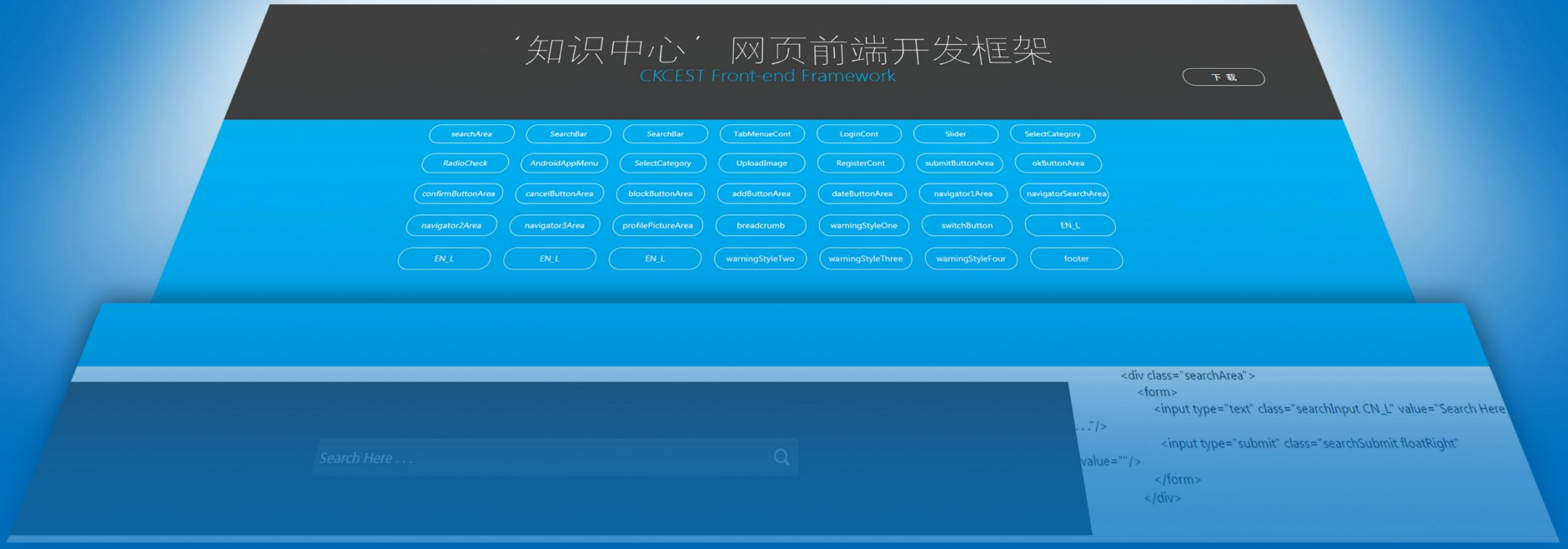
中国工程科技知识中心 网页规范&设计开发框架



中国工程科技知识中心 网页 前端规范

CKCEST Front-end Develop Standard

命名规则

HTML

CSS

JavaScript

命名规则

最佳原则

坚持这些原则。无论多少，找出不对的地方。如果你想要为这个编码指导做贡献，请访问新开一个 issue.
无论人数多少，代码都应该同出一门。

无论人数多少，代码都应该同出一门。

项目命名

项目名全部采用小写方式， 以中划线分隔。 比如： my-project-name

目录名

目录名参照上一条规则,有复数结构时，要采用复数命名法，比如说： scripts, styles, images, data-models

JavaScript 文件命名

所有 js 文件名，多个单词组成时，采用中划线连接方式，比如说： 账号模型文件 account-model.js

CSS，SCSS 文件命名

多个单词组成时，采用中划线连接方式，比如说：retina-sprites.scss

HTML 文件命名

多个单词组成时，采用中划线连接方式，比如说: error-report.html

HTML 01

语法

使用四个空格的 soft tabs — 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。
不要忽略可选的关闭标签（例如， 和 </body>）。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page title</title>
    </head>
<body>
    <img src="images/company-logo.png"
alt="Company">

    <h1 class="hello-world">Hello,
world!</h1>
</body>
</html>
```

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>
<html>
    <head>
    </head>
</html>
```

语言属性

针对 HTML5 :作者应在 html 的跟元素上加上这个文件的语言。这会给语音工具和翻译工具帮助，告诉它们应当怎么去发音和翻译。
通过 Sitepoint 得到一个语言代码列表.Sitepoint 只是给出了语言代码的大类，比如说中文就只给出了 ZH，但是没有区分香港，台湾，大陆等.

```
<html lang="en-us">
    <!-- ... -->
</html>
```

HTML 02

IE 兼容模式

Internet Explorer 支持使用一个文档属性标签 <meta> 来指出这个页面应当支持的 IE 的版本。除非另有规定，最好用最新的支持的模式.

