vue工具库搭建

技术栈

vue3 + vite + vuepress (monorepo)

内容：公共的组件、方法、hooks

vmono-seed

- as-vant-kit

使用手册

1. 字符串全局替换 vmono-seed -> 你的项目名(将作为所有子包的名称前缀)
2. 删除 README.pdf、更新 README.md 内容

整体框架搭建

项目初始化

1. 创建项目目录 vmono-seed
2. 运行 pnpm init，编辑部分字段

代码块

```
{
  "name": "vmono-seed",
  "version": "0.0.0",
  "type": "module",
  "author": "astfn",
  "description": "A project that includes common components from the H5 project (dependent on vant) and some utility functions",
  "scripts": {},
  "keywords": [],
  "license": "ISC",
}
```

3. 创建 monorepo 工作区配置文件：`pnpm-workspace.yaml` 其中配置的包目录，后续可以在整个项目中共享，实时引入最新代码。

```
代码块
1  packages:
2    - 'internal/*'
3    - 'packages/*'
4    - 'docs'
```

   - internal 用于放一些公共的内部配置
     - eslint-config
     - ts-config
   - packages 就是维护的工具包
     - 目前统一放在 vant-kit 目录中，后续可以将工具函数单独抽出去。
   - docs
     - 用于文档产出，使用 vuepress 构建
4. 创建对应的包目录

```
代码块
1  vmono-seed/
2  ├── internal/
3  │   ├── eslint-config/          # 通用的 eslint 配置
4  │   ├── ts-config/              # 通用的 ts 规则配置
5  │
6  ├── packages/
7  │   ├── vant-kit/            # Vue 工具库（组件 + Hook + 方法）
8  │
9  ├── docs/                       # VuePress 文档站点
10 │
11 ├── pnpm-workspace.yaml         # pnpm Monorepo 配置
12 ├── package.json
13 └── README.md
```

5. 进入 vant-kit 工具包，初始化 vite 项目

```
代码块
1  cd packages/vue-utils
2  pnpm init
```

修改部分字段

代码块

```
{
  "name": "@vmono-seed/vant-kit",
  "version": "0.0.0",
  "type": "module",
  "description": "A project that includes common components from the H5 project (dependent on vant) and some utility functions",
  "main": "index.ts",
  "scripts": {},
  "keywords": [],
  "author": "astfn",
  "license": "ISC",
}

```

安装依赖

代码块

```
pnpm add -D typescript vite @vitejs/plugin-vue vue vue-tsc
```

创建 `vite.config.ts` （顺便创建下入口文件 index.ts 做预留）

代码块

```
import { defineConfig } from 'vite';
import vue from '@vitejs/plugin-vue';

export default defineConfig({
  plugins: [vue()],
  build: {
    lib: {
      entry: './src/index.ts',
      name: 'VueUtils',
      fileName: (format) => `vue-utils.${format}.js`,
    },
    rollupOptions: {
      external: ['vue'],
      output: {
        globals: {
          vue: 'Vue',
        },
      },
```

```
19        },
20      },
21    });
```

6. Placeholder

配置 tsconfig

配置在哪

可以配置在全局，也可以配置在各个子包中。如果都配置了，则以当前包的为准。

由于在项目根层级中，目前不需要编写额外的 ts 代码，所以目前只在子包 (vant-kit) 中配置即可

配置复用

如果后续新增其它工具包 (例如把常用的工具函数、hooks单独抽成一个包)，那这些 tsconfig 都是通用的，保持风格一致。

因此有必要单独抽离一下，然后在子包中引入这些配置。

在抽离某些配置之前，要先看看这些配置是否支持插拔式引入。

tsconfig.json 文件是支持 extends 配置项的，可以直接引入外部包，继承其配置。因此我们的想法才可以进行实践。

配置抽离

在 internal/ts-config 中进行工具包的初始化

代码块

```
pnpm init
```

并修改 package.json 部分字段

代码块

```
{
  "name": "@vmono-seed/ts-config",
  "version": "0.0.0",
  "author": "astfn",
  "private": true,
  "files": [
    "tsconfig.json",
    "tsconfig.app.json",
```

```
    "tsconfig.node.json"
  ]
}
```

