前端开发规范手册

此手册主要实现的目标：代码一致性和最佳实践。通过代码风格的一致性，降低维护代码的成本以及改善多人协作的效率。同时遵守最佳实践，确保页面性能得到最佳优化和高效的代码。 此手册是在开发中积累下来的经...

目　录

致谢

当前文档 《前端开发规范手册》 由 进击的皇虫 使用 书栈网(BookStack.CN) 进行构建，生成于 2020-02-29。

书栈网仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈网难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到书栈网，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到书栈网获取最新的文档，以跟上知识更新换代的步伐。

内容来源：Aaaaaashu https://github.com/Aaaaaashu/Guide

文档地址：http://www.bookstack.cn/books/Aaaaaashu-Guide

书栈官网：https://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

前端开发规范手册

Github: 仓库地址

知笔墨: 手册地址

此手册主要实现的目标：代码一致性和最佳实践。通过代码风格的一致性，降低维护代码的成本以及改善多人协作的效率。同时遵守最佳实践，确保页面性能得到最佳优化和高效的代码。

此手册是在开发中积累下来的经验和参考其它规范/指南制定的，它只是起指导作用，除个别条目强制之外，大多数为非强制约束，开发者可根据自己的实际情况自行决定是否要遵守该指南只是保证大方向一致性和最佳实践的阶段性总结，不是最后结论，它会随着时间而变化。

目录


Front End Developing Style Guide
前端開發規範手册
HuskyPress®
阿樹
http://Aaaaaashu.me

基本原则

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。

缩进

统一两个空格缩进（总之缩进统一即可），不要使用 `Tab` 或者 `Tab` 、空格混搭。

文件编码

使用不带 `BOM` 的 **UTF-8** 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` （参考 `@charset`）；

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;
```

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` 问题，减小文件字节数。

其它协议（ `ftp` 等）的 **URL** 不省略。

```
<!-- Recommended -->
<script src="//www.google.com/js/gweb/analytics/autotrack.js"></script>

<!-- Not recommended -->
<script src="http://www.google.com/js/gweb/analytics/autotrack.js"></script>
```

```
/* Recommended */
.example {
  background: url(//www.google.com/images/example);
}

/* Not recommended */
.example {
  background: url(http://www.google.com/images/example);
}
```

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。

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)
-->
```

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;
}
```

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

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
    };
}
```

- 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.
 */
```

代码验证

- 使用 W3C HTML Validator 来验证你的HTML代码有效性；
- 使用 W3C CSS Validator 来验证你的CSS代码有效性；

代码验证不是最终目的，真的目的在于让开发者在经过多次的这种验证过程后，能够深刻理解到怎样的语法或写法是非标准和不推荐的，即使在某些场景下被迫要使用非标准写法，也可以做到心中有数。

HTML

尽量遵循 HTML 标准和语义，但是不要以牺牲实用性为代价。任何时候都要尽量使用最少的标签并保持最小的复杂度。

通用约定

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="images/google.png" 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="...">
```

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，采用中划线 `-` 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>
```

属性顺序

HTML 属性应该按照特定的顺序出现以保证易读性。

- id
- class
- name
- data-xxx
- src, for, type, href

- title, alt
- aria-xxx, role

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

<input class="form-control" type="text">

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

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>
```

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。

更多详情，参考WEB标准系列-HTML元素嵌套

布尔值属性

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

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

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

```
4.
5.  <select>
6.    <option value="1" selected>1</option>
7.  </select>
```

语义化

没有 `CSS` 的 `HTML` 是一个语义系统而不是 UI 系统。

通常情况下，每个标签都是有语义的，所谓语义就是你的衣服分为外套， 裤子，裙子，内裤等，各自有对应的功能和含义。所以你总不能把内裤套在脖子上吧。— 一丝

此外语义化的 `HTML` 结构，有助于机器（搜索引擎）理解，另一方面多人协作时，能迅速了解开发者意图。

常见标签语义

标签	语义
`<p>`	段落
`<h1> <h2> <h3> ...`	标题
`<ul>`	无序列表
`<ol>`	有序列表
`<blockquote>`	大段引用
`<cite>`	一般引用
`<b>`	为样式加粗而加粗
`<strong>`	为强调内容而加粗
`<i>`	为样式倾斜而倾斜
`<em>`	为强调内容而倾斜
`code`	代码标识
`abbr`	缩写

示例

将你构建的页面当作一本书，将标签的语义对应的其功能和含义；

- 书的名称：`<h1>`
- 书的每个章节标题：`<h2>`
- 章节内的文章标题：`<h3>`
- 小标题/副标题：`<h4> <h5> <h6>`
- 章节的段落：`<p>`

更多语义化的内容，参考 sofish 写的文章 这样去写你的 HTML。

HEAD

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>
```

语言属性

为什么使用 lang=”zh-cmn-Hans” 而不是我们通常写的 lang=”zh-CN” 呢？请参考知乎上的讨论：网页头部的声明应该是用 lang=”zh” 还是 lang=”zh-cn”？

