常用的 VueAPI 的最佳实践

对于 Vue 大家可能掌握了不少，而且相关其他成员（Vuex, vueRouter）大家也都基本掌握了，下面给大家罗列一下对于 VueAPI 比较好的实践方式。

1.始终在 v-for 中使用 :key

在需要操纵数据时，将 key 属性与 v-for 指令一起使用可以让程序保持恒定且可预测。

这是很有必要的，这样 Vue 就可以跟踪组件状态，并对不同的元素有一个常量引用。在使用动画或 Vue 转换时，key 非常有用。

如果没有 key，Vue 只会尝试使 DOM 尽可能高效。 这可能意味着 v-for 中的元素可能会出现乱序，或者它们的行为难以预测。 如果我们对每个元素都有唯一的键引用，那么我们可以更好地预测 Vue 应用程序将如何精确地处理 DOM 操作。

```
<!-- 不好的做法-->
<div v-for='product in products'>  </div>

<!-- 好的做法 -->
<div v-for='product in products' :key='product.id'>
```

2.使用驼峰式声明 props，并在模板中使用短横线命名来访问 props

最佳做法只是遵循每种语言的约定。 在 JS 中，驼峰式声明是标准，在 HTML 中，是短横线命名。 因此，我们相应地使用它们。

幸运的是，Vue 已经提供了驼峰式声明和短横线命名之间转换，因此除了实际声明它们之外，我们不必担心任何事情。

```
// 不好的做法
<PopupWindow titleText='hello world' />
props: { 'title-text': String }
```

```
// 好的做法
<PopupWindow title-text='hello world' />
props: { titleText: String }
```

3.data 应始终返回一个函数

声明组件 data 时，data 选项应始终返回一个函数。 如果返回的是一个对象，那么该 data 将在组件的所有实例之间共享。

```
// 不好的做法
data: {
  name: 'My Window',
  articles: []
}
```

但是，大多数情况下，我们的目标是构建可重用的组件，因此我们希望每个组件返回一个惟一的对象。我们通过在函数中返回数据对象来实现这一点。

```
// 好的做法
data () {
  return {
    name: 'My Window',
    articles: []
  }
}
```

4. 不要在同个元素上同时使用 v-if 和 v-for 指令

为了过滤数组中的元素，我们很容易将 v-if 与 v-for 在同个元素同时使用。

```
// 不好的做法
<div v-for='product in products' v-if='product.price < 500'>
```

问题是在 Vue 优先使用 v-for 指令，而不是 v-if 指令。它循环遍历每个元素，然后检查 v-if 条件。

```
this.products.map(function (product) {
  if (product.price < 500) {
    return product
  }
})
```

这意味着，即使我们只想渲染列表中的几个元素，也必须遍历整个数组。

这对我们来当然没有任何好处。

一个更聪明的解决方案是遍历一个计算属性，可以把上面的例子重构成下面这样的：

```
<div v-for='product in cheapProducts'>

computed: {
  cheapProducts: () => {
    return this.products.filter(function (product) {
      return product.price < 100
    })
  }
}
```

这么做有几个好处：

- 渲染效率更高，因为我们不会遍历所有元素
- 仅当依赖项更改时，才会重使用过滤后的列表
- 这写法有助于将组件逻辑从模板中分离出来，使组件更具可读性

5.用正确的定义验证我们的 props

可以这条是很重要，为什么？

在设计大型项目时，很容易忘记用于 props 的确切格式、类型和其他约定。如果你在一个更大的开发团队中，你的同事不会读心术，所以你要清楚地告诉他们如何使用你的组件。

因此，我们只需编写 props 验证即可，不必费力地跟踪组件来确定 props 的格式

从 Vue 文档中查看此示例。

```
props: {
  status: {
    type: String,
    required: true,
    validator: function (value) {
      return [
        'syncing',
        'synced',
```

```
        'version-conflict',
        'error'
      ].indexOf(value) !== -1
    }
  }
}
```

6. 基本组件应该相应地加上前缀

根据 Vue 样式指南，基本组件是仅包含以下内容的组件：

- HTML 元素
- 额外的基础组件
- 第三方的 UI 组件

为这些组件命名的最佳实践是为它们提供前缀 `Base`、`V` 或 `App`。同样，只要我们在整个项目中保持一致，可以使用其中任何一种。

```
BaseButton.vue
BaseIcon.vue
BaseHeading.vue
```

该命名约定的目的是使基本组件按字母顺序分组在文件系统中。 另外，通过使用 webpack 导入功能，我们可以搜索与命名约定模式匹配的组件，并将所有组件自动导入为 Vue 项目中的全局变量。

7 单实例组件命名应该带有前缀 The

与基本组件类似，单实例组件(每个页面使用一次，不接受任何 prop)应该有自己的命名约定。这些组件特定于我们的应用，通常是 `footer`，`header` 或 `sider`。

该组件只能有一个激活实例。

```
TheHeader.vue
```

```
TheFooter.vue
TheSidebar.vue
ThePopup.vue
```

8.保持指令简写的一致性

在 Vue 开发人员中，一种常见的技术是使用指令的简写。例如：

- @是 v-on 的简写
- : 是 v-bind 的简写

在你的 Vue 项目中使用这些缩写是很好的。但是要在整个项目中创建某种约定，总是使用它们或从不使用它们,会使我们的项目更具内聚性和可读性。

9. 模板表达式应该只有基本的 JS 表达式

在模板中添加尽可能多的内联功能是很自然的。但是这使得我们的模板不那么具有声明性，而且更加复杂，也让模板会变得非常混乱。

为此，让我们看看 Vue 样式指南中另一个规范化字符串的示例，看看它有多混乱。

```
//不好的做法
{{
  fullName.split(' ').map(function (word) {
    return word[0].toUpperCase() + word.slice(1)
  }).join(' ')
}}
```

基本上，我们希望模板中的所有内容都直观明了。 为了保持这一点，我们应该将复杂的表达式重构为适当命名的组件选项。

分离复杂表达式的另一个好处是可以重用这些值。

```
// 好的做法
{{ normalizedFullName }}
```

```
// The complex expression has been moved to a computed
property
computed: {
  normalizedFullName: function () {
    return this.fullName.split(' ').map(function (word) {
      return word[0].toUpperCase() + word.slice(1)
    }).join(' ')
  }
}
```

总结

这是几个最常见的最佳实践，它们将使我们的 Vue 代码更易于维护、可读性更好、更专业。