# web前端代码规范

目录

[web前端代码规范 1](#_Toc515533039)

[一、命名规则 4](#_Toc515533040)

[二、HTML 4](#_Toc515533041)

[1.1 缩进与换行 4](#_Toc515533042)

[1.2命名 4](#_Toc515533043)

[2 标签 5](#_Toc515533044)

[2.1属性顺序 5](#_Toc515533045)

[2.2 JS生成标签 5](#_Toc515533046)

[2.3 alt标签不为空 6](#_Toc515533047)

[2.4 嵌套 6](#_Toc515533048)

[2.5 图片 6](#_Toc515533049)

[3 HEAD 7](#_Toc515533050)

[3.1 HTML5 doctype 7](#_Toc515533051)

[3.2 lang属性 7](#_Toc515533052)

[3.3 字符编码 7](#_Toc515533053)

[3.4 引入CSS, JS 7](#_Toc515533054)

[3.5 SEO 优化 7](#_Toc515533055)

[3.6 viewport 7](#_Toc515533056)

[3.7 iOS 图标 8](#_Toc515533057)

[3.8 favicon 8](#_Toc515533058)

[3.9 HEAD 模板 8](#_Toc515533059)

[3.10 测试标记： 9](#_Toc515533060)

[4表单 9](#_Toc515533061)

[三、CSS 9](#_Toc515533062)

[1 编码总体原则 9](#_Toc515533063)

[1.1 空格 12](#_Toc515533064)

[1.2 命名 13](#_Toc515533065)

[1.3 颜色 14](#_Toc515533066)

[1.4 Class 和 ID 15](#_Toc515533067)

[1.5 声明顺序 15](#_Toc515533068)

[1.6 媒体查询（Media query）的位置 15](#_Toc515533069)

[1.7 不要使用 @import 16](#_Toc515533070)

[1.8 无需添加浏览器厂商前缀 16](#_Toc515533071)

[2 性能优化 16](#_Toc515533072)

[2.1 慎重选择高消耗的样式 16](#_Toc515533073)

[2.2 避免过分重排 16](#_Toc515533074)

[2.3 正确使用 Display 的属性 17](#_Toc515533075)

[2.4不滥用 Float 17](#_Toc515533076)

[2.5 动画性能优化 17](#_Toc515533077)

[2.6 多利用硬件能力，如通过 3D 变形开启 GPU 加速 18](#_Toc515533078)

[2.7 提升 CSS 选择器性能 19](#_Toc515533079)

[四 JavaScript 20](#_Toc515533080)

[1编码总体原则 20](#_Toc515533081)

[2 编码规范 21](#_Toc515533082)

[2.1 注释 21](#_Toc515533083)

[原则 21](#_Toc515533084)

[2.2 单行注释 21](#_Toc515533085)

[2.3多行注释 21](#_Toc515533086)

[2.4文件注释 22](#_Toc515533087)

[2.5命名 22](#_Toc515533088)

[2.6 命名语法 23](#_Toc515533089)

[2.7缩进 24](#_Toc515533090)

[2.8 单行长度 24](#_Toc515533091)

[2.9 分号 24](#_Toc515533092)

[2.10 空格 25](#_Toc515533093)

[2.11 空行 26](#_Toc515533094)

[2.12 换行 28](#_Toc515533095)

[2.13 单行注释 28](#_Toc515533096)

[2.14 多行注释 28](#_Toc515533097)

[2.15 文档注释 29](#_Toc515533098)

[2.16 引号 29](#_Toc515533099)

[2.17 变量声明 29](#_Toc515533100)

[2.18 函数 29](#_Toc515533101)

[2.19 数组、对象 30](#_Toc515533102)

[2.20 括号 31](#_Toc515533103)

[2.21 null 31](#_Toc515533104)

[2.22 undefined 31](#_Toc515533105)

[2.23 jshint 32](#_Toc515533106)

[2.24 调试 34](#_Toc515533107)

[2.25 避免全局命名空间污染 34](#_Toc515533108)

[2.26 严格模式 35](#_Toc515533109)

[2.27 变量声明 35](#_Toc515533110)

[3 性能优化 35](#_Toc515533111)

[1 避免不必要的 DOM 操作 35](#_Toc515533112)

[2 缓存数组长度 36](#_Toc515533113)

[3 异步加载第三方内容 36](#_Toc515533114)

[4 避免使用 jQuery 实现动画 37](#_Toc515533115)

[5 借力 CSS 精灵 37](#_Toc515533116)

[6 图片格式 37](#_Toc515533117)

[7 缓存 38](#_Toc515533118)

[8 把资源文件跨域分片 38](#_Toc515533119)

[9 避免用IFRAME 39](#_Toc515533120)

## 一、命名规则

项目命名：全部采用小写方式，以下划线分隔

例：my\_project\_name

目录命名：参照项目命名；有复数结构时，要采用复数命名法。

例：scripts, styles, images, data\_models

JS文件命名：参照项目命名。

例：account\_model.js

Css,scss文件命名：参照项目命名。

例：retina\_sprites.scss

html文件命名：参照项目命名。

例：error\_report.html

二、HTML  
1 代码风格

#### 1.1 缩进与换行

[强制]使用4个空格作为缩进，不允许使用2个空格 或 tab 字符。

#### 1.2命名

**[强制]** class 必须单词全字母小写，单词间以 - 分隔。**[强制]**class 必须代表相应模块或部件的内容或功能，不得以样式信息进行命名。**[强制]** 元素 id 必须保证页面唯一。

**[强制]**class 必须单词全字母小写，单词间以 - 分隔。**[强制]**class 必须代表相应模块或部件的内容或功能，不得以样式信息进行命名。

**[建议]** id 建议单词全字母小写，单词间以 - 分隔。同项目必须保持风格一致。

**[建议]** id、class 命名，在避免冲突并描述清楚的前提下尽可能短。

**[强制]** 同一页面，应避免使用相同的 name 与 id。解释：IE 浏览器会混淆元素的 id 和 name 属性， document.getElementById 可能获得不期望的元素。所以在对元素的 id 与 name属性的命名需要非常小心。一个比较好的实践是，为 id 和 name 使用不同的命名法。示例：

