# 前端代码规范V1.0

## 前言

前端开发是直接面对用户，给到用户最直接交互体验的一个开发环节。开发前端的脚本语言十分灵活，每个人都有每个人的开发风格，并且javascript是单线程编程语言，程序的运行效率在前端开发是比较重要的，所以对前端开发人员应有更高的编程要求。而现当下前端框架种类繁多，前端开发从刀耕火种时代发展到了一个百家争鸣的时代，这就更加造就了眼下不受约束的情况下，不同的前端工程师合作完成的项目风格不统一，导致一个周期长而大的项目难以维护甚至需要重构的后果，消耗大量的开发成本以及维护成本。在此前提下，定制前端开发代码规范成为了一件非常必要的事情。根据个人项目经验和项目重构经验总结以及参考百度前端开发规范，编写了这篇前端代码标准，意在统一大家的开发风格，提高前端代码质量，降低维护成本，后续需要大家一起来完善。

目录

[一、 命名规约 4](#_Toc20229)

[1、 文件命名 4](#_Toc32070)

[2、 Js代码命名 5](#_Toc12933)

[3、 css命名 5](#_Toc27982)

[4、 HTML命名 6](#_Toc1043)

[二、 代码规约 7](#_Toc9817)

[1、 js代码风格 7](#_Toc6136)

[1.1、缩进和空格 7](#_Toc21819)

[1.2、换行 7](#_Toc21698)

[1.3、语句 8](#_Toc10026)

[2、js语言特性 8](#_Toc6188)

[2.1、条件 8](#_Toc15083)

[2.2、变量 9](#_Toc9740)

[2.3、循环 9](#_Toc3193)

[2.4、类型 9](#_Toc15344)

[2.5、字符串 9](#_Toc14019)

[2.6、对象 10](#_Toc22100)

[2.7、数组 10](#_Toc1138)

[2.8、函数 11](#_Toc23768)

[2.9、面向对象 11](#_Toc30798)

[3.0、动态特性 12](#_Toc19277)

[3、浏览器环境 12](#_Toc11358)

[3.1、模块化 12](#_Toc13744)

[3.2、DOM 13](#_Toc9248)

[4、css代码风格 14](#_Toc18892)

[5、 css特性 14](#_Toc5442)

[5.1、通用 14](#_Toc3878)

[5.2、清除浮动 15](#_Toc6040)

[5.3、!important 15](#_Toc15127)

[5.4、z-index 15](#_Toc31879)

[5.5、响应式 15](#_Toc19740)

[5.6、浏览器兼容性 16](#_Toc4768)

[6、 html代码风格 16](#_Toc31357)

[三、 注释规约 17](#_Toc16150)

[1、 单行注释 17](#_Toc19563)

[2、 多行/块注释 17](#_Toc25598)

[3、 文档化注释 17](#_Toc11200)

[3.1、类型定义 17](#_Toc377)

[3.2、文件注释 17](#_Toc12291)

[3.3、命名空间注释 18](#_Toc12596)

[3.4、类注释 18](#_Toc29544)

[3.5、函数/方法注释 18](#_Toc22243)

[3.6、事件注释 18](#_Toc31708)

[3.7、常量注释 19](#_Toc30116)

[3.8、复杂类型注释 19](#_Toc2397)

[3.9、AMD模块注释 19](#_Toc22896)

[4.0、细节注释 19](#_Toc23600)

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 作者 | 维护团队 | 更新日期 |
| 1.0 | 张雪连 | 高新兴研发前端团队 | 2018-02-01 |

## 命名规约

### 文件命名

1. 【强制】名称全部以小写英文(有通用缩写可用英文缩写)命名，多个单词以下划线\_连接，如：case\_list，此命名规范同用于文件夹命名，html、js、css、img命名；
2. 【推荐】若项目按文件类型将css、js、html分开，则对于同一模块的文件夹命名一致，如：html/case\_list/case\_list.html、css/case\_list/case\_list.css、js/case\_list/case\_list.js；
3. 【推荐】若项目按模块将js与html文件放于同一文件夹下，并且同类型文件具有明显分层功能，如angular项目的controller文件，视图view文件，可在文件命名后加标识性命名，如system\_news.ctrl.js、system\_news.tpl.html。

