文档名称：	**桑格前端开发编码规范**
负责人：	袁娟娟
文档版本编号：	V1.0
文档版本日期：	2016.06.12

目录

一、前言 1

1.1 规范背景 1

1.2 规范目的 1

1.3 基本准则 1

2.1 命名 2

三、Html 编码规范 6

3.1 语义化标签 6

3.2 书写规范 6

3.3 CSS 和 Javascript 引入 6

四、Javascript 编码规范 7

4.1 命名 7

4.2 格式 8

4.3 注释 10

4.4 其它 12

一、前言

1.1 规范背景

国内基本各大互联网公司的前端都有自己的开发规范，但总的宗旨基本都是：代码简、易维护、性能高。对于一个大型项目经常会多人协作，这时必须要有一个好的规范才能顺利便捷地进行下去。

1.2 规范目的

为提高团队协作效率，便于后台人员添加功能及前端后期优化维护，输出高质量的文档,同是为网站有一个更好的前端架构，网站的发展及未来打好一个基础。

1.3 基本准则

符合 web 标准， 语义化 html，结构表现行为分离，兼容性优良； 页面性能方面，代码要求简洁明了有序， 尽可能的减小服务器负载，保证最快的解析速度。

二、CSS 编码规范

2.1 命名

2.1.1 格式及规范

1.命名要有意义；2.要用英文命名，严禁用汉语拼音；3.能用常见的缩写就不要用完整单词来命名； 4.命名的词用连字符连接；5.连字符用中划线

示例：

```
1.  .div1            ------------------------->  .first-block
2.  .fenxi           ------------------------->  . analyze
3.  #navigation      ------------------------->  #nav
4.  .demoimage       ------------------------->  . demo-image
5.  .error_status    ------------------------->  . error-status
```

[注：目前的项目 css 中的命名沿用了以前的下划线方式，为保证统一，现有的项目仍然沿用下划线方式，新项目必须用中划线]

2.1.2 参考命名

（1）页面结构

容器: container

页头：header

内容：content/container

页面主体：main

页尾：footer

导航：nav

侧栏：sidebar

栏目：column

页面外围控制整体佈局宽度：wrapper

左右中：left right center

（2）导航

导航：nav

主导航：mainnav

子导航：subnav

顶导航：topnav

边导航：sidebar

左导航：leftsidebar

右导航：rightsidebar

菜单：menu

子菜单：submenu
标题: title
摘要: summary

（3）功能

标志：logo
广告：banner
登陆：login
登录条：loginbar
注册：register
搜索：search
功能区：shop
标题：title
加入：joinus
状态：status
按钮：btn
滚动：scroll
标籤页：tab
文章列表：list
提示信息：msg
当前的: current
小技巧：tips
图标: icon
注释：note
指南：guild
服务：service
热点：hot
新闻：news
下载：download
投票：vote
合作伙伴：partner
友情链接：link
版权：copyright

2.2 顺序

（1）定位属性(position, top, right, z-index, display, float 等)
（2）自身属性(width, height, padding, margin)
（3）文本属性(font, line-height, letter-spacing, color- text-align 等)
（4）其它属性(background, borde, animation, transition 等)

2.3 选择器

（1）ID 是唯一的，所以没必要多此一举使用标签类型选择器来做限定；

（2）不同的标签类型尽可能不用相同的类名，所以也不必使用标签类型选择器来做限定。

示例：

```
（1）ul#menus        ------------------------->   #menus
（2）div.info        ------------------------->   .info
```

2.4 属性及值的简写

（1）padding,margin,font,border,background 等属性的简写；

（2）去掉小数点前的“0”;3.16 进制颜色代码缩写；4.0 后面不跟单位。

不规范	规范
border-top-style: none;	border-top: 0;
font-family: palatino, georgia, serif; font-size: 100%; line-height: 1.6;	font: 100%/1.6 palatino, georgia, serif;
padding-bottom: 2em; padding-left: 1em; padding-right: 1em; padding-top: 0;	padding: 0 1em 2em;
background-color:#00FF00; background-image:url('bgimage.gif'); background-repeat: no-repeat; background-attachment:fixed; background-position:top;	background: #00FF00 url('bgimage.gif') no-repeat fixed top;
margin: 0.8em;	margin: .8em;
color:#0033cc	color:#03c;
margin:0px auto;	margin:0 auto;

2.5 为特定浏览器单独写一个 CSS 文件

碰到浏览器兼容性方面的问题，首先考虑是否能换一种解决方案，若实在没有，则为特定浏览器单独写一个 css 文件，避免兼容代码与常规代码混合，以便于以后的维护。

示例：

```
<!--[if IE 7]>
<link rel="stylesheet" type="text/css" href="css/ie7.css" />
<![endif]-->
```

