前端代码规范(PC端Vue2.x)

规范目的

- 命名规范
- 结构化规范
- 注释规范
- 编码规范

规范目的

1. 为提高团队协作效率
2. 便于后台人员添加功能及前端后期优化维护
3. 输出高质量的文档

命名规范

1. 为了让大家书写可维护的代码，而不是一次性的代码
2. 让团队当中其他人看你的代码能一目了然
3. 甚至一段时间时候后你再看你某个时候写的代码也能看

普通变量命名规范

命名方法： 驼峰命名法
命名规范：

1. 命名必须是跟字段内容相关的词，比如说我想申明一个变量，用来表示‘我的学校’，那么我们可以这样定义 `const mySchool = "我的学校"`；
2. 命名是复数的时候需要加s,比如说我想申明一个数组，表示很多人的名字，那么我们可以这样定义 `const names = new Array()`；

常量

命名方法：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。

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

例外情况

1. 作用域不大临时变量可以简写，比如：str，num，bol，obj，fun，arr。
2. 简写字段应具有该变量特有的属性，如 const num = 2，禁止 const num = ’2’
3. 循环变量可以简写，比如：i，j，k 等。

组件命名规范

官方文档推荐及使用遵循规则：
PascalCase (单词首字母大写命名)是最通用的**声明**约定
kebab-case (短横线分隔命名) 是最通用的**使用**约定

- 组件名应该始终是多个单词的，根组件 App 除外
- 有意义的名词、简短、具有可读性 ，让阅读者能够一眼了解该组件的作用。
- 必须以纯英文字母/缩写命名,严禁使用拼音、英普混用、数字的等。

命名遵循 PascalCase 约定

  - 公用组件以 Abcd (公司名缩写、约定项目名简称) 开头，如（`JcDatePicker,Jc Table`）
  - 页面内部组件以组件模块名简写为开头，Item 为结尾，如（`StaffBenchToChargeItem, StaffBenchAppNotArrItem`）
-

使用遵循 kebab-case 约定

  - 在页面中使用组件需要前后闭合，并以短线分隔，如（`<jc-date-picker></jc-date-picker>, <jc-table></jc-table>`）
-
- 导入及注册组件时，遵循 PascalCase 约定
- 同时还需要注意：必须符合自定义元素规范: 切勿使用保留字（如default、export等）。

method 方法命名命名规范

- 驼峰式命名，统一使用动词或者动词+名词形式

```
//bad
go、nextPage、show、open、login

// good
jumpPage、openCarInfoDialog
```

- 请求数据方法，以 data 结尾

```
//bad
takeData、confirmData、getList、postForm

// good
getListData、postFormData
```

- init、refresh 单词除外
- 尽量使用常用单词开头（set、get、go、can、has、is）

附： 函数方法常用的动词:

```
get 获取    /set 设置
add 增加    /remove 删除
create 创建    /destory 移除
start 启动    /stop 停止
open 打开    /close 关闭
```

```
read 读取   /write 写入
load 载入   /save 保存,
create 创建   /destroy 销毁
begin 开始   /end 结束
backup 备份   /restore 恢复
import 导入   /export 导出
split 分割   /merge 合并
inject 注入   /extract 提取
attach 附着   /detach 脱离
bind 绑定   /separate 分离
view 查看   /browse 浏览
edit 编辑   /modify 修改
select 选取   /mark 标记
copy 复制   /paste 粘贴
undo 撤销   /redo 重做
insert 插入   /delete 移除
add 加入   /append 添加
clean 清理   /clear 清除
index 索引   /sort 排序
find 查找   /search 搜索
increase 增加   /decrease 减少
play 播放   /pause 暂停
launch 启动   /run 运行
compile 编译   /execute 执行
debug 调试   /trace 跟踪
observe 观察   /listen 监听
build 构建   /publish 发布
input 输入   /output 输出
encode 编码   /decode 解码
encrypt 加密   /decrypt 解密
compress 压缩   /decompress 解压缩
pack 打包   /unpack 解包,
parse 解析   /emit 生成
connect 连接   /disconnect 断开
send 发送   /receive 接收
download 下载/upload 上传
refresh 刷新   /synchronize 同步
update 更新   /revert 复原
lock 锁定   /unlock 解锁
check out 签出   /check in 签入
submit 提交   /commit 交付
push 推   /pull 拉
expand 展开   /collapse 折叠
begin 起始   /end 结束
start 开始   /finish 完成
enter 进入   /exit 退出
abort 放弃   /quit 离开
obsolete 废弃   /depreciate 废旧
collect 收集   /aggregate 聚集
```

