前端规范

前端普适性规范

黄金定律

不管有多少人共同参与同一项目，一定要确保每一行代码都像是同一个人编写的。

项目命名

项目名全部采用小写方式，以中划线分隔，禁止驼峰式命名。比如：my-project-name

文件命名

文件命名参照项目命名规则。比如: error-report.html

有复数结构时，要采用复数命名法，比如： scripts, styles, images, data-models

文件名中只可由小写英文字母 a~z 、排序数字 0~9 或间隔符 - 组成，禁止包含特殊符号，比如空格、$ 等

为了醒目，某些说明文件的文件名，可以使用大写字母，比如: README, LICENSE

为更好的表达语义，文件名使用英文名词命名，或英文简写。

不允许命名带有广告等英文的单词，例如ad,adv,adver,advertising，防止该模块被浏览器当成垃圾广告过滤掉。任何文件的命名均如此。

文件常用命名:

- index.shtml 引导页&首页
- main.shtml 首页
- download.shtml 下载页面
- act.html 活动列表页面
- video.html 视频
- base.css 基本样式
- layout.css 框架布局

- module.css 模块样式
- global.css 全局样式
- font.css 字体样式
- index.css 首页样式
- print.css 打印样式

HTML 规范

语法

使用四个空格的缩进，这是保证代码在各种环境下显示一致的唯一方式。

嵌套的节点应该缩进（四个空格）。

在属性上，使用双引号，不要使用单引号。

不要在自动闭合标签结尾处使用斜线 / - HTML5 规范 指出他们是可选的。

```
<img src="images/logo.png" alt="Company">
```

不要忽略可选的关闭标签（例如， 和 </body>）。

HTML5 doctype

在每个 HTML 页面开头使用这个简单地 doctype 来启用标准模式，使其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

```
<!DOCTYPE html>
```

语言属性

```
<html lang="en">

</html>
```

字符编码

通过明确声明字符编码，能够确保浏览器快速并容易的判断页面内容的渲染方式。这样 做的好处是，可以避免在 HTML 中使用字符实体标记（character entity），从而全部与 文档编码一致（一般采用 UTF-8 编码）。

```
<meta charset="UTF-8">
```

IE 兼容模式

IE 支持通过特定的 标签来确定绘制当前页面所应该采用的 IE 版本。除非有强烈 的特殊需求，否则最好是设置为 edge mode，从而通知 IE 采用其所支持的最新的模 式。

```
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
```

响应式

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

引入 CSS 和 JavaScript

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

```
<!-- External CSS -->
<link rel="stylesheet" href="code-guide.css">

<!-- In-document CSS -->
<style>
    /* ... */
</style>

<!-- JavaScript -->
<script src="code-guide.js"></script>
```

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。

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。参考下面的示例:

```
<!-- Not so great -->
```

```
<span class="avatar">
    <img src="...">
</span>

<!-- Better -->
<img class="avatar" src="...">
```

属性顺序

HTML 属性应该按照特定的顺序出现以保证易读性。

1. class
2. id
3. name
4. data-*
5. src, for, type, href, value , max-length, max, min, pattern
6. placeholder, title, alt
7. aria-*, role
8. required, readonly, disabled