<input name="foo"><div id="foo"></div><script> // IE6 将显示 INPUT alert(document.getElementById('foo').tagName);</script>

2 标签

下面是编写HTML标记的总体原则。提醒大家一点，在创建的HTML文档里总是要使用能够代表

内容语义的标记。

**[强制]**列表中的条目必须总是放置于<ul>、<ol>或<dl> 中，永远不要用一组 <div>或<p> 来表示。

**[强制]**对于无需自闭合的标签，不允许自闭合。例：input、br、img

**[强制]**标签使用必须符合标签嵌套规则。

**[建议]**段落分隔符要使用实际对应的<p>元素，而不是用多个<br>标签。

**[建议]**在合适的条件下，充分利用<dl> （定义列表）和<blockquote> 标签。

**[建议]**给每个表单里的字段加上 <label> 标签，其中的 for 属性必须和对应的输入字段对应，这样用户就可以点击标签。同理，给标签加上 cursor:pointer; 样式也是明智的做法。

**[建议]**不用使用输入字段中的 size 属性。该属性是和输入字段里文本的 font-size 相关的。应该使用CSS宽度。

**[建议]**在某些闭合的 </div> 标签旁边加上一段html注释，说明这里闭合的是什么元素。这在有大量嵌套和缩进的情况下会很有用。

**[建议]**不要把表格用于页面布局。

**[建议]**在合适的条件下，利用 <thead>、<tbody>和<th>标签 (以及Scope属性）。

标签小写、标签闭合  
**[建议]** HTML 标签的使用应该遵循标签的语义。

**[建议]** 在 CSS 可以实现相同需求的情况下不得使用表格进行布局。

**[建议]**尽量减少标签数量；

#### 2.1属性顺序

属性应该按照特定的顺序出现以保证易读性；

class

id

name

data-\*

src, for, type, href, value , max-length, max, min, pattern

placeholder, title, alt

aria-\*, role

required, readonly, disabled

class是为高可复用组件设计的，所以应处在第一位；

id更加具体且应该尽量少使用，所以将它放在第二位。

#### 2.2 JS生成标签

在JS文件中生成标签让内容变得更难查找，更难编辑，性能更差。应该尽量避免这种情况的出现。

#### 2.3 alt标签不为空

<img>标签的 alt 属性指定了替代文本，用于在图像无法显示或者用户禁用图像显示时，代替图像显示在浏览器中的内容。假设由于下列原因用户无法查看图像，alt 属性可以为图像提供替代的信息：

网速太慢

src 属性中的错误

浏览器禁用图像

用户使用的是屏幕阅读器

从SEO角度考虑，浏览器的爬虫爬不到图片的内容，所以我们要有文字告诉爬虫图片的内容

#### 2.4 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。

**2.4.1 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；

**2.4.2 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。

#### 2.5 图片

**[强制]** 禁止img的src取值为空。延迟加载的图片也要增加默认的src。

解释：

src 取值为空，会导致部分浏览器重新加载一次当前页面。

**[建议]** 避免为 img 添加不必要的 title 属性。多余的 title 影响看图体验，并且增加了页面尺寸。

**[建议]** 为重要图片添加 alt 属性，可以提高图片加载失败时的用户体验。

**[建议]** 添加 width 和 height 属性，以避免页面抖动。

**[建议]** 有下载需求的图片采用 img 标签实现，无下载需求的图片采用 CSS 背景图实现。

解释：

1.产品 logo、用户头像、用户产生的图片等有潜在下载需求的图片，以 img 形式实现，能方便用户下载。

2.无下载需求的图片，比如：icon、背景、代码使用的图片等，尽可能采用 CSS 背景图实现。

### 3 HEAD

#### 3.1 HTML5 doctype

在页面开头使用这个简单地doctype来启用标准模式，使其在每个浏览器中尽可能一致的展现；

虽然doctype不区分大小写，但是按照惯例，DOCTYPE大写

#### 3.2 lang属性

根据HTML5规范：

应在html标签上加上lang属性。这会给语音工具和翻译工具帮助，告诉它们应当怎么去发音和翻译。

#### 3.3 字符编码

通过声明一个明确的字符编码，让浏览器轻松、快速的确定适合网页内容的渲染方式，通常指定为'UTF-8'。

#### 3.4 引入CSS, JS

根据**HTML5规范**, 通常在引入CSS和JS时不需要指明 type，因为 text/css 和 text/javascript 分别是他们的默认值。

#### 3.5 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>

#### 3.6 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">

#### 3.7 iOS 图标

apple-touch-icon 图片自动处理成圆角和高光等效果;

apple-touch-icon-precomposed 禁止系统自动添加效果，直接显示设计原图;

<!-- iPhone 和 iTouch，默认 57x57 像素，必须有 -->

<link rel="apple-touch-icon-precomposed" href="/apple-touch-icon-57x57-precomposed.png">

