前端开发规范：

**命名规则**

**1、目录命名：按功能模块定义**

- module ---例:home,lib,detail,search,plugin

-**---** css

-**---** images

-**---** js

**2、js文件命名：**

- 按功能（或者page）名字定义.js

- 全部采用小写方式，有复数结构时，要采用复数命名法，以下划线分隔。

**3、css文件命名：**

- 全部采用小写方式，有复数结构时，要采用复数命名法，以下划线分隔。

**4、html文件命名：**

- 按功能（或者page）名字定义.html

- 全部采用小写方式，有复数结构时，要采用复数命名法，以下划线分隔。

**HTML篇**

**1、语法**

1、缩进使用soft tab（4个空格）；

2、嵌套的节点应该缩进；

3、在属性上，使用双引号，不要使用单引号；

4、属性名全小写，用中划线做分隔符；

5、不要忽略可选的关闭标签，例：</li> 和 </body>；

**2、结构**

**2.1 head部分**

* 设!DOCTYPE

以**<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"**

**"http://www.w3.org/TR/html4/loose.dtd">**开头

* lang属性：

<html lang="zh-cn">

* 标题:

<title>页面标题必须含有</title>

* 关键词：

<meta name="keywords" content="必须有kewords">

* 描述：

<meta name="description" content="必须有description">

* IE兼容模式：

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

* 字符编码：

<meta charset="UTF-8">

* 引入css文件：

一般页面有2个css文件即可,common.css,page.css

* 引入js:

这里只放3个js文件，

- jquery.min.js（v1.8.3版本）

- *<!--[if IE]>jquery.ie.js<![endif]-->*,

- layer.js

**2.2 body部分**

* 自上而下的结构顺序：

- <**div** id="header">公共头部</**div**>

- <**div** id="nav">公共导航</**div**>

- <**div** id="main">主体内容</**div**>

- <**div** id="side">侧边fixed,一般是客服、反馈、置顶</**div**>

- <**div** id="footer">公共底部</**div**>

- <**script**>指定相关变量值以及配置值，多是后端赋值的</**script**>

- <**script**>业务相关性js文件</**script**>

**2.3 标签部分**

* 块元素、内联元素、可变元素：

- 区别哪些标签是块元素、内联元素、可变元素

- 内联元素内不能再包含块元素

* 尽量减少标签数量：

- 在编写HTML代码时，需要尽量避免多余的父节点；

- 很多时候，需要通过迭代和重构来使HTML变得更少。

* 实用高于完美：

- 尽量遵循HTML标准和语义，但是不应该以浪费实用性作为代价；

- 任何时候都要用尽量小的复杂度和尽量少的标签来解决问题。

* h1：

- 适用网站logo、网站导航的第一个超链接、分类名称、产品标题名称，

- 里面不能包含html标签

- 整个页面有且仅有一个

* h2:

- 适用于分类页面、内容页面标题链接、列表名称，

- 里面不包含html标签

- 可以有多个

* h3:

- 适用于细分页面标题链接、tag页面、相关产品内容；

* h4~h6:

- 较少使用，建议用p代替

* ul:

- 无序列表

- 结构层次：ul-li-a-span|strong|small

* ol:

- 有序列表

- 结构层次：ol-li-a-span|strong|small

* dl:

- 适用于某小区域描述

- **dl** **dt** **dd**

- dt为标题，dd为内容

- 多行内容可以采用dd里面包含多个p，也可以并列多个**dd**

* a:

- 必须写title，

- href="链接"，如果不做任何操作href="javascript:;"，

- 根据业务决定是否写target="\_blank"

* img:

- 必须写alt，

- 建议写title，

- 要考虑onerror,onload,懒加载，

- 如果该图片只是为了样式的话，用css的background代替img标签

* button

- 可变元素

- 不用input **type**="button"，而用<button **type**="button">提交</button>

* strong/b:

- **strong：强调作用，搜索引擎会知道这是要强调的内容，在样式上为加粗；**

- **b：一样有加粗效果，但仅仅是样式**

* em/i:

