Vue3+TS组件库搭建

一.搭建`monorepo`环境

> 使用`pnpm`安装包速度快，磁盘空间利用率高效，使用`pnpm`可以快速建立`monorepo`，so ~ 这里我们使用`pnpm workspace`来实现`monorepo`

```
npm install pnpm -g # 全局安装pnpm
pnpm init  # 初始化package.json配置文件 私有库
pnpm install vue typescript -D # 全局下添加依赖
```

> 使用`pnpm`必须要建立`.npmrc`文件，`shamefully-hoist = true`，否则安装的模块无法放置到`node_modules`目录下

```
{
  "compilerOptions": {
    "module": "ESNext", // 打包模块类型ESNext
    "declaration": false, // 默认不要声明文件
    "noImplicitAny": false, // 支持类型不标注可以默认any
    "removeComments": true, // 删除注释
    "moduleResolution": "node", // 按照node模块来解析
    "esModuleInterop": true, // 支持es6,commonjs模块
    "jsx": "preserve", // jsx 不转
    "noLib": false, // 不处理类库
    "target": "es6", // 遵循es6版本
    "sourceMap": true,
    "lib": [ // 编译时用的库
      "ESNext",
      "DOM"
    ],
    "allowSyntheticDefaultImports": true, // 允许没有导出的模块中导入
    "experimentalDecorators": true, // 装饰器语法
    "forceConsistentCasingInFileNames": true, // 强制区分大小写
    "resolveJsonModule": true, // 解析json模块
    "strict": true, // 是否启动严格模式
    "skipLibCheck": true // 跳过类库检测
  },
  "exclude": [ // 排除掉哪些类库
    "node_modules",
    "**/__tests__",
    "dist/**"
  ]
}
```

> 在项目根目录下建立`pnpm-workspace.yaml`配置文件

```yaml
packages:
  - 'packages/**' # 存放编写组件的
  - docs # 存放文档的
  - play # 测试组件的
```

二.创建组件测试环境

```
pnpm create vite play  --template vue-ts
```

> 提供typescript声明文件 `typings/vue-shim.d.ts`

```ts
declare module '*.vue' {
    import type { DefineComponent } from 'vue'
    const component: DefineComponent<{}, {},
any>
    export default component
}
```

三.编写测试组件

```
packages
    ├─components   # 存放所有的组件
    ├─utils        # 存放工具方法
    └─theme-chalk # 存放对应的样式
```

```
cd components && pnpm init # @zi-shui/components
cd utils && pnpm init # @zi-shui/utils
cd theme-chalk && pnpm init # @zi-shui/theme-
chalk
```

> 在根模块下添加依赖

```
pnpm i @zi-shui/components -w
pnpm i @zi-shui/utils -w
pnpm i @zi-shui/theme-chalk -w
```

1).实现BEM规范

> **utils/create.ts**

```
const _bem = (prefixedName, blockSuffix,
element, modifier) => {
  if (blockSuffix) {
    prefixedName += `-${blockSuffix}`
  }
  if (element) {
    prefixedName += `__${element}`
  }
  if (modifier) {
    prefixedName += `--${modifier}`
  }
  return prefixedName
```

```
}
function createBEM(prefixedName: string) {
  const b = (blockSuffix = '') =>
_bem(prefixedName, blockSuffix, '', '')
  const e = (element = '') =>
    element ? _bem(prefixedName, '', element,
'') : ''
  const m = (modifier = '') =>
    modifier ? _bem(prefixedName, '', '',
modifier) : ''
  const be = (blockSuffix = '', element = '') =>
    blockSuffix && element ? _bem(prefixedName,
blockSuffix, element, '') : ''
  const em = (element, modifier) =>
    element && modifier ? _bem(prefixedName, '',
element, modifier) : ''
  const bm = (blockSuffix, modifier) =>
    blockSuffix && modifier ? _bem(prefixedName,
blockSuffix, '', modifier) : ''
  const bem = (blockSuffix, element, modifier)
=>
    blockSuffix && element && modifier
      ? _bem(prefixedName, blockSuffix, element,
modifier)
      : ''
  const is = (name, state) => (state ? `is-
${name}` : '')
```

```
  return {
    b,
    e,
    m,
    be,
    em,
    bm,
    bem,
    is
  }
}
export function createNamespace(name: string) {
  const prefixedName = `z-${name}`
  return createBEM(prefixedName)
}
const bem = createNamespace('button');
// z-button
// z-button-box
// z-button__element
// z-button--disabled
console.log(bem.b())
console.log(bem.b('box'))
console.log(bem.e('element'));
console.log(bem.m('disabled'))
console.log(bem.is('checked'))
console.log(bem.bem('box', 'element',
'disabled'))
```

