一、命名规则

1、文件命名

文件夹/文件的命名统一用小写 保证项目有良好的可移植性，可跨平台

2、文件引用路径

因为文件命名统一小写，引用也需要注意大小写问题

3、JavaScript变量

3.1 变量

命名方式：小驼峰

命名规范：前缀名词

命名建议：语义化

案例

```
// 友好
let maxCount = 10;
let tableTitle = 'LoginTable';
// 不友好
let setCount = 10;
let getTitle = 'LoginTable';
```

3.2 常量

命名方式：全部大写

命名规范：使用大写字母和下划线来组合命名，下划线用以分割单词

命名建议：语义化

案例

```
const MAX_COUNT = 10;
const URL = 'http://www.baidu.com';
```

3.3 函数

命名方式：小驼峰式命名法

命名规范：前缀应当为动词

命名建议：语义化

可以参考如下动词

动词	含义	返回值
can	判断是否可执行某个动作(权限)	函数返回一个布尔值。true：可执行；false：不可执行
has	判断是否含有某个值	函数返回一个布尔值。true：含有此值；false：不含有此值
is	判断是否为某个值	函数返回一个布尔值。true：为某个值；false：不为某个值
get	获取某个值	函数返回一个非布尔值
set	设置某个值	无返回值、返回是否设置成功或者返回链式对象
load	加载某些数据	无返回值或者返回是否加载完成的结果

```
// 是否可阅读
function canRead(): boolean {
  return true;
}
// 获取名称
function getName(): string {
  return this.name;
}
```

3.4 类、构造函数、组件

命名方式：大驼峰式命名法，首字母大写

命名规范：前缀为名称。

命名建议：语义化

案例

```
class Person {
  public name: string;
  constructor(name) {
    this.name = name;
  }
}
const person = new Person('mevyn');
```

公共属性和方法：跟变量和函数的命名一样。

私有属性和方法：前缀为_(下划线)，后面跟公共属性和方法一样的命名方式。

案例

```
class Person {
  private name: string;
  constructor() { }
  // 公共方法
  getName() {
    return this.name;
  }
  // 公共方法
  setName(name) {
    this._name = name;
  }
}
const person = new Person();
person.setName('mervyn');
person.getName(); // ->mervyn
```

3.5 代码风格

支持团队成员使用ESLint、TSLint等工具规范代码风格

4、css（class、id）命名规则BEM

我们还是使用大团队比较常用的BEM模式

1. class命名使用BEM其实是块（block）、元素（element）、修饰符（modifier）的缩写，利用不同的区块，功能以及样式来给元素命名。这三个部分使用__与--连接（这里用两个而不是一个是为了留下用于块儿的命名）。 命名约定的模式如下：

   ```
   .block{}
   .block__element{}
   .block--modifier{}

   block 代表了更高级别的抽象或组件
   block__element 代表 block 的后代，用于形成一个完整的 block 的整体
   block--modifier代表 block 的不同状态或不同版本
   ```

2. 文件及文件夹: 全部英文小写字母+数字或连接符"-，_"，不可出现其他字符。示例：`../css/style.css, jquery.1.x.x.js`

3. 文件：调用 `/libs` 文件需包含版本号，压缩文件需包含`min`关键词，其他插件则可不包含
   如：`/libs/jquery.1.9.1.js /libs/modernizr-1.7.min.js fileuploader.js plugins.js`

4. ID: 匈牙利命名法 & 小驼峰式命名法 如：`firstName topBoxList footerCopyright`

5. 尽量使用语义明确的单词命名，避免 `left bottom` 等方位性的词语

二、注释

1.单行注释

```
// 这个函数的执行条件，执行结果大概说明
dosomthing()
```

2.多行注释

```
/*
* xxxx  描述较多的时候可以使用多行注释
* xxxx
*/
dosomthing();
```

3.函数(方法)注释

参考：https://www.html.cn/doc/jsdoc/

支持团队成员使用IDE插件简化此类规范

注释名	语法	含义	示例
@param	@param 参数名 {参数类型} 描述信息	描述参数的信息	@param name {String} 传入名称
@return	@return {返回类型} 描述信息	描述返回值的信息	@return {Boolean} true:可执行;false:不可执行
@author	@author 作者信息 [附属信息：如邮箱、日期]	描述此函数作者的信息	@author 张三 2015/07/21
@version	@version XX.XX.XX	描述此函数的版本号	@version 1.0.3
@example	@example 示例代码	演示函数的使用	@example setTitle('测试')