- em全称：emphasize。同样起强调作用，搜索引擎也会知道，在样式上为斜体；

- i:一样有斜体效果，但仅仅是样式

**2.4 属性部分**

* id：

- 命名规则：全部采用小写方式，有复数结构时，要采用复数命名法，以下划线分隔。

- 页面唯一性且为大块体，

- 利于jquery的选择，

- 增加css的权重，

- 不要随便使用

- 命名和业务相关，而非布局板块，一看命名就知道是什么业务功能。

* class：

- 命名规则：全部采用小写方式，有复数结构时，要采用复数命名法，以中划线分隔。

- 能用标签语义化代替就尽量不写

* data-，aria-, role-：

- 书写规则：**data**-属性(即**key**)="值（即val）"

* boolean属性：

+ 只写属性而不指定属性值时，表示属性值为true

+ 如果想要将属性值设为false，可以不使用该属性

- **readonly**为true时只写**readonly**,而不用**readonly**="readonly"

- disabled为true时只写disabled,而不用disabled="disabled"

- **checked**为true时只写**checked**,而不用**checked**="checked"

- selected为true时只写selected,而不用selected="selected"

* 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

**3、css篇**

* 缩进：

- 使用soft tab（4个空格）

* 分号：

- 每个属性声明末尾都要加分号，即';'

* 空格的合理使用：

+ 以下几种情况不要加空格：

- 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

- 属性值中的','后

* 空行：

- 尽量少使用空行

* 换行：

+ 以下几种情况不需要换行：

- '{'前

+ 以下几种情况需要换行：

- '{'后和'}'前

- 每个属性独占一行

- 多个规则的分隔符','后

* 引号：

+ 最外层统一使用双引号；

+ url的内容要用引号；

- eg：background-image: url("logo.png");

- why：1、标准加；2、从安全角度来讲是要加上的，否则容易被xss，具体的不知详

+ 属性选择器中的属性值需要引号。

* 属性声明顺序：

+ 相关的属性声明按右边的顺序做分组处理，组之间需要有一个空行。

+ 以Positioning Model > Box Model > Typographic > Visual 的顺序书写，提高代码的可读性。

- Positioning Model 布局方式、位置，

相关属性包括：position, top, z-index, display, float等

- Box Model 盒模型，相关属性包括：width, height, padding,

margin，border,overflow

- Typographic 文本排版，相关属性包括：font, line-height,

text-align

- Visual 视觉外观，相关属性包括：color, background,

list-style, transform, animation

* 颜色:

+ 颜色16进制用小写字母:

- color: #ABCDEF;应该写成color: #abcdef;

- 颜色16进制尽量用简写;

- background-color: #001122;应该写成：background-color: #012;

* 属性简写:

+ 属性简写需要非常清楚属性值的正确顺序，

大多数情况下并不需要设置属性简写中包含的所有值，

所以建议尽量分开声明会更加清晰；

+ margin 和 padding 相反，需要使用简写；

+ 常见的属性简写包括：

- font

- background

- transition

- animation

* !important 规则例外

当 !important 规则被应用在一个样式声明中时，该样式声明会覆盖CSS中任何其他的声明。

!important规则还是与优先级毫无关系。

使用 !important 不是一个好习惯，因为它改变了你样式表本来的级联规则，从而使其难以调试。

一些经验法则：

**Always** 要优化考虑使用样式规则的优先级来解决问题而不是 !important

**Only** 只在需要覆盖全站或外部 css（例如引用的 **ExtJs** 或者 **YUI** ）的特定页面中使用 !important

**Never** 永远不要在全站范围的 css 上使用 !important

**Never** 永远不要在你的插件中使用 !important

* 杂项:

+ 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。

* 最佳写法

*/\* 这是某个模块 \*/*

.m-nav{}*/\* 模块容器 \*/*

.m-nav **li**,.m-nav **a**{}*/\* 先共性 优化组合 \*/*

.m-nav **li**{}*/\* 后个性 语义化标签选择器 \*/*

.m-nav **a**{}*/\* 后个性中的共性 按结构顺序 \*/*

.m-nav **a**.a1{}*/\* 后个性中的个性 \*/*

.m-nav **a**.a2{}*/\* 后个性中的个性 \*/*

.m-nav .z-crt **a**{}*/\* 交互状态变化 \*/*

.m-nav .z-crt **a**.a1{}

.m-nav .z-crt **a**.a2{}

.m-nav .btn{}*/\* 典型后代选择器 \*/*

.m-nav .btn-1{}*/\* 典型后代选择器扩展 \*/*

.m-nav .btn-dis{}*/\* 典型后代选择器扩展（状态） \*/*

.m-nav .btn.z-dis{}*/\* 作用同上，请二选一（如果可以不兼容IE6时使用） \*/*

.m-nav .m-sch{}*/\* 控制内部其他模块位置 \*/*

.m-nav .u-sel{}*/\* 控制内部其他元件位置 \*/*

.m-nav-1{}*/\* 模块扩展 \*/*

.m-nav-1 **li**{}

.m-nav-dis{}*/\* 模块扩展（状态） \*/*

.m-nav.z-dis{}*/\* 作用同上，请二选一（如果可以不兼容IE6时使用） \*/*

**4、js篇**

* 缩进：

- 使用soft tab（4个空格）

* 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**

* 空格

+ 以下几种情况不需要空格：

- 对象的属性名后

var a = {

b: 1

};

- 前缀一元运算符后

++x;

- 后缀一元运算符前

y++;

- 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

var a = [1, 2];

- 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

* 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

* 引号

+ 最外层统一使用单引号。

* 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。

* 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。

* 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（绝对不要加，IE7直接报语法错误）。

* 大括号

+ 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

* 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

* 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。

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

...

}

* jshint

+ 用'===', '!=='代替'==', '!='；

+ **for**-**in**里一定要有hasOwnProperty的判断；

- **for** (key **in** obj) {

**if** (obj.hasOwnProperty(key)) {

console.log(obj[key]);

}

}

+ 不要在内置对象的原型上添加方法，如Array, Date；

- Array.prototype.count = **function**(value) {

**return** 4;

};

+ 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；

- **var** person = **new** Person();

+ 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；

* 其他

+ 不要混用tab和space；不要在一处使用多个tab或space；

+ 对上下文this的引用只能使用'**that**'来命名；

+ 行尾不要有空白字符；

+ switch的falling **through**和no default的情况一定要有注释特别说明；

+ 不允许有空的代码块。

**5、常用的判断**

* 判断元素是否存在

$('#id').length

* 判断checkbox/radio元素是否选中

$('input[**type**=checkbox]').prop('checked') // 返回 true,false

* 判断元素是否显示

$("#elem\_id").is(":hidden") *//是否隐藏*

　　$("#elem\_id").is(":visible") *//是否可见*

　　$('#elem\_id').css('display') == 'none'

* 判断元素是否存在class

$("#elem\_id").hasClass("active")

* 判断元素上是否绑定某个事件

$("#foo").data("events")["click"]

* 判断值是否为空

var a = $*("#id)*.val*()*;

*//为空*

**if***( a == null || typeof a == undefined || a == "")* {

　　　　...

　　}

**if***( a.length === 0)*{

...

　　}

*//不为空*

**if***( a != null && typeof a !=undefined && a != "" )* {

...

}

**if***( a )*{

...