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

字符编码

通过声明一个明确的字符编码，让浏览器轻松、快速的确定适合网页内容的渲染方式。

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

HTML 03

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。

HTML5 规范链接

使用 link
使用 style
使用 script

```
<!-- 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。
属性顺序:HTML 属性应该按照特定的顺序出现以保证易读性。

HTML 04

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。
了解更多内容，参考 WhatWG section on boolean attributes:
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。
如果你必须为属性添加并不需要的取值，参照 WhatWG 的指引:
如果属性存在，他的取值必须是空
字符串或者 [...] 属性的规范名称，不要在首尾包含空白字符。
简而言之，不要为 Boolean 属性添加取值。

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

<input type="checkbox" value="1" checked>

<select>
   <option value="1" selected>1</option>
</select>
```

HTML 05

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="...">
```

```
<!-- Not so great -->
<span class="avatar">
    <img src="...">
</span>

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

JavaScript 生成标签

在 JavaScript 文件中生成标签让内容变得更难查找，更难编辑，性能更差。应该尽量避免这种情况的出现。

CSS 01

语法

使用四个空格的 soft tabs — 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。比如说 box-shadow
不要在颜色值 rgb() rgba() hsl() hsla()和 rect() 中增加空格，并且不要带有取值前面不必要的 0 (比如，使用 .5 替代 0.5)。This helps differentiate multiple color values (comma, no space) from multiple property values (comma with space).
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;。
对这里提到的规则有问题吗？参考 Wikipedia 中的 CSS 语法部分。

```
/* Bad CSS */
.selector, .selector-
secondary, .selector[type=text] {
    padding: 15px;
    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"] {
    padding: 15px;
    margin-bottom: 15px;
    background-color: rgba(0,0,0,.5);
    box-shadow: 0 1px 2px #ccc, inset 0 1px
0 #fff;
}
```

CSS 02

声明顺序

相关的属性声明应该以下面的顺序分组处理：
Positioning
Box model 盒模型
Typographic 排版
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。关于完整的属性以及他们的顺序，请参考 Recess。

```
.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;
}
```

CSS 03

不要使用 @import

与 <link>标签相比, @import 更慢, 增加了额外的页面请求，可能还会导致不可预见的问题。避免使用它们，改用以下的方法:
多用几个 <link> 标签
将你的 css 编译到一个文件里
用 Rails, Jekyll,以及其他环境提供的特性来间接这些 css

```
<!-- 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 { ... }
}.element { ... }
```

CSS 04

前缀属性

当使用厂商前缀属性时，通过缩进使取值垂直对齐以便多行编辑。

```
/* Prefixed properties */
.selector {
    -webkit-box-shadow: 0 1px 2px
rgba(0,0,0,.15);
            box-shadow: 0 1px 2px
rgba(0,0,0,.15);
}
```

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。

```
/* Single declarations on one line */
.span1 { width: 60px; }
.span2 { width: 140px; }
.span3 { width: 220px; }

/* Multiple declarations, one per line */
.sprite {
    display: inline-block;
    width: 16px;
    height: 15px;
    background-image:
url(../img/sprite.png);
}
.icon          { background-position: 0 0; }
.icon-home      { background-position: 0 -
20px; }
.icon-account   { background-position: 0 -
40px; }
```

CSS 05

关于属性简写

坚持限制属性取值简写的使用，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;
}
```

LESS 和 SASS 中的嵌套

避免不必要的嵌套。可以进行嵌套，不意味着你应该这样做。只有在需要给父元素增加样式并且同时存在多个子元素时才需要考虑嵌套。

```
// Without nesting
.table > thead > tr > th { … }
.table > thead > tr > td { … }

// With nesting
.table > thead > tr {
    > th { … }
    > td { … }
}
```

CSS 06

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。

```
/* Bad example */
/* Modal header */
.modal-header {
    ...
}
/* Good example */
/* Wrapping element for .modal-title
and .modal-close */
.modal-header {
    ...
}
```

Class 命名

保持 Class 命名为全小写，可以使用短划线。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-* classes 来表示行为(相对于样式)，但是不要在 CSS 中包含这些 classes。

```
/* Bad example */
.t { ... }
.red { ... }
.header { ... }

/* Good example */
.tweet { ... }
.important { ... }
.tweet-header { ... }
```

选择器

使用 classes 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).

