Vue 项目规范

(一) Vue 编码基础

vue 项目规范以 Vue 官方规范 （https://cn.vuejs.org/v2/style-guide/） 中的 A 规范为基础，在其上面进行项目开发，故所有代码均遵守该规范。

> 请仔仔细细阅读 Vue 官方规范，切记，此为第一步。

2.1.1. 组件规范

1) 组件名为多个单词。

组件名应该始终是多个单词组成（大于等于 2），且命名规范为KebabCase格式, 根组件 App 以及 、 之类的 Vue 内置组件除外。 这样做可以避免跟现有的以及未来的 HTML 元素相冲突，因为所有的 HTML 元素名称都是单个单词的。

正例：

```
export default {
  name: 'TodoItem'
  // ...
};
```

反例：

```
export default {
  name: 'Todo',
  // ...
}
export default {
  name: 'todo-item',
  // ...
}
```

2) 组件文件名为 KebabCase 格式

正例：

```
components/
|- MyComponent.vue
```

反例：

```
components/
|- myComponent.vue
|- my-component.vue
```

3) 基础组件文件名为 base 开头，使用完整单词而不是缩写。

正例：

```
components/
|- BaseButton.vue
|- BaseTable.vue
|- BaseIcon.vue
```

反例：

```
components/
|- MyButton.vue
|- VueTable.vue
|- Icon.vue
```

4) 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

正例：

```
components/
|- TodoList.vue
|- TodoListItem.vue
|- TodoListItemButton.vue
|- UserProfileOptions.vue （完整单词）
```

反例：

```
components/
|- TodoList.vue
|- TodoItem.vue
|- TodoButton.vue
|- UProfOpts.vue （使用了缩写）
```

5) 在 Template 模版中使用组件，应使用 PascalCase 模式，并且使用自闭合组件。

正例：

```
<!-- 在单文件组件、字符串模板和 JSX 中 -->
<my-component />
<todo-container><table :column="data"/></todo-container>
```

反例：

```
<MyComponent /> <Row><table :column="data"/></Row>
```

6) 组件的 data 必须是一个函数

当在组件中使用 data 属性的时候 (除了 new Vue 外的任何地方)，它的值必须是返回一个对象的函数。 因为如果直接是一个对象的话，子组件之间的属性值会互相影响。

正例：

```
export default {
  data () {
    return {
      name: 'jack'
    }
  }
}
```

反例：

```
export default {
  data: {
    name: 'jack'
  }
}
```

7) Prop 定义应该尽量详细

- 必须使用 camelCase 驼峰命名
- 必须指定类型
- 必须加上注释，表明其含义
- 必须加上 required 或者 default，两者二选其一
- 如果有业务需要，必须加上 validator 验证

正例：

```
 props: {
  // 组件状态，用于控制组件的颜色
```

```
    status: {
      type: String,
      required: true,
      validator: function (value) {
        return [
          'succ',
          'info',
          'error'
        ].indexOf(value) !== -1
      }
    },
     // 用户级别，用于显示皇冠个数
    userLevel: {
       type: String,
       required: true
    }
}
```

8) 为组件样式设置作用域

正例：

```
<template>
  <button class="btn btn-close">X</button>
</template>

<!-- 使用 `scoped` 特性 -->
<style scoped>
  .btn-close {
    background-color: red;
  }
</style>
```

反例：

```
<template>
  <button class="btn btn-close">X</button>
</template>
<!-- 没有使用 `scoped` 特性 -->
<style>
  .btn-close {
    background-color: red;
  }
</style>
```

9) 如果特性元素较多，应该主动换行。

正例：

```
<MyComponent foo="a" bar="b" baz="c"
    foo="a" bar="b" baz="c"
    foo="a" bar="b" baz="c"
 />
```

反例：

```
<MyComponent foo="a" bar="b" baz="c" foo="a" bar="b" baz="c" foo="a"
bar="b" baz="c" foo="a" bar="b" baz="c"/>
```

2.1.2. 模板中使用简单的表达式

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。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。

正例：

```
<template>
  <p>{{ normalizedFullName }}</p>
</template>

// 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(' ')
  }
}
```

反例：

```
<template>
  <p>
       {{
          fullName.split(' ').map(function (word) {
            return word[0].toUpperCase() + word.slice(1)
           }).join(' ')
        }}
  </p>
</template>
```

2.1.3 指令都使用缩写形式

指令推荐都使用缩写形式，(用 : 表示 v-bind: 、用 @ 表示 v-on: 和用 # 表示 v-slot:)

正例：

```
<input
  @input="onInput"
  @focus="onFocus"
>
```

反例：

```
<input
  v-on:input="onInput"
  @focus="onFocus"
>
```

2.1.4 标签顺序保持一致

单文件组件应该总是让标签顺序保持为 `<template>` 、`<script>`、 `<style>`

正例：

```
<template>...</template>
<script>...</script>
<style>...</style>
```

反例：

```
<template>...</template>
<style>...</style>
<script>...</script>
```

2.1.5 必须为 v-for 设置键值 key

2.1.6 v-show 与 v-if 选择

如果运行时，需要非常频繁地切换，使用 v-show ； 如果在运行时，条件很少改变，使用 v-if。

2.1.7 script 标签内部结构顺序

components > props > data > computed > watch > filter > 钩子函数（钩子函数按其执行顺序） > methods

2.1.8 Vue Router 规范

1) 页面跳转数据传递使用路由参数

页面跳转，例如 A 页面跳转到 B 页面，需要将 A 页面的数据传递到 B 页面，推荐使用 路由参数进行传参，而不是将需要传递的数据保存 vuex，然后在 B 页面取出 vuex 的数据，因为如果在 B 页面刷新会导致 vuex 数据丢失，导致 B 页面无法正常显示数据。

正例：

```
let id = ' 123';
this.$router.push({ name: 'userCenter', query: { id: id } });
```

2) 使用路由懒加载（延迟加载）机制

```
    {
        path: '/upload-attachment',
        name: 'UploadAttachment',
        meta: {
          title: '上传附件'
        },
        component: () => import('@/view/components/upload-
attachment/index.vue')
      },
```