开发规范

> 维护人：前端团队
> 适用版本：IDP4
> 最后更新时间：2021.12.06

前言

在团队协作中，良好的开发规范有利于提高程序的可读性、可维护性、可复用性，切实提高软件产品质量，众所周知，制订交通法规表面上是要限制行车权，实际上是保障公众的人身安全。试想如果没有限速，没有红绿灯，没有靠右行驶条款，谁还敢上路。同理，对软件来说，开发规约绝不是消灭代码内容的创造性、优雅性，而是限制过度个性化，推行相对标准化，以一种普遍认可的方式一起做事。
综上所述，开发规约的目标：

- 码出高效：标准统一，提升沟通效率和研发效能。
- 码出质量：防患未然，提升质量意识和系统可维护性，降低故障率。
- 码出情怀：工匠精神，追求极致的卓越精神，打磨精品代码。

单条规约的说明

规约级别

根据约束力强弱，一条规约依次分为强制、推荐、参考三个级别：

- 【强制】必须遵守。是不得不遵守的约定，违反本约定或将会引起严重的后果。
- 【推荐】尽量遵守。长期遵守这样的规定，有助于系统稳定性和合作效率的提升。
- 【参考】充分理解。技术意识的引导，是个人学习、团队沟通、项目合作的方向。

代码示例

为了更加直观，规约描述之后通常会配上代码示例，例如：

```JavaScript
// bad
function foo() {
  ••••
  let name;
}
// good
function foo() {
  ••
  let name;
}
```

我们约定用 bad 注释表示反例，用 good 注释表示正例。

除了 bad 和 good，有时你还会看到 disallowed、allowed、best 这几种注释，它们的含义如下：

```JavaScript
// disallowed - 禁止（用于部分明令禁止的用法）
// bad - 反例
// allowed - 中例（用于允许但不推荐的用法）
// good - 正例
// best - 最佳正例（多个正例中最好的实现）
```

Vue 编码规约

1.template 标准

标签

- 1.1【强制】不允许 template 中有解析错误。eslint: vue/no-parsing-error
  - 1.1.1 注释标签不能为空「abrupt-closing-of-empty-comment」

```JavaScript
// bad
<template><!--></template>
// good
<template><div><!--<slot name="reference"></slot>--></div></template>
```

- 1.1.2 注释中不能有 EOF「eof-in-comment」

JavaScript 复制代码

```
// bad
<template><!--

// good
 <template><div><!--<slot name="reference"></slot>--></div></template>
```

- 1.1.3 注释要正确开闭「incorrectly-opened-comment」「incorrectly-closed-comment」

JavaScript 复制代码

```
// bad
<template><!ELEMENT br EMPTY></template>
<template><!--comment--!></template>

// good
<template><div><!--<slot name="reference"></slot>--></div></template>
```

- 1.1.4 注释不能嵌套「nested-comment」

JavaScript 复制代码

```
// bad
<template><!--a<!--b--></template>

// good
<template><div><!--<slot name="reference"></slot>--></div></template>
```

- 1.1.5 数字字符引用中必须含有数字「absence-of-digits-in-numeric-character-reference」

JavaScript 复制代码

```
// bad
<template>&#qux;</template>

// good
<template>&#205;</template>
```

- 1.1.6 CDATA 只能用于 MathML 或者 SVG「cdata-in-html-content」

JavaScript 复制代码

```
// bad
<template><![CDATA[cdata]]></template>

// good
<template>
  <svg>
    <style>
      <![CDATA[
        #my-rect { fill: blue; }
      ]]>
    </style>
    <rect id="my-rect" x="0" y="0" width="10" height="10" />
  </svg>
</template>
```

- 1.1.7 字符引用不能超出 unicode 编码范围「character-reference-outside-unicode-range」

JavaScript 复制代码

```
// bad
<template>&#1234567;</template>

// good
<template>&#0x61;</template>
```

- 1.1.8 除 ASCII 空白字符之外，不能包含控制字符「control-character-in-input-stream」

JavaScript 复制代码

```
// bad
<template>\u0003</template>

// good
<template><div>demo</div></template>
```