2.6 多组合少继承

把 css 定义中的固定部分和可变部分分开定义，使得代码达到最大程度的重用，这样的结果是增加了元素上添加的 css 类个数，但是提高了代码的维护性和可读性。

示例：

```
<span class=”btn btn_red btn_big”></span>
```

三、Html 编码规范

3.1 语义化标签

如标题根据重要性用 h*(同一页面只能有一个 h1)，段落标记用 p，列表用 ul，内联元素中不可嵌套块级元素；

下面是常见的语义标签：

p - 段落
h1,h2,h3,h4,h5,h6 - 层级标题
strong,em - 强调
ins - 插入
del - 删除
abbr - 缩写
code - 代码标识
cite - 引述来源作品的标题
q - 引用
blockquote - 一段或长篇引用
ul - 无序列表
ol - 有序列表
dl,dt,dd - 定义列表

3.2 书写规范

（1）为 JavaScript 预留钩子的命名，请以 js_ 起始，比如: js_hide, js_show；

（2）重要图片必须加上 alt 属性；给重要的元素和截断的元素加上 title;

（3）给区块代码及重要功能(比如循环)加上注释，方便后台添加功能;

（4）特殊符号使用：尽可能使用代码替代：比如 <(<) & >(>) & 空格( ) & »(») 等等;

3.3 CSS 和 Javascript 引入

（1）引入 CSS 时必须指明 rel="stylesheet"。

示例：

```
<link rel="stylesheet" href="page.css">
```

[建议] 引入 CSS 和 JavaScript 时无须指明 type 属性。text/css 和 text/javascript 是 type 的默认值。

（2）<script src="filename.js">应尽量放到 body 的后面。这样可以减少因为载入 script 而造成其他页面内容载入也被延迟的问题。

四、Javascript 编码规范

4.1 命名

（1）变量、函数、函数的参数、类的方法/属性、命名空间均使用 Camel 命名法。

示例：

```
var loadingModules = {};
function stringFormat(source) {
}
function hear(theBells) {
}
function TextNode(value, engine) {
    this.value = value;
    this.engine = engine;
}
TextNode.prototype.clone = function () {
    return this;
};
equipments.heavyWeapons = {};
```

（2）常量使用全部字母大写，单词间下划线分隔的命名方式。

示例：

```
var HTML_ENTITY = {};
```

（3）类使用 Pascal 命名法。

示例：

（4）枚举变量使用 Pascal 命名法，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。

示例：

```
var TargetState = {
    READING: 1,
    READED: 2,
    APPLIED: 3,
    READY: 4
};
```

（5）类名使用名词；函数名使用动宾短语；Promise 对象用动宾短语的进行时。

（6）boolean 类型的变量使用 is 或 has 开头。

示例：

```
var isReady = false;
var hasMoreCommands = false;
```

4.2 格式

（1）缩进的单位为四个空格。避免使用 Tab 键来缩进。

[注: 现在很多编辑工具可以将 Tab 键设置为 4 个空格。]

（2）避免每行超过 80 个字符。当一条语句一行写不下时,请考虑折行。在运算符号,最好是逗号后换行。在运算符后换行可以减少因为复制粘贴产生的错误被分号掩盖的几率。

（3）函数名与左括号之间不应该有空格，这能帮助区分关键字和函数调用。

[注: 如果函数是匿名函数,则应有一个空格。如果省略了空格,否则会让人感觉函数名叫作 function。]

（4）函数名后的右括号与开始程序体的左大括号之间应插入一个空格。右大括号与声明函数的那一行代码头部对齐。

示例：

```
function outer(c, d) {
    var e = c * d;
    function inner(a, b) {
        return (e * a) + b;
    }
    return inner(0, 1);
}
```

（5）所有的二元操作符,除了“.”、“(”和“[”，应使用空格将其与操作数隔开。

（6）一元操作符与其操作数之间不应有空格,除非操作符是个单词,比如 typeof。

（7）每个在控制部分,比如 for 语句中的; (分号)后须跟一个空格。

（8）每个“,”后应跟一个空格。

（9）每个独立语句结束后必须换行；

（10）每行不得超过 120 个字符。

（11）运算符处换行时，运算符必须在新行的行首。

示例：

```
if (user.isAuthenticated()
    && user.isInRole('admin')
    && user.hasAuthority('add-admin')
    || user.hasAuthority('delete-admin')
) {
}
```

（12）同行为或逻辑的语句集，使用空行隔开，更易阅读。

示例：

```
function setStyle(element, property, value) {
    if (element == null) {
        return;
    }

    element.style[property] = value;
}
```

（13）于 if...else...、try...catch...finally 等语句，推荐使用在}号后添加一个换行的风格，使代码层次结构更清晰，阅读性更好。

示例：

```
if (condition) {
    // some statements;
}
else {
    // some statements;
}
try {
    // some statements;
}
catch (ex) {
    // some statements;
}
```

