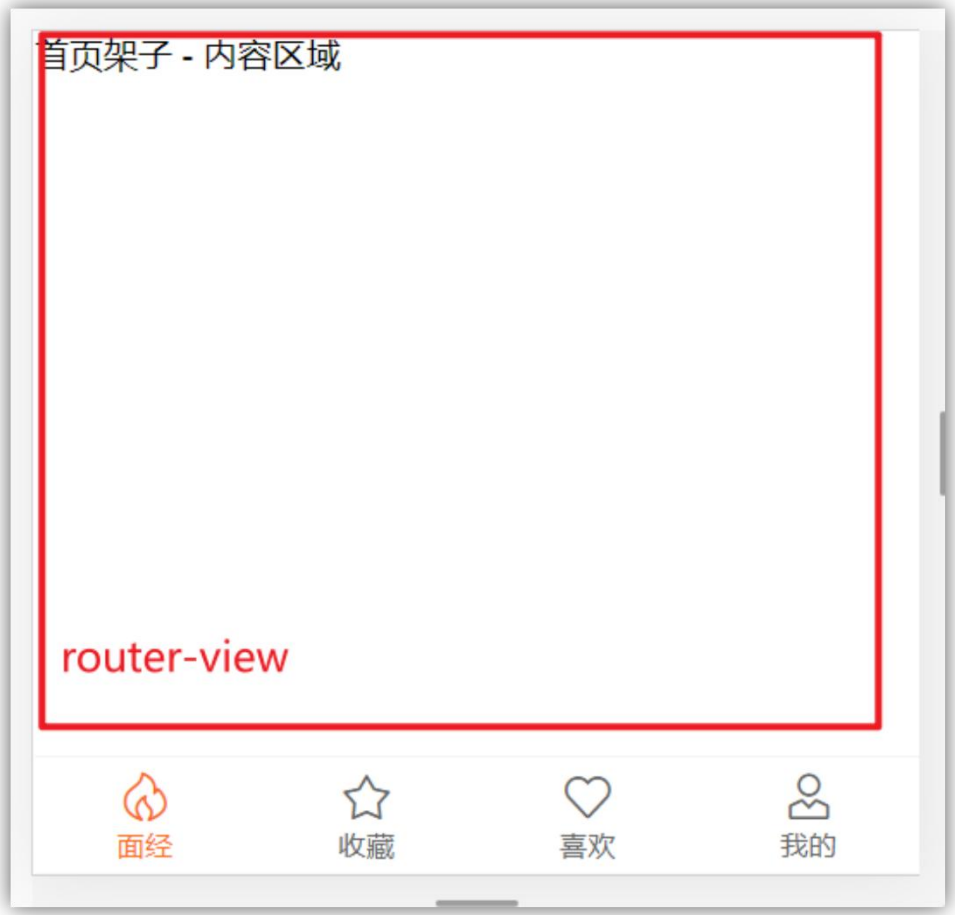
```
module.exports = {
  presets: [
    '@vue/cli-plugin-babel/preset'
  ],
  plugins: [
    ['import', {
      libraryName: 'vant',
      libraryDirectory: 'es',
      // 指定样式路径
      style: (name) => `${name}/style/less`
    }, 'vant']
  ]
}
```

② vue.config.js 覆盖变量

```
const { defineConfig } = require('@vue/cli-service')
module.exports = defineConfig({
  transpileDependencies: true,
  css: {
    loaderOptions: {
      less: {
        lessOptions: {
          modifyVars: {
            // 直接覆盖变量
            blue: '#FA6D1D'
          }
        }
      }
    }
  }
})
```