### Js代码命名

1. 【强制】**变量命名**：变量 使用 Camel命名法，如var loadingModules = {}；
2. 【强制】**常量命名**： 使用 全部字母大写，单词间下划线分隔 的命名方式，如var HTML\_ENTITY = {}；
3. 【强制】**函数以及函数参数命名**：使用 Camel命名法，如：function stringFormat(theBells) {}；
4. 【强制】**类命名：**使用 Pascal命名法，并且使用名词，如：function TextNode(options) {}，类属性/方法命名：使用 Camel命名法；
5. 【强制】**类的方法/属性：**使用 Camel命名法；
6. 【强制】**命名空间**：使用 Camel命名法；
7. 【强制】**枚举变量：** 枚举以及枚举的属性使用 Camel命名法；
8. 【强制】**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，如：function XMLParser() {}；function insertHTML(element, html) {}；var httpRequest = new HTTPRequest()；
9. 【强制】**类名：**使用名词，如function Engine(options) {}；
10. 【建议】**函数名：** 使用动宾短语，如function getStyle(element) {}；
11. 【建议】**boolean：**类型的变量使用is或has开头，如：var isReady = false;var hasMoreCommands = false；
12. 【建议】**Promis对象：**用动宾短语的进行时表达，如：var loadingData = ajax.get('url');loadingData.then(callback)。

### css命名

1. 【强制】**类选择器名：**名称全部以小写英文(有通用缩写可用英文缩写)命名，多个单词以-连接，如：.main-container{width:100%;height:100%}；
2. 【强制】**ID选择器名：**使用 Camel命名法，如#mainContainer{max-width:1024px}；
3. 【强制】禁止为了 hook 脚本，创建无样式信息的 class；
4. 【建议】**ID选择器：**ID在DOM节点中是唯一的，一般不建议使用ID选择器来写样式，因为不可复用，如果在文档中出现多个ID，则浏览器在渲染时查找文档只取第一个；
5. 【建议】**id/class 命名：**在避免冲突并描述清楚的前提下尽可能简短；
6. 【建议】同一页面，应避免使用相同的 name 与 id，IE 浏览器会混淆元素的 id 和 name 属性， document.getElementById 可能获得不期望的元素。所以在对元素的 id 与 name 属性的命名需要非常小心。

### HTML命名

1. 【强制】**标签名：**必须使用小写字母；
2. 【强制】 对于无需自闭合的标签，不允许自闭合，如：input、br、img、hr等；
3. 【强制】对 HTML5 中规定允许省略的闭合标签，不允许省略闭合标签，如：<ul><li></li></ul>;如果这么写<ul><li></ul>，IE8上不自动闭合；
4. 【强制】标签使用必须符合标签嵌套规则，如：比如 div 不得置于 p 中，tbody 必须置于 table 中；
5. 【建议】HTML 标签的使用应该遵循标签的语义；如：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......
6. 【强制】**属性名：**属性名必须使用小写字母；
7. 【强制】使用 HTML5 的 doctype 来启用标准模式，建议使用大写的 DOCTYPE，如：<!DOCTYPE html>；
8. 【建议】启用 IE Edge 模式；如：<meta http-equiv = "X-UA-Compatible" content = "IE=Edge,chrome=1" >；
9. 【建议】在 html 标签上设置正确的 lang 属性，如：<html lang="zh-CN">。

## 代码规约

### js代码风格

#### 1.1、缩进和空格

1. 【强制】使用tab字符做为一个缩进层级；
2. 【强制】switch 下的 case 和 default 必须增加一个缩进层级；
3. 【强制】二元运算符两侧必须有一个空格，一元运算符与操作对象之间不允许有空格，如：var a = !arr.length; a++; a = b + c;
4. 【强制】用作代码块起始的左花括号 { 前必须有一个空格，如：function test() {}；
5. 【强制】if / else / for / while / function / switch / do / try / catch / finally 关键字后，必须有一个空格；
6. 【强制】在对象创建时，属性中的 : 之后必须有空格，: 之前不允许有空格，如：var obj = { a: 1};
7. 【强制】函数声明、具名函数表达式、函数调用中，函数名和 ( 之间不允许有空格；
8. 【强制】, 和 ; 前不允许有空格；
9. 【强制】在函数调用、函数声明、括号表达式、属性访问、if / for / while / switch / catch 等语句中，() 和 [] 内紧贴括号部分不允许有空格，如：callFunc(param1, param2, param3);
10. 【强制】单行声明的数组与对象，如果包含元素，{} 和 [] 内紧贴括号部分不允许包含空格，如：var arr2 = [1, 2, 3];
11. 【强制】行尾不得有多余的空格。