（14）函数定义结束不允许添加分号；如果是函数表达式，必须添加分号。

示例：

```
function funcName() {
}
var funcName = function () {
};
```

（15）使用对象字面量{}创建新 Object。

（16）使用数组字面量[]创建新数组，除非想要创建的是指定长度的数组。

（17）空函数不使用 new Function()的形式。

示例：

```
var EMPTY_FUNCTION = function () {};
```

（18）对象创建时，如果一个对象的所有属性均可以不添加引号，则所有属性不得添加引号。如果任何一个属性需要添加引号，则所有属性必须添加引号。

示例：

```
var info = {
    name: 'someone',
    age: 28
};
var info = {
    'name': 'someone',
    'age': 28,
    'more-info': '...'
};
```

4.3 注释

不要吝啬注释；注释应书写良好且清晰明了；及时地更新注释；注释必须有意义；使用单行注释、块注释用于注释正式文档和无用代码；

4.3.1 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。

4.3.2 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。

4.3.3 文档化注释

为了便于代码阅读和自文档化，以下内容必须包含以/**...*/形式的块注释中，且文档注释前必须空一行。

1. 文件
2. namespace
3. 类
4. 函数或方法
5. 类属性
6. 事件
7. 全局变量
8. 常量
9. AMD 模块

[建议] 自文档化的文档说明 what，而不是 how。

4.3.4 文件注释

[强制] 文件顶部必须包含文件注释，用 **@file** 标识文件说明。

示例：

```
/**
     * @file Describe the file
*/
```

[解释]

开发者信息能够体现开发人员对文件的贡献，并且能够让遇到问题或希望了解相关信息的人找到维护人。通常情况文件在被创建时标识的是创建者。随着项目的进展，越来越多的人加入，参与这个文件的开发，新的作者应该被加入@author 标识。

@author 标识具有多人时，原则是按照责任进行排序。通常的说就是如果有问题，就是找第一个人应该比找第二个人有效。比如文件的创建者由于各种原因，模块移交给了其他人或其他团队，后来因为新增需求，其他人在新增代码时，添加@author 标识应该把自己的名字添加在创建人的前面。

@author 中的名字不允许被删除。任何劳动成果都应该被尊重。

业务项目中，一个文件可能被多人频繁修改，并且每个人的维护时间都可能不会很长，不建议为文件增加@author 标识。通过版本控制系统追踪变更，按业务逻辑单元确定模块的维护责任人，通过文档与 wiki 跟踪和查询，是更好的责任管理方式。

对于业务逻辑无关的技术型基础项目，特别是开源的公共项目，应使用@author 标识。

示例：

```
/**
 * @file Describe the file
 * @author author-name(mail-name@domain.com)
 *         author-name2(mail-name2@domain.com)
 */
```

4.2.5 细节注释

对于内部实现、不容易理解的逻辑说明、摘要信息等，我们可能需要编写细节注释。细节注释遵循单行注释的格式。

（1）说明必须换行时，每行是一个单行注释的起始。

示例：

```
function foo(p1, p2) {
    // 这里对具体内部逻辑进行说明
    // 说明太长需要换行
    for (...) {
        ....
    }
```

（2）有时我们会使用一些特殊标记进行说明。下面列举了一些常用标记：

示例：

```
function foo(p1, p2) {
    //TODO: 有功能待实现。此时需要对将要实现的功能进行简单说明。
    //FIXME: 该处代码运行没问题，但可能由于时间赶或者其他原因，需要修正。此时需要对如何修正进行简单说明。
    //HACK: 为修正某些问题而写的不太好或者使用了某些诡异手段的代码。此时需要对思路或诡异手段进行描述。
    //XXX: 该处存在陷阱。此时需要对陷阱进行描述。
    for (...) {
        ....
    }
```

4.4 其它

（1）所有的变量必须在使用前进行声明。将 var 语句放在函数的首部。最好把每个变量的声明语句单独放到一行,并加上注释说明。尽量减少全局变量的使用。不要让局部变量覆盖全局变量。

示例：

```
var currentEntry; // 当前选择项
var level;        // 缩进程度
var size;         // 表格大小
```

（2）尽量避免使用直接 eval。

[解释] 直接 eval，指的是以函数方式调用 eval 的调用方法。直接 eval 调用执行代码的作用域为本地作用域，应当避免。

[注：如果有特殊情况需要使用直接 eval，需在代码中用详细的注释说明为何必须使用直接 eval，不能使用其它动态执行代码的方式，同时需要其他资深工程师进行 Code Review。]

（3）尽量不要使用 with。

[解释] 使用 with 可能会增加代码的复杂度，不利于阅读和管理；也会对性能有影响。大多数使用 with 的场景都能使用其他方式较好的替代。所以，尽量不要使用 with。

（4）尽量减少 delete 的使用。

[解释] 如果没有特别的需求，减少或避免使用 delete。delete 的使用会破坏部分 JavaScript 引擎的性能优化。