F:\code\react\rtd-wb\rtd-wb-frontend2\src\assets\logo.png

实时决策

Web前端工程及开发规范

|  |  |  |
| --- | --- | --- |
| 文档状态 | 文档标识： |  |
| 草稿  正式  更改 | 当前版本： | V2.0 |
| 编 写： | 2018年7月24日 |
| 完成日期： | 2019年8月10日 |

**文档修改记录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 章节编号 | 章节名称 | 修订内容简述 | 修订日期 | 修改人 | 修订前版本号 |
|  |  |  | 2018年7月24日 | 曾子建 | 0.6 |
|  |  |  | 2019年8月10日 | 曾子建 | 2.0 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目 录

[第 1 章 概述 5](#_Toc497732834)

[1.1 期望目的 5](#_Toc497732835)

[1.2 适用人群 5](#_Toc497732836)

[第 2 章 工程 6](#_Toc497732837)

[2.1 工程构建 6](#_Toc497732838)

[2.2 前端架构 6](#_Toc497732839)

[2.3 开发测试 7](#_Toc497732840)

[2.4 详细说明 9](#_Toc497732841)

[2.5 打包发布 13](#_Toc497732842)

[2.6 发布一个NPM包 13](#_Toc497732843)

[第 3 章 规范 13](#_Toc497732844)

[3.1 基本准则 13](#_Toc497732845)

[3.2 一般规范 14](#_Toc497732846)

[3.2.1 文件/资源命名 14](#_Toc497732847)

[3.2.2 文本缩进 14](#_Toc497732848)

[3.2.3 注释 15](#_Toc497732849)

[3.3 编码规范 16](#_Toc497732850)

[3.3.1 HTML 16](#_Toc497732851)

[3.3.2 JavaScript 30](#_Toc497732852)

[3.3.3 CSS 42](#_Toc497732853)

[3.4 其他规范 49](#_Toc497732854)

[3.4.1 React 49](#_Toc497732855)

[3.4.2 MobX 57](#_Toc497732856)

[附录AAA 57](#_Toc497732857)

# 概述

## 期望目的

此文档旨在提高工作效率，便于后台人员添加功能及前端后期优化维护，输出高质量的文档，在项目开发中，使结构更加清晰，代码简明有序，有一个更好的前端架构。

我们知道，当一个团队开始指定并实行编码规范的话，错误就会变得更加显而易见。如果一段特定的代码不符合规范的话，它有可能只是代码风格错误，而也有可能会是 bug。

早期指定规范就使得代码审核得以更好的开展，并且可以更精确的地定位到错误。只要开发者们能够保证源代码源文件都严格遵循规范，那接下去所使用的混淆、压缩和编译工具则可投其所好不尽相同。

## 适用人群

此文档适用于本公司，现在正在开发的及以后要开发的所有涉及到前端开发的开发人员。

**进行前端开发之前请务必阅读此文档，并严格按照此文档中写明的规范及流程进行开发。**

# 工程

## 工程构建

使用npm进行包管理、使用webpack模块化构建工具，进行模块化开发及代码编译、代码发布、功能调试等操作。

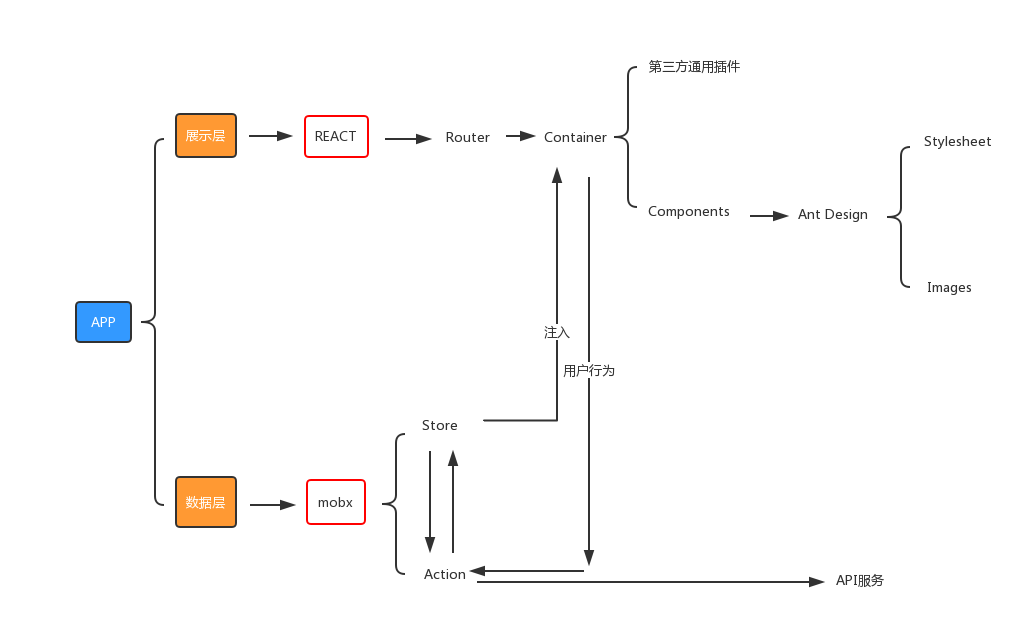
工程初始化时，使用npm命令安装依赖包。在开发过程中如果安装了功能开发所需要的插件时，按照插件的属性安装到dependencies或devDependencies域下，安装完成要把package.json提交，以免其他开发者出现加载不到插件的问题。

webpack.config.js是webpack的配置文件，webpack会根据其中的配置进行文件编译及工程打包，如果需要安装新模块配置，可以自行修改配置文件，并提交版本服务器。

第一次运行需要打包vendor，仅运行一次。当依赖改变时需要重新打包。

（参考附录）

## 前端架构



前端工程可以根据具体的业务场景来斟酌是否使用以上（React+Mobx+Antd）结构搭建，需达到统一框架结构，易于维护的目的。

可以根据具体的业务场景，业务规模来选择使用React-router来做静态路由，并且应该遵循以下原则：

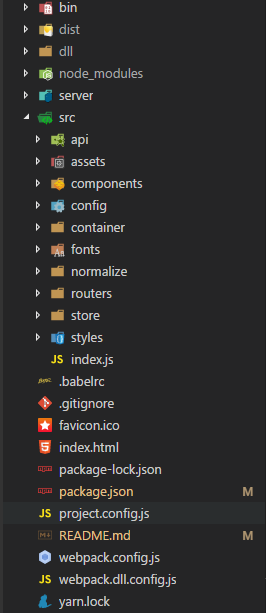
1. 如果系统功能模块很多，而且每个功能模块中存在较多的子模块（二级导航），应该尽量在二级导航的各个功能页面中使用静态路由，在一级大功能模块之间使用入口文件控制，进行静态页面跳转；
2. 使用路由时尽量采用Hash路由方式；

前端调用后台RESTFul接口进行数据交互，接口提供方须提供完善的接口说明文档，接口要返回类型准确、字段清晰的数据。

## 开发测试

进行页面交互逻辑开发之前，应该先按照产品文档编写视觉效果静态页面，交互开发时，直接用写好的静态页面来做组件划分及逻辑编写。

一般情况下我们应遵循以下目录结构：



目录说明：

bin/ 服务入口文件；

dist/ 工程打包后的文件目录；

dll/ dll插件打包vendor生成的文件目录；

node\_modules/ npm插件安装包；

server/ 服务文件；

src/ 源码目录：

api/ 数据请求接口定义；

assets/ 静态数据资源；

components/ 系统中自己的编写的组件目录；

config/ 配置文件目录；

container/ 各个组件的容器目录；

normalize/ 浏览器标准化；

routers/ 每个一级导航模块的入口文件目录；

store/ 数据仓库目录；

styles/ 页面样式文件目录；

功能开发时，应该在命令行工具中，cd到工程的根目录，执行npm start命令，此时会启动一个开发服务器，使用8080（可以在项目自定义）端口，通过访问[http://localhost:8080](http://localhost:8080/page/) 打开对应的文件（当前项目设置了运行后自动打开浏览器页面），进行功能的开发。

## 详细说明

1. routers：

此目录中的文件对应工程中大的栏目（一级目录）的路由入口文件。

1. container

此目录中的每个目录名称，应该对应routers中的文件名称。page目录中的每个的页面引用的为此目录中对应的文件。

每个二级目录的文件为一个Container组件，对应的是每个栏目的二级目录（即左侧导航中的每一项，或者其他操作时，hash路由对应的页面），每个Container是由若干个Component组成。

开发一个Container组件时，应首先按照开发规范在正确的目录创建一个合法的JS或者JSX文件，文件名称和Container的名称应该保持一致。在文件的开头部分应该首先导入必须的依赖，如：

import React, { Component } from 'react';

import PropTypes from 'prop-types';

import { Provider, observer, inject } from 'mobx-react';

然后导入Container中的组件（包括自己编写或第三方提供）如：

import RuleSetList from 'component/ruleSet/RuleSetList';

定义组件时应使用ES6语法来创建，继承Component，React的版本在15.3以上时应该继承Component，如：

class List extends Component

1. component

Component同Container的一样，为一个功能模块，一个React组件，构建方式也和Container一样。

common目录中的组件为公用组件，每个栏目的Container均可使用；其他的对应目录中为该文件栏目的专属组件。

1. store

此目录为数据仓库目录，通过文件的名称来定义对应Container的数据仓库，每个大栏目使用一个，其中有个全局的通用store，任何Container都可以调用这个数据仓库。

每个数据仓库中必须导入mobx组件中的observable和action对象，通过@observable修饰符来定义一个对象是可被组件（Container或Component）订阅的；@action修饰符来将个函数定义为一个action，用来修改observable对象的值（严格模式时必须用action来修改observable对象的值，详细mobx教程）。

1. 连接Store和Container

container每个子目录中都有一个index或者指定文件，在routers中通过import导入此文件。

在container每个子目录的index或者指定文件中，使用mobx-react组件中的Provider组件来连接、组合Container，并把Store对象传给Provider组件：

<Provider store={store}>

<div className='pannel'>

</div>

</Provider>

在定义Container类时，通过导入mobx-react组件来注入store：

import { observer, inject } from 'mobx-react'；

@inject('store') @observer

顺序不能颠倒，且保证@observer在最后注入才可订阅成功

class List extends Component {}

1. 组件的交互

React的组件在进行交互、通信时通过props的形式进行，每个组件都有自己的props，定义的方式为：

List.propTypes = {

name: PropTypes.string

};

List.defaultProps= {

Name: ‘’

};

子组件给父组件传值时使用函数类型的props：

Item.propTypes = {

onSearch: PropTypes.func

};

Item.defaultProps = {

onSearch: () => {}

}

定义一个组件时，一定要事先约定好所包含的props，以供其他人方便查阅。

## 打包发布

功能开发完成后可以在工程根目录，通过 npm run build目录进行打包，文件会打包到dist/目录。

## 发布一个NPM包

当需要发布一个自己的npm包时，首先我们应该先注册一个npm账号，然后在工程目录中执行命令行，npm init，一步步填写相关信息，编写我们自己的组件或组件库，然后使用npm adduser输入自己的账号，npm publish发布代码到npm 服务器。

值得注意的是，每次有更新后需要提交新的代码时，要修改package.json中的 version，才可以发布成功。

# 规范

## 基本准则

编写符合web标准，具有语义的标签，使结构、表现、行为分离，兼容性优良。进行页面性能优化，代码简洁、明了、有序，尽可能的减少服务器的负载，保证最快的解析速度。

## 一般规范

### 文件/资源命名

在 web 项目中，所有的文件名应该都遵循同一命名约定。以可读性而言，减号（-）是用来分隔文件名的不二之选。同时它也是常见的 URL 分隔符（i.e. //example.com/blog/my-blog-entry or //s.example.com/images/big-black-background.jpg），所以理所当然的，减号应该也是用来分隔资源名称的好选择。