class 是为高可复用组件设计的，理论上他们应处在第一位。id 更加具体而且应该尽量少使用（例如, 页内书签），所以他们处在第二位。

Boolean 属性

Boolean 属性指不需要声明取值的属性。XHTML 需要每个属性声明取值，但是 HTML5 并不需要。

一个元素中 Boolean 属性的存在表示取值 true，不存在则表示取值 false。

简而言之，不要为 Boolean 属性添加取值。

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

CSS 规范

语法

使用四个空格的缩进，这是保证代码在各种环境下显示一致的唯一方式。

使用组合选择器时，保持每个独立的选择器占用一行。

为了代码的易读性，在每个声明的左括号前增加一个空格。

声明块的右括号应该另起一行。

每条声明 : 后应该插入一个空格。

每条声明应该只占用一行来保证错误报告更加准确。

所有声明应该以分号结尾。虽然最后一条声明后的分号是可选的，但是如果没有他，你的代码会更容易出错。

逗号分隔的取值，都应该在逗号之后增加一个空格。

所有的十六进制值都应该使用小写字母，例如 #fff。因为小写字母有更多样的外形，在浏览文档时，他们能够更轻松的被区分开来。

尽可能使用短的十六进制数值，例如使用 #fff 替代 #ffffff。

为选择器中的属性取值添加引号，例如 input[type="text"]。 他们只在某些情况下可有可无，所以都使用引号可以增加一致性。

不要为 0 指明单位，比如使用 margin: 0; 而不是 margin: 0px;。

```
/* Bad CSS */
.selector, .selector-secondary, .selector[type=text] {
    margin: 0px 0px 15px;
    background-color: rgba(0, 0, 0, 0.5);
    box-shadow: 0 1px 2px #CCC, inset 0 1px 0 #FFFFFF
}

/* Good CSS */
.selector,
.selector-secondary,
.selector[type="text"] {
    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

相关的属性声明应该以下面的顺序分组处理：

1. Positioning
2. Box model 盒模型
3. Typographic 排版
4. Visual 外观

Positioning 处在第一位，因为他可以使一个元素脱离正常文本流，并且覆盖盒模型相关的样式。盒模型紧跟其后，因为他决定了一个组件的大小和位置。

其他属性只在组件内部起作用或者不会对前面两种情况的结果产生影响，所以他们排在后面。 这块只是建议，会采用postcss处理兼容和排序等后置处理。

```
.declaration-order {
```

```
    /* Positioning */
    position: absolute;
    top: 0;
    right: 0;
    bottom: 0;
    left: 0;
    z-index: 100;

    /* Box-model */
    display: block;
    float: right;
    width: 100px;
    height: 100px;

    /* Typography */
    font: normal 13px "Helvetica Neue", sans-serif;
    line-height: 1.5;
    color: #333;
    text-align: center;

    /* Visual */
    background-color: #f5f5f5;
    border: 1px solid #e5e5e5;
    border-radius: 3px;

    /* Misc */
    opacity: 1;
}
```

禁止使用 @import

与<link>相比，@import较慢，增加额外的页面请求，并可能导致其他不可预见的问题。

```
<!-- Use link elements -->
<link rel="stylesheet" href="core.css">

<!-- Avoid @imports -->
<style>
    @import url("more.css");
</style>
```

媒体查询位置

尽量将媒体查询的位置靠近他们相关的规则。不要将他们一起放到一个独立的样式文件中，或者丢在文档的最底部。这样做只会让大家以后更容易忘记他们。这里是一个典型的案例。

```
.element { ... }
```

```
.element-avatar { ... }
.element-selected { ... }

@media (min-width: 480px) {
    .element { ...}
    .element-avatar { ... }
    .element-selected { ... }
}
```

单条声明的声明块

在一个声明块中只包含一条声明的情况下，为了易读性和快速编辑可以考虑移除其中的换行。所有包含多条声明的声明块应该分为多行。

这样做的关键因素是错误检测 - 例如，一个 CSS 验证程序显示你在 183 行有一个语法错误,如果是一个单条声明的行，那就是他了。在多个声明的情况下，你必须为哪里出错了费下脑子。

```
.span1 { width: 60px; }
.span2 { width: 140px; }
.span3 { width: 220px; }
```

属性简写

尽量不使用属性简写的方式，属性简写需要你必须显式设置所有取值。常见的属性简写滥用包括:

- padding
- margin
- font
- background -border -border-radius

大多数情况下，我们并不需要设置属性简写中包含的所有值。例如，HTML 头部只设置上下的 margin，所以如果需要，只设置这两个值。过度使用属性简写往往会导致更混乱的代码，其中包含不必要的重写和意想不到的副作用。

