**集维管理端开发文档**

**项目简介**

jiwei-vben-admin基于 Vue3.0、Vite、 Ant-Design-Vue、TypeScript 的后台管理端解决方案。

**环境准备**

本地环境需要安装 pnpm、Node.js、Git、vscode。

推荐使用pnpm，否则依赖可能安装不上(***npm install -g pnpm***)。

Node.js 版本要求16.x及以上(官网：[nodejs.org](https://nodejs.org/en)，下载：<https://nodejs.org/dist/>，另可通过nvm安装多个NodeJs版本）。

vscode 版本要求，安装官网最近稳定版即可（<https://code.visualstudio.com/>）

nvm 安装说明（[nvm的安装和使用](https://blog.csdn.net/m0_64697285/article/details/127318141)）

**工具配置**

**如果您使用的 IDE 是vscode(推荐)的话，可以安装以下工具来提高开发效率及代码格式化**

* Iconify IntelliSense - Iconify 图标插件
* windicss IntelliSense - windicss 提示插件
* I18n-ally - i18n 插件
* Vetur - vue vue插件工具 （也可以选择 Volar）
* ESLint - 脚本代码检查
* Prettier - 代码格式化
* Stylelint - css 格式化
* DotENV - .env 文件高亮

**社区文档**

Ant-Design-Vue官网（组件框架）：<https://www.antdv.com/docs/vue/introduce-cn>

Vben-Admin 官网（整体框架）：<https://doc.vvbin.cn/guide/introduction.html>

Vue3 官网（基础框架）：<https://vuejs.org/> 中文官网：<https://cn.vuejs.org/>

ES6 技术资料（ES语法）：<https://es6.ruanyifeng.com/>

TypeScript 官网（基础语言）：<https://www.typescriptlang.org/>

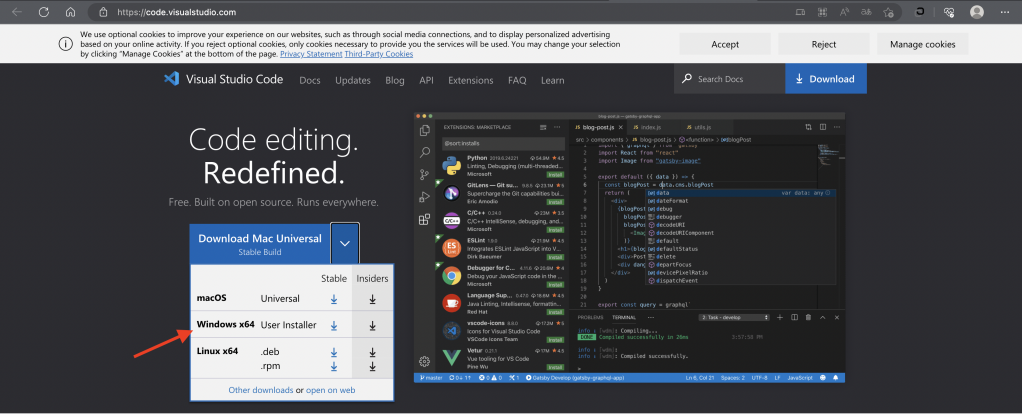
Vite 官网（打包框架）：<https://cn.vitejs.dev/guide/features.html>

Pinia 官网 (状态管理) : <https://pinia.vuejs.org/>

**安装 VSCode**

VSCode 官网：<https://code.visualstudio.com/>

根据电脑系统选择 windows版本、Mac 版本、Linux 版本，安装官网最新版即可。



***vscode 安装插件教程：[Visual Studio Code 插件安装\_vscode安装插件](https://blog.csdn.net/u014388408/article/details/131265539)***

**安装Node.js**

Node.js 官方下载地址：<https://nodejs.org/en>

Node.js 版本要求16.x及以上，可以安装当前长期支持版本 18.x，或者最新版本 20.x。

***安装完毕验证：***

***# 出现相应npm版本即可***

***npm -v***

***# 出现相应node版本即可***

***node -v***

***NodeJS版本选择：***

***官网地址：***[nodejs.org](https://nodejs.org/en) 版本选择：<https://nodejs.org/dist/>

* *v14: <https://nodejs.org/dist/latest-v14.x/>*
* *v16: <https://nodejs.org/dist/latest-v16.x/>*
* *v18: <https://nodejs.org/dist/latest-v18.x/>*
* *v20: <https://nodejs.org/dist/latest-v20.x/>*

