技术栈

- 编程语言：TypeScript 4.x + JavaScript
- 构建工具：Vite 2.x
- 前端框架：Vue 3.x
- 路由工具：Vue Router 4.x
- 状态管理：Vuex 4.x
- UI 框架：Element Plus
- CSS 预编译：Stylus / Sass / Less
- HTTP 工具：Axios
- Git Hook 工具：husky + lint-staged
- 代码规范：EditorConfig + Prettier + ESLint + Airbnb JavaScript Style Guide
- 提交规范：Commitizen + Commitlint
- 单元测试：vue-test-utils + jest + vue-jest + ts-jest
- 自动部署：GitHub Actions

Vue 3 + Typescript + Vite

This template should help get you started developing with Vue 3 and Typescript in Vite.

- 使用 Vite 快速初始化项目雏形 使用 Vite 构建工具，需要 Node.js 版本 >= 12.0.0。

1. 使用 NPM：

```
npm init @vitejs/app
```

2. 使用 Yarn

```
yarn create @vitejs/app
```

- 安装依赖

```
npm install
```

- 启动项目

```
npm run dev
```

- Vite 配置文件 vite.config.ts ,vite 官网 vitejs.dev/config/

集成路由 vue router

- 安装支持 vue3 的 vue-router@4

```
npm i vue-router@4
```

- 路由配置文件 src/router/index.js

```
import {createRouter,createWebHashHistory,RouteRecordRaw} from 'vue-router'

const routes:Array<RouteRecordRaw> =[
    {
        path:'',
        name:'',
        component:()=>import('')
    }
]

const router = createRouter({
    history:createWebHashHistory(),
    routes
})

export default router
```

- 在 main.ts 文件中挂载路由配置

```
import {createApp} from 'vue'
import App from './App.vue'
import router from './router/index'
createApp(App).use(router).mount('#app)
```

集成状态管理工具 vuex

- 安装支持 vue3 的状态管理工具 vuex@next

```
npm i vuex@next
```

- 状态配置文件 src/store/index.ts

```

```

```
import {createStore} from 'vuex'
export default createStore({
    state(){

    },
    mutations:{

    },
    actions:{

    },
    getters:{

    }
})
```

- 在 main.ts 文件中挂载状态配置

```
import {}
```

集成 ui 框架 element plus

- 安装支持 vue3 的 ui 框架 element plus

```
npm i element-plus
```

- 在 main.ts 文件中挂载 element plus

```
import ElementPlus from 'element-plus'
import 'element-plus/lib/theme-chalk/index.css'
```

集成 HTTP 工具 axios

- 安装 axios

```
npm i axios
```

- 配置文件 src/utils/axios.ts

```
import Axios from 'axios'
import {ElMessage} from 'element-plus'
const baseUrl = '';
const axios = Axios.create({
    baseUrl,
    timeout:2000
})
axios.interceptors.request.use(
  (response) => {
    return response
  },
  (error) => {
    return Promise.reject(error)
  }
)
axios.interceptors.response.use(
    (response) => {
        return response
    },
    (error) => {
        return Promise.reject(error)
    }
)
export default axios
```

代码规范

- 集成 EditorConfig EditorConfig 为不同编辑器处理同一个项目的多个开发人员维护一致的编码风格，官网：editorconfig.org,vscode 需要下载插件 editorconfig for vs code

1. 项目根目录下增加.editorconfig 文件

- 集成 Prettier prettier 代码格式化工具，官网 prettier.io/,vscode 需要下载插件 Prettier - Code formatter

1. 安装 prettier

```
npm i prettier -D
```

2. 配置文件.prettierrc

```
{
  "useTabs": false,
  "tabWidth": 2,
  "printWidth": 100,
  "singleQuote": true,
  "trailingComma": "none",
  "bracketSpacing": true,
```

```
  "semi": false
}
```