#### 1.2、换行

1. 【强制】每个独立语句结束后必须换行；
2. 【强制】每行不得超过 120 个字符，超长的不可分割的代码允许例外，比如复杂的正则表达式，长字符串不在例外之列；
3. 【强制】运算符处换行时，运算符必须在新行的行首；
4. 【强制】在函数声明、函数表达式、函数调用、对象创建、数组创建、for语句等场景中，不允许在 , 或 ; 前换行；
5. 【建议】不同行为或逻辑的语句集，使用空行隔开，更易阅读；
6. 【建议】在语句的行长度超过 120 时，根据逻辑条件合理缩进；
7. 【建议】 对于 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；
8. 【强制】不得省略语句结束的分号。

#### 1.3、语句

1. 【强制】在 if / else / for / do / while 语句中，即使只有一行，也不得省略块 {...}；
2. 【强制】函数定义结束不允许添加分号；
3. 【强制】IIFE 必须在函数表达式外添加 (，非 IIFE 不得在函数表达式外添加 (，如：var task = (function () {/\*your code\*/})()。

### 2、js语言特性

#### 2.1、条件

1. 【强制】在 Equality Expression 中使用类型严格的 ===。仅当判断 null 或 undefined 时，允许使用 == null，使用 === 可以避免等于判断中隐式的类型转换；
2. 【建议】尽可能使用简洁的表达式，如：使用if(!name)代替if(name === ‘’)；
3. 【建议】对于相同变量或表达式的多值条件，用 switch 代替 if；
4. 【建议】如果函数或全局中的 else 块后没有任何语句，可以删除 else。

#### 2.2、变量

1. 【强制】变量在使用前必须通过 var 定义，一般情况下，不允许创建全局变量，不允许污染环境，不允许污染原生对象；
2. 【强制】每个 var 只能声明一个变量；
3. 【强制】变量必须**即用即声明**，不得在函数或其它形式的代码块起始位置统一声明所有变量。

#### 2.3、循环

1. 【建议】不要在循环体中包含函数表达式，事先将函数提取到循环体外；
2. 【建议】对循环内多次使用的不变值，在循环外用变量缓存；
3. 【建议】对有序集合进行遍历时，缓存 length，虽然现代浏览器都对数组长度进行了缓存，但对于一些宿主对象和老旧浏览器的数组对象，在每次 length 访问时会动态计算元素个数，此时缓存 length 能有效提高程序性能；
4. 【建议】对有序集合进行顺序无关的遍历时，使用逆序遍历，逆序遍历可以节省变量，代码比较优化。

#### 2.4、类型

1. 【建议】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；
2. 【建议】转换成 string 时，使用 + ''，如：num + ’’；
3. 【建议】转换成 number 时，通常使用 + ，如：+ str；
4. 【强制】使用 parseInt 时，必须指定进制，如：parseInt(str, 10)；
5. 【建议】转换成 boolean 时，使用 !!；
6. 【建议】number 去除小数点，使用 Math.floor / Math.round / Math.ceil，不使用 parseInt。

#### 2.5、字符串

1. 【强制】**字符串开头和结束使用单引号 '**，因为实际使用中，字符串经常用来拼接 HTML，为方便 HTML 中包含双引号而不需要转义写法；
2. 【建议】**使用数组或 + 拼接字符串**，数组拼接用join()，使用 + 拼接字符串，如果拼接的全部是 StringLiteral，压缩工具可以对其进行自动合并的优化。所以，静态字符串建议使用 + 拼接；在现代浏览器下，使用 + 拼接字符串，性能较数组的方式要高；如需要兼顾老旧浏览器，应尽量使用数组拼接字符串；
3. 【建议】复杂的数据到视图字符串的转换过程，选用一种模板引擎，模板引擎有如下几种：artTemplate、dot.js、etpl、handlebars、hogon、nunjucks等，推荐使用artTemplate和etpl。