```
<!-- 中文 -->
<html lang="zh-Hans">

<!-- 简体中文 -->
<html lang="zh-cmn-Hans">

<!-- 繁体中文 -->
<html lang="zh-cmn-Hant">

<!-- English -->
<html lang="en">
```

字符编码

- 以无 BOM 的 utf-8 编码作为文件格式；
- 指定字符编码的 meta 必须是 head 的第一个直接子元素；请参考前端观察的博文： HTML5 Charset 能用吗？

```
<html>
  <head>
    <meta charset="utf-8">
    ......
  </head>
  <body>
    ......
  </body>
</html>
```

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">
```

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>
```

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">
```

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">
```

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">
```

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>
```

CSS

通用约定

代码组织

- 以组件为单位组织代码段；
- 制定一致的注释规范；
- `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；
- 如果使用了多个 CSS 文件，将其按照组件而非页面的形式分拆，因为页面会被重组，而组件只会被移动；

良好的注释是非常重要的。请留出时间来描述组件（component）的工作方式、局限性和构建它们的方法。不要让你的团队其它成员 来猜测一段不通用或不明显的代码的目的。

提示：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。

```
/* ========================================================================
   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;
}
```

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，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 {}
```

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;
}
```

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;
}
```

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?

```
/* 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"] {

}
```

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 { ... }
}
```

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
- 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;

链接的样式顺序：

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

无需添加浏览器厂商前缀

使用 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，按兼容的要求，对相应的 CSS 代码添加浏览器厂商前缀。

字体排印

暂时参考网页字体排印指南

TODO

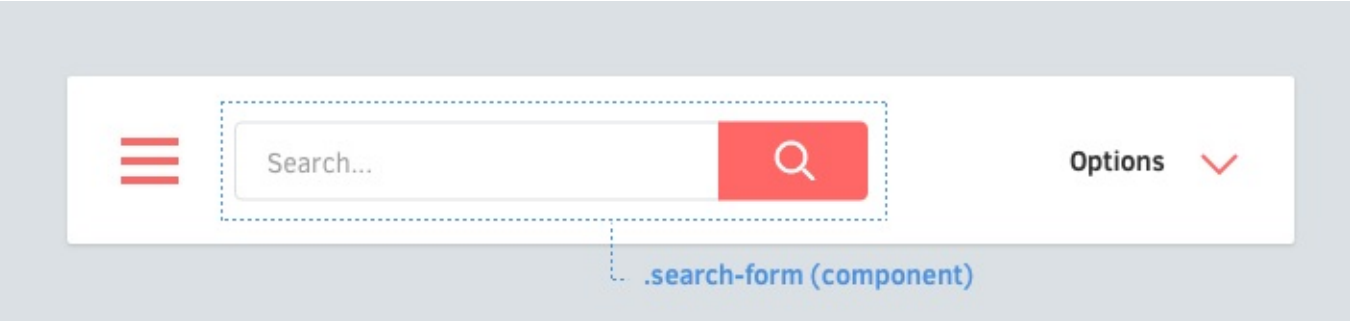
模块组织

任何超过 1000 行的 CSS 代码，你都曾经历过这样的体验：

1. 这个 class 到底是什么意思呢？
2. 这个 class 在哪里被使用呢？
3. 如果我创建一个 `xxoo` class，会造成冲突吗？

`Reasonable System for CSS Stylesheet Structure` 的目标就是解决以上问题，它不是一个框架，而是通过规范，让你构建更健壮和可维护的 CSS 代码。

Components（组件）



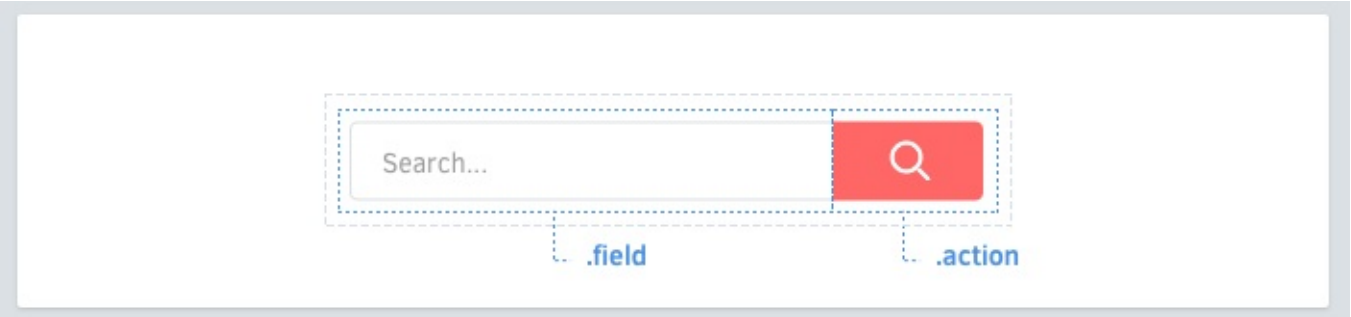
从 `Components` 的角度思考，将网站的模块都作为一个独立的 `Components` 。

Naming components （组件命名）

`Components` 最少以两个单词命名，通过 `-` 分离，例如：

- 点赞按钮（`.like-button`）
- 搜索框（`.search-form`）
- 文章卡片（`.article-card`）

Elements （元素）



`Elements` 是 `Components` 中的元素

Naming elements （元素命名）

`Elements` 的类名应尽可能仅有一个单词。

