### **JSHint配置选项**

**详细配置http://corncandy.github.io/2014/05/31/jshint-options/**

* **bitwise**

禁用位运算符(如^，|，&)

位运算符在JS中很少使用，性能也较差，出现&也很可能是想写&&。

* **camelcase**

使用驼峰命名(camelCase)或全大写下划线命名(UPPER\_CASE)

这是条最佳实践，关键不在于采用什么样的命名规则(比如纯小写配下划线)，而在于要有规则，在代码中看到不同的命名规则会让人头痛不已。

* **curly**

if和while等语句中使用{}来明确代码块

while (day)

shuffle();

sleep();

虽然缩进表示两条语句都在循环中，但事实却是只有一句循环。

* **eqeqeq**

使用===和!==替代==和!=

==和!=比较时会对前后元素进行自动转义，作为读者，需要动脑筋想这里可能有什么样的转义规则，加重负担；作为作者，其实很可能是不确定这段代码运行时是怎么样的，想要偷懒。

* **es3**

强制使用ECMAScript 3规范

* **forin**

在for in循环中使用Object.prototype.hasOwnProperty()来过滤原型链中的属性

for (key in obj) {

if (obj.hasOwnProperty(key)) {

// We are sure that obj[key] belongs to the object and was not inherieted.

}

}

for in遍历对象属性的时候，包括继承自原型链的属性，hasOwnProperty可以来判断一个属性是否是对象本身的属性而不是继承得来的。

* **freeze**

禁止复写原生对象(如Array, Date)的原型

/\* jshint freeze:true \*/Array.prototype.count = function (value) { return 4; };// -> Warning: Extending prototype of native object: 'Array'.

为原生对象添加属性确实看上去很方便，但也带来了潜在的问题，一是如果项目中有多处为同一个对象添加了同样的属性（或函数），则很可能产生冲突；二是如果某段逻辑依赖于对象属性遍历，则可能产生错误。

* **immed**

匿名函数调用必须

(function() {

// body

}());

而不是

(function() {

// body

})();

这是为了表明，表达式的值是函数的结果，而不是函数本身。

* **indent**

代码缩进宽度（空格数）

前面几个项目我比较喜欢4，新项目我又在尝试2。关键不在于是几，而在于大家都要设成一样的。

* **latedef**

变量定义前禁止使用

JS的变量是“函数级作用域”，而不是通常所见的“块级作用域”，简单说

function sum(numbers) {

for (var i = 0, n = numbers.length; i < n; i++) {

var sum = sum + numbers[i];

}

return sum;

}

相当于

function sum(numbers) {

var i, n, sum;

for (i = 0, n = numbers.length; i < n; i++) {

sum = sum + numbers[i];

}

return sum;

}

这个行为叫做“变量声明提升”，为了不产生混淆，这条规则建议函数都使用第二种写法。

* **newcap**

构造函数名首字母必须大写

这条最佳实践是为了方便区分构造函数和普通函数，这样在直接调用大写字母开头的函数时，使用者就会想想是不是自己写错了。

不通过new而直接调用构造函数，会使得构造函数中的this指向global对象，从而产生错误。

PS. 有些高手可以通过在构造函数中判断this的指向来判断是否重新new自身，从而让构造函数也能直接调用产生新对象。但这有些高深，加重开发人员和使用人员的负担，也不利于统一编码风格。

* **noarg**

禁止使用arguments.caller和arguments.callee

一方面这两个属性不是所有的浏览器都支持，另一方面这两个属性的使用会导致JS引擎很难优化代码，在未来的JS规范中会被去掉，所以不建议使用。

* **noempty**

禁止出现空的代码块

空的代码块并不是有害的，但是出现的话我们需要考虑下为什么。

* **nonbsp**

禁止"non-breaking whitespace"

这是Mac键盘在某种情况下可以键入的字符，据说会破坏非UTF8编码的页面。

* **nonew**

禁止使用构造器

new MyConstructor();

构造一个对象，却不给它赋值到某个变量，只是利用构造函数中的逻辑。这个行为完全可以用一个普通函数来完成，不应该借助构造器。

* **plusplus**

禁止使用++和--

不是很赞成把这个选项打成true，不过乱用自增/自减确实也会带来阅读上的障碍。

* **quotemark**

统一使用单引号或双引号

这个最佳实践要求代码风格统一，我比较喜欢统一成单引号。

这是为什么规定最佳实践的一个好例子，在写到字符串的时候我们就不用考虑使用单引号好还是用双引号好，就都用单引号，这在一定程度上也减轻了我们的思考负担。

* **undef**

禁止使用不在全局变量列表中的未定义的变量

function test() {

var myVar = 'Hello, World';

console.log(myvar); // Oops, typoed here. JSHint with undef will complain

}

如果本地作用域里的变量没有使用var来声明，则会被放到全局作用域下面，众所周知，全局变量时罪恶的源泉。

* **unused**

禁止定义变量却不使用

function test(a, b) {

var c, d = 2;

return a + d;

}test(1, 2);// Line 1: 'b' was defined but never used.// Line 2: 'c' was defined but never used.