#### 2.6、对象

1. 【强制】使用对象字面量 {} 创建新 Object；
2. 【强制】 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；
3. 【强制】对象创建时，如果任何一个属性需要添加引号，则所有属性必须添加单引号’，如果属性不符合 Identifier 和 NumberLiteral 的形式，就需要以 StringLiteral 的形式提供；
4. 【强制】不允许修改和扩展任何原生对象和宿主对象的原型；
5. 【建议】属性访问时，尽量使用 . ，属性名符合 Identifier 的要求，就可以通过 . 来访问，否则就只能通过 [expr] 方式访问；
6. 【建议】for in 遍历对象时, 使用 hasOwnProperty 过滤掉原型中的属性。

#### 2.7、数组

1. 【强制】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；
2. 【强制】遍历数组不使用 for in，数组对象可能存在数字以外的属性, 这种情况下 for in 不会得到正确结果；
3. 【建议】不因为性能的原因自己实现数组排序功能，尽量使用数组的 sort 方法；
4. 【建议】清空数组使用 .length = 0。

#### 2.8、函数

1. 【建议】一个函数的长度控制在 50 行以内，将过多的逻辑单元混在一个大函数中，易导致难以维护；一个清晰易懂的函数应该完成单一的逻辑单元。复杂的操作应进一步抽取，通过函数的调用来体现流程；特定算法等不可分割的逻辑允许例外；
2. 【建议】一个函数的参数控制在 6 个以内；
3. 【建议】通过 options 参数传递非数据输入型参数，可参考ocx2.0的封装方法；
4. 【建议】在适当的时候将闭包内大对象置为 null；
5. 【建议】使用 IIFE 避免 Lift 效应；
6. 【建议】空函数不使用 new Function() 的形式，而应采用：var emptyFunction = function () {};
7. 【建议】对于性能有高要求的场合，建议存在一个空函数的常量，供多处使用共享。

#### 2.9、面向对象

1. 【强制】类的继承方案，实现时需要修正 constructor，通常使用其他 library 的类继承方案都会进行 constructor 修正。如果是自己实现的类继承方案，需要进行 constructor 修正；
2. 【建议】声明类时，保证 constructor 的正确性；
3. 【建议】属性在构造函数中声明，方法在原型中声明，因为原型对象的成员被所有实例共享，能节约内存占用；所以编码时我们应该遵守这样的原则：原型对象包含程序不会修改的成员，如方法函数或配置项；
4. 【强制】自定义事件的事件名必须全小写，因为在 JavaScript 广泛应用的浏览器环境，绝大多数 DOM 事件名称都是全小写的；为了遵循大多数 JavaScript 开发者的习惯，在设计自定义事件时，事件名也应该全小写；
5. 【强制】自定义事件只能有一个 event 参数。如果事件需要传递较多信息，应仔细设计事件对象；
6. 【建议】设计自定义事件时，应考虑禁止默认行为。

#### 3.0、动态特性

1. 【强制】避免使用直接 eval 函数，因为直接 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；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；
2. 【建议】使用 new Function 执行动态代码；
3. 【建议】尽量不要使用 with；
4. 【建议】减少 delete 的使用；
5. 【建议】处理 delete 可能产生的异常；
6. 【建议】避免修改外部传入的对象；
7. 【建议】具备强类型的设计。

### 3、浏览器环境

#### 3.1、模块化

1. 【强制】使用 AMD 作为模块定义，建议使用官方实现的requirejs；
2. 【建议】定义模块时不要指明 id 和 dependencies，因为在 AMD 的设计思想里，模块名称是和所在路径相关的，匿名的模块更利于封包和迁移；模块依赖应在模块定义内部通过 local require 引用；所以，推荐使用 define(factory) 的形式进行模块定义；
3. 【建议】使用 return 来返回模块定义；
4. 【强制】全局运行环境中，require 必须以 async require 形式调用，因为模块的加载过程是异步的，同步调用并无法保证得到正确的结果；
5. 【强制】模块定义中只允许使用 local require，不允许使用 global require；
6. 【强制】Package在实现时，内部模块的 require 必须使用 relative id；
7. 【建议】不会被调用的依赖模块，在 factory 开始处统一 require；
8. 【强制】模块 id 必须符合标准，模块 id 必须符合以下约束条件：
9. 类型为 string，并且是由 / 分割的一系列 terms 来组成。例如：this/is/a/module。
10. term 应该符合 [a-zA-Z0-9\_-]+ 规则。
11. 不应该有 .js 后缀。
12. 跟文件的路径保持一致。