- 1.1.9 除 ASCII 空白字符之外，引用字符不能引用控制字符「control-character-reference」

JavaScript 复制代码

```
// bad
<template>&#0003;</template>

// good
<template>&#0000;</template>
```

- 1.1.10 「eof-before-tag-name」

JavaScript 复制代码

```
// bad
<template><

// good
<template><div>demo</div></template>
```

- 1.1.11 「eof-in-cdata」

JavaScript 复制代码

```
// bad
<template><svg><![CDATA[cdata

// good
<template><div>demo</div></template>
```

- 1.1.12 「eof-in-tag」

JavaScript 复制代码

```
// bad
<template><div class=""

// good
<template><div>demo</div></template>
```

- 1.1.13 「missing-attribute-value」

JavaScript 复制代码

```
// bad
<template><div id=></template>

// good
<template><div id="demo">demo</div></template>
```

- 1.1.14 「missing-end-tag-name」

JavaScript 复制代码

```
// bad
<template></></template>

// good
<template><i/></template>
```

- 1.1.15 字符引用之后要加分号「missing-semicolon-after-character-reference」

JavaScript 复制代码

```
// bad
<template>&amp</template>

// good
<template>&amp;</template>
```

- 1.1.16 属性之间加空格「missing-whitespace-between-attributes」

JavaScript 复制代码

```
// bad
<template><div id="foo"class="bar"></template>

// good
<template><div id="foo" class="bar"></template>
```

- 1.1.17「noncharacter-character-reference」

JavaScript 复制代码

```
// bad
<template>&#xFFFE;</template>

// good
<template>&amp;</template>
```

- 1.1.18「noncharacter-in-input-stream」

JavaScript 复制代码

```
// bad
<template>\uFFFE</template>

// good
<template>demo</template>
```

- 1.1.19 数字字符引用不能引用 NULL「null-character-reference」

JavaScript 复制代码

```
// bad
<template>&#0000;</template>

// good
<template>&#205;</template>
```

- 1.1.20 数字字符引用不能引用 surrogate「null-character-reference」

JavaScript 复制代码

```
// bad
<template>&#xD800;</template>

// good
<template>&#205;</template>
```

- 1.1.21 html 中不能使用 surrogate「surrogate-in-input-stream」

JavaScript 复制代码

```
// bad
<template>\uD800</template>

// good
<template>demo</template>
```

- 1.1.22 属性名不能含有字符 ',",< 「unexpected-character-in-attribute-name」

JavaScript 复制代码

```
// bad
<template><div a"bc=""></template>

// good
<template><div abc="111">demo</div></template>
```

- 1.1.23 当属性值没有写在引号中时，属性值不能包含 ",',<,=,` 「unexpected-character-in-unquoted-attribute-value」

JavaScript 复制代码

```
// bad
<template><div foo=b'ar'></template>

// good
<template><div abc="111">demo</div></template>
```

- 1.1.24 禁止在属性名之前使用等号「unexpected-equals-sign-before-attribute-name」

JavaScript 复制代码

```
// bad
<template><div =foo></template>

// good
<template><div foo="demo"></template>
```

- 1.1.25「unexpected-null-character」

JavaScript 复制代码

```
// bad
<template>\u0000</template>

// good
<template>demo</template>
```

- 1.1.26「unexpected-question-mark-instead-of-tag-name」

JavaScript 复制代码

```
// bad
<template><?xml?></template>

// good
<template><div>demo</div></template>

```

- 1.1.27「unexpected-solidus-in-tag」

JavaScript 复制代码

```
// bad
<template><div id="" / class=""></template>

// good
<template><div id="demo" class="demo"></template>
```

- 1.1.28「unknown-named-character-reference」

JavaScript 复制代码

```
// bad
<template>&unknown;</template>

// good
<template>&amp;</template>
```

- 1.1.29 结束标签不能包含属性「end-tag-with-attributes」

JavaScript 复制代码

```
// bad
<template><div></div id="end"></template>

// good
<template><div>demo</div></template>
```

- 1.1.30 同一个标签不能包含重复的属性「duplicate-attribute」

JavaScript 复制代码

```
// bad
<template><div id="" id=""></div></template>

// good
<template><div id=""></div></template>
```