**安装pnpm**

# 使用 npm 全局安装 pnpm：

***npm install -g pnpm***

# 使用 yarn 全局安装 pnpm：

***yarn global add pnpm***

# 使用 npx 运行 pnpm：  
***npx pnpm install***

***# 验证***

***pnpm -v # 出现对应版本号即代表安装成功***

**本地运行**

拉去项目：

git clone <http://172.20.112.101:3001/wan/scomms/code/standard-edition/frontdemo.git>

切换到开发分支：

git checkout develop

安装项目依赖：

cd [项目目录]

[sudo] npm install -g pnpm

[npm install] | [pnpm install] | [yarn]

运行项目:

[npm run dev] | [pnpm run dev] | [yarn dev]

*注意：存放代码的目录及所有父级目录不能存在中文、韩文、日文以及空格，否则安装依赖后启动会出错。*

***Package.json 配置说明***

"scripts": {

# 安装依赖

"bootstrap": "yarn install",

# 运行项目

"serve": "npm run dev",

# 运行项目

"dev": "vite",

# 构建项目

"build": "vite build && esno ./build/script/postBuild.ts",

# 清空缓存后构建项目

"build:no-cache": "yarn clean:cache && npm run build",

# 生成打包分析，在 `Mac OS` 电脑上执行完成后会自动打开界面，在 `Window` 电脑上执行完成后需要打开 `./build/.cache/stats.html` 查看

"report": "cross-env REPORT=true npm run build",

# 类型检查

"type:check": "vue-tsc --noEmit --skipLibCheck",

# 预览打包后的内容（先打包在进行预览）

"preview": "npm run build && vite preview",

# 直接预览本地 dist 文件目录

"preview:dist": "vite preview",

# 生成 ChangeLog

"log": "conventional-changelog -p angular -i CHANGELOG.md -s",

# 删除缓存

"clean:cache": "rimraf node\_modules/.cache/ && rimraf node\_modules/.vite",

# 删除 node\_modules (`window` 系统手动删除该目录较慢，可以使用该命令来进行删除)

"clean:lib": "rimraf node\_modules",

# 执行 eslint 校验，并修复部分问题

"lint:eslint": "eslint \"{src,mock}/\*\*/\*.{vue,ts,tsx}\" --fix",

# 执行 prettier 格式化（该命令会对项目所有代码进行 prettier 格式化，请谨慎执行）

"lint:prettier": "prettier --write --loglevel warn \"src/\*\*/\*.{js,json,tsx,css,less,scss,vue,html,md}\"",

# 对打包结果进行 gzip 测试

"test:gzip": "http-server dist --cors --gzip -c-1",

# 对打包目录进行 brotli 测试

"test:br": "http-server dist --cors --brotli -c-1",

# 重新安装依赖，见下方说明

"reinstall": "rimraf yarn.lock && rimraf package.lock.json && rimraf node\_modules && npm run bootstrap",

