# 前端规范

目录

[前端规范 1](#_Toc523235345)

[1. 基本原则 5](#_Toc523235346)

[1.1 结构、样式、行为分离 5](#_Toc523235347)

[1.2 缩进 5](#_Toc523235348)

[1.3 文件编码 5](#_Toc523235349)

[1.4 一律使用小写字母 5](#_Toc523235350)

[1.5 省略外链资源 URL 协议部分 6](#_Toc523235351)

[1.6 统一注释 6](#_Toc523235352)

[2. 文件命名规范 9](#_Toc523235353)

[2.1 文件夹命名 9](#_Toc523235354)

[2.2 **html**文件命名 9](#_Toc523235355)

[2.3 **js**文件命名 9](#_Toc523235356)

[2.4 **css**文件命名 10](#_Toc523235357)

[2.5 图片文件命名 10](#_Toc523235358)

[2.6 vue文件命名 10](#_Toc523235359)

[3. HTML规范 11](#_Toc523235360)

[3.1 文档类型 11](#_Toc523235361)

[3.2 字符编码 11](#_Toc523235362)

[3.3 IE兼容模式 11](#_Toc523235363)

[3.4 **SEO** 优化 12](#_Toc523235364)

[3.5 **viewport** 12](#_Toc523235365)

[3.6 引入 **CSS** 和 **JavaScript** 文件 12](#_Toc523235366)

[3.7 **favicon** 13](#_Toc523235367)

[3.8 标签 13](#_Toc523235368)

[3.9 **Class** 与 **ID** 14](#_Toc523235369)

[3.10 属性顺序 14](#_Toc523235370)

[3.11 引号 15](#_Toc523235371)

[3.12 嵌套 15](#_Toc523235372)

[3.13 布尔值属性 15](#_Toc523235373)

[4. CSS规范 16](#_Toc523235374)

[4.1 **Class** 和 **ID** 16](#_Toc523235375)

[4.2 声明块格式 17](#_Toc523235376)

[4.3 声明顺序 18](#_Toc523235377)

[4.4 引号使用 19](#_Toc523235378)

[4.5 媒体查询**（Media query）**的位置 20](#_Toc523235379)

[4.6 不要使用**@import** 20](#_Toc523235380)

[4.7 链接的样式顺序： 21](#_Toc523235381)

[4.8 无需添加浏览器厂商前缀 21](#_Toc523235382)

[4.9 单行规则声明 21](#_Toc523235383)

[4.10 简写形式的属性声明 22](#_Toc523235384)

[4.11 慎重选择高消耗的样式 23](#_Toc523235385)

[4.12 避免过分重排 23](#_Toc523235386)

[4.13 正确使用 **Display** 的属性 25](#_Toc523235387)

[4.14 不滥用 **Float** 25](#_Toc523235388)

[4.15 动画性能优化 25](#_Toc523235389)

[4.16 多利用硬件能力，如通过 **3D** 变形开启 **GPU** 加速 26](#_Toc523235390)

[4.17 提升 **CSS** 选择器性能 27](#_Toc523235391)

[4.18 **Less** 和 **Sass** 中的嵌套 29](#_Toc523235392)

[4.19 **Less** 和 **Sass** 中的操作符 29](#_Toc523235393)

[4.20 注释 29](#_Toc523235394)

[5. JavaScript规范 30](#_Toc523235395)

[5.1 注释 30](#_Toc523235396)

[5.2 命名 32](#_Toc523235397)

[5.3 命名语法 33](#_Toc523235398)

[5.4 **True** 和 **False** 布尔表达式 34](#_Toc523235399)

[5.5 不要在 Array 上使用 for-in 循环 35](#_Toc523235400)

[5.6 二元和三元操作符 36](#_Toc523235401)

[5.7 条件(三元)操作符 (?:) 36](#_Toc523235402)

[5.8 **&&** 和 **||** 36](#_Toc523235403)

[5.9 避免不必要的 **DOM** 操作 37](#_Toc523235404)

[5.10 缓存数组长度 38](#_Toc523235405)

[5.11 **DOM**选择器 38](#_Toc523235406)

[5.12 异步加载第三方内容 39](#_Toc523235407)

[5.13 **click**的**300ms**延迟响应 39](#_Toc523235408)

[5.14 移动事件 40](#_Toc523235409)

## 1. 基本原则

### 1.1 结构、样式、行为分离

尽量确保文档和模板只包含 HTML 结构，样式都放到样式表里，行为都放到脚本里。

### 1.2 缩进

统一四个空格缩进（总之缩进统一即可），不要使用 Tab 或者 Tab、空格混搭。

### 1.3 文件编码

在 HTML中指定编码 <meta charset="utf-8"> ；

无需使用 @charset 指定样式表的编码，它默认为 UTF-8 ；

### 1.4 一律使用小写字母

<!-- Recommended -->

<**img** src="google.png" alt="Google">

<!-- Not recommended -->

<**A** HREF="/">Home</**A**>

/\* Recommended \*/

color: #e5e5e5;

/\* Not recommended \*/

color: #E5E5E5;

### 1.5 省略外链资源 URL 协议部分

省略外链资源（图片及其它媒体资源）URL 中的 http / https 协议，使 URL 成为相对地址，避免[Mixed Content](https://developer.mozilla.org/en-US/docs/Security/MixedContent" \t "_blank) 问题，减小文件字节数。