**文件夹：**所有的文件夹应该尽量使用一个小写单词做为名称，如果迫不得已使用多个单词命名的，应该使用减号（-）分隔，如report、user-data等。

**React组件、Store数据仓库：**所有的**React组件、Store数据仓库**文件名称都要和组件的名称保持一致，应该以大写字母开头，每一个单词开头字母都要大写，如：List.js、UserData.js等。

**其他资源文件：**所有的其他文件如图片、样式、字体等资源的命名规则都要遵循和文件夹一样的命名规则。

### 文本缩进（可使用ide的美化插件即可）

一次缩进四个空格。



### 注释

注释是你自己与你的小伙伴们了解代码写法和目的的唯一途径。特别是在写一些看似琐碎的无关紧要的代码时，由于记忆点不深刻，注释就变得尤为重要了。

编写自解释代码只是一个传说，没有任何代码是可以完全自解释的。而代码注释，则是永远也不嫌多。

当你写注释时一定要注意：不要写你的代码都干了些什么，而要写你的代码为什么要这么写，背后的考量是什么。当然也可以加入所思考问题或是解决方案的链接地址。



## 编码规范

### HTML

#### 文档类型

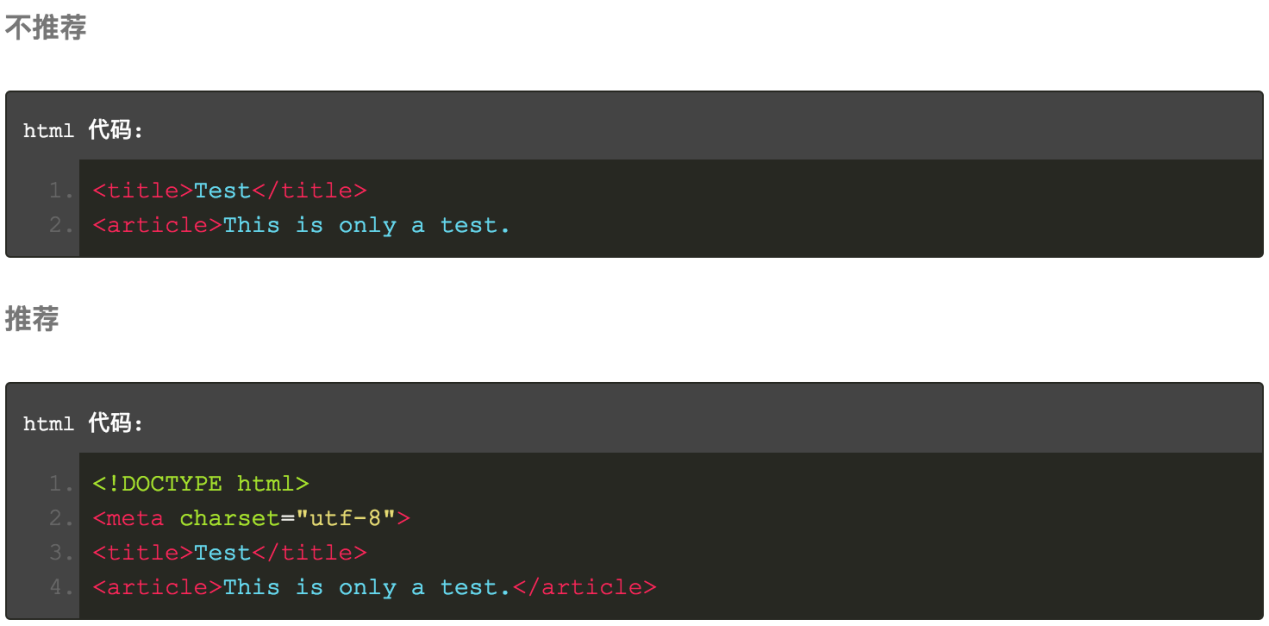
推荐使用 HTML5 的文档类型申明： <!DOCTYPE html>。（建议使用 text/html 格式的 HTML。避免使用 XHTML。XHTML 以及它的属性，比如 application/xhtml+xml 在浏览器中的应用支持与优化空间都十分有限）。

#### HTML 验证

一般情况下，建议使用能通过标准规范验证的 HTML 代码，除非在性能优化和控制文件大小上不得不做出让步。

使用诸如 W3C HTML validator 这样的工具来进行检测。

规范化的 HTML 是显现技术要求与局限的显著质量基线，它促进了 HTML 被更好地运用。



#### 省略可选标签

HTML5 规范中规定了 HTML 标签是可以省略的。但从可读性来说，在开发的源文件中最好不要这样做，因为省略标签可能会导致一些问题。

省略一些可选的标签确实使得页面大小减少，这很有用，尤其是对于一些大型网站来说。为了达到这一目的，我们可以在开发后期对页面进行压缩处理，在这个环节中这些可选的标签完全就可以省略掉了。

#### 脚本加载

一段脚本放置在 <head> 内，比如 <script src="main.js"></script>，其加载会一直阻塞 DOM 解析，直至它完全地加载和执行完毕。这会造成页面显示的延迟。特别是一些重量级的脚本，对用户体验来说那真是一个巨大的影响。

脚本引用写在 body 结束标签之前，这就大大降低了其阻塞影响。

（延伸： 所有具有src的标签都会阻塞 DOM 解析）



#### 语义化

根据元素（有时被错误地称作“标签”）其被创造出来时的初始意义来使用它。打个比方，用 heading 元素来定义头部标题，p 元素来定义文字段落，用 a 元素来定义链接锚点，等等。

有根据有目的地使用 HTML 元素，对于可访问性、代码重用、代码效率来说意义重大。



**推荐**

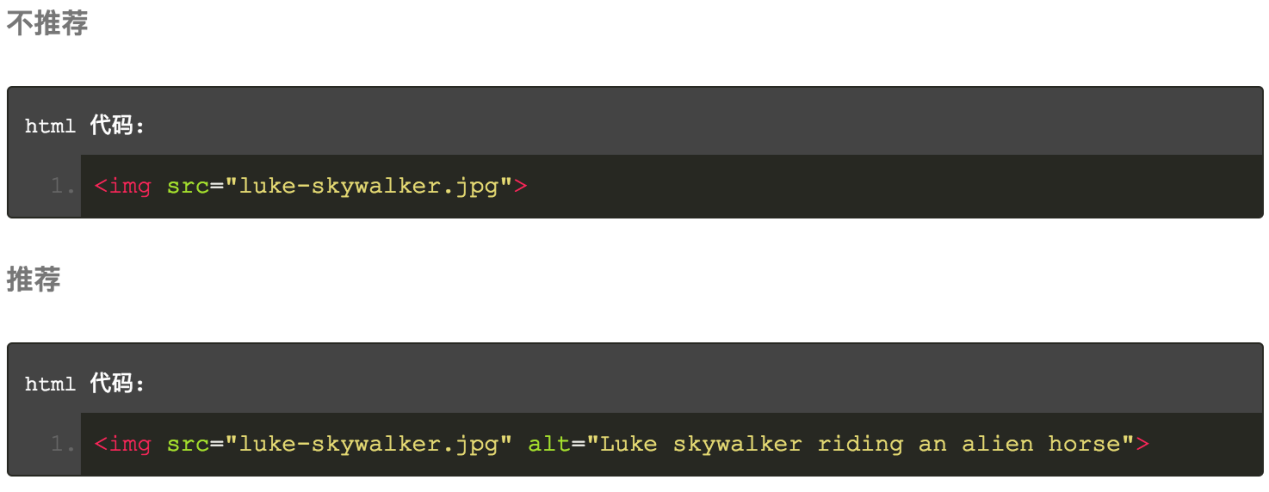
1. <!-- The page header should go into a header element -->
2. <header>
3. <!-- As this title belongs to the page structure it's a heading and h1 should be used -->
4. <h1>My page title</h1>
5. </header>
7. <!-- All navigation should go into a nav element -->
8. <nav class="top-navigation">
9. <!-- A listing of elements should always go to UL (OL for ordered listings) -->
10. <ul>
11. <li class="nav-item"><a href="#home">Home</a></li>
12. <li class="nav-item"><a href="#news">News</a></li>
13. <li class="nav-item"><a href="#about">About</a></li>
14. </ul>
15. </nav>
17. <!-- The main part of the page should go into a main element (also use role="main" for accessibility) -->
18. <main class="news-page" role="main">
19. <!-- A section of a page should go into a section element. Divide a page into sections with semantic elements. -->
20. <section class="page-section news">
21. <!-- A section header should go into a section element -->
22. <header>
23. <!-- As a page section belongs to the page structure heading elements should be used (in this case h2) -->
24. <h2 class="title">All news articles</h2>
25. </header>
27. <!-- If a section / module can be seen as an article (news article, blog entry, products teaser, any other
28. re-usable module / section that can occur multiple times on a page) a article element should be used -->
29. <article class="news-article">
30. <!-- An article can contain a header that contains the summary / introduction information of the article -->
31. <header>
32. <!-- As a article title does not belong to the overall page structure there should not be any heading tag! -->
33. <div class="article-title">Good article</div>
34. <!-- Small can optionally be used to reduce importance -->
35. <small class="intro">Introduction sub-title</small>
36. </header>
38. <!-- For the main content in a section or article there is no semantic element -->
39. <div class="content">
40. <p>This is a good example for HTML semantics</p>
41. </div>
42. <!-- For content that is represented as side note or less important information in a given context use aside -->
43. <aside class="article-side-notes">
44. <p>I think I'm more on the side and should not receive the main credits</p>
45. </aside>
46. <!-- Articles can also contain footers. If you have footnotes for an article place them into a footer element -->
47. <footer class="article-foot-notes">
48. <!-- The time element can be used to annotate a timestamp. Use the datetime attribute to specify ISO time
49. while the actual text in the time element can also be more human readable / relative -->
50. <p>This article was created by David <time datetime="2014-01-01 00:00" class="time">1 month ago</time></p>
51. </footer>
52. </article>
54. <!-- In a section, footnotes or similar information can also go into a footer element -->
55. <footer class="section-footer">
56. <p>Related sections: Events, Public holidays</p>
57. </footer>
58. </section>
59. </main>
61. <!-- Your page footer should go into a global footer element -->
62. <footer class="page-footer">
63. Copyright 2014
64. </footer>

#### 多媒体回溯

对页面上的媒体而言，像图片、视频、canvas 动画等，要确保其有可替代的接入接口。图片文件我们可采用有意义的备选文本（alt），视频和音频文件我们可以为其加上说明文字或字幕。

提供可替代内容对可用性来说十分重要。试想，一位盲人用户如何能知晓一张图片是什么，要是没有 @alt 的话。

（图片的 alt 属性是可不填写内容的，纯装饰性的图片就可用这么做：alt=""）。



#### 关注点分离

理解 web 中如何和为何区分不同的关注点，这很重要。这里的关注点主要指的是：信息（HTML 结构）、外观（CSS）和行为（JavaScript）。为了使它们成为可维护的干净整洁的代码，我们要尽可能的将它们分离开来。

