



# 基础研发平台《Web前端项目编码规范》

目录

版本号	基础研发平台《Web前端项目编码规范》 V1.0
制定团队	基础研发平台Web前端编码规范专家组（以姓氏首字母排序）： 蔡安法、陈俊林、杜万智、韩紫东、金林成宇、李阳、林业、马蒙光、庞军、晨宇、王萌、王阳、王永青、温华剑、逢淑辉、杨天、周明建、朱民、邹瑞
适用范围	基础研发平台全体人员
签发单位	基础研发平台研发流程标准化委员会
生效日期	2024年8月23日

## 目录

- 一、前言
- 二、原则
  - 2.1 适用范围
  - 2.2 规范类别说明
  - 2.3 基本原则
- 三、名词解释
- 四、文件、目录和包规范
  - 4.1 命名
  - 4.2 文件
  - 4.3 目录
  - 4.4 包
- 五、编码风格
- 六、语言规范
- 七、框架编码规范
- 八、单元测试规范
- 九、检查工具与卡控规范
- 十、附则

## 一、前言

为保证产品线上质量、提升项目的可维护性、提升迭代和协作效率，需要在基础研发平台各前端团队，建立统一的前端项目编码规范，旨在帮助开发者编写一致、可读性高、易于维护的代码。因此，Web前端编码规范组征询了本部门各前端团队建议，结合业界最佳实践，经过慎重讨论后，拟定了用于前端项目开发场景的编码规范——《基础研发平台Web前端项目编码规范》（下文称“本规范”）。本规范主要约束了前端项目中 JavaScript、TypeScript 两大常用编程语言的格式和语言特性，及其单测用例编写规范，以及 React、Vue 两大常用编程框架的用法。

## 二、原则

### 2.1 适用范围

- 以项目维度定义适用范围：代码作用于Web应用、小程序应用、Electron和React Native应用等主要使用JavaScript、TypeScript语言的项目。
- 以代码开发人员维度定义的适用范围：在基础研发平台组织架构内的所有员工。
- 本规范不适用于项目引入的依赖库和项目编译或构建的产物。

### 2.2 规范类别说明

本规范参考 [RFC2119](#) 中的术语，分为【强制】（对应 MUST / MUST NOT）、【建议】（对应 SHOULD / SHOULD NOT）两个类别，其释义如下：

图表2：图表名称

类别	说明
强制	通常情况下 <b>必须</b> 采取的行为；特殊情况下，经过主管与 QA 主管审批后，可临
建议	通常情况下 <b>应该</b> 采取的行为，可以根据实际情况做适当调整。

### 2.3 基本原则

- 可读性**：代码应该易于阅读和理解。选择有意义的变量名、函数名和注释，避免使用难以理解的缩写或者术语，除非它们已经是行业内公认的标准。
- 一致性**：整个项目中的编码风格应该保持一致。
- 简洁性**：尽量保持代码简洁，避免不必要的复杂性。使用简单的逻辑和清晰的流程控制来表达程序的意图。
- 可维护性**：编写代码时应考虑到长期维护的便利性。使用清晰的模块化和良好的接口设计，使得代码易于修改和扩展。
- 可复用性**：不要重复编写相同的代码块。使用函数、模块或者类来重用代码，增加代码的复用性。
- 性能考量**：编写代码时应考虑到性能的影响。尽量优化算法和数据结构，减少不必要的计算和内存使用。
- 错误处理**：代码应该能够优雅地处理错误和异常情况，避免程序崩溃或产生不可预期的行为。
- 安全性**：在编写代码时，始终考虑安全性。避免常见的安全漏洞，比如SQL注入、跨站脚本攻击等。
- 注释和文档**：编写清晰的注释和文档，解释代码的功能、参数、返回值和任何复杂的逻辑。设计和逻辑发生变化时，及时更新文档和注释，但同时避免过度注释。
- 可测试性**：编写可测试的代码，确保代码的质量。

## 三、名词解释

- 前端项目**：特指本规范适用的项目。

- **编码：**本规范特指编写应用程序源代码。
- **命名法：**命名文件、类、方法、常量、变量等标识符的方法。本文涉及的命名法如下表（表 1：文件命名法列表）所示。

表 1：文件命名法列表

命名法	要求	举例
连字符命名法	只能包含英文小写字母（a-z）、连字符（-）和数字（0-9）。禁止首位字符出现数字。单词间以连字符分隔。	user-
驼峰命名法	只能包含英文大小写字母（A-Za-z）和数字（0-9）。禁止首位字符出现数字。单词间没有分隔字符。第一个单词的首字母可采用小写。从第二个单词开始，每个单词的首字母采用大写。	小驼峰 userInterface 大驼峰 UserProfile
帕斯卡命名法	只能包含英文大小写字母（A-Za-z）和数字（0-9）。禁止首位字符出现数字。单词间没有分隔字符。每个单词的首字母采用大写字母。	Car、
蛇形命名法	只能包含英文小写字母（a-z）、下划线（_）和数字（0-9）。禁止首位字符出现数字。单词间以下划线分隔。	first_user
宏命名法	只能包含英文大写字母（A-Z）、下划线（_）和数字（0-9）。禁止首位字符出现数字。单词间以下划线分隔。	MAX

## 四、文件、目录和包规范

### 4.1 命名

1. **强制** 必须从下面几种方式中选择一种作为文件和目录命名的方式：
  - 如果文件中只有一个类，则采用**帕斯卡命名法**（详见表 1：文件命名法列表）。