其它协议（ftp 等）的 URL 不省略。

<!-- Recommended -->

<**script** src="//www.w3cschool.cn/statics/js/autotrack.js"></**script**>

<!-- Not recommended -->

<**script** src="http://www.w3cschool.cn/statics/js/autotrack.js"></**script**>

/\* Recommended \*/

.example {

**background**: url(//www.google.com/images/example);

}

/\* Not recommended \*/

.example {

**background**: url(http://www.google.com/images/example);

}

### 1.6 统一注释

通过配置编辑器，可以提供快捷键来输出一致认可的注释模式。

#### 1.6.1 HTML 注释

模块注释

<!-- 文章列表列表模块 -->

<**div** class="article-list">

...

</**div**>

区块注释

<!--

@name: Drop Down Menu

@description: Style of top bar drop down menu.

@author: Ashu(Aaaaaashu@gmail.com)

-->

#### 1.6.2 CSS 注释

组件块和子组件块以及声明块之间使用一空行分隔，子组件块之间三空行分隔；

/\* =============================================================

组件块

============================================================== \*/

/\* 子组件块

============================================================= \*/

.selector {

**padding**: 15px;

**margin-bottom**: 15px;

}

/\* 子组件块

============================================================= \*/

.selector-secondary {

**display**: block; /\* 注释\*/

}

.selector-three {

**display**: span;

}

#### 1.6.3 JavaScript 注释

单行注释

必须独占一行。// 后跟一个空格，缩进与下一行被注释说明的代码一致。

多行注释

避免使用 /\*...\*/ 这样的多行注释。有多行注释内容时，使用多个单行注释。

函数/方法注释

函数/方法注释必须包含函数说明，有参数和返回值时必须使用注释标识。；

参数和返回值注释必须包含类型信息和说明；

当函数是内部函数，外部不可访问时，可以使用 @inner 标识；

/\*\*

\* 函数描述

\*

\* **@param** {string} p1 参数1的说明

\* **@param** {string} p2 参数2的说明，比较长

\* 那就换行了.

\* **@param** {number=} p3 参数3的说明（可选）

\* **@return** {Object} 返回值描述

\*/

**function** **foo**(p1, p2, p3) {

**var** p3 = p3 || 10;

**return** {

p1: p1,

p2: p2,

p3: p3

};

}

文件注释

文件注释用于告诉不熟悉这段代码的读者这个文件中包含哪些东西。 应该提供文件的大体内容, 它的作者, 依赖关系和兼容性信息。如下:

/\*\*

\* @fileoverview Description of file, its uses **and** information

\* about its dependencies.

\* @author user@meizu.com (Firstname Lastname)

\* Copyright 2015 Meizu Inc. All Rights Reserved.

\*/

## 2. 文件命名规范

文件名称统一用小写的英文字母、数字和下划线的组合,其中不得包含汉字、空格和特殊字符;命名原则的指导思想一是使得你自己和工作组的每一个成员能够方便的理解每一个文件的意义,二是当我们在文件夹中使用“按名称排例”的命令时,同一种大类的文件能够排列在一起,以便我们查找、修改、替换、计算负载量等等操作。

### 2.1 文件夹命名

统一用小写的英文字母、下划线的组合。

有复数结构时，要采用复数命名法。

例：scripts, styles, images, data\_models

### 2.2 **html**文件命名

统一用小写的英文字母。

首先先应该以栏目名的英语翻译取单一单词为名称。

例如: 关于我们 / aboutus 信息反馈 / feedback 产 品 / product 。

如果栏目名称多而复杂并不好以英文单词命名,则统一使用该栏目名称拼音或拼音的首字母表示。

### 2.3 **js**文件命名

统一用小写的英文字母、下划线的组合。

共用public.js, 其他依实际模块需求命名。

### 2.4 **css**文件命名

统一用小写的英文字母、下划线的组合。

共用public.css, 其他依实际模块需求命名。

### 2.5 图片文件命名

统一用小写的英文字母、下划线的组合。

图片的名称分为头尾两部分，用下划线隔开，头部分表示此图片的大类性质。

例如：广告、标志、菜单、按钮等等。

放置在页面顶部的广告、装饰图案等长方形的图片取名： banner

标志性的图片取名为： logo

在页面上位置不固定并且带有链接的小图片我们取名为 button

在页面上某一个位置连续出现，性质相同的链接栏目的图片我们取名： menu

装饰用的照片我们取名： pic

不带链接表示标题的图片我们取名： title

范例：banner\_sohu.gif banner\_sina.gif menu\_aboutus.gif menu\_job.gif title\_news.gif logo\_police.gif logo\_national.gif pic\_people.jpg

鼠标感应效果图片命名规范为”图片名+\_+on/off”。

例如：menu1\_on.gif menu1\_off.gif

### 2.6 vue文件命名

使用 Pascal 命名法。

例如：AboutUs.vue FeedBack.vue。

## 3. HTML规范

### 3.1 文档类型

为每个 HTML 页面的第一行添加标准模式（standard mode）的声明， 这样能够确保在每个浏览器中拥有一致的表现。

<!DOCTYPE html>

<html>

<head>

</head>

</html>

### 3.2 字符编码

通过明确声明字符编码，能够确保浏览器快速并容易的判断页面内容的渲染方式。这样做的好处是，可以避免在 HTML 中使用字符实体标记（character entity），从而全部与文档编码一致（一般采用 UTF-8 编码）。

<html>

<**head**>

<**meta** charset="utf-8">

......