严格地保证结构、表现、行为三者分离，并尽量使三者之间没有太多的交互和联系。

就是说，尽量在文档和模板中只包含结构性的 HTML；而将所有表现代码，移入样式表中；将所有动作行为，移入脚本之中。

在此之外，为使得它们之间的联系尽可能的小，在文档和模板中也尽量少地引入样式和脚本文件。

清晰的分层意味着：

不使用超过一到两张样式表（i.e. main.css, vendor.css）

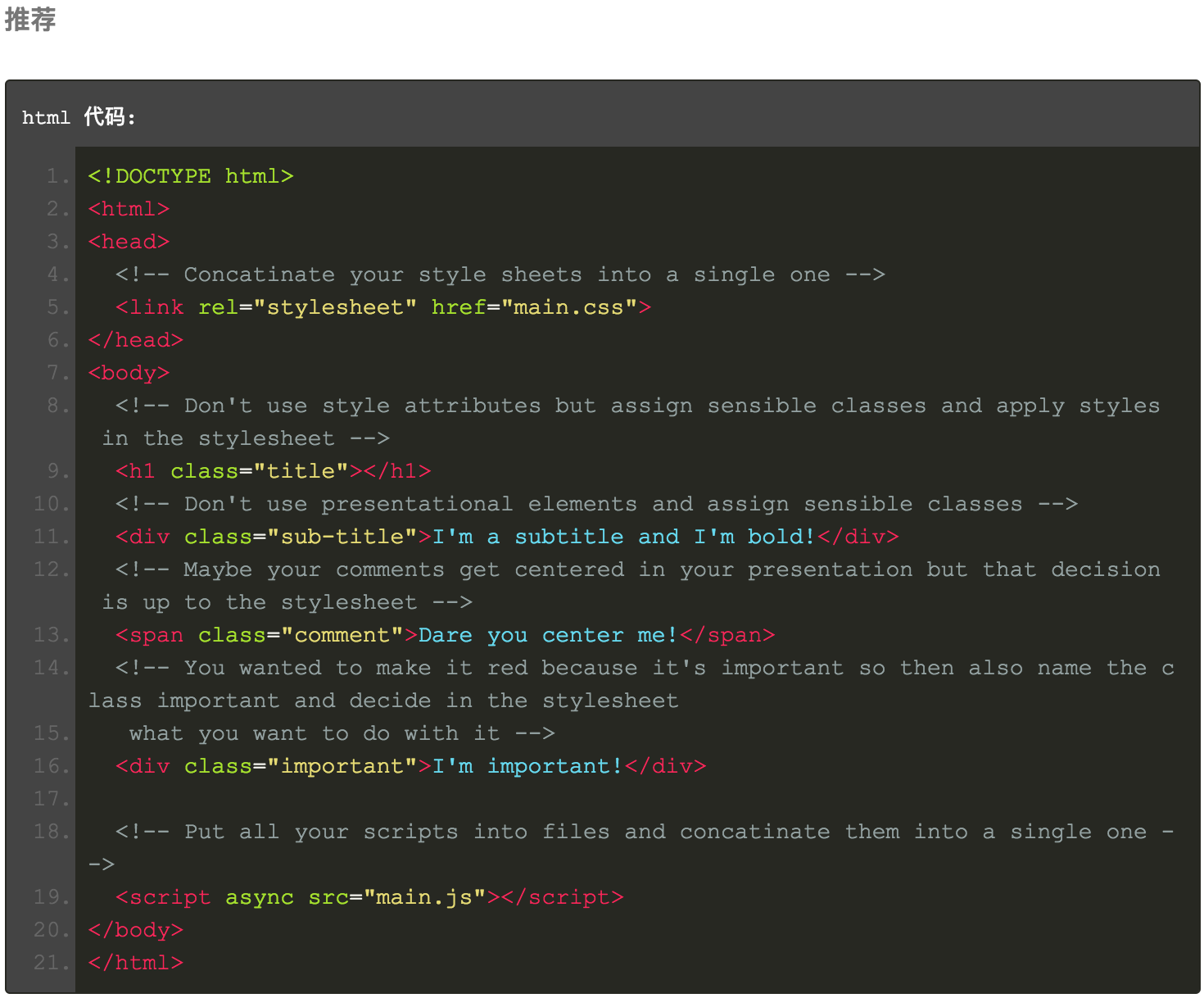
不使用超过一到两个脚本（学会用合并脚本）

不使用行内脚本（<script>alert('no good')</script>）

不使用表象元素（i.e. <b>, <u>, <center>, <font>, <b>）

不使用表象 class 名（i.e. red, left, center）





#### HTML 内容至上

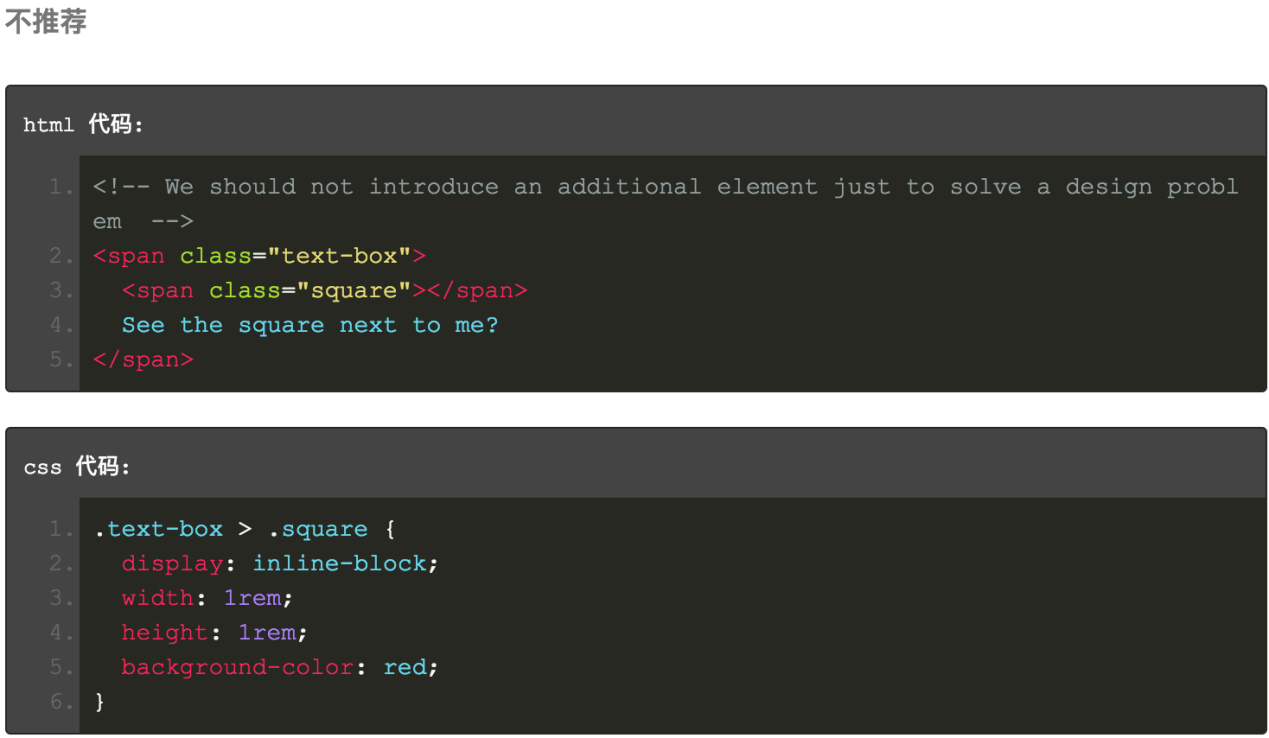
不要让非内容信息污染了你的 HTML。现在貌似有一种倾向：通过 HTML 来解决设计问题，这是显然是不对的。HTML 就应该只关注内容。

