## vue代码规范文档

## 规则等级

1. 必要的

这些规则会帮你规避错误，所以学习并接受它们带来的全部代价吧。这里面可能存在例外，但应该非常少，且只有你同时精通 JavaScript 和 Vue 才可以这样做。

1. 强烈推荐

这些规则能够在绝大多数工程中改善可读性和开发体验。即使你违反了，代码还是能照常运行，但例外应该尽可能少且有合理的理由。

1. 推荐

当存在多个同样好的选项，选任意一个都可以确保一致性。在这些规则里，我们描述了每个选项并建议一个默认的选择。也就是说只要保持一致且理由充分，你可以随意在你的代码库中做出不同的选择。请务必给出一个好的理由！通过接受社区的标准，你将会：

1. 训练你的大脑，以便更容易的处理你在社区遇到的代码；
2. 不做修改就可以直接复制粘贴社区的代码示例；
3. 能够经常招聘到和你编码习惯相同的新人，至少跟 Vue   
    相关的东西是这样的。
4. 谨慎使用

有些 Vue 特性的存在是为了照顾极端情况或帮助老代码的平稳迁移。当被过度使用时，这些特性会让你的代码难于维护甚至变成 bug 的来源。这些规则是为了给有潜在风险的特性敲个警钟，并说明它们什么时候不应该使用以及为什么。

# 必要的

1. 组件名：

**组件名应该始终是多个单词的，防止冲突**

1. 组件数据：

组件的 data 必须是一个函数。

1. Prop定义：

在你提交的代码中，prop 的定义应该尽量详细，至少需要指定其类型。

1. 为 v-for 设置键值：

在组件上总是必须用 key 配合 v-for，以便维护内部组件及其子树的状态。甚至在元素上维护可预测的行为，比如动画中的对象固化 (object constancy)，也是一种好的做法。

1. 避免 v-if 和 v-for 用在一起：

为了过滤一个列表中的项目 (比如 v-for="user in users" v-if="user.isActive")。在这种情形下，请将 users 替换为一个计算属性 (比如 activeUsers)，让其返回过滤后的列表

1. 为组件样式设置作用域：

这条规则只和单文件组件有关。你不一定要使用 scoped 特性。设置作用域也可以通过 CSS Modules，那是一个基于 class 的类似 BEM 的策略，当然你也可以使用其它的库或约定。

1. 私有属性名：

使用模块作用域保持不允许外部访问的函数的私有性。如果无法做到这一点，就始终为插件、混入等不考虑作为对外公共 API 的自定义私有属性使用 $\_ 前缀。并附带一个命名空间以回避和其它作者的冲突 (比如 $\_yourPluginName\_)。

# 强烈推荐 (增强可读性)

1. 组件文件:

只要有能够拼接文件的构建系统，就把每个组件单独分成文件。

1. 单文件组件文件的大小写:

单文件组件的文件名应该要么始终是单词大写开头 (PascalCase)，要么始终是横线连接 (kebab-case)。

1. 基础组件名:

应用特定样式和约定的基础组件 (也就是展示类的、无逻辑的或无状态的组件) 应该全部以一个特定的前缀开头，比如 Base、App 或 V。

1. 单例组件名:

只应该拥有单个活跃实例的组件应该以 The 前缀命名，以示其唯一性。

这不意味着组件只可用于一个单页面，而是每个页面只使用一次。这些组件永远不接受任何 prop，因为它们是为你的应用定制的，而不是它们在你的应用中的上下文。如果你发现有必要添加 prop，那就表明这实际上是一个可复用的组件，只是目前在每个页面里只使用一次。

1. 紧密耦合的组件名:

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。

1. 组件名中的单词顺序:

组件名应该以高级别的 (通常是一般化描述的) 单词开头，以描述性的修饰词结尾。

1. 自闭合组件:

在单文件组件、字符串模板和 JSX 中没有内容的组件应该是自闭合的——但在 DOM 模板里永远不要这样做。

1. 模板中的组件名大小写:

