前端 LMES 开发文档

一、 背景

lmes-cli 搭建工具，为了快速创建 lmes 外部组件开发模板而开发的工具，减少增删改查等一系列重复工作，提升开发效率。

https://www.npmjs.com/package/lmes-cli

安装

```
yarn add -g lmes-cli

or

npm install -g lmes-cli

or

pnpm install -g lmes-cli
```

创建项目

```
lmes create my-project

or

lmes-cli
```

启动项目

```
cd my-project

yarn dev

or

cd my-project

npm run dev
```

二、 安装

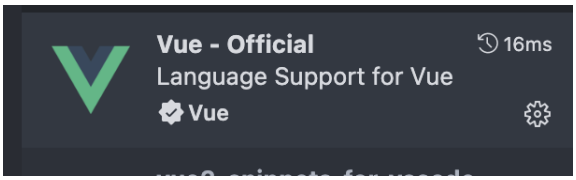
开发工具

推荐使用 VS-Code

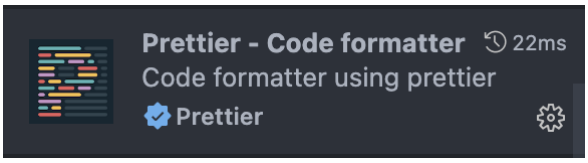
[Visual Studio Code - Code Editing. Redefined]

安装必备的插件

1. Vue - Official



2. Prettier - Code formatter



Nodejs

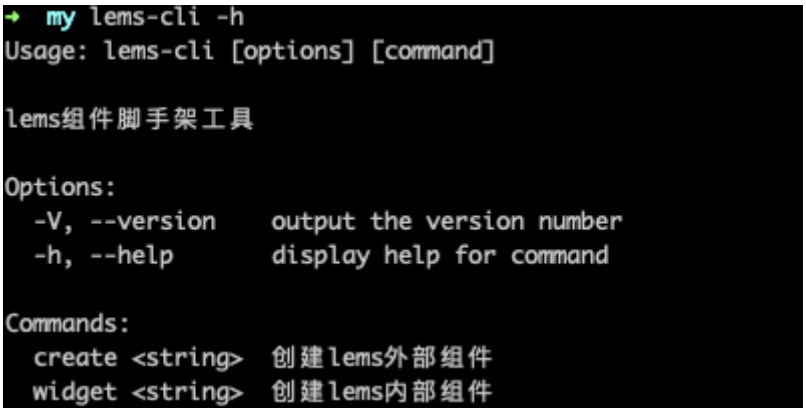
```
nodeJS >= 18
```

[下载 | Node.js 中文网]

```
npm install -g lmes-cli
```

安装完成后，运行以下命令查看帮助

```
lmes-cli -h
```



运行命令快速创建 lmes 开发模板

```
cd [文件夹]
npm run dev
```

```
➜  my cd Test
➜  Test npm run dev

> Test@1.0.0 dev
> npm run menu && vite --host


> Test@1.0.0 menu
> node ./script/autoMenu.js

执行时间：  5 ms
The CJS build of Vite's Node API is deprecated. See https://vitejs.dev/guide/troubleshoot
[mode] development
[env] {
  VITE_PORT: '8001',
  VITE_APP_TITLE: 'CMS',
  VITE_API_URL: 'http://192.168.1.18:18000',
  VITE_APP_NAMESPACE: 'cs'
}

  VITE v5.1.5  ready in 494 ms

  ➜  Local:   http://localhost:8001/
  ➜  Network: http://192.168.2.28:8001/
  ➜  Network: http://10.211.55.2:8001/
  ➜  Network: http://10.37.129.2:8001/
  ➜  press h + enter to show help
```