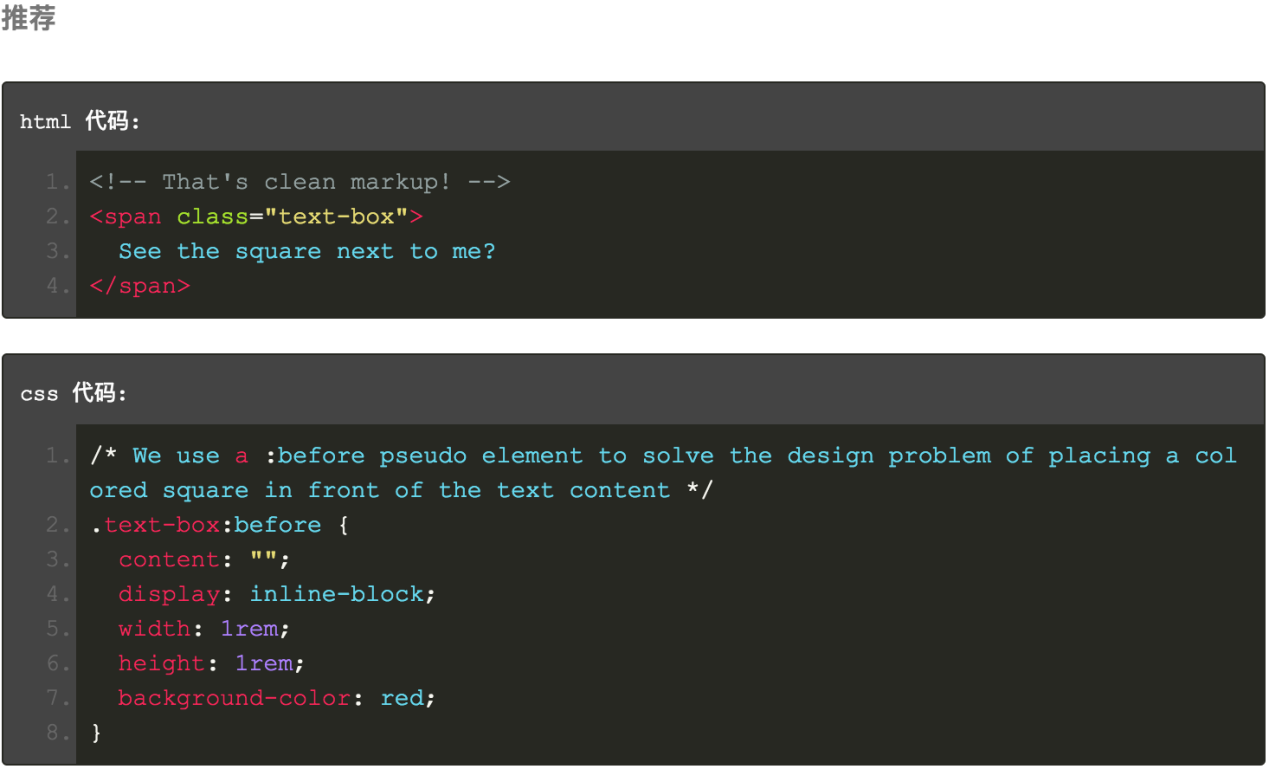
HTML 标签的目的，就是为了不断地展示内容信息。

不要引入一些特定的 HTML 结构来解决一些视觉设计问题

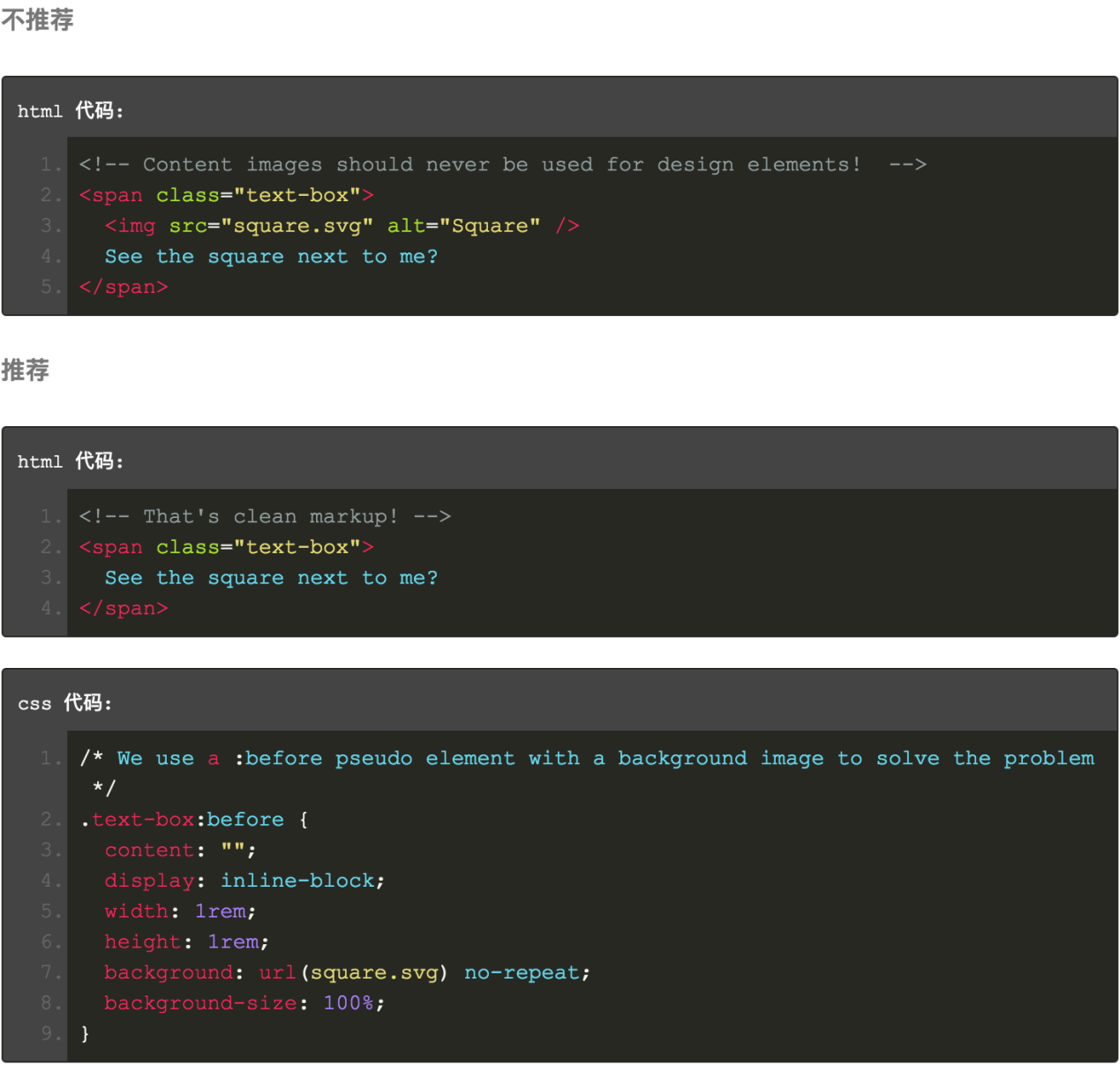
不要将 img 元素当做专门用来做视觉设计的元素

以下例子展示了误将 HTML 用来解决设计问题的这两种情况：





图片和 SVG 图形能被引入到 HTML 中的唯一理由是它们呈现出了与内容相关的一些信息。



#### Type 属性

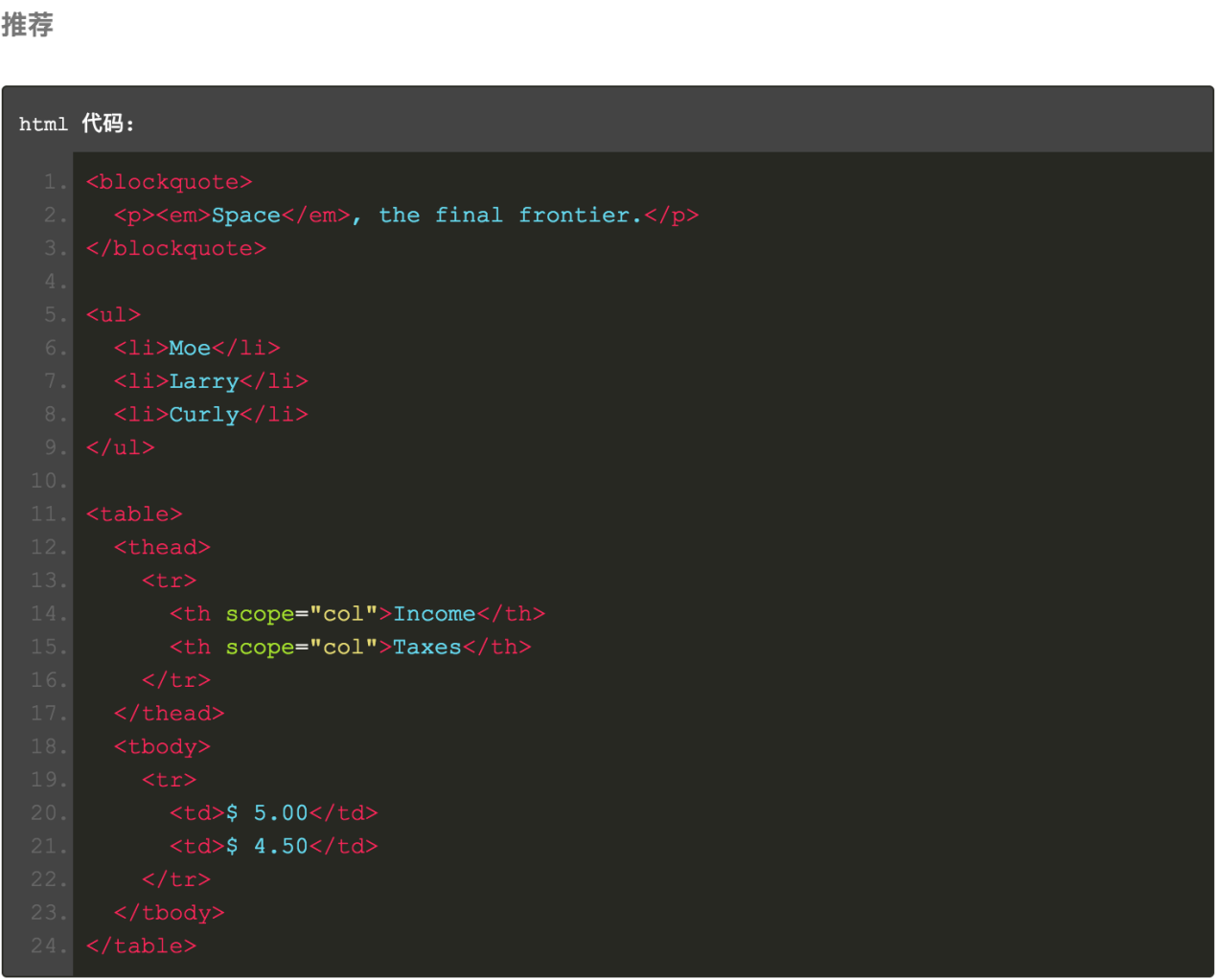
省略样式表与脚本上的 type 属性。鉴于 HTML5 中以上两者默认的 type 值就是 text/css 和 text/javascript，所以 type 属性一般是可以忽略掉的。甚至在老旧版本的浏览器中这么做也是安全可靠的。



#### 格式化规则（可以用ide的美化插件或者格式化即可）

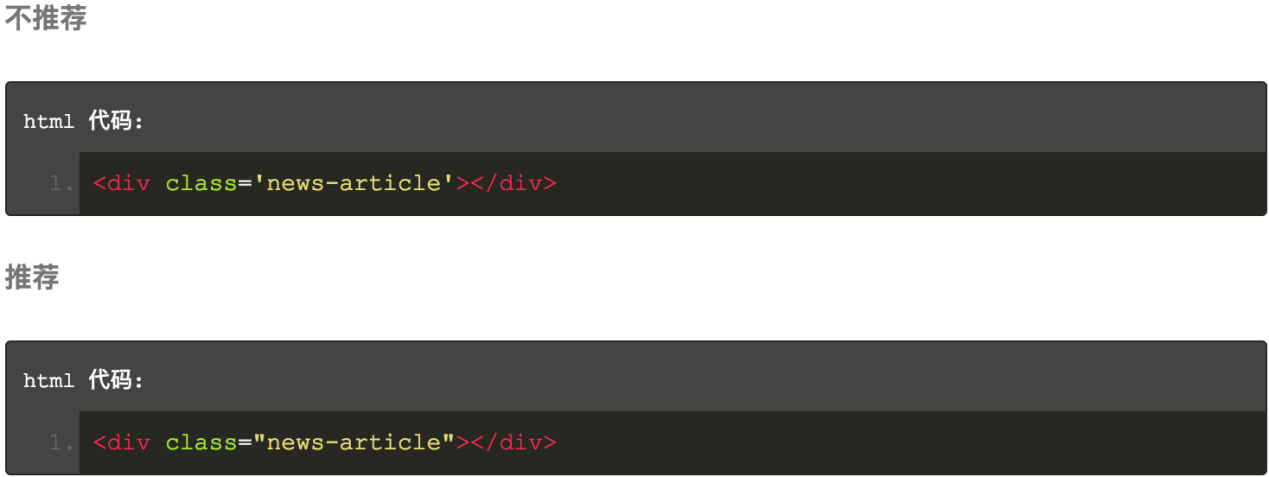
在每一个块状元素，列表元素和表格元素后，加上一新空白行，并对其子孙元素进行缩进。内联元素写在一行内，块状元素还有列表和表格要另起一行。

（如果由于换行的空格引发了不可预计的问题，那将所有元素并入一行也是可以接受的，格式警告总好过错误警告）。



#### HTML 引号

使用双引号(“”) 而不是单引号(‘’) 。



### JavaScript

#### 严格模式

ECMAScript 5 严格模式可在整个脚本或独个方法内被激活。它对应不同的 javascript 语境会做更加严格的错误检查。严格模式也确保了 javascript 代码更加的健壮，运行的也更加快速。

严格模式会阻止使用在未来很可能被引入的预留关键字。

你应该在你的脚本中启用严格模式，最好是在独立的 IIFE 中应用它。避免在你的脚本第一行使用它而导致你的所有脚本都启动了严格模式，这有可能会引发一些第三方类库的问题。



#### 变量声明

总是使用 var 来声明变量。如不指定 var，变量将被隐式地声明为全局变量，这将对变量难以控制。如果没有声明，变量处于什么定义域就变得不清（可以是在 Document 或 Window 中，也可以很容易地进入本地定义域）。所以，请总是使用 var 来声明变量。

采用严格模式带来的好处是，当你手误输入错误的变量名时，它可以通过报错信息来帮助你定位错误出处。



#### 总是使用带类型判断的比较判断

总是使用 === 精确的比较操作符，避免在判断的过程中，由 JavaScript 的强制类型转换所造成的困扰。

如果你使用 === 操作符，那比较的双方必须是同一类型为前提的条件下才会有效。

如果你想了解更多关于强制类型转换的信息，你可以读一读 [Dmitry Soshnikov 的这篇文章](http://dmitrysoshnikov.com/notes/note-2-ecmascript-equality-operators/)。

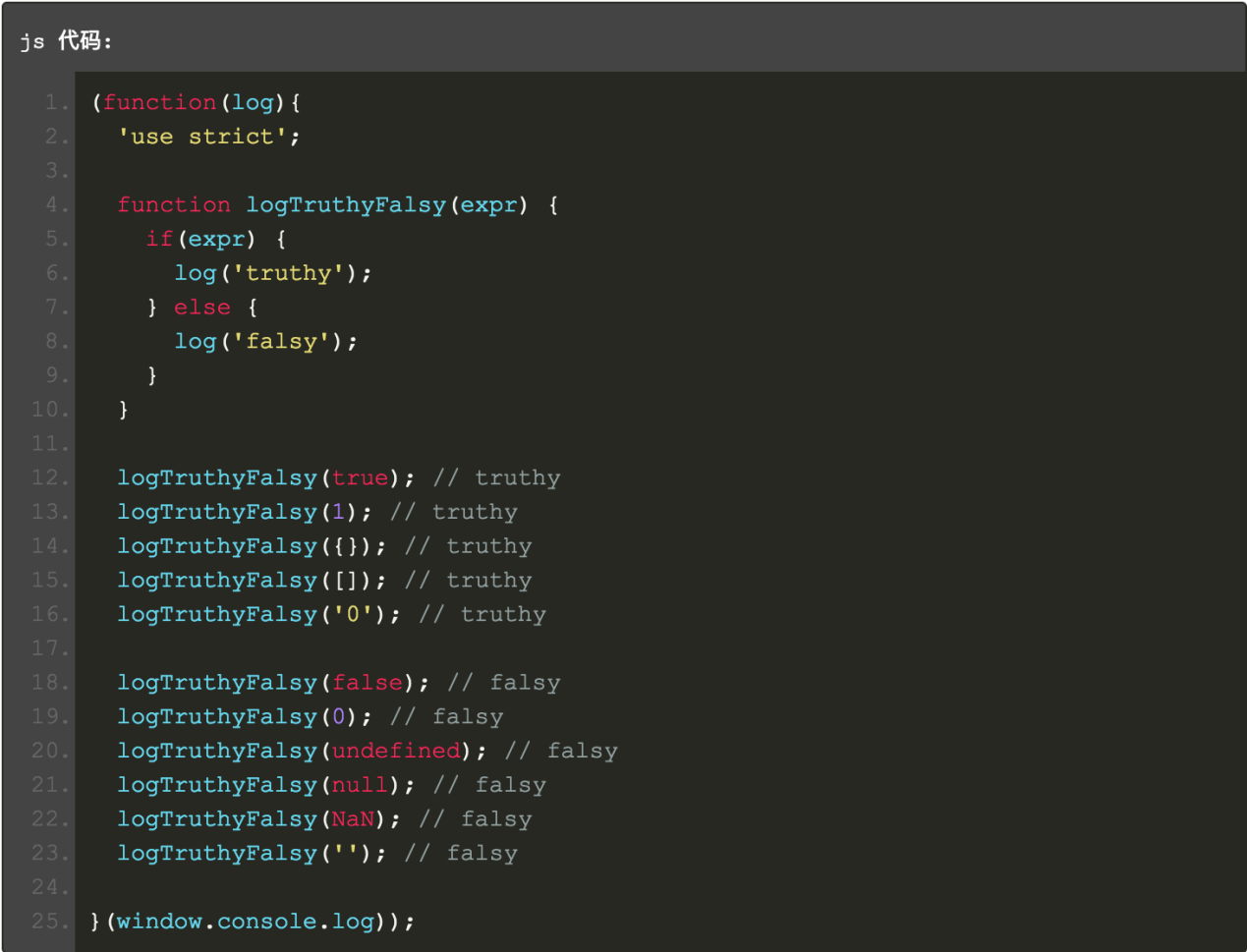
在只使用 == 的情况下，JavaScript 所带来的强制类型转换使得判断结果跟踪变得复杂，下面的例子可以看出这样的结果有多怪了：



#### 明智地使用真假判断

当我们在一个 if 条件语句中使用变量或表达式时，会做真假判断。if(a == true) 是不同于 if(a)的。后者的判断比较特殊，我们称其为真假判断。这种判断会通过特殊的操作将其转换为 true 或 false，下列表达式统统返回 false：false, 0, undefined, null, NaN, ''（空字符串）.