views 下的文件命名

- 尽量是名词,且使用驼峰命名法
- 开头的单词就是所属模块名字（workbenchIndex、workbenchList、workbenchEdit）
- 名字至少两个单词（good: workbenchIndex）（bad:workbench）

props 命名

在声明 prop 的时候，其命名应该始终使用 camelCase，而在模板中应该始终使用 kebab-case

```
<!-- bad -->
<script>
props: {
  'greeting-text': String
}
</script>

<welcome-message greetingText="hi"></welcome-message>

<!-- good -->
<script>
props: {
  greetingText: String
}
</script>

<welcome-message greeting-text="hi"></welcome-message>
```

Props 规范

Props 定义应该尽量详细

```
// bad
props: ['status']

// good
props: {
  status: {
    type: String,
    required: true,
    validator: function (value) {
      return [
        'syncing',
        'synced',
        'version-conflict',
        'error'
      ].indexOf(value) !== -1
```

```
    }
  }
}
```

结构化规范

目录文件夹及子文件规范

- 以下统一管理处均对应相应模块
- 以下全局文件文件均以 index.js 导出，并在 main.js 中导入
- 以下临时文件，在使用后，接口已经有了，发版后清除

```
src                                 源码目录
|-- api                                 接口，统一管理
|-- assets                              静态资源，统一管理
|-- components                          公用组件，全局文件
|-- filters                             过滤器，全局工具
|-- icons                               图标，全局资源
|-- 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
|   |-- |-- staffWorkbenchIndex.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
复制代码
```

vue 文件基本结构

```
  <template>
    <div>
      <!--必须在div中编写页面-->
    </div>
  </template>
  <script>
    export default {
      components : {
      },
      data () {
        return {
        }
      },
```

```
      mounted() {
      },
      methods: {
      }
   }
  </script>
  <!--声明语言，并且添加scoped-->
  <style lang="scss" scoped>
  </style>
复制代码
```

多个特性的元素规范

多个特性的元素应该分多行撰写，每个特性一行。(增强更易读)

```
<!-- bad -->
<img src="https://vuejs.org/images/logo.png" alt="Vue Logo">
<my-component foo="a" bar="b" baz="c"></my-component>

<!-- good -->
<img
  src="https://vuejs.org/images/logo.png"
  alt="Vue Logo"
>
<my-component
  foo="a"
  bar="b"
  baz="c"
>
</my-component>
```

注释规范

代码注释在一个项目的后期维护中显的尤为重要，所以我们要为每一个被复用的组件编写组件使用说明，为组件中每一个方法编写方法说明

务必添加注释列表

1. 公共组件使用说明
2. 各组件中重要函数或者类说明
3. 复杂的业务逻辑处理说明
4. 特殊情况的代码处理说明,对于代码中特殊用途的变量、存在临界值、函数中使用的 hack、使用了某种算法或思路等需要进行注释描述
5. 多重 if 判断语句
6. 注释块必须以 `/**（至少两个星号）开头**/`
7. 单行注释使用//

单行注释

注释单独一行，不要在代码后的同一行内加注释。例如：

```
bad

var name =”abc”; // 姓名

good

// 姓名
var name = “abc”;
```

多行注释

```
组件使用说明，和调用说明
      /**
      * 组件名称
      * @module 组件存放位置
      * @desc 组件描述
      * @author 组件作者
      * @date 2017年12月05日17:22:43
      * @param {Object} [title]    - 参数说明
      * @param {String} [columns] - 参数说明
      * @example 调用示例
      *  <jc-table :title="title" :columns="columns" :table-data="tableData'
      **/
```