三、 创建组件

当我们有需求，想要重新创建一个组件时，不需要手动一个个去创建文件和文件夹，只需要通过运行以下命令，即可达到相同的效果，并且支持基础的增删改查。

```
lmes-cli widget FlowManagement

? 请输入实体名称(例如: Process) Flowdefinition
```

```
→  Test lems-cli widget FlowManagement
? 请输入实体名称(例如: Process) Flowdefinition
替换文件内容
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Controllers/MyEntityName.ts
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Controllers/MyEntityNameDrawer.ts
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/enum.ts
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/index.ts
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Models/MyEntityName.ts
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Models/MyEntityNameDrawer.ts
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Models/Service/MyEntityName.ts
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Models/Service/MyEntityNameDrawer.ts
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/type/Type.d.ts
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Views/config/MyEntityName.json
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Views/MyPluginName.module.scss
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Views/MyPluginName.tsx
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Views/Pages/Dialog/MyEntityNameDrawer/MyEntityNameDrawer.t
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Views/Pages/MyEntityName/MyEntityName.module.scss
/Users/guangguang/.nvm/versions/node/v18.14.0/lib/node_modules/lems-cli/src/bak/Views/Pages/MyEntityName/MyEntityName.tsx
替换文件内容完成
内容替换成功,正在替换组件名称...
文件名称替换成功: MyPluginName.tsx -> FlowManagement.tsx
文件名称替换成功: MyPluginName.module.scss -> FlowManagement.module.scss
文件名称替换成功: MyEntityName.json -> Flowdefinition.json
文件名称替换成功: MyEntityName -> Flowdefinition
文件名称替换成功: MyEntityName.tsx -> Flowdefinition.tsx
文件名称替换成功: MyEntityName.module.scss -> Flowdefinition.module.scss
文件名称替换成功: MyEntityNameDrawer -> FlowdefinitionDrawer
文件名称替换成功: MyEntityNameDrawer.tsx -> FlowdefinitionDrawer.tsx
文件名称替换成功: MyEntityNameDrawer.module.scss -> FlowdefinitionDrawer.module.scss
文件名称替换成功: MyEntityNameDrawer.ts -> FlowdefinitionDrawer.ts
文件名称替换成功: MyEntityName.ts -> Flowdefinition.ts
文件名称替换成功: MyEntityNameDrawer.ts -> FlowdefinitionDrawer.ts
文件名称替换成功: MyEntityName.ts -> Flowdefinition.ts
文件名称替换成功: MyEntityNameDrawer.ts -> FlowdefinitionDrawer.ts
文件名称替换成功: MyEntityName.ts -> Flowdefinition.ts
组件创建成功
→  Test
```