这种真假判断在我们只求结果而不关心过程的情况下，非常的有帮助。以下示例展示了真假判断是如何工作的：



#### 变量赋值时的逻辑操作

逻辑操作符 || 和 && 也可被用来返回布尔值。如果操作对象为非布尔对象，那每个表达式将会被自左向右地做真假判断。基于此操作，最终总有一个表达式被返回回来。这在变量赋值时，是可以用来简化你的代码的。



这一小技巧经常用来给方法设定默认的参数。

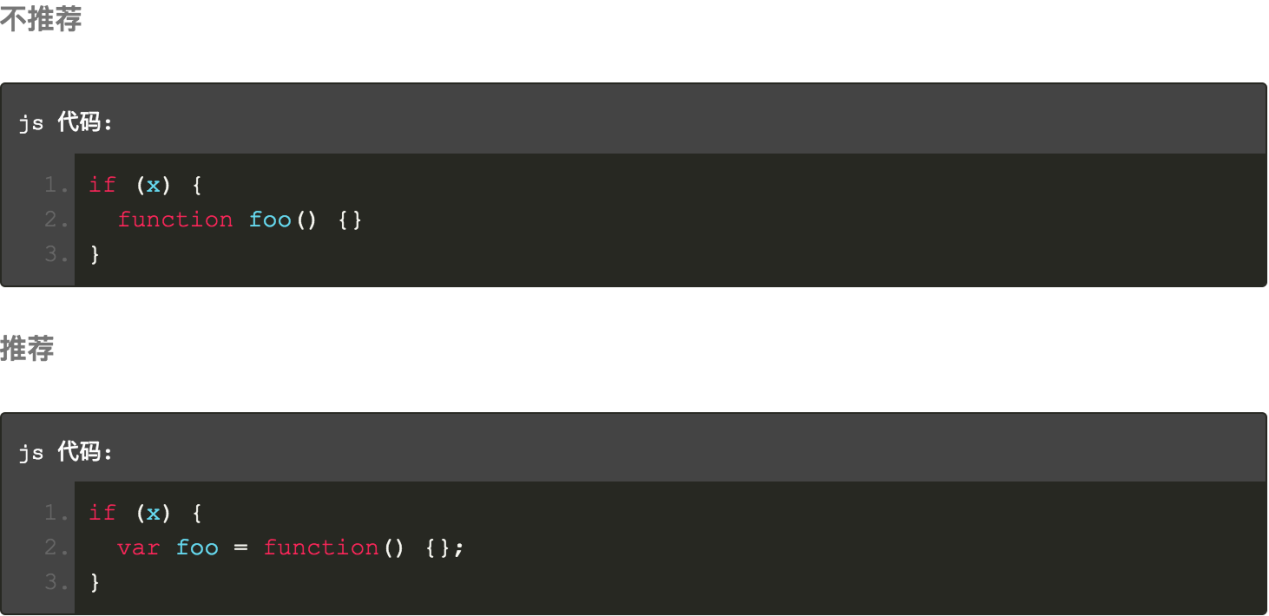


#### 分号

总是使用分号，因为隐式的代码嵌套会引发难以察觉的问题。JavaScript 中语句要以分号结束，否则它将会继续执行下去，不管换不换行。

#### 语句块内的函数声明

切勿在语句块内声明函数，在 ECMAScript 5 的严格模式下，这是不合法的。函数声明应该在定义域的顶层。但在语句块内可将函数申明转化为函数表达式赋值给变量。

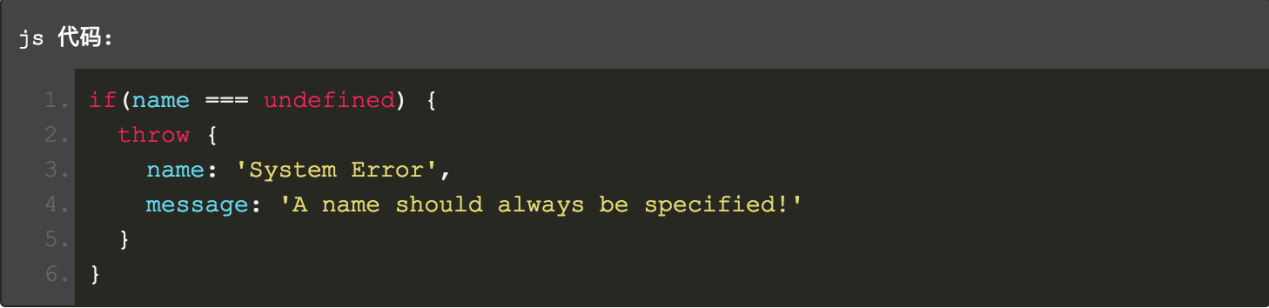


#### 异常

基本上你无法避免出现异常，特别是在做大型开发时（使用应用开发框架等等）。

在没有自定义异常的情况下，从有返回值的函数中返回错误信息一定非常的棘手，更别提多不优雅了。不好的解决方案包括了传第一个引用类型来接纳错误信息，或总是返回一个对象列表，其中包含着可能的错误对象。以上方式基本上是比较简陋的异常处理方式。适时可做自定义异常处理。

在复杂的环境中，你可以考虑抛出对象而不仅仅是字符串（默认的抛出值）。

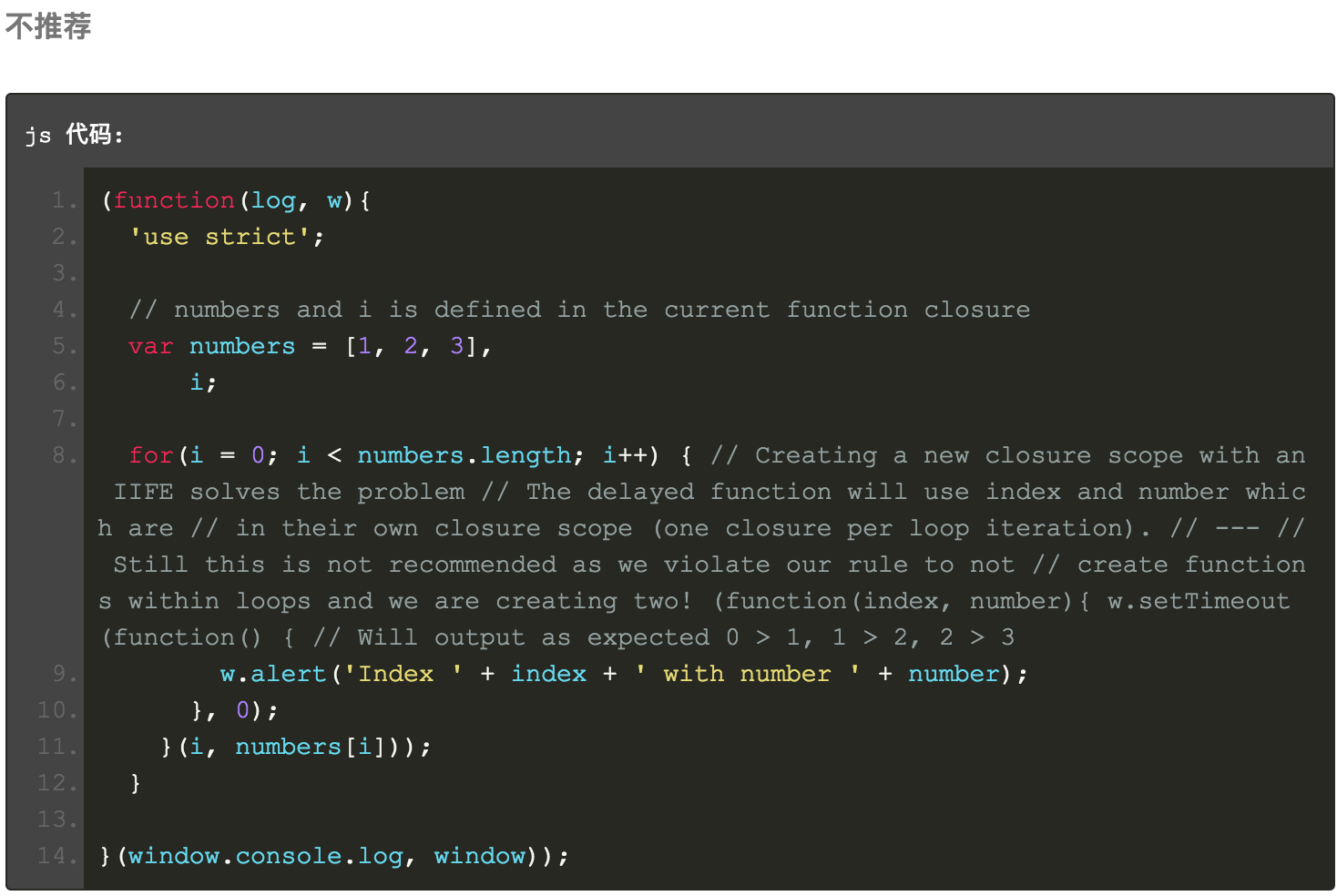


#### 切勿在循环中创建函数

在简单的循环语句中加入函数是非常容易形成闭包而带来隐患的。下面的例子就是一个典型的陷阱：



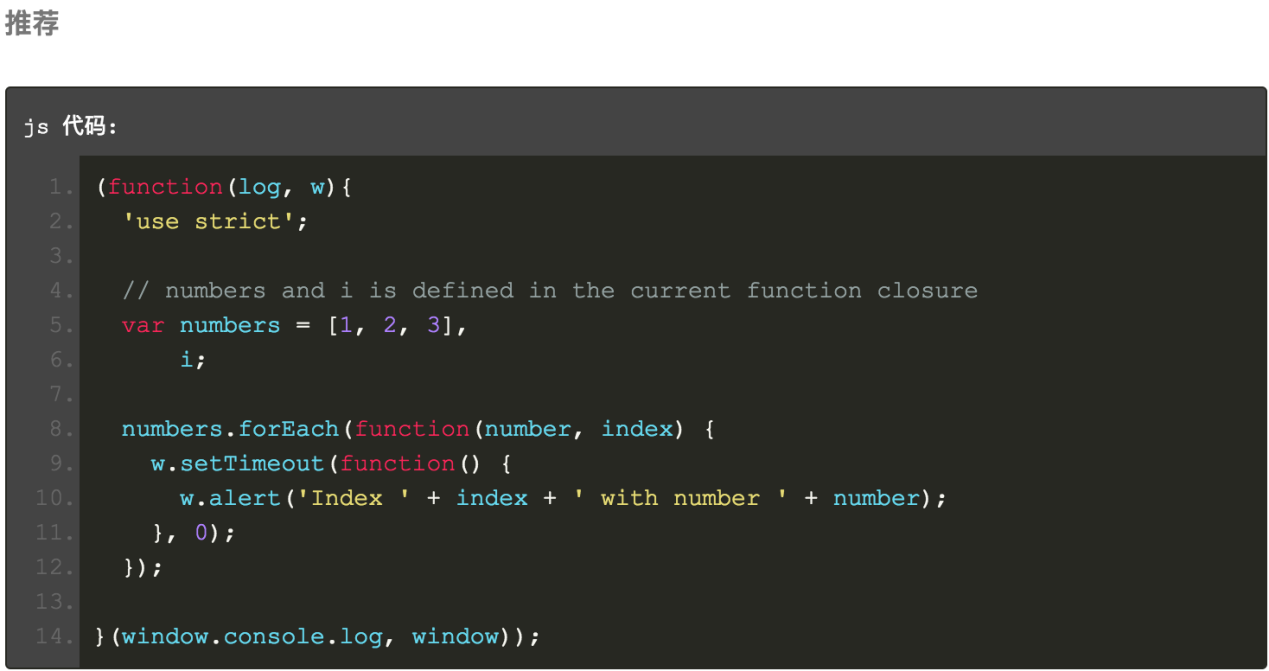
接下来的改进虽然已经解决了上述例子中的问题或 bug，但还是违反了不在循环中创建函数或闭包的原则。