2).实现Icon组件

> icon组件编写 `components/icon/src/icon.ts`

```
import { ExtractPropTypes, PropType } from 'vue'

export const iconProps = {
  size: [Number, String] as PropType<number |
string>,
  color: String
} as const
export type IconProps = ExtractPropTypes<typeof
iconProps>
```

> icon组件编写 `components/icon/src/icon.vue`

```
pnpm i unplugin-vue-define-options -D # 识别name
属性
```

```
<template>
    <i :class="bem.b()" :style="style">
        <slot></slot>
    </i>
</template>

<script lang="ts" setup>
```

```
import { createNamespace } from '@zi-shui/utils/create';
import { computed, CSSProperties } from 'vue';
import { iconProps } from './icon';
const bem = createNamespace('icon')
defineOptions({
    name:'ZIcon'
})
const props = defineProps(iconProps);

const style = computed<CSSProperties>(() => {
    if (!props.size && !props.color) {
        return {}
    }
    return {
        ...(props.size ? { 'font-size': props.size + 'px' } : {}),
        ...(props.color ? { 'color': props.color } : {})
    }
});
</script>
```

3).导出Icon组件

> 每个组件都需要增添install方法，我们在`utils`中增添`withInstall.ts`

```
import { Plugin } from "vue";
export type SFCWithInstall<T> = T & Plugin; // 添加插件类型
export function withInstall<T>(comp: T) {
    (comp as SFCWithInstall<T>).install = function (app) {
        const { name } = comp as unknown as { name: string }
        app.component(name, comp);// 注册全局组件
    }
    return comp as SFCWithInstall<T>;
}
```

这样我们可以在`components`下使用`utils`模块了。

```
import { withInstall } from '@zi-shui/utils/withInstall';
import _Icon from './src/icon.vue';
const Icon = withInstall(_Icon); // 生成带有install方法的组件
export default Icon; // 导出Icon组件
export type { IconProps } from './src/icon'
declare module 'vue' {
    export interface GlobalComponents {
        ZIcon: typeof Icon
    }
}
```

4).展示组件

```
import { createApp } from 'vue'
import App from './App.vue'
import Icon from '@zi-shui/components/icon';
const app = createApp(App);
app.use(Icon);
app.mount('#app')
```

5).`svg`图标

> 安装`@vicons/ionicons5`作为图标库

```
<script setup lang="ts">
import { CashOutline } from '@vicons/ionicons5'
</script>
<template>
  <ZIcon color="red" size="40">
    <CashOutline></CashOutline>
  </ZIcon>
</template>
```

四.scss编写

1).结构目录

```
theme-chalk
|   └─src
|       └─mixins
|         └─config.scss # BEM规范命名
```

2).sass配置

> `mixins/config.scss`

```
$namespace: 'z';
$element-separator: '__';
$modifier-separator:'--';
$state-prefix:'is-';
```

> `mixins/mixins.scss`

```
@use 'config' as *;
@forward 'config';
// .z-button{}
@mixin b($block) {
    $B: $namespace+'-'+$block;
    .#{$B}{
        @content;
    }
}
```

```scss
// .z-button.is-xxx
@mixin when($state) {
    @at-root {
        &.#{$state-prefix + $state} {
            @content;
        }
    }
}
// &--primary => .z-button--primary
@mixin m($modifier) {
    @at-root {
        #{&+$modifier-separator+$modifier} {
            @content;
        }
    }
}
// &__header  => .z-button__header
@mixin e($element) {
    @at-root {
        #{&+$element-separator+$element} {
            @content;
        }
    }
}
```

