# [Javascript技术规范](http://192.168.4.100:8090/pages/viewpage.action?pageId=1017951)

* 变量声明必须加var关键字，严格控制作用域；
* 建议使用驼峰式命名变量和函数，如：functionNamesLikeThis, variableNamesLikeThis, ClassNamesLikeThis；
* 使用单引号来定义字符串；
* 文件名必须全部用小写，文件名分隔符用**中划线**连接，版本连接符用**实心点**，合并文件的文件名连接符用**下划线**，如：mer-core.min.js和reset-1.0\_utils-1.0.css；

## 【以下部分参考airbnb规范】

## 数据类型

**原始类型(Primitives)**：当你给一个原始类型赋值时，返回的是这个值的本身。

* + string
  + number
  + boolean
  + null
  + undefined

var foo = 1,

bar = foo;

bar = 9;

console.log(foo, bar); // => 1, 9

**对象类型**:当你给一个对象类型赋值时，返回的是这个值的引用。

* + object
  + array
  + function

var foo = [1, 2],

bar = foo;

bar[0] = 9;

console.log(foo[0], bar[0]); // => 9, 9

## [对象](https://github.com/DeanXu/javascript-code-style" \l "对象)

新建一个对象的语法

//不推荐var item = new Object();

//推荐var item = {};

不要使用[保留字](http://es5.github.io/" \l "x7.6.1)作为键值，否则在IE8下面会出现问题([详情](https://github.com/airbnb/javascript/issues/61))。

//不推荐var superman = {

default: { clark: 'kent'},

private: true

};

//推荐var superman ={

defaults: { clark: 'kent'},

hidden: true

};

使用可读性强的同义词代替保留字

//不推荐var superman = {

class: 'alien'

};

//不推荐var superman = {

klass: 'alien'

};

//推荐var superman = {

type: 'alien'

};

## [数组](https://github.com/DeanXu/javascript-code-style" \l "数组)

新建一个数组的语法

//不推荐var items = new Array();

//推荐var items = [];

如果你不知道数组的长度可以使用push将元素加入。

var someStack = [];

//不推荐

someStack[someStack.length] = 'something';

//推荐someStack.push('something');

当你需要复制一个数组的时候使用slice。[jsPerf](http://jsperf.com/converting-arguments-to-an-array/7)

var len = items.length,

itemsCopy = [],

i;

//不推荐for (i = 0; i < len; i++){

itemsCopy[i] = items[i];

}

//推荐

itemsCopy = items.slice();

用slice转换伪数组对象到数组

function trigger() {

var args = Array.prototype.slice.call(arguments);

...

}

## [String类型](https://github.com/DeanXu/javascript-code-style" \l "string类型)

使用单引号''

//不推荐var name = "Bob Parr";

//推荐var name = 'Bob Parr';

//不推荐var fullName - "Bob " + this.lastName;

//推荐var fullName = 'Bob ' + this.lastName;

当字符串长度超过80个时，应该通过字符串连接多行显示。

注意：过度使用字符串连接将会影响性能。[jsPerf](http://jsperf.com/ya-string-concat)&[Discussion](https://github.com/airbnb/javascript/issues/40)

//不推荐var errorMessage = 'This is a super long error that was thrown because of Batman. When you stop to think about how Batman had anything to do with this, you would get nowhere fast.';

//不推荐var errorMessage = 'This is a super long error that \was thrown because of Batman. \When you stop to think about \how Batman had anything to do \with this, you would get nowhere \fast.';

//推荐var errorMessage = 'This is a super long error that ' +

'was thrown because of Batman.' +

'When you stop to think about ' +

'how Batman had anything to do ' +

'with this, you would get nowhere ' +

'fast.';

当程序建立一个字符串时， 使用join代替字符串连接。特别是在IE下：[jsPerf](http://jsperf.com/string-vs-array-concat/2)

var items,

messages,

length, i;

messages = [{

state: 'success',

message: 'This one worked.'

},{

state: 'success',

message: 'This one worked as well.'

},{

state: 'error',

message: 'This one did not work.'