</**head**>

<**body**>

......

</**body**>

</**html**>

### 3.3 IE兼容模式

优先使用最新版本的IE 和 Chrome 内核

<**meta** http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

### 3.4 **SEO** 优化

<**head**>

<**meta** charset="utf-8">

<**meta** http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<!-- SEO -->

<**title**>Style Guide</**title**>

<**meta** name="keywords" content="your keywords">

<**meta** name="description" content="your description">

<**meta** name="author" content="author,email address">

</**head**>

### 3.5 **viewport**

viewport: 一般指的是浏览器窗口内容区的大小，不包含工具条、选项卡等内容；

width: 浏览器宽度，输出设备中的页面可见区域宽度；

device-width: 设备分辨率宽度，输出设备的屏幕可见宽度；

initial-scale: 初始缩放比例；

maximum-scale: 最大缩放比例；

为移动端设备优化，设置可见区域的宽度和初始缩放比例。

<**meta** name="viewport" content="width=device-width, initial-scale=1.0">

### 3.6 引入 **CSS** 和 **JavaScript** 文件

根据 HTML5 规范，在引入 CSS 和 JavaScript 文件时一般不需要指定 type 属性，因为 text/css 和 text/javascript 分别是它们的默认值。

<!-- External CSS -->

<link rel="stylesheet" href="code-guide.css">

<!-- In-document CSS -->

<style>

/\* ... \*/

</style>

<!-- JavaScript -->

<script src="code-guide.js"></script>

### 3.7 **favicon**

在未指定 favicon 时，大多数浏览器会请求 Web Server 根目录下的 favicon.ico 。为了保证 favicon 可访问，避免404，必须遵循以下两种方法之一：

（1）在 Web Server 根目录放置 favicon.ico 文件；

（2）使用 link 指定 favicon；

<**link** rel="shortcut icon" href="path/to/favicon.ico">

### 3.8 标签

自闭合（self-closing）标签，无需闭合 ( 例如： img input br hr 等 )；

可选的闭合标签（closing tag），需闭合 ( 例如：</li> 或 </body> )；

尽量减少标签数量；

<**img** src=".image/logo" alt="Google">

<**input** type="text" name="title">

<**ul**>

<**li**>Style</**li**>

<**li**>Guide</**li**>

</**ul**>

<!-- Not recommended -->

<**span** class="avatar">

<**img** src="...">

</**span**>

<!-- Recommended -->

<**img** class="avatar" src="...">

### 3.9 **Class** 与 **ID**

class 应以功能或内容命名，不以表现形式命名；

class 与 id 单词字母小写，多个单词组成时，采用下划线\_分隔

使用唯一的 id 作为 Javascript hook, 同时避免创建无样式信息的 class；

<!-- Not recommended -->

<**div** class="j\_hook left contentWrapper"></**div**>

<!-- Recommended -->

<**div** id="j\_hook" class="sidebar content\_wrapper"></**div**>

### 3.10 属性顺序

HTML 属性应该按照特定的顺序出现以保证易读性。

（1）Id

（2）class

（3）name

（4）data-xxx

（5）src, for, type, href

（6）title, alt

（7）aria-xxx, role

<a id="..." **class**="..." data-modal="toggle" href="###"></**a**>

<input **class**="form\_control" type="text">

<**img** src="..." alt="...">

### 3.11 引号

属性的定义，统一使用双引号。

<!-- Not recommended -->

<**span** id='j\_hook' class=text>Google</**span**>

<!-- Recommended -->

<**span** id="j\_hook" class="text">Google</**span**>

### 3.12 嵌套

a 不允许嵌套 div这种约束属于语义嵌套约束，与之区别的约束还有严格嵌套约束，比如a 不允许嵌套 a。

严格嵌套约束在所有的浏览器下都不被允许；而语义嵌套约束，浏览器大多会容错处理，生成的文档树可能相互不太一样。

**语义嵌套约束**

<li> 用于 <ul> 或 <ol> 下；

<dd>, <dt> 用于 <dl> 下；

<thead>, <tbody>, <tfoot>, <tr>, <td> 用于 <table> 下；

**严格嵌套约束**

inline-Level 元素，仅可以包含文本或其它 inline-Level 元素;

<a>里不可以嵌套交互式元素<a>、<button>、<select>等;