　　}

* 判断浏览器

*//1、判断是否IE浏览器*

**var** isIE=window.ActiveXObject || "ActiveXObject" **in** window

*//取得浏览器的userAgent字符串*

**var** userAgent = navigator.userAgent.toLowerCase();

*//2、判断是否Opera浏览器*

**var** isOpera = userAgent.indexOf("Opera") > -1;

*//3、判断是否IE的Edge浏览器*

**var** isEdge = userAgent.indexOf("Edge") > -1;

*//4、判断是否Firefox浏览器*

**var** isFF = userAgent.indexOf("Firefox") > -1;

*//5、判断是否Safari浏览器*

**var** isSafari = userAgent.indexOf("Safari") > -1 &&

userAgent.indexOf("Chrome") == -1;

*//6、判断Chrome浏览器*

**var** isChrome = userAgent.indexOf("Chrome") > -1 &&

userAgent.indexOf("Safari") > -1&&!isEdge;

**6、性能优化篇**

* 加载优化

加载过程是最为耗时的过程，可能会占到总耗时的80%时间，因此是优化的重点

+ 减少HTTP请求

- 合并CSS、JavaScript；

- 合并小图片，使用雪碧图；

+ 缓存

- 缓存一切可缓存的资源；

- 使用长Cache（使用时间戳更新Cache）；

- 使用外联式引用CSS、JavaScript；

- 依赖后端动态赋值的常量定义放在页面内，且位于业务JavaScript上面；

+ 压缩HTML、CSS、JavaScript

- 压缩（例如，多余的空格、换行符和缩进）

启用GZip

+ 无阻塞

- CSS放在页面头部并使用Link方式引入, 避免在HTML标签中写Style

- JavaScript放在页面头部的必须是无异步的，一般是jquery，或者其他依赖插件

- JavaScript业务操作类放在页面尾部或使用异步方式加载

+ 使用首屏加载

- 首屏的快速显示，可以大大提升用户对页面速度的感知；

- 应尽量针对首屏的快速显示做优化。

+ 按需加载

- 将不影响首屏的资源和当前屏幕资源不用的资源放到用户需要时才加载；

- 可以大大提升重要资源的显示速度和降低总体流量；

- PS：按需加载会导致大量重绘，影响渲染性能

- LazyLoad；

- 滚屏加载；

- 通过Media Query加载；

+ 预加载

- 大型重资源页面（如游戏）可使用增加Loading的方法，资源加载完成后再显示页面。

- 但Loading时间过长，会造成用户流失。

- 可感知Loading(如进入空间游戏的Loading）；

- 不可感知的Loading（如提前加载下一页）；

+ ajax请求合理选择get/post方式

- GET只使用一个TCP数据包（Http Header与data）发送数据，所以首选GET方法;

- POST请求时分成两步，先发送Http Header，再发送data;