#### 3.2、DOM

1. 【建议】对于单个元素，尽可能使用 document.getElementById 获取，避免使用document.all；
2. 【建议】对于多个元素的集合，尽可能使用 context.getElementsByTagName 获取。其中 context 可以为 document 或其他元素。指定 tagName 参数为 \* 可以获得所有子元素；
3. 【建议】遍历元素集合时，尽量缓存集合长度。如需多次操作同一集合，则应将集合转为数组；
4. 【建议】获取元素的直接子元素时使用 children。避免使用childNodes，除非预期是需要包含文本、注释和属性类型的节点；
5. 【建议】尽量减少 DOM 操作，因为DOM 操作也是非常耗时的一种操作，减少 DOM 操作有助于提高性能。

### 4、css代码风格

1. 【建议】CSS 文件使用无 BOM 的 UTF-8 编码；
2. 【强制】使用 tab 字符做为一个缩进层级，不允许空格来缩进；
3. 【强制】选择器 与 { 之间不包含空格；
4. 【强制】属性名 与之后的 : 之间不允许包含空格， : 与 属性值 之间必须包含空格；
5. 【强制】列表型属性值书写在单行时，逗号后必须跟一个空格；
6. 【强制】每行不得超过 120 个字符，除非单行不可分割；
7. 【建议】对于超长的样式，在样式值的 空格 处或 , 后换行，建议按逻辑分组；
8. 【强制】当一个 rule 包含多个 selector 时，每个选择器声明必须独占一行；
9. 【强制】>、+、~ 选择器的两边各保留一个空格，冒号不保留空格；
10. 【强制】属性选择器中的值必须用双引号包围；
11. 【强制】属性定义必须另起一行；
12. 【强制】属性定义后必须以分号结尾。

### css特性

#### 5.1、通用

1. 【强制】如无必要，不得为 id、class 选择器添加类型选择器进行限定；
2. 【建议】在可以使用缩写的情况下，尽量使用属性缩写；
3. 【建议】使用 border / margin / padding 等缩写时，应注意隐含值对实际数值的影响，确实需要设置多个方向的值时才使用缩写；
4. 【建议】同一 rule set 下的属性在书写时，应按功能进行分组，并以 Formatting Model（布局方式、位置） > Box Model（尺寸） > Typographic（文本相关） > Visual（视觉效果） 的顺序书写，以提高代码的可读性。

#### 5.2、清除浮动

1. 【建议】 当元素需要撑起高度以包含内部的浮动元素时，通过对伪类设置 clear 或触发 BFC 的方式进行 clearfix，尽量不使用增加空标签的方式，触发 BFC 的方式主要有以下几种：float 非 none、position 非 static、overflow 非 visible，对已经触发 BFC 的元素不需要再进行 clearfix。

#### 5.3、!important

1. 【建议】尽量不使用 !important 声明；
2. 【建议】当需要强制指定样式且不允许任何场景覆盖时，通过标签内联和 !important 定义样式，css权重计算规则：

第一等：代表内联样式，如: style=””，权值为1000；

第二等：代表ID选择器，如：#content，权值为0100；

第三等：代表类，伪类和属性选择器，如.content，权值为0010；

第四等：代表类型选择器和伪元素选择器，如div p，权值为0001；

通配符、子选择器、相邻选择器等的。如\*、>、+,权值为0000；

继承的样式没有权值；

#### 5.4、z-index

1. 【建议】将 z-index 进行分层，对文档流外绝对定位元素的视觉层级关系进行管理；
2. 【建议】在可控环境下，期望显示在最上层的元素，z-index 指定为 999999；
3. 【建议】在第三方环境下，期望显示在最上层的元素，通过标签内联和 !important，将 z-index 指定为 2147483647。

#### 5.5、响应式

1. 【强制】Media Query 不得单独编排，必须与相关的规则一起定义；
2. 【强制】Media Query 如果有多个逗号分隔的条件时，应将每个条件放在单独一行中。

#### 5.6、浏览器兼容性

1. 【强制】不能使用耗资源的htc文件来兼容IE8上的css3样式，容易造成内存泄露，并且导致页面加载非常卡；
2. 【强制】带私有前缀的属性由长到短排列，按冒号位置对齐；
3. 【建议】需要添加 hack 时应尽可能考虑是否可以采用其他方式解决，如果能通过合理的 HTML 结构或使用其他的 CSS 定义达到理想的样式，则不应该使用 hack 手段解决问题。通常 hack 会导致维护成本的增加；
4. 【建议】尽量使用选择器hack 处理兼容性，而非属性hack，尽量使用符合 CSS 语法的 selector hack，可以避免一些第三方库无法识别 hack 语法的问题；
5. 【强制】禁止使用 Expression。