3.命令格式化代码

```
npx prettier --write .
```

- 集成 ESLint ESLint 用于查找并报告代码中的问题的工具,VSCode 需要去插件市场下载插件 ESLint 。

1. 安装 ESLint

```
npm i eslint -D
```

2. 配置 ESLint

```
npx eslint --init
```

3. Airbnb: github.com/airbnb/javascript
4. 执行完命令自动生成 ESLint 配置文件 .eslintrc.js
5. 设置编辑器保存文件时自动执行 eslint --fix 命令进行代码风格修复,VSCode 在 settings.json 设置文件中，增加以下代码：

```
"editor.codeActionsOnSave": {
    "source.fixAll.eslint": true
 }
```

- 解决 Prettier 和 ESLint 的冲突 解决两者冲突问题，需要用到 eslint-plugin-prettier 和 eslint-config-prettier

1. eslint-plugin-prettier 将 Prettier 的规则设置到 ESLint 的规则中。
2. eslint-config-prettier 关闭 ESLint 中与 Prettier 中会发生冲突的规则。 最后形成优先级：Prettier 配置规则 > ESLint 配置规则。
3. 安装插件

```
npm i eslint-plugin-prettier eslint-config-prettier -D
```

4. 在 .eslintrc.js 添加 prettier 插件

```
module.exports = {
  ...
  extends: [
    'plugin:vue/essential',
    'airbnb-base',
    'plugin:prettier/recommended' // 添加 prettier 插件
  ],
  ...
}
```

这样，我们在执行 eslint --fix 命令时，ESLint 就会按照 Prettier 的配置规则来格式化代码，轻松解决二者冲突问题

提交规范

commit message 由 Header 组成。

- Header Header 部分包括三个字段 type（必需）和 subject（必需）。：

1. type

值	描述
feat	新增一个功能
fix	修复一个 Bug
docs	文档变更
style	代码格式（不影响功能，例如空格、分号等格式修正）
refactor	代码重构
perf	改善性能
test	测试
build	变更项目构建或外部依赖（例如 scopes: webpack、gulp、npm 等）
ci	更改持续集成软件的配置文件和 package 中的 scripts 命令
chore	变更构建流程或辅助工具
revert	代码回退

2. subject subject 是本次 commit 的简洁描述，长度约定在 50 个字符以内，通常遵循以下几个规范：

> 用动词开头，第一人称现在时表述，例如：change 代替 changed 或 changes 第一个字母小写 结尾不加句号（.）

单元测试

使用 Vue 官方提供的 vue-test-utils 和社区流行的测试工具 jest 来进行 Vue 组件的单元测试

- 安装

```
npm i @vue/test-utils@next jest vue-jest@next ts-jest -D
```

- 在项目根目录下新建 jest.config.js 文件：

```
module.exports = {
  moduleFileExtensions: ['vue', 'js', 'ts'],
  preset: 'ts-jest',
  testEnvironment: 'jsdom',
  transform: {
    '^.+\\.vue$': 'vue-jest', // vue 文件用 vue-jest 转换
    '^.+\\.ts$': 'ts-jest' // ts 文件用 ts-jest 转换
  },
  // 匹配 __tests__ 目录下的 .js/.ts 文件 或其他目录下的 xx.test.js/ts xx.spec.js/ts
  testRegex: '(/__tests__/.*|(\\.|/)(test|spec))\\.(ts)$'
}
```

- 创建单元测试文件 在 src/tests/xx.test.ts 或者 src/tests/xx.spec.ts

```
import { mount } from '@vue/test-utils'
import Test from '../src/views/Test.vue'

test('Test.vue', async () => {
  const wrapper = mount(Test)
  expect(wrapper.html()).toContain('Unit Test Page')
  expect(wrapper.html()).toContain('count is: 0')
  await wrapper.find('button').trigger('click')
  expect(wrapper.html()).toContain('count is: 1')
})
```

1. 使用 VSCode / WebStrom / IDEA 等编辑器时，在单元测试文件中，IDE 会提示某些方法不存在（如 test、describe、it、expect 等），安装 @types/jest 即可解决。

```
npm i @types/jest -D
```

2. TypeScript 的编译器也会提示 jest 的方法和类型找不到，我们还需把 @types/jest 添加根目录下的 ts.config.json（TypeScript 配置文件）中：

```
{
  "compilerOptions": {
    ...
```

```
    "types": ["vite/client", "jest"]
  },
}
```