# 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

children

配置导航链接

```
<van-tabbar route>
  <van-tabbar-item to="/article" ...>
    面经
  </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 & 注册功能

**目标：阅读文档，实现 登录页 & 注册页 静态布局**


## 登录 & 注册功能

**目标：阅读文档，实现 登录页 & 注册页 静态布局**



van-nav-bar

van-form

van-field

# 登录 & 注册功能

**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


安装 axios → 新建 request 模块 → 创建实例 / 自定义配置 / 导出实例 → 测试使用

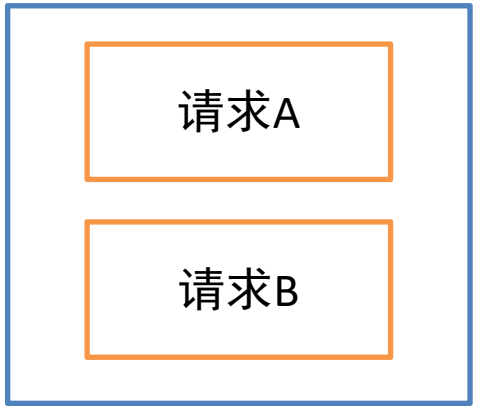
utils/request.js

接口文档地址　　基地址：http://interview-api-t.itheima.net/h5/

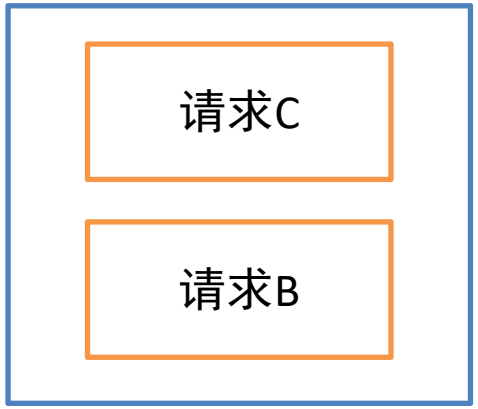
axios文档地址

## 登录 & 注册功能

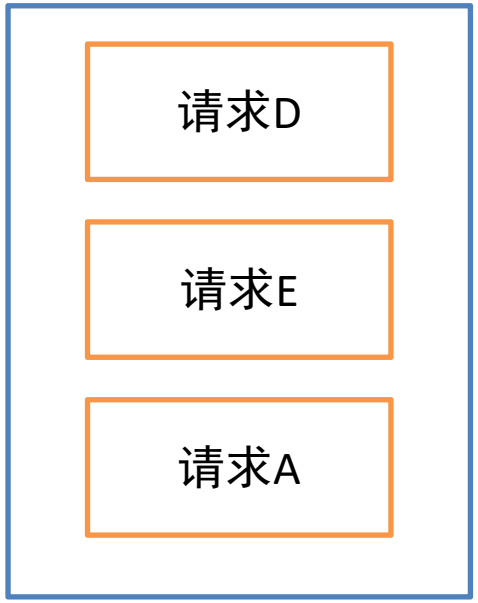
**目标：将请求封装成方法，统一存放到 api 模块，与页面分离**




页面1




页面2



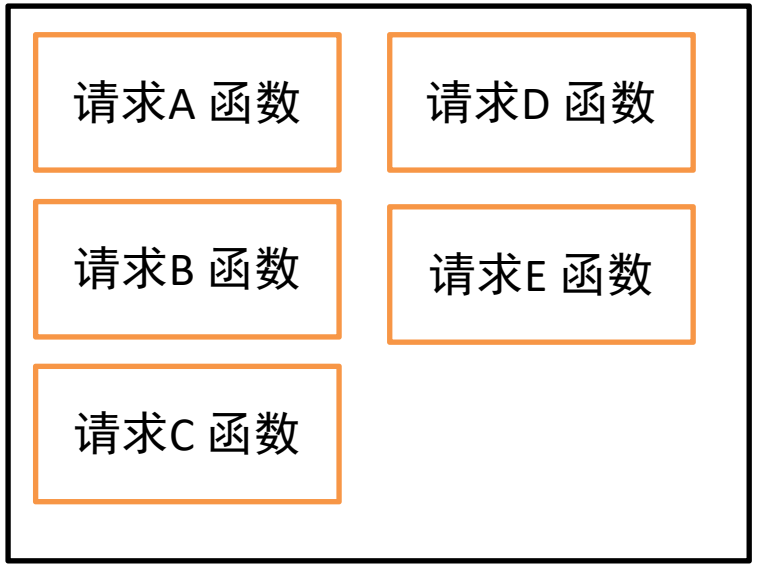

页面3

以前的模式：

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

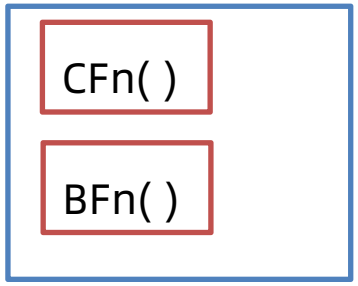
## 登录 & 注册功能

**目标：将请求封装成方法，统一存放到 api 模块，与页面分离**

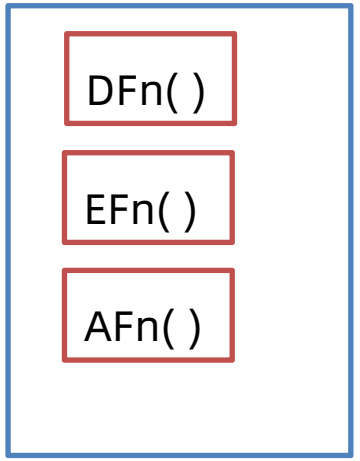



API模块 (存放封装好的请求函数)


页面1




页面2



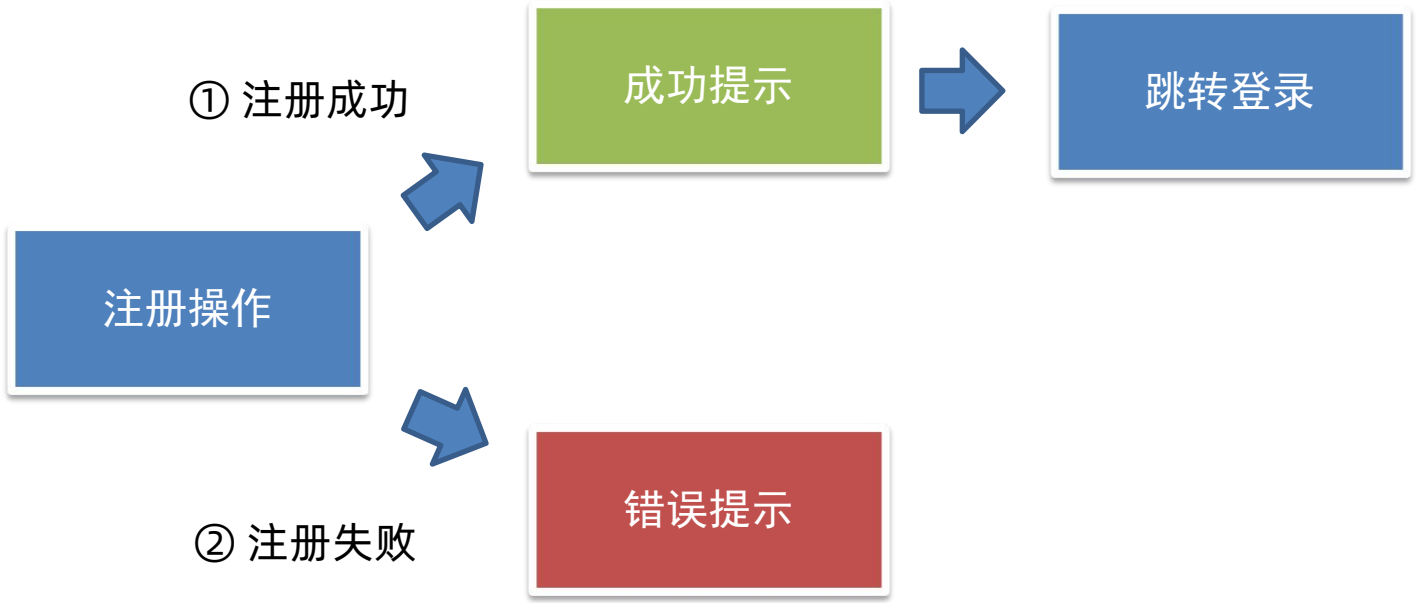

页面3

封装api模块：

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

# 登录 & 注册功能

**目标：阅读文档 toast轻提示，完成 注册功能 提示&跳转**




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（非组件内）

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

② 组件内 this 直接调用