<!-- iPad，72x72 像素，可以没有，但推荐有 -->

<link rel="apple-touch-icon-precomposed" href="/apple-touch-icon-72x72-precomposed.png" sizes="72x72">

<!-- Retina iPhone 和 Retina iTouch，114x114 像素，可以没有，但推荐有 -->

<link rel="apple-touch-icon-precomposed" href="/apple-touch-icon-114x114-precomposed.png" sizes="114x114">

<!-- Retina iPad，144x144 像素，可以没有，但推荐有 -->

<link rel="apple-touch-icon-precomposed" href="/apple-touch-icon-144x144-precomposed.png" sizes="144x144">

#### 3.8 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：

在 Web Server 根目录放置 favicon.ico 文件；

使用 link 指定 favicon；

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

#### 3.9 HEAD 模板

<!DOCTYPE html>

<html lang="zh-cmn-Hans">

<head>

<meta charset="utf-8">

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

<title>Style Guide</title>

<meta name="description" content="不超过150个字符">

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

<meta name="author" content="name, email@gmail.com">

<!-- 为移动设备添加 viewport -->

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

<!-- iOS 图标 -->

<link rel="apple-touch-icon-precomposed" href="/apple-touch-icon-57x57-precomposed.png">

<link rel="alternate" type="application/rss+xml" title="RSS" href="/rss.xml" />

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

</head>

#### 3.10 测试标记：

[W3C校验器](https://validator.w3.org/)

### 4表单

**[强制] 有文本标题的控件必须使用 label 标签将其与其标题相关联。**

**[强制] 使用 button 元素时必须指明 type 属性值**

**[建议] 尽量不要使用按钮类元素的 name 属性。**

解释：

由于浏览器兼容性问题，使用按钮的 name 属性会带来许多难以发现的问题。具体情况可参考[此文](http://w3help.org/zh-cn/causes/CM2001)。

## 三、CSS

### 1 编码总体原则

* 从外部文件加载CSS，尽可能减少文件数。加载标签必须放在文件的 HEAD 部分。
* 用 LINK 标签加载，[永远不要用@import](http://blog.amodernfable.com/2008/01/21/thoughts-on-linking-to-stylesheets/)。加载样式表

<link rel="stylesheet" type="text/css" href="myStylesheet.css" />

不要用内联式样式

<p style="font-size:12px; color:#FFFFFF">This is poor form </p>

* 不要在文件中用内联式引入的样式，不管它是定义在样式标签里还是直接定义在元素上。这样会很难追踪样式规则。
* 定义样式的时候，对样式在页面只出现一次的元素用id，其他的用class。
* 编写性能优化的选择器。尽可能避免使用开销大的CSS选择器。例如，避免 \* 通配符选择器，也不要叠加限定条件到 ID 选择器（例如 div#myid）或 class 选择器（例如 table.results）上。这对于速度至上并包含了成千上万个DOM元素的web应用来说尤为重要。更多相关内容请参阅 MDN 上的这篇[《编写高效CSS》](https://developer.mozilla.org/en/Writing_Efficient_CSS)。
* 不允许有空的规则；
* 元素选择器用小写字母；
* 去掉小数点前面的0；
* 去掉数字中不必要的小数点和末尾的0；
* 属性值'0'后面不要加单位；
* 同个属性不同前缀的写法需要在垂直方向保持对齐，具体参照右边的写法；
* 无前缀的标准属性应该写在有前缀的属性后面；
* 不要在同个规则里出现重复的属性，如果重复的属性是连续的则没关系；
* 不要在一个文件里出现两个相同的规则；
* 用 border: 0; 代替 border: none;；
* 选择器不要超过4层（在scss中如果超过4层应该考虑用嵌套的方式来写）；
* 发布的代码中不要有 @import；
* 尽量少用'\*'选择器。

/\* not good \*/

.element {

}

/\* not good \*/

LI {

...

}

/\* good \*/

li {

...

}

/\* not good \*/

.element {

color: rgba(0, 0, 0, 0.5);

}

/\* good \*/

.element {

color: rgba(0, 0, 0, .5);

}

/\* not good \*/

.element {

width: 50.0px;

}

/\* good \*/

.element {

width: 50px;

}

/\* not good \*/

.element {

width: 0px;

}

/\* good \*/

.element {

width: 0;

}

/\* not good \*/

.element {

border-radius: 3px;

-webkit-border-radius: 3px;

-moz-border-radius: 3px;

background: linear-gradient(to bottom, #fff 0, #eee 100%);

background: -webkit-linear-gradient(top, #fff 0, #eee 100%);

background: -moz-linear-gradient(top, #fff 0, #eee 100%);

}

/\* good \*/

.element {

-webkit-border-radius: 3px;

-moz-border-radius: 3px;

border-radius: 3px;

background: -webkit-linear-gradient(top, #fff 0, #eee 100%);

background: -moz-linear-gradient(top, #fff 0, #eee 100%);

background: linear-gradient(to bottom, #fff 0, #eee 100%);

}

/\* not good \*/

.element {

color: rgb(0, 0, 0);

width: 50px;

color: rgba(0, 0, 0, .5);

}

/\* good \*/

.element {

color: rgb(0, 0, 0);

color: rgba(0, 0, 0, .5);

}

#### 1.1 空格

以下几种情况不需要空格：

* 属性名后
* 多个规则的分隔符','前
* !important '!'后
* 属性值中'('后和')'前
* 行末不要有多余的空格

以下几种情况需要空格：

* 属性值前
* 选择器'>', '+', '~'前后
* '{'前
* !important '!'前
* @else 前后
* 属性值中的','后
* 注释'/\*'后和'\*/'前

/\* not good \*/

.element {

color :red! important;

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

}

/\* good \*/

.element {

color: red !important;

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

}

/\* not good \*/

.element ,

.dialog{

...