# 生成图标集，见下方说明

"gen:icon": "esno ./build/generate/icon/index.ts",

},

**目录说明**

├── build # 打包脚本相关

│ ├── config # 配置文件

│ ├── generate # 生成器

│ ├── script # 脚本

│ └── vite # vite配置

├── mock # mock文件夹

├── public # 公共静态资源目录

├── src # 主目录

│ ├── api # 接口文件

│ ├── assets # 资源文件

│ │ ├── icons # icon sprite 图标文件夹

│ │ ├── images # 项目存放图片的文件夹

│ │ └── svg # 项目存放svg图片的文件夹

│ ├── components # 公共组件

│ ├── design # 样式文件

│ ├── directives # 指令

│ ├── enums # 枚举/常量

│ ├── hooks # hook

│ │ ├── component # 组件相关hook

│ │ ├── core # 基础hook

│ │ ├── event # 事件相关hook

│ │ ├── setting # 配置相关hook

│ │ └── web # web相关hook

│ ├── layouts # 布局文件

│ │ ├── default # 默认布局

│ │ ├── iframe # iframe布局

│ │ └── page # 页面布局

│ ├── locales # 多语言

│ ├── logics # 逻辑

│ ├── main.ts # 主入口

│ ├── router # 路由配置

│ ├── settings # 项目配置

│ │ ├── componentSetting.ts # 组件配置

│ │ ├── designSetting.ts # 样式配置

│ │ ├── encryptionSetting.ts # 加密配置

│ │ ├── localeSetting.ts # 多语言配置

│ │ ├── projectSetting.ts # 项目配置

│ │ └── siteSetting.ts # 站点配置

│ ├── store # 数据仓库

│ ├── utils # 工具类

│ └── views # 页面

│ └── server # 测试用到的服务

│ ├── api # 测试服务器

│ ├── upload # 测试上传服务器

│ └── websocket # 测试ws服务器

├── types # 类型文件

├── vite.config.ts # vite配置文件

└── windi.config.ts # windcss配置文件

**环境变量配置**

项目的环境变量配置位于项目根目录下的 .env、.env.development、.env.production

**配置项说明**

**.env 所有环境适用**

# 端口号

VITE\_PORT=3100

# 网站标题

VITE\_GLOB\_APP\_TITLE=vben admin

# 简称，用于配置文件名字 不要出现空格、数字开头等特殊字符

VITE\_GLOB\_APP\_SHORT\_NAME=vben\_admin

**.env.development 开发环境适用**

# 是否开启mock数据，关闭时需要自行对接后台接口

VITE\_USE\_MOCK=true

# 资源公共路径,需要以 /开头和结尾

VITE\_PUBLIC\_PATH=/

# 是否删除Console.log

VITE\_DROP\_CONSOLE=false

# 本地开发代理，可以解决跨域及多地址代理

# 如果接口地址匹配到，则会转发到http://localhost:3000，防止本地出现跨域问题

# 可以有多个，注意多个不能换行，否则代理将会失效

VITE\_PROXY=[["/api","http://localhost:3000"],["api1","http://localhost:3001"],["/upload","http://localhost:3001/upload"]]

# 接口地址

# 如果没有跨域问题，直接在这里配置即可

VITE\_GLOB\_API\_URL=/api

# 文件上传接口 可选

VITE\_GLOB\_UPLOAD\_URL=/upload

# 接口地址前缀，有些系统所有接口地址都有前缀，可以在这里统一加，方便切换

VITE\_GLOB\_API\_URL\_PREFIX=

**.env.production生产环境适用**

# 是否开启mock

VITE\_USE\_MOCK=true

# 接口地址 可以由nginx做转发或者直接写实际地址

VITE\_GLOB\_API\_URL=/api

# 文件上传地址 可以由nginx做转发或者直接写实际地址

VITE\_GLOB\_UPLOAD\_URL=/upload

# 接口地址前缀，有些系统所有接口地址都有前缀，可以在这里统一加，方便切换

VITE\_GLOB\_API\_URL\_PREFIX=

# 是否删除Console.log

VITE\_DROP\_CONSOLE=true

# 资源公共路径,需要以 / 开头和结尾

VITE\_PUBLIC\_PATH=/

# 打包是否输出gz｜br文件

# 可选: gzip | brotli | none

# 也可以有多个, 例如 ‘gzip’|'brotli',这样会同时生成 .gz和.br文件

VITE\_BUILD\_COMPRESS = 'gzip'

# 打包是否压缩图片

VITE\_USE\_IMAGEMIN = false

# 打包是否开启pwa功能

VITE\_USE\_PWA = false

# 是否兼容旧版浏览器。开启后打包时间会慢一倍左右。会多打出旧浏览器兼容包,且会根据浏览器兼容性自动使用相应的版本

VITE\_LEGACY = false

**多语言配置**

用于配置多语言信息

在 src/settings/localeSetting.ts 内配置

export const LOCALE: { [key: string]: LocaleType } = {

ZH\_CN: 'zh\_CN',

EN\_US: 'en',

};

export const localeSetting: LocaleSetting = {

// 是否显示语言选择器

showPicker: true,

// 当前语言

locale: LOCALE.ZH\_CN,

// 默认语言

fallback: LOCALE.ZH\_CN,

// 允许的语言

availableLocales: [LOCALE.ZH\_CN, LOCALE.EN\_US],

};