重新运行 npm run dev 即可查看刚才组件的效果

四、 前端组件库 information-ui

在信息化组件开发过程中，根据当前设计稿，抽象出来了一批具备一定扩展性的组件，该组件

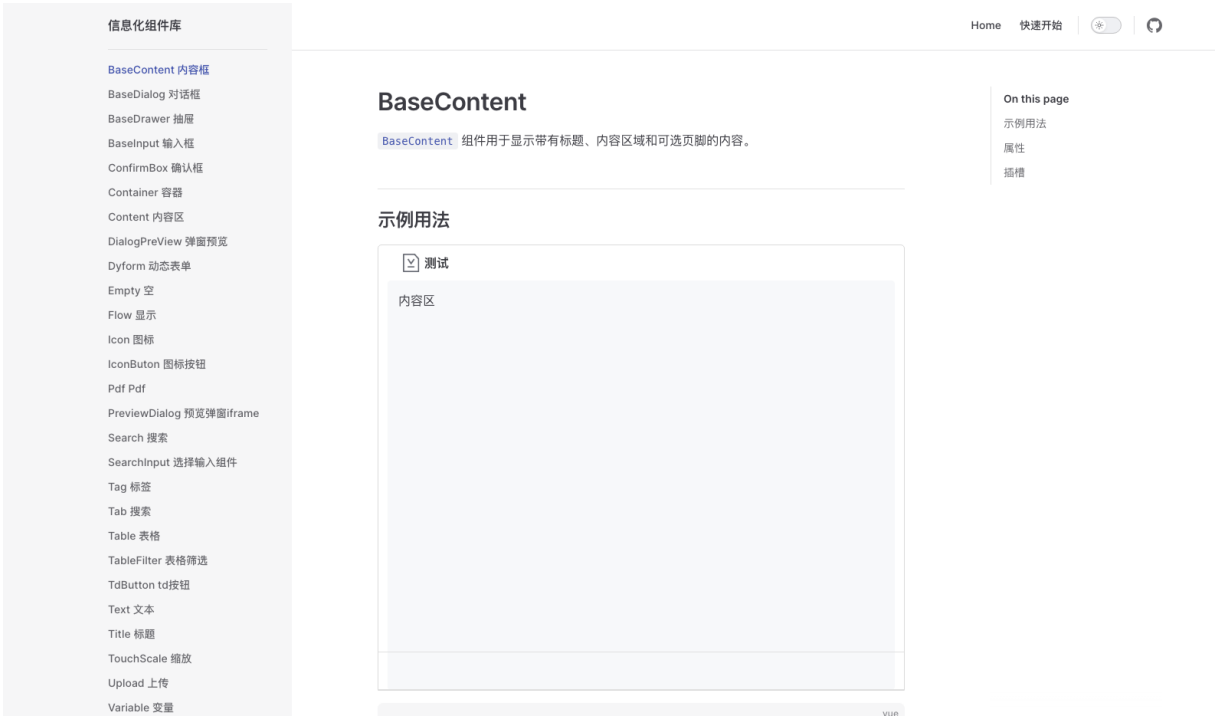
information-ui
前端信息化组件库

快速开始 查看组件

集成
集成于信息化标准组件开发

复用
组件API复用，减少开发成本

样式统一
样式基于信息化组件标准，省去样式走查步骤



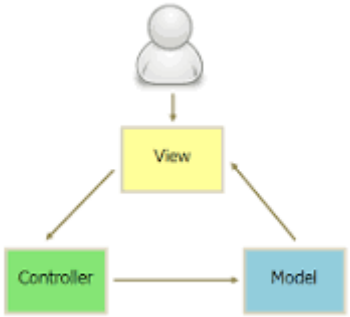
五、 开发规范

目录

```
|- public
|- script
|- src
|----api
|----assets //资源
|----cms //cms sdk 依赖
|----libs
|--------Create //创建对象
|--------Base //基础模块
|--------Permission // 权限
|--------Store //公共数据
|----components //cms sdk 依赖
|------BaseDialog //封装的弹窗，样式统一
|------Table//封装的表格，样式统一
|------other...//其他
|----provider //全局注入，element 命名空间
|------provider.ts // h
|------provider.vue // render 组件
|----utils //工具
|------enum //枚举
|----widgets //组件
|------hook.ts //钩子
|------...
index.html
```

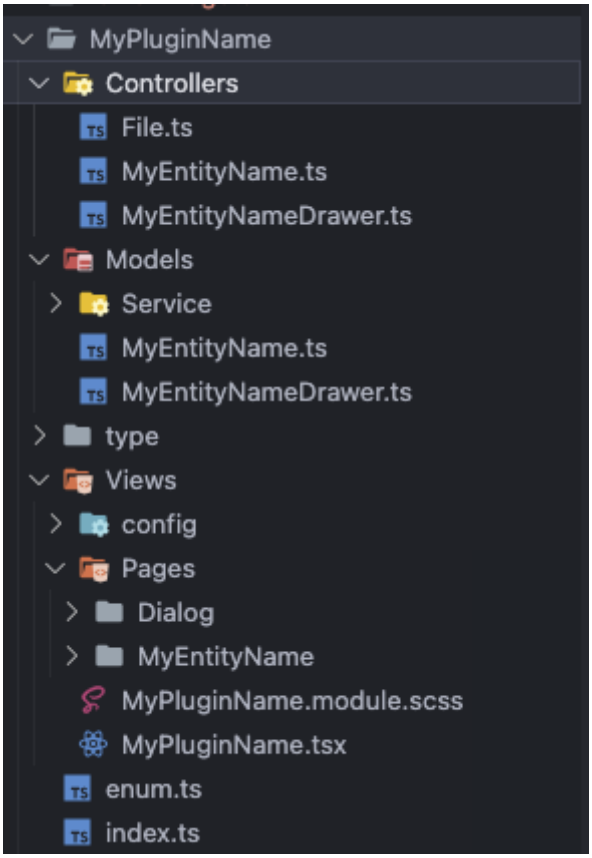
架构

前端组件开发按照 MVC 的设计模式来进行开发，分别为 Controlls、Models、Views



Controller 为业务控制端，在代码中可理解为 hook，关联 models 的数据模型，最终影响 View 端的视图展示。

在代码结构中如以下展示



保证一个 View,对应一个 Model，对应一个 Controller

Model

每个 Model 所对应在全局组件下，都是可以访问的，可单独访问一个 model，也可以访问全局 model，保证数据模型的共用

```
import { injectModel, injectModels } from
'@/libs/Provider/Provider'
// 单独模型
...
 const myEntityName = injectModel<MyEntityName>('myEntityName')
...
//全局模型数据
...
const models = injectModels()
```