```
.search-form {
  > .field { /* ... */ }
  > .action { /* ... */ }
}
```

On multiple words （多个单词）

对于倘若需要两个或以上单词表达的 `Elements` 类名，不应使用中划线和下划线连接，应直接连接。

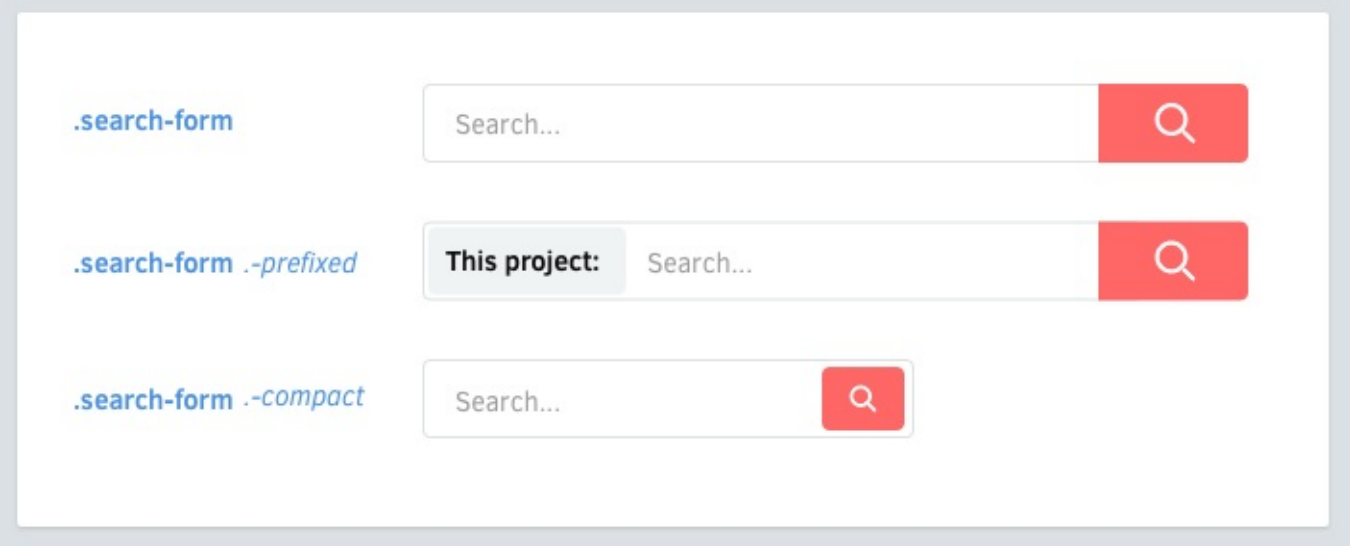
```
.profile-box {
  > .firstname { /* ... */ }
  > .lastname { /* ... */ }
  > .avatar { /* ... */ }
}
```

Avoid tag selectors （避免标签选择器）

任何时候尽可能使用 `classnames` 。标签选择器在使用上没有问题，但是其性能上稍弱，并且表意不明确。

```
.article-card {
  > h3    { /* ✗ avoid */ }
  > .name { /* ✓ better */ }
}
```

Variants （变体）



`Components` 和 `Elements` 可能都会拥有 `Variants` 。

Naming variants （变体命名）

`Variants` 的 `classname` 应带有前缀中划线 `-`

