前端开发规范

大纲

- 基本原则
- HTML
  - 通用约定
  - 语义化
  - HEAD
- CSS
  - 通用约定
  - 字体排印
  - 模块组织
  - Less规范
  - 性能优化
- JavaScript
  - 通用约定
  - Jquery规范
  - 性能优化
- 移动端优化

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
  - 在 HTML中指定编码 <meta charset=“utf-8”>
- 一律使用小写字母

```
<!-- Recommended -->
<img src="google.png" alt="Google">

<!-- Not recommended -->
<A HREF="/">Home</A>
```

- 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。
  - 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。

基本原则

- HTML 注释
  - 模块注释

```
<!-- 文章列表列表模块 -->
<div class="article-list">
...
</div>
```

```
<!--
@name: Drop Down Menu
@description: Style of top bar drop down menu.
@author: Ashu(Aaaaaashu@gmail.com)
-->
```

- CSS 注释

```
/**
 * default development theme for jQRangeSlider
 * Using fam fam icon set from Mark James, http://www.famfamfam.com/lab/icons/silk/ (Creative Commons Attribution 2.5 License)
 */

.ui-rangeSlider{
  height:22px;
}
```

基本原则

- 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。
    - 函数注释

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

HTML-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： 或 </body> ）；
  - 尽量减少标签数量；
- 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，同时避免创建无样式信息的 class；
- 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>
```

HTML-通用约定

- 嵌套
  - 语义嵌套约束
    - 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
    - <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 、<h1>~<h6>、<p>、<ul>/<ol>/<li>、<dl>/<dt>/<dd>、<form>等。
  - 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>

<select>
  <option value="1" selected>1</option>
</select>
```

HTML-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。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。
- 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>

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

```
<!-- 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">
```

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；
  - 良好的注释是非常重要的。请留出时间来描述组件的工作方式、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。

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

CSS-通用约定

- 声明顺序
  - 相关属性应为一组，推荐的样式编写顺序
    - Positioning
    - Box model
    - Typographic
    - Visual

```
.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;
}
```

  - 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。

CSS-通用约定

- 引号使用

```
/* Not recommended */
html {
  font-family: 'open sans', arial, sans-serif;
}

/* Recommended */
html {
  font-family: "open sans", arial, sans-serif;
}
```

- 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。关于媒体查询：CSS媒体查询

```
@media (max-width: 768px) {
  .element { ...}
  .element-avatar { ... }
  .element-selected { ... }
}
```

CSS-通用约定

- 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；CSS Less Sass 介绍
    - 其他 CSS 文件合并工具；
- 链接的样式顺序：

```
a:link -> a:visited -> a:hover -> a:active
```

CSS字体排印+性能优化

- 字体排印
  - 网页字体排印指南
- 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）　　关于CSS filters
  - 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。　　前端工程师需要明白的「浏览器渲染」
  - 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。

CSS性能优化

- 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;
}
```

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。
- 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。
    - `function stringFormat(source) {}`
  - 函数的参数，使用 Camel 命名法。
    - `function hear(theBells) {}`

JavaScript通用约定

- 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
  - 可读性强，见名晓义；
  - 尽量不与 jQuery 社区已有的习惯冲突；
  - 尽量写全。不用缩写，除非是下面列表中约定的；
- 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。

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

```
// Not recommended
var $productIds = $("#products .class");

// Recommended
var $productIds = $("#products").find(".class");
```

jquery规范

- DOM 操作
  - 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；
  - 使用字符串连接或 array.join 要比 .append()性能更好；

```
// 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(''));
```

jquery规范

- 事件
  - 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；
  - 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；

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

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

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

- 链式写法
  - 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；
  - 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；

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

JavaScript性能优化

- 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

```
// 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>";
}
```

JavaScript性能优化

- 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++) {

}
```

JavaScript性能优化

- 避免使用 jQuery 实现动画
  - 慎重使用 slideUp/Down() fadeIn/fadeOut() 等方法；
  - 尽量不使用 animate() 方法；

移动端优化

- 快速回弹滚动
  - 快速回弹滚动在手机浏览器上的发展历史：
  - 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;
  - 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;
  - 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;
  - 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：

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

移动端优化

- 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/);
```

移动端优化

- 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 */
}
```

移动端优化

- 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-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

```
<!-- 关闭电话号码识别： -->
<meta name="format-detection" content="telephone=no" />
<!-- 开启电话功能： -->
<a href="tel:123456">123456</a>
<!-- 开启短信功能： -->
<a href="sms:123456">123456</a>
```

移动端优化

- 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>
```