// 语言列表

export const localeList: DropMenu[] = [

{

text: '简体中文',

event: LOCALE.ZH\_CN,

},

{

text: 'English',

event: LOCALE.EN\_US,

},

];

**路由**

项目路由配置存放于 src/router/routes

路由模块配置存放于src/router/routes/modules

**模块说明**

在 src/router/routes/modules 内的 .ts 文件会被视为一个路由模块。

一个路由模块包含以下结构:

import type { AppRouteModule } from '/@/router/types';

import { LAYOUT } from '/@/router/constant';

import { t } from '/@/hooks/web/useI18n';

// 运维管理菜单配置位置

const decision: AppRouteModule = {

path: '/decision',

name: 'decision',

component: LAYOUT,

redirect: '/decision/analysis',

meta: {

orderNo: 10,

icon: 'ion:grid-outline',

title: t('routes.decision.analysis'),

},

children: [

{

path: 'analysis',

name: 'analysis',

component: () => import('/@/views/decisionAnalysis/analysis.vue'),

meta: {

title: t('routes.decision.analysis'),

},

},

],

};

export default decision;

*注意事项:*

*整个项目所有路由 name 不能重复*

*除了 layout 对应的 path 前面需要加 /，其余子路由都不要以/开头*

**新增路由**

**如何新增一个路由模块**

1. **在 src/router/routes/modules 内新增模块目录和路由文件。**
2. **在 src/router/routes/index.ts 中 新增路由配置。**

**路由配置如下：**

const xxxNameModules = import.meta.glob('./modules/prodMonitoring/\*.ts', { eager: true }); // xxxx菜单路由配置导入

// xxxx菜单路由加入到路由集合中

Object.keys(xxxNameModules).forEach((key) => {

const mod = (xxxNameModules as Recordable)[key].default || {};

const modList = Array.isArray(mod) ? [...mod] : [mod];

routeModuleList.push(...modList);

});

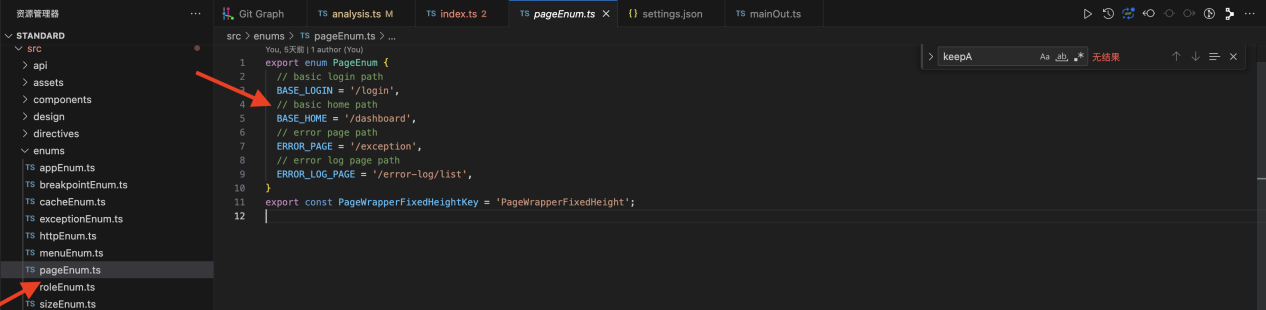
**如何更改首页路由**

**首页路由指的是应用程序中的默认路由，当不输入其他任何路由时，会自动重定向到该路由下，并且该路由在Tab上是固定的，即使设置affix: false也不允许关闭.**

**例：首页路由配置的是/dashboard/analysis，**

**那么当直接访问 http://localhost:3100/ 会自动跳转到http://localhost:3100/#/dashboard/analysis 上**

**可以将pageEnum.ts中的BASE\_HOME更改为需要你想设置的首页即可**

****

**菜单**

项目菜单配置存放于 src/router/menus下面，菜单必须和路由匹配才能显示。

**菜单模块**

**一个菜单文件会被当作一个模块**

***注意：children 的 path 字段不需要以/开头***

import type { MenuModule } from '/@/router/types';

import { t } from '/@/hooks/web/useI18n';

const menu: MenuModule = {

orderNo: 10,

menu: {

name: t('routes.dashboard.dashboard'),

path: '/dashboard',

children: [

{

path: 'analysis',

name: t('routes.dashboard.analysis'),},

{

path: 'workbench',

name: t('routes.dashboard.workbench'),

},

],

},

};

export default menu;