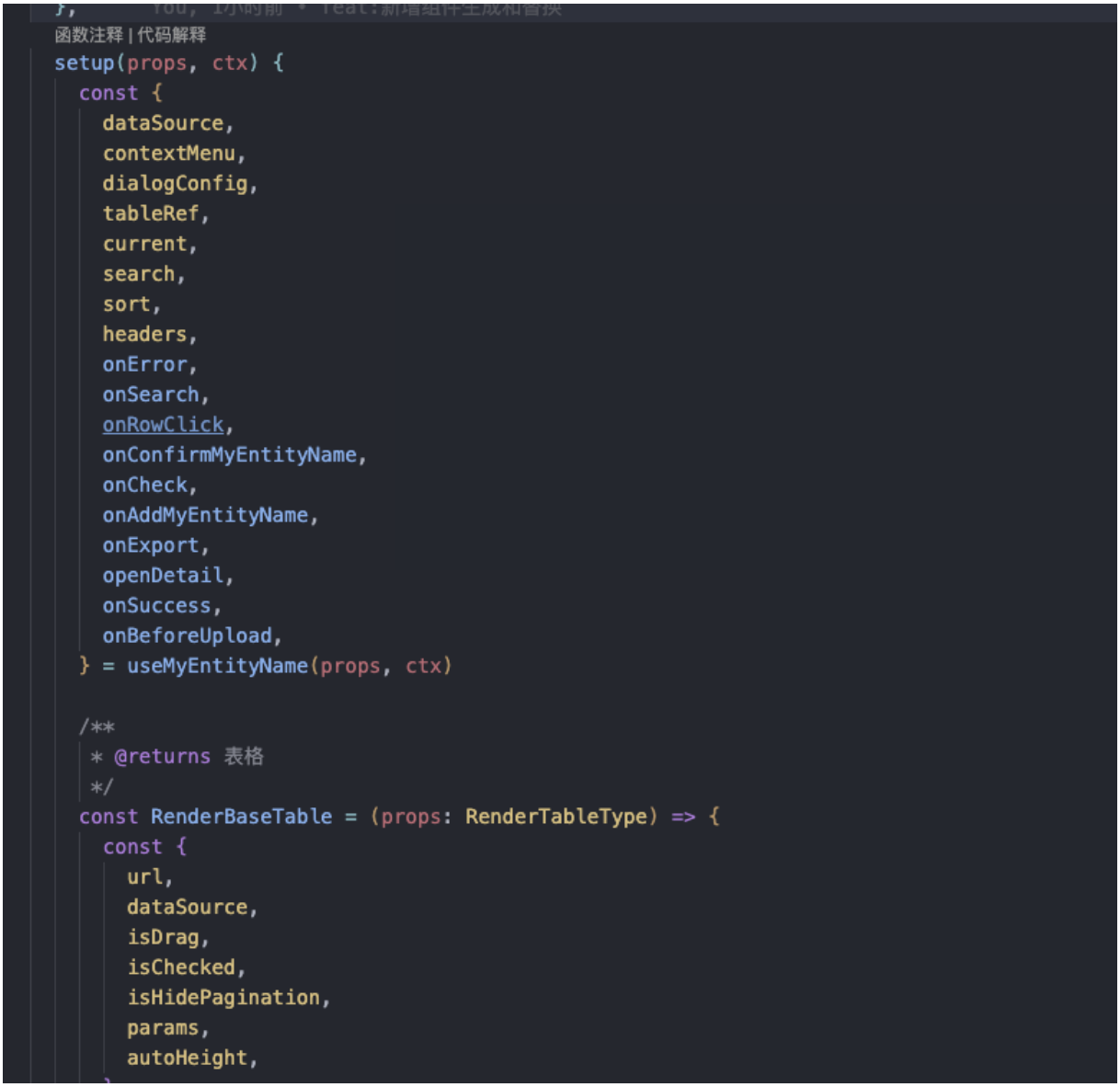
可自行打印结果查看其结构

每个组件 Model 中可传入对象，可通过面向对象的方式拿取使用。

```
export class MyEntityName extends Base<{ [key: string]: any }> {
  constructor() {
    super({
      data: [],
      custom: {}
    })
  }

  getList() {
  }
  ...
}

// 使用 myEntityName 的数据
myEntityName.data.value
myEntityName.custom.value
myEntityName.getList()
```