}

/\* good \*/

.element,

.dialog {

}

/\* not good \*/

.element>.dialog{

...

}

/\* good \*/

.element > .dialog{

...

}

/\* not good \*/

.element{

...

}

/\* good \*/

.element {

...

}

/\* not good \*/

@if{

...

}@else{

...

}

/\* good \*/

@if {

...

} @else {

...

}

#### 1.2 命名

* 名使用小写字母，以中划线分隔
* id采用驼峰式命名
* scss中的变量、函数、混合、placeholder采用驼峰式命名

/\* class \*/

.element-content {

...

}

/\* id \*/

#myDialog {

...

}

/\* 变量 \*/

$colorBlack: #000;

/\* 函数 \*/

@function pxToRem($px) {

...

}

/\* 混合 \*/

@mixin centerBlock {

...

}

/\* placeholder \*/

%myDialog {

...

}

#### 1.3 颜色

用也违颜色16进制用小写字母；

颜色16进制尽量用简写。

/\* not good \*/

.element {

color: #ABCDEF;

background-color: #001122;

}

/\* good \*/

.element {

color: #abcdef;

background-color: #012;

}

#### 1.4 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，不要驼峰命名法和下划线；
* 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 {}

#### 1.5 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。

#### 1.6 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 { ... }

}

#### 1.7 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；

参考[don’t use @import](http://www.stevesouders.com/blog/2009/04/09/dont-use-import/)；

#### 1.8 无需添加浏览器厂商前缀

使用 [Autoprefixer](https://github.com/postcss/autoprefixer) 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。

### 2 性能优化

#### 2.1 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：

* box-shadows
* border-radius
* transparency
* transforms
* CSS filters（性能杀手）

#### 2.2 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，[更多详情](http://www.jianshu.com/p/e305ace24ddf)。

常见的重排元素:

* width
* height
* padding
* margin
* display
* border-width
* position
* top
* left
* right
* bottom
* font-size
* float
* text-align
* overflow-y
* font-weight
* overflow
* font-family
* line-height
* vertical-align
* clear
* white-space
* min-height

#### 2.3 正确使用 Display 的属性

Display 属性会影响页面的渲染，请合理使用。

* display: inline后不应该再使用 width、height、margin、padding 以及 float；
* display: inline-block 后不应该再使用 float；
* display: block 后不应该再使用 vertical-align；
* display: table-\* 后不应该再使用 margin 或者 float；

#### 2.4不滥用 Float

Float在渲染时计算量比较大，尽量减少使用。

#### 2.5 动画性能优化

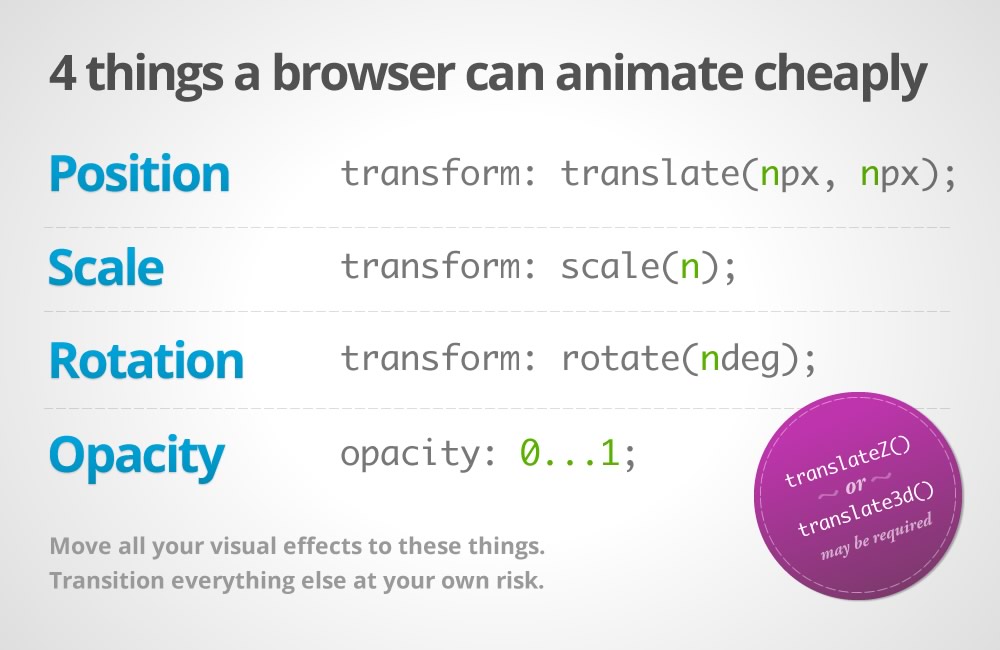
动画的实现原理，是利用了人眼的“视觉暂留”现象，在短时间内连续播放数幅静止的画面，使肉眼因视觉残象产生错觉，而误以为画面在“动”。

动画的基本概念：

* 帧：在动画过程中，每一幅静止画面即为一“帧”;
* 帧率：即每秒钟播放的静止画面的数量，单位是fps(Frame per second);
* 帧时长：即每一幅静止画面的停留时间，单位一般是ms(毫秒);
* 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。

