FE Style Guide

re

编码规范

目录组织

```
kubernetes-platform-console/
├── src/
│ ├── api/
│ ├── assets/
│ ├── common-styles/
│ ├── components/
│ ├── constants/
│ ├── helpers/
│ ├── hocs/
│ ├── hooks/
│ ├── states/
│ ├── App.tsx
│ ├── global.css
│ ├── index.html
│ └── index.tsx
├── stories/
├── typings/
├── deploy/
├── .babelrc
├── .editorconfig
├── .eslintrc.json
├── gitlab-ci.yml
├── .prettierignore
├── commitlint.config.js
├── nginx.conf
├── tsconfig.json
└── webpack.config.babel.js
```

上面各个文件夹/文件分别代表：

- src：这个文件存放项目的源代码，包括下面的子目录：
- api：存放api调用代码
- assets: 存放项目的静态文件
- common-styles: 存放项目公共的styled文件
- constants: 存放项目的所有常量
- helpers: 存放项目的所有帮助函数
- hocs: 存放项目的高阶组件
- hooks: 存放项目的公共hooks
- states：存放全局的state
- App.tsx: 定义主应用
- index.tsx: 渲染主应用到html上面
- index.html: 应用的html文件
- global.css: 全局css样式
- stories: 存放项目的公共组件的storybook的story
- typings: TypeScript的global typing
- deploy：存储项目部署配置
- .babelrc: babel配置
- .editorconfig: vscode配置
- .eslintrc.json: eslint配置
- gitlab-ci.yml: gitlab ci配置
- .prettierignore: prettier ignore配置
- .commitlint.config.js: commitlint配置
- nginx.conf: nginx配置
- tsconfig.json: typescript配置
- webpack.config.babel.js: webpack配置

命名变量

- 好的变量命名应该可以帮助别人或者未来的自己更好地理解代码的思路和逻辑，变量命名需要满足下面规范：
- 一般变量需要用`驼峰法`命名，如`thisIsANormalVariable`，常量要全大写加下划线命名，例如`THIS_IS_A_CONSTANT`。
- 变量名尽可能使用单词的全称，例如用`response`而不是`res`或者`resp`，`value`而不是`v`。如果变量名称很长我们也要争取每一个单词都要写全，例如`getServiceDetailForMainPage`等。注意：对于一些大家都知道全称的名词例如标识符这种，可以使用`id`而不用全写`identifier`。
- 变量名应该要尽可能具体到类型。对于一个代表`service`的数组，可能有下面这些命名：
  - service（要用复数）
  - services（）
  - serviceArray（）

    - serviceList（）
- 布尔值的变量命名应该是`肯定语气`并且要`符合英语语法`。对于用来指示页面某个按钮是不是被禁用状态的变量，可能有下面这些命名：
    - notEnabled（违反肯定语气）
    - beDisabled（语法错误）
    - isDisable（语法错误）
    - isDisabled ()
- 方法命名要体现`动词`和被作用的对象（`宾语`），而且要保证`语法正确`。如果该方法是用来进行数据转换的话，方法名要出现`from`或者`to`等字眼。对于一个用来将数组转换成对象的变量，可能有下面这些命名：
    - toObject（没有动词）
    - transformArray（没有目标数据信息）
    - transformToObject（没有源数据信息）
    - objectify（没有源数据和目标数据信息）
    - transformArrayToObject（）
- 类的`名称`和`文件名`都要用`大写`
- enum的type要使用大写开头的驼峰命名，key值要使用全大写命名，非特殊情况下采用数字枚举

> *备注：你写代码时为了让**别人或者将来的自己看懂的**，当我们命名一个变量的时候应该多想想**如果另外一个人读他能读懂吗？**

数据管理

对于页面上的数据保存在那里的问题，有下面这些规范：

- 如果该数据只有本组件用到就把该数据保存在组件的`state`里面
- 如果该数据在2层以内传递把该数据保存在`state`里面通过`props`，如果该数据传递层数超过2层就把该数据保存在`context`里面
- 如果该数据在页面上多个不同的地方使用到（分散的组件）就把该数据放在全局状态state里面

API调用

- 使用`hooks/useAsyncIntervalFn`来管理API调用的`loading状态`，`轮询`和`错误信息展示`
- 使用`generator-swagger-tap`来自动生成调用后端API的代码和相关数据类型