View

相比较之前的开发，view 更多是跟 Controll 打交道，所以不要将方法放到 view 文件中。保证代码清爽。



```
函数注释 | 代码解释
setup(props, ctx) {
  const {
    dataSource,
    contextMenu,
    dialogConfig,
    tableRef,
    current,
    search,
    sort,
    headers,
    onError,
    onSearch,
    onRowClick,
    onConfirmMyEntityName,
    onCheck,
    onAddMyEntityName,
    onExport,
    openDetail,
    onSuccess,
    onBeforeUpload,
  } = useMyEntityName(props, ctx)

  /**
   * @returns 表格
   */
  const RenderBaseTable = (props: RenderTableType) => {
    const {
      url,
      dataSource,
      isDrag,
      isChecked,
      isHidePagination,
      params,
      autoHeight,
```

Controller

保证组件 hook 和 models 的 正确使用,其他使用参考组件文档（information-ui）

Permission 权限

针对之前的权限不好使用，比较杂乱的问题，统一进行整改，使用 vue 指令进行权限校验。

需要在组件根目录组件中进行引入，包括权限体系

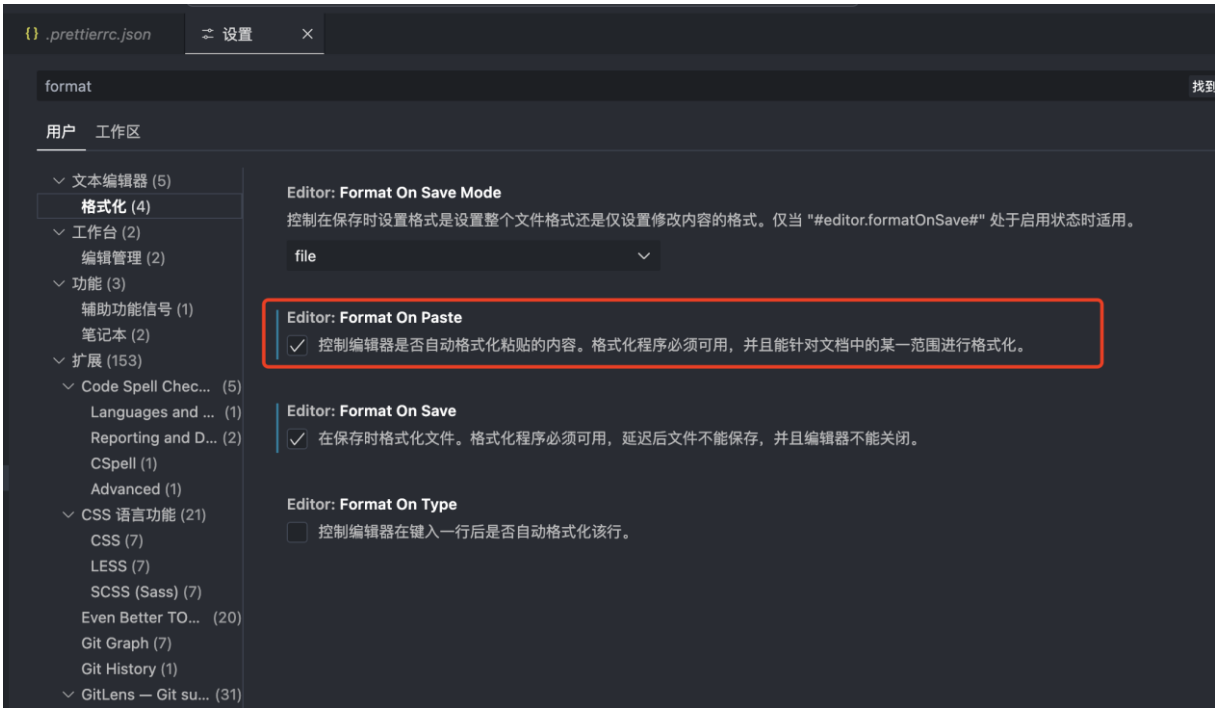
```
import { usePermission } from '@/libs/Permission/Permission'
...

export default defineComponent({
  name: 'MyPluginName',

  setup(props, ctx: SetupContext) {
    useProvideModels()
    usePermission(props, permissionCodes)
...
```