```
this.$toast('提示内容')
```

# 登录 & 注册功能

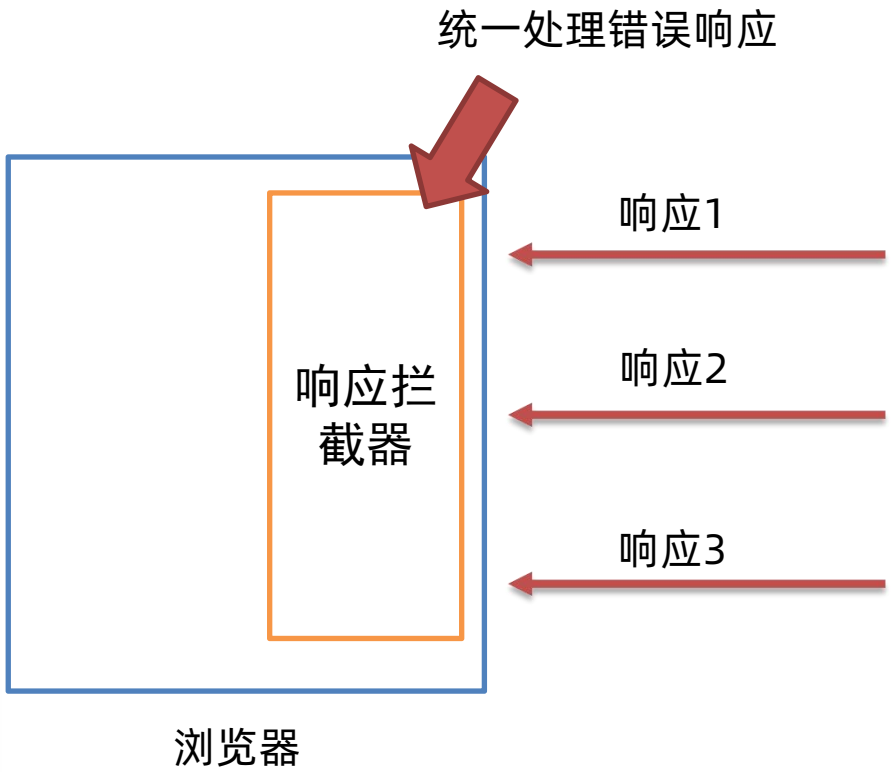
**目标：响应拦截器中统一处理错误提示**

问题：每次请求，都会有可能会错误，就都需要错误提示，

每次 try catch 很麻烦，能不能统一处理呢？

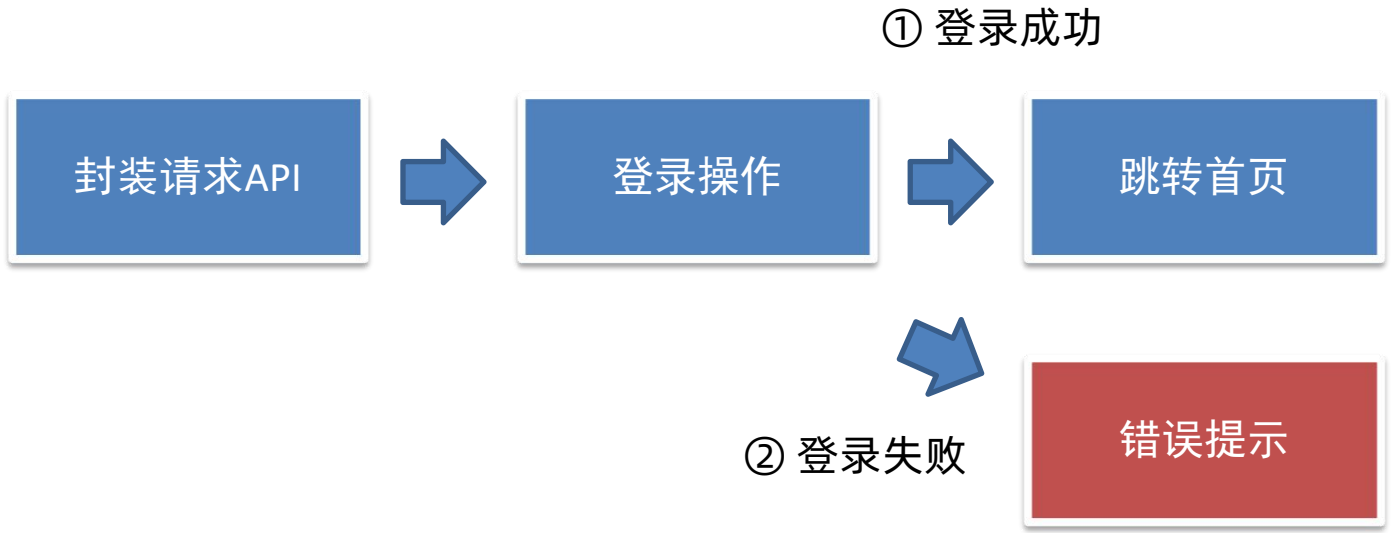
```
try {
  await register(values)
  this.$toast.success('注册成功')
  this.$router.push('/login')
} catc
  this
}
```

