前端开发规范

目录结构规范

项目目录结构

在项目中，目录结构根据组件和业务耦合来进行划分，分为展示组件和容器组件两种。

组件类型	关注点	外部数据源
展示组件	UI 展示、交互	Props
容器组件	业务逻辑、数据处理、副作用管理	状态管理器、Props、Service 请求

项目目录结构

```
.
 common                  #
 components              #
 containers              #
 hooks                   #  Hooks
 pages                   #
 services                #
 store                   #
 types                   #
 utils                   #
```

组件目录结构

组件内部根据文件的类型、功能等划分不同的目录结构。

组件目录结构

```
.
 HelloWorld              #
     Hello               #
        World.less
        World.tsx       #
     hooks               #  Hooks
     index.tsx           #
     types               #
     utils               #
```

组件内部结构规范

编码规范

Props 定义规范

组件内部 Props 定义以 **I + 组件名 + Props** 的形式导出，同时，内部注释使用多行注释，不要使用单行或者末尾注释。

Props 定义规范

```
export interface IHelloProps {
  /**
   *
   */
  name: string;
}
```

代码导入与类型导入分离

代码导入与类型导入分离

```
//
import Hello from '@/components/Hello';


//
import type { IHelloProps } from '@/components/Hello';
```

使用 React.FC 定义组件

使用 React.FC 定义组件

```
const Hello: React.FC<IHelloProps> = ({ name }) => {
  // ...
};
```

禁止使用 any 类型

无论是显示还是隐式的方式，都不要使用 any 类型。

禁止使用 any 类型

```
//
const foo: any = {};


//
const foo: IFoo = {};
```

一个组件对应一个样式文件

样式文件与组件本身保持一致，不交叉引入，这样会导致重构混乱，无法明确当前这个样式被多少个组件使用。

一个组件对应一个样式文件

```
// Hello.tsx
import styles from './Hello.less';
```

内联样式非必要情况禁止使用

内联样式非必要禁止使用

```
//
const style = {
  // ...
};

return <div style={style}></div>;
```

组件代码行数保持在 140 行以下

保持组件内部逻辑清晰不混乱，鼓励大家对组件进行拆分优化。

事件函数命名区分

内部函数按照 **handle[Type][Event]** 的形式进行命名，如 **handleNameChange**。

暴露到外部的函数按照 **on[Type][Event]** 的形式进行命名，如 **onNameChange**。

事件函数命名

```
const handleNameChange = () => {
  // ...
};


const onNameChange = () => {
  // ...
};
```

不要直接使用 export default 导出组件

这种方式导出的组件在 React Inspector 查看时会显示为 Unknown。

不要直接使用 export default 的形式导出组件

```
//
export default () => {};

//
const Hello: React.FC<IHelloProps> = () => {};

export default Hello;
```