

vfront开发规范

目标

本 vfront 开发规范 提供了一种统一的编码和开发规范来进行开发，这使得项目具有如下的特性：

- 使其它开发者或是团队成员更容易阅读和理解；
- 使IDE更容易理解代码，从而提供高亮、格式化等辅助功能；
- 更容易使用现有的工具；
- 容易实现缓存以及代码包的分拆；

本开发规范参考[RiotJS 编码规范](#)；

本开发规范优先参考[Vue 风格指南](#)和[iView 开发规范](#)，再遵照以上两个规范前提下，按如下开发规范进行开发。

文件规范

文件路径

vfront 框架的src文件夹为开发目录，其中有如下的文件：

```
src
|- api      # 【DEV】接口，用于发送网络请求
|- components # 【DEV】组件库，提供高级组件或业务组件
|- config    # 【ARC】配置文件
|- images    # 【DEV】图片资源，所有图片资源存放在此
|- lib       # 【ARC】系统插件，可引用后使用
|- middleware # 【ARC】中间件，自动加载
|- mock      # 【DEV】模拟数据，自动加载，生产环境移除
|- plugin    # 【DEV】插件，开发人员开发的类库或者插件都放这
|- store     # 【DEV】数据层，自动加载，按层级编写
|- styles    # 【DEV】样式库
|- template  # 【ARC】模版库，用于生成静态文件
|- views     # 【DEV】视图层，用于开发页面视图
|
|- app.vue   # 【DEV】应用入口视图
|- main.js   # 【DEV】应用入口函数
|- mock.js   # 【ARC】模拟数据的入口函数
|- router.js # 【DEV】路由的配置函数
└ vendors.js # 【ARC】架构预留
```

注意：**【DEV】表示需要开发人员开发的；【ARC】**表示架构提供，作为开发人员不应该变更里面的东西，如果遇到架构内部的问题应该及时提出给架构师修订。

文件命名

vue组件

视图和组件的文件名、命名均采用 kebab-case 的命名规范：

```
<!-- 文件名为： user-list.vue-->
<template>
...
</template>
<script>
export default {
  // 命名为 user-list
  name: 'user-list',
  data () {...}
};
</script>
```

其他文件

其他文件及文件夹一律采用全小写的命名方式，无论js、cs、html还是jpg、png。单词较长的，只能使用`.`或者`-`来区分，例如：

```
# 正确的命名  
main.js  
axios.log.js  
mock-plugin.js  
# 错误的命名  
Main.js  
axiosLog.js  
GlobalFunction.js
```

文件组织

`api` 文件夹下没有目录，只有文件。目录如下：

```
src  
└ api  
    ├ user.js # 用户模块的API  
    ├ order.js # 订单模块的API  
    └ session.js # 会话模块的API
```

`components` 和 `views` 下可以有文件，也可以有文件夹。如果是文件，则代表一个公共可复用的组件或视图，如果是文件夹，则代表由几个零件化组件构成的组件或视图，其文件夹内部的零件组件对其内部可复用但对外部不可复用，除了主页 `index.vue`。

```
src  
└ views  
    ├ login.vue      # 登陆页面  
    └ userlist  
        ├ index.vue # 用户列表的主页面，可供其他页面复用  
        └ search.vue # 用户列表的搜索项，对外部不可用  
    └ user-detail.vue # 用户用户详情，可供其他页面复用
```

`mock` 文件夹下没有目录，只有文件，所有的文件都是自动加载的：

```
src
└ mock
  ├ user.js # 用户模块数据的模拟
  ├ order.js # 订单模块数据的模拟
  └ session.js # 会话模块数据的模拟
```

- `store` 文件夹下一级文件为架构提供，其他文件需手动创建，创建规则为：文件夹为模块名，文件夹下的 `index.js` 为模块内部逻辑。如果希望模块内还有其他模块，以此类推。

变量命名

组件引用

在对视图和组件进行引用时，`变量名`按照 `PascalCase` 命名法，但 `template` 中的引用仍然按照 `kebab-case` 的写法：

```
<!-- app.vue -->
<template>
<!-- 引用 user-list.vue -->
<user-list> </user-list>
</template>
<script>
// 引用user-list.vue
import UserList from 'user-list.vue';
export default {
  components: [UserList],
  data () {...}
};
</script>
```

其他变量

- `const` 命名的全局变量一律用大写加 `kebab-case` 写法；
- `const` 命名的局部变量或导出变量一律使用 `camelCase`；
- 自定义工具函数或者类函数，使用 `PascalCase` 命名法；

- 引用的工具函数和类，保留原文档命名方法；
- 事件使用 `on-` 开头
- 其他变量和函数，使用标准的 `camelCase`；

举例：

```
// 引入的特定的类库
import qs from 'qs';
// 全局的常量
const USER_DEFAULT = { ... };
// 导出变量
export const store = { ... };
// 自定义工具函数
const Util = {
  parse () { return qs.parse },
  toString () { ... }
};
// 自定义类
function Person(name){
  this.name = name;
  return this;
};
// 其他函数或变量
Person.prototype.getName = function () {
  return this.name;
};
var personZhang = new Person('张三');
console.log(personZhang.getName());
```

组件样式规范

Vue.js 的组件是自定义元素，这非常适合用来作为样式的根作用域空间。可以将组件名作为 css 类的命名空间。使用组件名作为样式命名的前缀，可基于 BEM 或 OOCSS 范式。同时给 `style` 标签加上 `scoped` 属性。加上 `scoped` 属性编译后会给组件的 `class` 自动加上唯一的前缀从而避免样式的冲突。例如：

```
<!-- 加上scoped属性以标记作用域 -->
<style scoped>
.myApp {}
.myApp button {}
</style >
```