新建 tsconfig.json 文件，将 tsconfig.app.json、tsconfig.node.json 再抽成单独的文件配置

代码块

```
{
  "files": [],
  "references": [{ "path": "./tsconfig.app.json" }, { "path":
"./tsconfig.node.json" }]
}

```

新建 tsconfig.app.json 文件

代码块

```
{
  "compilerOptions": {
    "target": "ES2020",
    "noImplicitAny": false,
    "useDefineForClassFields": true,
    "module": "ESNext",
    "lib": ["ES2020", "DOM", "DOM.Iterable"],
    "skipLibCheck": true,
    "baseUrl": ".",
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "isolatedModules": true,
    "moduleDetection": "force",
    "noEmit": true,
    "jsx": "preserve",
    "sourceMap": true,
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true
  },
  "include": ["src/**/*.ts", "src/**/*.tsx", "src/**/*.vue", "*.d.ts"]
}

```

新建 tsconfig.node.json 文件

代码块

```
{
  "compilerOptions": {
    "target": "ES2022",
    "lib": ["ES2023"],
    "module": "ESNext",
    "skipLibCheck": true,
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "isolatedModules": true,
    "moduleDetection": "force",
    "noEmit": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true
  },
  "include": ["vite.config.ts"]
}

```

使用配置

进入 vant-kit 包，先把抽离的 tsconfig 依赖添加到 package.json 中，并执行 pnpm i 进行下载

代码块

```
{
  "name": "@vmono-seed/vant-kit",
   ……,
  "devDependencies": {
    "@vmono-seed/ts-config": "workspace:*",
    ……
  }
}

```

新建 tsconfig.json 文件，同理也将 tsconfig.app.json、tsconfig.node.json 再抽成单独的文件配置

- 复用 @vmono-seed/ts-config 中 tsconfig.json 配置

代码块

```
{
  "extends": "@vmono-seed/ts-config/tsconfig.json",
  "files": [],
```

```
4     "references": [{ "path": "./tsconfig.app.json" }, { "path":
    "./tsconfig.node.json" }]
5   }
6
```

新建 tsconfig.app.json 文件，复用 @vmono-seed/ts-config 中 tsconfig.app.json 配置的同时，再针对该包，新增一些配置

代码块

```
1   {
2     "extends": "@vmono-seed/ts-config/tsconfig.app.json",
3     "compilerOptions": {
4       "baseUrl": ".",
5       "paths": {
6         "@/*": ["src/*"]
7       }
8     },
9     "include": ["src/**/*.ts", "src/**/*.tsx", "src/**/*.vue", "*.d.ts"]
10  }
```

新建 tsconfig.node.json 文件，复用 @vmono-seed/ts-config 中 tsconfig.node.json 配置的同时，再针对该包，新增一些配置

代码块

```
1   {
2     "extends": "@vmono-seed/ts-config/tsconfig.node.json",
3     "include": ["vite.config.ts"]
4   }
```

到这里就把 vant-kit 包的 tsconfig 配置完了，如果后续新增其它子包，可遵循相同的配置过程。

配置 eslint

配置在哪

可以配置在全局，也可以配置在各个子包中。如果都配置了，则以当前包的为准。

由于我们是在一个大的 monorepo 项目中，eslint 风格对于所有包来说应该是要一致的，不像 tsconfig 那样需要针对不同包需要个性化配置。

因此配置在项目根目录中，再加上 husky 其实就已经可以实现整个项目在代码提交时遵循同一套 lint 规则进行代码校验。

但在进行实际开发过程中，你会发现，尽管在根项目中配置了 eslint，但 VSCode 的 ESLint 插件默认只会查找当前打开文件所在目录下的 eslint 配置文件，如果找不到，就不会激活 ESLint 校验。

而你在子包中没有配置 `eslint.config.js`，所以 VSCode 不知道要使用根项目的 ESLint 配置，因此在进行子包开发时，如果违反了 eslint 规则，但是编辑器不会爆红，只会在提交时，通过控制台看到错误。

因此为了达到最佳的开发体验，我们还是要在各个子包中配置 `eslint.config.js` 。

配置复用

同理，在抽离配置之前，要先看看这些配置是否支持插拔式引入。

当前最新版本 eslint9.x 统一改为了使用 js 配置，配置风格也变成了扁平化的形式。既然是 js 配置文件，意味着我们可以将配置抽离为方法，供外部使用，并通过函数传参实现更灵活的配置。