### html代码风格

1. 【强制】页面必须使用精简形式，明确指定字符编码。指定字符编码的 meta 必须是 head 的第一个直接子元素；
2. 【建议】JavaScript 应当放在页面末尾，或采用异步加载；
3. 【建议】移动环境或只针对现代浏览器设计的 Web 应用，如果引用外部资源的 URL 协议部分与页面相同，建议省略协议前缀；
4. 【强制】保证 favicon 可访问；
5. 【建议】若页面欲对移动设备友好，需指定页面的 viewport；
6. 【强制】禁止 img 的 src 取值为空。延迟加载的图片也要增加默认的 src；
7. 【强制】使用 button 元素时必须指明 type 属性值，因为button 元素的默认 type 为 submit，如果被置于 form 元素中，点击后将导致表单提交。为显示区分其作用方便理解，必须给出 type 属性。

## 注释规约

### 单行注释

1. 【强制】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/块注释

1. 【建议】 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。

### 文档化注释

1. 【强制】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。

#### 3.1、类型定义

1. 【强制】类型定义都是以{开始, 以}结束，常用类型如：{string}, {number}, {boolean}, {Object}, {Function}, {RegExp}, {Array}, {Date}；
2. 【强制】对于基本类型 {string}, {number}, {boolean}，首字母必须小写。

#### 3.2、文件注释

1. 【强制】文件顶部必须包含文件注释，用 @file 标识文件说明。
2. 【强制】 文件注释中可以用 @author 标识开发者信息，请以此为标准模板：

/\*\*

\* @file require config and app entry

\* @Author: zhangxuelian

\* @Date: 2017-09-13 11:14:46

\* @Last Modified by: zhangxuelian

\* @Last Modified time: 2018-02-01 17:34:53

\*\*/

#### 3.3、命名空间注释

1. 【建议】命名空间使用 @namespace 标识。

#### 3.4、类注释

1. 【建议】使用 @class 标记类或构造函数；
2. 【建议】使用 @extends 标记类的继承信息；
3. 【强制】使用包装方式扩展类成员时， 必须通过 @lends 进行重新指向；
4. 【强制】类的属性或方法等成员信息使用 @public / @protected / @private 中的任意一个，指明可访问性。

#### 3.5、函数/方法注释

1. 【强制】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. 【强制】参数和返回值注释必须包含类型信息和说明；
3. 【建议】当函数是内部函数，外部不可访问时，可以使用 @inner 标识；
4. 【强制】对 Object 中各项的描述， 必须使用 @param 标识；
5. 【建议】重写父类方法时， 应当添加 @override 标识。如果重写的形参个数、类型、顺序和返回值类型均未发生变化，可省略 @param、@return，仅用 @override 标识，否则仍应作完整注释。

#### 3.6、事件注释

1. 【强制】必须使用 @event 标识事件，事件参数的标识与方法描述的参数标识相同；
2. 【强制】在会广播事件的函数前使用 @fires 标识广播的事件，在广播事件代码前使用 @event 标识事件；
3. 【建议】对于事件对象的注释，使用 @param 标识，生成文档时可读性更好。

#### 3.7、常量注释

1. 【强制】常量必须使用 @const 标记，并包含说明和类型信息。

#### 3.8、复杂类型注释

1. 【建议】 对于类型未定义的复杂结构的注释，可以使用 @typedef 标识来定义；

#### 3.9、AMD模块注释

1. 【强制】AMD 模块使用 @module 或 @exports 标识；
2. 【强制】对于已使用 @module 标识为 AMD模块 的引用，在 namepaths 中必须增加 module: 作前缀；
3. 【建议】对于类定义的模块，可以使用 @alias 标识构建函数；
4. 【建议】多模块定义时，可以使用 @exports 标识各个模块；
5. 【建议】对于 exports 为 Object 的模块，可以使用@namespace标识；
6. 【建议】对于 exports 为类名的模块，使用 @class 和 @exports 标识。

#### 4.0、细节注释

1. 【建议】细节注释遵循单行注释的格式。说明必须换行时，每行是一个单行注释的起始；
2. 【强制】有时我们会使用一些特殊标记进行说明。特殊标记必须使用单行注释的形式。