> `icon.scss`

```
@use 'mixins/mixins' as *;
@include b('icon') {
  height: 1em;
  width: 1em;
  line-height: 1em;
  display: inline-block;
  vertical-align: middle;
  svg {
    height: 1em;
    width: 1em;
  }
}
```

> `index.scss`

```
@use './icon.scss';
```

> 最后在`main.ts`中引入此`scss`即可，这里编译`sass`还需要安装`sass`

五.`Eslint`配置

> 开发项目需要安装 `vscode` 插件 `volar`

```
npx eslint --init
```

校验语法并提示错误行数

```
? How would you like to use ESLint? ...
  To check syntax only
> To check syntax and find problems
  To check syntax, find problems, and enforce code style
```

采用`js-module`

```
? What type of modules does your project use? ...
> JavaScript modules (import/export)
  CommonJS (require/exports)
  None of these
```

项目采用`vue`语法

```
? Which framework does your project use? ...
  React
> Vue.js
  None of these
```

手动安装插件

```
pnpm i eslint-plugin-vue@latest @typescript-eslint/eslint-plugin@latest @typescript-eslint/parser@latest eslint@latest -D -w
```

> 支持 vue 中 ts `eslint`配置

```
pnpm i @vue/eslint-config-typescript -D -w
```

`.eslintrc配置`

```
module.exports = {
    "env": {
        "browser": true,
        "es2021": true,
        "node": true
    },
    "extends": [
        "eslint:recommended",
        "plugin:vue/vue3-recommended", // vue3解析 https://eslint.vuejs.org/
        "plugin:@typescript-eslint/recommended",
        "@vue/typescript/recommended"
    ],
    "parserOptions": {
        "ecmaVersion": "latest",
        "parser": "@typescript-eslint/parser",
        "sourceType": "module"
    },
    "rules": {
        "vue/html-self-closing": "off",
        "vue/max-attributes-per-line": "off",
        "vue/singleline-html-element-content-newline": "off",
            "vue/multi-word-component-names": "off",
        },
        globals: {
            defineProps: "readonly"
        }
}
```

`.eslintignore`配置

```
node_modules
dist
*.css
*.jpg
*.jpeg
*.png
*.gif
*.d.ts
```

> 最终安装`vscode`中`eslint`插件：`eslint`只是检测代码规范

六.Prettier配置

`.prettierrc.js`

```
module.exports = {
    singleQuote: true, //使用单引号
    semi: false, // 使用分号
    trailingComma: "none", // 末尾逗号
    arrowParens: "avoid", //  箭头函数括号
    endOfLine: "auto" // 结尾换行自动
}
```

`.prettierignore`

```
node_modules
dist
```

> 最终安装`vscode`中`Prettier`插件：`prettier`只是用来格式化代码
>
> 这里需要配置`Format On Save` 为启用，保存时自动格式化 `Default Formatter`选择`Prettier - Code formatter`

七.编辑器配置

`.editorconfig`

```
root = true

[*]
charset = utf-8
indent_style = space
indent_size = 2
end_of_line = lf
```

> 最终安装`vscode`中`EditorConfig for VS Code`插件

八.lint-staged配置

1).提交检测代码

```
git init
pnpm install mrm husky lint-staged -w -D
npx mrm lint-staged
```

2).代码提交检测

```
pnpm install @commitlint/cli @commitlint/config-conventional -D -w
npx husky add .husky/commit-msg "npx --no-install commitlint --edit $1"
```

> `commitlint.config.js配置`

```
module.exports = {
  extends: ["@commitlint/config-conventional"],
  rules: {
    "type-enum": [
      2,
      "always",
      [
        "build",      // 编译相关的修改，例如发布版本、对项目构建或者依赖的改动
        "chore",      // 其他修改，比如改变构建流程、或者增加依赖库、工具等
        "ci",         // 持续集成修改
        "docs",       // 文档修改
        "feature",    //新特性、新功能
        "fix",        // 修改 bug
        "perf",       // 优化相关，比如提升性能、体验
        "refactor",   // 代码重构
        "revert",     // 回滚到上一个版本
        "style",      // 代码格式修改
        "test"        // 测试用例修改
      ]
    ]
  }
}
```