<p>里不可以嵌套块级元素<div>、<h1>~<h6>、<p>、<ul>/<ol>/<li>、<dl>/<dt>/<dd>、<form>等。

### 3.13 布尔值属性

HTML5 规范中 disabled、checked、selected 等属性不用设置值。

布尔型属性可以在声明时不赋值。XHTML 规范要求为其赋值，但是 HTML5 规范不需要。

*元素的布尔型属性如果有值，就是 true，如果没有值，就是 false。*

如果一定要为其赋值的话，请参考 WhatWG 规范：

*如果属性存在，其值必须是空字符串或 [...] 属性的规范名称，并且不要在首尾添加空白符。*

**简单来说，就是不用赋值。**

<**input** type="text" disabled>

<**input** type="checkbox" value="1" checked>

<**select**>

<**option** value="1" selected>1</**option**>

</**select**>

## 4. CSS规范

### 4.1 **Class** 和 **ID**

使用语义化、通用的命名方式；

使用连字符 \_ 作为 ID、Class 名称界定符，不要驼峰命名法和中划线；

避免选择器嵌套层级过多，尽量少于 3 级；

避免选择器和 Class、ID 叠加使用；

出于[性能考量](http://www.stevesouders.com/blog/2009/06/18/simplifying-css-selectors/)，在没有必要的情况下避免元素选择器叠加 Class、ID 使用。

元素选择器和 ID、Class 混合使用也违反关注分离原则。如果HTML标签修改了，就要再去修改 CSS 代码，不利于后期维护。

/\* Not recommended \*/

.red {}

.box-green {}

.page .header .login #username **input** {}

**ul**#example {}

/\* Recommended \*/

#nav {}

.box\_video {}

#username **input** {}

#example {}

### 4.2 声明块格式

选择器分组时，保持独立的选择器占用一行；

声明块的左括号 { 前添加一个空格；

声明块的右括号 } 应单独成行；

声明语句中的 : 后应添加一个空格；

声明语句应以分号 ; 结尾；

一般以逗号分隔的属性值，每个逗号后应添加一个空格；

rgb()、rgba()、hsl()、hsla() 或 rect() 括号内的值，逗号分隔，但逗号后不添加一个空格；

对于属性值或颜色参数，省略小于 1 的小数前面的 0 （例如，.5 代替 0.5；-.5px 代替-0.5px）；

十六进制值应该全部小写和尽量简写，例如，#fff 代替 #ffffff；

避免为 0 值指定单位，例如，用 margin: 0; 代替 margin: 0px;；

/\* Not recommended \*/

.selector, .selector\_secondary, .selector[type=text] {

**padding**:15px;

**margin**:0px 0px 15px;

**background-color**:rgba(0, 0, 0, 0.5);

**box-shadow**:0px 1px 2px #CCC,inset 0 1px 0 #FFFFFF

}

/\* Recommended \*/

.selector,

.selector\_secondary,

.selector[type="text"] {

**padding**: 15px;

**margin-bottom**: 15px;

**background-color**: rgba(0,0,0,.5);

**box-shadow**: 0 1px 2px #ccc, inset 0 1px 0 #fff;

}

### 4.3 声明顺序

相关属性应为一组，推荐的样式编写顺序

（1）Positioning

（2）Box model

（3）Typographic

（4）Visual

由于定位（positioning）可以从正常的文档流中移除元素，并且还能覆盖盒模型（box model）相关的样式，因此排在首位。盒模型决定了组件的尺寸和位置，因此排在第二位。

其他属性只是影响组件的内部（inside）或者是不影响前两组属性，因此排在后面。

.declaration\_order {

/\* Positioning \*/

**position**: absolute;

**top**: 0;

**right**: 0;

**bottom**: 0;

**left**: 0;

**z-index**: 100;

/\* Box model \*/

**display**: block;

**box-sizing**: border-box;

**width**: 100px;

**height**: 100px;

**padding**: 10px;

**border**: 1px solid #e5e5e5;

**border-radius**: 3px;

**margin**: 10px;

**float**: right;

**overflow**: hidden;

/\* Typographic \*/

**font**: normal 13px "Helvetica Neue", sans-serif;

**line-height**: 1.5;

**text-align**: center;

/\* Visual \*/

**background-color**: #f5f5f5;

**color**: #fff;

**opacity**: .8;

/\* Other \*/

**cursor**: pointer;

}

### 4.4 引号使用

url() 、属性选择符、属性值使用双引号。 参考 [Is quoting the value of url() really necessary?](http://stackoverflow.com/questions/2168855/is-quoting-the-value-of-url-really-necessary)

/\* Not recommended \*/

@**import** url(//www.google.com/css/maia.css);

**html** {

**font-family**: 'open sans', arial, sans-serif;

}

/\* Recommended \*/

@**import** url("//www.google.com/css/maia.css");

**html** {

**font-family**: "open sans", arial, sans-serif;

}

.selector[type="text"] {

}

### 4.5 媒体查询**（Media query）**的位置

将媒体查询放在尽可能相关规则的附近。不要将他们打包放在一个单一样式文件中或者放在文档底部。如果你把他们分开了，将来只会被大家遗忘。

.element { ... }

.element-avatar { ... }

.element-selected { ... }

@media (max-width: 768px) {

.element { ...}

.element-avatar { ... }

.element-selected { ... }

}

### 4.6 不要使用**@import**

与 <link> 相比，@import 要慢很多，不光增加额外的请求数，还会导致不可预料的问题。

替代办法：

使用多个 元素；

通过 Sass 或 Less 类似的 CSS 预处理器将多个 CSS 文件编译为一个文件；

其他 CSS 文件合并工具；

### 4.7 链接的样式顺序：

a:link -> a:visited -> a:hover -> a:active（LoVeHAte）

### 4.8 无需添加浏览器厂商前缀

使用 [Autoprefixer](https://github.com/postcss/autoprefixer" \t "_blank) 自动添加浏览器厂商前缀，编写 CSS 时不需要添加浏览器前缀，直接使用标准的 CSS 编写。