```
/* Bad example */
.element {
    margin: 0 0 10px;
    background: red;
    background: url("image.jpg");
    border-radius: 3px 3px 0 0;
}

/* Good example */
.element {
    margin-bottom: 10px;
    background-color: red;
```

```
    background-image: url("image.jpg");
    border-top-left-radius: 3px;
    border-top-right-radius: 3px;
}
```

代码注释

代码是由人来编写和维护的。保证你的代码是描述性的，包含好的注释，并且容易被他人理解。好的代码注释传达上下文和目标。不要简单地重申组件或者 class 名称。

class 命名

保持 class 命名为全小写，可以使用短划线（不要使用下划线和 camelCase 命名）。短划线应该作为相关类的自然间断。(例如，.btn 和 .btn-danger)。

避免过度使用简写。.btn 可以很好地描述 button，但是 .s 不能代表任何元素。

class 的命名应该尽量短，也要尽量明确。

使用有意义的名称；使用结构化或者作用目标相关，而不是抽象的名称。

命名时使用最近的父节点或者父 class 作为前缀。

使用 .js-* 来表示行为(相对于样式)，但是不要在 CSS 中包含这些 class。

选择器

使用 class 而不是通用元素标签来优化渲染性能。

避免在经常出现的组件中使用一些属性选择器 (例如，[class^="..."])。浏览器性能会受到这些情况的影响。

减少选择器的长度，每个组合选择器选择器的条目应该尽量控制在 3 个以内。

只在必要的情况下使用后代选择器 (例如，没有使用带前缀 classes 的情况).

代码组织

以组件为单位组织代码。

制定一个一致的注释层级结构。

使用一致的空白来分割代码块，这样做在查看大的文档时更有优势。

当使用多个 CSS 文件时，通过组件而不是页面来区分他们。页面会被重新排列，而组件移动就可以了。

编辑器配置

根据以下的设置来配置你的编辑器，将这些设置应用到项目的 .editorconfig 文件，来避免常见的代码不一致和丑陋的 diffs。

- 使用四个空格的缩进。

- 在保存时删除尾部的空白字符。
- 设置文件编码为 UTF-8。
- 在文件结尾添加一个空白行。

JS 规范

语法

使用四个空格的缩进，这是保证代码在各种环境下显示一致的唯一方式。

声明之后一律以分号结束， 不可以省略

完全避免 == != 的使用， 用严格比较条件 === !==

eval 非特殊情况， 禁用！！！

with 非特殊情况， 禁用！！！

单行长度，理论上不要超过80列，不过如果编辑器开启"自动换行"的话可以不考虑单行长度

接上一条，如果需要换行，存在操作符的情况，一定在操作符后换行，然后换的行缩进4个空格

这里要注意，如果是多次换行的话就没有必要继续缩进了，比如说下面这种就是最佳格式。

```
if (typeof qqfind === "undefined" ||
    typeof qqfind.cdnrejected === "undefined" ||
    qqfind.cdnrejected !== true) {
    url = "http://pub.idqqimg.com/qqfind/js/location4.js";
} else {
    url = "http://find.qq.com/js/location4.js";
}
```

空行

方法之间加

单行或多行注释前加

逻辑块之间加空行增加可读性

变量命名

标准变量采用驼峰标识

使用的ID的地方一定全大写

使用的URL的地方一定全大写, 比如说 reportURL

涉及Android的，一律大写第一个字母

涉及IOS的，一律大写

常量采用大写字母，下划线连接的方式

构造函数，大写第一个字母

```
var thisIsMyName;

var goodID;

var AndroidVersion;

var iOSVersion;

var MAX_COUNT = 10;

function Person(name) {
    this.name = name
}
```

null的使用场景

初始化可能以后分配对象值的变量

与一个可能或可能没有对象值的初始化变量进行比较

传入一个预期对象的函数

从预期对象的函数返回

不适合null的使用场景

不要使用null来测试是否提供参数

不要测试值为null的未初始化变量

undefined使用场景

永远不要直接使用undefined进行变量判断

使用字符串 "undefined" 对变量进行判断