上面我们已经说过了，为了达到最佳开发体验，项目根目录路和各个子包都要配置 eslint，而现在 eslint 配置支持可插拔形式，就大大降低了维护成本，并且还能在复用全局配置的同时，对子包的 lint 规则进行定制化，虽然目前没有定制需求。

配置抽离

在 internal/eslint-config 中进行工具包的初始化

代码块

```
pnpm init
```

并修改 package.json 部分字段

代码块

```
{
  "name": "@vmono-seed/eslint-config",
  "version": "0.0.0",
  "author": "astfn",
  "type": "module",
  "private": true,
  "main": "index.js",
  "files": [
    "index.js"
  ],
```

```
11  }
```

安装依赖

代码块

```
1  pnpm add @eslint/js eslint-config-prettier eslint-plugin-vue globals
   typescript-eslint vue-eslint-parser
```

新建 index.js 文件

- 将 ts 相关的 lint 规则抽离到 ./tsLintConfig.js 中
- 将 vue 相关的 lint 规则抽离到 ./vueLintConfig.js 中

代码块

```
1  import globals from 'globals';
2  import eslint from '@eslint/js';
3  import eslintConfigPrettier from 'eslint-config-prettier';
4  import { genTsLintConfig } from './tsLintConfig.js';
5  import { genVueLintConfigArr } from './vueLintConfig.js';
6
7  export * from './tsLintConfig.js';
8  export * from './vueLintConfig.js';
9
10 export default [
11   {
12     ignores: ['.history/**', '.husky/**', '.vscode/**', 'coverage/**',
   'lib/**', 'public/**', 'node_modules/**'],
13   },
14   { languageOptions: { globals: { ...globals.browser, ...globals.node } } },
15   eslint.configs.recommended,
16   ...genTsLintConfig(),
17   ...genVueLintConfigArr(),
18   eslintConfigPrettier,
19 ];
```

新建 tsLintConfig.js

代码块

```
1  import tseslint from 'typescript-eslint';
2
3  export const genTsNormalRules = ({ customRules }) => {
4    return {
```

```
    'no-sparse-arrays': 'off',
    '@typescript-eslint/no-explicit-any': 'off',
    '@typescript-eslint/no-unused-expressions': 'off',
    '@typescript-eslint/no-unsafe-function-type': 'off',
    '@typescript-eslint/consistent-type-imports': 'off',
    'no-console': ['error', { allow: ['warn', 'error'] }],
    '@typescript-eslint/no-unused-vars': [
      'error',
      {
        argsIgnorePattern: '^_',
        varsIgnorePattern: '^_',
      },
    ],
    ...(customRules ?? {}),
  };
};

export const genTsLintConfig = ({ customRules } = {}) => {
  return [
    ...tseslint.configs.recommended,
    {
      files: ['**/*.ts'],
      plugins: {
        '@typescript-eslint': tseslint.plugin,
      },
      languageOptions: {
        parser: tseslint.parser,
      },
      rules: genTsNormalRules({ customRules }),
    },
  ];
};
```

新建 vueLintConfig.js

代码块

```
import vueParser from 'vue-eslint-parser';
import { genTsNormalRules } from './tsLintConfig.js';
import pluginVue from 'eslint-plugin-vue';
import tseslint from 'typescript-eslint';

export const genVueNormalRules = ({ customRules }) => {
  return {
    'vue/no-v-html': 'off',
    'vue/attributes-order': 'off',
```

```
      'vue/require-emit-validator': 'warn',
      'vue/multi-word-component-names': 'off',
      'vue/no-setup-props-destructure': 'off',
      'vue/v-on-event-hyphenation': 'off',
      'vue/no-mutating-props': 'off',
      'vue/html-self-closing': [
        'error',
        {
          html: {
            void: 'always',
            normal: 'never',
            component: 'always',
          },
          svg: 'always',
          math: 'always',
        },
      ],
      ...(customRules ?? {}),
    };
};

export const genVueLintConfigArr = ({ customRules } = {}) => {
  return [
    ...pluginVue.configs['flat/recommended'],
    {
      files: ['**/*.vue'],
      plugins: {
        '@typescript-eslint': tseslint.plugin,
      },
      languageOptions: {
        parser: vueParser, // 使用vue解析器，这个可以识别vue文件
        parserOptions: {
          parser: tseslint.parser, // 在vue文件上使用ts解析器
          sourceType: 'module',
        },
      },
      rules: {
        ...genTsNormalRules({}),
        ...genVueNormalRules({ customRules }),
      },
    },
  ];
};
```