```
// 添加响应拦截器
instance.interceptors.response.use(function (response) {
  // 对响应数据做点什么
  return response
}, function (error) {
  // console.log(error)
  // 有错误响应，后台正常返回了错误信息
  if (error.response) {
    // 有错误响应，提示错误消息
    // this.$toast(error.response.data.message)
    Toast(error.response.data.message)
  }
  // 对响应错误做点什么
  return Promise.reject(error)
})
```



# 登录 & 注册功能

**目标：封装 api 完成登录功能**

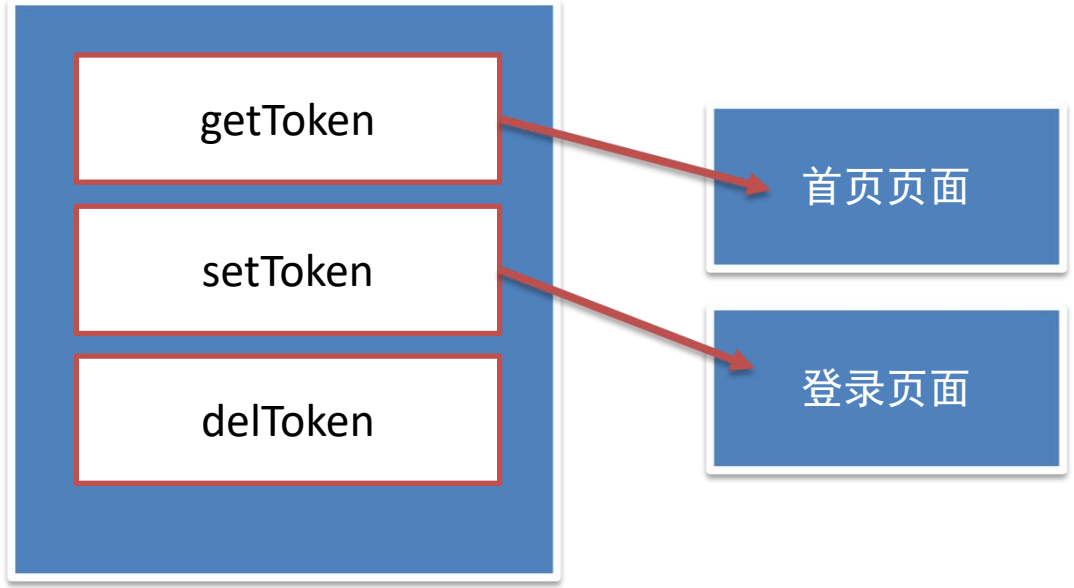

已在响应拦截器统一处理

## 登录 & 注册功能

**目标：local 模块，本地存储封装**

问题：刚才我们存了 token，为了防止重名，起的名字很长，方便使用不？

```
// 将token存入本地
localStorage.setItem('vant-mobile-exp-token', data.data.token)
```



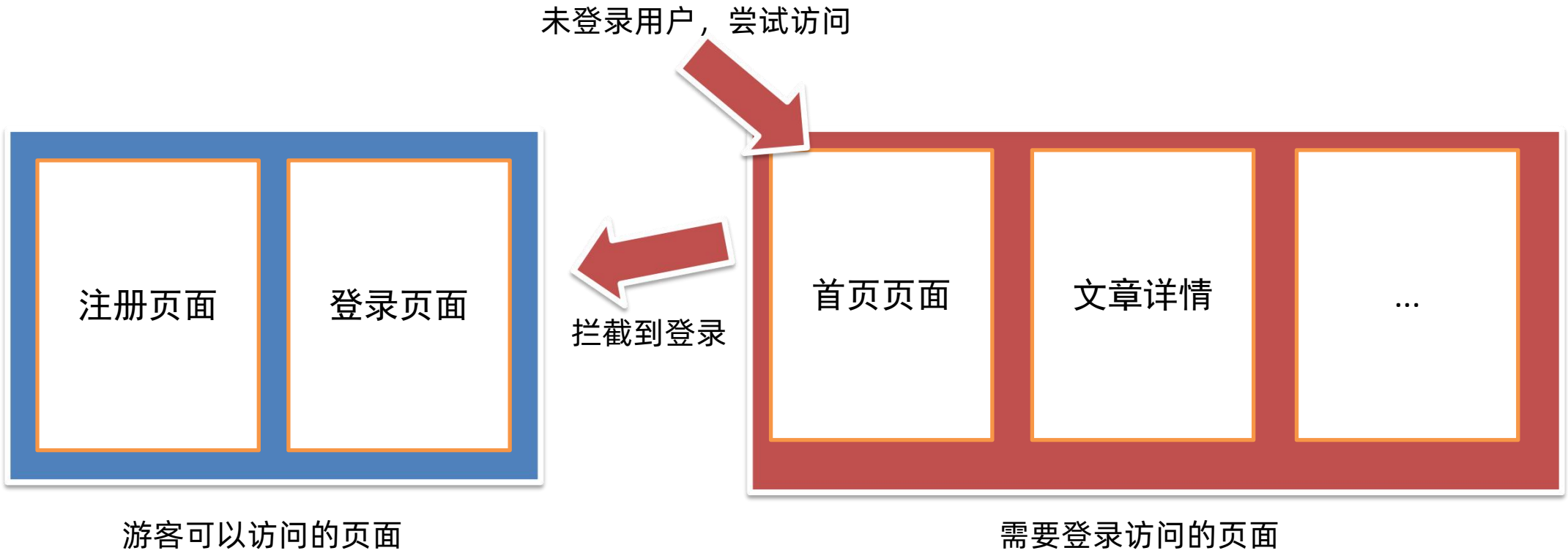

local模块