```
// Bad
var person;
console.log(person === undefined);    //true

// Good
console.log(typeof person);    // "undefined"
```

对象字面量

```
// Bad
var team = new Team();
team.title = "AlloyTeam";
team.count = 25;

// Good
var team = {
    title: "AlloyTeam",
    count: 25
};
```

数组声明

```
// Bad
var colors = new Array("red", "green", "blue");
var numbers = new Array(1, 2, 3, 4);


// Good
var colors = [ "red", "green", "blue" ];
var numbers = [ 1, 2, 3, 4 ];
```

单行注释

双斜线后，必须跟注释内容保留一个空格

与下一行代码缩进保持一致

可位于一个代码行的末尾，双斜线距离分号四个空格

```
// Good
if (condition) {
    // if you made it here, then all security checks passed
    allowed();
}

var zhangsan = "zhangsan";    // 双斜线距离分号四个空格，双斜线后始终保留一个空格
```

多行注释格式

最少三行

前边留空一行

```
/**
 * 注释内容与星标前保留一个空格
 */
```

何时使用多行注释格式

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

想吐槽的产品逻辑, 合作同事

文档注释

各类标签 @param @method 等 参考 http://usejsdoc.org/

用于：方法、构造函数、对象

```
/**
 * here boy, look here , here is girl
 * @method lookGril
 * @param {Object} balabalabala
 * @return {Object} balabalabala
 */
```

括号对齐

标准示例 括号前后有空格，花括号起始不另换行，结尾新起一行

花括号必须要，即使内容只有一行

涉及 if for while do...while try...catch...finally 的地方都必须使用花括号，即使内容只有一行

if else 前后留有空格

```
if (condition) {
    doSomething();
} else {
    doSomethingElse();
}
```

switch

switch和括号之间有空格，case需要缩进，break之后跟下一个case中间留一个空白行

花括号必须要， 即使内容只有一行。

switch 的 falling through 一定要有注释特别说明，no default 的情况也需要注释特别说明况

```
switch (condition) {
    case "first":
        // code
        break;

    case "second":
        // code
        break;

    default:
    // code
}
```

for

普通for循环, 分号后留有一个空格， 判断条件等内的操作符两边不留空格

前置条件如果有多个，逗号后留一个空格

for-in 一定要有 hasOwnProperty 的判断， 否则 JSLint 或者 JSHint 都会有一个 warn

```
for (var i=0, len=values.length; i<len; i++) {
    process(values[i]);
}


var prop;

for (prop in object) {
    // 注意这里一定要有 hasOwnProperty 的判断， 否则 JSLint 或者 JSHint 都会有一个
warn !
    if (object.hasOwnProperty(prop)) {
        console.log("Property name is " + prop);
        console.log("Property value is " + object[prop]);
    }
}
```

变量声明

所有函数内变量声明放在函数内头部，只使用一个 var(多了JSLint报错)， 一个变量一行,for中的var除外

函数声明

一定先声明再使用， 不要利用 JavaScript engine的变量提升特性, 违反了这个规则 JSLint都会报 warn

function declaration 和 function expression 的不同，function expression 的（）前后必须有空格，而function declaration 在有函数名的时候不需要空格，没有函数名的时候需要空格。

函数调用括号前后不需要空格

立即执行函数的写法, 最外层必须包一层括号

"use strict" 决不允许全局使用， 必须放在函数的第一行， 可以用自执行函数包含大的代码段, 如果 "use strict" 在函数外使用， JSLint均会报错

```
function doSomething(item) {
    // do something
}

var doSomething = function (item) {
    // do something
}


// Good
doSomething(item);

// Bad: Looks like a block statement
doSomething (item);

// Good
(function() {
    "use strict";

    function doSomething() {
        // code
    }
})();
```