- 1.1.31 结束标签不能加尾部斜杠「end-tag-with-trailing-solidus」

JavaScript 复制代码

```
// bad
<template><div></div/></template>

// good
<template><div id=""></div></template>
```

- 1.1.32 开始标签不能加尾部斜杠「non-void-html-element-start-tag-with-trailing-solidus」

JavaScript 复制代码

```
// bad
<template><div/></template>

// good
<template><div id=""></div></template>
```

- 1.1.33 命名空间不合法「x-invalid-namespace」

JavaScript 复制代码

```
// bad
<template><div xmlns=""></template>

// good
<div xmlns="http://www.w3.org/1999/Math/MathMl">x3/x</div>
```

2.语言特性

2.1 基本

- 2.1.1【推荐】不要定义未被使用的组件。eslint: vue/no-unused-components

JavaScript 复制代码

```
// bad
<template>
  <div>
    <h2>Lorem ipsum</h2>
  </div>
</template>
<script>
  export default {
    components: {
      TheButton
    },
  }
</script>

// good
<template>
  <div>
    <TheButton />
  </div>
</template>
<script>
  export default {
    components: {
      TheButton
    },
  }
</script>
```

- 2.1.2【强制】组件的 data 必须是一个函数。eslint: vue/no-shared-component-data

JavaScript | 复制代码

```
// bad
<script lang="js">
  export default {
    data: {
      foo: 'bar'
    }
  }
</script>

// good
<script lang="js">
  export default {
    data() {
      return {
        foo: 'bar'
      }
    }
  }
</script>
```

- 2.1.3【强制】禁止定义同名属性。eslint: vue/no-dupe-keys

JavaScript | 复制代码

```
// bad
<script lang="js">
  export default {
    props: {
      foo: String
    },
    computed: {
      foo: {
        get () {}
      }
    },
    data: {
      foo: null
    },
    methods: {
      foo () {}
    }
  }
</script>

// good
<script lang="js">
  export default {
    props: {
      foo: String
    },
    computed: {
      bar: {
        get () {}
      }
    }
  }
  </script>
```

- 2.1.4【强制】标签中禁止有重复的属性。eslint: vue/require-prop-type-constructor

JavaScript 复制代码

```
// bad
<template>
  <div>
    <MyComponent :foo="abc" foo="def" />
    <MyComponent foo="abc" :foo="def" />
    <MyComponent foo="abc" foo="def" />
    <MyComponent :foo.a="abc" :foo.b="def" />
    <MyComponent class="abc" class="def" />
  </div>
</template>

// good
<template>
  <div>
    <MyComponent :foo="abc" />
    <MyComponent class="abc" :class="def" />
  </div>
</template>
```

- 2.1.5【推荐】在使用 vue 的内置组件 component 时，必须使用 v-bind:is 属性。eslint: vue/require-component-is

JavaScript 复制代码

```
// bad
<template>
  <div>
    <component/>
    <component is="type"/>
  </div>
</template>

// good
<template>
  <div>
    <component :is="type"/>
    <component v-bind:is="type"/>
  </div>
</template>
```

- 2.1.6【强制】render 函数必须有返回值。eslint: vue/require-render-return

JavaScript 复制代码

```
// bad
<script>
export default {
  render (h) {
    if (foo) {
      return h('div', 'hello')
    }
  }
}
</script>

// good
<script>
export default {
  render (h) {
    return h('div', 'hello')
  }
}
</script>
```

- 2.1.7【强制】禁止使用 vue 中的关键字。eslint: vue/no-reserved-keys

JavaScript 复制代码

```
// bad
<script>
export default {
  props: {
    $el: String
  },
}
</script>

// good
<script>
export default {
  props: {
    custom: String
  },
}
</script>
```

- 2.1.8【推荐】禁止在 template 标签中使用 key 属性。 eslint: vue/no-template-key

JavaScript 复制代码

```
// bad
<template key="foo"> ... </template>
<template v-bind:key="bar"> ... </template>
<template :key="baz"> ... </template>

// good
<template> demo </template>
```

- 2.1.9【强制】禁止在 textarea 中出现 {{ message }}。 eslint: vue/no-textarea-mustache

JavaScript 复制代码

```
// bad
<template>
  <textarea>{{ message }}</textarea>
</template>

// good
<template>
  <textarea v-model="message" />
</template>
```