在需要使用权限的地方引入权限指令

```
import { vPermission } from '@/libs/Permission/Permission'

...

export default defineComponent({
  name: 'MyEntityName',
  directives: {
    permission: vPermission,
  },
...

 <IconButton
    v-permission="myEntityName-add"
    icon="add-p"
    onClick={onAddMyEntityName}
    type="primary"
  >
    添加
  </IconButton>
```

六：代码格式化

默认使用.prettierrc 进行代码格式化，设置 vscode 保存自动格式化



如果格式化失效，请检查编辑器状态栏，位于编辑器的最下方：

点击后在编辑器的提示中，选择 prettierrc 进行配置。

七、 代码规范

组件开发规范

变量及注释

1. 变量命名需要采用语义化命名，不要出现 a,b,c,d 等。
2. 变量名及部分函数需要增加注释，近可能简短重点表达其含义
3. 函数逻辑变更时，及时更新注释。
4. 常量及不会再次赋值的变量使用 const

枚举

需要将枚举类型抽象到单独文件中，例如：

```
export const PRODUCTION_TYPE = {
    0: '线下',
    1: '线上'
}
...
import { PRODUCTION_TYPE } from './enum.ts'
type ProductionType = typeof PRODUCTION_TYPE
```

Request

需要以组件为单位去新建请求 api

widgets/[组件]/api/index.ts

```
import sdk from 'sdk'
const { request } = sdk.utils
// 获变量绑定
function getVariableBinding() {
  const url = `/api/v1/showroom/dashboard/binding`
  const method = 'get'
  return sdk.request({ url, method })
}
```

组件引入

检查组件依赖关系，尽可能避免循环依赖

Typescript

命名约定

1. 使用驼峰式命名：变量、函数和方法名首字母小写，类名首字母大写。
2. 使用描述性名称，避免使用缩写除非是通用缩写（如 HTTP）。

类型定义

1. 明确指定变量、函数参数和返回值的类型。
2. 尽可能使用 TypeScript 内置的基本类型（如 **number**, **string**, **boolean** 等）或自定义类型。

使用接口和类型别名

利用接口和类型别名来定义自定义数据类型和结构。

避免使用 any 类型

尽量避免使用 **any** 类型，除非必要，因为它会破坏 TypeScript 的类型检查。

文件命名

采用首字母大写，中间不加横岗

.vue 文件：Test.vue

.tsx 文件：Test/Test.tsx

空格缩进

设置文件新建时，缩进为 2 个字符

css 规范

模版开发时，需要使用 scoped 进行作用域设置

```
<style lang='scss' scoped>
...
</style>
```

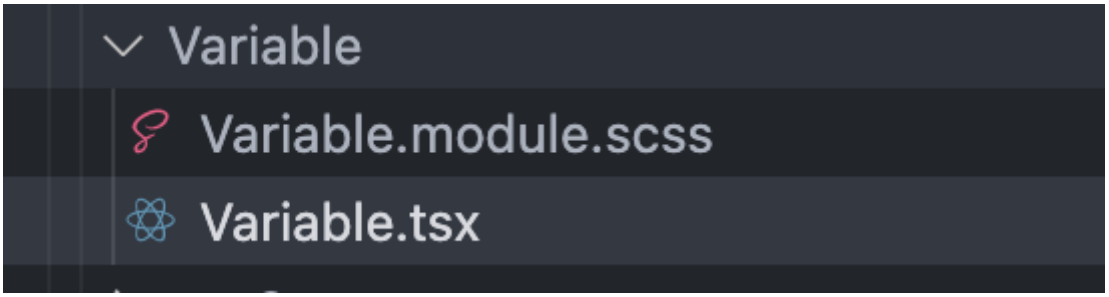
class 命名

采用连接符号'-'进行分割，保证语义化

如：.list-box .content 等

tsx 方式开发时

采用模块化的方式，例如：



```
import styles from './Variable.module.scss'

...
<div class={styles.box}>...</div>
```