对于绝大多数项目来说，在单文件组件和字符串模板中组件名应该总是 PascalCase 的——但是在 DOM 模板中总是 kebab-case 的。

1. JS/JSX 中的组件名大小写:

JS/JSX 中的组件名应该始终是 PascalCase 的，尽管在较为简单的应用中只使用 Vue.component 进行全局组件注册时，可以使用 kebab-case 字符串。

1. 完整单词的组件名:

组件名应该倾向于完整单词而不是缩写。

1. Prop 名大小写:

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

1. 多个特性的元素:

多个特性(属性)的元素应该分多行撰写，每个特性一行。

1. 模板中简单的表达式：

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。

1. 简单的计算属性：

应该把复杂计算属性分割为尽可能多的更简单的属性。

1. 带引号的特性(属性)值：

非空 HTML 特性值应该始终带引号 (单引号或双引号，选你 JS 里不用的那个)。

1. 指令缩写

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。

# 推荐 (将选择和认知成本最小化)

1. 组件/实例的选项的顺序

副作用 (触发组件外的影响)

el

全局感知 (要求组件以外的知识)

name

parent

组件类型 (更改组件的类型)

functional

模板修改器 (改变模板的编译方式)

delimiters

comments

模板依赖 (模板内使用的资源)

components

directives

filters

组合 (向选项里合并属性)

extends

mixins

接口 (组件的接口)

inheritAttrs

model

props/propsData

本地状态 (本地的响应式属性)

data

computed

事件 (通过响应式事件触发的回调)

watch

生命周期钩子 (按照它们被调用的顺序)

beforeCreate

created

beforeMount

mounted

beforeUpdate

updated

activated

deactivated

beforeDestroy

destroyed

非响应式的属性 (不依赖响应系统的实例属性)

methods

渲染 (组件输出的声明式描述)

template/render

renderError

1. 元素特性的顺序:

定义 (提供组件的选项)

is

列表渲染 (创建多个变化的相同元素)

v-for

条件渲染 (元素是否渲染/显示)

v-if

v-else-if

v-else

v-show

v-cloak

渲染方式 (改变元素的渲染方式)

v-pre

v-once

全局感知 (需要超越组件的知识)

id

唯一的特性 (需要唯一值的特性)

ref

key

slot

双向绑定 (把绑定和事件结合起来)

v-model

其它特性 (所有普通的绑定或未绑定的特性)

事件 (组件事件监听器)

v-on

内容 (覆写元素的内容)

v-html

v-text

1. 组件/实例选项中的空行

你可能想在多个属性之间增加一个空行，特别是在这些选项一屏放不下，需要滚动才能都看到的时候。

当你的组件开始觉得密集或难以阅读时，在多个属性之间添加空行可以让其变得容易。在一些诸如 Vim 的编辑器里，这样格式化后的选项还能通过键盘被快速导航。

1. 单文件组件的顶级元素的顺序:

单文件组件应该总是让 <script>、<template> 和 <style> 标签的顺序保持一致。且 <style> 要放在最后，因为另外两个标签至少要有一个。

# 谨慎使用 (有潜在危险的模式)

1. 没有在 v-if/v-else-if/v-else 中使用 key

如果一组 v-if + v-else 的元素类型相同，最好使用 key (比如两个 <div> 元素)。

默认情况下，Vue 会尽可能高效的更新 DOM。这意味着其在相同类型的元素之间切换时，会修补已存在的元素，而不是将旧的元素移除然后在同一位置添加一个新元素。如果本不相同的元素被识别为相同，则会出现意料之外的结果。

1. scoped 中的元素选择器

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. 隐性的父子组件通信

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。然而，在一些边界情况下 prop 的变更或 this.$parent 能够简化两个深度耦合的组件。

问题在于，这种做法在很多简单的场景下可能会更方便。但请当心，不要为了一时方便 (少写代码) 而牺牲数据流向的简洁性 (易于理解)。

1. 非 Flux 的全局状态管理

应该优先通过 Vuex 管理全局状态，而不是通过 this.$root 或一个全局事件总线。