**前端开发规范2017-V1.0**

1. **基本原则**

**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。

## 2、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

## 3、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](https://developer.mozilla.org/en-US/docs/Web/CSS/@charset)）；

**Html,css一律使用小写字母**

## 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) 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);}

/\*Notrecommended\*/

.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](mailto: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
* 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.

\*/

## 代码验证

* 使用 [W3C HTML Validator](http://validator.w3.org/) 来验证你的HTML代码有效性；
* 使用 [W3C CSS Validator](http://jigsaw.w3.org/css-validator/validator.html.zh-cn) 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。

1. [**HTML通用约定**](http://www.w3cschool.cn/webdevelopment/8yd31ozt.html)

## 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="//img.w3cschool.cn/attachments/image/cimg/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](http://www.smallni.com/element-nesting/)

## 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**>

1. [**css通用约定**](http://www.w3cschool.cn/webdevelopment/8yd31ozt.html)

## 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](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 {}

## 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?](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"] {

}

## 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 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/)；

## 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](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。

1. [**JavaScript通用约定**](http://www.w3cschool.cn/webdevelopment/jpbfrozt.html)

## 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().

**do**Something().

**do**SomethingElse();

## 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;

// ...

}

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

**if** (kid) {

foo(kid);

}

1. [**移动端优化**](http://www.w3cschool.cn/webdevelopment/dv4ocozt.html)

## 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。解决300ms有几种方法：

1、使用[fastclick.js](http://amazeui.org/1.x/javascript/fastclick/)。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。FastClick会产生穿透bug以及其他bug。

window.addEventListener( "load", **function**() {

FastClick.attach( document.body );

}, false );

2、使用[hammer.js](http://blog.csdn.net/kongjiea/article/details/43016085)。除了解决了延迟300ms的点击问题，还包含了缩放、滑动、平移都事件。

<div id="test" class="test"></div>

<script type="text/javascript">

//创建一个新的hammer对象并且在初始化时指定要处理的dom元素

var hammertime = new Hammer(document.getElementById("test"));

//为该dom元素指定点击事件

hammertime.on("tap", function (ev) {

//控制台输出

console.log(ev);

});

</script>

3、使用[zepto.js](http://www.css88.com/doc/zeptojs_api/)。Zepto是一个轻量级的针对现代高级浏览器的JavaScript库， 它与jquery有着类似的api。 如果你会用jquery，那么你也会用zepto。不过zepto的tap事件也会有穿透的bug，与jquery相比，也会少了一些api，因此对于基于jquery的一些插件可能会不支持。

4、使用原生的[touch事件](http://www.2cto.com/kf/201402/276737.html)。同样存在穿透的bug。

## [网上关于穿透的解决办法](https://segmentfault.com/a/1190000003848737)。

## 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：

.element {

**overflow**: auto; /\* auto | scroll \*/

**-webkit-overflow-scrolling**: touch;

}

除了 [iScroll](http://cubiq.org/iscroll-4) 之外，还有一个更加强大的滚动插件 [Swiper](http://www.swiper.com.cn)，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-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**>

### 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 -->

1. [**css性能优化**](http://www.w3cschool.cn/webdevelopment/xdy8nozt.html)

## 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](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

## 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。使用flex布局。

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](http://www.jianshu.com/p/268c7f3dd7a6)

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) }

1. [**js**](http://www.w3cschool.cn/webdevelopment/xdy8nozt.html)**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');

}());

1. [**其他性能优化**](http://www.w3cschool.cn/webdevelopment/xdy8nozt.html)

小图标，容易使用**css实现**的，尽量不用图片，比如箭头。

## 使用 CSS Sprites 整合图像，减少图标请求次数。

在要求不是很高清的情况下，**图片应该压缩80%**质量，甚至可以更低。大部分情况下。Png格式会比jpg格式占用内存。在不影响图片质量的情况下，建议使用内存占用较低的图片格式。

**避免死链**，比如图片如果不存在，给个默认的图片

应该为网页添加icon。

<link rel="icon" href="32x32.png" sizes="32x32" />

<link rel="icon" href="192x192.png" sizes="192x192" />

Js文件，大部分情况下，应该置于body结束符前，文档加载顺序是从上至下的，放在head区域会阻塞页面渲染。

对于频繁操作的事件，比如scroll，resize。应当使用节流函数。<www.cnblogs.com/haoxl/archive/2016/03/07/5252409.html>

1. [**前端开发参考资料**](http://www.w3cschool.cn/webdevelopment/xdy8nozt.html)

主要来源：<http://www.w3cschool.cn/webdevelopment/index.html>

支付宝移动Web解决方案

<http://am-team.github.io/amg/dev-exp-doc.html>

移动端前端笔记大全

http://www.w3cmark.com/2015/398.html