}];

length = messages.length;

// 不推荐function inbox(messages) {

items = '<ul>';

for (i = 0; i < length; i++) {

items += '<li>' + messages[i].message + '</li>';

}

return items + '</ul>';

}

// 推荐function inbox(messages) {

items = [];

for (i = 0; i < length; i++) {

items[i] = messages[i].message;

}

return '<ul><li>' + items.join('</li><li>') + '</li></ul>';

}

## [函数](https://github.com/DeanXu/javascript-code-style" \l "函数)

函数表达式：

// 匿名函数表达式var anonymous = function(){

return true;

}

// 命名函数表达式var named = function named() {

return true;

};

// 立即执行的函数表达式（IIFE）

(function(){

console.log('Welcome to the Internet. Please follow me.');

})();

不要将函数声明放在如if/while循环或其他任何语句中。但可以用函数表达式来替代函数声明这么做。一些浏览器可能的确可以在语句中使用函数声明。但是在解析方面的处理各不相同，各种浏览器下兼容性很不好。

**注意:** ECMA-262定义了一系列的语句，但是函数声明并没有被归类为真正的语句。[关于这点可查看ECMA-262的文档](http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf" \l "page=97)

//不推荐if (currentUser){

function test() {

console.log('Nope.');

}

}

//推荐if (currentUser){

var test = function test() {

console.log('Yup.');

}

}

arguments 不能作为一个参数的名字， 因为这会覆盖每一个函数内的arguments对象。

//不推荐function nope(name, options, arguments) {

// ...stuff...

}

//推荐function yup(name, options, args) {

// ...stuff...

}

## [属性](https://github.com/DeanXu/javascript-code-style" \l "属性)

访问一个属性时，使用点的形式取值。

var luke = {

jedi: true,

age: 28

};

// 不推荐var isJedi = luke['jedi'];

// 推荐var isJedi = luke.jedi;

需要一个变量访问一个属性时，使用“[]”来取值。

var luke = {

jedi: true,

age: 28

};

function getProp(prop) {

return luke[prop];

}

var isJedi = getProp('jedi');

## [变量](https://github.com/DeanXu/javascript-code-style" \l "变量)

总是使用 var 来定义变量。如果不这么做将定义一个全局变量出来。我们希望避免全局命名空间的污染。

// 不推荐

superPower = new SuperPower();

// 推荐var superPower = new SuperPower();

使用一个var 声明多个变量，并且每声明一个变量就换一行。

// 不推荐var items = getItems();var goSportsTeam = true;var dragonball = 'z';

// 推荐var items = getItems(),

goSportsTeam = true,

dragonball = 'z';

声明多个变量时，把不赋值的变量放在后面。这样做是有好处的，如果日后你想给未赋值变量赋值的时候，可能要引用到上面已经赋值的变量。

// 不推荐var i, len, dragonball,

items = getItems(),

goSportsTeam = true;

// 不推荐var i, items = getItems(),

dragonball,

goSportsTeam = true,

len;

// 推荐var items = getItems(),

goSportsTeam = true,

dragonball,

length,

i;

在一个作用域的顶部给一个变量赋值。这样有助于避开，变量声明和声明提前的分配问题。

// 不推荐function() {

test();

console.log('doing stuff..');

//..other stuff..

var name = getName();

if (name === 'test') {

return false;

}

return name;

}

// 推荐function() {

var name = getName();

test();

console.log('doing stuff..');

//..other stuff..

if (name === 'test') {

return false;

}

return name;

}

// 不推荐function() {

var name = getName();

if (!arguments.length) {

return false;

}

return true;

}

// 推荐function() {

if (!arguments.length) {

return false;

}

var name = getName();

return true;

}

## [声明提前](https://github.com/DeanXu/javascript-code-style" \l "声明提前)

不管你在何处给一个变量声明或赋值，javascript解析器都会事先在作用域的顶端做声明提前（Hoisting）。

// 我们知道下面将不能正常运行（假设没有全局变量）function example() {

console.log(notDefined); // => 抛出一个引用错误

}

// 在引用这个变量之后，给这个变量赋值将不会抛异常，这是因为javascript解析器有声明提前。// 注意：赋的“true”值，不会被提前。function example() {

console.log(declaredButNotAssigned); // => undefined

var declaredButNotAssigned = true;

}

// javascript解析器,会在作用域的顶部提前声明变量。// 用代码描述出来，其实就等同于下面这种情况。function example() {

var declaredButNotAssigned;

console.log(declaredButNotAssigned); // => undefined

declaredButNotAssigned = true;

}

匿名函数表达式将该变量名做了提前声明，没有给该变量赋值函数。

function example() {

console.log(anonymous); // => undefined

anonymous(); // => 抛出异常，anonymous 不是一个函数

var anonymous = function() {

console.log('anonymous function expression');

};

}

和匿名一样，有名函数表达式将该变量名做了提前声明，没有给该变量赋值函数名和函数体。

function example() {

console.log(named); // => undefined

named(); // => 抛出异常， named 不是一个函数

superPower(); // => 抛出异常， superPower 没定义

var named = function superPower() {

console.log('Flying');

};

}

// 把函数名改成和变量名一样，也得出同样的结果。function example() {

console.log(named); // => undefined

named(); // => 抛出异常， named 不是一个函数

var named = function named() {

console.log('named');

};