* 如果使用基于 javaScript 的动画，尽量使用 requestAnimationFrame. 避免使用 setTimeout, setInterval.
* 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。
* 使用 translate 取代 absolute 定位就会得到更好的 fps，动画会更顺滑。



#### 2.6 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;

}

#### 2.7 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**。——[更多详情](http://www.jianshu.com/p/268c7f3dd7a6)

**2.7.1 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。

**2.7.2 避免使用标签或 class 选择器限制 id 选择器**

/\* Not recommended \*/

button#backButton {…}

/\* Recommended \*/

#newMenuIcon {…}

**2.7.3 避免使用标签限制 class 选择器**

/\* Not recommended \*/

treecell.indented {…}

/\* Recommended \*/

.treecell-indented {…}

/\* Much to recommended \*/

.hierarchy-deep {…}

**2.7.4 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 {…}

**2.7.5、避免使用子选择器**

/\* Not recommended \*/

treehead treerow treecell {…}

/\* Recommended \*/

treehead > treerow > treecell {…}

/\* Much to recommended \*/

.treecell-header {…}

**2.7.6、使用继承**

/\* Not recommended \*/

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

/\* Recommended \*/

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

## 四 JavaScript

### 1编码总体原则

* 99%的代码必须封装在外部Javascript文件中。这些文件必须在 BODY 标签的尾部引入，让页面的性能最大化。
* 不要依赖于 user-agent 字符串。进行适当的特性检测. （更多信息参见 [深入 HTML5： 检测](http://diveintohtml5.info/detect.html) 和 [jQuery 支持文档](http://api.jquery.com/jQuery.support/)）
* 不要使用 document.write()。
* 所有布尔变量的命名必须用 "is" 开头
* 给变量和函数的命名要有逻辑意义：例如： popUpWindowForAd 就比 myWindow 好多了。
* 不要人为缩短命名到最小。除了传统的 for 循环中的计数器 i 等简化的情况，变量命名必须长到有明确意义。
* 文档必须遵循 [NaturalDocs](http://www.naturaldocs.org/documenting.html) 结构。
* 常量或配置变量（例如动画持续时间等）必须放在文件的顶部。
* 尽力编写可通用化的函数，让它接受参数并返回值。这样有利于充分的代码重用，而且一旦与引入及外部脚本配合起来，能在脚本需要修改时减少开销。例如，相比硬编码一个带有窗口大小、选项和url的弹出式窗口，不如编写一个接受大小、url和选项作为变量的函数。
* 给代码添加注释！这会有利于减少在调试Javascript函数上花费的时间。
* 不要把时间浪费在用 <!-- --> 给你的内联Javascript加注释上，除非你还在关注 Netscape 4。 :)
* 把你的代码组织成一套 [对象常量/单例](http://kaijaeger.com/articles/the-singleton-design-pattern-in-javascript.html)，按照 [模块化模式](http://www.yuiblog.com/blog/2007/06/12/module-pattern/)，或做成 [带构造器的对象](http://mckoss.com/jscript/object.htm)。
* 最小化全局变量 - 你创建的全局变量越少越好。一般来说，用于你的应用命名空间，1会是个好的数字

### 2 编码规范

#### 2.1 注释

#### **原则**

* 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。
* 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。

#### ****2.2 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。

#### ****2.3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。

**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

};

}

#### ****2.4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.

\*/

#### ****2.5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 = {};

**函数**, 使用 Camel 命名法。

函数的**参数**, 使用 Camel 命名法。

function stringFormat(source) {}

function hear(theBells) {}

**类**, 使用 Pascal 命名法

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;

};

**枚举变量** 使用 Pascal 命名法。

**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();

#### 2.6 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);

**jquery对象**必须以'$'开头命名

#### 2.7缩进

使用soft tab（4个空格）。

var x = 1,

y = 1;

if (x < y) {

x += 10;

} else {

x += 1;

}

#### 2.8 单行长度

不要超过80，但如果编辑器开启word wrap可以不考虑单行长度。

#### 2.9 分号

以下几种情况后需加分号：

* 变量声明
* 表达式
* return
* throw
* break
* continue
* do-while

/\* var declaration \*/

var x = 1;

/\* expression statement \*/

x++;

/\* do-while \*/

do {

x++;

} while (x < 10);

#### 2.10 空格

以下几种情况不需要空格：

* 对象的属性名后
* 前缀一元运算符后
* 后缀一元运算符前
* 函数调用括号前
* 无论是函数声明还是函数表达式，'('前不要空格
* 数组的'['后和']'前
* 对象的'{'后和'}'前
* 运算符'('后和')'前

以下几种情况需要空格：

* 二元运算符前后
* 三元运算符'?:'前后
* 代码块'{'前
* 下列关键字前：else, while, catch, finally
* 下列关键字后：if, else, for, while, do, switch, case, try, catch, finally, with, return, typeof
* 单行注释'//'后（若单行注释和代码同行，则'//'前也需要），多行注释'\*'后
* 对象的属性值前
* for循环，分号后留有一个空格，前置条件如果有多个，逗号后留一个空格
* 无论是函数声明还是函数表达式，'{'前一定要有空格
* 函数的参数之间

// not good

var a = {

b :1

};

// good

var a = {

b: 1

};

// not good

++ x;

y ++;

z = x?1:2;

// good

++x;

y++;

z = x ? 1 : 2;

// not good

var a = [ 1, 2 ];

// good

var a = [1, 2];

// not good

var a = ( 1+2 )\*3;