使用配置

分别在全局、子包中执行以下步骤

1. 在 package.json 的 devDependencies 中添加 `"@vmono-seed/eslint-config": "workspace:*"` ，并执行 `pnpm i` 下载
2. 新建 eslint.config.js 文件，引入公共配置，并复用

```
代码块
1  import eslintConfig from '@vmono-seed/eslint-config';
2
3  export default eslintConfig;
```

配置 husky & prettier

这两个配置都是全局性质的，只需要在根目录进行统一配置即可

在根目录安装 husky 和 prettier

```
代码块
1  pnpm add -D -w husky prettier
```

prettier

如果不同子包需要定制 prettier，也在子包中配置 prettier 即可，会以子包中的为准。

配置 .prettierrc 文件

```
代码块
1  {
2    "semi": true,
3    "singleQuote": true,
4    "trailingComma": "es5",
5    "printWidth": 120,
6    "tabWidth": 2,
7    "useTabs": false,
8    "bracketSpacing": true,
9    "jsxBracketSameLine": true,
10   "arrowParens": "always",
```

```
  "endOfLine": "lf"
}
```

配置 .prettierignore

代码块

```
.history
.husky
.vscode
coverage
dist
node_modules
public
```

husky

配置 .lintstagedrc 文件

代码块

```
{
  "*.{js?(x),ts?(x),vue,json}": ["npm run format", "npm run lint"],
  "*.{html,css,less,scss}": ["npm run format"]
}
```

更新根目录 package.json 的脚本指令

代码块

```
{
  "name": "vmono-seed",
   ……
  "scripts": {
    "prepare": "husky install",
    "format": "prettier --config .prettierrc . --write",
    "lint": "eslint --config eslint.config.js"
  }
}

```

运行 pnpm i ，会自动生成 .husky 目录，在其中创建 pre-commit (做提交前的代码校验逻辑)、commit-msg (做提交时 message 的校验逻辑)

pre-commit 文件

代码块

```
echo "🚄 lint-staged 正在执行 🚄"
npx lint-staged --quiet
echo "🎉 lint-staged 检测完毕,通过校验 🎉"
```

commit-msg 文件

代码块

```
echo "commit msg 验证中🕑"
npx --no-install commitlint --edit "$1"
echo "commit msg 验证通过🌞"
```

安装 commit-msg 所需依赖

- @commitlint/config-conventional 是比较通用的校验规则包

代码块

```
pnpm add -D -w  @commitlint/cli @commitlint/config-conventional
```

在根目录创建 .commitlintrc 配置文件，继承通用的规则校验包

代码块

```
{
  "extends": [
    "@commitlint/config-conventional"
  ]
}
```

其它基础配置

.npmrc

代码块

```
registry=https://registry.npmmirror.com
```

.gitignore

代码块

```
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
pnpm-debug.log*
lerna-debug.log*

node_modules
dist
dist-ssr
*.local

# Editor directories and files
.vscode/*
!.vscode/extensions.json
.idea
.DS_Store
*.suo
*.ntvs*
*.njsproj
*.sln
*.sw?
```

.gitattributes

代码块

```
# 告诉 Git：这些是文本文件，并统一使用 LF 换行符
*.txt text eol=lf
*.md text eol=lf
*.js text eol=lf
*.ts text eol=lf
*.tsx text eol=lf
*.vue text eol=lf
*.json text eol=lf
*.json5 text eol=lf
*.mjs text eol=lf
```

```
*.cjs text eol=lf
*.css text eol=lf
*.scss text eol=lf
*.html text eol=lf
*.yaml text eol=lf
*.yml text eol=lf
*.toml text eol=lf
*.lock text eol=lf
*.log text eol=lf
*.env text eol=lf
*.prettierignore text eol=lf
*.gitignore text eol=lf
*.editorconfig text eol=lf
```