Autoprefixer 通过 [Can I use](http://caniuse.com/)，按兼容的要求，对相应的 CSS 代码添加浏览器厂商前缀。

### 4.9 单行规则声明

对于只包含一条声明的样式，为了易读性和便于快速编辑，建议将语句放在同一行。对于带有多条声明的样式，还是应当将声明分为多行。

这样做的关键因素是为了错误检测 -- 例如，CSS 校验器指出在 183 行有语法错误。如果是单行单条声明，你就不会忽略这个错误；如果是单行多条声明的话，你就要仔细分析避免漏掉错误了。

/\* Single declarations on one line \*/

.span1 { width: 60px; }

.span2 { width: 140px; }

.span3 { width: 220px; }

/\* Multiple declarations, one per line \*/

.sprite {

display: inline-block;

width: 16px;

height: 15px;

background-image: url(../img/sprite.png);

}

.icon { background-position: 0 0; }

.icon-home { background-position: 0 -20px; }

.icon-account { background-position: 0 -40px; }

### 4.10 简写形式的属性声明

在需要显示地设置所有值的情况下，应当尽量限制使用简写形式的属性声明。常见的滥用简写属性声明的情况如下：

（1）padding

（2）margin

（3）font

（4）background

（5）border

（6）border-radius

大部分情况下，我们不需要为简写形式的属性声明指定所有值。例如，HTML 的 heading 元素只需要设置上、下边距（margin）的值，因此，在必要的时候，只需覆盖这两个值就可以。过度使用简写形式的属性声明会导致代码混乱，并且会对属性值带来不必要的覆盖从而引起意外的副作用。

/\* Bad example \*/

.element {

margin: 0 0 10px;

background: red;

background: url("image.jpg");

border-radius: 3px 3px 0 0;

}

/\* Good example \*/

.element {

margin-bottom: 10px;

background-color: red;

background-image: url("image.jpg");

border-top-left-radius: 3px;

border-top-right-radius: 3px;

}

### 4.11 慎重选择高消耗的样式

高消耗属性在绘制前需要浏览器进行大量计算：

（1）box-shadows

（2）border-radius

（3）transparency

（4）transforms

（5）CSS filters（性能杀手）

### 4.12 避免过分重排

当发生重排的时候，浏览器需要重新计算布局位置与大小。

常见的重排元素:

（1）width

（2）height

（3）padding

（4）margin

（5）display

（6）border-width

（7）position

（8）top

（9）left

（10）right

（11）bottom

（12）font-size

（13）float

（14）text-align

（15）overflow-y

（16）font-weight

（17）overflow

（18）font-family

（19）line-height

（20）vertical-align

（21）clear

（22）white-space

（23）min-height

### 4.13 正确使用 **Display** 的属性

Display 属性会影响页面的渲染，请合理使用。

（1）display: inline后不应该再使用 width、height、margin、padding 以及 float；

（2）display: inline-block 后不应该再使用 float；

（3）display: block 后不应该再使用 vertical-align；

（4）display: table-\* 后不应该再使用 margin 或者 float；

### 4.14 不滥用 **Float**

Float在渲染时计算量比较大，尽量减少使用。

### 4.15 动画性能优化

动画的实现原理，是利用了人眼的“视觉暂留”现象，在短时间内连续播放数幅静止的画面，使肉眼因视觉残象产生错觉，而误以为画面在“动”。

动画的基本概念：

（1）帧：在动画过程中，每一幅静止画面即为一“帧”;

（2）帧率：即每秒钟播放的静止画面的数量，单位是fps(Frame per second);

（3）帧时长：即每一幅静止画面的停留时间，单位一般是ms(毫秒);

（4）跳帧(掉帧/丢帧)：在帧率固定的动画中，某一帧的时长远高于平均帧时长，导致其后续数帧被挤压而丢失的现象。

一般浏览器的渲染刷新频率是 60 fps，所以在网页当中，帧率如果达到 50-60 fps 的动画将会相当流畅，让人感到舒适。

（1）如果使用基于 javaScript 的动画，尽量使用 requestAnimationFrame. 避免使用 setTimeout, setInterval.

（2）避免通过类似 jQuery animate()-style 改变每帧的样式，使用 CSS 声明动画会得到更好的浏览器优化。

（3）使用 translate 取代 absolute 定位就会得到更好的 fps，动画会更顺滑。

### 4.16 多利用硬件能力，如通过 **3D** 变形开启 **GPU** 加速

一般在 Chrome 中，3D或透视变换（perspective transform）CSS属性和对 opacity 进行 CSS 动画会创建新的图层，在硬件加速渲染通道的优化下，GPU 完成 3D 变形等操作后，将图层进行复合操作（Compesite Layers），从而避免触发浏览器大面积重绘和重排。

注：3D 变形会消耗更多的内存和功耗。

使用 translate3d 右移 500px 的动画流畅度要明显优于直接使用 left：

.ball\_1 {

transition: -webkit-transform .5s ease;

-webkit-transform: translate3d(0, 0, 0);