```
.like-button {
```

```
    &.-wide { /* ... */ }
    &.-short { /* ... */ }
    &.-disabled { /* ... */ }
  }
```

Element variants （元素变体）

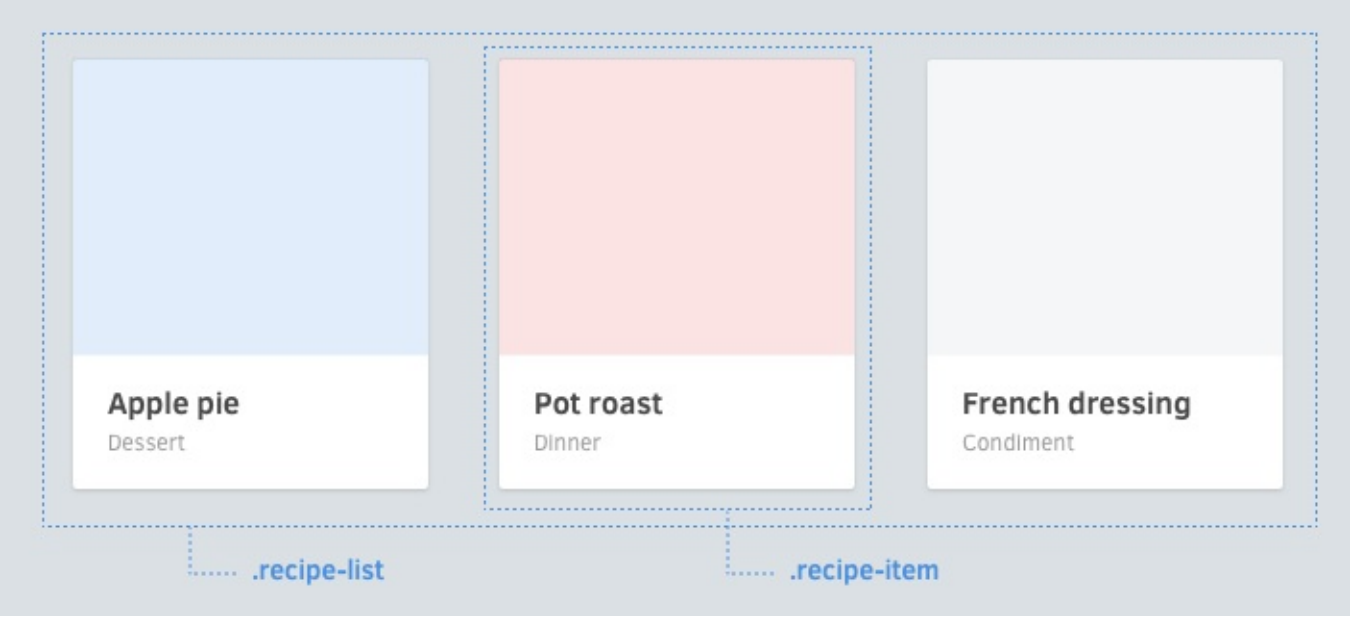
```
  .shopping-card {
    > .title { /* ... */ }
    > .title.-small { /* ... */ }
  }
```

Dash prefixes （中划线前缀）

为什么使用中划线作为变体的前缀？

- 它可以避免歧义与 `Elements`
- CSS class 仅能以单词和 `_` 或 `-` 开头
- 中划线比下划线更容易输出

Layout （布局）



Avoid positioning properties （避免定位属性）

Components 应该在不同的上下文中都可以复用，所以应避免设置以下属性：

- Positioning (position, top, left, right, bottom)
- Floats (float, clear)
- Margins (margin)
- Dimensions (width, height) *

Fixed dimensions （固定尺寸）

头像和 logos 这些元素应该设置固定尺寸（宽度，高度…）。

Define positioning in parents （在父元素中设置定位）

倘若你需要为组件设置定位，应将在组件的上下文（父元素）中进行处理，比如以下例子中，将 `widths` 和 `floats` 应用在 `list component(.article-list)` 当中，而不是 `component(.article-card)` 自身。

```
.article-list {
  & {
    @include clearfix;
  }

  > .article-card {
    width: 33.3%;
    float: left;
  }
}

.article-card {
  & { /* ... */ }
  > .image { /* ... */ }
  > .title { /* ... */ }
  > .category { /* ... */ }
}
```

Avoid over-nesting （避免过分嵌套）

当出现多个嵌套的时候容易失去控制，应保持不超过一个嵌套。

```
/* ✗ Avoid: 3 levels of nesting */
.image-frame {
  > .description {
    /* ... */

    > .icon {
      /* ... */
    }
  }
}

/* ✓ Better: 2 levels */
.image-frame {
  > .description { /* ... */ }
  > .description > .icon { /* ... */ }
}
```

Apprehensions （顾虑）

- 中划线 `-` 是一坨糟糕的玩意：其实你可以选择性的使用，只要将 `Components, Elements, Variants` 记在心上即可。
- 我有时候想不出两个单词唉：有些组件的确使用一个单词就能表意，比如 `aleter` 。但其实你可以使用后缀，使其意识更加明确。

比如块级元素：

- `.alert-box`
- `.alert-card`
- `.alert-block`

或行内级元素

- `.link-button`
- `.link-span`

Terminologies （术语）

RSCSS 与其他 CSS 模块组织系统相似的概念

RSCSS	BEM	SMACSS
Component	Block	Module
Element	Element	?
Layout	?	Layout
Variant	Modifier	Theme & State

Summary （总结）

- 以 `Components` 的角度思考，以两个单词命名（ `.screenshot-image` ）
- `Components` 中的 `Elements` ，以一个单词命名（ `.blog-post .title` ）
- `Variants` ，以中划线 `-` 作为前缀（ `.shop-banner.-with-icon` ）
- `Components` 可以互相嵌套
- 记住，你可以通过继承让事情变得更简单

译自:Reasonable System for CSS Stylesheet Structure

LESS

代码组织

代码按一下顺序组织：

1. @import
2. 变量声明
3. 样式声明

```
@import "mixins/size.less";

@default-text-color: #333;

.page {
  width: 960px;
  margin: 0 auto;
}
```

@import 语句

@import 语句引用的文需要写在一对引号内，.less 后缀不得省略。引号使用 ' 和 " 均可，但在同一项目内需统一。

```
/* Not recommended */
@import "mixins/size";
@import 'mixins/grid.less';