- 2.1.10【推荐】禁止在 v-for 等指令或者 scope 中申明没有使用到的变量。 eslint: vue/no-unused-vars

JavaScript 复制代码

```
// bad
<template>
  <ol v-for="i in 5">
    <li>item</li>
  </ol>
</template>

// good
<template>
  <ol v-for="i in 5">
    <li>{{ i }}</li>
  </ol>
</template>
```

2.2 Props

- 2.2.1【强制】Prop 定义类型应该是构造函数。eslint: vue/require-prop-type-constructor

JavaScript | 复制代码

```
// bad
<script>
export default {
  props: {
    myProp: 'Number',
    anotherType: ['Number', 'String'],
    extraProp: {
      type: 'Number',
      default: 10
    },
    lastProp: {
      type: ['Boolean']
    },
    nullProp: 'null'
  }
}
</script>

// good
<script>
export default {
  props: {
    myProp: Number,
    anotherProp: [Number, String],
    myFieldWithBadType: {
      type: Object,
      default: function() {
        return {}
      },
    },
    myOtherFieldWithBadType: {
      type: Number,
      default: 1,
    }
  }
}
</script>
```

- 2.2.2【强制】Prop 的默认值必须匹配它的类型。eslint: vue/require-valid-default-prop

JavaScript | 复制代码

```
// bad
<script>
export default {
  props: {
    myProp: {
      type: String,
      default: {}
    }
  }
}
</script>

// good
<script>
export default {
  props: {
    myProp: {
      type: String,
      default: 'demo'
    }
  }
}
</script>
```

- 2.2.3【强制】计算属性禁止包含异步方法。eslint: vue/no-async-in-computed-properties

JavaScript 复制代码

```
// bad
<script>
export default {
  computed: {
    pro () {
      return Promise.all([new Promise((resolve, reject) => {})])
    },
    foo: async function () {
      return await someFunc()
    }
  }
}
</script>

// good
<script>
export default {
  computed: {
    pro () {
      return 'pro'
    }
  }
}
</script>
```

- 2.2.4【强制】计算属性必须有返回值。eslint: vue/return-in-computed-property

JavaScript 复制代码

```
// bad
<script>
export default {
  computed: {
    pro () {
      if (this.baf) {
        return this.baf
      }
    }
  }
}
</script>

// good
<script>
export default {
  computed: {
    pro () {
      return 'pro'
    }
  }
}
</script>
```

- 2.2.5【强制】为 v-for 设置键值。eslint: vue/require-v-for-key

JavaScript 复制代码

```
// bad
<template>
  <div v-for="todo in todos"/>
</template>

// good
<template>
  <div v-for="todo in todos" :key="todo.id"/>
</template>
```

- 2.2.6【强制】禁止在计算属性中对属性修改。eslint: vue/no-side-effects-in-computed-properties

JavaScript 复制代码

```
// bad
<script>
export default {
  computed: {
    fullName () {
      this.firstName = 'lorem' // <- side effect
      return `${this.firstName} ${this.lastName}`
    },
    reversedArray () {
      return this.array.reverse() // <- side effect - orginal array is being mutated
    }
  }
}
</script>

// good
<script>
export default {
  computed: {
    fullName () {
      return `${this.firstName} ${this.lastName}`
    },
    reversedArray () {
      return this.array.slice(0).reverse() // <- side effect - orginal array is being mutated
    }
  }
}
</script>
```

2.3 指令

- 2.3.1【推荐】避免 v-if 和 v-for 用在一起。eslint: vue/no-use-v-if-with-v-for

JavaScript 复制代码

```
// bad
<template>
  <TodoItem
      v-for="todo in todos"
      v-if="todo.shown"
      :todo="todo"
  />
</template>

// good
<template>
  <ul v-if="complete">
    <TodoItem
      v-for="todo in todos"
      :todo="todo"
    />
  </ul>
</template>
```

- 2.3.2【强制】强制在 v-on 指令使用 exact 修饰符，当同一个标签上有另一个带修饰符的 v-on 指令。eslint: vue/use-v-on-exact