4.文件注释

在文件开头撰写的注释，用于描述作者、时间、文件内容等

支持团队成员使用IDE插件简化此类规范

```
<!--
 * @Description: Ruebin edited
 * @Author: Ruebin
 * @Date: 2019-10-23 21:03:33
 * @LastEditTime: 2019-11-05 20:03:01
 * @LastEditors: Ruebin
 -->
```

三、开发规范

1. 文件目录

1.1 Vue

```
src                                 源码目录
|-- config                          配置文件
|-- docs                            文档
|-- api                             接口及其封装类，统一管理
|-- assets                          静态资源，统一管理
| |--font
| |--css
| |--js
| |--images
|-- components                      公用组件，全局文件
|-- filters                         过滤器，全局工具
|-- utils                           工具类
|-- datas                           模拟数据，临时存放
|-- lib                             外部引用的插件存放及修改文件
|-- mock                            模拟接口，临时存放
|-- router                          路由，统一管理
|-- store                           vuex，统一管理
|-- views                           视图目录
|   |-- staffWorkbench              视图模块名
|   |-- |-- staffWorkbench.vue     模块入口页面
|   |-- |-- indexComponents        模块页面级组件文件夹
|   |-- |-- components             模块通用组件文件夹
```

1.2 JS

1.2.1 JS 文件结构

```
---/js/
|---- /libs/plugin-1/       使用到的js插件1
|---- /libs/plugin-2/       使用到的js插件2
|---- /libs/plugin-3/       使用到的js插件3
|---- script.js             单独书写的js
```

```
|---- plugins.js            调用的plugins汇总
|---- jquery-1.9.x.min.js   调用jq库文件
```

1.3 CSS

1.3.1 CSS 文件结构

```
--- ../css/
 |---- css/libs/reset.css                 CSS reset文件
 |---- … …
 |---- css/images/                        主 CSS-sprite 图片
 |---- css/style.css                      主 CSS 样式表
 |---- … …
 |---- css/images/xxx/sprite.png          xxx 的 CSS-sprite 图片
 |---- css/xxx-style.css                  xxx 的 样式表
```

1.3.2 CSS(含Less) 文件结构

```
--- ../css/
 |---- css/libs/reset.less           CSS reset文件
 |---- css/libs/elements.less        Less 函数复用文件
 |---- … …
 |---- css/images/                        主 CSS-sprite 图片
 |---- css/style.less                     主样式Less
 |---- css/style.css                      less -> css 自动生成
 |---- … …
 |---- css/images/xxx/sprite.png          xxx 的 CSS-sprite 图片
 |---- css/xxx-style.less                 xxx 的 Less
 |---- css/xxx-style.css                  less -> css 自动生成
```

2. git规范

2.1 分支命名

- master: master 分支就叫master 分支，线上环境正在使用的，每一次更新master都需要打tag
- test: test分支就供测试环境使用的分支
- develop: develop 分支就叫develop 分支
- feature: feature 分支 咱们暂时可以按 feature/name/version 这种命名规范来，后面有更好的命名规范咱们再改。version 表示 当前迭代的版本号，name 表示当前迭代的名称
- hotfix: hotfix 分支的命名我们暂时可以按 hotfix/name/version 这种来进行命名,verison表示这次修复的版本的版本号，name表示本次热修复的内容标题 (斜杠的方式 在source-tree中有归类的作用)

2.2 提交模版

commit的内容要包括Header、Body和Footer

```
<type>(<scope>): <subject>
<空行>
<body>
<空行>
<footer>
```