/* Recommended */
@import "mixins/size.less";
@import "mixins/grid.less";
```

混入（Mixin）

1. 在定义 mixin 时，如果 mixin 名称不是一个需要使用的 className，必须加上括号，否则即使不被调用也会输出到 CSS 中。
2. 如果混入的是本身不输出内容的 mixin，需要在 mixin 后添加括号（即使不传参数），以区分这是否是一个 className。

```
/* Not recommended */
.big-text {
  font-size: 2em;
}

```

```
h3 {
  .big-text;
  .clearfix;
}

/* Recommended */
.big-text() {
  font-size: 2em;
}

h3 {
  .big-text(); /* 1 */
  .clearfix(); /* 2 */
}
```

避免嵌套层级过多

- 将嵌套深度限制在2级。对于超过3级的嵌套，给予重新评估。这可以避免出现过于详实的CSS选择器。
- 避免大量的嵌套规则。当可读性受到影响时，将之打断。推荐避免出现多于20行的嵌套规则出现。

字符串插值

变量可以用类似ruby和php的方式嵌入到字符串中，像@{name}这样的结构：`@base-url: "http://assets.fnord.com";` `background-image: url("@{base-url}/images/bg.png");`

性能优化

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）

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。

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

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;

不滥用 Float

Float在渲染时计算量比较大，尽量减少使用。

动画性能优化

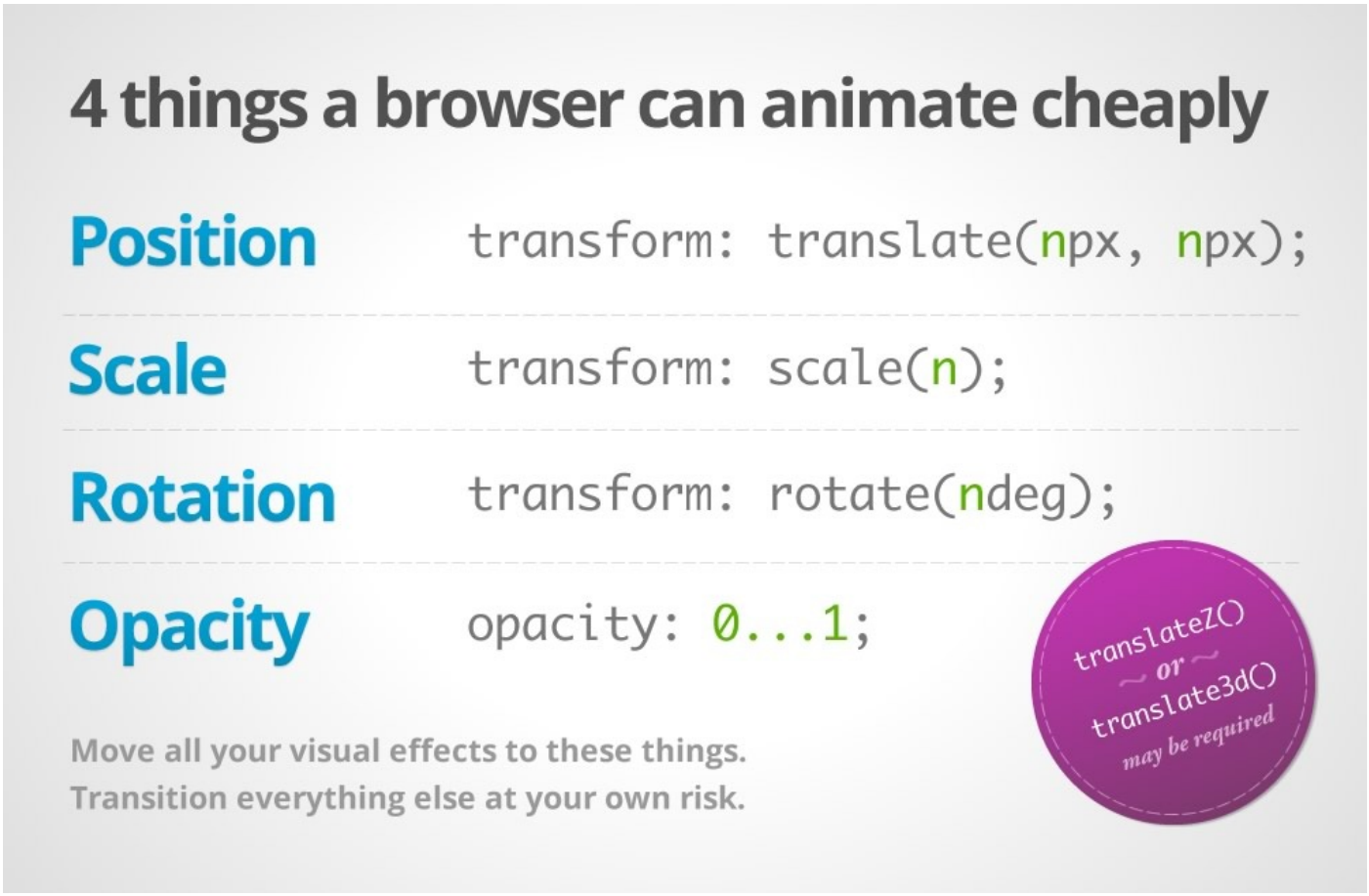
动画的实现原理，是利用了人眼的“视觉暂留”现象，在短时间内连续播放数幅静止的画面，使肉眼因视觉残象产生错觉，而误以为画面在“动”。

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。



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;
}
```