JavaScript 复制代码

```
// bad
<template>
  <button v-on:click="foo" v-on:click.ctrl="foo"></button>
</template>

// good
<template>
  <button v-on:click.exact="foo" v-on:click.ctrl="foo"></button>
</template>
```

- 2.3.3【强制】检查 root 标签的合法性。eslint: vue/valid-template-root

JavaScript 复制代码

```
// bad
<template></template>
<template>The root is text</template>
<template>
  <div>hello</div>
  <div>There are multiple root elements</div>
</template>
<template>
  <div v-for="item in items"/>
</template>
<template>
  <slot />
</template>

// good
<template>
  <div>demo</div>
</template>
```

- 2.3.4【强制】检查 bind 指令的合法性。eslint: vue/valid-v-bind

JavaScript 复制代码

```
// bad
<template>
  <div v-bind/>
  <div :aaa/>
  <div v-bind:aaa.bbb="foo"/>
</template>


// good
<template>
  <div v-bind="foo"/>
  <div v-bind:aaa="foo"/>
  <div :aaa="foo"/>
  <div :aaa.prop="foo"/>
</template>
```

- 2.3.5【强制】检查 v-cloak 指令的合法性。eslint: vue/valid-v-cloak

JavaScript 复制代码

```
// bad
<template>
  <div v-cloak:aaa/>
  <div v-cloak.bbb/>
  <div v-cloak="ccc"/>
</template>


// good
<template>
  <div v-cloak/>
</template>

```

- 2.3.6【强制】检查 v-else-if 指令的合法性。eslint: vue/valid-v-else-if

JavaScript 复制代码

```
// bad
<template>
  <div v-else-if/>
  <div v-else-if:aaa="foo"/>
  <div v-else-if.bbb="foo"/>
</template>


// good
<template>
  <div v-if="foo"/>
  <div v-else-if="bar"/>
</template>
```

- 2.3.7【强制】检查 v-else 指令的合法性。eslint: vue/valid-v-else

JavaScript 复制代码

```
// bad
<template>
  <div v-else="foo"/>
  <div v-else:aaa/>
  <div v-else.bbb/>
</template>


// good
<template>
  <div v-if="foo"/>
  <div v-else/>
</template>
```

- 2.3.8【强制】检查 v-for 指令的合法性。eslint: vue/valid-v-for

JavaScript 复制代码

```
// bad
<template>
  <div v-for/>
  <div v-for:aaa="todo in todos"/>
  <div v-for.bbb="todo in todos"/>
  <div
    v-for="todo in todos"
    is="MyComponent"
  />
  <MyComponent v-for="todo in todos"/>
  <MyComponent
    v-for="todo in todos"
    :key="foo"
  />
</template>

// good
<template>
  <div v-for="todo in todos"/>
  <MyComponent
    v-for="todo in todos"
    :key="todo.id"
  />
  <div
    v-for="todo in todos"
    :is="MyComponent"
    :key="todo.id"
  />
</template>
```

- 2.3.9【强制】检查 v-html 指令的合法性。eslint: vue/valid-v-html

JavaScript 复制代码

```
// bad
<template>
  <div v-html/>
  <div v-html:aaa="foo"/>
  <div v-html.bbb="foo"/>
</template>

// good
<template>
  <div v-html="foo"/>
</template>
```

- 2.3.10【强制】检查 v-if 指令的合法性。eslint: vue/valid-v-if

JavaScript 复制代码

```
// bad
<template>
  <div v-if/>
  <div v-if:aaa="foo"/>
  <div v-if.bbb="foo"/>
  <div
    v-if="foo"
    v-else
  />
  <div
    v-if="foo"
    v-else-if="bar"
  />
</template>

// good
<template>
  <div v-if="foo"/>
  <div v-else-if="bar"/>
  <div v-else/>
</template>
```

- 2.3.11【强制】检查 v-model 指令的合法性。eslint: vue/valid-v-model

JavaScript 复制代码

```
// bad
<template>
  <input v-model>
  <input v-model:aaa="foo">
  <input v-model.bbb="foo">
  <input v-model="foo + bar">
  <div v-model="foo"/>
  <div v-for="todo in todos">
    <input v-model="todo">
  </div>
</template>

// good
<template>
  <input v-model="foo">
  <input v-model.lazy="foo">
  <textarea v-model="foo"/>
  <MyComponent v-model="foo"/>
  <div v-for="todo in todos">
    <input v-model="todo.name">
  </div>
</template>
```