^ 正面例子

TypeScript

// 文件 **Car.ts**

```
export default class Car {}
```

- UI 组件目录和对应的文件，采用**帕斯卡命名法**（详见表 1：文件命名法列表）。

^ 正面例子

TypeScript

Toast // **目录名**

├─ index.scss

└─ index.tsx

- 如果文件中只有唯一类的实例、类型、函数等情况，则文件跟导出的变量、类型、函数同名。

^ 正面例子

TypeScript

```
1 // 文件 stopWatch.ts
2 class Stopwatch{}
3 export const stopWatch = new Stopwatch();
4 // Enum 类型 TabStatus.ts
5 export default enum TabStatus {
6     UNINITIALIZED,
7     LOADING,
8     LOADED,
9     SHOWN,
10    HIDDEN,
11    ERROR = -1
12 }
13 // 文件 parseArgs.ts
14 export default function parseArgs(){}

```

- 除以上三种情况，文件和目录均使用连字符命名法（详见表 1：文件命名法列表）。

落地方式：CodeReview

2. 建议 目录命名遵循简洁原则，有习惯性缩写的单词采用容易理解的缩写。如：源代码目录使用src，不使用source。下面是更多例子：

目录名	说明
img	图片。不使用image、images、imgs等。
js	javascript脚本。不使用script、scripts等。
css	样式表。不使用style、styles等。
deps	引入的第三方依赖包目录。
docs	文档目录。
typings	TypeScript类型目录。
__tests__	测试用例目录。
components	通用组件、或纯组件，跟业务绑定浅。
utils	通用的工具函数， <span>建议</span> 一个项目保留一个utils目录，保证纯函数复用。
helpers	业务逻辑抽象，与utils的区别是 ❶ utils 面向整个项目，helpers 面向具体则上只有一个 utils，但 helpers 目录会散落在业务模块中。
views	view相关组件/页面。
pages	页面相关的目录。
modules	具体业务模块组件，一般跟 Stores、React Context 耦合。

落地方式：CodeReview

3. **建议** 文件名中不应重复目录名。

反面例子

Plain Text

1

api

2

└─ api-log.js // 文件名中有了api

正面例子

Plain Text

1

api

2

└─ log.js

落地方式：CodeReview

4.2 文件

1. **强制** 文件使用无 BOM 的 UTF-8 编码。

说明：UTF-8 编码具有更广泛的适应性。BOM 在使用程序或工具处理文件时可能造成不必要的干扰。

落地方式：CodeReview

2. **建议** 在文件结尾处，保留一个空行。

落地方式：CodeReview

4.3 目录

1. **强制** 同一目录下禁止拥有同名的 .js/.ts 和 .jsx/.tsx 文件。

说明：在使用模块导入时，倾向于不添加后缀，如果存在同名但不同后缀的文件，构建工具将无法决定哪一个是需要引入的模块。

反面例子

TypeScript

1

└─ Demo

2

└─ index.js // 同时存在同名的 index.js 和 index.ts

3

└─ index.ts

落地方式：CodeReview

2. **强制** 目录下禁止同时存在目录与js/jsx/ts/tsx文件同名的情况

说明：如果同时存在目录与js/ts等文件同名的情况，那么在导入时可能不清楚应该导入哪个文件，导致混淆。可能会引起构建工具的兼容性问题。

反面例子

TypeScript

1

└─ badcase

2

└─ Demo.ts // 同时存在 Demo.ts Demo.tsx 和 Demo 目录

3

└─ Demo.tsx

4

└─ Demo

5

└─ index.ts

```
6
7 // 这种情况下需要避免重名，如果Demo和Demo/index.ts同时存在则需要精确的引入
8 import Demo from './Demo/index.ts'
```

落地方式：CodeReview

#### 4.4 包

- 强制** NPM包项目，源码必须使用ES Module（ESM）语法，即 import / export

落地方式：CodeReview

- 建议** 组件、库等作为三方对外提供的NPM包建议采用ESM发布

说明：ESM是现代JavaScript的官方标准模块系统，它提供了更好的静态分析能力和模块化  
管理，有利于应用方进行按需打包和更好的tree-shaking。

落地方式：CodeReview

### 五、编码风格

编码风格是JavaScript和TypeScript，以及React和Vue框架都需要遵守的风格规范，详见文档  
[编码风格](#)。

### 六、语言规范

**建议** 建议新项目采用TypeScript编码，具体语言规范详见：

- [JavaScript 编码规范](#)
- [TypeScript 编码规范](#)

### 七、框架编码规范

- [React 框架编码规范](#)
- [Vue 框架编码规范](#)

### 八、单元测试规范

[单元测试规范](#)

### 九、检查工具与卡控规范

对应规范的ESLint工具和PR卡控流程还在开发中，请关注后续更新。

- 强制** 项目必须引入 ESLint 工具检查代码是否合规，并配置 ESLint 的检查规则以匹配本规范关于语言和框架的要求。
- 强制** 如果检查时项目代码完全符合本规范的 **强制** 要求，则检查通过，否则检查不通过。
- 强制** 代码检查如果失败，则必须产出报告，其中能够明确指出违反的规范（或检查规则）和违规代码行列，
- 强制** 项目仓库的配置中必须添加代码合规<sup>[1]</sup>的 Pull Reqeust 检查项。

5. 

强制

如果 Pull Request 检查项的代码合规检查失败则禁止当前 Pull Request 合入。
6. 

建议

Pull Request 检查项中执行的代码合规检查可仅覆盖代码变更部分。

十、附则

1. 本规范由基础研发平台Web前端编码规范组负责制定、修改和解释。
2. 此前流程与本规范不符的，以本规范规定为准。
3. 本规范未规定事项，参照公司其他相关规范予以实施。
4. 本规范自正式发布之日起施行。

注释：

[1] 代码合规：ESLint检测工具

⚠ 仅供内部使用，未经授权，切勿外传