+ 尽量减少iframe的使用

- <iframe>的优点：

· 可以用来加载速度较慢的第三方资源，如广告、徽章；

· 可用作安全沙箱；

· 可以并行下载脚本。

- <iframe>的缺点：

· 加载代价昂贵，即使是空的页面；

· 阻塞页面 load 事件触发；

Iframe 完全加载以后，父页面才会触发 load 事件。

Safari、Chrome 中通过 JavaScript 动态设置 iframe src可以避免这个问题。

· 缺乏语义。

+ 避免404错误

- 外部脚本返回404，不仅阻塞其他资源下载；

- 浏览器还会尝试把404页面内容当作JavaScript解析，消耗更多资源。

+ 避免空的src和href

- 虽然src属性为空字符串，但浏览器仍然会向服务器发起一个HTTP请求：

· IE 向页面所在的目录发送请求；

· Safari、Chrome、Firefox向页面本身发送请求；

· Opera不执行任何操作。

- 空src产生请求的后果不容小觑：

· 给服务器造成意外的流量负担，尤其时日 PV 较大时；

· 浪费服务器计算资源；

· 可能产生报错。

- 空的href属性也存在类似问题。用户点击空链接时，浏览器也会向服务器发送HTTP请求，

· 可以通过JavaScript阻止空链接的默认的行为。

+ 减少Cookie

- Cookie会影响加载速度，所以静态资源域名不使用Cookie。

+ 避免重定向

- 重定向会影响加载速度，所以在服务器正确设置避免重定向。

+ 异步加载第三方资源

- 第三方资源不可控会影响页面的加载和显示，因此要异步加载第三方资源。

* 图片优化

+ 使用时选择最合适的格式

- webP优于JPG

- PNG8优于GIF

+ 使用其它方式代替图片

- 使用CSS3

- 使用SVG

- 使用IconFont

+ 使用Srcset

+ 合适的大小

- 首次加载不大于1014KB

- 不宽于640

- 使用tinypng.com压缩

* css优化

+ 尽量避免写在HTML标签中写Style属性

+ 避免CSS表达式

- CSS表达式的执行需跳出CSS树的渲染，因此请避免CSS表达式。

+ 移除空的CSS规则

- 空的CSS规则增加了CSS文件的大小，且影响CSS树的执行，所以需移除空的CSS规则。

+ 正确使用Display的属性

- 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。

+ 不滥用Web字体

- Web字体需要下载，解析，重绘当前页面，尽量减少使用。

+ 不声明过多的Font-size

- 过多的Font-size引发CSS树的效率。

+ 值为0时不需要任何单位

- 为了浏览器的兼容性和性能，值为0时不要带单位。

+ 标准化各种浏览器前缀

- 无前缀应放在最后

- CSS动画只用 （-webkit- 无前缀）两种即可

- 其它前缀为 -webkit- -moz- -ms- 无前缀 四种

+ 避免让选择符看起来像正则表达式

- 高级选择器执行耗时长且不易读懂，避免使用。

* js优化

+ 减少重绘和回流

- 避免不必要的Dom操作

- 尽量改变Class而不是Style，使用classList代替className

- 避免使用document.write

- 减少drawImage

+ 避免全局查找

- 函数中会用到全局对象存储为局部变量来减少全局查找

- 访问局部变量的速度要比访问全局变量的速度更快

**function** **fun**(){

**var** \_location = window.location;

console.log(location.href + location.host);

}

+ 定时器

- 如果针对的是不断运行的代码，不应该使用setTimeout，而应该是用setInterval

- setTimeout每一次都会初始化一个定时器

- setInterval只会在开始的时候初始化一个定时器

+ 字符串连接

- 如果要连接多个字符串，应该少使用+=

- x+=a; x+=b; x+=c; 应该写成 x+= a + b + c；

- 多次对同一个字符串进行+=操作的话,使用JavaScript数组来收集，最后使用join方法连接起来

+ 避免**with**语句

- 性能问题

- 语义不明，调试困难

+ 数字转换成字符串

- 性能上来说：("" +) > String() > .toString() > **new** String()

+ 浮点数转换成整型

- parseInt()是用于将字符串转换成数字，而不是浮点数和整型之间的转换

- 应该使用Math.floor()或者Math.round()

+ 各种类型转换

- 如果定义了toString()方法来进行类型转换的话，推荐显式调用toString()

因为内部的操作在尝试所有可能性之后，会尝试对象的toString()方法尝试能否转化为String

所以直接调用这个方法效率会更高

+ 多个类型声明

- 在JavaScript中所有变量都可以使用单个**var**语句来声明,以减少整个脚本的执行时间

+ 插入迭代器

- 如**var** a=arr[i]; i++;前面两条语句可以写成**var** a=arr[i++]

+ 使用直接量

- **var** arr = **new** Array(); 替换为 **var** arr = [];

- **var** obj = **new** Object(); 替换为 **var** obj = {};

- **var** reg = **new** RegEXP(); 替换为 **var** reg = /.../;

- **var** obj2 = **new** O(); obj2.color = 'red';obj2.name = 'pig';

替换为 **var** obj2 = { color: 'red', name: 'pig' };