接下来的改进已解决问题，而且也遵循了规范。可是，你会发现看上去似乎过于复杂繁冗了，应该会有更好的解决方案吧。



将循环语句转换为函数执行的方式问题能得到立马解决，每一次循环都会对应地创建一次闭包。函数式的风格更加值得推荐，而且看上去也更加地自然和可预料。



#### eval 函数（魔鬼）

eval() 不但混淆语境还很危险，总会有比这更好、更清晰、更安全的另一种方案来写你的代码，因此尽量不要使用 evil 函数。

#### this 关键字

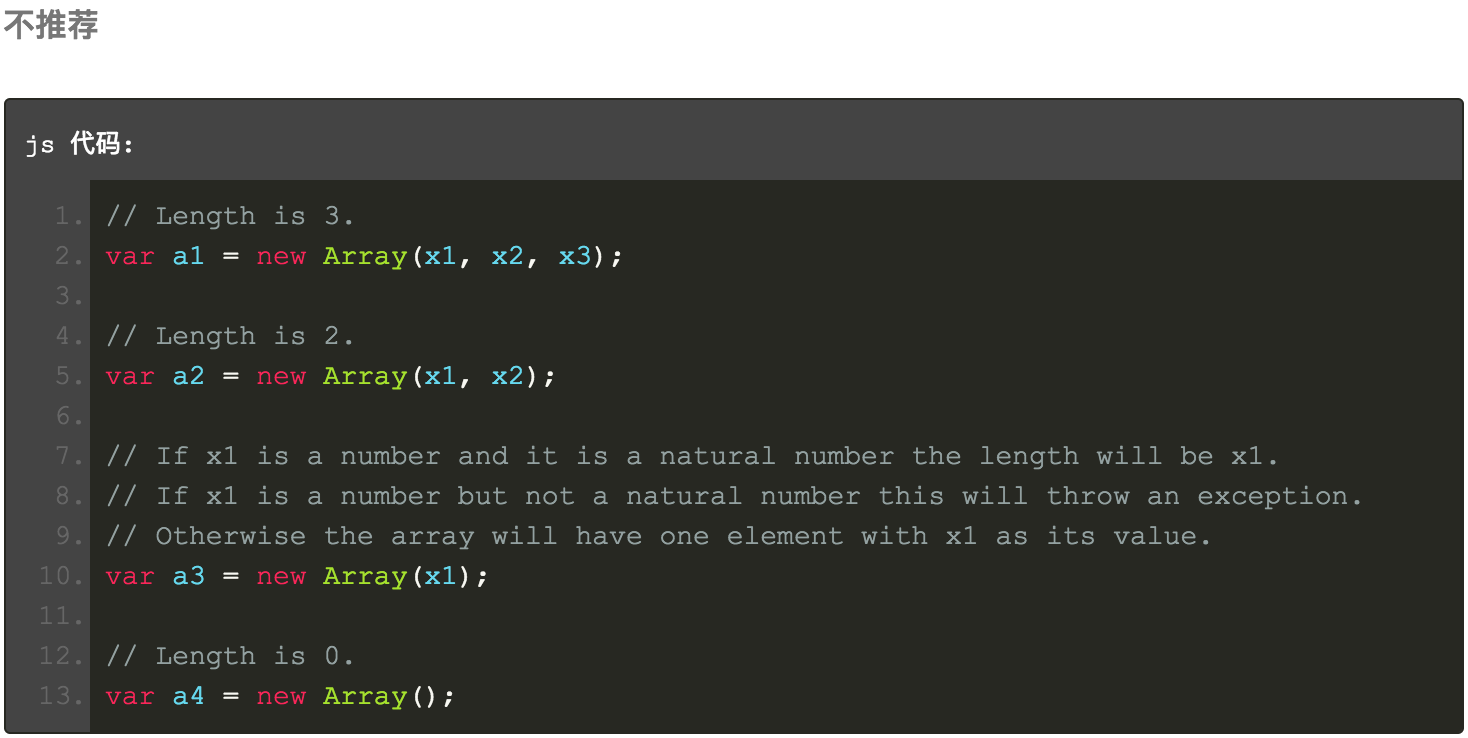
只在对象构造器、方法和在设定的闭包中使用 this 关键字。this 的语义在此有些误导。它时而指向全局对象（大多数时），时而指向调用者的定义域（在 eval 中），时而指向 DOM 树中的某一节点（当用事件处理绑定到 HTML 属性上时），时而指向一个新创建的对象（在构造器中），还时而指向其它的一些对象（如果函数被 call() 和 apply() 执行和调用时）。

正因为它是如此容易地被搞错，请限制它的使用场景：

* 在构造函数中
* 在对象的方法中（包括由此创建出的闭包内）

#### 数组和对象字面量

用数组和对象字面量来代替数组和对象构造器。数组构造器很容易让人在它的参数上犯错。



正因如此，如果将代码传参从两个变为一个，那数组很有可能发生意料不到的长度变化。为避免此类怪异状况，请总是采用更多可读的数组字面量。



对象构造器不会有类似的问题，但是为了可读性和统一性，我们应该使用对象字面量。



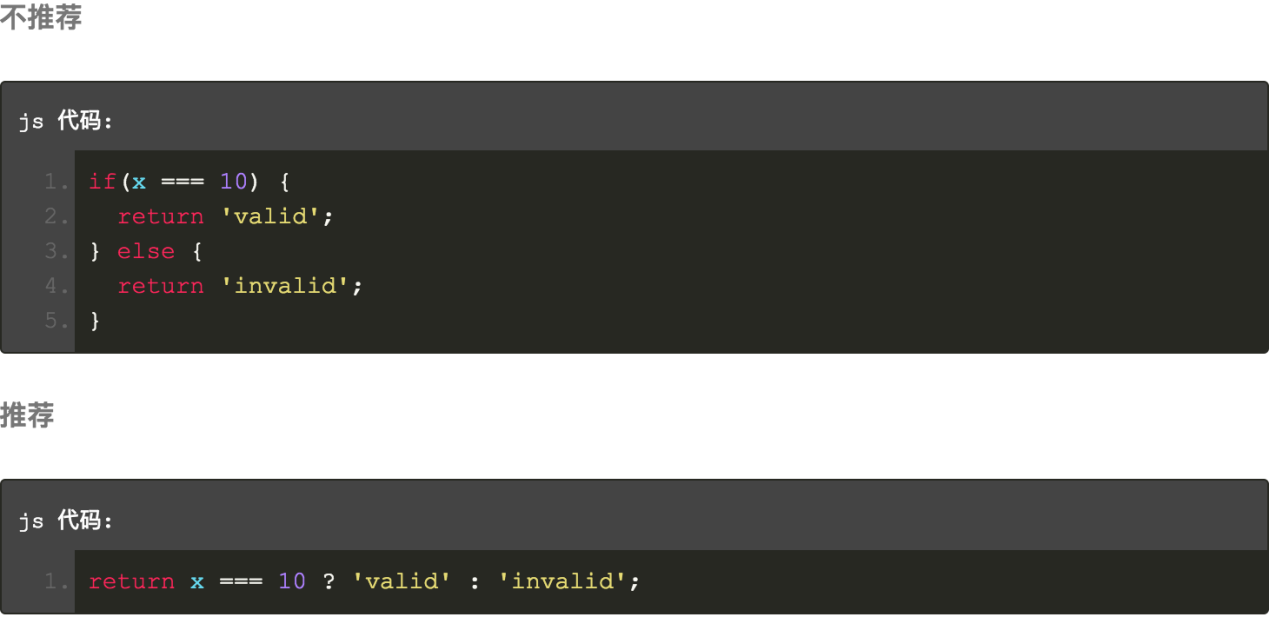


#### 字符串

统一使用单引号(‘)，不使用双引号(“)。这在创建 HTML 字符串非常有好处。

#### 三元条件判断（if 的快捷方法）

用三元操作符分配或返回语句。在比较简单的情况下使用，避免在复杂的情况下使用。没人愿意用 10 行三元操作符把自己的脑子绕晕。



### CSS

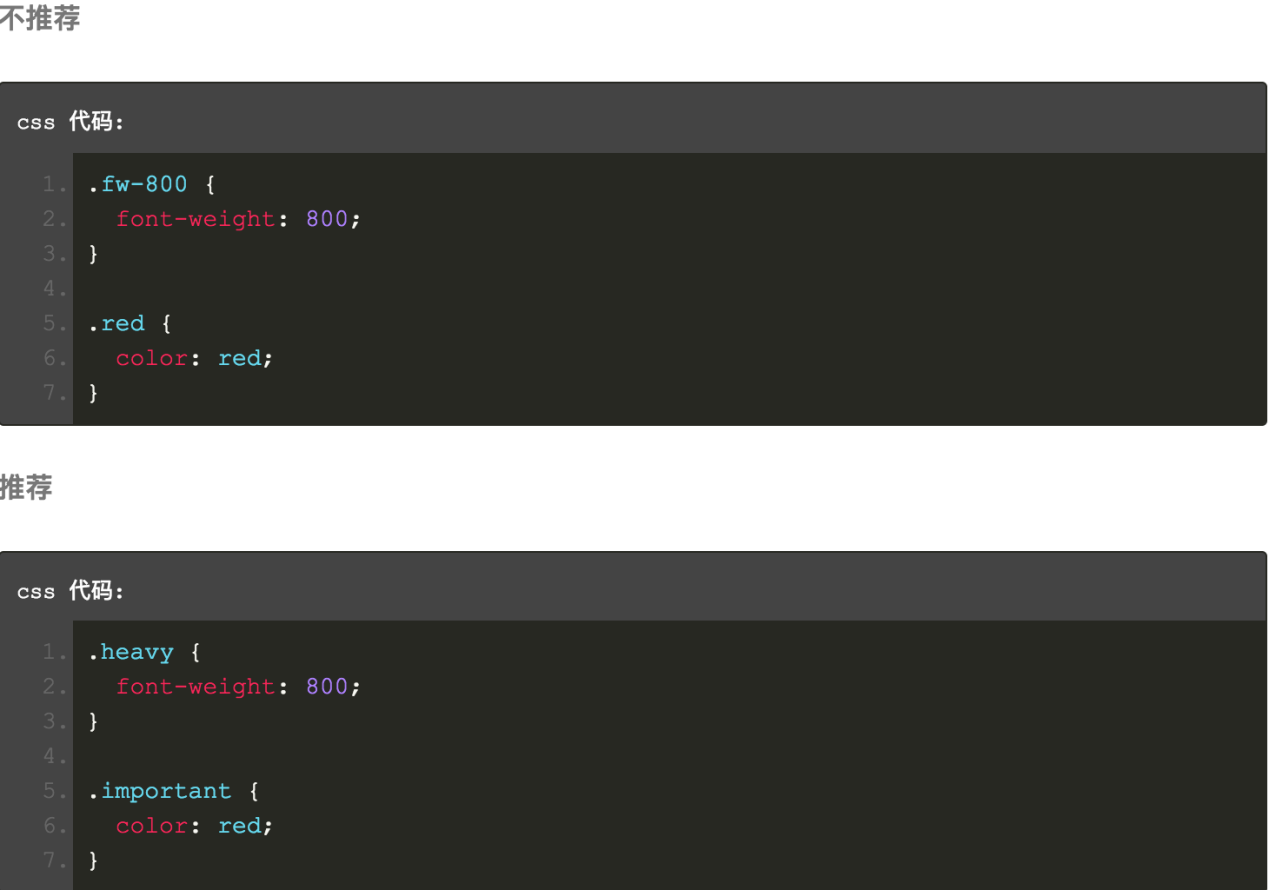
#### 命名

ID和class(类)名总是使用可以反应元素目的和用途的名称，或其他通用名称。代替表象和晦涩难懂的名称。

应该首选具体和反映元素目的的名称，因为这些是最可以理解的，而且发生变化的可能性最小。

通用名称只是多个元素的备用名，他们兄弟元素之间是一样的，没有特别意义。区分他们，使他们具有特殊意义，通常需要为“帮手”。

尽管class(类)名和ID 的语义化对于计算机解析来说没有什么实际的意义，语义化的名称 通常是正确的选择，因为它们所代表的信息含义，不包含表现的限制。



#### 合理的避免使用ID

一般情况下ID不应该被应用于样式。  
ID的样式不能被复用并且每个页面中你只能使用一次ID。  
使用ID唯一有效的是确定网页或整个站点中的位置。  
尽管如此，你应该始终考虑使用class，而不是id，除非只使用一次。



