## 前端规范

目录

**[CSS规范](#_Toc14763_WPSOffice_Level1)** **[2](#_Toc14763_WPSOffice_Level1)**

**[JS规范](#_Toc6155_WPSOffice_Level1)** **[3](#_Toc6155_WPSOffice_Level1)**

**[Vue规范](#_Toc30625_WPSOffice_Level1)** **[4](#_Toc30625_WPSOffice_Level1)**

**[检测机制](#_Toc26019_WPSOffice_Level1)** **[10](#_Toc26019_WPSOffice_Level1)**

# BEM--源自Yandex的CSS 命名规范（实验）

在计算机科学领域，只有2个非常难解决的问题：一个是缓存失效，而另一个则是命名，所以命名是很重要的。

BEM - Block Element Modfier（块元素编辑器）是一个很有用的方法，它可以帮助你创建出可以复用的前端组件和前端代码，BEM 最大的作用是为了让 CSS 会说话

它有如下3个特性：

* 易用性 如果想使用BEM，你只需要采用BEM命名规范即可
* 单元性 独立的块和CSS选择器，可以使得你的代码可重用和单元化
* 灵活性 使用BEM之后，方法和工具可以按照你喜欢的方式去组织和配置

下面将从2个方面来分析BEM到底是什么？

* 介绍  
  BEM是一个高可用的，强大的，而且简单的命名规范，它可以使得你的前端代码更加易读和理解，容易与他人协作，容易扩展，更加强壮和明确，关键是更加严谨。
* 命名  
  BEM的方法，可以确保参与网站开发的每一个人，都能够使用一个代码库并且使用同一种语言。使用BEM格式的命名规范，可以从容应对需求变更。

详情：<https://www.jianshu.com/p/339fdb93e155>

**esLint 推荐规则--js**

Eslint关于js的规则分为两类

1. 推荐的规则（eslint:recommended）

推荐的规则在这里 <https://cn.eslint.org/docs/rules/>

1. 单独配置的规则

单独配置的规则：

禁用 console

强制把变量的使用限制在其定义的作用域范围内

要求 switch 语句中有 default 分支

禁用 eval()

禁止使用类似 eval() 的方法

禁用 \_\_proto\_\_ 属性

禁用一成不变的循环条件

禁止使用不带 await 表达式的 async 函数

**Vue规范**

1. 组件名称为多个单词

Good:SearchBar,MenuNav

Bad：Search,Menu

这样做可以避免跟现有的以及未来的 HTML 元素[相冲突](https://w3c.github.io/webcomponents/spec/custom/" \l "valid-custom-element-name" \t "https://cn.vuejs.org/v2/style-guide/_blank)，因为所有的 HTML 元素名称都是单个单词的

1. 组件的 data 必须是一个函数

Good:

Vue.component('some-comp', {

data: function () {

return {

foo: 'bar'

}

}

})

Bad:

Vue.component('some-comp', {

data: {

foo: 'bar'

}

})

我们希望每个组件实例都管理其自己的数据。为了做到这一点，每个实例必须生成一个独立的数据对象。

1. Prop 定义应该尽量详细

Good:

Prop:{

status:String

}

//更好的做法

props: {

status: {

type: String,

required: true,

validator: function (value) {

return [

'syncing',

'synced',

'version-conflict',

'error'

].indexOf(value) !== -1

}

}

}

Bad:

Prop:['status']

细致的 [prop 定义](https://cn.vuejs.org/v2/guide/components-props.html" \l "Prop-%E9%AA%8C%E8%AF%81)有两个好处：

它们写明了组件的 API，所以很容易看懂组件的用法

在开发环境下，如果向一个组件提供格式不正确的 prop，Vue 将告警，以帮助你捕获潜在的错误来源。

1. 总是用 key 配合 v-for

在更新 DOM 的时候，Vue 将会优化渲染把可能的 DOM 变动降到最低

1. 永远不要把 v-if 和 v-for 同时用在同一个元素上

Good:

<ul>

<li

v-for="user in activeUsers"

:key="user.id"

>

{{ user.name }}

</li>

</ul>

Bad:

<ul>

<li

v-for="user in users"

v-if="user.isActive"

:key="user.id">

{{ user.name }}

</li>

</ul>

v-for比v-if优先，如果每一次都需要遍历整个数组，将会影响速度，尤其是当之需要渲染很小一部分的时候

1. 为组件样式设置作用域

Good:

<style scoped>

Bad:

<style>

如果你和其他开发者一起开发一个大型工程，或有时引入三方 HTML/CSS (比如来自 Auth0)，设置一致的作用域会确保你的样式只会运用在它们想要作用的组件上

1. [单文件组件](https://cn.vuejs.org/v2/guide/single-file-components.html)的文件名应该要么始终是单词大写开头