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

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) }
```

JavaScript

通用约定

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。

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

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
    };
}
```

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.
 */
```

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();
```

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);
```

接口命名规范

1. 可读性强，见名晓义；
2. 尽量不与 jQuery 社区已有的习惯冲突；
3. 尽量写全。不用缩写，除非是下面列表中约定的；（变量以表达清楚为目标，uglify 会完成压缩体积工作）

常用词	说明
options	表示选项，与 jQuery 社区保持一致，不要用 config, opts 等
active	表示当前，不要用 current 等

index	表示索引，不要用 idx 等
trigger	触点元素
triggerType	触发类型、方式
context	表示传入的 this 对象
object	推荐写全，不推荐简写为 o, obj 等
element	推荐写全，不推荐简写为 el, elem 等
length	不要写成 len, l
prev	previous 的缩写
next	next 下一个
constructor	不能写成 ctor
easing	示动画平滑函数
min	minimize 的缩写
max	maximize 的缩写
DOM	不要写成 dom, Dom
.hbs	使用 hbs 后缀表示模版
btn	button 的缩写
link	超链接
title	主要文本
img	图片路径（img标签src属性）
dataset	html5 data-xxx 数据接口
theme	主题
className	类名
classNameSpace	class 命名空间

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:

- null
- undefined
- '' 空字符串
- 0 数字0

但小心下面的, 可都返回 true:

- '0' 字符串0
- [] 空数组
- {} 空对象

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]);
  }
}
```

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().
    doSomething().
    doSomethingElse();
```

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();
```

&& 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;
  // ...
```

```
24. }
25.
26. var kid = node && node.kids && node.kids[index];
27. if (kid) {
28.   foo(kid);
29. }
```

jQuery 规范

使用最新版本的 jQuery

最新版本的 jQuery 会改进性能和增加新功能，若不是为了兼容旧浏览器，建议使用最新版本的 jQuery。以下是三条常见的 jQuery 语句，版本越新，性能越好：

```
1. $('.elem')
2. $('.elem', context)
3. context.find('.elem')
```

分别使用 *1.4.2*、*1.4.4*、*1.6.2* 三个版本测试浏览器在一秒内能够执行多少次，结果 *1.6.2* 版执行次数远超两个老版本。

jQuery 变量

1. 存放 jQuery 对象的变量以 `$` 开头；
2. 将 jQuery 选择器返回的对象缓存到本地变量中复用；
3. 使用驼峰命名变量；

```
1. var $myDiv = $("#myDiv");
2. $myDiv.click(function(){...});
```

选择器

1. 尽可能的使用 ID 选择器，因为它会调用浏览器原生方法 `document.getElementById` 查找元素。当然直接使用原生 `document.getElementById` 方法性能会更好；
2. 在父元素中选择子元素使用 `.find()` 方法性能会更好，因为 ID 选择器没有使用到 Sizzle 选择器引擎来查找元素；

```
1. // Not recommended
2. var $productIds = $("#products .class");
3.
4. // Recommended
5. var $productIds = $("#products").find(".class");
```

DOM 操作

1. 当要操作 DOM 元素的时候，尽量将其分离节点，操作结束后，再插入节点；
2. 使用字符串连接或 `array.join` 要比 `.append()` 性能更好；

```
1. var $myList = $("#list-container > ul").detach();
2. //...a lot of complicated things on $myList
3. $myList.appendTo("#list-container");
```

```
// Not recommended
var $myList = $("#list");
for(var i = 0; i < 10000; i++){
    $myList.append("<li>"+i+"</li>");
}

// Recommended
var $myList = $("#list");
var list = "";
for(var i = 0; i < 10000; i++){
    list += "<li>"+i+"</li>";
}
$myList.html(list);

// Much to recommended
var array = [];
for(var i = 0; i < 10000; i++){
    array[i] = "<li>"+i+"</li>";
}
$myList.html(array.join(''));
```

事件

1. 如果需要，对事件使用自定义的 `namespace`，这样容易解绑特定的事件，而不会影响到此 DOM 元素的其他事件监听；
2. 对 Ajax 加载的 DOM 元素绑定事件时尽量使用事件委托。事件委托允许在父元素绑定事件，子代元素可以响应事件，也包括 Ajax 加载后添加的子代元素；

```
$("#myLink").on("click.mySpecialClick", myEventHandler);
$("#myLink").unbind("click.mySpecialClick");
```

```
// Not recommended
$("#list a").on("click", myClickHandler);

// Recommended
$("#list").on("click", "a", myClickHandler);
```

链式写法

1. 尽量使用链式写法而不是用变量缓存或者多次调用选择器方法；
2. 当链式写法超过三次或者因为事件绑定变得复杂后，使用换行和缩进保持代码可读性；