> `git commit -m"feature: 初始化工程"`

九.`Vitepress`编写组件文档

```
pnpm install vitepress -D # 在doc目录下安装
```

```
{
    "scripts": {
        "dev": "vitepress dev ."
    }
}
```

> 在根项目中增添启动命令

```
"docs:dev": "pnpm -C docs dev"
```

1).首页配置

```
---
layout: home

hero:
  name: z-ui 组件库
  text: 基于 Vue 3 的组件库.
  tagline: 掌握 vue3 组件编写
  actions:
    - theme: brand
      text: 快速开始
      link: /guide/quieStart

features:
  - icon: 🛠
    title: 组件库构建流程
    details: Vue3 组件库构建...
  - icon: ⚙️
    title: 组件库单元测试
    details: Vue3 组件库测试...
---
```

2).文档配置文件

`.vitepress/config.js`

```
module.exports = {
  title: 'Z-UI',
  description: 'zi-shui UI',
  themeConfig: {
    lastUpdated: '最后更新时间',
    docsDir: 'docs',
    editLinks: true,
    editLinkText: '编辑此网站',
    repo: 'https://gitee.com/login',
    footer: {
      message: 'Released under the MIT
License.',
      copyright: 'Copyright © 2022-present Zi
Shui'
    },
    nav: [
      { text: '指南', link:
'/guide/installation', activeMatch: '/guide/' },
      { text: '组件', link: '/component/icon',
activeMatch: '/component/' }
    ],
    sidebar: {
      '/guide/': [
        {
          text: '指南',
          items: [
            { text: '安装', link:
'/guide/installation' },
            { text: '快速开始', link:
'/guide/quieStart' }
          ]
        }
      ],
      '/component/': [
        {
          text: '基础组件',
          items: [{ text: 'Icon', link:
'/component/icon' }]
        }
        ]
      }
    }
  }
```

3).主题配置

`.vitepress/theme/index.ts`

```
import DefaultTheme from 'vitepress/theme'
import '@zi-shui/theme-chalk/src/index.scss'
import NIcon from '@zi-shui/components/icon'

export default {
  ...DefaultTheme,
  enhanceApp({ app }) {
    app.use(NIcon); // 注册组件
  }
}
```

> 添加`vite.config.ts` 让其也支持 `defineOptions`

4).Icon组件编写

```
# Icon 图标
```

~~~
z-ui 推荐使用 xicons 作为图标库。

```

$ pnpm install @vicons/ionicons5
```


## 使用图标

- 如果你想像用例一样直接使用，你需要全局注册组件，才能够直接在项目里使用。

<script setup lang="ts">
import { CashOutline } from '@vicons/ionicons5'
</script>
<ZIcon color="red" size="40">
  <CashOutline/>
</ZIcon>

<ZIcon color="green" size="40">
  <CashOutline/>
</ZIcon>
<ZIcon color="blue" size="40">
  <CashOutline/>
</ZIcon>
<div>

<ZIcon color="red" size="60">
~~~

~~~markdown
  <CashOutline/>
</ZIcon>

<ZIcon color="green" size="60">
  <CashOutline/>
</ZIcon>

<ZIcon color="blue" size="60">
  <CashOutline/>
</ZIcon>
</div>

```vue
<script setup lang="ts">
import { CashOutline } from '@vicons/ionicons5'
</script>
<template>
  <ZIcon color="red" size="40">
    <CashOutline />
  </ZIcon>
</template>
```


## API


### Icon Props

~~~

```
| 名称   | 类型                     | 默认值      | 说明       |
| ----- | ----------------- | ---------- | --------- |
| color | string                   | undefined | 图标颜色 |
| size  | number \| string | undefined | 图片大小 |
```