Good:

components/

|- MyComponent.vue

Bad:

components/

|- myComponent.vue

单词大写开头对于代码编辑器的自动补全最为友好

1. 基础组件使用前缀Base+组件名进行命名

Good:

components/

|- BaseButton.vue

|- BaseTable.vue

|- BaseIcon.vue

Bad:

components/

|- MyButton.vue

|- VueTable.vue

|- Icon.vue

基础组件是指没有业务逻辑，没有依赖全局状态(比如vuex)，这样的好处是再编辑器中基础组件会全部列在一起，更容易识别,更容易管理。

1. 和父组件紧密耦合的子组件应该以父组件名作为前缀命名

Good:

Project.vue/

|-components/

|- ProjectButton.vue

|- ProjectTable.vue

|- ProjectIcon.vue

Bad:

Project.vue/

|-components/

|- Button.vue

|- Table.vue

|- Icon.vue

如果一个组件只在某个父组件的场景下有意义，这层关系应该体现在其名字上。因为编辑器通常会按字母顺序组织文件，所以这样做可以把相关联的文件排在一起

1. 组件名应该倾向于完整单词而不是缩写

编辑器中的自动补全已经让书写长命名的代价非常之低了，而其带来的明确性却是非常宝贵的。不常用的缩写尤其应该避免

1. 组件模板应该只包含简单的表达式，复杂的表达式则应该重构为计算属性或方法

Good:

<!-- 在模板中 -->

{{ normalizedFullName }}

// 复杂表达式已经移入一个计算属性

computed: {

normalizedFullName: function () {

return this.fullName.split(' ').map(function (word) {

return word[0].toUpperCase() + word.slice(1)

}).join(' ')

}

}

Bad:

{{

fullName.split(' ').map(function (word) {

return word[0].toUpperCase() + word.slice(1)

}).join(' ')

}}}}

复杂表达式会让你的模板变得不那么声明式。我们应该尽量描述应该出现的是什么，而非如何计算那个值。而且计算属性和方法使得代码可以重用

1. 复杂计算属性分割为尽可能多的更简单的属性

Good:

computed: {

basePrice: function () {

return this.manufactureCost / (1 - this.profitMargin)

},

discount: function () {

return this.basePrice \* (this.discountPercent || 0)

},

finalPrice: function () {

return this.basePrice - this.discount

}

}

Bad:

computed: {

price: function () {

var basePrice = this.manufactureCost / (1 - this.profitMargin)

return (

basePrice -

basePrice \* (this.discountPercent || 0)

)

}

}

**易于测试**

当每个计算属性都包含一个非常简单且很少依赖的表达式时，撰写测试以确保其正确工作就会更加容易。

**易于阅读**

简化计算属性要求你为每一个值都起一个描述性的名称，即便它不可复用。这使得其他开发者 (以及未来的你) 更容易专注在他们关心的代码上并搞清楚发生了什么。

**更好的“拥抱变化”**

任何能够命名的值都可能用在视图上。举个例子，我们可能打算展示一个信息，告诉用户他们存了多少钱；也可能打算计算税费，但是可能会分开展现，而不是作为总价的一部分。

小的、专注的计算属性减少了信息使用时的假设性限制，所以需求变更时也用不着那么多重构了。

1. HTML 特性值应该始终带引号

Good:

<input type=text>

<AppSidebar :style={width:sidebarWidth+'px'}>

Bad:

<input type="text">

<AppSidebar :style="{width:sidebarWidth+'px'}">

在 HTML 中不带空格的特性值是可以没有引号的，但这鼓励了大家在特征值里不写空格，导致可读性变差

1. 指令缩写 (用 : 表示 v-bind: 、用 @ 表示 v-on: 和用 # 表示 v-slot:) 应该要么都用要么都不用

要么页面全部指令都使用全称，要么都使用缩写，不要混合使用。混合使用导致页面不美观，很乱。

1. 元素选择器应该避免在 scoped 中出现

在 scoped 样式中，类选择器比元素选择器更好，因为大量使用元素选择器是很慢的

1. 应该优先通过 prop 和事件进行父子组件之间的通信，而不是 this.$parent 或改变 prop

一个理想的 Vue 应用是 prop 向下传递，事件向上传递的。遵循这一约定会让你的组件更易于理解，数据流向更加简洁。

规范来源于官方文档，整理了一些适用于新后台的规范

<https://cn.vuejs.org/v2/style-guide/>

**检测机制**

目前js规范和vue规范（部分）可以通过eslint检测，每次提交pr的代码必须通过eslint检测。