```
$("#myDiv").addClass("error").show();
```

```
$("#myLink")
  .addClass("bold")
  .on("click", myClickHandler)
  .on("mouseover", myMouseOverHandler)
  .show();
```

其他

1. 多个参数使用对象字面量存储;
2. 不要将 CSS 写在 jQuery 里面;
3. 正则表达式仅准用 .test() 和 .exec() 。不准用 “string”.match() ;

jQuery 插件模板

```
// jQuery Plugin Boilerplate
// A boilerplate for jumpstarting jQuery plugins development
// version 1.1, May 14th, 2011
// by Stefan Gabos

// remember to change every instance of "pluginName" to the name of your plugin!
(function($) {

    // here we go!
    $.pluginName = function(element, options) {

        // plugin's default options
        // this is private property and is  accessible only from inside the plugin
        var defaults = {

            foo: 'bar',

            // if your plugin is event-driven, you may provide callback capabilities
            // for its events. execute these functions before or after events of your
            // plugin, so that users may customize those particular events without
            // changing the plugin's code
            onFoo: function() {}

        }

        // to avoid confusions, use "plugin" to reference the
        // current instance of the object
        var plugin = this;

        // this will hold the merged default, and user-provided options
        // plugin's properties will be available through this object like:
        // plugin.settings.propertyName from inside the plugin or
        // element.data('pluginName').settings.propertyName from outside the plugin,
```

```
34.         // where "element" is the element the plugin is attached to;
35.         plugin.settings = {}
36.
37.         var $element = $(element), // reference to the jQuery version of DOM element
38.              element = element;    // reference to the actual DOM element
39.
40.         // the "constructor" method that gets called when the object is created
41.         plugin.init = function() {
42.
43.             // the plugin's final properties are the merged default and
44.             // user-provided options (if any)
45.             plugin.settings = $.extend({}, defaults, options);
46.
47.             // code goes here
48.
49.         }
50.
51.         // public methods
52.         // these methods can be called like:
53.         // plugin.methodName(arg1, arg2, ... argn) from inside the plugin or
54.         // element.data('pluginName').publicMethod(arg1, arg2, ... argn) from outside
55.         // the plugin, where "element" is the element the plugin is attached to;
56.
57.         // a public method. for demonstration purposes only - remove it!
58.         plugin.foo_public_method = function() {
59.
60.             // code goes here
61.
62.         }
63.
64.         // private methods
65.         // these methods can be called only from inside the plugin like:
66.         // methodName(arg1, arg2, ... argn)
67.
68.         // a private method. for demonstration purposes only - remove it!
69.         var foo_private_method = function() {
70.
71.             // code goes here
72.
73.         }
74.
75.         // fire up the plugin!
76.         // call the "constructor" method
77.         plugin.init();
78.
79.     }
80.
81.     // add the plugin to the jQuery.fn object
```

```
    $.fn.pluginName = function(options) {

        // iterate through the DOM elements we are attaching the plugin to
        return this.each(function() {

            // if plugin has not already been attached to the element
            if (undefined == $(this).data('pluginName')) {

                // create a new instance of the plugin
                // pass the DOM element and the user-provided options as arguments
                var plugin = new $.pluginName(this, options);

                // in the jQuery version of the element
                // store a reference to the plugin object
                // you can later access the plugin and its methods and properties like
                // element.data('pluginName').publicMethod(arg1, arg2, ... argn) or
                // element.data('pluginName').settings.propertyName
                $(this).data('pluginName', plugin);

            }

        });

    }

})(jQuery);
```

此 jQuery 插件模板出自: jQuery Plugin Boilerplate, revisited

性能优化

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>";
}
```

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++) {

}
```

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');

}());
```

避免使用 jQuery 实现动画

1. 禁止使用 slideUp/Down() fadeIn/fadeOut() 等方法;
2. 尽量不使用 animate() 方法;

移动端优化

click 的 300ms 延迟响应

click 的 300ms 延迟是由双击缩放(double tap to zoom)所导致的，由于用户可以进行双击缩放或者双击滚动的操作，当用户一次点击屏幕之后，浏览器并不能立刻判断用户是确实要打开这个链接，还是想要进行双击操作。因此，移动端浏览器就等待 300 毫秒，以判断用户是否再次点击了屏幕。

随着响应式网页逐渐增多，用户使用双击缩放机会减少，这 300ms 的延迟就更不可接受了。浏览器开发商也随之提供相应的解决方案。这些方案在5 Ways to Prevent the 300ms Click Delay on Mobile Devices 中，被提及的包括「禁用缩放」和「width=device-width」等方案，但这些方案并不完美，需要针对某些版本浏览器，又或仅在 Android 的浏览器上使用。

所以这时候就需要一个更简单通用的解决方案，其中 FT Labs 专门为解决移动端浏览器 300 毫秒点击延迟问题所开发的一个轻量级的库 FastClick 就是很好的选择。FastClick 在检测到 touchend 事件的时候，会通过 DOM 自定义事件立即触发一个模拟 click 事件，并把浏览器在 300 毫秒之后真正触发的 click 事件阻止掉。