配置

开发一个组件，需要在 src/widgets 下新建文件夹，文件夹命名需要保证跟 src/widgets/index.ts 中的 name 一致，如下：

```
import Example from './Example.vue'
import ExampleSettings from './Example.settings.vue'
import { provider } from '@/provider/index'

export default {
  is: 'Example',
  name: '例子',
  category: 'run',
  authorizationRequired: true,
  icon: 'icon-tiaomaguanli',
  canvasView: provider(Example),
  settingsView: ExampleSettings,
}
```

需要注意的是，authorizationRequired: true 在 cms2.2.0 后，需要进行组件授权才可使用。

关于 provider，是针对 cms2.0 中二开组件样式会被影响所采取的方案，使用 element-plus 中的作用域进行封装，可以解决在开发中存在的样式冲突与 zIndex 等全局问题。

```
import { h } from 'vue'
import Provider from './index.vue'

export const provider = (Widget: any) => {
  return (arg: any) => {
    return h(Provider, null, {
      default: (props: any) => {
        return h(Widget, {
          props,
          ...arg,
        })
      },
    })
  }
}
```

变量弹窗

可以使用组件提供已经封装好的 Variable 组件进行开发

```
const a = 1
const b = '1'
if(a === b) {
    // ...
}
```

DOM 与选择器

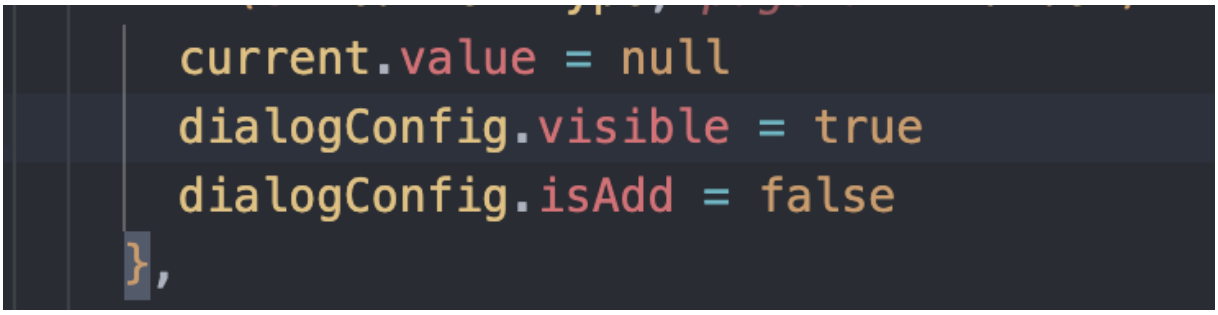
尽量避免在代码中使用 document 等 Dom 操作符，可以使用 ref 来获取 dom

```
<div ref="divRef"><div>
```

分号

代码结束后，无需写”;“，换行即可

需保证代码格式化的正常



注释

每个变量及函数，需要进行注释，保证后续维护的可读性

变量声明

禁止使用 var，需使用 let 与 const

对于可变变量使用 let，常量使用 const

```
const initiateData: Ref<Record<string, any>> = ref({})
const formData = reactive<Record<string, any>>({})
const paramsData = ref([])
const formulaData = ref([])
const materielData = ref([])

// ref
const formRef = ref()
const paramRef = ref()
const formulaRef = ref()
const materielRef = ref()
const materialTypeOptions = ref<
  {
```

Git 规范

commit 提交

1. feat - 新功能 feature
2. fix - 修复 bug ----需要增加 bug Id 及简要描述，便于排查 fix:#[bugId] 修复表格显示问题
3. docs - 文档注释
4. style - 代码格式(不影响代码运行的变动)
5. refactor - 重构、优化(既不增加新功能，也不是修复 bug)
6. perf - 性能优化
7. test - 增加测试
8. chore - 构建过程或辅助工具的变动
9. revert - 回退
10. build - 打包

git 分支

开发状态

feature/[用户]/[模块

发布状态

release/[日期] 如：releaes/1230

修复状态

fix/[用户]/模块

八、打包及编译

```
// 编译所有组件
npm run build
// 编译指定组件
npm run build [组件名]
```

将打包的产物以下位置，即可完成组件的开发

[cms edit]/host/wwwroot/widgets/