3. 没通过 ESLint 规则检验。因此，我们还需要在 ESLint 中增加 eslint-plugin-jest 插件来解除对 jest 的校验。

> 安装 eslint-plugin-jest

```
npm i eslint-plugin-jest -D
```

> 添加 eslint-plugin-jest 到 ESLint 配置文件 .eslintrc.js 中

```
module.exports = {
  ...
  extends: [
    ...
    'plugin:jest/recommended'
  ],
  ...
}
```

- 执行单元测试 在根目录下 package.json 文件的 scripts 中，添加一条单元测试命令： "test": "jest"。

```
"script":{
    "test":"jest"
}
```

执行命令 npm run test 即可进行单元测试，jest 会根据 jest.config.js 配置文件去查找 **tests** 目录下的 .ts 文件或其他任意目录下的 .spec.ts 和 .test.ts 文件，然后执行单元测试方法。

自动部署

GitHub Actions 是 GitHub 的持续集成服务，持续集成由很多操作组成，比如抓取代码、运行测试、登录远程服务器、发布到第三方服务等等，GitHub 把这些操作称为 actions。

- 配置 GitHub Actions

1. master 分支存储项目源代码
2. gh-pages 分支存储打包后的静态文件

> gh-pages 分支，是 GitHub Pages 服务的固定分支，可以通过 HTTP 的方式访问到这个分支的静态文件资源。

- 创建 GitHub Token

1. 创建一个有 repo 和 workflow 权限的 GitHub Token
2. 新生成的 Token 只会显示一次，保存起来，后面要用到。如有遗失，重新生成即可。

- 在仓库中添加 secret

1. 仓库 -> settings -> Secrets -> New repository secret

- 创建 Actions 配置文件

1. 在项目根目录下创建 .github 目录。
2. 在 .github 目录下创建 workflows 目录。
3. 在 workflows 目录下创建 deploy.yml 文件。

```
name: deploy

on:
  push:
    branches: [master] # master 分支有 push 时触发

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Setup Node.js v14.x
        uses: actions/setup-node@v1
        with:
          node-version: '14.x'

      - name: Install
        run: npm install # 安装依赖

      - name: Build
        run: npm run build # 打包

      - name: Deploy
        uses: peaceiris/actions-gh-pages@v3 # 使用部署到 GitHub pages 的 action
        with:
          publish_dir: ./dist # 部署打包后的 dist 目录
          github_token: ${{ secrets.VUE3_DEPLOY }} # secret 名
          user_name: ${{ secrets.MY_USER_NAME }}
          user_email: ${{ secrets.MY_USER_EMAIL }}
          commit_message: Update Vite2.x + Vue3.x + TypeScript Starter # 部署时的
git 提交信息，自由填写
```

- 当有新提交的代码 push 到 GitHub 仓库时，就会触发 GitHub Actions，在 GitHub 服务器上执行 Action 配置文件里面的命令，例如：安装依赖、项目打包等，然后将打包好的静态文件部署到 GitHub Pages 上，最后，我们就能通过域名访问了。

> 通过域名 vite-vue3-starter.xpoet.cn/ 访问本项目

Recommended IDE Setup

VSCode + Vetur. Make sure to enable `vetur.experimental.templateInterpolationService` in settings!

If Using `<script setup>`

`<script setup>` is a feature that is currently in RFC stage. To get proper IDE support for the syntax, use Volar instead of Vetur (and disable Vetur).

Type Support For `.vue` Imports in TS

Since TypeScript cannot handle type information for `.vue` imports, they are shimmed to be a generic Vue component type by default. In most cases this is fine if you don't really care about component prop types outside of templates. However, if you wish to get actual prop types in `.vue` imports (for example to get props validation when using manual `h(...)` calls), you can use the following:

If Using Volar

Run `Volar: Switch TS Plugin on/off` from VSCode command palette.

If Using Vetur

1. Install and add `@vuedx/typescript-plugin-vue` to the plugins section in `tsconfig.json`
2. Delete `src/shims-vue.d.ts` as it is no longer needed to provide module info to Typescript
3. Open `src/main.ts` in VSCode
4. Open the VSCode command palette
5. Search and run "Select TypeScript version" -> "Use workspace version"