另一个反对使用ID的观点是含有ID选择器权重很高。  
一个只包含一个ID选择器权重高于包含1000个class(类)名的选择器，这使得它很奇怪。



#### CSS选择器中避免标签名

当构建选择器时应该使用清晰， 准确和有语义的class(类)名。不要使用标签选择器。 如果你只关心你的class(类)名  
，而不是你的代码元素，这样会更容易维护。

从分离的角度考虑,在表现层中不应该分配html标记/语义。  
它可能是一个有序列表需要被改成一个无序列表，或者一个div将被转换成article。  
 如果你只使用具有实际意义的class(类)名，并且不使用元素选择器，那么你只需要改变你的html标记，而不用改动你的CSS。



#### ID 和 Class（类）名的分隔符

使用连字符（中划线）分隔ID和Class（类）名中的单词。为了增强课理解性，在选择器中不要使用除了连字符（中划线）以为的任何字符（包括没有）来连接单词和缩写。

另外，作为该标准，预设属性选择器能识别连字符（中划线）作为单词[attribute|=value]的分隔符，所以最好的坚持使用连字符作为分隔符。



#### 声明顺序

这是一个选择器内书写CSS属性顺序的大致轮廓。这是为了保证更好的可读性和可扫描重要。

作为最佳实践，我们应该遵循以下顺序（应该按照下表的顺序）：

结构性属性：

display

position, left, top, right etc.

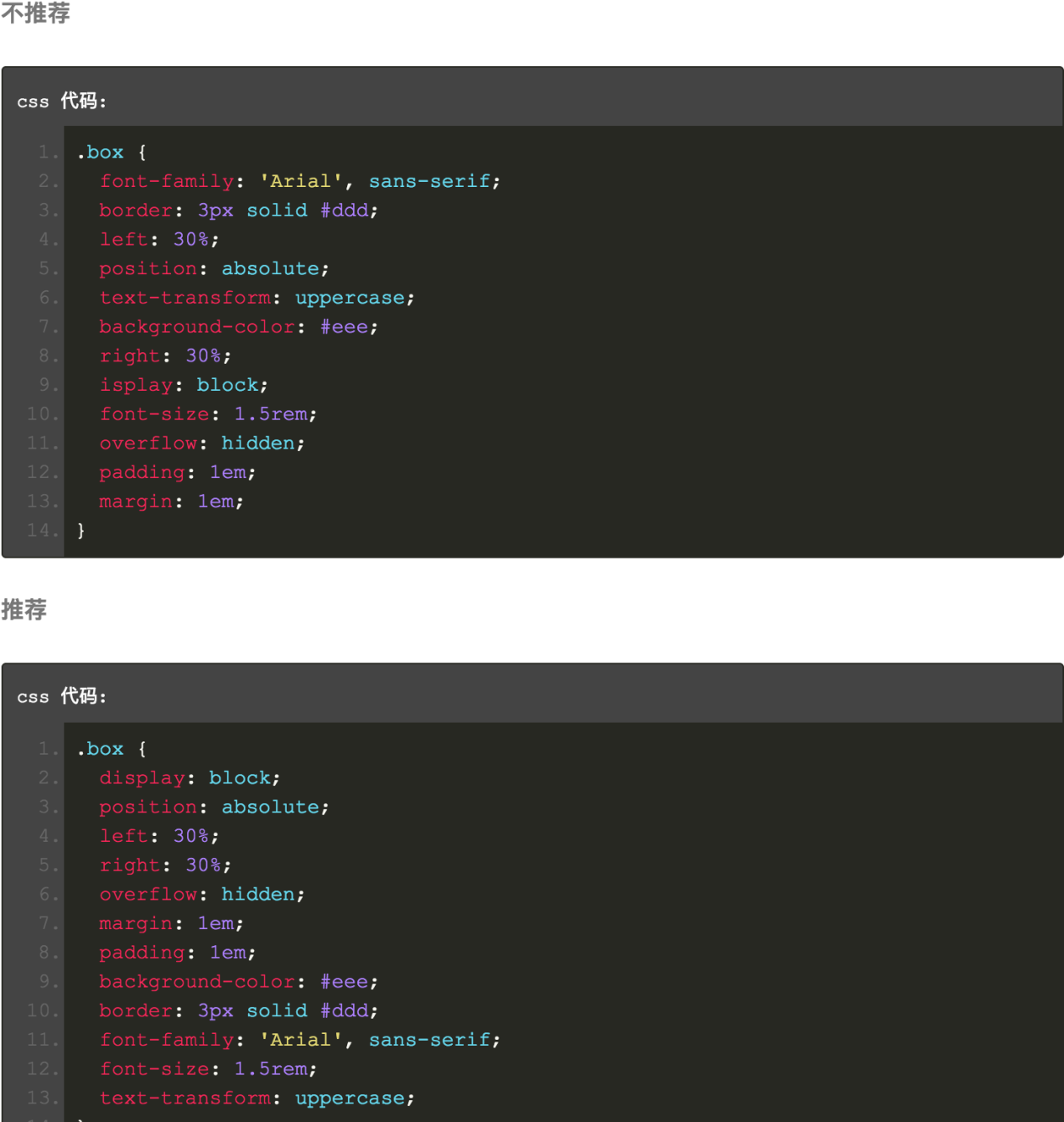
overflow, float, clear etc.

margin, padding

表现性属性：

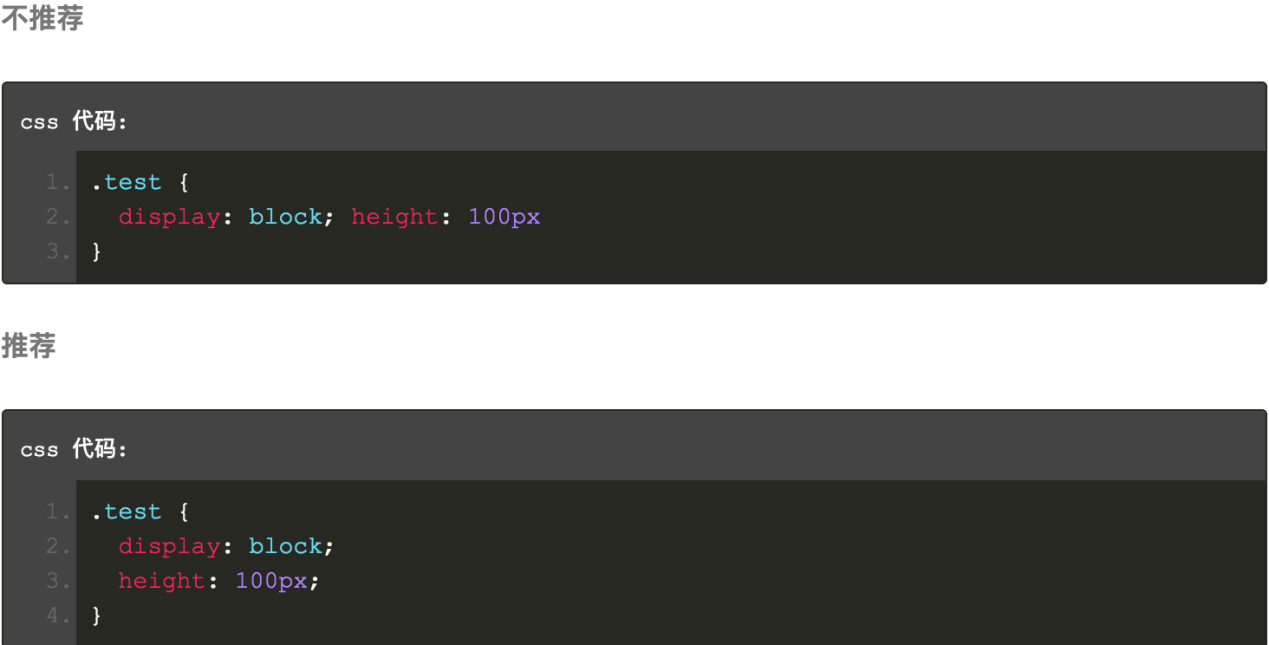
background, border etc.