**新增菜单**

直接在 src/router/routes/modules 内新增一个模块文件即可。

不需要手动引入，放在src/router/routes/modules 内的文件会自动被加载。

**菜单排序**

在菜单模块内，设置 orderNo 变量，数值越大，排序越靠后

**构建&部署**

**构建**

**项目开发完成之后，执行以下命令进行构建**

**yarn build | npm run build | pnpm run build**

**构建打包成功之后，会在根目录生成 dist 文件夹，里面就是构建打包好的文件**

**旧版浏览器兼容**

**在 .env.production 内**

**设置 VITE\_LEGACY=true，即可打包出兼容旧版浏览器的代码**

**VITE\_LEGACY = true**

**本地预览**

发布之前可以在本地进行预览，有多种方式，这里介绍两种

不能直接打开构建后的 html 文件

**使用项目自定的命令进行预览**

# 先打包再进行预览

**yarn preview**

# 直接预览本地 dist 文件目录

**yarn preview:dist**

**本地服务器预览**

**# 1.全局安装live-server**

**yarn global add live-server**

**# 2. 进入打包的后目录**

**cd ./dist**

**# 本地预览，默认端口8080 live-server**

**live-server --port 9000**

**压缩**

**开启 gzip 压缩 开启 gzip，并配合 nginx 的 gzip\_static 功能**

**可以大大加快页面访问速度**

**设置 VITE\_BUILD\_COMPRESS='gzip' 即可在打包的同时生成 .gz 文件**

**部署**

前端路由与服务端的结合

项目前端路由使用的是 vue-router，所以你可以选择两种方式：**history** 和 **hash**。

* hash 默认会在 url 后面拼接 #
* history 则不会，不过 history 需要服务器配合

**可在 src/router/index.ts 内进行 mode 修改：**

import { createRouter, createWebHashHistory, createWebHistory } from 'vue-router';

createRouter({

history: createWebHashHistory(), // Hash模式

history: createWebHistory(), // History 模式

});

***注意：开启 history 模式需要服务器配置，更多的服务器配置详情可以看 [history-mode](https://router.vuejs.org/guide/essentials/history-mode.html" \l "html5-mode)***

**Nginx配置**

**部署到根目录**

server {

listen 80;

location / {

# 用于配合 History 使用

try\_files $uri $uri/ /index.html;

}

}

**部署到非根目录**

1. 首先需要在打包的时候更改配置

# 在.env.production内，配置子目录路径

VITE\_PUBLIC\_PATH = /sub/

1. 配置 nginx

server {

listen 80;

server\_name localhost;

location /sub/ {

# 这里是vue打包文件dist内的文件的存放路径

alias /srv/www/project/;

index index.html index.htm;

try\_files $uri $uri/ /sub/index.html;

}

}

**使用 nginx 处理跨域**

**使用 nginx 处理项目部署后的跨域问题**

1. **配置前端项目接口地址**

# 在.env.production内，配置接口地址

VITE\_GLOB\_API\_URL=/api

1. **在 nginx 配置请求转发到后台**

server {

listen 8080;

server\_name localhost;

#接口代理，用于解决跨域问题

location /api {

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

# 后台接口地址

proxy\_pass http://110.110.1.1:8080/api;

proxy\_redirect default;

add\_header Access-Control-Allow-Origin \*;

add\_header Access-Control-Allow-Headers X-Requested-With;

add\_header Access-Control-Allow-Methods GET,POST,OPTIONS;

}

}

**跨域处理**

**跨越产生原因**

**跨域产生的原因是由于前端地址与后台接口不是同源，从而导致 ajax 不能发送**

**非同源产生的问题**

**Cookie、LocalStorage 和 IndexDB 无法获取**

**DOM 无法获得**

**AJAX 请求不能发送**

**同源条件**

**协议，端口，主机 三者相同即为同源**

**反之，其中只要 某一个 不一样则为不同源**

**生产环境跨域**

**生产环境一般使用下面 2 种方式进行处理**

1. **后台开启 cors**
2. **使用 nginx 转发请求**

**后台开启 cors 不需要前端做任何改动**

**nginx 配置文件可以查看上面nginx 跨越配置**