这种变量通常是写作过程中遗留下来的垃圾，需要及时清理掉。

* **strict**

强制使用ES5的严格模式

Strict Mode是对JS用法的一些限制，过滤掉了容易出错的特性和不容易优化的特性。

通过在函数开头处加入'use strict';来触发严格模式，不要在文件头部加入，因为在JS链接的时候很可能就失效了。

* **trailing**

禁止行尾空格

* **maxparams**

函数可以接受的最大参数数量

函数参数数量应该控制在3个以内，超出则可能造成使用困难，比如需要记忆参数顺序，难以设定默认值等。另外，在JS中可以很方便的使用参数对象来封装多个参数。

* **maxdepth**

代码块中可以嵌入{}的最大深度

* **maxstatement**

函数中最大语句数

* **maxcomplexity**

函数的最大圈复杂度

* **maxlen**

一行中最大字符数

这个是为了减轻代码阅读的困难，简单说就是不要折行。

上面四个参数最终都是为了减小代码的复杂程度，简单轻巧的代码片段更容易阅读和维护。

## **松弛参数(Relaxing Options)**

本类参数设为true，JSHint会产生更少告警。

* **asi**

允许省略分号

JavaScript的语法允许自动补全分号，但是这一特性也会造成难以定位的错误，所以建议写代码时不要省略分号。

* **boss**

允许在if，for，while语句中使用赋值

在条件语句中使用赋值经常是笔误if (a = 10) {}，但是牛人(boss)可以把这个特性用的很好，我们作为普通人就算了。

* **debug**

允许debugger语句

debugger语句在产品代码中应该去掉。

* **eqnull**

允许==null

==null通常用来比较=== null || === undefined

* **esnext**

允许ECMAScript 6规约

目前ES6的特性不是所有的浏览器都支持。

* **evil**

允许使用eval

eval有“注入攻击”的危险，另一方面也不利于JS引擎优化代码，所以尽量不要使用。

* **expr**

允许应该出现赋值或函数调用的地方使用表达式

* **funcscope**

允许在控制体内定义变量而在外部使用

function test() {

if (true) {

var x = 0;

}

x += 1; // Default: 'x' used out of scope.

// No warning when funcscope:true

}

虽然“变量声明提升”使得上面的代码可以运行通过，但是读者还是会感到头晕。

* **globalstrict**

允许全局严格模式

在**strict**中解释了，'use strict';放在全局域可能造成JS文件链接错误。

* **iterator**

允许\_\_iterator\_\_

不是所有的浏览器都支持\_\_iterator\_\_。

* **lastsemic**

允许单行控制块省略分号

var name = (function() { return 'Anton' }());

高手用得到的特性，我们还是坚持加上分号吧。

* **laxbreak**

允许不安全的行中断(与**laxcomma**配合使用)

* **laxcomma**

允许逗号开头的编码样式

var obj = { name: 'Anton'

, handle: 'valueof'

, role: 'SW Engineer'

};

* **loopfunc**

允许循环中定义函数

在循环中定义函数经常会导致错误：

var nums = [];

for (var i = 0; i < 10; i++) {

nums[i] = function (j) {

return i + j;

};

}

nums[0](2); // Prints 12 instead of 2

错误的根源在于function(j)中的i是对循环中的i的引用，而不是赋值。所以在最终函数执行时，i的值是10。

修改的方法是使用闭包：

var nums = [];

for (var i = 0; i < 10; i++) {

(function (i) {

nums[i] = function (j) {

return i + j;

};

}(i));

}

* **maxerr**

JSHint中断扫描前允许的最大错误数

因为最终我们需要清零JSHint报错的，所以这个值用在对已有项目的扫描中。

* **multistr**

允许多行字符串

* **notypeof**

允许非法的typeof操作

* **proto**

允许 **proto**

不是所有的浏览器都支持\_\_proto\_\_.

* **smarttabs**

允许混合tab和space排版

SmartTabs方法使用tab进行缩进，使用空格进行代码对齐。比较高级的用法，有兴趣的话可以尝试下。

* **shadow**

允许变量shadow

function test() {

var x = 10;

if (true) {

var x = 20;

}

return x;

}

基于“函数作用域”，多次定义变量和单次定义是没有区别的，但是会造成阅读障碍。

* **sub**

允许person['name']

JSHint推荐使用person.name代替person['name']

* **supernew**

允许new function() {...}和new Object;

* **validthis**

允许严格模式下在非构造函数中使用this

* **noyield**

允许发生器中没有yield语句