页面路由

定义

如果某个页面是用来展示某个资源详情信息的，这个页面的路由必须包含该页面的`主资源`以及该`主资源的所有父级节点`（用来展示面包屑）的`name/id`信息。

代码管理

为了方便代码管理，和路由相关的代码都放到了`constants/routes`文件夹，这个文件夹包含下面这些文件：

- routes.tx: 这个文件用来存储页面上的所有路由，而且为这些路由实现了对应的`buildXXXRoute`方法，这些方法是某个页面路由的构造器。如果你需要从一个页面点击跳转到另外一个页面，你就需要使用对应的路由构造器`根据被跳转的资源的数据拼接出该资源详情页面的路由`，然后使用`history.push`方法来进行跳转
- name.tsx: 这个文件来用存储应用使用到的资源的`name/id`，同样也会有对应的`buildXXXName`函数来构建这些资源的`name/id`
- identifier.tsx: 这个文件用来存储`name.tsx`里面的资源的`name`用到的`hardcoded`的字符串。将所有`hardcoded`的字符串统一起来存放的目的是为了`方便路由更改`

其它

不能使用`history.pushState`来在不同路由之间传递数据，页面之间传递数据的唯一方式应该体现在导航栏上面的路由上（path/queryString/hash）。

组件

命名

项目的组件一般可以分为两种类型，分别是`路由型组件`，`资源展示型组件`和`资源编辑型组件`：

路由型组件

这种组件是用来`组织页面路由`或者`聚合页面信息`的，这种组件一般是名词的组合，例如`ApplicationManagement`，`ApplicationDetail`和`Layout`等。

资源展示型组件

这种组件是用来展示资源信息的，命名规范是`资源类型` + `组件类型`。资源类型就是这个组件展示的资源，例如`Service`，`Application`和`Deployment`等，而`组件类型`则是指这个组件的`UI类型`，例如`Table`，`List`，`Editor`，`Form`和`Drawer`等。符合规范的组件例子有`ServiceTable`，`ApplicationEditor`和`DeploymentList`等。

资源编辑型组件

资源编辑型组件是对某个资源增删改的组件，这类组件的命名规范是`动词` + `资源类型` + `组件类型（可选）`。`资源类型`和`组件类型`上面说过了，而动词代表的是我们要对这个资源进行的操作，例如新建`必须是Create`、编辑`必须是Edit`而删除`必须是Delete`。例如一个新增`Application`的组件可以叫做`CreateApplicationForm`。

Function Component VS Class Component

`如无特殊情况都要使用Function Component`。

Props

组件的props命名要符合[变量命名规范](#变量命名)。除此之外，对于组件的`自定义事件`，它们必须以`on`作为开头。下面是一些例子：

- submit（需要以on开头）
- error（需要以on开头）
- succeed（需要以on开头）
- onSubmit（）
- onError（）
- onSucceed（）

这些组件的父级组件在传递参数给该组件时，需要给这些自定义事件的参数传递一个叫做`handleXXX`的事件处理函数（handler），下面是一些例子：

- handleSubmit（）
- handleError（）
- handleSucceed（）

文件路径

路由关联组件

和页面路由关联的组件按照`路由层级关系来组织`，例如：
`ApplicationsManagement/ApplicationDetail/Event`

公共组件

全局公共组件放在`components/Common`文件夹底下

某个大的路由底下的子组件用到的公共组件放到最近的`Common`文件夹底下

import路径规则

如果配置了路径别名或者绝对引入，路径应该一律使用别名或者绝对引入

文件组织

每个组件都是一个文件夹，该文件夹的名称就是这个组件的名称，组件的jsx逻辑放在这个文件夹的`index.tsx`里面，该组件的`style`逻辑放在该文件夹底下的`style.ts`底下。