```
/* Bad example */
span { ... }
.page-container #stream .stream-
item .tweet .tweet-header .username { ... }
.avatar { ... }
/* Good example */
.avatar { ... }
.tweet-header .username { ... }
.tweet .avatar { ... }
```

CSS 07

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。

```
/*
 * Component section heading
 */

.element { ... }


/*
 * Component section heading
 *
 * Sometimes you need to include optional
context for the entire component. Do that
up here if it's important enough.
 */

.element { ... }

/* Contextual sub-component or modifer */
.element-heading { ... }
```

编辑器配置

根据以下的设置来配置你的编辑器，来避免常见的代码不一致和丑陋的 diffs。
使用四个空格的 soft-tabs。
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。

JavaScript 01

缩进,分号,单行长度

一律使用 4 个空格
连续缩进 同样适用 4 个空格，跟上一行对齐
Statement 之后一律以分号结束， 不可以省略
单行长度，理论上不要超过 80 列，不过如果编辑器开启 soft wrap 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，比如说右边第二段这种就是最佳格式。

```
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

JavaScript 02

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，一律小写第一个，大写后两个字母
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
}
```

字符常量

一般情况下统一使用 '' 单引号

JavaScript 03

null 的使用场景

初始化一个将来可能被声明为一个对象的变量。
与一个可能是对象或者非对象的初始化变量相比。
传入一个对象待定的函数。
作为一个对象待定的函数的返回值。

不适合 null 的使用场景

不要使用 null 来测试一个一个变量是否存在。
不要用 null 来测试一个没声明的变量。

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

Object Literals

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

// Good   semi colon 采用 Followed by space 的形式
var team = {
      title: "AlloyTeam",
      count: 25
};
```

JavaScript 04

Array Literals

```
// 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
可独占一行，前边不许有空行，缩进与下一行代码保持一致
可位于一个代码行的末尾，注意这里的格式

```
// 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，格式如右
前边留空一行

何时使用

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
想吐槽的产品逻辑，合作同事
业务逻辑强相关的代码

```
/*
 * 注释内容与星标前保留一个空格
 */
```

JavaScript 05

文档注释

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

用在哪里

所有的方法
所有的构造函数
所有的全局变量

```
/**
 * 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， 决不允许右边第二种情况
涉及 if for while do...while try...catch...finally 的地方都必须使用花括号

```
// Good
if (condition) {
    doSomething();
}

if (condition)
    doSomething();
    doSomethingElse();
```

JavaScript 06

switch

采用右边的格式， 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 blank line
花括号必须要， 即使内容只有一行， 决不允许右边第二种情 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情况

```
switch (condition) {
    case "first":
        // code
        break;
    case "third":
        // code
        break;
    default:
    // code
}
switch (condition) {
    // obvious fall through    // 这里为啥 JSHint 默认就会放过，因为 case "first" 内无内容
    case "first":
    case "second":
        // code
        break;
    case "third":
    // code
    /* falls through */ // 这里为啥要加这样的注释， 明确告知 JSHint 放过此处告警
    default:
    // code
}
switch(condition) {
    case "first":
        // code
        break;
    case "second":
        // code
        break;

    // no default
}
```

JavaScript 07

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

```
var values = [ 1, 2, 3, 4, 5, 6, 7 ],
    i, len;

for (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， 在行末跟注释

```
function doSomethingWithItems(items) {
    var value = 10,    // 注释
    result = value + 10,    // 注释
        i,    // 注释
        len;    // 注释
    for (i=0, len=items.length; i < len; i++) {
        doSomething(items[i]);
    }
}
```

JavaScript 08

函数声明

一定先声明再使用， 不要利用 JavaScript engine 的 hoist 特性，违反了这个规则 JSLint 和 JSHint 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 和 JSHint 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
var value = (function() {

    // function body
    return {
        message: "Hi"
    }
}());
// Good
(function() {
    "use strict";
    function doSomething() {
        // code
    }
    function doSomethingElse() {
        // code
    }
})();
```

欢迎指正

/* 201 Multi-Media Lab */

pengren@zju.edu.cn