编码规范

优秀的项目源码，即使是多人开发，看代码也如出一人之手。统一的编码规范，可使代码更易于阅读，易于理解，易于维护。尽量按照 ESLint 格式要求编写代码

源码风格

使用 ES6 风格编码

1. 定义变量使用 let ,定义常量使用 const
2. 静态字符串一律使用单引号或反引号，动态字符串使用反引号

```
// bad
const a = 'foobar'
const b = 'foo' + a + 'bar'

// acceptable
const c = `foobar`

// good
const a = 'foobar'
```

```
  const b = `foo${a}bar`
  const c = 'foobar'
复制代码
```

1. 解构赋值

- 数组成员对变量赋值时，优先使用解构赋值

```
  // 数组解构赋值
  const arr = [1, 2, 3, 4]
  // bad
  const first = arr[0]
  const second = arr[1]

  // good
  const [first, second] = arr
复制代码
```

- 函数的参数如果是对象的成员，优先使用解构赋值

```
  // 对象解构赋值
  // bad
  function getFullName(user) {
    const firstName = user.firstName
    const lastName = user.lastName
  }

  // good
  function getFullName(obj) {
    const { firstName, lastName } = obj
  }

  // best
  function getFullName({ firstName, lastName }) {}
```

拷贝数组

1. 使用扩展运算符（...）拷贝数组。

```
  const items = [1, 2, 3, 4, 5]

  // bad
  const itemsCopy = items

  // good
  const itemsCopy = [...items]
```

1. 模块

- 如果模块只有一个输出值，就使用 export default，如果模块有多个输出值，就不使用 export default，export default 与普通的 export 不要同时使用

```
// bad
import * as myObject from './importModule'

// good
import myObject from './importModule'
```

- 如果模块默认输出一个函数，函数名的首字母应该小写。

```
function makeStyleGuide() {
}

export default makeStyleGuide;
```

- 如果模块默认输出一个对象，对象名的首字母应该大写。

```
const StyleGuide = {
  es6: {
  }
};

export default StyleGuide;
```
复制代码

指令规范

1. 指令有缩写一律采用缩写形式

```
// bad
v-bind:class="{'show-left'：true}"
v-on:click="getListData"

// good
:class="{'show-left'：true}"
@click="getListData"
```

1. v-for 循环必须加上 key 属性，在整个 for 循环中 key 需要唯一

```
<!-- good -->
<ul>
  <li v-for="todo in todos" :key="todo.id">
    {{ todo.text }}
  </li>
</ul>
```

```
<!-- bad -->
<ul>
  <li v-for="todo in todos">
    {{ todo.text }}
  </li>
</ul>
```

避免 v-if 和 v-for 同时用在一个元素上（性能问题）

1. 以下为两种解决方案：

- 将数据替换为一个计算属性，让其返回过滤后的列表

```
<!-- bad -->
<ul>
  <li v-for="user in users" v-if="user.isActive" :key="user.id">
    {{ user.name }}
  </li>
</ul>

<!-- good -->
<ul>
  <li v-for="user in activeUsers" :key="user.id">
    {{ user.name }}
  </li>
</ul>

<script>
computed: {
  activeUsers: function () {
    return this.users.filter(function (user) {
      return user.isActive
    })
  }
}
</script>
```

- 将 v-if 移动至容器元素上 (比如 ul, ol)

```
<!-- bad -->
<ul>
  <li v-for="user in users" v-if="shouldShowUsers" :key="user.id">
    {{ user.name }}
  </li>
</ul>

<!-- good -->
<ul v-if="shouldShowUsers">
  <li v-for="user in users" :key="user.id">
```

```
      {{ user.name }}
    </li>
  </ul>
```

其他

1. 避免 this.$parent
2. 调试信息 console.log() debugger 使用完及时删除
3. 除了三目运算，if,else 等禁止简写

```
// bad
if (true)
    alert(name);
console.log(name);

// bad
if (true)
alert(name);
console.log(name)

// good
if (true) {
    alert(name);
}
console.log(name);
```