- type：用于说明commit的类别，规定为如下几种
    - feat：新增功能；
    - fix：修复bug；
    - docs：修改文档；
    - refactor：代码重构，未新增任何功能和修复任何bug；
    - build：改变构建流程，新增依赖库、工具等（例如webpack修改）；
    - style：仅仅修改了空格、缩进等，不改变代码逻辑；
    - perf：改善性能和体现的修改；
    - chore：非src和test的修改；
    - test：测试用例的修改；
    - ci：自动化流程配置修改；
    - revert：回滚到上一个版本；
- scope：【可选】用于说明commit的影响范围
- subject：commit的简要说明，尽量简短
- Body
    - 对本次commit的详细描述，可分多行
- 尾部（Footer）
    - 不兼容变动：需要描述相关信息
    - 关闭指定Issue：输入Issue信息

2.3 Tag

采用三段式，v版本.里程碑.序号，如v1.2.1

- 架构升级或架构重大调整，修改第2位
- 新功能上线或者模块大的调整，修改第2位
- bug修复上线，修改第3位

3. Vue组件

每个 Vue 组件的代码建议精简，一般来说如果超出200行代码，建议拆分组件，但这不是强制规定 组件一般情况下是可以拆成基础/ui部分和业务部分，基础组件一般是承载呈现，基础功能，不和业务耦合部分。 业务组件一般包含业务功能业务特殊数据等等

3.1 UI组件/基础组件

和当前业务耦合性比较低

开发的时候注意可拓展性，支持数据传参进行渲染，支持插槽slot

设置有mixin，mixin中放了基础信息和方法

3.2 容器组件

和当前业务耦合性比较高，由多个基础组件组成，可承载当前页的业务接口请求和数据(vuex)

3.3 组件存放位置

1. ui组件存放在src/components/ 中 包含xxx.vue和 xxmixin.js 和 readme.md

```
xxx.vue 表示ui部分
xxmixin.js 表示js部分
readme.md 中描述组件的基本信息
引用组件的时候 直接引入xxmixin.js 即可。在引用组件的页面可以对mixin里面的方法进行重构
```

名词	含义	案例
@name	组件名称	筛选下拉框
@version	版本	v1.0.0
@updateTime	更新日期	2018.09.18
@describe	使用场景描述	某某场景下
@props	参数	['data']
@author	作者	dd

2. 业务组件就放在业务模块部分即可

3.4 组件通讯

- 避免数据的分发源混乱，不建议使用eventBus控制数据
- 应使用props来和$emit来进行简单父子组件数据分发和传送
- 可使用状态管理，但项目负责人必须先规划好哪些模块数据需要用到状态管理，而不是无组织地在Vuex中定义变量

3.5 组件的挂载和销毁

（1）通过v-if控制组件的挂载和销毁

```
<testcomponent v-if='componentActive'> </testcomponent>
```

（2）通过is控制组件的挂在和销毁

```
<component is='componentName'> </component>
```

3.6 跨项目组件共用

公共组件存放在位置src/components/，定时抽取共用次数多的组件将其上传到npm，供下载引用

4. Vue开发

4.1 组件名

组件名为多个单词，并且用连接线（-）连接，避免与 HTML 标签冲突，并且结构更加清晰。

示例：

```
// 反例
export default {
    name: 'item'
}

// 正例
export default {
    name: 'page-article-item'
}
```

4.2 组件文件

组件的名字应该始终是以连接线（-）连接的单词，一方面可与组件名一致，使项目更加清晰，另一方面这样的写法对编辑器引入也很友好。

示例：

```
// 反例
├── index.html
├── main.js
└── components
    ├── pageheader
    ├── pagearticle
    └── pageheader

// 正例
├── index.html
├── main.js
└── components
    ├── page-header
    ├── page-article
    └── page-header
```