```
const KEY = 'vant-mobile-exp-token'
// 获取
export const getToken = () => {
  return localStorage.getItem(KEY)
}
// 设置
export const setToken = (newToken) => {
  localStorage.setItem(KEY, newToken)
}
// 删除
export const delToken = () => {
  localStorage.removeItem(KEY)
}
```

# 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：面经移动端项目，只对 登录用户 开放，如果未登录，一律拦截到登录



# 页面访问拦截

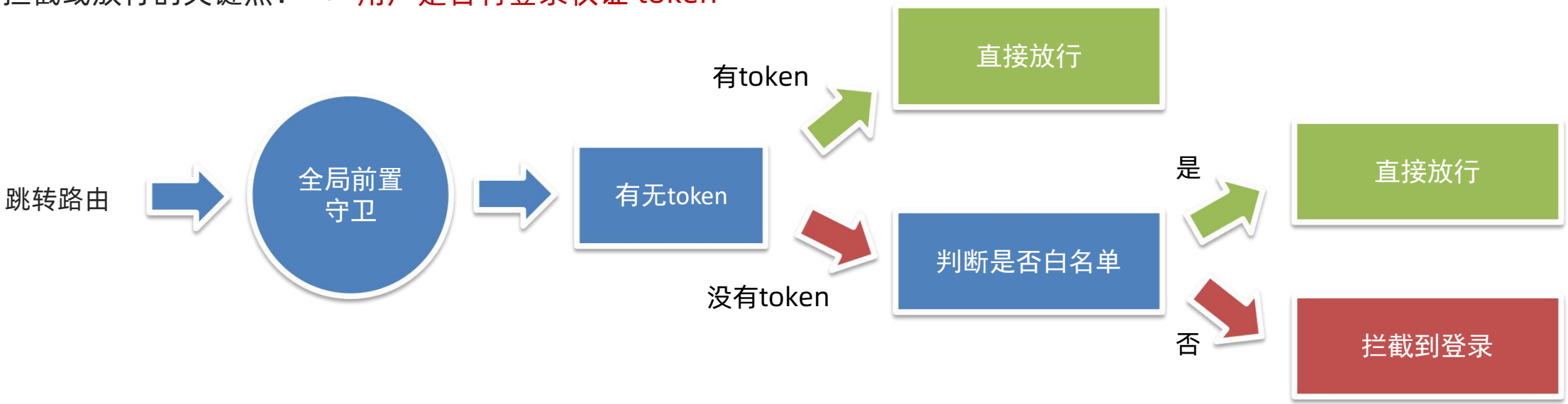
**目标：基于全局前置守卫，进行页面访问拦截处理**