单元测试

哪些代码需要单测

下面类型的代码要进行单元测试：

- 公共的`hook`

- 公共的`hoc`
- 公共的`Component`
- 状态管理复杂的`Component`
- 通用的帮助函数`helpers`

何时编写单测

提交代码的时候就要编写对应的`单元测试`，而不是开发完成之后。

如何编写单测

详情参考这个文章。

代码重构

重构的目的

- 提高代码可读性
- 提高代码的可维护性
- 提高代码的可扩展性
- 复用重复代码，提高项目后期的开发效率

哪些代码需要重构

- 设计不规范的，为了方便别人阅读和添加新的功能就需要重构
- 重复逻辑很多的，需要通过提取公共hook，公共方法和公共组件进行重构

重构的原则

- 通过网站refactor-guru和书本refactoring学习关于重构的一些最佳实践
- 通过单元测试来保证重构不会改变原来代码的行为

逻辑抽象

抽象逻辑的目的

- 提高代码可读性
- 提高开发效率

何时抽象逻辑

当同样的代码逻辑出现`超过两次`就需要进行逻辑抽象了。

如何抽象逻辑

- UI公共逻辑在对应的`Common`文件夹编写公共组件来复用
- 非UI的业务逻辑在`hooks/`文件夹底下添加新的`hook`来复用
- 其它hook覆盖不到的组件逻辑可以考虑使用`hoc`来复用
- 公共数据处理逻辑考虑在`helpers/`文件夹底下编写帮助函数来复用

代码注释

代码注释的目的

- 帮助别人或者自己理解某个函数，hook或者组件的使用方法，而不需要直接去查看源代码
- 提醒别人或者自己去做某些事情
- 告诉别人或者自己当时为什么需要这么写代码

什么代码需要注释

- 公共组件，hook和帮助函数一定要有注释

- 未完成的功能，需要编写`//TODO xxxx`进行注释，而且注释一定要写明白`什么人需要做什么事情`，不能只有一个干巴巴的`TODO`
- 需要完善的代码，需要编写`//FIXME xxx`进行注释，来告诉别人`需要处理另外的情况`或者`find a better solution to this workaround`
- 一些特殊原因（后端或者框架的bug），需要写注释来提醒自己或别人自己当时为什么要这样编码，例如`//hack react route bug: https://buglink.com`

常见交互场景

下面是一些管理平台前端的常见交互场景的正确交互（UX设计师通常会忽略）

新建资源

- 新建资源完成后要`跳转到新建资源的详情页面`
- 如果交互图没有指定成功提示的话需要使用antd的`message.success`进行提示新建资源成功
- 需要在提交按钮展示`loading`状态
- 如果表单有错误或者当前不能提交，则`submit`按钮需要置灰（disabled）

更新资源

- 更新资源成功后需要重新获取该资源的信息来更新页面视图
- 如果交互图没有指定成功提示的话需要使用antd的`message.success`进行错误提示
- 需要在提交按钮展示`loading`状态
- 如果表单有错误或者当前不能提交，则`submit`按钮需要置灰（disabled）
- 如果在某个资源正处于编辑状态时进行了`路由跳转`，则需要弹窗提示是否放弃此次编辑

删除资源

- 如果在该资源详情页删除该资源，需要跳转到该资源的父级资源列表
- 如果在该资源的父级资源列表中删除了该资源则需要刷新该父级资源的资源列表
- 如果交互图没有指定成功提示的话需要使用antd的`message.success`进行删除资源成功的提示
- 需要在提交按钮展示`loading`状态

代码提交规范

提交代码

每次提交的代码需要满足下面条件：

- 代码是可以运行的
- 所有单元测试必须全部通过
- 不能有eslint的错误
- 不能有多余的`console.log`
- 提交内容尽可能少而且是逻辑独立的，不然`reviewer`很难对代码进行review
- 提交的内容和编写的`commit message`必须一一对应，不能包含其它无关的内容
- 提交之前用`git add xxx`而不是`git add .`
- 提交的内容要多看几遍！多看几遍！多看几遍！

commit message

- 需要使用`英语`
- 不能有语法错误，一般要有`动词` + `宾语`，语态是`第一人称一般现在时`。
- 多用程序员conventional的单词，不能硬搬google translate
- 需要符合conventional commit的规范，小组分享链接

代码合并

- 每个merge request都要有对应的`jira ticket`
- 每个merge request都要通过所有的CI检测，`不能merge有CI错误的代码提交`

项目依赖规范

- 项目使用`yarn`来管理项目依赖

- 只在开发过程中使用的依赖使用`yarn add -D xxx`来安装，这个命令添加的是`devDependencies`，例如`webpack`，`babel`等。项目在运行的时候使用到的依赖使用`yarn add xxx`来安装，这个命令添加的是`dependencies`。
- 依赖的`registry`要是`https://npm.shopee.io`