- 2.3.12【强制】检查 v-on 指令的合法性。eslint: vue/valid-v-on

JavaScript 复制代码

```
// bad
<template>
  <div v-on/>
  <div v-on:click/>
  <div v-on:click.aaa="foo"/>
  <div @click/>
</template>

// good
<template>
  <div v-on="foo"/>
  <div v-on:click="foo"/>
  <div @click="foo"/>
  <div @click.left="foo"/>
  <div @click.prevent/>
  <div @click.stop />
</template>
```

- 2.3.13【强制】检查 v-once 指令的合法性。eslint: vue/valid-v-once

JavaScript 复制代码

```
// bad
<template>
  <div v-once:aaa/>
  <div v-once.bbb/>
  <div v-once="ccc"/>
</template>

// good
<template>
  <div v-once/>
</template>
```

- 2.3.14【强制】检查 v-once 指令的合法性。eslint: vue/valid-v-once

JavaScript 复制代码

```
// bad
<template>
  <div v-once:aaa/>
  <div v-once.bbb/>
  <div v-once="ccc"/>
</template>

// good
<template>
  <div v-once/>
</template>
```

- 2.3.15【强制】检查 v-pre 指令的合法性。eslint: vue/valid-v-pre

JavaScript 复制代码

```
// bad
<template>
  <div v-pre:aaa/>
  <div v-pre.bbb/>
  <div v-pre="ccc"/>
</template>

// good
<template>
  <div v-pre/>
</template>
```

- 2.3.16【强制】检查 v-show 指令的合法性。eslint: vue/valid-v-show

JavaScript 复制代码

```
// bad
<template>
  <div v-show/>
  <div v-show:aaa="foo"/>
  <div v-show.bbb="foo"/>
</template>

// good
<template>
  <div v-show="foo"/>
</template>
```

- 2.3.17【强制】检查 v-text 指令的合法性。eslint: vue/valid-v-text

JavaScript 复制代码

```
// bad
<template>
  <div v-text/>
  <div v-text:aaa="foo"/>
  <div v-text.bbb="foo"/>
</template>

// good
<template>
  <div v-text="foo"/>
</template>
```

参考资料

- eslint-plugin-vue
- Style Guide – Vue

Git 相关规约

1 Git 提交日志格式规约

1.1 前言

为什么要对 Git 提交日志（message）的格式进行规约约束?

1. 更方便、快捷地浏览和了解项目的历史信息。
2. 有利于保证提交内容的独立性，避免把所有改动都放在一个提交里面。
3. 便于通过脚本自动化生成 CHANGELOG。

1.2 基本的 message 格式

JavaScript 复制代码

```
<type>[optional scope]: <subject>

[optional body]

[optional footer(s)]
```

其中 message header（即首行）必选，scope、body 和 footer 可选。

1.2.1 字数限制

- header（首行）：只有一行，不超过 50 个字符
- body：每行不超过 72 个字符
- footer：每行不超过 72 个字符

> 为什么要有字数限制?
>
> - header： 像 Linux、Git 这样的开源项目，是以邮件列表作为代码评审的平台，header 要作为邮件的标题，而邮件标题本身就有长度的限制，并且标题太长也不利于浏览和信息获取。
> - body 和 footer：源于大部分编程语言的编码规范，最初源于打字机宽度等物理设备的限制，后来慢慢成为默认遵守的规范。

1.2.2 语言选择

在开源项目中：推荐使用英文，因为项目的开发者和参与者可能来自世界各地，使用英文可以更广泛的传递信息。

在内部项目（并且短时间内也不涉及开源）中：应该选择内部人员普遍能够熟练表达的语言。

例如在国内的团队中，可以使用中文。

> 关于使用中文可能会出现乱码的问题：处理字符集和字符编码的问题应该是每个程序员的必修课。
>
> 关于使用英文天然正确性的问题：因地制宜，适合的才是最好的。

1.3 message header

1.3.1 type

type 用来描述本次提交的改动类型，可选值及对应含义如下：

- feat: 新增功能
- fix: 修复 bug
- docs: 文档相关的改动
- style: 对代码的格式化改动，代码逻辑并未产生任何变化(例如代码缩进，分号的移除和添加)
- test: 新增或修改测试用例
- refactor: 重构代码或其他优化举措
- chore: 项目工程方面的改动，代码逻辑并未产生任何变化
- revert: 恢复之前的提交