// good

var a = (1 + 2) \* 3;

// no space before '(', one space before '{', one space between function parameters

var doSomething = function(a, b, c) {

// do something

};

// no space before '('

doSomething(item);

// not good

for(i=0;i<6;i++){

x++;

}

// good

for (i = 0; i < 6; i++) {

x++;

}

#### 2.11 空行

以下几种情况需要空行：

* 变量声明后（当变量声明在代码块的最后一行时，则无需空行）
* 注释前（当注释在代码块的第一行时，则无需空行）
* 代码块后（在函数调用、数组、对象中则无需空行）
* 文件最后保留一个空行

// need blank line after variable declaration

var x = 1;

// not need blank line when variable declaration is last expression in the current block

if (x >= 1) {

var y = x + 1;

}

var a = 2;

// need blank line before line comment

a++;

function b() {

// not need blank line when comment is first line of block

return a;

}

// need blank line after blocks

for (var i = 0; i < 2; i++) {

if (true) {

return false;

}

continue;

}

var obj = {

foo: function() {

return 1;

},

bar: function() {

return 2;

}

};

// not need blank line when in argument list, array, object

func(

2,

function() {

a++;

},

3

);

var foo = [

2,

function() {

a++;

},

3

];

var foo = {

a: 2,

b: function() {

a++;

},

c: 3

};

#### 2.12 换行

换行的地方，行末必须有','或者运算符；

以下几种情况不需要换行：

* 下列关键字后：else, catch, finally
* 代码块'{'前

以下几种情况需要换行：

* 代码块'{'后和'}'前
* 变量赋值后

#### 2.13 单行注释

双斜线后，必须跟一个空格；

缩进与下一行代码保持一致；

可位于一个代码行的末尾，与代码间隔一个空格。

#### 2.14 多行注释

最少三行, '\*'后跟一个空格，具体参照右边的写法；

建议在以下情况下使用：

* 难于理解的代码段
* 可能存在错误的代码段
* 浏览器特殊的HACK代码
* 业务逻辑强相关的代码

/\*

\* one space after '\*'

\*/

var x = 1;

#### 2.15 文档注释

各类标签@param, @method等请参考[usejsdoc](http://usejsdoc.org/)和[JSDoc Guide](http://yuri4ever.github.io/jsdoc/)；

建议在以下情况下使用：

* 所有常量
* 所有函数
* 所有类

/\*\*

\* @func

\* @desc 一个带参数的函数

\* @param {string} a - 参数a

\* @param {number} b=1 - 参数b默认值为1

\* @param {string} c=1 - 参数c有两种支持的取值</br>1—表示x</br>2—表示xx

\* @param {object} d - 参数d为一个对象

\* @param {string} d.e - 参数d的e属性

\* @param {string} d.f - 参数d的f属性

\* @param {object[]} g - 参数g为一个对象数组

\* @param {string} g.h - 参数g数组中一项的h属性

\* @param {string} g.i - 参数g数组中一项的i属性

\* @param {string} [j] - 参数j是一个可选参数

\*/

function foo(a, b, c, d, g, j) {

...

}

#### 2.16 引号

最外层统一使用单引号。

#### ****2.17 变量声明****

一个函数作用域中所有的变量声明尽量提到函数首部，用一个var声明，不允许出现两个连续的var声明。

#### 2.18 函数

无论是函数声明还是函数表达式，'('前不要空格，但'{'前一定要有空格；

函数调用括号前不需要空格；

立即执行函数外必须包一层括号；

不要给inline function命名；

参数之间用', '分隔，注意逗号后有一个空格。

// no space before '(', but one space before'{'

var doSomething = function(item) {

// do something

};

function doSomething(item) {

// do something

}

// not good

doSomething (item);

// good

doSomething(item);

// requires parentheses around immediately invoked function expressions

(function() {

return 1;

})();

// not good

[1, 2].forEach(function x() {

...

});

// good

[1, 2].forEach(function() {

...

});

// not good

var a = [1, 2, function a() {

...

}];

// good

var a = [1, 2, function() {

...

}];

// use ', ' between function parameters

var doSomething = function(a, b, c) {

// do something

};

#### 2.19 数组、对象

对象属性名不需要加引号；

对象以缩进的形式书写，不要写在一行；

数组、对象最后不要有逗号。

#### 2.20 括号

下列关键字后必须有大括号（即使代码块的内容只有一行）：if, else, for, while, do, switch, try, catch, finally, with。

#### 2.21 null

适用场景：

* 初始化一个将来可能被赋值为对象的变量
* 与已经初始化的变量做比较
* 作为一个参数为对象的函数的调用传参
* 作为一个返回对象的函数的返回值

不适用场景：

* 不要用null来判断函数调用时有无传参
* 不要与未初始化的变量做比较

// not good

function test(a, b) {

if (b === null) {

// not mean b is not supply

...

}

}

var a;

if (a === null) {

...

}

// good

var a = null;

if (a === null) {

...

}

#### 2.22 undefined

永远不要直接使用undefined进行变量判断；

使用typeof和字符串'undefined'对变量进行判断。

// not good

if (person === undefined) {

...

}

// good

if (typeof person === 'undefined') {

...