FastClick 的使用方法非常简单，在 window load 事件之后，在 `<body>` 上调用 `FastClick.attach()` 即可。

```
1.  window.addEventListener( "load", function() {
2.      FastClick.attach( document.body );
3.  }, false );
```

快速回弹滚动

快速回弹滚动在手机浏览器上的发展历史：

1. 早期的时候，移动端的浏览器都不支持非 body 元素的滚动条，所以一般都借助 iScroll;
2. Android 3.0 / iOS 解决了非 body 元素的滚动问题，但滚动条不可见，同时 iOS 上只能通过2个手指进行滚动;
3. Android 4.0 解决了滚动条不可见及增加了快速回弹滚动效果，不过随后这个特性又被移除;
4. iOS从5.0开始解决了滚动条不可见及增加了快速回弹滚动效果

如果想要为某个元素拥有 Native 般的滚动效果，可以这样操作：

```
1.  .element {
2.      overflow: auto; /* auto | scroll */
3.      -webkit-overflow-scrolling: touch;
4.  }
```

除了 iScroll 之外，还有一个更加强大的滚动插件 Swiper，支持 3D 和内置滚动条等。

设备检测

```
// 这段代码引用自：https://github.com/binnng/device.js

var WIN = window;
var LOC = WIN["location"];
var NA = WIN.navigator;
var UA = NA.userAgent.toLowerCase();

function test(needle) {
  return needle.test(UA);
}

var IsTouch = "ontouchend" in WIN;
var IsAndroid = test(/android|htc/) || /linux/i.test(NA.platform + "");
var IsIPad = !IsAndroid && test(/ipad/);
var IsIPhone = !IsAndroid && test(/ipod|iphone/);
var IsIOS = IsIPad || IsIPhone;
var IsWinPhone = test(/windows phone/);
var IsWebapp = !!NA["standalone"];
var IsXiaoMi = IsAndroid && test(/mi\s+/);
var IsUC = test(/ucbrowser/);
var IsWeixin = test(/micromessenger/);
var IsBaiduBrowser = test(/baidubrowser/);
var IsChrome = !!WIN["chrome"];
var IsBaiduBox = test(/baiduboxapp/);
var IsPC = !IsAndroid && !IsIOS && !IsWinPhone;
var IsHTC = IsAndroid && test(/htc\s+/);
var IsBaiduWallet = test(/baiduwallet/);
```

获取滚动条值

PC 端滚动条的值是通过 `document.scrollTop` 和 `document.scrollLeft` 获得，但在 iOS 中并没有滚动条的概念，所以仅能通过 windows.scroll 获取，同时也能兼容 Android 。

```
window.scrollY
window.scrollX
```

清除输入框内阴影

在 iOS 上，输入框默认有内部阴影，但无法使用 box-shadow 来清除，如果不需要阴影，可以这样操作：

```
input,
textarea {
    border: 0; /* 方法1 */
    -webkit-appearance: none; /* 方法2 */
}
```

Meta 相关

页面窗口自动调整到设备宽度，并禁止用户缩放页面

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-
1.  scale=1.0, user-scalable=no" />
```

电话号码识别

iOS Safari (Android 或其他浏览器不会) 会自动识别看起来像电话号码的数字，将其处理为电话号码链接，比如:

- 7位数字，形如: 1234567
- 带括号及加号的数字，形如: (+86)123456789
- 双连接线的数字，形如: 00-00-00111
- 11位数字，形如: 13800138000

```
1.  <!-- 关闭电话号码识别： -->
2.  <meta name="format-detection" content="telephone=no" />
3.
4.  <!-- 开启电话功能： -->
5.  <a href="tel:123456">123456</a>
6.
7.  <!-- 开启短信功能： -->
8.  <a href="sms:123456">123456</a>
```

邮箱地址的识别

在 Android (iOS 不会) 上，浏览器会自动识别看起来像邮箱地址的字符串，不论有你没有加上邮箱链接，当你在这个字符串上长按，会弹出发邮件的提示。

```
<!-- 关闭邮箱地址识别： -->
<meta name="format-detection" content="email=no" />

<!-- 开启邮件发送： -->
<a mailto:>mobile@gmail.com">mobile@gmail.com</a>
```

指定 iOS 的 safari 顶端状态条的样式

```
<meta name="apple-mobile-web-app-status-bar-style" content="black" />
<!-- 可选default、black、black-translucent -->
```

工具箱

HTMLelement.info

Can I use

垂直居中

Mastering the :nth-child

Regulex

YOU MIGHT NOT NEED JQUERY

Flexbox in 5 minutes

待续…

参考

编码规范 by @mdo

Baidu 前端规范

Qunar 前端规范

常用的 HTML 头部标签

Google Style Guide

Amazeui Style Guide

CSS Guidelines

20 Snippets You should be using from Html5 Boilerplate

谈谈CSS性能

Web动画性能指南

Why Moving Elements With Translate() Is Better Than Pos:abs Top/left

High Performance Animations

jQuery Standards

Learn jQuery

无线Web开发经验谈

移动端前端笔记大全