> 注意：
>
> 1. commit message 的 type 和 changelog 的 type 存在紧密联系，然而它们两者之间并非一一对应，比如在 changelog 中一般不会指出文档 docs 或测试用例 test 等方面发生的变化
> 2. css 样式文件的修改一般属于 feat 或者 fix，并不是 style

1.3.2 scope

scope 用来描述本次提交所涉及到的改动范围（例如模块、功能或其他任何限定的范围）。

scope 的具体取值视项目而定。以淘宝详情页为例，取值可以是：header, footer, favorite, sku, etc...

如果是 monorepo 的项目，scope 取值可以是 subpackage 的名称。例如 babel 项目中对某个 package 的修改：

JavaScript | 复制代码

```
chore(babel-helper-plugin-utils): add npmignore
```

1.3.3 subject

subject 用来概括和描述本次提交的改动内容，需注意以下几点：

- 时态方面使用一般现在时，不要使用过去时。虽然查看 message 时，message 内容本身都发生在过去，然而对于主题来说，使用现在时的时态更简洁明确，并且更易达成一致性：

```JavaScript
// good
docs: delete redundant docs

// bad
docs: deleted redundant docs
```

- 句式使用祈使句。即一般情况不要增加主语。因为在绝大情况下，主语都是作者『我』：

```JavaScript
// good
docs: delete redundant docs

// bad
docs: i delete redundant docs
```

- 句首无需大写，句尾无需结束标点。因为主题（或标题）本身不用形成完整的句子：

```JavaScript
// good
docs: delete redundant docs

// bad
docs: Delete redundant docs.
```

1.4 message body

日志的内容主体 body 用来描述详细的提交内容，可写可不写，需注意以下几点：

1. 时态方面使用一般现在时，不要用过去时态。
2. 句式视情况而定，一般使用祈使句式。
3. 标点方面遵循一般的文档格式规约。

1.5 message footer

footer 通常用于代码评审过程记录、作者签名等。例如：

```
Reported-by: User1 <user1@example.com>
Helped-by: User2 <user2@example.com>
Reviewed-by: User3 <user3@example.com>
Signed-off-by: Author <author@example.com>
```

> 为什么要有签名区?
> 因为一个提交的元信息中只有作者（author）、提交者（committer）两个字段，而一段代码的诞生，参与的人往往不止于此，还可能有问题报告者（Reported-by）、代码评审者（Reviewed-by）、上游 Committer 的签名（Signed-off-by）。为此一些开源项目（如 Git、Linux）的一个约定俗成的习惯，是在提交的最后加上签名，每个贡献者一行，从上到下可以看到这段代码诞生的过程

还可以添加其他元信息，例如：

引用 Issues

使用 Closes 关键字，后面跟上（Gitlab / Redmine 或其他平台的）issue ID 来表明该提交关闭解决了某个 Issue：`Closes#120`
关闭多个 Issues 使用如下格式：`Closes #35, #38, #49`

破坏性变动（Breaking changes）

如果本次提交的改动是破坏性的，需要在这里声明：

```
BREAKING CHANGE: 为了组件 API 规范的统一，本次升级将 size 属性的 value 值从 `s|m|l` 替换为 `small|medium|large`。

请按照如下方式升级：

<Button size="s">提交</Button>
-->
<Button size="small">提交</Button>

继续使用 size="m" 可能会导致样式错误。
```

2 Git 分支命名规约

2.1 分支模型选择的说明

目前互联网和社区中流传最广泛的一个分支模型 Git Flow 出自 a-successful-git-branching-model 这篇十年前的文章，文章作者 Vincent Driessen 在 2020 年三月份的时候已经公开表示，该分支模型已经不适用于现如今持续交付的软件工程方式，推荐在持续交付的软件工程中使用更简单的 Github Flow 模型。