+ 重复使用的调用结果，事先保存到局部变量

- 避免多次取值的调用开销

+ 优化循环

- 减值迭代

大多数循环使用一个从0开始、增加到某个特定值的迭代器，

在很多情况下，从最大值开始，在循环中不断减值的迭代器更加高效。

- 简化终止条件

由于每次循环过程都会计算终止条件，

所以必须保证它尽可能快，也就是说避免属性查找或者其它的操作，

最好是将循环控制量保存到局部变量中，

也就是说对数组或列表对象的遍历时，提前将length保存到局部变量中，

避免在循环的每一步重复取值。

- 简化循环体

循环体是执行最多的，所以要确保其被最大限度的优化

- 使用后测试循环

在JavaScript中，我们可以使用**for**(;;),**while**(),**for**(**in**)三种循环，

事实上，这三种循环中**for**(**in**)的效率极差，因为他需要查询散列键。

**for**(;;)和**while**循环，**while**循环的效率要优于**for**(;;)，

可能是因为**for**(;;)结构的问题，需要经常跳转回去。

+ 尽量少使用eval函数

- 使用eval相当于在运行时再次调用解释引擎对内容进行运行，需要消耗大量时间；

- 而且使用Eval带来的安全性问题也是不容忽视的。

+ 条件分支

- 缩短否定检测

- 将条件分支，按可能性顺序从高到低排列

- 使用**switch**优于**if**：**switch**分支选择的效率高于**if**，在IE下尤为明显

- 使用三目运算符替代条件分支

+ 使用常量

- 重复值:任何在多处用到的值都应该抽取为一个常量

- 用户界面字符串:任何用于显示给用户的字符串，都应该抽取出来以方便国际化

- URLs:在Web应用中，资源位置很容易变更，所以推荐用一个公共地方存放所有的URL

- 任意可能会更改的值:每当你用到字面量值的时候，都要问一下自己这个值在未来是不是会变化，

如果答案是“是”，那么这个值就应该被提取出来作为一个常量。

+ 避免与null进行比较

- 如果值应为一个引用类型，使用**instanceof**操作符检查其构造函数；

- 如果值应为一个基本类型，作用**typeof**检查其类型，

- 如果是希望对象包含某个特定的方法名，则使用**typeof**操作符确保指定名字的方法存在于对象上

+ 避免全局量

- 全局变量应该全部字母大写，各单词之间用\_下划线来连接

- 尽可能避免全局变量和函数,尽量减少全局变量的使用；

因为在一个页面中包含的所有JavaScript都在同一个域中运行

所以如果你的代码中声明了全局变量或者全局函数的话，

后面的代码中载入的脚本文件中的同名变量和函数会覆盖掉（overwrite）你的

+ 缓存Dom选择与计算

- 每次Dom选择都要计算，缓存他。

- 缓存列表.length（每次.length都要计算，用一个变量保存这个值）

+ 尽量使用事件代理，避免批量绑定事件

- 尽量使用ID选择器，ID选择器是最快的。

+ TOUCH事件优化

- 使用touchstart、touchend代替click。但应注意Touch响应过快，易引发误操作

* 渲染优化

主要针对于移动端H5

+ HTML使用Viewport

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

+ 减少Dom节点

- Dom节点太多影响页面的渲染

+ 动画优化

- 尽量使用CSS3动画

- 合理使用requestAnimationFrame动画代替setTimeout

- 适当使用Canvas动画 5个元素以内使用css动画，5个以上使用Canvas动画（iOS8可使用webGL）

+ 高频事件优化

- 使用requestAnimationFrame监听帧变化，使得在正确的时间进行渲染

- 增加响应变化的时间间隔，减少重绘次数

+ GPU加速

- CSS中以下属性来触发GPU渲染，请合理使用。

CSS3 transitions、CSS3 3D transforms、Opacity、Canvas、WebGL、Video

- 但是过渡使用会引发手机过耗电增加。

**7、注释篇**

* css注释

+ 注释统一用'/\* \*/'（scss中也不要用'//'），具体参照右边的写法；

+ 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。

+ 尽量少写注释，命名规范基本上就可以看出该代码行是做什么用的

* js注释

+ 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。

+ 多行注释：

- 最少三行, '\*'后跟一个空格，具体参照右边的写法；

/\*

\* **one** space after '\*'

\*/

var **x** = 1*;*

+ 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

+ 文档注释：

- 各类标签**@param**, **@method**

/\*\*

\* **@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) {

...

}

- 在以下情况下使用：

· 所有常量

· 所有函数

· 所有类