对于例如按钮、下拉框或表格这样的基础组件应该始终以一个特定的前缀开头，区别与其他业务组件。

示例：

```
// 反例
├── index.html
├── main.js
└── components
    ├── page-header
    ├── page-article
    ├── page-header
    ├── mazey-button
    ├── mazey-select
    └── mazey-table

// 正例
├── index.html
├── main.js
└── components
|   ├── page-header
|   ├── page-article
|   └── page-header
└── base
    ├── mazey-button
    ├── mazey-select
    └── mazey-table
```

4.3 Prop

定义 Prop 的时候应该始终以驼峰格式（camelCase）命名，在父组件赋值的时候使用连接线（-）。这里遵循每个语言的特性，因为在 HTML 标记中对大小写是不敏感的，使用连接线更加友好；而在 JavaScript 中更自然的是驼峰命名。

示例：

```
// 反例
// Vue
props: {
    article-status: Boolean
}
// HTML
<article-item :articleStatus="true"></article-item>

// 正例
// Vue
props: {
    articleStatus: Boolean
}
```

```
// HTML
<article-item :article-status="true"></article-item>
```

Prop 的定义应该尽量详细的指定其类型、默认值和验证。

示例：

```
// 反例
props: ['attrM', 'attrA', 'attrZ']

// 正例
props: {
    attrM: Number,
    attrA: {
        type: String,
        required: true
    },
    attrZ: {
        type: Object,
        // 数组/对象的默认值应该由一个工厂函数返回
        default: function () {
            return {
                msg: '成就你我'
            }
        }
    },
    attrE: {
        type: String,
        validator: function (v) {
            return !(['success', 'fail'].indexOf(v) === -1)
        }
    }
}
```

4.4 v-for

在执行 v-for 遍历的时候，总是应该带上 key 值使更新 DOM 时渲染效率更高（官方规定）。

示例：

```
// 反例
<ul>
    <li v-for="item in list">
        {{ item.title }}
    </li>
</ul>

// 正例
<ul>
```

```
    <li v-for="item in list" :key="item.id">
        {{ item.title }}
    </li>
</ul>
```

v-for 应该避免与 v-if 在同一个元素（例如：``）上使用，因为 v-for 的优先级比 v-if 更高，为了避免无效计算和渲染，应该尽量将 v-if 放到容器的父元素之上。

示例：

```
// 反例
<ul>
    <li v-for="item in list" :key="item.id" v-if="showList">
        {{ item.title }}
    </li>
</ul>

// 正例
<ul v-if="showList">
    <li v-for="item in list" :key="item.id">
        {{ item.title }}
    </li>
</ul>
```

4.5 v-if / v-else-if / v-else

若同一组 v-if 逻辑控制中的元素逻辑相同，Vue 为了更高效的元素切换，会复用相同的部分，例如：`value`。为了避免复用带来的不合理效果，应该在同种元素上加上 key 做标识。

示例：

```
// 反例
<div v-if="hasData">
    <span>{{ mazeyData }}</span>
</div>
<div v-else>
    <span>无数据</span>
</div>

// 正例
<div v-if="hasData" key="mazey-data">
    <span>{{ mazeyData }}</span>
</div>
<div v-else key="mazey-none">
    <span>无数据</span>
</div>
```

4.6 指令缩写

为了统一规范始终使用指令缩写，使用`v-bind`，`v-on`并没有什么不好，这里仅为了统一规范。

示例：

```
<input :value="mazeyUser" @click="verifyUser">
```

四、codeReview

1. 规则 所有影响到以往流程的功能需求更改发版前都需要codeReview（代码复审）
2. 执行者 应该由项目负责人或者能力更强、经验丰富的项目成员执行codeReview
3. 反馈 每次codereView都需要有反馈，要对本次codeReview负责

   反馈内容基本如

   ```
   功能：本次主要是修改了什么功能或者bug
   模块：本次发版影响的模块
   代码问题：codereview过程中发现的代码问题，比如代码性能，写法，代码风格等等
   业务问题：比如发现了某某影响到其他模块的逻辑问题，如果没有发现就写:无
   ```