2.2 分支命名

新建分支的命名格式为：`{type}-{issue id}-the-thing-you-do`

- `type`：和上文 1.3.1 章节中的 type 保持一致
- `issue id`：与分支内容相关的 issue id，如果无关，则可以忽略

比如以下格式都满足规范：

- `feat-ssr-prefetch`：新增 ssr prefetch 功能
- `fix-1379-component-insert-order`：修复 issue 1379 中提到的组件插入顺序 bug
- `revert-14218-memory-leak-on-unmount`：回退版本解决 issue 14218 提到的组件卸载时内存泄露的问题

> 注 1：该命名规约只针对新建的临时分支，其他常驻分支如 master，develop 不受影响。
> 注 2：在工程领域中，为了区别发布资源的版本，往往还需要在分支中加入版本的信息，
> `prefix`：用于简要描述迭代信息，命名遵循本节定义的 `{type}-{issue id}-the-thing-you-do` 规范
> `semver`：本次发布的迭代版本号，格式需要遵循 semantic version
> 比如以下格式都满足规范：
>
> - `feat-xxx/1.0.0`：新增 xxx 功能, 迭代版本号为 1.0.0
> - `feat-xxx/1.0.0-alpha.1`：新增 xxx 功能，迭代版本号为 1.0.0-alpha.1, 适用于 tnpm 发布流程的 alpha 版本包
> - `fix-xxx/1.0.0`：修复 xxx 问题，迭代版本号为 1.0.0
>
> 不推荐 `daily/0.0.1` 这种命名方式（但 DEF 为了历史兼容依然支持）

2.3 多版本分支命名

在需要同时维护多个版本的项目中，只使用 master 作为主干分支显然是无法满足需求的，但是又不想使用 Git Flow 这种复杂、繁琐的分支结构，那么就可以每发布一个新的版本就单独拉一个新的分支，例如：

> - `1.0.0-stable`
> - `2.0.0-stable`

3 Git tag 命名规约

Git tag 就是通过语义化的名称来给仓库标注一个个具体的节点。与此同时还可以根据标签名称来大致了解当前项目的兼容性和迭代情况。

命名格式为 `v{semver}`，`semver` 是遵循 semantic version 的版本号，例如 `v1.2.3`。

相比于使用例如 `git tag v1.2.3` 这种「轻量标签」，更推荐使用如下命令生成「附注标签」：

`git tag -a v1.2.3 -m "发布经销商管理模块"`

4 项目分支命名规约

在需要同时维护多个项目时，只使用定版分支显然是不行的，我们需要把项目使用的分支有规范的命名，比如项目是从哪个版本开的分支，代表的是哪个项目等都需要在分支上体现清楚。
那么什么是定版分支？在分支中，release-{版本号}则是定版分支，我们可以基于定版分支开项目分支。
如：release-v4.15
项目主分支规则 v{x.x}-project-{项目简称} => 如：v1.5-project-PG
如：xxx 项目需要在 v4.15 上开出分支来作为项目分支，那么则可以这样命名 v4.15-project-xxx。

但是，我们有的工程有四个模块，四个仓库，是否都需要创建项目分支呢？当然不是啦！比如我们需要基于 4.15 版本对登录模块进行定制，那么只需要创建登录模块的项目分支，其余仓库使用定版分支即可：

项目/仓库	Owner	IDP4-Login	IDP4-ITManager	IDP4-Enduser	IDP4-Developer	IDP4-Root	对应后端分支
xxx	xxx	v4.15-project-xxx	release-v4.15	release-v4.15	release-v4.15	release-v4.15【这个模块永远都是定版分支】	填写后端对应分支即可

以上就是一个完整的项目分支命名规范，请严格按照此规范对项目分支进行管理。

配套工具

- commitlint-config-ali：本规约配套的 commitlint 配置包，可用于校验 commit message。

参考资料

1. AngularJS 代码贡献指南
2. AngularJS Git Commit Message Conventions
3. Karma 的 Git 日志规约
4. StackOverflow – 在 Git 日志中我该用过去时态还是现在时态?
5. 一个成功的 Git 分支模型
6. Git 基础打标签
7. 每行字符数
8. Conventional Commits