```
router.beforeEach((to, from, next) => {
  // 1. to   往哪里去，  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
})
```

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

拦截或放行的关键点？ → 用户是否有登录权证 token



# 面经列表

**目标：认识 Cell组件，准备首页面经的基本布局**

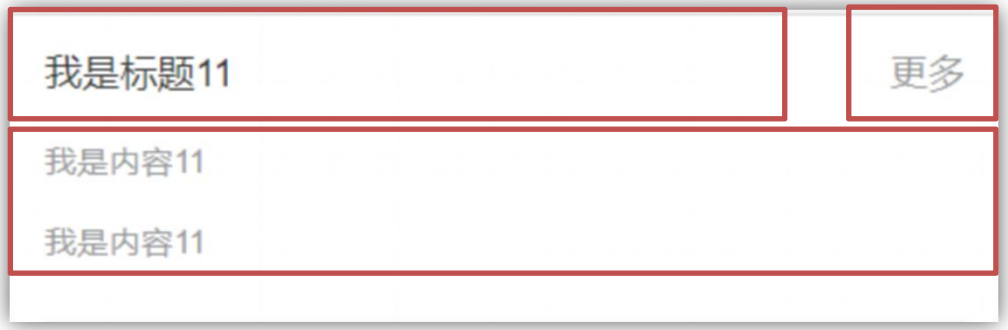
普通导航部分


一篇文章

van-cell组件

① title插槽

③ default插槽




② label插槽

Cell 单元格组件

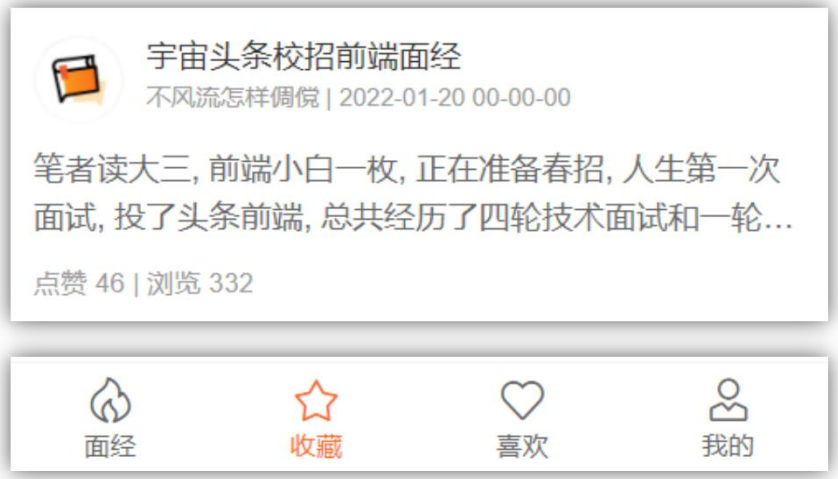
# 面经列表

**目标：封装通用组件 - 文章组件**

说明：每个文章列表项，其实就是一个整体，封装成一个组件 → 可阅读性 & 复用性

步骤：

1. 新建 components/ArticleItem.vue 组件，贴入内容
2. 注册成全局组件
3. Article.vue 页面中应用





文章组件

# 面经列表

**目标：封装接口 - 获取文章列表数据**

说明：目前文章列表是有的，但是数据是写死的，需要请求真实的文章列表数据回来

步骤：

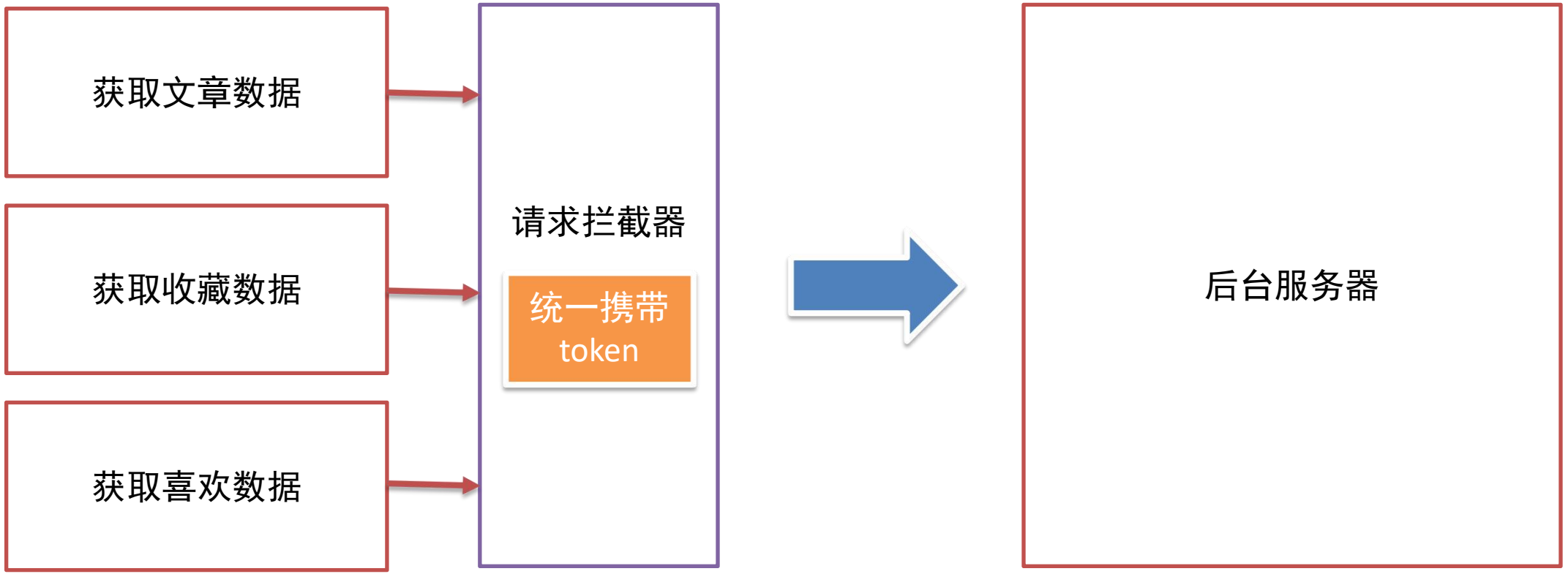
1. 阅读接口文档，在 api/article.js 封装接口函数
2. 页面调用，请求测试
3. 发现 401 错误，需要请求头 headers 中，携带 token

```
// 获取文章列表接口封装完成
export const getArticles = (obj) => {
  const token = getToken()
  return request.get('/interview/query', {
    // get请求参数
    params: {
      current: obj.current || 1, // 当前页 → 做分页
      pageSize: obj.pageSize || 10, // 每页条数 → 做分页
      sorter: obj.sorter // 排序字段 → 推荐(weight_desc) & 最新(不传
    },
    // headers请求头，配置token
    headers: {
      Authorization: `Bearer ${token}`
    }
  })
}
```

## 面经列表

**目标：请求拦截器统一携带 token**

说明：每次请求 自己携带 token，太麻烦！通过请求拦截器统一携带 token 更方便



# 面经列表

**目标：响应拦截器 - 处理 token 过期**

说明：token 是有过期时间的 (6h)，一旦 过期 或 失效 就无法正确获取到数据！



过期token 进行请求 → 后台返回 401 error → 跳转登录页