}

.ball\_1.slidein{

-webkit-transform: translate3d(500px, 0, 0);

}

.ball\_2 {

transition: left .5s ease; left：0;

}

.ball\_2.slidein {

left：500px;

}

### 4.17 提升 **CSS** 选择器性能

**CSS** 选择器对性能的影响源于浏览器匹配选择器和文档元素时所消耗的时间，所以优化选择器的原则是应尽量避免使用消耗更多匹配时间的选择器。而在这之前我们需要了解 **CSS** 选择器匹配的机制， 如子选择器规则：

#header > **a** {**font-weight**:blod;}

我们中的大多数人都是从左到右的阅读习惯，会习惯性的设定浏览器也是从左到右的方式进行匹配规则，推测这条规则的开销并不高。

我们会假设浏览器以这样的方式工作：寻找 **id** 为 **header** 的元素，然后将样式规则应用到直系子元素中的 **a** 元素上。我们知道文档中只有一个 **id** 为 **header** 的元素，并且它只有几个 **a** 元素的子节点，所以这个 **CSS** 选择器应该相当高效。

事实上，却恰恰相反，**CSS** 选择器是从右到左进行规则匹配。了解这个机制后，例子中看似高效的选择器在实际中的匹配开销是很高的，浏览器必须遍历页面中所有的 **a** 元素并且确定其父元素的 **id** 是否为 **header** 。

如果把例子的子选择器改为后代选择器则会开销更多，在遍历页面中所有 **a** 元素后还需向其上级遍历直到根节点。

#header **a** {**font-weight**:blod;}

理解了**CSS**选择器从右到左匹配的机制后，明白只要当前选择符的左边还有其他选择符，样式系统就会继续向左移动，直到找到和规则匹配的选择符，或者因为不匹配而退出。我们把最右边选择符称之为**关键选择器**。

1、避免使用通用选择器

/\* Not recommended \*/

.content \* {**color**: red;}

浏览器匹配文档中所有的元素后分别向上逐级匹配 **class** 为 **content** 的元素，直到文档的根节点。因此其匹配开销是非常大的，所以应避免使用关键选择器是通配选择器的情况。

2、避免使用标签或 **class** 选择器限制 **id** 选择器

/\* Not recommended \*/

button#backButton {…}

/\* Recommended \*/

#newMenuIcon {…}

3、避免使用标签限制 **class** 选择器

/\* Not recommended \*/

treecell.indented {…}

/\* Recommended \*/

.treecell\_indented {…}

/\* Much to recommended \*/

.hierarchy\_deep {…}

4、避免使用多层标签选择器。使用 **class** 选择器替换，减少**css**查找

/\* Not recommended \*/

treeitem[mailfolder="true"] > treerow > treecell {…}

/\* Recommended \*/

.treecell\_mailfolder {…}

5、避免使用子选择器

/\* Not recommended \*/

treehead treerow treecell {…}

/\* Recommended \*/

treehead > treerow > treecell {…}

/\* Much to recommended \*/

.treecell\_header {…}

6、使用继承

/\* Not recommended \*/

#bookmarkMenuItem > .menu-left { **list-style-image**: url(blah) }

/\* Recommended \*/

#bookmarkMenuItem { **list-style-image**: url(blah) }

### 4.18 **Less** 和 **Sass** 中的嵌套

避免不必要的嵌套。这是因为虽然你可以使用嵌套，但是并不意味着应该使用嵌套。只有在必须将样式限制在父元素内（也就是后代选择器），并且存在多个需要嵌套的元素时才使用嵌套。

// Without nesting

.table > thead > tr > th { … }

.table > thead > tr > td { … }

// With nesting

.table > thead > tr {

> th { … }

> td { … }

}

### 4.19 **Less** 和 **Sass** 中的操作符

为了提高可读性，在圆括号中的数学计算表达式的数值、变量和操作符之间均添加一个空格。

// Bad example

.element {

margin: 10px 0 @variable\*2 10px;

}

// Good example

.element {

margin: 10px 0 (@variable \* 2) 10px;

}

### 4.20 注释

代码是由人编写并维护的。请确保你的代码能够自描述、注释良好并且易于他人理解。好的代码注释能够传达上下文关系和代码目的。不要简单地重申组件或 **class** 名称。

对于较长的注释，务必书写完整的句子；对于一般性注解，可以书写简洁的短语。

/\* Bad example \*/

/\* Modal header \*/

.modal\_header {

...

}

/\* Good example \*/

/\* Wrapping element for .modal-title and .modal-close \*/

.modal\_header {

...

}

## 5. JavaScript规范

### 5.1 注释

#### 5.1.1 原则

（1）As short as possible（如无必要，勿增注释）：尽量提高代码本身的清晰性、可读性。

（2）As long as necessary（如有必要，尽量详尽）：合理的注释、空行排版等，可以让代码更易阅读、更具美感。

#### **5.1.2 单行注释**

必须独占一行。// 后跟一个空格，缩进与下一行被注释说明的代码一致。

#### **5.1.3 多行注释**

避免使用 /\*...\*/ 这样的多行注释。有多行注释内容时，使用多个单行注释。

#### 5.1.4 函数/方法注释

（1）函数/方法注释必须包含函数说明，有参数和返回值时必须使用注释标识。；

（2）参数和返回值注释必须包含类型信息和说明；

（3）当函数是内部函数，外部不可访问时，可以使用 @inner 标识；

/\*\*

\* 函数描述

\*

\* **@param** {string} p1 参数1的说明

\* **@param** {string} p2 参数2的说明，比较长

\* 那就换行了.