font, text



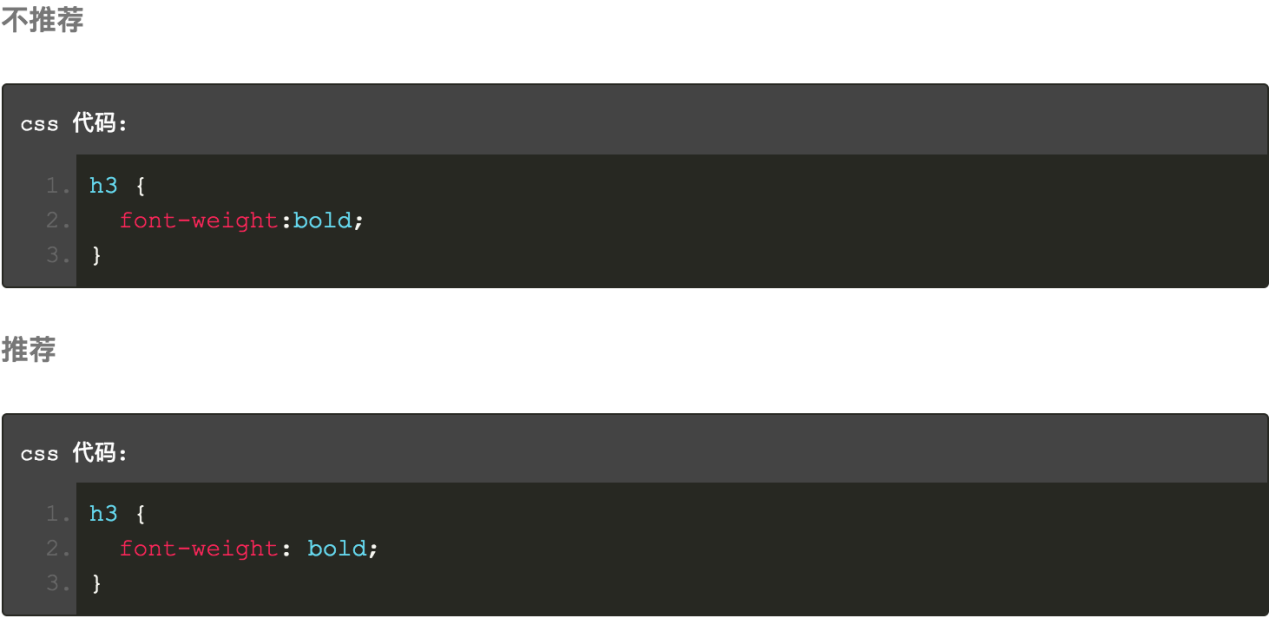
#### 声明结束

为了保证一致性和可扩展性，每个声明应该用分号结束，每个声明换行。



#### 属性名结束

属性名的冒号后使用一个空格。出于一致性的原因，属性和值（但属性和冒号之间没有空格）的之间始终使用一个空格。



#### CSS引号

属性选择器或属性值用双引号（””），而不是单引号（”）括起来。URI值（url()）不要使用引号。



## 其他规范

### React

#### 基本概念

实例化生命周期

getDefaultProps

getInitialState

componentWillMount

render

componentDidMount

更新期生命周期

getInitialState

componentWillMount

render

componentDidMount

运行期生命周期

componentWillReceiveProps

shouldComponentUpdate

componentWillUpdate

render

componentDidUpdate

销毁期生命周期

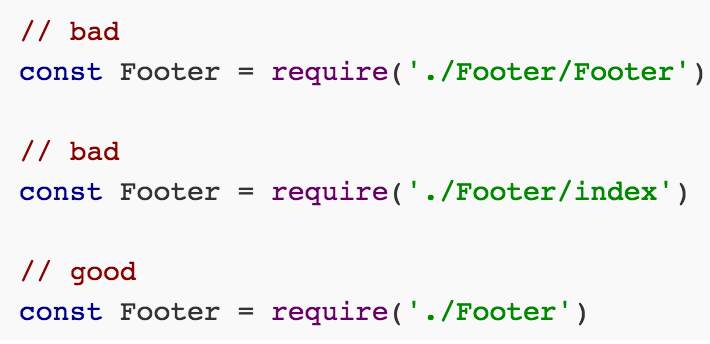
componentWillUnmount

#### 书写规则

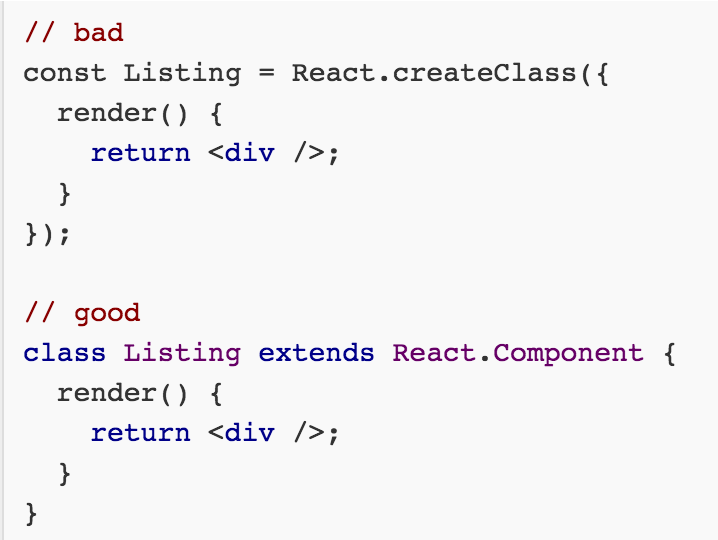
**必须**使用PascalCase作为文件名。

**必须**只包含一个React Component在一个JS文件中。

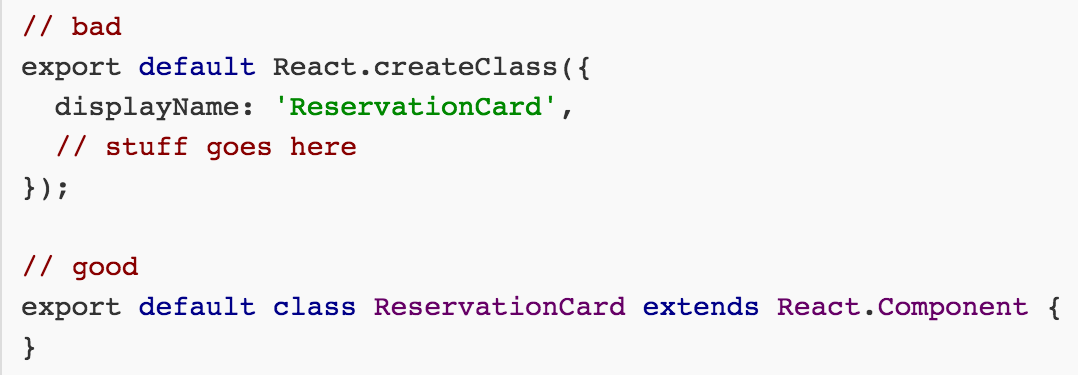
**必须**令文件名与所包含的React Component名字相同。

例如：ReservationCard.js 组件的引用名应该是 ReservationCard。然而，对于一个目录的根组件，应该使用 index.js 作为文件名，使用目录名作为组件名。

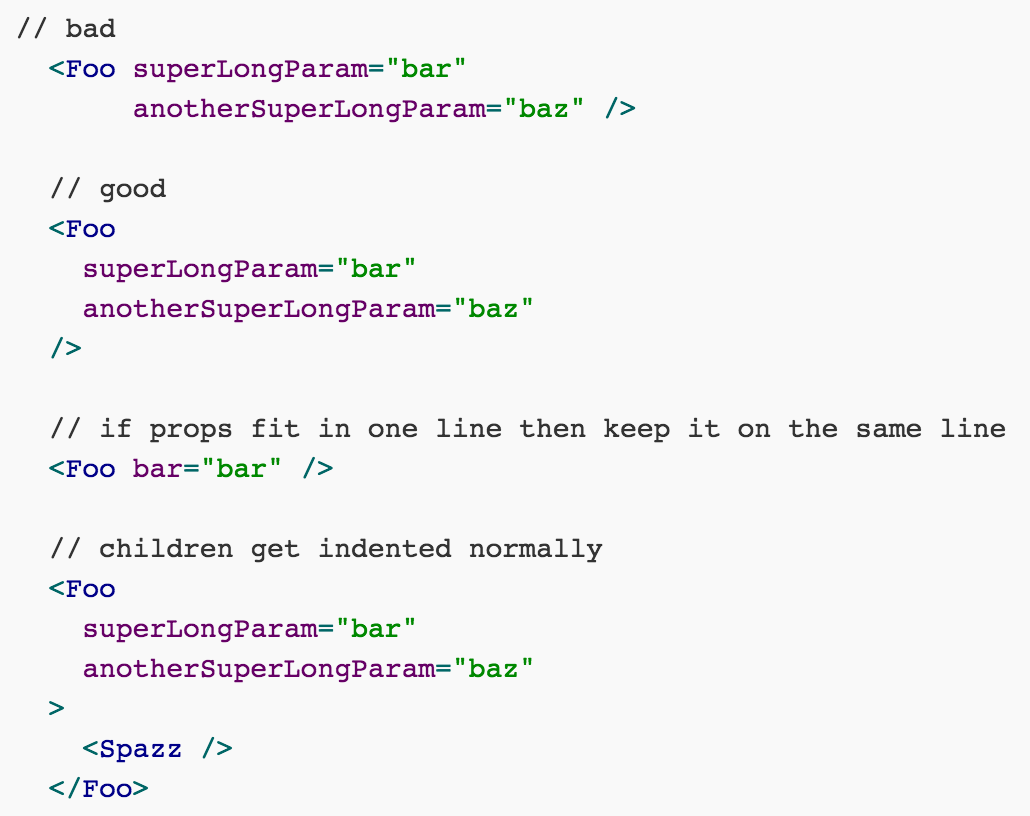
除非有更好的理由使用混淆(mixins)，否则就使用组件类继承 React.Component。



不要通过 displayName 来命名组件，通过引用来命名组件

****

对于 JSX 语法，遵循下面的对齐风格。

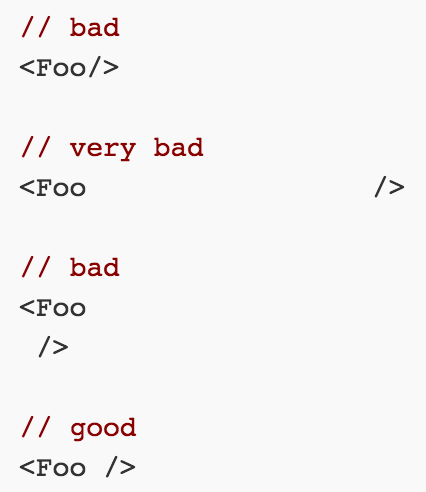
****

对于 JSX 使用双引号，对其它所有 JS 属性使用单引号

为什么？因为 JSX 属性[不能包含被转移的引号](http://eslint.org/docs/rules/jsx-quotes" \t "_blank)，并且双引号使得如 "don't" 一样的连接词很容易被输入。常规的 HTML 属性也应该使用双引号而不是单引号，JSX 属性反映了这个约定。



在自闭和标签之前留一个空格



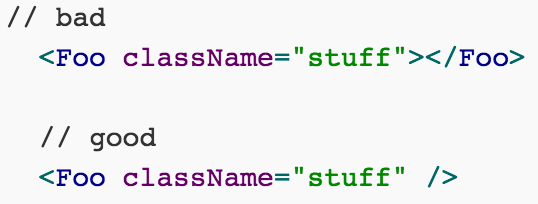
属性名采用驼峰式命名法



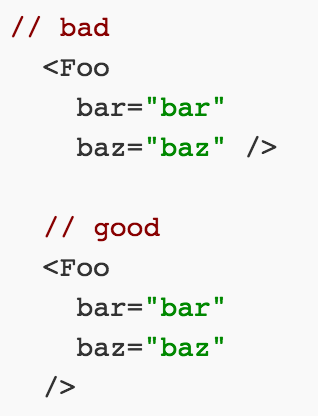
当组件跨行时，要用括号包裹 JSX 标签。



没有子组件的父组件使用自闭和标签。



如果组件有多行属性，闭合标签应写在新的一行上。



继承 React.Component 的类的方法遵循下面的顺序

1. constructor
2. optional static methods
3. getChildContext
4. componentWillMount
5. componentDidMount
6. componentWillReceiveProps
7. shouldComponentUpdate
8. componentWillUpdate
9. componentDidUpdate
10. componentWillUnmount
11. clickHandlers or eventHandlers like onClickSubmit() or onChangeDescription()
12. getter methods for render like getSelectReason() or getFooterContent()
13. Optional render methods like renderNavigation() or renderProfilePicture()
14. render

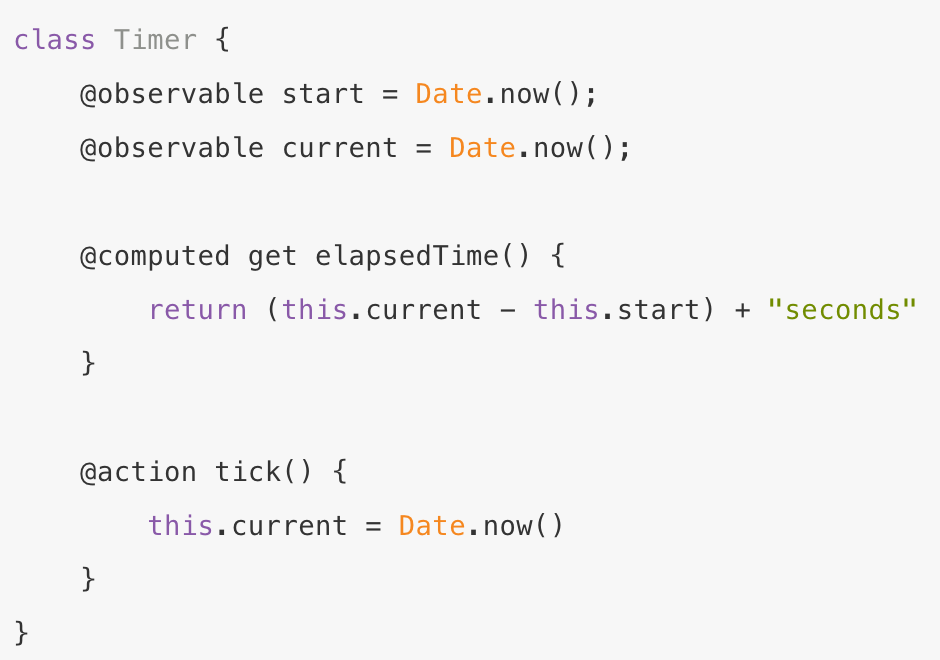
怎么定义 propTypes，defaultProps，contextTypes 等等



### MobX

必须开启严格模式，需要强制使用 action。

使用装饰器创建 observable 属性



# 附录AAA

工具：

webpack [https://doc.webpack-china.org](https://doc.webpack-china.org/)

npm https://www.npmjs.com.cn/

框架：

React组件生命周期过程说明 <http://react-china.org/t/react/1740>

MobX <https://mobx.js.org/>

其他：

浏览器缓存知识小结及应用 <http://web.jobbole.com/84888/>

前端工程与性能优化 <https://github.com/fouber/blog/issues/3>