```
// 添加响应拦截器
instance.interceptors.response.use(function (r
  // 对响应数据做点什么
  return response
}, function (error) {
  // console.log(error)
  // 有错误响应，后台正常返回了错误信息
  if (error.response) {
    if (error.response.status === 401) {
      // 清除掉无效的token
      delToken()
      // 拦截到登录
      router.push('/login')
    } else {
      // 有错误响应，提示错误消息
      // this.$toast(error.response.data.messa
      Toast(error.response.data.message)
    }
  }
```

响应拦截器处理

# 面经列表

**目标：动态渲染文章列表页**

```
▼ rows: Array(10)
  ▶ 0: {id: "41164", stem: "小红书前端开发面经", co…
  ▶ 1: {id: "41135", stem: "前端实习生-北京字节跳动…
  ▶ 2: {id: "40977", stem: "1文思海辉前端面经1", c…
  ▶ 3: {id: "41155", stem: "美团前端面经1", conten…
  ▶ 4: {id: "41177", stem: "测试", content: "<p>s…
  ▶ 5: {id: "41112", stem: "2021阿里前端 淘宝+河马…
```


存储数据 → v-for渲染 → 父传子传数据 → 子组件渲染

# 面经列表

**目标：响应拦截器 - 简化响应**

```
// 添加响应拦截器
instance.interceptors.response.use(function
  // 对响应数据做点什么
  return response.data
}, function (error) {
```

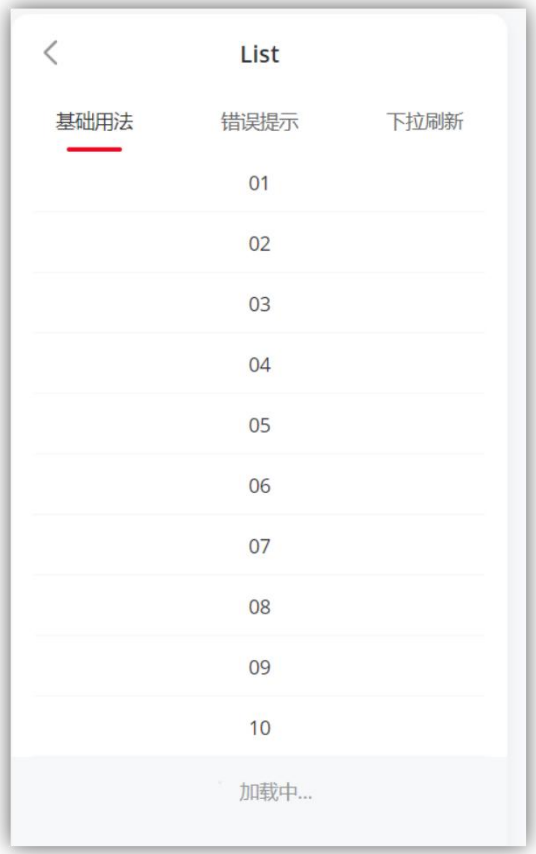
```
async created () {
// 获取推荐的，第1页的10条数据
  const res = await getArticles({
    current: this.current,
    sorter: this.sorter
  })
  this.list = res.data.data.rows
  console.log(res.data.data.rows)
},
```