\* **@param** {number=} p3 参数3的说明（可选）

\* **@return** {Object} 返回值描述

\*/

**function** **foo**(p1, p2, p3) {

**var** p3 = p3 || 10;

**return** {

p1: p1,

p2: p2,

p3: p3

};

}

#### 5.1.5 文件注释

文件注释用于告诉不熟悉这段代码的读者这个文件中包含哪些东西。 应该提供文件的大体内容, 它的作者, 依赖关系和兼容性信息。如下:

/\*\*

\* @fileoverview Description of file, its uses **and** information

\* about its dependencies.

\* @author user@meizu.com (Firstname Lastname)

\* Copyright 2009 Meizu Inc. All Rights Reserved.

\*/

### 5.2 命名

**变量**, 使用 **Camel** 命名法。

**var** loadingModules = {};

**私有属性**、**变量**和**方法**以下划线 \_ 开头。

**var** \_privateMethod = {};

**常量**, 使用全部字母大写，单词间下划线分隔的命名方式。

**var** HTML\_ENTITY = {};

（1）**函数**, 使用 **Camel** 命名法。

（2）函数的**参数**, 使用 **Camel** 命名法。

**function** **stringFormat**(source) {}

**function** **hear**(theBells) {}

（1）**类**, 使用 **Pascal** 命名法

（2）类的 **方法 / 属性**, 使用 **Camel** 命名法

**function** **TextNode**(value, engine) {

**this**.value = value;

**this**.engine = engine;

}

TextNode.prototype.clone = **function** () {

**return** **this**;

};

（1）**枚举变量** 使用 **Pascal** 命名法。

（2）**枚举的属性**， 使用全部字母大写，单词间下划线分隔的命名方式。

**var** TargetState = {

READING: 1,

READED: 2,

APPLIED: 3,

READY: 4

};

由多个单词组成的 **缩写词**，在命名中，根据当前命名法和出现的位置，所有字母的大小写与首字母的大小写保持一致。

**function** **XMLParser**() {}

**function** **insertHTML**(element, html) {}

**var** httpRequest = **new** HTTPRequest();

### 5.3 命名语法

**类名**，使用名词。

**function** **Engine**(options) {}

**函数名**，使用动宾短语。

**function** **getStyle**(element) {}

**boolean** 类型的变量使用 **is** 或 **has** 开头。

**var** isReady = false;

**var** hasMoreCommands = false;

**Promise 对象**用动宾短语的进行时表达。

**var** loadingData = ajax.**get**('url');

loadingData.then(callback);

### 5.4 **True** 和 **False** 布尔表达式

类型检测优先使用 **typeof**。对象类型检测使用 **instanceof**。**null** 或 **undefined** 的检测使用 **== null**。

下面的布尔表达式都返回 **false**:

（1）**null**

（2）**undefined**

（3）**''** 空字符串

（4）**0** 数字**0**

但小心下面的, 可都返回 **true**:

（1）**'0'** 字符串**0**

（2）**[]** 空数组

（3）**{}** 空对象

### 5.5 不要在 Array 上使用 for-in 循环

**for-in** 循环只用于 object/map/hash 的遍历, 对 Array 用 **for-in** 循环有时会出错. 因为它并不是从 **0** 到 **length - 1** 进行遍历, 而是所有出现在对象及其原型链的键值。

// Not recommended

**function** **printArray**(arr) {

**for** (**var** key **in** arr) {

print(arr[key]);

}

}

printArray([0,1,2,3]); // This works.

**var** a = **new** Array(10);

printArray(a); // This is wrong.

a = document.getElementsByTagName('\*');

printArray(a); // This is wrong.

a = [0,1,2,3];

a.buhu = 'wine';

printArray(a); // This is wrong again.

a = **new** Array;

a[3] = 3;

printArray(a); // This is wrong again.

// Recommended

**function** **printArray**(arr) {

**var** l = arr.length;

**for** (**var** i = 0; i < l; i++) {

print(arr[i]);

}

}

### 5.6 二元和三元操作符

操作符始终写在前一行, 以免**分号的隐式插入**产生预想不到的问题。

**var** x = a ? b : c;

**var** y = a ?

longButSimpleOperandB : longButSimpleOperandC;

**var** z = a ?

moreComplicatedB :

moreComplicatedC;

. 操作符也是如此：

var x = foo.bar().

**do**Something().

**do**SomethingElse();

### 5.7 条件(三元)操作符 (?:)

三元操作符用于替代 **if** 条件判断语句。

// Not recommended

**if** (val != 0) {

**return** foo();

} **else** {

**return** bar();

}

// Recommended

**return** val ? foo() : bar();

### 5.8 **&&** 和 **||**

二元布尔操作符是可短路的, 只有在必要时才会计算到最后一项。

// Not recommended

**function** **foo**(opt\_win) {

**var** win;

**if** (opt\_win) {

win = opt\_win;

} **else** {

win = window;

}

// ...