}

函数声明会将函数名和函数体声明提前。

function example() {

superPower(); // => Flying

function superPower() {

console.log('Flying');

}

}

更多信息请参照 [Ben Cherry](http://www.adequatelygood.com/) 的 [JavaScript Scoping & Hoisting](http://www.adequatelygood.com/2010/2/JavaScript-Scoping-and-Hoisting)

## [条件表达式与等式](https://github.com/DeanXu/javascript-code-style" \l "条件表达式与等式)

使用 === 和 !== 代替 == 和 !=。

条件表达式 会通过 ToBoolean 来进行强制转化，而且遵循以下的规则：

* + **对象**被转化为**true**
  + **Undefined**被转化为**false**
  + **Null**被转化为**false**
  + **布尔值**被转化为**相应的布尔值**
  + **数字**当值为**+0**,**-0**或**NaN**时转化为**false**，其他的转化为**true**
  + **Strings类型**如果为空时转化为**false**,否则转化为**true**

if ([0]) {

// true

// 因为数组是对象，对象会被转化为 true

}

使用快捷方式

// 不推荐if (name !== '') {

// ...stuff...

}

// 推荐if (name) {

// ...stuff...

}

// 不推荐if (collection.length > 0) {

// ...stuff...

}

// 推荐if (collection.length) {

// ...stuff...

}

* 更多的信息 请看 Angus Croll 的 [Truth Equality and JavaScript](http://javascriptweblog.wordpress.com/2011/02/07/truth-equality-and-javascript/" \l "more-2108)

## [块](https://github.com/DeanXu/javascript-code-style" \l "块)

给多行的块，使用大括号

// 不推荐if (test)

return false;

// 推荐if (test) return false;

// 推荐if (test) {

return false;

}

// 不推荐function() { return false; }

// 推荐function() {

return false;

}

## [注释](https://github.com/DeanXu/javascript-code-style" \l "注释)

使用 /\*\*...\*/ 进行多行注释。注释要包括描述、指定类型、参数值和返回值。

// 不推荐

// make() returns a new element// based on the passed in tag name//// @param <String> tag// @return <Element> elementfunction make(tag) {

// ...stuff...

return element;

}

// 推荐/\*\* \* make() returns a new element \* based on the passed in tag name \* \* @param <String> tag \* @return <Element> element \*/function make(tag) {

// ...stuff...

return element;

}

使用 // 进行单行注释。注释单独占一行，并写在需要注释对象的上面。在注释的上面留一个空行。

// 不推荐var active = true; // is current tab

// 推荐// is current tabvar active = true;

// 不推荐function getType() {

console.log('fetching type...');

// set the default type to 'no type'

var type = this.\_type || 'no type';

return type;

}

// 推荐function getType() {

console.log('fetching type...');

// set the default type to 'no type'

var type = this.\_type || 'no type';

return type;

}

给你的代码加前缀，比如FIXME或TODO，这样有助于其他开发者可以迅速理解你指出的需要被处理的问题。 如果想更清晰一点你还可以在后面加上描述，比如: FIXME -- need to figure this out或TODO -- need to implement.

使用 // FIXME: 去注释问题

function Calculator() {

// FIXME: shouldn't use a global here

total = 0;

return this;

}

使用 // TODO: 来注释解决方法

function Calculator() {

// TODO: total should be configurable by an options param

this.total = 0;

return this;

}

## [空格](https://github.com/DeanXu/javascript-code-style" \l "空格)

将tab键设成2个空格

// 不推荐function() {

∙∙∙∙var name;

}

// 不推荐function() {

∙var name;

}

// 推荐function() {

∙∙var name;

}

逗号/冒号/小括号后面留一个空格

// 不推荐function test(){

console.log('test');

}

// 推荐function test() {

console.log('test');

}

// 不推荐dog.set('attr',{

age: '1 year',

breed: 'Bernese Mountain Dog'

});

// 推荐dog.set('attr', {

age: '1 year',

breed: 'Bernese Mountain Dog'

});

在文件的最后留一个空行

// 不推荐

(function(global) {

// ...stuff...

})(this);

// 推荐

(function(global) {

// ...stuff...

})(this);

## [逗号](https://github.com/DeanXu/javascript-code-style" \l "逗号)

逗号不要前置

// 不推荐var once

, upon

, aTime;

// 推荐var once,

upon,

aTime;

// 不推荐var hero = {

firstName: 'Bob'

, lastName: 'Parr'

, heroName: 'Mr. Incredible'

, superPower: 'strength'

};

// 推荐var hero = {

firstName: 'Bob',

lastName: 'Parr',

heroName: 'Mr. Incredible',

superPower: 'strength'

};

最后一个元素不可以加逗号。这在IE6和IE7还有IE9的怪异模式下出错。