```
async created () {
// 获取推荐的，第1页的10条数据
  const res = await getArticles({
    current: this.current,
    sorter: this.sorter
  })
  this.list = res.data.rows
  console.log(res.data.rows)
},
```

# 面经列表

**目标：认识 vant list 组件，实现分页渲染**

说明：首页获取了第一页数据，但很显然不够，滑动到底部需要加载下一页数据...


两个变量 + 一个事件 + 一个文本：

① loading 布尔值，是否在加载中...

true: 加载中，此时不会重复触发加载

false: 可以加载更多数据

② finished 布尔值，是否加载完成

true：数据加载完成，不会再触发加载

false：数据还没加载完成，还可以触发加载

③ load事件

组件滚动到底部，会触发load事件，将loading 改 true

④ finished-text 配置没有更多数据的文本

# 面经列表

**目标：点击 推荐 或 最新，实现文章列表数据切换**


点击事件
传递不同值

重置数据, 根据新条件,
重新请求第一页数据

处理导航高亮

推荐：weight_desc

最新：null

this.current = 1

this.list = [ ]

this.finished = false

# 面经详情 - 实战练习

**目标：配置动态路由，跳转页面传参，传递 id → 基于 id 动态渲染**


# 面经详情 - 实战练习

**目标：实现点赞收藏功能**



封装接口

↓

注册事件

↓

调用接口请求

↓

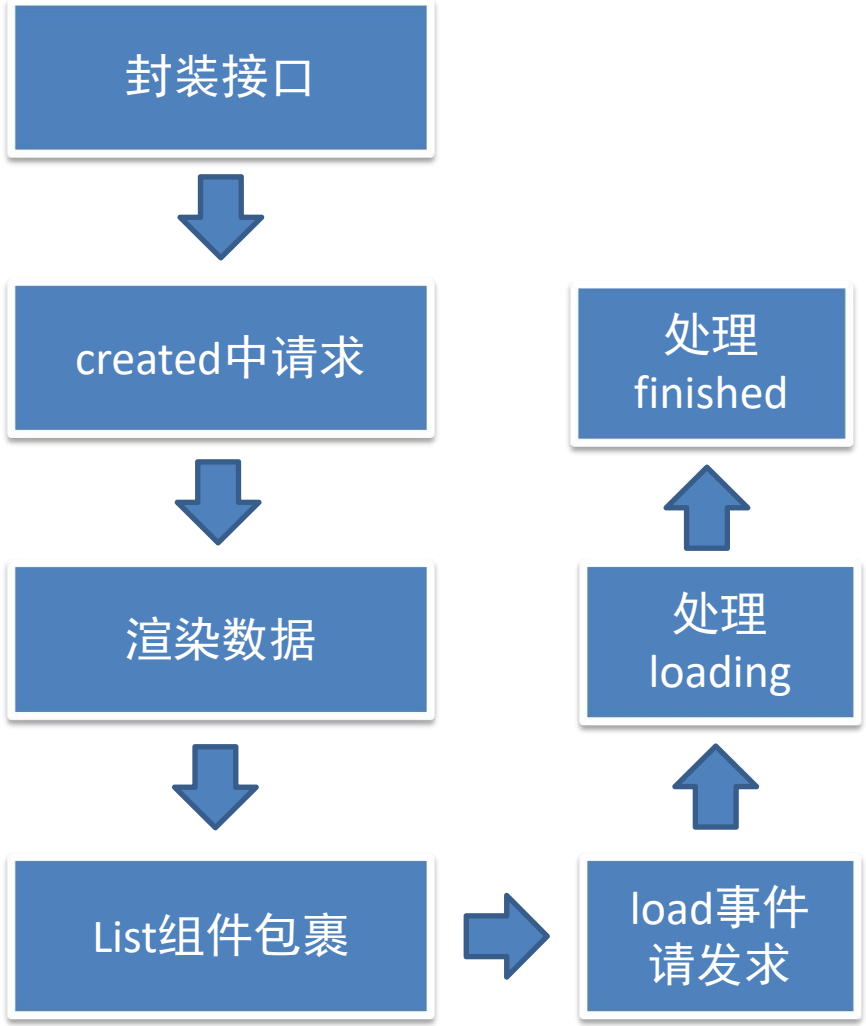
修改状态
修改文本
提示消息

# 我的收藏 - 实战练习

**目标：实现我的收藏功能**


van-nav-bar

van-list ArticleItem



# 我的喜欢 & 个人中心 - 快速实现

**目标：快速实现我的喜欢 和 个人中心模块**


# 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

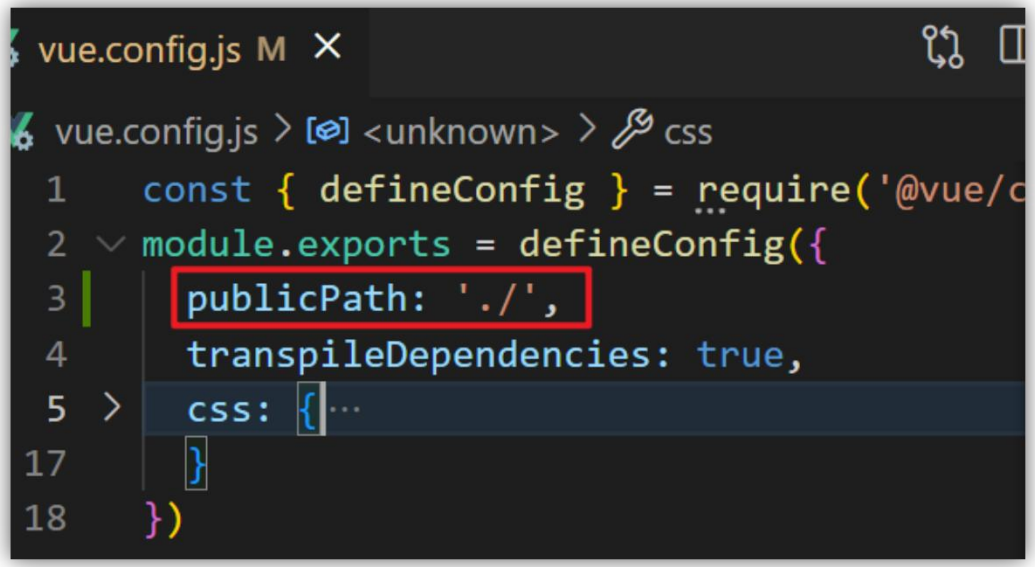
**目标：打包的命令 和 配置**

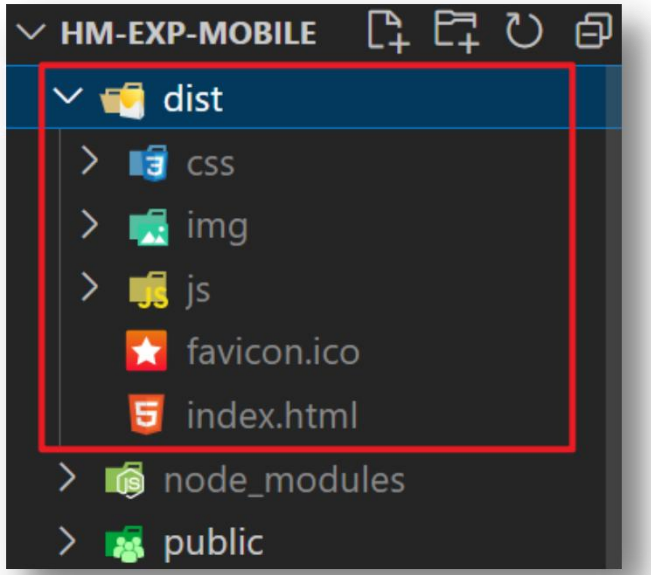
**说明：**vue脚手架工具已经提供了打包命令，直接使用即可。

**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



# 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 Detail = () => import('@/views/detail')
const Register = () => import('@/views/register')
...
```

步骤2： 路由中应用

```
const router = new VueRouter({
  routes: [
    ...
    { path: '/register', component: Register },
    { path: '/article/:id', component: Detail },
    ...
```


传智教育旗下高端IT教育品牌