}

#### 2.23 jshint

用'===', '!=='代替'==', '!='；

for-in里一定要有hasOwnProperty的判断；

不要在内置对象的原型上添加方法，如Array, Date；

不要在内层作用域的代码里声明了变量，之后却访问到了外层作用域的同名变量；

变量不要先使用后声明；

不要在一句代码中单单使用构造函数，记得将其赋值给某个变量；

不要在同个作用域下声明同名变量；

不要在一些不需要的地方加括号，例：delete(a.b)；

不要使用未声明的变量（全局变量需要加到.jshintrc文件的globals属性里面）；

不要声明了变量却不使用；

不要在应该做比较的地方做赋值；

debugger不要出现在提交的代码里；

数组中不要存在空元素；

不要在循环内部声明函数；

不要像这样使用构造函数，例：new function () { ... }, new Object；

// not good

if (a == 1) {

a++;

}

// good

if (a === 1) {

a++;

}

// good

for (key in obj) {

if (obj.hasOwnProperty(key)) {

// be sure that obj[key] belongs to the object and was not inherited

console.log(obj[key]);

}

}

// not good

Array.prototype.count = function(value) {

return 4;

};

// not good

var x = 1;

function test() {

if (true) {

var x = 0;

}

x += 1;

}

// not good

function test() {

console.log(x);

var x = 1;

}

// not good

new Person();

// good

var person = new Person();

// not good

delete(obj.attr);

// good

delete obj.attr;

// not good

if (a = 10) {

a++;

}

// not good

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

// not good

var nums = [];

for (var i = 0; i < 10; i++) {

(function(i) {

nums[i] = function(j) {

return i + j;

};

}(i));

}

// not good

var singleton = new function() {

var privateVar;

this.publicMethod = function() {

privateVar = 1;

};

this.publicMethod2 = function() {

privateVar = 2;

};

};

#### 2.24 调试

即使采用了最好的校验器，浏览器的怪异性也会不可避免地产生一些问题。有几个堪称无价之宝的工具可以帮助优化代码的健全性和加载速度。很重要的一点是，你手头准备好了这些工具，不管你主要用来开发的浏览器是哪个。我们推荐先在Firefox和Safari上做开发，然后用Google Chrome和Opera，最后针对IE用条件判断做一些额外的微调。下面列出的是一些有帮助的调试器和速度分析器：

* **Firefox**: [Firebug](http://getfirebug.com/), [Page Speed](http://code.google.com/speed/page-speed/), [YSlow](http://developer.yahoo.com/yslow/)
* **Safari**: [Web Inspector](http://developer.apple.com/safari/library/documentation/AppleApplications/Conceptual/Safari_Developer_Guide/UsingtheWebInspector/UsingtheWebInspector.html)
* **Google Chrome**: [Developer Tools](http://google.com/chrome/intl/en/webmasters-faq.html#tools)
* **Opera**: [Dragonfly](http://opera.com/dragonfly/)
* **Internet Explorer 6-7**: [Developer Toolbar](http://microsoft.com/downloads/details.aspx?familyid=E59C3964-672D-4511-BB3E-2D5E1DB91038)
* **Internet Explorer 8-10**: [Developer Tools](http://msdn.microsoft.com/en-us/library/dd565625(v=VS.85).aspx)

#### 2.25 避免全局命名空间污染

防止全局命名空间被污染，我们通常的做法是将代码包裹成一个 IIFE(Immediately-Invoked Function Expression)，创建独立隔绝的定义域。也使得内存在执行完后立即释放。

IIFE 还可确保你的代码不会轻易被其它全局命名空间里的代码所修改（i.e. 第三方库，window 引用，被覆盖的未定义的关键字等等）。

(function($, w, d){ 'use strict'; $(function() { w.alert(d.querySelectorAll('div').length); }); }(jQuery, window,

document));

#### 2.26 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。

#### 2.27 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，例如

var a = b = 0; //b会被隐式的创建为全局变量

所以，请总是使用 var 来声明变量，并且使用单var模式（将所有的变量在函数最前面只使用一个var定义）。例如：

(function (){

'use strict'

var a = 0,

b = 0,

c = 0,

i,

j,

myObject();

}())

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。

### 3 性能优化

#### 1 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>";

}

#### 2 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++) {

}

#### 3 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');

}());

#### 4 避免使用 jQuery 实现动画

禁止使用 slideUp/Down() fadeIn/fadeOut() 等方法；

尽量不使用 animate() 方法；

#### 5 借力 CSS 精灵

CSS 精灵（Sprites）基本上就是把一批图片资源合并成单个图片文件。然后每个部分用 CSS background-position 来展现。典型的 CSS 看起来是这样的

1.a.expandbox { display:block; width: 75px; height: 15px; text-indent: -999px; overflow:hidden;

2.background: url(../img/sprite.png) no-repeat -50px -5px; }

接力 sprites 实现 :hover 悬停效果是很普遍的。这种情况下你会看到类似于这样的定义：

1.a.expandbox:hover { background-position: -50px -20px; }

使用 sprites 可以减少页面大小，也减少了HTTP连接数，这会加速页面加载。 [在 css-tricks.com 上有更多总体性的技术和概述](http://css-tricks.com/css-sprites/)。尽可能地利用CSS 精灵总体来说是一项最佳实践。

除了主要的sprite之外，很多开发者还会使用一个垂直方向的sprite。这个垂直方向的sprite的宽度（以及高度）会**小于或等于100px**，包含了通常一个挨着一个的图标，诸如列表元素的着重号或对应功能的链接和按钮。Yahoo 就用到了一些，[例如这个](http://l.yimg.com/a/i/ww/sp/pa-icons3.gif)。