}

**if** (node) {

**if** (node.kids) {

**if** (node.kids[index]) {

foo(node.kids[index]);

}

}

}

// Recommended

**function** **foo**(opt\_win) {

**var** win = opt\_win || window;

// ...

}

**var** kid = node && node.kids && node.kids[index];

**if** (kid) {

foo(kid);

}

### 5.9 避免不必要的 **DOM** 操作

浏览器遍历 DOM 元素的代价是昂贵的。最简单优化 DOM 树查询的方案是，当一个元素出现多次时，将它保存在一个变量中，就避免多次查询 DOM 树了。

// Recommended

**var** myList = "";

**var** myListHTML = document.getElementById("myList").innerHTML;

**for** (**var** i = 0; i < 100; i++) {

myList += "<span>" + i + "</span>";

}

myListHTML = myList;

// Not recommended

**for** (**var** i = 0; i < 100; i++) {

document.getElementById("myList").innerHTML += "<span>" + i + "</span>";

}

### 5.10 缓存数组长度

循环无疑是和 **JavaScript** 性能非常相关的一部分。通过存储数组的长度，可以有效避免每次循环重新计算。

注: 虽然现代浏览器引擎会自动优化这个过程，但是不要忘记还有旧的浏览器。

**var** arr = **new** Array(1000),

len, i;

// Recommended - size is calculated only 1 time and then stored

**for** (i = 0, len = arr.length; i < len; i++) {

}

// Not recommended - size needs to be recalculated 1000 times

**for** (i = 0; i < arr.length; i++) {

}

### 5.11 **DOM**选择器

**html5**为了我们提供了一个非常好的**DOM**选择器，就是**document.querySelector**和**document.querySelectorAll**这两个方法，这两个方法在**android2.1+**以及**ios3+**以后，都可以使用，其接受的参数为**css**选择器。在实际**web**开发中，有一部大部分工作会用到**DOM**的操作，通过这个神器，可以解决大多数的**DOM**的操作。建议大家使用的时候，可以多多使用这两个方法。

其他的**DOM**的选择器的兼容性并不是太好，建议不要使用。

### 5.12 异步加载第三方内容

当你无法保证嵌入第三方内容比如 **Youtube** 视频或者一个 **like/tweet** 按钮可以正常工作的时候，你需要考虑用异步加载这些代码，避免阻塞整个页面加载。

(**function**() {

**var** script,

scripts = document.getElementsByTagName('script')[0];

**function** **load**(url) {

script = document.createElement('script');

script.async = true;

script.src = url;

scripts.parentNode.insertBefore(script, scripts);

}

load('//apis.google.com/js/plusone.js');

load('//platform.twitter.com/widgets.js');

load('//s.widgetsite.com/widget.js');

}());

### 5.13 **click**的**300ms**延迟响应

说到移动开发，不得不说一下这个**click**事件，在手机上被叫的最多的就是点击的反应慢，就是**click**惹出来的事情。情况是在这样，在手机早期，浏览器有系统级的放大和缩小的功能，用户在屏幕上点击两次之后，系统会触发站点的放大/缩小功能。不过由于系统需要判断用户在点击之后，有没有接下来的第二次点击，因此在用户点击第一次的时候，会强制等待**300ms**，等待用户在这个时间内，是否有用户第二次的提交，如果没有的话，就会**click**的事件，否则就会触发放大/缩小的效果。

这个设计本来没有问题，但是在绝大多数的手机操作中，用户的单击事件的概率大大大于双击的，因此所有用户的点击都必须要等**300ms**，才能触发**click**事件，造成给用户给反应迟钝的反应，这个难以解决。业界普遍解决的方案是自己通过**touch**的事件完成**tap**，替代**click**。不过**tap**事件来实际的应用中存在下面所说的问题。

不过有个好消息，就是手机版chrome21.0之后，对于**viewport width=device-width**，并且禁止缩放的设置，**click**点击将取消**300ms**的强制等待时间，这个会是**web**的响应时间大大提升。**ios**至今还没有此类消息。不过这个还需要有一段时间。

### 5.14 移动事件

**javascript**有很多用户交互相关事件，在移动上有一些比较特有的事件，大家在日常开发中，可能会接触到，这些事件的特性，这里说一下：

**orientationchange** 这个事件是在当设备发生旋转的时候，发生的事件。这个在某些场合会非常的实用。

**touchstart**、**touchmove**、**touchend**、**touchcancel**等四个触摸事件，在所有移动**web**的中，都支持这四个事件。通过这两个事件，可以模拟出各种用户的手势，不过由于其处理比较复杂，可能模拟最多的是**tap**事件。很多**web**移动类库，都有**tap**的事件的实现，不过从实践中，**tap**都不是处理的很好，**tap**的主要问题，有两个，一个是**tap**和滚动同时触发的时候，往往会触发**tap**事件，二是**tap**的敏感度，经常会失误触发**tap**。

**scroll**事件 这个事件在**PC**上的触发时机和手机上的触发时机不同，**scroll**事件在手机上，只有在滚动停止的时候才会发生,因此这个事件在移动端用的比较少，因为触发的时机已经晚了。对于需要在移动中，改变页面结构的功能，用**scroll**是无法完成的。