有个注意事项就是别把sprite弄得太大，不管是高还是宽超过1000px都会导致用掉大量内存。你可以学习一下 [使用sprite的时机及内存占用](http://blog.vlad1.com/2009/06/22/to-sprite-or-not-to-sprite/)，如果需要了解更多关于创建sprite的总体性提示和技巧，请参阅 [Mozilla 开发博客](http://blog.mozilla.com/webdev/2009/03/27/css-spriting-tips/)。

#### 6 图片格式

你应该会用到四种主要的图片格式，细节如下：

**JPEG.** - 这种格式涵盖了所有基于摄影的图片，例如主页和目录页推销广告图片，或任何颜色数很高的图片。

**PNG24.** - 这种格式在Photoshop里用起来很方便，它输出高颜色数图片，并完全支持像素级的渐变透明度。相对来说，它是相当重量级的格式，达到KB级大小。它也是IE6唯一需要执行pngfix的格式。在那种情况下，本公司推荐使用 [DD\_belatedPNG 脚本](http://www.dillerdesign.com/experiment/DD_belatedPNG/) （这是一个 pngfix ，它能修正 PNG24 在 IE6 里冒出灰色或浅蓝色背景的问题。它们并不总是和background-position 兼容）。

在很多情况下，你可以使用GIF替代PNG24作为对IE6的应变方案。这在需要用PNG24做sprite的情况下尤其适用。所有的 pngfixes 都会特别慢而且开销巨大，所以最好不要用它们。

**PNG8.** - 在256色中可以抓取到如此多样的颜色，所以用JPG之前尝试一下PNG是值得的。而且，PNG比GIF有更高的可压缩度（使用类似 pngcrush 和 pngquant的工具）。这种格式支持在几乎所有浏览器中实现渐变透明度，但在IE6中那些半透明像素会显示为100%透明。不过在很多情况下这也够用了。它也无需运行pngfix脚本，所以对于速度是优化的。

Photoshop 无法正确输出这些半透明文件，但是Fireworks 可以。更多细节请参阅： <http://www.sitepoint.com/png8-the-clear-winner/>

**透明 GIF 89a.** - GIF 89a 提供透明度的灵活性和广泛的浏览器支持，但也有限制，它不支持渐变透明度和超过256的色深。就我们的经验而言，64的色深就可以提供质量很好的缩略图，并保持相对小的文件大小。

所有低颜色数图片，例如图表或主题图形应该用PNG8格式制作，因为它是这四种格式中具有最高文件大小效率的。PNG8是我们对于大部分网站图形的主要推荐方案。

关于PNG格式、浏览器支持以及各种格式优缺点的详细信息可以在 [这篇优秀的文章](http://calendar.perfplanet.com/2010/png-that-works/) 中找到。

如需进一步优化所有这些格式，从 [Yahoo的 Smush.It](http://smush.it/) 查看图片可以发现缩小它们的办法。

#### 7 缓存

对于静态内容，浏览器应该把文件在本地缓存，在合理的前提下，保留尽可能长的时间。应该较长远期缓存的内容包括：

CSS 和 JavaScript

产品图片

主题图形

favicon.ico

flash .swf's

优惠信息图片（可能缓存时间会略短）

为了最佳缓存效果，利用http头部的Expires。下面是一个远期的Expires头，它告诉浏览器这个响应在2015年5月15日之前都无需更新：

Expires: Thu, 15 Apr 2015 20:00:00 GMT

如果你的服务器是Apache，可以使用ExpiresDefault指令设置相对当前日期的失效日期。下面的例子设置失效日期为请求时间的一年之后：

ExpiresDefault "access plus 1 year"

http头部的 Expires 必须设为据现在一个月到一年（远期）之间的值。缓存只适用于那个指定的URL，所以文件名或任何资源的改变都会产生一个新的拷贝。很多开发者使用build过程来给它们的资源增加一个版本号或时间戳。每个随后的build会开始一个全新的缓存版本，让你在使用远期缓存日期时无需担心。[Google 的这篇文章里有更多关于浏览器缓存的细节信息](http://code.google.com/speed/page-speed/docs/caching.html#LeverageBrowserCaching)。

#### 8 把资源文件跨域分片

静态内容当然应该由不同于HTML所在服务器的另一个域提供访问。这是优化的方案，这样 [对所有静态内容的请求就无需额外的cookies头](http://developer.yahoo.com/performance/rules.html#cookie_free)。而且为整个域编写缓存规则也就容易得多了。（同时，当前域的任何子域会继承当前域的cookies，所以使用全新的域是值得的）。

不过，对于足够多的资源（特别是图片），请求数的增加足以让页面加载变慢。很多浏览器对于他们从每个域能并发下载的资源数有比较低的限制。（在IE6和IE7，这个数字仅仅是2）。在这种情况下，我们可以把资源存放在多个子域，例如：

static1.otherdomain.com

static2.otherdomain.com

static3.otherdomain.com

[Google Speed 上更多有关域分片的信息](http://code.google.com/speed/page-speed/docs/rtt.html#ParallelizeDownloads)

#### 9 避免用IFRAME

Iframe 是能添加到指定页面的各种元素中上开销最大的一个。它们会阻塞页面让它无法触发onload事件，直到它们加载完成。有时候它们被其他机构用来处理追踪脚本。例如 Doubleclick floodlight 标签就是一个 iframe，管理员可以从他们的管理面板向里面增加追踪像素。在加入iFrame的任何情况下，它应该在window的onload事件被触发之后再动态加入到DOM中。 [更多细节请参阅 Yahoo 的性能站点。](http://developer.yahoo.com/performance/rules.html#iframes)